Arvind Krishnamurthy
Bernhard Plattner (Eds.)

LNCS 6032

Passive and Active Measurement

11th International Conference, PAM 2010
Zurich, Switzerland, April 2010
Proceedings

Springer

Arvind Krishnamurthy
Bernhard Plattner (Eds.)

Passive and Active Measurement

11th International Conference, PAM 2010
Zurich, Switzerland, April 7-9, 2010
Proceedings

Springer

Volume Editors

Arvind Krishnamurthy
Department of Computer Science and Engineering
University of Washington
185 Stevens Way
Seattle, WA, 98195, USA
E-mail: arvind@cs.washington.edu

Bernhard Plattner
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology (ETH)
Gloriastrasse 35 8092 Zurich, Switzerland
E-mail: plattner@tik.ee.ethz.ch

Library of Congress Control Number: 2010923740

CR Subject Classification (1998): C.2, H.4, K.6.5, D.2, D.4.6, E.3

LNCS Sublibrary: SL 5 – Computer Communication Networks
and Telecommunications

ISSN	0302-9743
ISBN-10	3-642-12333-3 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-12333-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

The 2010 edition of the Passive and Active Measurement Conference was the 11th of a series of successful events. Since 2000, the Passive and Active Measurement (PAM) conference has provided a forum for presenting and discussing innovative and early work in the area of Internet measurements. PAM has a tradition of being a workshop-like conference with lively discussion and active participation from all attendees. This event focuses on research and practical applications of network measurement and analysis techniques. This year's conference was held at ETH Zurich, Switzerland.

PAM 2010 attracted 79 submissions. Each paper was carefully reviewed by at least three members of the Technical Program Committee. The reviewing process led to the acceptance of 23 papers. The papers were arranged in nine sessions covering the following areas: routing, transport protocols, mobile devices, topology, measurement infrastructure, characterizing network usage, analysis techniques, traffic analysis, and the Web.

We are very grateful to Endace Ltd. (New Zealand), Cisco Systems Inc. (USA), armasuisse (Switzerland) and the COST Action TMA whose sponsoring allowed us to keep registration costs low and to offer several travel grants to PhD students. We are also grateful to ETH Zurich for sponsoring PAM as a host.

April 2010

Bernhard Plattner
Arvind Krishnamurthy

Organization

Organizing Committee

General Chair:	Bernhard Plattner (ETH Zurich, Switzerland)
Program Chair:	Arvind Krishnamurthy (University of Washington, USA)
Local Arrangements Chair:	Xenofontas Dimitropoulos (ETH Zurich, Switzerland)
Publicity Chair:	Marco Mellia (Politecnico di Torino, Italy)
Poster Chair:	Alan Mislove (Northeastern University, USA)

Program Committee

Matthew Caesar	UIUC, USA
Mark Crovella	Boston University, USA
Constantine Dovrolis	Georgia Tech, USA
Jaeyeong Jung	Intel Research Seattle, USA
Sachin Katti	Stanford University, USA
Thomas Karagiannis	Microsoft Research, Cambridge, UK
Simon Leinen	Switch, Switzerland
Craig Labowitz	Arbor Networks, USA
Sridhar Machiraju	Google, USA
Bruce Maggs	Duke University & Akamai, USA
Morley Mao	University of Michigan, Ann Arbor, USA
Alan Mislove	Northeastern University, USA
Sue Moon	KAIST, Korea
Neil Spring	University of Maryland, College Park, USA
Joel Sommers	Colgate University, USA
Renata Teixeira	CNRS and UPMC Paris Universitas, France
Arun Venkataramani	University of Massachusetts, Amherst, USA
Jia Wang	AT&T Research, USA
Yinglian Xie	Microsoft Research, Silicon Valley, USA
Ming Zhang	Microsoft Research, Redmond, USA
Yin Zhang	University of Texas, Austin, USA

Steering Committee

Nevil Brownlee	University of Auckland, New Zealand
Mark Claypool	Worcester Polytechnic Institute, USA
Ian Graham	Endace, New Zealand
Arvind Krishnamurthy	University of Washington, USA
Sue Moon	KAIST, Korea

Sponsoring Institutions

Endace Ltd.
CISCO Systems Inc.
armasuisse
COST Action TMA

Table of Contents

Characterizing the Global Impact of P2P Overlays on the AS-Level Underlay

Amir Hassan Rasti[1], Reza Rejaie[1], and Walter Willinger[2]

[1] University of Oregon
{amir,reza}@cs.uoregon.edu
[2] AT&T Labs Research
walter@research.att.com

Abstract. This paper examines the problem of characterizing and assessing the global impact of the load imposed by a Peer-to-Peer (P2P) overlay on the AS-level underlay. In particular, we capture Gnutella snapshots for four consecutive years, obtain the corresponding AS-level topology snapshots of the Internet and infer the AS-paths associated with each overlay connection. Assuming a simple model of overlay traffic, we analyze the observed load imposed by these Gnutella snapshots on the AS-level underlay using metrics that characterize the load seen on individual AS-paths and by the transit ASes, illustrate the churn among the top transit ASes during this 4-year period, and describe the propagation of traffic within the AS-level hierarchy.

Keywords: Overlay networks, AS-level topology, BGP simulation.

1 Introduction

The large volume of traffic associated with Peer-to-Peer (P2P) applications has led to a growing concern among ISPs which need to carry the P2P traffic relayed by their costumers. This concern has led researchers and practitioners to focus on the idea of reducing the volume of external P2P traffic for edge ISPs by localizing the connectivity of the P2P overlay (for recent work, see for example [1,2]). However, such an approach only deals with the local effect of an overlay on individual edge ASes. Even though the volume of P2P traffic on the Internet is large and growing, assessing the *global* impact of a P2P overlay on the individual ASes in the network, which we call the *AS-level underlay*, remains a challenging problem and is not well understood. This is in part due to the fact that investigating this problem requires a solid understanding of an array of issues in two different domains: *(i)* design and characterization of overlay-based applications, and *(ii)* characterization of AS-level underlay topology and BGP routing in this underlay. Another significant challenge is dealing with inaccurate, missing, or ambiguous information about the AS-level underlay topology, AS relationships and tier properties, and BGP routing policies.

This paper investigates the problem of assessing the load imposed by a given overlay on the AS-level underlay. We show that assessing this impact requires tackling a number of challenging problems, including *(i)* capturing accurate snapshots of the desired overlay, *(ii)* estimating the load associated with individual overlay connections, and *(iii)* determining the AS-path in the underlay that corresponds to individual overlay connections. Toward this end, this paper makes two main contributions. First, we present a methodology for assessing the impact of an overlay on the AS-level underlay. Our methodology incorporates a collection of the best known practices for capturing accurate snapshots of a P2P overlay and, more importantly, for determining the AS-path corresponding to each overlay connection. We rely on snapshots of the AS-level Internet topology provided by CAIDA where each link between two ASes is annotated with the relationship between them. Using a BGP simulator called *C-BGP* [4], we perform a detailed simulation of BGP routing over these annotated snapshots of the AS-level underlay to infer the corresponding AS-path for each overlay connection and determine the aggregate load crossing individual ASes. To assess the propagation of overlay traffic through the AS-level hierarchy, we also infer the tier information for individual ASes using the *TierClassify* tool [3].

Second, we illustrate our methodology by characterizing the impact of four snapshots of the *Gnutella* overlay that were captured over four successive years on the AS-level underlay snapshots of the Internet taken on the same dates the Gnutella overlay snapshots were obtained. We characterize the load imposed by these overlays on the corresponding underlay in a number of different ways: *(i)* observed load on individual AS-paths and its diversity, *(ii)* observed load on individual transit ASes, *(iii)* AS-path length, and *(iv)* the propagation of overlay traffic through the AS-level hierarchy. Our analysis provides valuable insight into how changes in overlay connectivity and underlay topology affect the mapping of load on the AS-level underlay.

The rest of this paper is organized as follows. In Section 2, we further elaborate on the problem of mapping an overlay on the AS-level underlay, describe the challenges involved, and present our methodology. Section 3 describes our datasets and presents our characterization of the load imposed by the Gnutella overlay on the corresponding AS-level underlay, spanning a 4-year period. We conclude the paper and sketch our future plans in Section 4.

2 The Problem and Our Methodology

Our goal is to map the traffic associated with a P2P overlay to the AS-level underlay. The input to this process is a representation of a P2P overlay structure consisting of the IP addresses (and port numbers) of the participating peers together with their neighbor lists. The output is the aggregate load on all affected ASes and between each pair of affected ASes that have a peering link with one another (in each direction). Our methodology to tackle this problem consists of the following intuitive steps:

1. Capturing the topology of a P2P overlay,
2. Estimating the load on individual connections in the overlay,
3. Inferring the AS-paths associated with individual overlay connections,

4. Determining the aggregate load on each AS and between connected ASes (in each direction separately).

In this section, we discuss the challenges posed by each step, clarify our assumptions, and describe our approach for each step.

2.1 Capturing the Overlay Topology

Capturing a snapshot of the overlay topology for a P2P application is feasible if the list of neighbors for individual peers can be obtained. For example, in Gnutella it is possible to query individual peers and retrieve their neighbor lists. Therefore, a Gnutella-specific crawler can be developed to progressively collect this information until a complete snapshot of the overlay is captured.

In our earlier work, we have developed a fast P2P crawler that can capture accurate snapshots of the Gnutella network in a few minutes [5]. Using this crawler, we have captured tens of thousands of snapshots of the Gnutella overlay topology over the past several years. In this study, we use a few of these snapshots for the top-level overlay of Gnutella (an overlay consisting of Gnutella *Ultrapeers*). While other P2P applications such as BitTorrent are responsible for a significantly larger volume of traffic over the Internet than Gnutella and would therefore provide a more relevant P2P system for this study, we are not aware of any reliable technique to capture accurate snapshots of the corresponding overlays. Since accuracy of the overlay topology is important in this study, we focus on Gnutella. However, our methodology is not restricted to this application and can be used with other P2P systems.

2.2 Estimating the Load of Individual Overlay Connections

The load of individual overlay connections depends on the subtle interactions between several factors including: *(i)* the number of peers that generate traffic (*i.e.*, sources), the rate and pattern of traffic generation by these peers, and their relative location in the overlay, *(ii)* the topology of the overlay, and *(iii)* the relaying (*i.e.*, routing) strategy at individual peers. Capturing these factors in a single model is a non-trivial task and could be application-specific. For example, the load of individual connections for live P2P video streaming is more or less constant, whereas the load of individual BitTorrent connections may vary significantly over time.

In the absence of any reliable model for per-connection traffic, without loss of generality, we assume in our analysis that all connections of the overlay experience the same average load in both directions. This simplifying assumption allows us to focus on the mapping of the overlay topology on the underlying AS-level topology. If a more reliable model for the load of individual connections is available, it can be easily plugged into our methodology by assigning proper weights (one in each direction) to each connection of the overlay. In this paper, we simply assume that the weight for all connections in both directions is one.

2.3 Inferring AS-Paths for Individual Overlay Connections

For each connection in the overlay, determining the corresponding AS-path in the underlay is clearly the most important and most challenging part of our methodology. We use a popular BGP simulator to determine the AS-path between any given pair of ASes, but note that carefully-designed measurement-based approaches may provide viable alternatives. Our simulation-based method consists of the following steps:

Mapping Peers to ASes: We use archived BGP snapshots from RouteViews [6] to map the IP addresses of individual peers to their corresponding ASes that we call edge ASes. Therefore, determining the AS-path for the overlay connection between two peers translates into determining the path between their corresponding edge ASes.

Capturing AS-level Topology and Inter-AS Relationships: In this study, we rely on the AS-level topologies provided by CAIDA [7]. These topologies have been widely used in the past, even though more recent work has shown that the provided topologies are missing a significant portion of peering links between lower-tiered ASes [8,9]. Note that our approach is not tied to using the CAIDA-provided AS-level topologies, and any more complete AS-level topology can be incorporated once it becomes available. To properly simulate BGP routing, we need to determine the AS relationship between connected ASes in the AS-level topology. Toward this end, we use the fact that CAIDA's snapshots of the AS-level topology [7] are annotated with the inferred relationships between each pair of connected ASes. In these snapshots, AS relationships are inferred using the algorithm initially proposed by Gao [10] and extended by Dimitropoulos *et al.* [11]. This algorithm, mainly based on the concept of "valley-free routing" in BGP (along with some other intuitive assumptions), categorizes the AS relationships into three categories: *(i)* Customer-Provider, *(ii)* Peer-Peer, or *(iii)* Sibling-Sibling.

Simulating BGP: We determine the AS-path between any pair of edge ASes that host connected peers in the overlay (*i.e.*, infer the corresponding AS-path) by simulating BGP over the annotated AS-level topology using the *C-BGP* simulator [4]. C-BGP abstracts the AS-level topology as a collection of interconnected routers, where each router represents an AS. It simulates the desired BGP routing policies for each relation between connected ASes. We use a set of intuitive BGP policies for each type of AS relationships that are specified by C-BGP. In particular, these policies *(i)* ensure that the routes through one's customers have the highest preference and those passing through its providers have the lowest preference, and *(ii)* prevent ASes with multiple providers from acting as transit node among their providers. We noticed that some characteristics of CAIDA's annotated AS-level topology, in particular the presence of circular provider-costumer relationships among a group of ASes, prevent our C-BGP simulations to converge with the above policies. To resolve these problems, we systematically change a small number of relationships (*e.g.*, to break a cycle in customer-provider relationships). Further details of this process are described in our related technical report [12]. We select snapshots of both the AS-level topology and the overlay topology of the same dates so as to minimize any potential error due to asynchrony in the snapshots.

Clearly, representing each AS by a single router results in inferring only one AS-path between each pair of ASes. This implies that multiple AS-paths that may exist in practice between two ASes [13] are not accounted for in our simulations. While this assumption simplifies the problem in a way that is not easily quantifiable, we are not aware of any existing technique that can reliably capture and account for this subtle behavior of BGP routing.

Assessing AS Tiers: To characterize the propagation of P2P traffic through the AS-level hierarchy, we first need to assess the location of each AS in this hierarchy. We use the "TierClassify" tool [3] to identify the *tier* of each individual AS. The algorithm used in this tool relies mainly on the assumption that all tier-1 ASes should be interconnected with one another. Therefore it tries to find a clique among the ASes with highest degrees. Once the tier-1 clique is identified, the algorithm simply follows provider-customer relationships and classifies other ASes such that each tier n AS can reach the tier-1 clique in $n - 1$ hops.

2.4 Determining Aggregate Load on and between Individual ASes

Given the corresponding AS-path for each overlay connection, we can easily determine the aggregate load (in terms of the number of connections) that passes through each AS, as well as the transit load (in each direction) between each pair of connected ASes in the topology.

3 Effect of Overlays on the Underlay

In this section, we characterize the effect of a P2P overlay on the AS-level underlay using four snapshots of the Gnutella top-level overlay. We broadly divide ASes into two groups: *Edge ASes* that host peers in an overlay, and *Transit* (or *Core*) *ASes* that provide connectivity between edge ASes. We first describe our datasets (*i.e.*, the snapshots of overlay and the corresponding AS-level underlay topologies), and then we characterize the imposed load on the underlay using the following measures: *(i)* diversity and load on individual AS-paths, *(ii)* load on individual transit ASes, *(iii)* identity and evolution of the top transit ASes, *(iv)* AS-path length, and *(v)* propagation of traffic through the AS-level hierarchy.

Datasets: We use four snapshots of the top-level Gnutella overlay that were collected in four consecutive years starting in 2004. Examining overlay snapshots over time enables us to assess some trends that are associated with the evolution of the AS-level topology.

We use the labels G-xx to refer to the snapshot taken in year 20xx. The left columns of Table 1 (labeled "Gnutella snapshots") summarize the capture date, number of peers and edges for these overlay snapshots. The table shows that the population of Gnutella peers in the top-level overlay and their pairwise connections have both increased by $\approx 600\%$ during this four-year period.

We also use daily snapshots of the BGP routing table retrieved from the Route-Views archive collected at the same dates as our overlay snapshots. The middle

Table 1. Data profile: Gnutella snapshots, BGP snapshots and mapping overlay connections to the underlay. Imp. AS-paths are those with +100 overlay connections.

		Gnutella Snapshots		BGP Snapshots		AS-Paths	
Snapshot	Date	#Peers	#Conn.	#Prefixes	#ASes	#Unique	%Important
G-04	04-11-20	177k	1.46M	165k	18.7k	192k	2.0
G-05	05-08-30	681k	5.83M	185k	20.6k	384k	2.9
G-06	06-08-25	1.0M	8.64M	210k	23.2k	605k	2.8
G-07	07-03-15	1.2M	9.80M	229k	24.9k	684k	2.7

columns in Table 1 (labeled "BGP snapshots") give the number of IP prefixes and the total number of ASes in each BGP snapshot. These numbers show that the AS-level topology has also grown significantly during this four-year period.

Diversity and Load on Individual AS-Paths: One way to characterize the impact of an overlay on the underlay is to determine the number of unique AS-paths that all overlay connections are mapped on as well as distribution of load among those AS-paths. The right columns of Table 1 (labeled "AS-paths") show the number of unique AS-paths for all connections of each overlay along with the percentage of those paths that carry more than 100 overlay connections. The number of unique AS-paths is growing over time but at a lower pace compared to the number of overlay connections. This suggests that there is more similarity in AS-paths among overlay connections as the overlay grows in size over time.

To examine the mapping of overlay connections to AS-paths more closely, Figure 1(a) depicts the CCDF of the number of overlay connections that map to individual AS-paths in log-log scale for all four overlay snapshots. The skewed shape of these distributions indicates that a small number of AS-paths carry a large fraction of load. For example, whereas around 10% of paths carry more than 10 connections, only 1% of the paths carry more than 200 connections. Interestingly, the distributions of overlay connections that map to AS-paths are very similar across different snapshots despite significant changes in the identity of peers and in the topologies of overlay and underlay.

Observed Load on Individual Transit ASes: Since we assumed that all overlay connections have the same load, we simply quantify the load on each transit AS by the number of overlay connections crossing that AS. Figure 1(b) depicts the number of overlay connections that cross each transit AS in log-log scale, where ASes are ranked (from high to low) based on their overall observed load. The figure shows that the load on transit ASes is very skewed. A small number of them carry a large volume of traffic while the load on most transit ASes is rather small. Again, we observe that the overall shape of the resulting curves is very similar for all four snapshots, except for the outward shift in the more recent snapshots caused by the increasing size of the overlay over time. This similarity in the skewness of the observed load on transit ASes despite significant changes in the overlay and underlay topologies over time could be due to the dominance of one the following factors: *(i)* the stability over time of the top-10

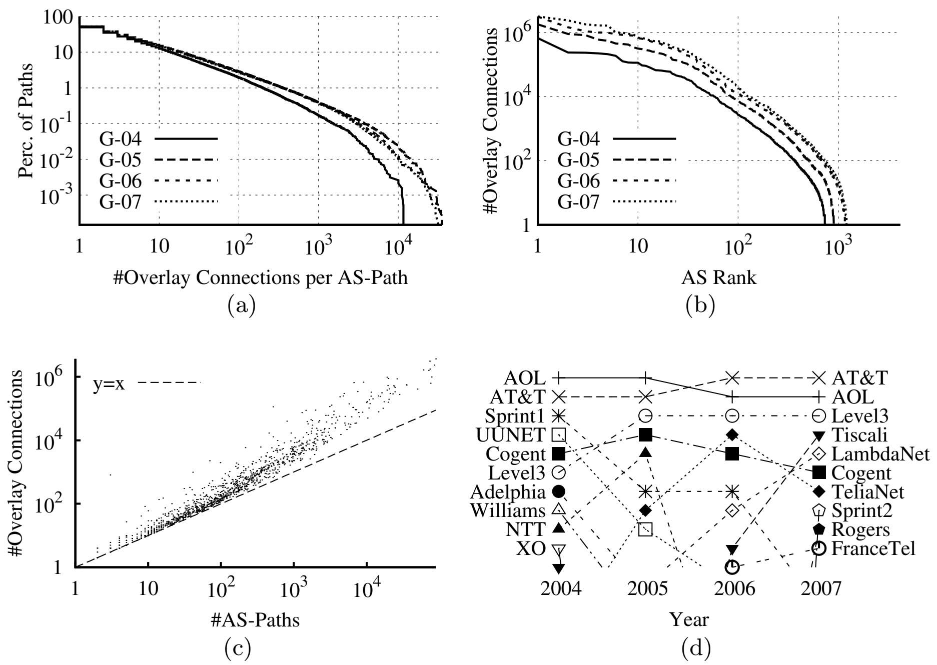


Fig. 1. (a) Distribution of load across AS-paths, (b) Overlay connections passing through transit ASes, (c) Scatterplot of number of relevant AS-paths vs. load, (d) Identity and evolution of top-10 transit ASes carrying the largest number of overlay connections

ASes that host most peers, and *(ii)* the constraint imposed by valley-free routing over the hierarchical structure of the AS-level underlay.

To further investigate the underlying causes for the observed skewed nature of observed load on transit ASes, we examine the distribution of the number of unique AS-paths (associated with overlay connections) that pass through each transit AS. The shape of this distribution is very similar to Figure 1(b) (not shown), suggesting that the number of crossing connections for individual ASes is primarily determined by the underlay shape and routing rather than connectivity and footprint of the overlay. Figure 1(c) validates this observation by showing a scatterplot of the number of crossing AS-paths (x-axis) and number of overlay connections (y-axis) through each transit AS. This figure essentially relates the previous two distributions and confirms that the observed load on individual transit ASes depends primarily on the number of unique AS-paths crossing those ASes. Note that once the number of cross AS-paths exceeds a certain threshold (a few hundreds), the observed load increases at a much faster pace.

Identity and Evolution of Transit ASes: To investigate the observed load by transit ASes from a different angle, we examine and present the identity of the top-10 transit ASes that carry the highest number of crossing overlay

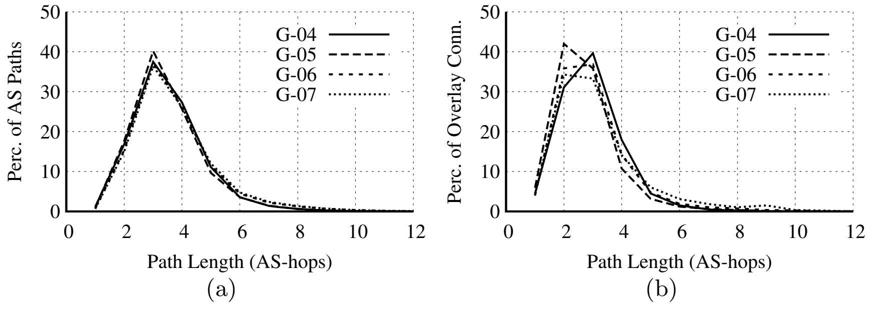


Fig. 2. (a) Distribution of AS-path length between connected edge ASes, (b) Distribution of AS-path length for all overlay connections

connections (and their evolution over time) in Figure 1(d). For each of the four overlay snapshots, the transit ASes are rank-ordered (highest load first), and the figure depicts their standings in these rank-ordered lists over time. We observe that only four transit ASes (*i.e.*, AT&T, AOL, Level3, and Cogent) remain in the top-10 list across all four snapshots and that the changes in the other transit ASes is more chaotic. This is due to the fact that ranking of transit ASes is affected by a combination of factors including changes in the topology of AS-level underlay, in routing policies, and in the location of peers. Disentangling these different factors and trying to identify the root causes for the observed churn among the top-10 transit ASes over time remains a challenging open problem.

AS-Path Length: One way to quantify the impact of an overlay on the AS-level underlay is to characterize the length of AS-paths for individual overlay connections. Figure 2(a) shows the empirical density of the length of all AS-paths between edge ASes for each of the four snapshots. We observe that around 40% of the paths are three AS-hops long, while 80% of the paths in each overlay are at most 4 AS-hops long.

Figure 2(b) depicts the empirical density of AS-path length across all *overlay connections* for each of the four snapshots. In essence, this plot can be viewed as a *weighted* version of Figure 2(a) described above where the length of each path is weighted by the number of overlay connections crossing it. The figure shows a very similar pattern across all overlay snapshots despite the changes in the number of peers and their connections. The two figures are very similar, however the average path length across the overlay connections is slightly shorter indicating that a slightly higher fraction of connections are associated with shorter paths. (*e.g.*, for G-07, the average length of all AS-paths is 3.2 hops while the average path length across overlay connections is 3.7 hops.)

Propagation of Traffic through the AS-Level Hierarchy: An interesting way to quantify the load that an overlay imposes on the AS-level underlay is to determine the fraction of load that is propagated upward in the AS-level hierarchy towards the top-tiered ASes. Table 2 gives the percentage of paths

Table 2. Percentage of paths/connections reaching each tier of AS hierarchy

	Tier-1		Tier-2		Tier-3	
Snapshot	Path	Conn	Path	Conn	Path	Conn
G-04	51	84	46	16	2.4	0.0
G-05	59	73	38	27	3.0	0.0
G-06	52	64	38	36	10	0.0
G-07	55	63	41	37	3.6	0.1

and percentage of overlay connections whose top AS is a tier-1, tier-2, and tier-3 AS, respectively, in each overlay snapshot. The columns marked "Path" give the percentage of the relevant AS-paths reaching each tier while the columns marked "Conn" represent the percentage of the overlay connections (*i.e.*, aggregate load) reaching each tier. We note that more than half of the paths reach a tier-1 AS, and roughly 40% of the paths peak at a tier-2 AS across all four snapshots.

The percentage of connections that reach a tier-1 AS is even higher than that for paths, indicating that a larger fraction of connections are mapped to these paths. At the same time, a lower percentage of connections reach a tier-2 AS (16% to 37%) compared to paths that peak in tier-2 ASes. Interestingly, the percentage of connections that reach a tier-1 AS decreases over time while the percentage of connections that peak in a tier-2 AS is increasing. A plausible explanation of this trend is the increasing connectivity over time between ASes in the lower tiers which reduces the fraction of connections that have to climb the hierarchy up to tier-1 ASes. A closer examination (not shown here) confirmed that this shift in traffic towards lower tiers is indeed primarily due to the presence of shortcuts between lower-tier ASes in the AS topology (*e.g.*, more aggressive peering at Internet exchange points over time). In particular, the observed shift has little to do with changes in the overlay topology, mainly because the connectivity of the Gnutella overlay has not become significantly more localized over time.

4 Conclusion and Future Work

In this paper, we studied the problem of quantifying the load that a particular overlay imposes on the AS-level underlay. We identified the challenging aspects of this problem and described existing techniques to address each of these aspects. Relying on an existing set of best practices, we presented a methodology for mapping the load of an application-level overlay onto the AS-level underlay. We illustrated our methodology with an example of a real-world P2P overlay (*i.e.*, Gnutella). While our study contributes to a deeper understanding of the interactions between application-level overlays and the AS-level underlay in today's Internet, a more detailed analysis of the sensitivity of our results to known overlay-specific issues, known underlay-related problems (*e.g.*, incomplete AS graph, ambiguous AS relationships), and known BGP-related difficulties (*e.g.*, , limitations of the C-BGP simulator) looms as important next step.

As part of our future work, we plan to investigate how changing the geographical location of peers and their connectivity affect the load imposed on the AS-level underlay. Furthermore, we plan to derive realistic traffic models for different P2P application and incorporate them into our methodology. Finally, we intend to examine pricing models that are used by ISPs to determine how structure and workload of an overlay affect the revenues of the various ISPs in the AS hierarchy of the underlay.

Acknowledgment

We would like to thank Matthew Roughan for helpful discussions, Daniel Stutzbach for sharing tools and data, and Bruno Quoitin and Wolfgang Muehlbauer for promptly answering all of our C-BGP-related inquiries. This work is supported in part by the National Science Foundation (NSF) under Grant No. Nets-NBD-0627202 and an unrestricted gift from Cisco Systems. Part of this work was performed while the first author was visiting IPAM (UCLA) in the Fall of 2008 as a participant of the "Internet Multi-Resolution Analysis" program.

References

1. Aggarwal, V., Feldmann, A., Schneideler, C.: Can ISPs and P2P systems cooperate for improved performance? ACM SIGCOMM Computer Communication Review 37(3), 29–40 (2007)
2. Choffnes, D.R., Bustamante, F.E.: Taming the torrent: A practical approach to reducing cross-ISP traffic in P2P systems. In: ACM SIGCOMM (August 2008)
3. Ge, Z., Figueiredo, D.R., Jaiswal, S., Gao, L.: On the Hierarchical Structure of the Logical Internet Graph. In: SPIE ITCom (November 2001)
4. Quoitin, B., Uhlig, S.: Modeling the Routing of an Autonomous System with C-BGP. IEEE Network 19(6) (2005)
5. Stutzbach, D., Rejaie, R., Sen, S.: Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems. In: ACM IMC (October 2005)
6. University of Oregon, RouteViews Project, http://routeviews.org
7. CAIDA, Cooperative Association for Internet Data Analysis, http://caida.org
8. Oliveira, R., Pei, D., Willinger, W., Zhang, B., Zhang, L.: In search of the elusive ground truth: the internet's as-level connectivity structure. In: ACM SIGMETRICS (June 2008)
9. Roughan, M., Tuke, S.J., Maennel, O.: Bigfoot, sasquatch, the yeti and other missing links: what we don't know about the as graph (October 2008)
10. Gao, L.: On Inferring Autonomous System Relationships in the Internet. IEEE/ACM Transactions on Networking 9, 733–745 (2000)
11. Dimitropoulos, X., Krioukov, D., Fomenkov, M., Huffaker, B., Hyun, Y., Claffy, K., Riley, G.: AS Relationships: Inference and Validation. ACM SIGCOMM Computer Communication Review 37(1), 29–40 (2007)
12. Rasti, A.H., Rejaie, R., Willinger, W.: Characterizing the Global Impact of the P2P Overlay on the AS-level Underlay. University of Oregon, Tech. Rep. CIS-TR-2010-01 (January 2010), http://mirage.cs.uoregon.edu/pub/tr10-01.pdf
13. Muhlbauer, W., Feldmann, A., Maennel, O., Roughan, M., Uhlig, S.: Building an AS-topology model that captures route diversity. ACM SIGCOMM Computer Communication Review 36(4), 195–206 (2006)

Investigating Occurrence of Duplicate Updates in BGP Announcements

Jong Han Park[1], Dan Jen[1], Mohit Lad[2], Shane Amante[3], Danny McPherson[4], and Lixia Zhang[1]

[1] University of California, Los Angeles
[2] ThousandEyes
[3] Level-3 Communications Inc.
[4] Arbor Networks

Abstract. BGP is a hard-state protocol that uses TCP connections to reliably exchange routing state updates between neighbor BGP routers. According to the protocol, only routing changes should trigger a BGP router to generate updates; updates that do not express any routing changes are superfluous and should not occur. Nonetheless, such ‘duplicate’ BGP updates have been observed in reports as early as 1998 and as recently as 2007. To date, no quantitative measurement has been conducted on how many of these duplicates get sent, who is sending them, when they are observed, what impact they have on the global health of the Internet, or why these ‘duplicate’ updates are even being generated. In this paper, we address all of the above through a systematic assessment on the BGP duplicate updates. We first show that duplicates can have a negative impact on router processing loads; routers can receive upto 86.42% duplicates during their busiest times. We then reveal that there is a significant number of duplicates on the Internet - about 13% of all BGP routing updates are duplicates. Finally, through a detailed investigation of duplicate properties, we manage to discover the major cause behind the generation of pathological duplicate BGP updates.

1 Introduction

BGP is the de facto standard inter-domain routing protocol used to exchange destination reachability information on the Internet. BGP was designed as a hard-state protocol, so all BGP updates sent by a router should always communicate some change or addition to the most current routing information reported by the router [7]. However, actual observations of BGP dynamics reveal that routers tend to occassionally send BGP updates with absolutely no change to the most current routing information reported by the router. In fact, there are many cases where routers send exact copies of the most recent update previously sent. To date, there has been no explanation as to why these ‘duplicate’ routing updates occur in BGP today.

Existence of duplicate updates in BGP was first reported in 1998. Labovitz’s [2] seminal work on BGP measurements showed that the actual number of BGP updates observed were an order of a magnitude more than expected. Labovitz revealed that a large portion of the total updates were in fact duplicates, and he attributed this to problems with routers from specific vendors. The industry quickly responded with a software fix to address the duplicate generation problem, and it was believed that the fix would eliminate the duplicate pathology observed in [2]. However, in 2007 Li *et al.* [4] re-examined the health of BGP dynamics and discovered that, despite industry attempts to stop duplicate generation, duplicates were still seen in BGP. To date, nobody has been able to determine the cause of these duplicates. There also have never been any reports on the effects, if any, that duplicates have on Internet health.

In this paper, we make the following contributions.

– We provide a better understanding of the nature of duplicate generation by quantifying the amount of duplicate updates from different points on the Internet. We also look at duplicates from different moments in time.
– We reveal the impact of duplicates on Internet health. Unlike the common belief that duplicates are relatively benign, we show that they can negatively impact the instantaneous router processing load.
– As part of our work towards understanding duplicates, we provide a methodology for mapping eBGP updates to their corresponding iBGP updates. We believe that our methodology can be useful toward future studies that require a mapping of eBGP to iBGP updates, or vice versa.
– Using our observations of duplicate behavior, we manage to finally determine the exact cause behind duplicate generation.

2 Background

In this section, we review some routing details that are particularly relevant to our study of duplicates. Specifically, we discuss the definition of 'duplicate updates' and BGP peering topologies.

2.1 Definition of Duplicates

A BGP update for prefix p sent by router r is a 'duplicate' if and only if all attributes in the update are the same as the most recent previous update for prefix p sent by router r, and both the update and the previous update belong to the same BGP session.

2.2 BGP Peering Topologies

Today, BGP is used for both inter-domain routing (eBGP), as well as intra-domain routing (iBGP). Here we briefly describe the common peering topologies for both inter and intra-domain routing.

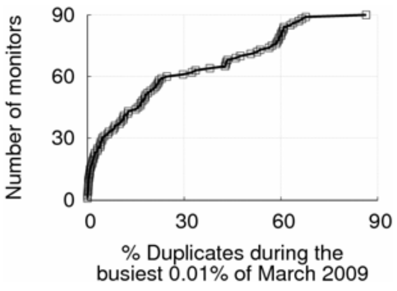
External BGP: When BGP is used to convey reachability information between two routers that reside in different domains (inter-domain routing), the session between these two routers is called an eBGP session. The routing information in each update is conveyed in the form of BGP attributes. Some of the more relevant attributes to this paper are Next-hop, MED, Local-pref, and Community.

Internal BGP: iBGP is used to distribute reachability information received from eBGP peers to routers within one domain. To avoid forming a routing information loop, it was originally required that all iBGP speakers are fully meshed and the reachability learned from an iBGP speaker is not propagated to another iBGP speaker. In practice, this approach is not scalable and too expensive to manage. This leads to the use of route reflection (RR) [1] and AS confederations [8], which relaxes this full-mesh requirement among all iBGP peers. However, having to traverse more than one hop for an update from an iBGP peer to another iBGP peer re-introduces the possibility of routing information loops under both schemes. To avoid forming a routing loop, route reflection and AS confederation define new attributes, namely Cluster-list and AS-confed-sequence respectively, and use them in the similar way that AS-path is used in eBGP.

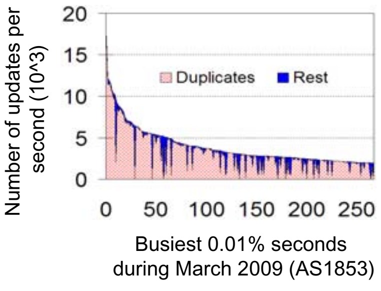
iBGP and eBGP interaction: A router that peers with both iBGP peers and eBGP peers changes or even removes certain attributes when it sends reachability information received from an iBGP peer to its eBGP peer. Some attributes defined both in iBGP and eBGP such as Next-hop, MED, and Local-pref may have changes in their values before sent out to eBGP peers. Furthermore, certain attributes that are only defined in iBGP such as Cluster-list and AS-confed-sequence are removed and not sent out to eBGP peers.

3 Impact of Duplicates on Routers

We start by measuring the impact that duplicates have on Internet health. Up until now, it was believed that duplicates do not hinder routing efficiency in any significant way [3]. However, we find that duplicates are responsible for the majority of router processing loads during their busiest times. Previous studies have shown that higher processing loads can lead to more session resets, routing loops, and packet losses [9]. Thus, we measured how much duplicates contribute to the router processing loads during their busiest times during the month of March 2009. We define ‘busiest times’ as the top 0.01% of seconds within which the largest number of updates were generated. Our data set consists of a specific subset of all RouteViews/RIPE monitors. The monitors were carefully chosen such that each monitor was available for the entire month of March 2009 and that there was at most one monitor per AS in our dataset. The number of stub,



(a) Distribution of % duplicates



(b) # of duplicates during the busiest seconds

Fig. 1. Impact on processing loads

Table 1. Aggregated number of updates and duplicates

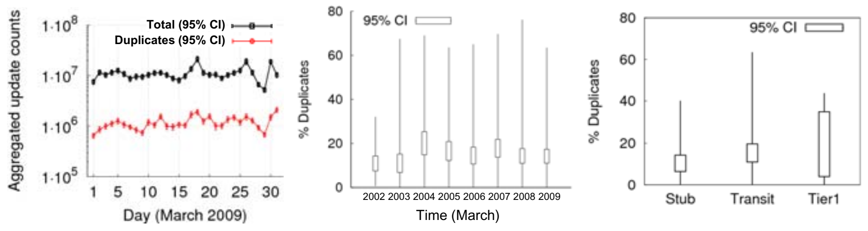
Year	2002	2003	2004	2005	2006	2007	2008	2009
Number of selected monitors	27	37	54	67	79	100	109	90
Number of total updates (10^6)	129.5	207.3	316.4	426.5	423.7	511.2	652.2	677.4
Number of duplicate updates (10^6)	12.7	32.0	68.9	74.6	63.8	137.1	111.0	91.3

transit, and tier-1 monitors we ended up with were 27, 55, and 8 respectively, for a total of 90 monitors. We preprocess all of our data using the minimum collection algorithm (MCT) [10] to filter out updates due to session resets before performing any of the measurements presented in this paper.

Figure 1(a) shows the percent of duplicate traffic during busiest times for each of the 90 ASes we monitored. Notice that for 22% (20 out of 90) of all monitored ASes, duplicates contribute 50% or more of the update traffic during busiest times. Later in Section 5, we describe how these duplicate bursts are generated in detail as we reveal the causes of duplicates. Figure 1(b) is a close-up look at a particularly bad case of our measurement, AS1853. Overall, 86.42% of total updates during the top 0.01% of busiest times were duplicates. During the busiest second the router in AS1853 had to process about 175,000 updates in Figure 1(b).

4 Understanding Duplicates across Time and Space

Now that we understand the negative impact that duplicates can have on Internet health, we analyze duplicate generation in detail to gain a better understanding of this duplicate pathology, and maybe even discover the cause of duplicate generation. Not only do we measure the prevalence of duplicates updates on the Internet today, we also measure the number of duplicates that we have seen over the past few years. We then explore whether topological factors (such as size of AS or connectivity) show any correlation with occurrences of duplicates. Our data set consisted of the same 90 monitors we used for our measurements in section 3.



(a) During March 2009 (b) From 2002 to 2009 (c) From different AS type

Fig. 2. Amount of duplicate updates

4.1 Are Duplicates Observed at All Times?

Figure 2(a) shows the amount of duplicates along with the total number of updates from all 90 monitors during March 2009. It turns out that duplicate generation is not just a pathological behavior rarely seen on the Internet. In this month alone, the total aggregated number of updates was about 677 million. Among those, about 91 million updates were duplicates. Thus duplicates make up 13.4% of aggregated BGP traffic.

Figure 2(b) shows how long duplicates have existed in BGP by showing the maximum, minimum, and 95% confidence intervals of % duplicates observed by different monitors for the month of March from 2002 through 2009. For each year, we selected monitors based on the criteria described in section 3. Table 1 shows the number of monitors we used from 2002 through 2009. The number of qualified monitors generally increase over time, mainly because more ASes peered with RouteViews and RIPE over time.[1] We performed the same measurement for other months from 2002 through 2009, and the results were all similar. The amount of duplicates we counted also agree with the amount observed in previous studies [4].

4.2 Are Duplicates Observed from All Networks?

Our next measurement is aimed at understanding if size or type (e.g. stub, tier-1) of network has any correlation with observed duplicates. We measured the percentage of duplicates out of total updates that each network generated for the month of March 2009.

Figure 2(c) summarizes our findings. All three types of networks generate duplicates with some variation in their percentages. The large confidence interval range for tier-1s is mainly due to the small number of data points available to us. Minimum % duplicates were very low in all three cases. At the same time maximum % duplicates were quite high for all types, showing a large variation in behavior even amongst networks of the same type. Later in section 6, we discuss why the amount of duplicates observed varies so widely amongst networks of the same type.

4.3 Where Do Duplicates Originate?

So far, we have observed duplicates from different monitors. However, we do not quite know where these duplicates originate. By specification, a BGP router should not propagate a duplicate it receives. Thus, when we observe a duplicate at AS X with a path X-Y-Z, where Z is the origin AS, we hypothesized that the duplicate message must be generated by X and not by Y or Z. Our next exercise is to verify our hypothesis.

For this, we looked specifically at duplicates for particular prefixes where the following was true. First, the observed duplicate for prefix p from AS X had an AS-path ending with X-Y. Second, we had to have monitors for both AS X and AS Y. With this, we can see whether the duplicates actually originate at (or within) AS X, or whether

[1] The exception was between 2008 and 2009. This was because some of the collectors in RIPE had problems during March 2009, and we did not use any monitors that did not have complete data for the month.

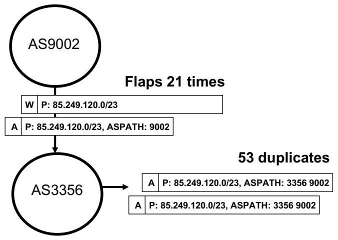

AS9002
Flaps 21 times
W P: 85.249.120.0/23
A P: 85.249.120.0/23, ASPATH: 9002
AS3356
53 duplicates
A P: 85.249.120.0/23, ASPATH: 3356 9002
A P: 85.249.120.0/23, ASPATH: 3356 9002

Fig. 3. External view of duplicate generation

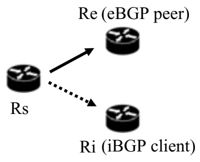


Fig. 4. Data collection

they were sent to X from Y. Our case study consisted of prefix 85.249.120.0/23 advertised by AS 9002, a direct customer of AS 3356. We had monitors in both AS 9002 and AS 3356.

Figure 3 summaries our results. During March 2009, AS 9002 announced and withdrew prefix 85.249.120.0/23 21 times. Upon receipt of these announcement and withdrawal pairs, AS 3356 sends out the announcement to the monitor with prepended AS-path, but AS 3356 never sends the withdrawal. Instead, AS 3356 sends a duplicate announcement to our monitor. In total, AS 3356 generates 53 duplicates on prefix 85.249.120.0/23 after receiving 21 pairs of announcement and withdraw messages. Not only does this observation back up our hypothesis that the sender of duplicates is the originator of duplicates, but it also suggests that the cause of duplicates may have something to do with the way internal topology dynamics interact with eBGP updates.

5 Discovering the Cause of Duplicates

Once we suspected that duplicates may be generated due to some interaction between iBGP and eBGP, we ran an experiment designed to compare eBGP update+duplicate pairs, match them with their iBGP counterparts, and compare these iBGP updates to see what we might learn about duplicate generation.

5.1 Passive Measurement Using iBGP and eBGP Data

Our first step was to obtain the data needed for our investigation. We teamed up with a tier-1 ISP who provided us with access to both iBGP and eBGP updates generated by one of their routers. Figure 4 illustrates our data-collection setup. R_s is the router sending updates to our two collector boxes, R_i and R_e. R_i is configured as an iBGP client of R_s (*i.e.* route reflector client), collecting iBGP data from R_s. R_e is an eBGP peer of R_s, collecting eBGP updates from R_s. Both the iBGP and eBGP sessions have their MRAI timers disabled, so that R_s will send updates to our collectors as R_s has updates to send.

Now that we obtained the necessary data, we needed a way to match up eBGP updates to their corresponding iBGP updates for comparison. There are two challenges in mapping iBGP update sequence with that of eBGP. First, the time that two updates, triggered by the same event, are sent out from R_s can be different. This is due to the non-deterministic nature of R_s. Second, R_i and R_e's system clocks may not be synchronized. We resolve these timing issues by introducing the notion of update 'signatures', which we now describe.

$$sig(u) = \text{peer} \parallel \text{asn} \parallel \text{prefix} \parallel \text{aspath} \parallel \text{origin} \parallel \text{comm} \parallel \text{agg}$$

The signature of an update contains all of BGP's transitive attributes that should be the same in R_s's updates to either R_i or R_e. By using the notion of signature, we calculate the time differences t_d observed between eBGP updates and their iBGP counterparts. We first generate signatures of all updates received during the t_{ebgp} second, and then search for the second in iBGP, t_{ibgp}, that yields the maximum fraction of matched signatures. In our case, the peak fraction of matched signatures was about 0.7 at a lag value of 0 (*i.e.* t_d = 0). The remaining 0.3 were dispersed within a 10-second range centered at t_{ebgp}. This means that the system times of R_i and R_e have synchronized system clocks to the second precision.

After discovering t_d, we were able to map eBGP updates to their iBGP counterparts using a heuristic algorithm involving signature and timestamp comparisons. We collected one day of iBGP and eBGP updates, putting them in sequential order as sent from R_s. We start with the first eBGP update in the sequence. As we moved down the sequence, we kept per-prefix history of signatures for every update we encounter for a time window of 60 seconds. For each eBGP duplicate update for prefix p we found as we moved forward, we looked at the corresponding iBGP time window to find a match for the sequence of signatures we recorded in eBGP for this prefix p. We say the sequence has a match when there is the *exact* sequence of update signatures within the iBGP time window. Using our heuristic, we were able to match 95.61% of eBGP update+duplicate pairs to their iBGP counterparts. 4.39% of eBGP update+duplicate pairs could not be mapped to any iBGP counterparts. The missing pairs were due to how the router processed updates.[2]

After mapping eBGP updates to their iBGP counterparts, we took each eBGP update+duplicate pair and compared the contents of their corresponding iBGP updates. For 100% of the 176,266 matched ebgp+duplicate pairs, we observed that their iBGP counterparts had differing non-mandatory attribute values. Table 5 shows our results. 0.15% of pairs were exceptions, only differing in MED values. For the other 99.85% of eBGP update+duplicate pairs, we observed corresponding iBGP update pairs with either Cluster-list and/or Originator-id differences. These attribute differences represent changes in intra-domain routing path selections.

5.2 The Cause of Duplicates

The results of our experiment allowed us to determine the main cause of eBGP duplicate updates. Our theory proved to be correct; duplicates are caused by an *unintended*

[2] When two or more updates are received on the same prefix in a very short time, the router sometimes sends out different number of updates to different peers. So, there were cases that the number of updates sent to iBGP client is different than that of updates sent to eBGP peer, in which case we declared that there is no match.

eBGP duplicate count	% Total	Observed iBGP differences
173,594	94.77	Cluster-list only
244	0.13	Cluster-list and others
1,371	0.75	Originator-id and others
1,057	0.58	Cluster-list + Originator-id + others
269	0.15	MED
6,647	3.63	No match found
Total: 183,182	100.00	

Fig. 5. Matched iBGP updates

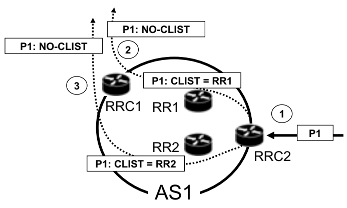


Fig. 6. Inferred cause of duplicates

interaction between eBGP and iBGP. The reason that duplicates are generated is that routers are receiving updates via iBGP which differ in iBGP attribute values alone, and thus the router believes the updates to be unique. However, once the router processes the update, strips the iBGP attribute values, and sends the update to its eBGP peer, the two updates look identical from the point of view of the eBGP peer.[3] Figure 6 illustrates a case where duplicates are generated due to changes in an iBGP attribute (Cluster-list in this case).

The main cause of eBGP duplicate updates showed that certain iBGP attribute changes (Cluster-list and Originator-id) can generate eBGP duplicate updates. We wondered if other iBGP attribute changes might also generate eBGP duplicate updates. To check for this, we performed a simple controlled experiment. We set up two ASes (AS1 and AS2). In AS1, we placed a BGP update injector and a router R_1. The injector maintains an iBGP session with R_1 and sends controlled iBGP updates. R_1 peers with a router, R_2, in AS2 using an eBGP session.[4]

After injecting pairs of iBGP updates that only differ in one attribute, we observed that a pair of iBGP updates differing in either Next-hop, Local-pref, or MED attributes will generate an eBGP duplicate update.

The experiments we have done so far shed light on how the duplicate bursts, which we discussed in Section 3, are generated. When a router used to reach a set of prefixes fails, this failure (or flapping) event generates updates that only differ in Next-hop for the set of prefixes. All of these updates become duplicates as they are sent to the eBGP peers. Using the iBGP/eBGP data collected from our tier-1 ISP, we verified that indeed duplicate bursts are preceded by an iBGP route flapping.

6 Differences in the Amount of Observed Duplicates

As observed in 4.2, ASes of the same type vary in the proportion of duplicates they generate. One reason may be a difference in MRAI timer settings amongst the networks.

[3] In our study, we observed that duplicates are generated due to changes in Cluster-list and Originator-id oscillations under route reflection. In a similar way, we believe ASes using AS confederation architectures will also generate duplicates due to the use of a non-mandatory non-transitive attribute named AS-confed-sequence, which is essentially the AS confederation version of the Cluster-list attribute under route reflector architectures.

[4] Here, R_1 is a Cisco 7200 router running IOS v12.2, and R_2 is a Quagga router which we use as a BGP update collector.

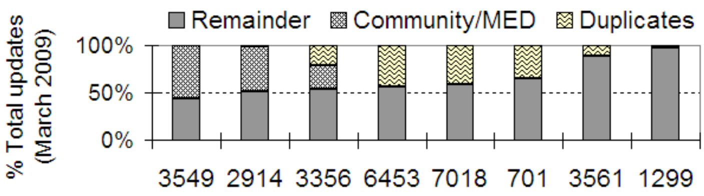


Fig. 7. Other potential noises

Duplicates are generated during internal routing changes. During the changes, updates come in bursts, and thus MRAI timers can prevent many updates from being sent.

MRAI timer differences do not fully explain why the amount of observed duplicates varies so much from one AS to another. During our experiments involving eBGP and iBGP interactions, we noticed that Cluster-list changes were often coupled with a change in Community or MED attribute values. In these cases, we observed potentially wasteful updates with fluctuating Community/MED values rather than duplicates. We asked operators at our tier-1 ISP and they confirmed that this was quite deliberate; routers were configured to make changes in certain transitive attribute values whenever there was a change in certain non-mandatory attribute values in accordance with [5,6]. [5,6] suggests using Community attribute values as a general purpose attribute to convey informational tags as well as action tags to receiving networks. MED values were also used for traffic engineering purposes. However, operators admit that not all peers need or use this Community information, and for those routers that do not use the Community information, these BGP updates are as useless to them as duplicates. However, such updates can be more detrimental than duplicates in one significant way; with duplicates, the negative impact is limited to the direct neighbors. As described earlier, duplicates do not travel more than one hop. However, if some other (optional) transitive attributes such as Community is changed, then the update is no longer a duplicate and can potentially be propagated more than one hop.Community value changes are not useful to networks that are more than one hop away, and yet these networks still must suffer the same negative impacts of receiving a superfluous BGP update.[5]

Our discovery of these potentially wasteful BGP updates led us to wonder if other ASes generated similarly potentially wasteful updates. We looked at all updates from tier-1s observed by our monitors for the month of March 2009, and classified the updates into 3 types - duplicates, Community/MED change, and remainder. Figure 7 shows our results. While AS3549 and AS2914 generated almost no duplicates, 50% or more of their total updates were Community/MED change updates. We suspect that many of these updates could be useless to many networks that receive the update. We intend on verifying our suspicion in future work.

[5] Different router vendors implemented different default behavior in sending Community attribute. For ISPs that use network equipment where the default behavior is to send communities (*e.g.* Juniper), then the effect of this problem are likely to be amplified. However, for ISPs that use network equipment where the default behavior is to not send communities by default (*e.g.* Cisco), then the effect of this problem are likely to be less.

7 Conclusion

In this paper, we conducted the first comprehensive measurement study quantifying the prevalence of duplicates on the Internet across space and time. We discovered that duplicates make up over 10% of all BGP update traffic. We examined the impact that duplicates have on the overall health of the Internet, and discovered that routers can receive upto 86.4% duplicates during their busiest times. We developed a heuristic to match eBGP updates with their corresponding iBGP counterparts. Finally, we combined our observations with our heuristic to discover the causes of duplicates on the Internet - duplicates are caused by an *unintended interaction* between iBGP and eBGP.

While pure duplicates are clearly unnecessary BGP overhead, our work revealed that duplicates may not be the only superfluous BGP updates floating around on the Internet. As described in section 6, updates that couple non-transitive attribute changes with transitive attribute changes may not be useful to all recipients. It would be interesting to identify all forms of superfluous BGP updates and gain an exact measure of how much BGP traffic is simply unwanted noise. We hope that our work allows the Internet community to take a significant step towards a optimal and clean routing communication system.

References

1. Bates, T., Chen, E., Chandra, R.: RFC 4456. In: BGP Route Reflection: An Alternative to Full Mesh Internal BGP (April 2006)
2. Labovitz, C., Malan, G.R., Jahanian, F.: Internet routing instability. ACM/IEEE Transactions on Networking 6(5), 515–528 (1998)
3. Labovitz, C., Malan, G.R., Jahanian, F.: Origins of internet routing instability. ACM/IEEE Infocom 1, 218–226 (1999)
4. Li, J., Guidero, M., Wu, Z., Purpus, E., Ehrenkranz, T.: BGP Dynamics Revisited. In: ACM Sigcomm Computer Communications Review (April 2007)
5. Meyer, D.: RFC 4384. BGP Communities for data collection (2006)
6. Steenbergen, R.A., Scholl, T.: BGP Communities: a guide for service provider networks (2007)
7. Traina, P.: RFC 1774. In: BGP-4 protocol analysis (1995)
8. Traina, P., McPherson, D., Scudder, J.: RFC 5065. In: Autonomous System Confederations for BGP (August 2007)
9. Wang, L., Zhao, X., Pei, D., Bush, R., Massey, D., Mankin, A., Wu, S.F., Zhang, L.: Observation and analysis of BGP behavior under stress. In: IMW 2002: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment, Marseille, France, pp. 183–195. ACM, New York (2002)
10. Zhang, B., Kambhampati, V., Lad, M., Massey, D., Zhang, L.: Identifying BGP routing table transfers. In: MineNet 2005: Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data, pp. 213–218. ACM Press, New York (2005)

A Measurement Study of the Origins of End-to-End Delay Variations

Yaron Schwartz, Yuval Shavitt, and Udi Weinsberg

School of Electrical Engineering
Tel-Aviv University, Israel

Abstract. The end-to-end (e2e) stability of Internet routing has been studied for over a decade, focusing on routes and delays. This paper presents a novel technique for uncovering the origins of delay variations by measuring the overlap between delay distribution of probed routes, and how these are affected by route stability.

Evaluation is performed using two large scale experiments from 2006 and 2009, each measuring between more than 100 broadly distributed vantage points. Our main finding is that in both years, about 70% of the measured source-destination pairs and roughly 95% of the academic pairs, have delay variations mostly within the routes, while only 15-20% of the pairs and less than 5% of the academic pairs witness a clear difference between the delays of different routes.

1 Introduction

The Internet has evolved in recent years to become a complex network, with increasing usage of load-balancing and traffic shaping devices. These devices change the way packets flow, therefore affecting the observed stability of routes and delays between hosts. This, in turn, affects various delay and jitter sensitive applications, such as VoIP and IPTV. On the other hand, load on devices is not constant and may change the delay packets observe along the same route significantly. Therefore, it is important to understand both the delay stability along the path and to identify the source of the delay variability when such variability exists.

Wang *et al.* [1] and more recently Pucha *et al.* [2] studied the impact that specific routing events have on the overall delay. They showed that although routing changes can result in significant round trip delay increase, their variability is small for most of the measured path transitions, therefore allowing applications to make use of such stability.

Augustin *et al.* [3] examined the delay between different parallel routes at a short time epoch. They compared the minimum delay of each route, and found that only 12% have a delay difference which is larger than 1ms. Using similar techniques, Pathak *et al.* [4] studied the delay asymmetry and found that there is a strong correlation between changes in the one-way delay and corresponding route changes.

Unlike previous work, we study the RTT delay along longer time periods, hours and days, and examine how different is the delay *distribution* between parallel routes. For this purpose we use delay samples to define an interval in which the delay of each route resides, and look at the overlapping between intervals of parallel routes. If the two intervals are disjoint we know that the e2e delay value mostly depends on the route in use and not on the variance in the route. As the overlap between the intervals increases, the delay variance is mostly attributed to changes along the route itself, e.g., due to change in load.

Evaluation is performed by conducting two large-scale experiments in 2006 and in 2009. Using DIMES [5], a highly distributed community-based measurements infrastructure, we planned these two 96-hours experiments each utilized more than 100 actively measuring vantage points (VPs), located in a broad set of ASes and geographical locations, contributing more than 200k e2e routes.

Our main finding is that in about 70% of the measured source-destination pairs, in both experiments, the delay variations are mainly explained by changes within the routes, while only 15-20% of the pairs witness a clear difference between the delays of different routes. The remaining 10-15% of the pairs witness a mixture of the above, with a higher tendency for intra-route changes as contributors to the delay variance. Pairs that have their source and destination in academic ASes exhibit much higher route stability, which further increases the percentage of delay variations within the routes to 95%.

2 Quantifying Route and Delay Stability

2.1 Definitions

The input data is a collection of traceroute measurements for a set P of ordered source destination pairs, $P_i = \{S_i, D_i\}$. For each pair, P_i, the set of e2e IP-level traceroutes, TR_i, is partitioned into k_i equivalence subsets (i.e., any two traceroutes in each subset are the same), denoted by E^i. The size of the subset $|E^i_j|$ is the total number of traceroutes it contains. Each equivalence subset $E^i_j, 1 \leq j \leq k_i$ has a single representing route $R(E^i_j)$ which is the measured path between the source and the destination.

For each pair P_i we define the *dominant route* as the route $R(E^i_j)$ whose subset size, $|E^i_j|$, is the largest. It is possible that several equivalence subsets have the same size, therefore they are all considered dominant routes. For brevity, we assume for now that each pair has a single dominant route, with index r.

2.2 Measurement Setup

The data used in this paper is obtained from DIMES [5], a community-based Internet measurements system. DIMES performs active measurements using hundreds of software agents installed on users' PCs. Agents perform roughly two measurements per minute (either traceroutes or ping using either ICMP or UDP) by following a script that is sent to them from a central server.

DIMES provides researchers with the ability to run "experiments" by defining the set of agents, probing protocols and a set of destinations. Since some agents are installed on end-users machines, the number of measurements may vary depending on their availability. Usually, more than 80% of the planned measurements are performed.

For the purpose of this paper we performed two similar experiments that took place in December 2006 and September 2009. In each experiment, we selected over 100 globally distributed agents and designed 96-hours experiments in which each agent executed UDP and ICMP traceroute measurements to all other agents in a round robin fashion. For each traceroute measurement we take the minimum delay of at most four probes sent over a period of a few seconds (in case of a lost probe we do not send another one instead). Since DIMES is a community based platform not all of the agents are constantly active during the experiments period. Moreover, since there is a certain churn in users along time, not the same agents were selected in both experiments. Thus, 120 agents were selected, making sure that there will be valid results from more than 100 agents. The scripts we wrote had one UDP and one ICMP measurement to each of the 120 destination IP addresses. Therefore, an agent probes each IP address twice every two hours. Agents repeated the same script for four days. In total, each of these experiments result in over one million traceroute measurements results.

Note that traceroutes probe the forward-path of routes, while the delays are round-trip. Pathak *et al.* [4] analyzed the delay asymmetry and showed that one-way delay can be different than round-trip, meaning that it is possible that our delay measurements actually capture instability that exists in both the forward and reverse paths. Following Pucha *et al.* [2], we analyzed the stability of routes as measured from opposite directions in our dataset, and found that over 90% of the pairs have forward and reverse path RouteISM that are different by less than 0.3 (not shown due to lack of space). This indicates that the stability of the forward path can serve as an indication to the reverse path. We attribute this to the observation that even non-symmetric routes share similar hops that can contribute instability to both directions. Thus, comparing the instability of RTT delay with the routing instability of the forward route is meaningful.

2.3 Pair and Route Identification

When comparing two routes we seek to answer if they are equal and if not, quantify their difference. Several difficulties arise in both aims. Since DIMES is a community-based project, most traceroutes start with several private IP addresses before reaching the routable Internet. Moreover, some use laptops and may travel during the time of the experiment. In order to decrease the chance of over-estimating instability, only the routable section of each traceroute is considered for the analysis. The identification of a pair is done using the first and last hops of the routable traceroute. This help us mitigate instability that might appear in the non-routable networks, which are presumed to have little affect on the overall delay instability. In the analysis, we only include pairs that witness at least 20 traceroutes.

Two (routable parts of) traceroutes are considered equal when their ordered list of IP addresses are exactly the same. To quantify the difference between two traceroutes we calculate their Edit Distance [6] (ED) value by counting the minimal number of insert, delete, and modify operations that are needed in order to make the two routes equal. Obviously, ED is highly correlated with the length of the compared routes. To be able to compare ED values that are calculated on routes with various lengths, the ED is normalized by the length of the longest route of the two input routes. This technique is similar to the one described by He *et al.* [7] who used it for quantifying AS-level asymmetry. We extend here the technique to consider stability instead of symmetry. Since the ED cannot be greater than the longest route, the normalized ED value is between 0 and 1, where 0 means that the two routes are identical and 1 means that they are completely different.

2.4 Route Stability

We use two methods for quantifying the stability of a route. The overall appearance ratio (i.e., prevalence [8]) of a route with index j, i.e., $R(E^i_j)$, in pair P_i is the portion of traceroutes in the set E^i_j. The prevalence of the dominant route $R(E^i_r)$ is used as the first indication to the stability of routing for each pair, since having a dominant route with high prevalence suggests that the remaining paths are relatively rare.

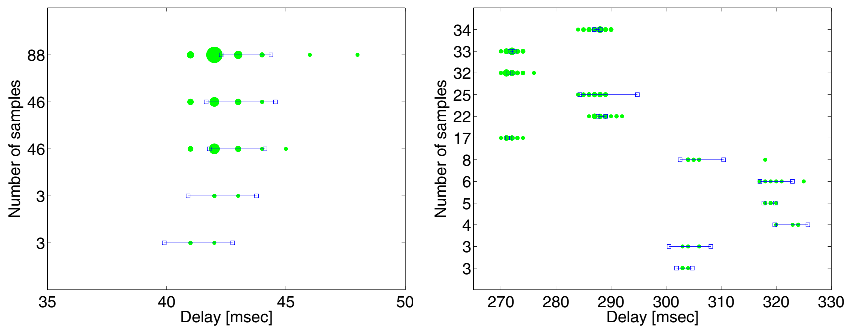
The second estimation of pair P_i stability is calculated by finding the normalized ED between the dominant route, $R(E^i_r)$, to all other non-dominant routes, $R(E^i_j), j \neq r$. For pairs that have more than a single dominant route, we use the dominant route that is closest to each route in number of hops. We define the *Route Instability Measure* (RouteISM) of a pair as the weighted average of all normalized ED measures as depicted in Eq. (1). Thus, an ISM value close to 1 indicates high instability.

$$RouteISM_i = \sum_{j \neq r} \left(|E^i_j| \cdot \widehat{ED}^i_{jr} \right) / \sum_{j \neq r} |E^i_j| \qquad (1)$$

Two techniques were used in the past to measure distance between routes. Pucha *et al.* [2,4] defined the similarity coefficient for calculating AS level route symmetry as the number of similar elements divided by the total number of distinct elements in the two routes $\frac{|P_i \cap P_j|}{|P_i \cup P_j|}$. He *et al.* [7] used string matching which is similar to our ED. We follow the latter and argue that ED better captures stability since it takes into account the *order* of elements in each route.

2.5 Delay Stability

We are interested in the expected e2e round trip delay of a route over time and not in short term congestion. Recall that we take the minimum delay of at most four probes sent over a period of a few seconds, and repeat each traceroute roughly twice an hour (UDP and ICMP) over a period of four days.



(a) 217.0.116.82 → 80.91.184.206 (b) 69.134.208.1 → 134.159.160.58

Fig. 1. Examples of pairs with overlapping and non-overlapping confidence intervals. The segments show the confidence intervals of a routes, calculated using the delay samples which are shown as varying sized circles (larger radius means more samples).

For a given pair P_i, each equivalence set E^i_j, has several different e2e RTT delay samples (henceforward "delays"), denoted by $RTT(E^i_j)$. We wish to quantify the stability of pair delays and whether their variance is the result of delay dynamics of each route or delay difference between different routes. This analysis can uncover whether delay instability is mainly the result of traffic anomalies in a route (e.g., congested routers), or the result of route diversity due to load-balancers.

For each route E^i_j, we have the group $RTT(E^i_j)$ of delay measurements. To find the region of expected delay for the route, we treat the measured delays as samples of some distribution and calculate the average and the 95% confidence interval [9] around it. This confidence interval, denoted by $CI(E^i_j)$, provides us with a segment surrounding the measured mean of $RTT(E^i_j)$. Within this interval we expect to find the route delay. Note that this is an unorthodox use of confidence interval, but we believe it gives us a good characterization of the expected route delay (as is nicely shown in Fig. 1). Measurements with high variance result in larger segments than measurements with small variance, indicating that they are less stable.

For a source-destination pair, the normalized overlap between two segments $CI(E^i_j)$ and $CI(E^i_k)$ is defined by

$$\widehat{O}^i_{jk} = \frac{CI(E^i_j) \bigcap CI(E^i_j)}{\min\{|CI(E^i_j)|, |CI(E^i_j)|\}}, \forall j \neq k \qquad (2)$$

The normalized overlap is equal to 0 when the two segments do not overlap, meaning that their delays are significantly different. This indicates that changes in the route delay are mainly the result of having different routes. When it is equal to 1, the segments completely overlap or one contains the other, meaning that different routes exhibit similar delay distribution, indicating that instability is not the result of multiple routes between the source and destination, but due to changes within the routes. For example, Fig. 1(a) shows the routes from Deutsche Telekom in Germany to Datagroup in the Ukraine. There are five different routes with more than 30 measurements (the y-axis label is the number of measurements per route) but they are all overlapping, namely they have roughly the same delay average. On the other hand, Fig. 1(b) shows the routes from ParaCom Technologies in USA to Reach Networks in Australia. While in the previous figure the delay changes are attributed to variance of delays inside the routes, this figure clearly shows that the delay changes is the result of multiple routes with four distinct mean delays.

When the number of measurements in the route is small, the statistical significance of the samples is small, and the confidence interval can be very large and not meaningful. Fig. 4(a) shows that for 80% of the routes the confidence interval is below 0.2 of the average delay. Since the number of routes with statistical significance change between pairs, we calculated the overlap only between the two largest equivalence groups (routes) of each pair, providing each has at least 30 delay measurements.

3 Dataset Analysis

3.1 Distribution of Vantage Points

Using a community based platform, results in a certain churn in the availability of measuring agents. Therefore, during the planning of the experiments, we selected measuring agents that hold all of the following criteria: (a) they were active in the past week, (b) distributed in a large set of ASes, and (c) distributed in a large set of geographical regions. The first criterion is to maximize the chance that the selected agent will indeed be active during the experiment period. The other two criteria were selected to achieve e2e routes with diverse lengths that traverse through various ASes spread across different countries and continents, as an attempt to capture an accurate image of the Internet [10].

In the 2006 experiment, 102 agents returned slightly over a million traceroutes, providing us with 6861 source-destination directed pairs. Most VPs are distributed in the USA and Canada (70), followed by Western Europe (14), Australia and New Zealand (10), Russia (6) and Israel (2). In the 2009 experiment, 105 agents returned 1.01 million traceroutes, resulting in almost 10950 source-destination directed pairs. VPs are distributed in numerous countries in Western Europe (41), followed by USA and Canada (38), Russia and the Ukraine (14), Australia (4), South America (2), Israel (2), Japan (1), Taiwan(1), Singapore (1) and the Maldives (1).

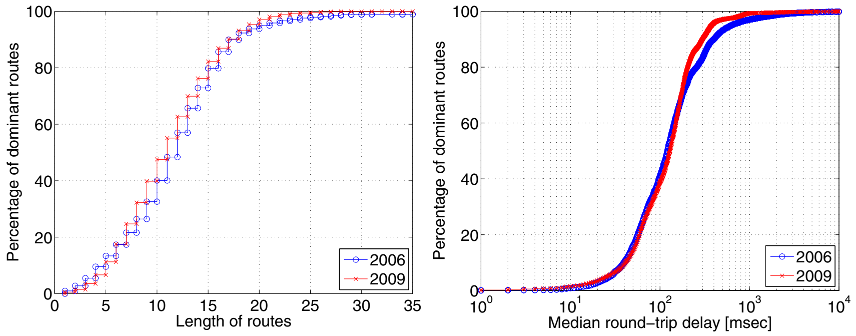
Using the list of AS types provided by Dimitropoulos *et al.* [11], we infer the type of each VP. In 2006, 18% of the VPs are tier-1, 78% tier-2, 3% smaller companies and 1% educational (a single agent). In 2009, DIMES agents were installed in PlanetLab [12], which increased the number of educational VPs to 28% while reducing tier-1 VPs to 14% and tier-2 to 58%. Only 7 VPs appeared in both experiments. This is due to the change in users that are running DIMES agents over this time period.

In both experiments, a variety of ASes were traversed. Most of the tier-1 ASes were traversed and the majority of the traversed ASes are tier-2.

3.2 Dataset Statistics

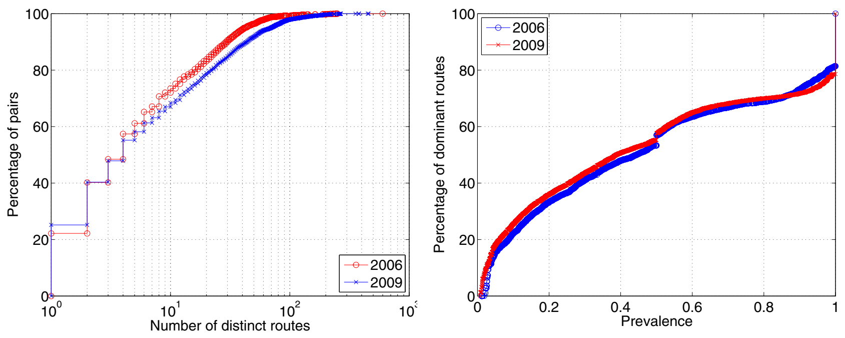
The cumulative distributions of the dominant route length and dominant route median delay are shown in Fig. 2 (recall that there can be more than one dominant route per source-destination pair). Fig. 2(a) shows that both experiments have roughly the same path lengths, with 2009 being slightly shorter. The median of the dominant route length is 12 for 2006 and 11 for 2009; pairs with academic source and destination ASes have even shorter routes, with median of 11 hops in 2009; the majority of the routes (97%) traverse less than 20 hops. Our measured routes are shorter than reported by Paxson [8] in 1995. Paxson reported mean route length between 15 and 16, using routable (and mostly academic) source and destination hosts. Since the Internet has been growing at high rate since 1995, we attribute this reduction to the reacher connectivity among ASes and increased adoption of layer-2 tunnels, which significantly reduces the number of IP-level hops.

Fig. 2(b) exhibits an almost identical median delay distribution of 2006 and 2009, with 2009 having slightly shorter delays, which correlates with the shorter paths witnessed in Fig. 2(a). Over 80% of the routes in both years have a delay of less than 200msec. However, there are almost 3% of the routes that have a delay of over 1 second. Pairs that have end-points in the USA are have shorter delays, with 80% of them having a delay less than 150msec. However, pairs with academic end-points have significantly shorter delays, with 90% of them having a delay of less than 100msec.



(a) Route Length (b) Median Delay

Fig. 2. Cumulative distributions of route lengths and median delay



(a) Distinct routes per pair (b) Dominant route prevalence

Fig. 3. Cumulative distributions of distinct routes and prevalence, showing (a) the number of distinct routes per pair, and (b) the prevalence of the dominant route

4 Results

4.1 Route Stability

Fig. 3 shows the cumulative distributions of distinct routes per pair and prevalence of dominant routes. The figures show that over 20% of the pairs in 2006 and almost 30% of the pairs in 2009 witness a single route. Fig. 3(b) has a clear jump at 50% prevalence, which we attribute to load-balanced routes with equal per-packet balancing. This jump is not visible in routes between academic end-points, due to their minimal usage of load-balancers. Furthermore, over 55% of the pairs that have both source and destination in academic ASes, which is the case when using PlanetLab, have a single route. Pairs that have both end-points in the USA have slightly higher route stability, with roughly 35% of them having a single route. These observations stress the need for a diverse set of VPs when doing e2e Internet analysis.

Analysis of the RouteISM (not shown due to lack of space) supports the observation of an overall stable e2e routing in the Internet, as over 90% of the pairs (and 95% of the academic pairs) have RouteISM smaller than 0.2. This values is used in Sec. 4.2 as a threshold between stable and non-stable pairs.

4.2 Origin of Delay Instability

We first show that our use of confidence interval is meaningful. Fig. 4(a) plots the cumulative distribution of the ratio between a route's confidence interval and its mean delay. The figure shows that, for both years, 90% of the routes have a ratio of less than 0.25. This indicates that the delay confidence intervals are not 'too long' in general, and extend only for routes with large variance (as shown in the examples in Fig. 1).

Fig. 4(b) shows that for both data sets, over 40% of the pairs have an overlap of 1 and an additional 30% of the routes have overlap of over 0.8. Namely, in 70%

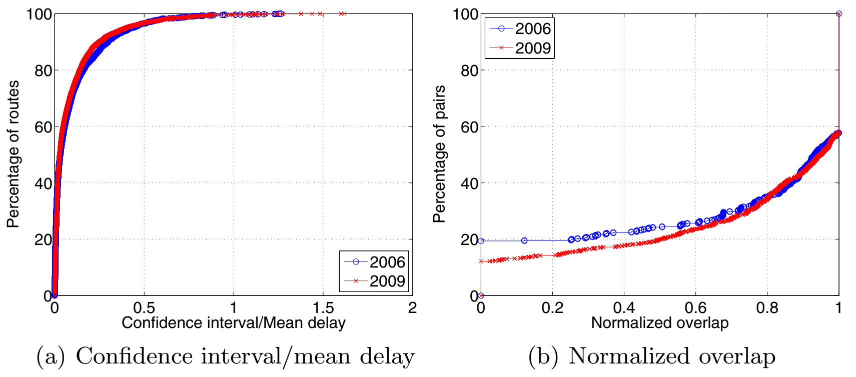


Fig. 4. Confidence interval statistics

of the cases changes in route delay cannot be attributed to multiple path routing but rather to changes between the routes. In 15% of the cases (20% in the 2006 data sets) the change in delay is mainly due to route changes as the overlap is zero or close to 0. Over 95% of the pairs that have academic source and destination ASes have an overlap of over 0.7. This is mainly the result of academic networks having small routes difference (induced by local load-balancing) and little usage of "spill-over" backup routes. Only 5% of the pairs that have both source and destination in the USA witnessed overlap of 0.

Finally, we evaluate how the route stability affects the overlap of delays. Fig. 5(a) shows that routes with high RouteISM (≥ 0.2) have higher percentage of non-overlap delay intervals. Namely, when the difference between the routes is larger, there are higher chances that their delay distribution will be different. Fig. 5(b) shows that, unlike RouteISM, the prevalence of the dominant route does not significantly affect the level of overlap.

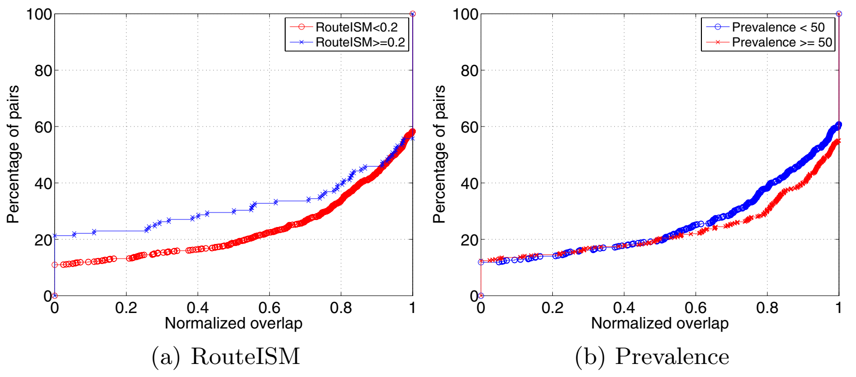


Fig. 5. Effect of route stability on normalized overlap

5 Conclusion

This work presents a measurement study of the e2e delay variance and its origins. Given a set of probed RTT delays, we find a confidence interval which better captures the delay of each observed route. We then compute the overlap of these intervals for uncovering the origin of these variations. Additionally, we develop techniques for quantifying route stability and measure its affect on the origin of delay variance. We find that for roughly 70% of the pairs and for over 95% of the academic pairs, the delay variations are mostly within the routes and not between different routes.

Acknowledgment. This work was partially funded by the OneLab II and the MOMENT consortia that are part of the EU FP7; and the Israeli Science Foundation, grant 1685/07.

References

1. Wang, F., Mao, Z.M., Wang, J., Gao, L., Bush, R.: A measurement study on the impact of routing events on end-to-end Internet path performance. ACM SIGCOMM CCR 36(4), 375–386 (2006)
2. Pucha, H., Zhang, Y., Mao, Z.M., Hu, Y.C.: Understanding network delay changes caused by routing events. SIGMETRICS 35, 73–84 (2007)
3. Augustin, B., Friedman, T., Teixeira, R.: Measuring load-balanced paths in the Internet. In: IMC (2007)
4. Pathak, A., Pucha, H., Zhang, Y., Mao, Z.M., Hu, Y.C.: A Measurement Study of Internet Delay Asymmetry. In: PAM (2008)
5. Shavitt, Y., Shir, E.: DIMES: Let the internet measure itself. ACM SIGCOMM CCR 35(5), 71–74 (2005)
6. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10(8), 707–710 (1966)
7. He, Y., Faloutsos, M., Krishnamurthy, S.: Quantifying routing asymmetry in the internet at the AS level. In: GLOBECOMM (2004)
8. Paxson, V.: End-to-End Routing Behavior in the Internet. IEEE/ACM Transactions on Networking, 601–615 (1996)
9. Bolle, R.M., Ratha, N.K., Pankanti, S.: An evaluation of error confidence interval estimation methods. In: International Conference on Pattern Recognition, vol. 3 (2004)
10. Shavitt, Y., Weinsberg, U.: Quantifying the importance of vantage point distribution in Internet topology measurements. In: Infocom (2009)
11. Dimitropoulos, X., Krioukov, D., Riley, G., Claffy, K.: Revealing the AS taxonomy: The machine learning approach. In: PAM (2006)
12. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: An overlay testbed for broad-coverage services. ACM SIGCOMM CCR 33(3) (July 2003)

Yes, We LEDBAT: Playing with the New BitTorrent Congestion Control Algorithm

Dario Rossi, Claudio Testa, and Silvio Valenti

Telecom ParisTech, Paris, France
first.last@enst.fr

Abstract. Since December 2008, the official BitTorrent client is using a new congestion-control protocol for data transfer, implemented at the application layer and built over UDP at the transport-layer: this new protocol undergoes the name of LEDBAT, for Low Extra Delay Background Transport.

In this paper, we study different flavors of the LEDBAT protocol, corresponding to different milestones in the BitTorrent software evolution, by means of an active testbed. Focusing on single flow scenario, we investigate emulated artificial network conditions, such as additional delay and capacity limitation. Then, in order to better grasp the potential impact of LEDBAT on the current Internet traffic, we consider a multiple flows scenario, and investigate the performance of a mixture of TCP and LEDBAT flows, so to better assess what "lower-than best effort" means in practice. Our results show that LEDBAT has already fulfilled some of its original design goals, though some issues still need to be addressed.

1 Introduction

Last December 2008, BitTorrent announced in the developer forum [1] that data transfer would move to UDP: shortly after this announcement, panic started spreading on popular websites [2], since the announcement directly led to the misbelief that BitTorrent plus UDP would equate with Internet meltdown. Yet, as already discussed at IETF [3] and later recognized on the Web [4], the BitTorrent development process embraces both ISP-friendliness (through AS-aware peer selection process) and TCP-friendliness (through a novel congestion control protocol for data transfer).

This work focuses precisely on this latter aspect of BitTorrent evolution: BitTorrent co-chairs an IETF Working Group on Low Extra Delay Background Transport (LEDBAT), which has very recently released its first draft document. To better understand the motivations behind LEDBAT, let us recall that the standard TCP congestion control mechanism needs losses to back off. Under a drop-tail FIFO queuing discipline, this means that TCP necessarily fills the buffer: as uplink devices of low-capacity home access networks have very large buffers, this may translate into poor performance of interactive applications (e.g., slow Web browsing and bad gaming/VoIP quality). LEDBAT attempts at avoiding this drawback, by implementing a distributed congestion control mechanism, tailored for the transport of non-interactive traffic with lower than Best Effort (i.e., TCP) priority. As stated in [5], among the main design goals of LEDBAT there are the ability (i) to minimize the extra delay it induces in the bottleneck, while (ii) saturating the available capacity at the same time. To fulfill these goals, LEDBAT has been designed as a windowed protocol (i) able to infer *earlier* than TCP the occurrence of congestion, by estimating the queuing delay variation on the end-to-end path, (ii) to which it reacts by continuously modulating the congestion window growth/shrink by a proportional-integral-derivative (PID) controller.

The aim of this work is twofold. On the one hand, we target at understanding the performance of LEDBAT in a number of simple single flow scenarios, considering multiple versions of the official client so to better clutch its evolution. On the other hand, by means of multiple flows scenarios, we aim at gathering a preliminary understanding of the implication that a widespread adoption of LEDBAT could have on the current Internet landscape. We tackle the above issues with an active-measurement black-box study of the official BitTorrent client. Since LEDBAT is openly described in a IETF draft, the performance of the protocol could be assessed by means of simulations as we did indeed in [6]. Yet we still find active testbed experimentation extremely useful for several reasons. First, the BitTorrent implementation of the LEDBAT protocol may differ from any draft-compliant implementation by some design choices or parameter setting, that may have a deep impact on the protocol performance. Second, the most widespread LEDBAT implementation on the Internet will be the official BitTorrent version, rather than a legacy implementation, which motivates a direct evaluation of this client. Third, from our point of view, the analysis of proprietary applications by independent observers has the benefit of sheding light on the protocol inner workings. Finally real-world dynamics introduced by network devices are often much more complex than the synthetic ones that a simulation environment, although accurate, can reproduce.

With such an approach, we conduct a preliminary yet insightful evaluation of the protocol performance. First, we are able to report on the entire evolution of the protocol implementation, from the first (immature) version to the last (nearly stable) one. We point out that LEDBAT is able to, at least partly, fulfill its original design goals: under both controlled testbed and Internet experiments, LEDBAT avoids an uncontrolled queuing (unlike TCP), and is, under a range of conditions, able to saturate the available capacity (or, in case capacity is not saturated, this could be done by a simple tweaking of LEDBAT parameters). At the same time, we identify some open points regarding the protocol efficiency: for instance, TCP traffic on the "unrelated" backward path is able to slow down LEDBAT transmission on the forward path, whose capacity may be then significantly underutilized. Finally, we stress that the precise meaning of "lower-than best effort" should be carefully specified, as the mutual influence of TCP and LEDBAT traffic may significantly differ depending on the TCP flavor and settings as well.

2 Methodology and Preliminary Insights

For the investigation of the LEDBAT, we adopt an active-measurements black-box experimental approach, consisting in the analysis of the traffic generated by the BitTorrent client on different network scenarios. We run several versions of the new BitTorrent client on PCs equipped with dual-core processors featuring (i) unless otherwise stated, native installations of Windows XP or (ii) BitTorrent clients running on Linux using the `wine` Windows emulator. PCs are either (i) connected to the Internet through ISPs offering ADSL access, or (ii) in a local LAN testbed via Ethernet cards. In the first case we leave the default modem settings unchanged, while in the second one we disable the interrupt coalescing feature and avoid the usage of jumbo frames. Moreover in the LAN testbed, the traffic is routed through a middlebox running a 2.6.28 Linux kernel, which acts also as network emulator by means of `netem`, in order to enforce artificial network conditions.

As formerly stated, in our experiments we consider both single flow and multiple flows scenarios. Single flow experiments are useful to understand the protocol performance under a range of different network conditions, while multiple flows experiments are needed to quantify the level of inter-protocol priority (e.g., with respect to TCP flows) and intra-protocol fairness (e.g., with respect to other LEDBAT flows) achieved by the distributed control algorithm. Under the classic BitTorrent terminology, every LEDBAT sender-receiver pair is a seeder-leecher pair, so that data transfer happens in a single direction. In case of multiple-flows experiments, every pair of actors belongs to a different torrent, so that no data exchange happens between different leechers.

We start by providing some insights on the BitTorrent evolution with the help of Fig. 1. Every picture refers to a different experiment, of which we report the first minute, corresponding to a different BitTorrent flavor. The seeder connects to the middlebox with a 100 Mbps Ethernet link, while between the middlebox and the leecher there is a 10 Mbps Ethernet bottleneck link. No other traffic is present on the bottleneck, and the one-way delay on the forward path is forced to 50 ms, to loosely emulate a scenario where two faraway peers with high speed Internet access (e.g., ADSL2+, FTTH or Ethernet) are connected together.

Pictures are arranged so that the macroscopic timescale of BitTorrent evolution also grows from left to right: Fig. 1-(a) shows, as a reference, the old open-source TCP-based client, while Fig. 1-(b) refers to the first closed-source version α_1, released December 2008. Then, Fig. 1-(c) depicts the α_2 version, released roughly at the same time of the first IETF draft [5] in March 2009. Finally, Fig. 1-(d) refers to the β_1 version, released after the draft was accepted as an official IETF WG item in August 2009.

The comparison of different versions of the protocol yields several interesting observations. First, notice that all versions analyzed correspond to important milestones in the development process of the protocol: thus, they provide a valuable perspective which highlights the flaws as well as the improvements of the subsequent steps of LEDBAT evolution. In particular, the α_1 version (which precedes the draft specification and motivates a black-box approach) was particularly instable and soon superseded. Moreover, from this study it emerges that the LEDBAT implementation is *constantly* evolving: as such, we believe that picking a single version, such as the most recent one, would limit the scope of our study.

For each flavor represented in Fig. 1, pictures depict the packet size on the y-axis, measured at the sender side, with time of the experiment running on the x-axis. As it can be seen, the application-layer segmentation policy is remarkably variable across different LEDBAT flavors. In contrast with TCP, which always transmits segments of maximum size, LEDBAT instead uses variable packet sizes. For instance, the α_1 implementation of Fig. 1-(b) mostly used small segments of about 350 bytes, transmitted at very high rate. Although this allows a finer tuning of the congestion window size, (e.g., likely to be more reactive to network condition), it definitively results in an

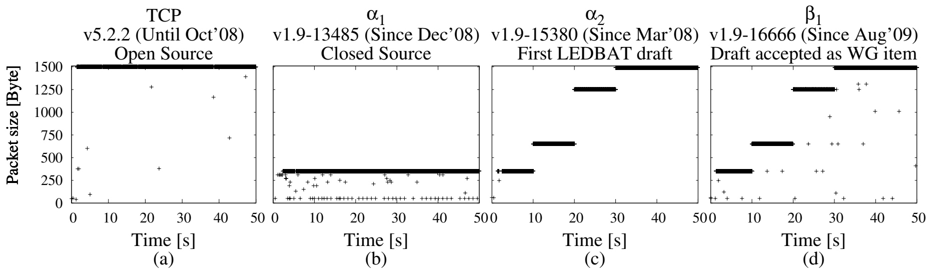


Fig. 1. The last few months of BitTorrent client evolution: Temporal plot of packet-level traces for different BitTorrent flavors, reporting packet size during the first minute of the transfer

unnecessary overhead. This segmentation policy is a bad choice for large transfers, and was indeed soon dropped in favor of larger segment sizes. As can be gathered from Fig. 1-(c) and Fig. 1-(d), newer BitTorrent flavors start by segmenting data in small-size segments, and then gradually increase the segment size over time, rarely changing it once the full-payload segment size is reached. In case of α_2 flavor, we observe subsequent phases, about 10-seconds long, where only a single segment size is used: it takes about 40 seconds to the application-layer segmentation policy to settle to full-payload segment size. The β_1 flavor behaves similarly, although a wider range of segment sizes is employed during the whole experiment, probably to obtain a finer byte-wise control of the congestion window.

The corresponding time evolution of the achieved throughput, measured over 1 s time-windows is depicted in Fig. 2-(a), using a longer timeframe of about 4 minutes. We merely superpose the curves for the sake of comparison, but experiments have been independently performed. It can be seen that, shortly after achieving a sustained throughput of about 9 Mbps during about 50 seconds, the sending rate of the α_1 version suddenly drops, and about 2 minutes are necessary to recover from this starvation (this unstable behavior was observed under a wide range of conditions). In contrast, α_2 and β_1 achieve a lower but steady throughput, slightly above 4 and 7 Mbps respectively.

As a reference, we also report the throughput of a BitTorrent client using TCP running on the native Windows and Linux networking stacks under their default settings. The networking stack implementation and configuration dramatically impacts the protocol performance also in the TCP case. As reported in [7], in Windows XP, for transmission rates between 10-100 Mbps the default receive window is set to 17520 Bytes, whereas the default value of the Linux receive window (set in `net.ipv4.tcp_mem`) is about 6 times larger. Notice that in the Windows XP case, due to the 50 ms delay, the default value of the maximum window is not large enough to allow full saturation of the bottleneck pipe. This is an important, though not novel, observation on which we will come back later on Sec. 3.2.

3 Experimental Results

In this section, we start with simple single flow scenarios so to refine the performance pictures of the different flavors by testing the impact of varying network conditions.

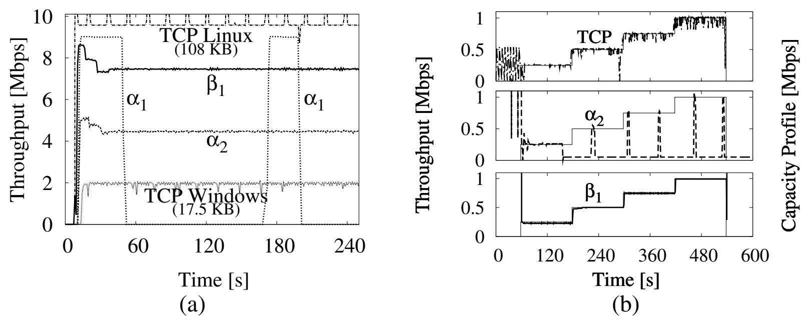


Fig. 2. Throughput for different flavors (a) without and (b) with bottleneck capacity limitations

Among the several experiments conducted, we report the most relevant for our performance evaluation. In more detail, we consider (i) bottleneck capacity limitations, (ii) one-way delay impairment on either the forward or the backward path and (iii) different access technologies. We finally consider a scenario in which (iv) a single TCP flow interferes with LEDBAT on either the forward or the backward path, and (iv) multiple flows share the same bottleneck link, varying the ratio of LEDBAT and TCP flows so to better assess the protocols mutual influence.

3.1 Single Flow: Bottleneck Capacity, Delay and Access Impact

Let us start by testing how BitTorrent copes with changing bottleneck capacity. We use a setup similar to the former experiment, but in this case the capacity of the link between the middlebox and the leecher is limited by means of the Hierarchical Token Bucket (HTB), available in `netem`. In more detail, we start at t=60 s to let LEDBAT throughput settle to a steady state, and then we turn on the HTB shaper. We initially tune it to 250 Kbps, increasing then the available capacity in steps of 250 Kbps every 2 minutes, as shown by the solid line capacity profile in Fig. 2-(b). A decreasing capacity profile yields to similar results and is thus not shown in the figure.

Time evolution of the throughput is reported for the new α_2, β_1 flavors as well as for the old TCP client. Flavor α_2 proves to be unable to quickly adapt to the changing link rate: it periodically enters a probing (or slow-start) phase, where it likely tries to infer network conditions by varying the segment size and sending rate. However, this phase is apparently unsuccessful and α_2 throughput starves (we did not observe such a starvation phenomenon for bottleneck larger than 1000 Kbps). This bug has been fixed by later releases: β_1 matches the available bandwidth, and moreover LEDBAT shows a much smoother curve than TCP. In this case, we may say that one of the LEDBAT design goals, namely, to efficiently exploit the available capacity, seems to be perfectly achieved.

Then, consider that the LEDBAT congestion control is based on a linear adaptation (i.e., growth/shrink) of the sender window to variations in the queuing delay on the forward data path (i.e., as inferred by the decrease/increase of the one-way delay, with respect to the minimum measured one as reference): it is thus critical to assess its reaction

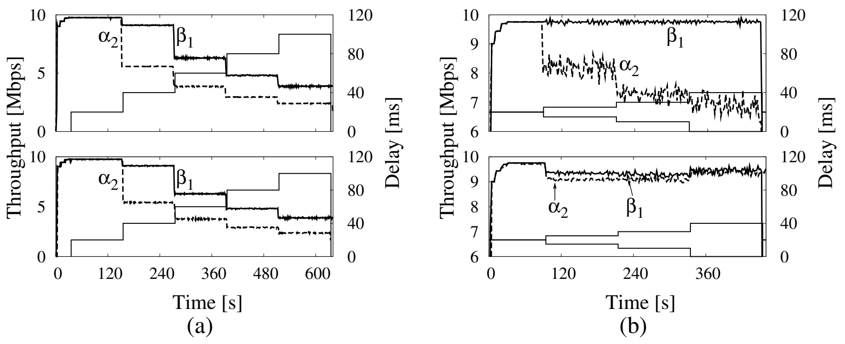


Fig. 3. Throughput evolution for different delay settings on the forward (top) and backward (bottom) path: (a) average delay increases over time, delay is equal for all packets (b) average delay is constant over time, delay variance increases over time

to the measured one-way (OWD) delay. However, the sender response to queuing delay variations is nevertheless based on a closed-loop reaction with the receiver: therefore, we argue that the time instants at which the sender window growth/shrink decisions will be taken are also affected by the two-way delay, or Round Trip Time (RTT).

Thus, we setup and experiment in which we add an incremental OWD on either the forward (data) or backward (acknowledgement) paths. As before, after LEDBAT settles we increase the additional delay in steps of 20 ms every 2 minutes, for an RTT spanning on the 20–100 ms range as shown by the stepwise profile in Fig. 3-(a). The amount of OWD delay is added either to the forward path (top) or backward (bottom) path: in the former case, the delay incrementally adds to the OWD estimation performed by the sender so that it may directly affect the congestion control loop, while in the latter case it only delays the acknowledgement and may only indirectly affect the control loop.

As it can be seen from the comparison of the top and bottom plots of Fig. 3-(a), the overall effect on performance is the same: BitTorrent throughput decreases for increasing RTT, which is due to an upper bound of the receiver window (analogously to what seen before for TCP). With some back-of-the-envelope calculation based on the experimental results shown in Fig. 3-(a), one can gather that the receiver window limit has been increased from 20 full-payload segments of α_2 to 30 full-payload segment of β_1. While the picture shows that this limit may not be enough to fully utilize the link capacity (e.g., β_1 achieves about 4 Mbps throughput on a 10 Mbps link with RTT$=$100 ms), in practice it is not a severe constraint, as the capacity will likely be shared across several flows established with multiple peers of a BitTorrent swarm (or the receiver window limit could be increased).

In Fig. 3-(b) we instead investigate the effects of a variable OWD delay, that changes for each packet uniformly at random, with average OWD equal to 20 ms. In this case we keep the average constant but increase the delay *variance* every 2 minutes, so that the profile reports the minimum and maximum delays of the uniform distribution. The variable delay also implies that packet order is not guaranteed, because packets encountering a larger delay will be received later and thus out-of-order. Again, delay variance is enforced on either the forward (top) or backward (bottom) path. As it can be

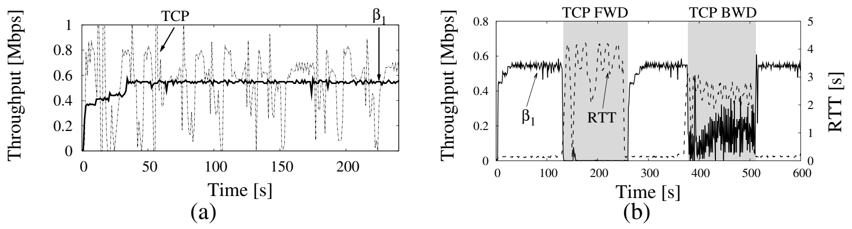


Fig. 4. Real Internet experiments: (a) different flavors and (b) interfering traffic

expected, LEDBAT is rather robust to a variable jitter on the backward path, where we observe only a minimal throughput reduction. Conversely, variance in the forward path has a much more pronounced performance impact: interestingly, α_2 throughput significantly drops, whereas β_1 performance is practically unchanged. This probably hints to the use of a more sophisticated noise filtering algorithm (e.g., that discards delay samples of out-of-order packets), although a more careful analysis is needed to support this assertion.

We finally perform an experiment using PCs connected through ADSL modems to the wild Internet. Thus, in this case we no longer have complete control over the network environment, but we still can assume that no congestion happens in the network and that the access link constitutes the capacity bottleneck. It can be seen from Fig. 4-(a) that in a realistic scenario, when the end-hosts only run LEDBAT, β_1 achieves a smooth throughput whose absolute value closely matches the nominal ADSL uplink capacity (640 Kbps). In contrast, TCP throughput is more fluctuating due to self-induced congestion, which causes fairly large queues before eventual losses occur. This confirms that the goal of avoiding self-induced congestion at the access is also met.

3.2 Multiple Flows

We now explore scenarios with several concurrent flows, starting with the simple one where a single LEDBAT flow interacts with a single TCP flow. Considering two PCs connected through ADSL modems to the wild Internet, Fig. 4-(b) reports an experiment where, during a single LEDBAT transfer, we alternate periods in which PCs generate no traffic other than LEDBAT, to periods (i.e., the gray ones) in which we superpose TCP traffic on either the forward or backward path.

The plot reports the time evolution of the LEDBAT throughput as well as the RTT delay measured by ICMP (as a rough estimation of the queue size seen by LEDBAT). During the silence periods (0–120 s and 240–360 s), as bottleneck is placed at the edge of the network, LEDBAT is able to efficiently exploit the link rate. As soon as a backlogged TCP transfer is started on the forward path (120–240 s), LEDBAT congestion control correctly puts the traffic in low priority. Notice that in this case, ICMP reports that a fairly large queue of TCP data packets builds up in the ADSL line (roughly 4 seconds, corresponding to about 300 KB of buffer space for the nominal ADSL rate). Conversely, whenever the backlogged TCP transfer is started on the backward path (360–480 s), LEDBAT transfer on the forward direction should only be minimally

Table 1. Efficiency and Fairness between multiple TCP and LEDBAT flows

		TCP$_W$, LEDBAT β_1							TCP$_L$, LEDBAT β_1						
TCP	LEDBAT	$\%_1$	$\%_2$	$\%_3$	$\%_4$	η	Fairness	RTX%	$\%_1$	$\%_2$	$\%_3$	$\%_4$	η	Fairness	RTX%
4	0	0.25	0.25	0.25	0.25	0.67	1.00	5e-4	0.25	0.25	0.25	0.25	0.98	1.00	0.06
3	1	0.14	0.14	0.14	0.57	0.94	0.64	-	0.35	0.32	0.32	0.00	0.98	0.75	0.14
2	2	0.10	0.10	0.40	0.40	0.93	0.74	-	0.43	0.51	0.03	0.03	0.98	0.56	4e-3
1	3	0.08	0.31	0.31	0.31	0.92	0.87	-	0.87	0.04	0.04	0.05	0.98	0.33	-
0	4	0.25	0.27	0.24	0.24	0.96	1.00	-	0.25	0.27	0.24	0.24	0.96	1.00	-

affected by the amount of acknowledgement TCP traffic flowing in the forward direction. However, as it can be seen from Fig. 4-(b), the LEDBAT throughput drastically drops, further exhibiting very wide fluctuations (notice also that the ADSL modem buffer space of the receiver appears to be smaller, as the RTT is shorter). Notice that in this case, LEDBAT forward data path shares the link capacity only with TCP acknowledgements, which account for a very low, but likely very bursty, throughput: this may led LEDBAT into a messy queuing delay estimate, and as a result, the uplink capacity of the device is heavily underutilized (about 74% of wasted resources).

We finally perform experiments to analyze the interaction of several flows. In this case, we setup several torrents, one for every different LEDBAT seeder-leecher pair, so that no data exchange happens between leechers of different pairs. Thus, flows are independent at the application layer, though their are dependent at the transport layer, as they share the same physical 10 Mbps RTT=50 ms bottleneck.

We consider a fixed number of F=4 flows, and vary the number of TCP and LEDBAT-β_1 connections to explore their mutual influence. All flows start at time $t = 0$, experiments last 10 minutes and results refer to the last 9 minutes of the experiment. We generate TCP traffic using Linux (so that we can reliably gather retransmission statistics using `netstat`), setting the congestion control flavor to NewReno. We perform two set of experiments, using either the Windows or Linux defaults values for the maximum receiver windows as early stressed in Fig. 2-(a): in our setup, the Windows-like TCP settings (TCP$_W$) are thus less aggressive than Linux ones (TCP$_L$).

For each experiment, we evaluate user-centric performance by means of the breakdown of the resources acquired by each flow, while we express network-centric performance in terms of the link utilization η. To further quantify the protocol mutual influence, we use the Jain's fairness index of the flows throughput and evaluate the percentage of TCP retransmissions (RTX). Results are reported in Tab. 1, with Windows and Linux settings on the left and right respectively. Comparing the two table portions, we argue that the exact meaning of "low-priority" may be fuzzy in the real-world. Indeed, while LEDBAT-β_1 is lower priority than an "aggressive" TCP, it may be competing more fairly against a more gentle set of parameters, thus being at least as high priority as TCP. In fact while LEDBAT is practically starved by TCP$_L$, LEDBAT is able to achieve a slightly higher priority than TCP$_W$. Although we recognize that results may change using more realistic and heterogeneous network scenarios, or using the real Windows stack instead of simply emulating its settings, we believe that an important point remains open: i.e., the precise meaning of "lower than best effort", as the mutual influence of TCP and LEDBAT traffic may significantly differ depending on the TCP flavor as well.

4 Related Work

Two bodies of work are related to this study. On the one hand, BitTorrent has been studied by means of theoretical analysis [8], simulation [9,10,6] or measurements [11]. On the other hand, there is a large literature on Internet congestion control that use either on fields measurement [12,13,14], or simulation and modeling [15,16,17,18,19,20]. Due to BitTorrent very recent evolution, with the exception [6], where we study LEDBAT by means of simulation, previous work on BitTorrent [8,9,10,11] focused on complementary aspects to those analyzed in this work. In [8] a fluid model is used to determine the average download time of a single file. Simulation has instead been used in [9] to propose incentive mechanism to avoid free-riding and in [10] to assess the performance of a locality-aware peer selection strategy. Finally, measurements study [11] analyzes the log of a BitTorrent tracker, examining flash-crowd effect, popularity and download speed of a single file. Congestion control work closer to our adopts a black-box experimental measurements approach to unveil proprietary algorithms of, e.g., Skype [12,13] or P2P-TV applications [14]. More precisely, [12,14] analyzes system reaction to emulated network conditions, whereas [13] investigates the bottleneck share of multiple flows. Finally, relevant work has been devoted to the design of lower-than best effort protocols similar to LEDBAT, as for instance [17,18,19,20].

5 Conclusions

This paper presented an experimental evaluation of LEDBAT, the novel BitTorrent congestion control protocol. Single-flow experiments in a controlled environment show some of the fallacies of earlier LEDBAT flavors (e.g., instability, small packets overkill, starvation at low throughput, tuning of maximum receiver windows, wrong estimate of one-way delay in case of packet reordering, etc.), that have been addressed by the latest release. Experiments in a real Internet environment, instead, show that, although LEDBAT seems a promising protocol (e.g., achieving a much smoother throughput and keeping thus the delay on the link low), some issues still need to be worked out (e.g., performance in case of reverse path traffic). Finally, multiple-flows experiments show that “low-priority” meaning significantly varies depending on the TCP settings as well.

This work constitutes a first step toward the analysis LEDBAT performance. More effort is indeed needed to build a full relief picture of the LEDBAT impact on other interactive applications (e.g., VoIP, gaming), explicitly taking into account the QoE resulting from their interaction. Also, the methodology could be refined by, e.g., instrumenting the Linux kernel to measure the queue size, or by inferring the OWD measured by LEDBAT by sniffing traffic at both the sender and receiver, etc. Finally, the boundaries of the investigation could be widened by taking into account the effects of LEDBAT adoption on the BitTorrent P2P system itself, as for instance LEDBAT interaction with throughput based peer-selection mechanism, or its impact on files download time.

Acknowledgement

This work has been funded by the Celtic project TRANS.

References

1. Morris, S.: μTorrent release 1.9 alpha 13485 (December 2008), http://forum.utorrent.com/viewtopic.php?pid=379206#p379206
2. Bennett, R.: The next Internet meltdown (December 2008), http://www.theregister.co.uk/2008/12/01/richard_bennett_utorrent_udp
3. Shalunov, S., Klinker, E.: Users want P2P, we make it work. In: IETF P2P Infrastructure Workshop (May 2008)
4. BitTorrent Calls UDP Report "Utter Nonsense" (December 2008), http://tech.slashdot.org/article.pl?sid=08/12/01/2331257
5. Shalunov, S.: Low extra delay background transport (LEDBAT). IETF Draft (March 2009)
6. Rossi, D., Testa, C., Valenti, S., Veglia, P., Muscariello, L.: News from the internet congestion control world. Technical Report (August 2009)
7. MS Windows Developer Center: Tcp receive window size and window scaling, http://msdn.microsoft.com/en-us/library/ms819736.aspx
8. Qiu, D., Srikant, R.: Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In: ACM SIGCOMM 2004, Portland, Oregon, USA (August 2004)
9. Bharambe, A.R., Herley, C., Padmanabhan, V.N.: Analyzing and Improving a BitTorrent Networks Performance Mechanisms. In: IEEE INFOCOM 2006, Barcelona, Spain (April 2006)
10. Bindal, R., Cao, P., Chan, W., Medved, J., Suwala, G., Bates, T., Zhang, A.: Improving Traffic Locality in BitTorrent via Biased Neighbor Selection. In: IEEE ICDCS 2006, Lisboa, Portugal (July 2006)
11. Izal, M., Urvoy-Keller, G., Biersack, E.W., Felber, P., Al Hamra, A., Garcés-Erice, L.: Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In: Barakat, C., Pratt, I. (eds.) PAM 2004. LNCS, vol. 3015, pp. 1–11. Springer, Heidelberg (2004)
12. Bonfiglio, D., Mellia, M., Meo, M., Rossi, D.: Detailed Analysis of Skype Traffic. IEEE Transaction on Multimedia 11(1) (January 2009)
13. De Cicco, L., Mascolo, S., Palmisano, V.: Skype video responsiveness to bandwidth variations. In: ACM NOSSDAV 2008, Braunschweig, Germany (May 2008)
14. Alessandria, E., Gallo, M., Leonardi, E., Mellia, M., Meo, M.: P2P-TV Systems under Adverse Network Conditions: A Measurement Study. In: IEEE INFOCOM 2009 (April 2009)
15. Padhye, J., Firoiu, V., Towsley, D., Kurose, J.: Modeling TCP throughput: a simple model and its empirical validation. ACM SIGCOMM Comp. Comm. Rev. 24(4) (October 1998)
16. Brakmo, L., O'Malley, S., Peterson, L.: TCP Vegas: New techniques for congestion detection and avoidance. In: ACM SIGCOMM 1994, London, UK (August 1994)
17. Venkataramani, A., Kokku, R., Dahlin, M.: TCP Nice: a mechanism for background transfers. In: USENIX OSDI 2002, Boston, MA, US (December 2002)
18. Kuzmanovic, A., Knightly, E.: TCP-LP: low-priority service via end-point congestion control. IEEE/ACM Transaction on Networking 14(4) (August 2006)
19. Liu, S., Vojnovic, M., Gunawardena, D.: Competitive and Considerate Congestion Control for Bulk Data Transfers. In: IWQoS 2007, Evaston, IL, US (June 2007)
20. Key, P., Massoulié, L., Wang, B.: Emulating low-priority transport at the application layer: a background transfer service. In: ACM SIGMETRICS 2004, New York, NY, USA (January 2004)

Measuring and Evaluating TCP Splitting for Cloud Services

Abhinav Pathak[1], Y. Angela Wang[2], Cheng Huang[3], Albert Greenberg[3], Y. Charlie Hu[1], Randy Kern[3], Jin Li[3], and Keith W. Ross[2]

[1] Purdue University
[2] Polytechnic Institute of NYU, New York
[3] Microsoft Corporation, Redmond

Abstract. In this paper, we examine the benefits of split-TCP proxies, deployed in an operational world-wide network, for accelerating cloud services. We consider a fraction of a network consisting of a large number of *satellite* datacenters, which host split-TCP proxies, and a smaller number of *mega* datacenters, which ultimately perform computation or provide storage. Using web search as an exemplary case study, our detailed measurements reveal that a vanilla TCP splitting solution deployed at the satellite DCs reduces the 95^{th} percentile of latency by as much as 43% when compared to serving queries directly from the mega DCs. Through careful dissection of the measurement results, we characterize how individual components, including proxy stacks, network protocols, packet losses and network load, can impact the latency. Finally, we shed light on further optimizations that can fully realize the potential of the TCP splitting solution.

1 Introduction

Cloud Services are delivered with large pools of computational or storage resources that are concentrated in *mega* datacenters, being built only in a hand full of remote locations world-wide. The continued growth of Cloud Services, however, critically depends on providing a level of responsiveness comparable with what can be obtained directly from dedicated on-site infrastructures. A key challenge here is to make remote infrastructures appear to end-users as if they were nearby.

Split-TCP proxies [1][2] can be very effective in improving the responsiveness of Cloud Services. In particular, Cloud Service providers can deploy *satellite* datacenters – within or outside of their own networks – close to the end-users. These satellite DCs host split-TCP proxies, which maintain persistent connections over long-haul links to the mega DCs that ultimately perform computation or provide storage. Through bypassing TCP *slow start* and avoiding a number of round trips on the long-haul links, this architecture reduces response time very effectively. Many Cloud Services, whose computation or storage cannot be easily or cost-effectively geo-distributed and thus have to remain in the mega DCs, can benefit from this architecture, including Internet search, collaborative online editing, concurrent social gaming, cloud-based storage, and so on. Although there are existing deployments of split-TCP proxies in commercial systems (e.g., [3][4]), there are very few published studies on performance gains based on real-world Internet measurements. Furthermore, to our knowledge, there is no thorough study that dissects each component of a TCP splitting solution, with the aim at identifying further optimizations and fully realizing its potential.

In this paper, we deploy an experimental TCP splitting solution on a fraction of a network of satellite DCs hosted by Microsoft's global distribution and cloud service network. We conduct detailed measurements to quantify the gain experienced by real-world end-users. Using web search as an exemplary case study, we show that, compared to directly sending queries to the mega DCs, a vanilla TCP splitting solution can reduce the 95^{th} percentile latency by 43%[1]. Through careful dissection of the measurement results, we characterize how individual components – including proxy stacks, network protocols, packet losses and network load – can impact the latency. Finally, we shed light on further optimizations that can fully realize the potential of the TCP splitting solution.

2 A Web Search Case Study

Web search is one of the most important and popular cloud services. Clearly, the relevance of search result is critical to the success of the service. In addition, the speed of search response is essential. Delay of an extra half a second can affect user satisfaction and cause a significant drop in traffic [7].

2.1 Search Response: Empirical Results

The amount of data in a search response is typically very small. To measure its size and time, we identified about 200,000 common search terms from anonymized reports from real-world users using 'MSN Toolbar'. For each search term, we issued a query to obtain a uncompressed HTML result page, against a popular Internet search engine (the search engine name is anonymized). We issued all the queries from a single client located on a university campus network. We measured the response size and the response time, that is, the time elapsed from TCP SYN is sent until the last packet of the HTML result page is received. We also extracted the time – taken within the search datacenter to complete the actual search and construct the result – as reported on the resultant HTML page.

Figure 1 plots the CDF response size of 200K search queries. We see that the size of a typical uncompressed search response is 20-40KBytes, sometimes exceeding 50KBytes. With a TCP Maximum Segment Size (MSS) of about 1500 bytes, this corresponds to 15 to 27 TCP data packets utilizing 4 TCP windows of data transfer (as suggested by RFC 3390 [8]). Figure 2 plots the CDF of response time of 200K search queries as observed by the client and the CDF of search time within the datacenter (denoted "Search Time") as reported on the result page. From the figure, we see that a typical response takes between 0.6 and 1.0 second. The RTT between the client and the datacenter during the measurement period was around 100 milliseconds. We remark that our measurement

[1] In this paper, we focus on optimizing split-TCP solutions under given satellite DCs. We cover an orthogonal and equally important issue – the choice of satellite DC locations – in separate studies [5][6].

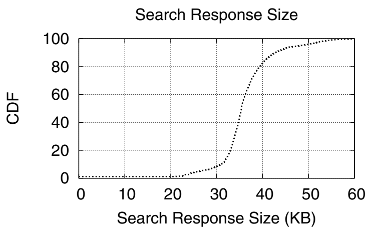


Fig. 1. CDF of response sizes of 200K search queries from a popular search engine

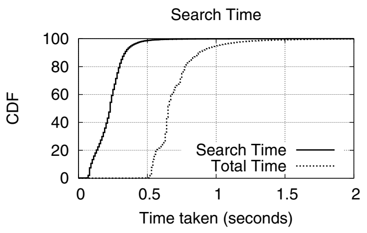


Fig. 2. CDF of response time of 200K search queries by popular search engine for search reply

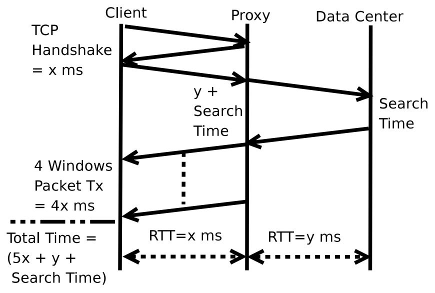


Fig. 3. TCP packet exchange diagram between an HTTP client and a search server with a proxy between them

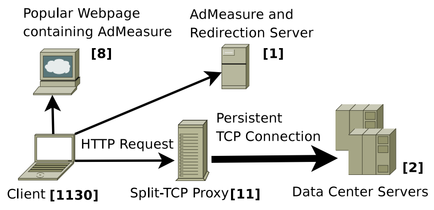

Popular Webpage containing AdMeasure
[8]
AdMeasure and Redirection Server
[1]
HTTP Request
Persistent TCP Connection
[2]
Client [1130]
Split-TCP Proxy[11]
Data Center Servers

Fig. 4. TCP splitting platform - The [number] beside each component represents the number(s) of that component in our measurement system

client was connected from a well-provisioned network. As we will see later, the response time can be a lot worse for clients in the wild. From the figure, we see that the search time within the datacenter ranges almost uniformly between 50 and 400 msec. We also note that 5.24% of the search queries took one second or more to finish. The 95th percentile of the response time was roughly one second.

2.2 Simple Model for Response Latency

The total time for a query and response is composed of the time to send the query, the time to construct the result page and the time to transfer the page (Other factors that influence the user perceived time includes page rendering speed, speed of scripts execution on browser, etc. While these are important factors, we only study the network part of latency in this paper). The client first establishes a TCP connection with the datacenter server through a three-way TCP handshake. The client then sends an HTTP request to the server, which includes the search query. The sever performs the search, incurring $search\_time$, and then ships the result page to the client. We assume that four TCP windows (as discussed earlier) are required to transfer the result page to the client when there is no packet loss. The total time taken in this case is $(5RTT+search\_time)$.

Now, consider the potential improvement of TCP splitting, where a proxy is inserted, close to the client, between the client and the datacenter, as shown in Figure 3. In such a design, the proxy maintains a persistent TCP connection with the data center server, where the TCP window size is large, compared to the amount of data in individual search result pages. A client establishes a TCP connection and sends a search query to the proxy. The proxy forwards the query to the datacenter server over the persistent connection with a large TCP window (We ignore the CPU processing overhead incurred at proxies throughout this paper as this overhead was estimated to be as low as 3.2ms in userspace split TCP implementation and 0.1ms in kernel level split TCP implementation [2]). The datacenter server processes the search query, incurring $search\_time$, and transmits the resulting page back to the proxy within one round trip (given the large TCP window). The proxy then transfers the data to the client, which goes through a TCP slow-start phase and takes several round trips.

The total time taken in this case is $(5x + y + search\_time)$, where x is the RTT between the client and the proxy and y is the RTT between the proxy and the datacenter. Comparing this with the no-proxy case, we see that TCP splitting can potentially reduce the response time by $(5RTT - (5x + y))$. When $x + y \approx RTT$ (i.e., the proxy detour overhead is negligible), this reduction becomes $4(RTT - x)$; when further $x << RTT$, i.e., the client-proxy distance is small when compared to the proxy-datacenter distance, this reduction becomes approximately $4RTT$, which can be quite substantial for interactive applications.

3 Experimental TCP Splitting System

To understand the gain of TCP splitting on search queries in the wild, as well as to characterize individual components in a TCP splitting design, we've implemented an experimental TCP splitting system, deployed it in a global distribution and cloud service network, and characterized the system with search traffic from real-world users. Our experimental system has two major components: a client measurement platform and a TCP splitting platform.

3.1 Measurement System

Client Measurement Platform: Our goal is to measure query latencies for real clients in the wild - with and without split-TCP proxies. To this end, we exploit AdMeasure, a measurement platform we recently developed [6]. In a nutshell, AdMeasure deploys a Flash object (implemented in 300 LOC in ActionScript) on multiple popular web pages. When a client visits any one of these web pages (as shown in Figure 4), the AdMeasure Flash object is loaded into the client at the end of the web page (so as not to affect user-perceived page load time). The Flash object retrieves a workload list from a central AdMeasure server, performs pre-configured Internet measurements such as issuing search queries to the IPs contained in the workload list, and submits results back to the AdMeasure server.

In our TCP splitting experiments, the AdMeasure server uses the client's geographic location (from its IP address) and instructs the client to issue search queries to the closest proxy (This step incurs overhead for each query. It is a simplified implementation and should be replaced by a DNS server in a production system, where the DNS resolution overhead is amortized over many queries and thus minimal). The proxy with minimum geographic distance is a reasonable approximation to the proxy with minimum RTT, as shown in [9]. For simplicity, we use a fixed search query term "Barack Obama" with/without proxies. We verify that the repetition of the same query term experiences similar "search time" in the mega datacenters - i.e., there is no caching of search results at the datacenter when same queries are issued repeatedly. We deploy the AdMeasure Flash object on multiple popular partner websites, including the front page for Microsoft Research, the front page for Polytechnic Inst. of NYU, the front pages of three small online gaming websites, as well as a few personal homepages.

TCP Splitting Platform: Our TCP splitting platform consists of two parts: split-TCP proxies and mega datacenters. We deploy split-TCP proxy (about 2K LOC in C++) in a fraction of the satellite DCs of Microsoft's global distribution and cloud service network (11 locations worldwide - 6 in US, 3 in Europe and 2 in Asia). We choose 2 Live Search mega DCs (both in US). Each proxy forwards search queries to the closer (in terms of RTT) Live Search datacenter. The proxy does not cache the results of any search query. The proxy relays the queries on behalf of clients over a persistent HTTP connection to the datacenter. It stores statistics like response time, content length, query id, etc. In addition, packet traces in form of tcpdumps are recorded.

Through AdMeasure, each client is instructed to issue 6 back-to-back queries to the closest of the 11 proxies, which forwards to the closer of the two datacenters; and each query starts a new TCP connection to the proxy. We ignore the first two queries, which are meant to warm up the TCP transmission window between the datacenter and the proxy. This is to emulate production environments, where many queries and responses are multiplexed over the same datacenter-proxy connection. To understand the degree to which TCP splitting helps, as a baseline, each client also issues six queries directly to the datacenter.

3.2 Measurement Results

Through AdMeasure, we collected one week's worth of data consisting of 5,584 search queries through proxies from 1,130 unique clients out of which 952 were located in North America (covering 193 cities). The bias in clients' location originates from the fact that the websites, where AdMeasure was deployed, were popular mostly in North America. Using one week's worth of data, we now report our experimental results in this subsection. Since the current deployment of AdMeasure attracts significantly more users from North America than other continents, we report only clients originating from North America.

Latency Model Validation: We first validate whether the simple model described in Figure 3 indeed holds true with the real clients. We separate out the traces with packet loss in either proxy to client or datacenter to proxy communication (traces with loss will be re-visited later). We identify packet loss from the proxy-side tcpdump outputs. For simplicity, we assume that retransmission implies data packet loss. ACK loss is not easy to identify, but turns out *not* to be rare. Here, we apply a simple heuristic to infer ACK loss – the sequence number gap between any two consecutive ACKs is calculated; if the

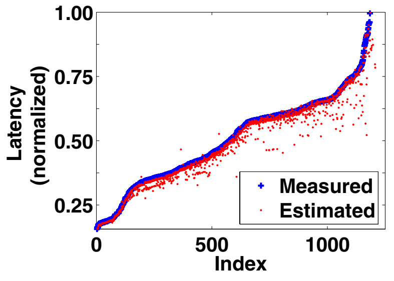


Fig. 5. Latency Model Validation

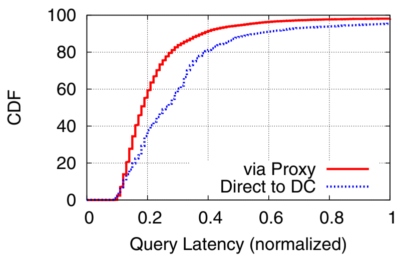


Fig. 6. Gain of TCP Splitting

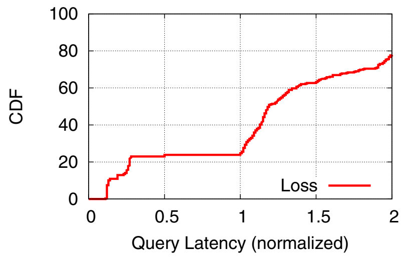


Fig. 7. Impact of Packet Loss

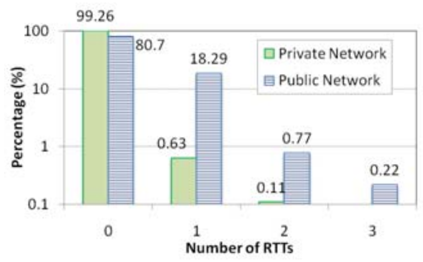


Fig. 8. Time (*RTT) between first and last response packet at proxy from datacenter

gap is bigger than two MSS (taking into account delayed acknowledgment [10]), we assume there is an ACK loss (this simple heuristic might over-estimate, but is nevertheless conservative for weeding out connections with loss).

Considering all traces without data packet/ACK loss, we now calculate an estimated latency as: 6 times RTT[2] between client and proxy ($= 6 * x$ - notation follows from Figure 3) + 1 RTT between proxy and datacenter ($= y$) + datacenter search time ($= search\_time$). Here, ($y + search\_time$) is obtained directly from tcpdump, as the time from when the proxy forwards the query to the datacenter until it gets the first response packet. We also obtain the true total latency from tcpdump, as the time from when the proxy receives the SYN until the final ACK from the client.

Figure 5 (best viewed in color) plots the estimated and true (normalized) latency of minimum of the six search queries per client visit (a few clients visited multiple times) sorted by measured latency. Using a linear scale, we normalized (and hence anonymized) all the real latencies with a constant large latency value throughout the paper. It is clear that, without packet/ACK loss, the simple model nearly approximates the true latency. Note that towards right in the figure, where the total latency is large, the RTTs between the clients and the proxies are also large. Larger RTTs show more variations, which tend to have a larger impact on inaccuracy.

[2] As for "Barack Obama" Query, response size is 55KB which is 5 TCP windows of data, implying, 1 RTT for TCP Handshake + 5 window packet transfer (5*RTT).

Table 1. TCP window comparison, for different OS's

	Linux	Win2003	Win2008
Window Size in Bytes			
1st	2,920	2,520	2,920
2nd	4,380	3,780	5,840
3rd	5,840	4,980	11,680
4th	8,760	7,560	23,360
5th	11,680	11,340	
6th	16,060	16,380	
# of Windows for 50KB+ data			
	7	7	5

Impact of Proxy OS: In the course of our experiment, we found that the operating systems on the proxies have a large impact on the latency, as they exhibit very different TCP window behavior. Table 1 compares the transmission window for Linux kernel 2.6.20, Windows Server 2003, and Windows Server 2008, using real data collected from our proxies. As we can see, both Windows Server 2003 and Linux show a similar window growth rate of about 1.5, whereas Windows Server 2008 shows a growth rate of 2. We believe the difference is because Windows Server 2008 implements Appropriate Byte Counting (ABC) [11], while Linux and Windows Server 2003 do *not*. ABC increases the TCP congestion window by the number of bytes acknowledged, compared to by the number of acknowledgements conventionally. Under delayed ACK (most common), the difference is exactly 2 vs. 1.5. Using Windows Server 2008 can immediately reduce the total latency by two round trips between the client and the proxy. Hence, all results reported in the paper use proxies hosted on Windows Server 2008 machines.

How Much Does TCP Splitting Help? Recall that each client issues six queries through the proxy and the first two responses are used to warm up the TCP transmission window between the datacenter and the proxy. The performance of the remaining four queries should reflect latencies that end-users will experience.

We now include all the cases - with and without data packet or ACK loss and present the main finding of this section: a comparison of the end-to-end latency with and without TCP splitting. Figure 6 plots the CDF of search latency with and without TCP splitting. We see that at 95^{th} percentile, TCP splitting reduces latency from 0.93 to 0.53 (both values are normalized) – a savings of 43%!

Impact of Packet Loss: Loss can occur either between the datacenters and the proxies, or between the proxies and the clients. But, as we will see next, the latter case is much more common. Indeed, from the packet traces, we observe that 7% of the TCP sessions between the proxies and the clients have at least one packet retransmission. To examine the impact of loss, Figure 7 plots the CDF of the response time for these 7% queries. From the figure, we observe that (1) when compared to all the queries (the "via proxy" curve in Figure 6), loss significantly impacts the latency – the (normalized) response time of 76% queries is greater than 1.0; and (2) some of the queries are heavily affected - the (normalized) response time of 22% queries with packet loss is greater than 2.0.

An implication of the finding is - if loss in the last mile can be handled effectively, the latency of the 7% queries can be drastically reduced.

Latency in Hauling Data from Datacenter: In our experimental deployment, each proxy maintains a persistent TCP connection with the datacenter. If packet loss occurs between the proxy and the datacenter, additional round trips are required to transmit a response. Using the last four of the total six queries per test, we now examine whether the datacenter can always transmit the entire response to the proxies in one transmission window. In particular, we examine the time gap between the proxy receiving the first and last packet from the datacenter. Ideally, this time should be close to 0, if all packets arrive in a single window. Figure 8 (*private-network* bars) shows that this is typically the case for the proxies within Microsoft's global distribution network. For comparison purposes, we have also deployed a split-TCP proxy inside the Abilene network at Purdue University (*public-network* bars in figure 8). For this proxy, in sharp contrast, about 20% of the cases take one RTT (round trip time between this proxy and datacenter), indicating that at least one packet loss has occurred. An implication of this finding is - when Cloud Service providers start to deploy satellite DCs beyond their private networks, they are more likely to encounter the so-called "middle-mile" problem [3]. In that case, a customized FEC-based low latency reliable protocol (e.g.,[12]) between the proxy and the datacenter should be beneficial.

Stress Testing: During our measurements, we were able to attract over 1000 clients in one week's time frame. This rate does not give us the opportunity to measure the performance of our system under load. Specifically, we wanted to measure the response time between the proxy and the datacenter under load. As nearby queries go through the same connection between the proxy and the datacenter, they would suffer the same fate , i.e., if one query experiences a loss, the queries succeeding it would also suffer due to TCP semantics.

To stress-test our split-TCP platform, we conducted the following experiment in all the 11 proxy locations. Due to the lack of high client arrival rate, we made each proxy issue back-to-back queries to itself destined to the datacenter. Each query fetches a single image of about 50KB size from a web server in the datacenter, over a single persistent HTTP connection. We choose to fetch a single static image since this would incur negligible time at web server and we would have only the network under the microscope. The web server was configured with unlimited HTTP requests over the persistent HTTP connection. The query rate was varied from 1 request/sec to 1000 request/sec. At each rate, we dispatched 10000 queries and waited for all the responses to be received. For every query, we measured the response time. Since we were fetching a static image from the web server, the latency of the web server itself should be negligible. Over the persistent HTTP connection with a large TCP window, the response time should be a single RTT.

Figure 9 plots the results of stress testing for 8 of the 11 proxy locations (the rest 3 similar). For every location, we plot one bar for each request rate indicating the percent of requests that took 2 or more RTT to complete. For example, for the proxy in Amsterdam, at 1 request/sec, 0.02% of the requests took 2 or more RTT. We issued a total of 10,000 requests (1 per second). This means that 2 out of the 10,000 requests took

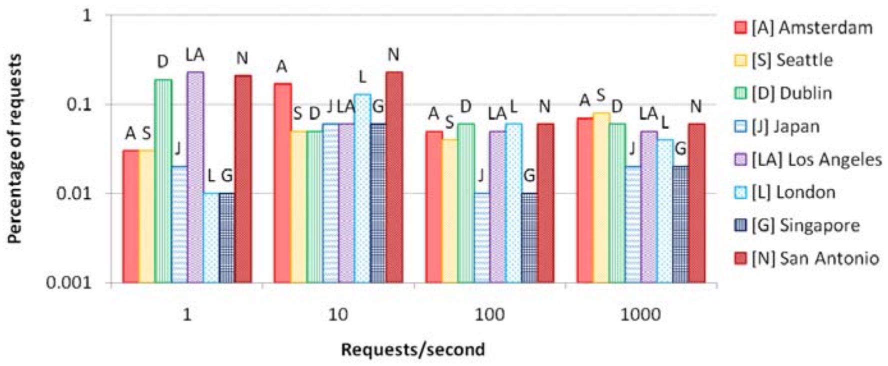


Fig. 9. Stress Test: Percentage of requests that took more 2 or more RTTs to complete

more than 2 RTT. Those were in fact the first two requests that warmed up the TCP connection. Even at a high request rate, less than 0.1% of the queries incur more than 2 RTTs. This experiment shows that, in the private network of satellite datacenters, even high load can be readily handled by persistent HTTP connections.

4 Related Work

Proposals for using persistent-connection HTTP and split-TCP proxy to improve web transfer performance can be dated back at least to the mid 90's. Early important work includes that of Padmanaban [13] and Mogul [14]. Proxies can provide additional benefit through clever techniques, such as using static content to open up TCP window [4]. Furthermore, proxies can provide benefit through adaptations of piggyback mechanisms [15]. Authors in [1][2] evaluate performance of split-TCP proxies using a very limited set of clients. Through emulation, [1] evaluated socket-level TCP splice using a single client and various latency/loss rate. The focus was to estimate the number of CPU cycles that a proxy spends processing requests. [2] estimated the latency penalties incurred by split-TCP proxies. Authors estimated that a kernel level split-TCP implementation incurs only 0.1ms. However, none of the studies were deployed and evaluated using a production environment and through a large number of real-world end-users. Moreover, none of studies dissects the splitting TCP solution from as many aspects as we do, nor do they outline the directions for further optimizations.

5 Conclusion and Future Work

In this paper, we investigate the benefits and optimizations of TCP splitting for accelerating Cloud Services. Using web search as an exemplary case study and through an experimental system deployed in a production environment, we show that TCP splitting can indeed reduce the response time of Cloud Services significantly. We also identify a number of directions for further optimizations in order to achieve the full benefit of TCP splitting.

During our experimental deployment, we observe that packet loss is rather common between the end-users and the proxies, even though the proxies are deployed in a well-provisioned and well-connected production network. This is a bit surprising, but yet consistent with observations from other production networks [4]. As an ongoing work, such reality prompts us to pursue TCP stack modifications on the proxy so as to more effectively handle packet loss and improve the latency performance.

Furthermore, along with optimizing each component in the TCP splitting system, expanding the presence of the global distribution network (and thus proxies) will also help. The holy grail question being – how many locations will be sufficient and where should these locations be? We are developing new methodologies [6] and conducting large scale studies in order to answer this question conclusively.

References

1. Ibm, R.U., Rosu, D.: An Evaluation of TCP Splice Benefits in Web Proxy Servers. In: WWW. ACM Press, New York (2002)
2. Maltz, D.A., Bhagwat, P.: TCP Splicing for Application Layer Proxy Performance. Technical report, IBM Research Report 21139 (Computer Science/Mathematics) (1998)
3. Akamai: Akamai's EdgePlatform for Application Acceleration. Akamai, Inc. (2007)
4. Tariq, M., Zeitoun, A., Valancius, V., Feamster, N., Ammar, M.: Answering What-If Deployment and Configuration Questions with WISE. In: ACM SIGCOMM (August 2008)
5. Huang, C., Wang, Y.A., Li, J., Ross, K.W.: Measuring and Evaluating Large-Scale CDNs. MSR Technical Report MSR-TR-2008-106 (2008)
6. Wang, Y.A., Huang, C., Li, J., Ross, K.W.: Measuring Network Performance for Cloud Services with AdMeasure (2009) (Submitted)
7. Mayer, M.: Web 2.0, http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html
8. Allman, M., Floyd, S., Partridge, C.: Increasing TCP's Initial Window. RFC 3390 (October 2002)
9. Krishnan, R., Madhyastha, H.V., Srinivasan, S., Jain, S., Krishnamurthy, A., Anderson, T., Gao, J.: Moving Beyond End-to-End Path Information to Optimize CDN Performance. In: ACM IMC (2009)
10. Allman, M., Paxson, V., Stevens, W.: TCP Congestion Control. RFC 2581 (April 1999) (Updated by RFC 3390)
11. Allman, M.: Tcp byte counting refinements. SIGCOMM Comput. Commun. Rev. (1999)
12. Huang, Y., Mehrotra, S., Li, J.: A Hybrid FEC-ARQ Protocol for Low-Delay Lossless Sequential Data Streaming. In: ICME (2009)
13. Padmanabhan, V.N., Mogul, J.C.: Improving HTTP Latency. In: WWW Conference (1994)
14. Mogul, J.C.: The Case for Persistent-Connection HTTP. ACM CCR (1995)
15. Cohen, E., Krishnamurthy, B., Rexford, J.: Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters. ACM CCR (1998)

The Myth of Spatial Reuse with Directional Antennas in Indoor Wireless Networks

Sriram Lakshmanan[1], Karthikeyan Sundaresan[2], Sampath Rangarajan[2], and Raghupathy Sivakumar[1]

[1] Georgia Institute of Technology, Atlanta, GA, U.S.A.
[2] NEC Labs. America, Princeton, NJ, U.S.A.

Abstract. Interference among co-channel users is a fundamental problem in wireless networks, which prevents nearby links from operating concurrently. Directional antennas allow the radiation patterns of wireless transmitters to be shaped to form directed beams. Conventionally, such beams are assumed to improve the spatial reuse (i.e. concurrency) in indoor wireless networks. In this paper, we use experiments in an indoor office setting of Wifi Access points equipped with directional antennas, to study their potential for interference mitigation and spatial reuse. In contrast to conventional wisdom, we observe that the interference mitigation benefits of directional antennas are minimal. On analyzing our experimental traces we observe that directional links do *not* reduce interference to nearby links due to the lack of signal confinement due to indoor multipath fading. We then use the insights derived from our study to develop an alternative approach that provides better interference reduction in indoor networks compared to directional links.

Keywords: Indoor wireless networks, directional antennas, spatial reuse.

1 Introduction

The growing density of wireless deployments and limited spectrum availability have motivated the need for techniques that provide increased capacity using the same spectrum. This makes smart antennas a natural solution for improving wireless network performance. Offered by several commercial vendors [4,5], smart antennas are an array of multiple antenna elements, available in different form factors. Depending on the sophistication of signal processing involved, these can be broadly classified as *multiple-input multiple-output* (MIMO) and *beamforming* antennas. In MIMO, multiple antenna elements at both the transmitter (Tx) and receiver (Rx) along with the multipath nature of the environment, help create several virtual pipes between the Tx and Rx, over which multiple independent data streams (spatial multiplexing) or dependent data streams (space-time coding) can be sent [7]. On the other hand, beamforming is a process whereby the Tx forms a beam pattern to reinforce signal reception at the Rx [7]; allowing for operation even with omni-directional clients. Using a predetermined set of fixed beam patterns is referred to as switched beamforming; forming arbitrary beam patterns on the fly in the signal space is referred to as adaptive beamforming, whose sophistication comes at the expense of channel estimation and large feedback overhead from Rx to Tx. Given the tradeoff between complexity, performance and ease of deployment, switched beamforming provides a nice balance and is consequently popular in practical WLANs [3,4].

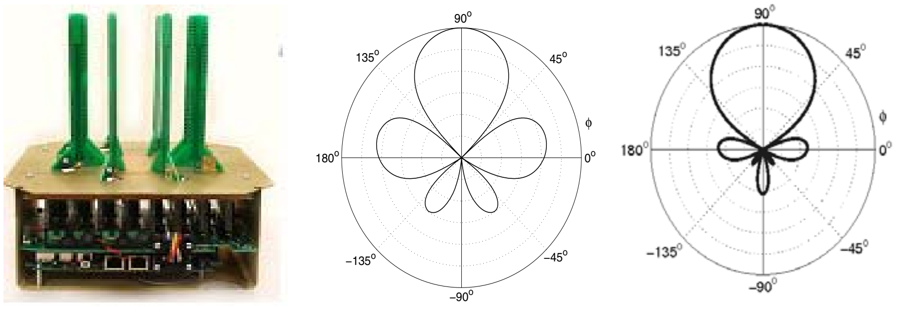
Switched beamforming is realized using an antenna array and a set of beams each of which typically focuses signal energy in a specific direction. Hence they are also called directional antennas. In addition to improving the *link quality* (SNR), they also enable multiple links to operate concurrently i.e. *spatial reuse*, by suppressing energy in unwanted directions. While their benefits are well understood in outdoor environments [8,9,10], when it comes to indoors, conventional wisdom appears to be that the multipath nature of the environment has a detrimental effect on their link quality benefits [7]. In our earlier work [2], through experimental measurements, we showed that although the effectiveness of directional antennas is reduced indoors, they still hold promise to improve the link quality. This has prompted researchers to design practical solutions [3,6] for leveraging their benefits indoors. However, given the growing density of wireless networks, improving spatial reuse becomes the more critical problem and existing works [3,6,9] assume the occurrence of spatial reuse when using directional antennas without establishing how much of it is actually available in practical indoor scenarios. Thus, it becomes critical to understand and address the following basic questions. *(i) What is the potential of directional antennas to improve spatial reuse in indoor wireless networks? (ii) Are there simple yet effective strategies that provide improved spatial reuse in indoor environments?*

To understand the potential of directional antennas for interference reduction, we perform measurements with the help of Wifi APs with eight element arrays in an indoor office. Our studies reveal the following key inferences. (1) Directional antennas *do not* hold much promise for improved spatial reuse (unlike for link quality) due to multipath scattering in the environment which cause signal leakage in unwanted directions. (2) Indoor multi-antenna channel exhibits significant channel gain variation across antennas thereby affecting directional antennas which typically split the power equally across antenna elements. (3) Selecting only a subset of the available antenna elements and splitting power across them helps strike a more efficient balance between improving the desired link quality and reducing the interference caused to other concurrent links.

The rest of the paper is organized as follows. In Section 2, we present some background on directional antennas, followed by the experimental setup and methodology in Section 3. We evaluate the potential of directional antennas for spatial reuse in Section 4. We analyze the reasons for the observed performance and propose an alternate reuse strategy in Section 5. We conclude with some discussions in Section 6.

2 Background

Conventional antennas are referred to as omni-directional antennas since they radiate power equally in all directions. Beamforming is an advancement in antenna



(a) Phocus Array (b) Directional pattern (c) Low side lobe pattern

Fig. 1. Antenna array and directional beams

technology that adjusts the shape of the radiation pattern using an array of antenna elements as shown in Figure 1(a). It is typically achieved by applying complex weights (containing a magnitude and phase) to each of the antenna elements. By applying an appropriate set of weights, the signals can be reinforced in a required direction to cause a high Signal To Noise Ratio (over an omnidirectional pattern) at the receiver in that direction. The beam patterns with such a high gain direction (main-lobe) are called directional beam patterns (e.g. Figure 1(b)).Such beam patterns also incur a spillover of energy in unwanted directions (referred to as the side-lobes). The weights can be optimized to reduce sidelobes (as illustrated in Figure 1(c)) although at the cost of reduced main lobe gain or increased beamwidth.

3 Measurement Methodology

Setup: Our experimental setup, shown in Figure 2, comprises six access points and eight clients distributed in an indoor office building. Each of the access points is a commercial 802.11g device fitted with an eight element antenna array from Fidelity-Comtech [1]. Each client is a laptop running Ubuntu 8.10 equipped

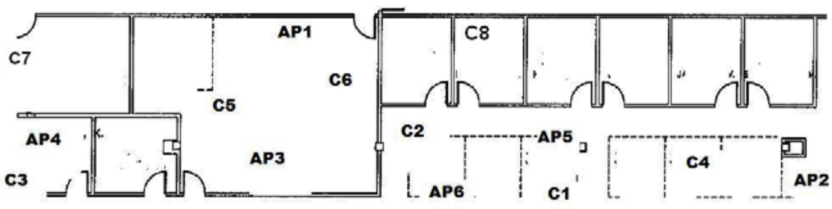


Fig. 2. Layout of Access Points and Clients

with a D-Link 802.11g card based on the Atheros AR5212 chipset. The nodes run Linux kernel v2.4.26 and the MadWiFi driver and their WLAN radios connect to external omnidirectional antennas with a gain of 6dBi.

Methodology: Each access point is pre-loaded with sixteen directional beams provided by the manufacturer that together span 360 °. These have also been used in related work [3]. For reference, we use a fixed element (element 0) for the omni-directional pattern. Throughout the experiments, the total power transmitted from each AP is fixed at 10 dBm. All experiments are performed on channel 6 in the 2.4 GHz band in a building with no external Wifi interference. Using a channel scan, it is ensured that there is no extraneous interference before each measurement. Multiple time-spaced experiments are performed for increased confidence.We use Iperf to generate traffic from each AP to its client and the 'athstats' madwifi utility on each laptop to obtain fine grained statistics from the card. In each run, 128 Byte UDP packets are sent from the APs to the intended clients. The antenna pattern at APi is fixed at 1 and that at AP$_j$ is consecutively changed from 1 to 16 after which the pattern at AP$_i$ is changed to 2 and so on. In all, this yields a total of 256 pattern combinations for a two AP two client scenario. This is repeated for other AP pairs. The received signal strength (RSSI) at clients is the metric of interest and is a measure of the received SNR reported by the card. We compute the aggregate rate of two concurrent links, using the signal and interference powers received at each client from the desired and interfering APs respectively, along with the noise floor of the radio. We consider the default association model that is prevalent in current WLANs where a client uses the strongest AP for association. We use all unique AP-client pairs (equal to 60 after taking into account the association) for the aggregate results and a representative set of two APs, AP$_1$ and AP$_2$ with clients C$_1$ through C$_8$, for the microscopic results.

Strategies: When considering the operation of multiple links in a single-hop wireless network, the following four strategies are possible: (1) Omni-Joint: Two links operate concurrently using omni-directional antennas with potential interference; (2) Omni-TDMA: Two links operate in a time division manner using omni antennas without interference; (3) Dir-Joint: Two links operate concurrently using one directional beam each from the available set with potential interference; (4) Dir-TDMA: Two links operate using one directional beam each in a time division manner without interference.

Metrics: In addition to aggregate rate of the links, we introduce two new metrics to characterize the spatial reuse.
i. Aggregate rate: We use both the 802.11g SINR table and the Shannon Capacity expression($C = \log_2(1 + SINR)$) to study the aggregate rate with practical and ideal rate adaptation. The aggregate rate for *joint* strategies is the sum of rates of the concurrent links taking interference into account ($SINR$). For *TDMA* strategies, it is given by the sum of rates on the individual links without interference (SNR) and scaled by half (due to time multiplexing).

ii. Interference power ratio: It is the factor by which the interference power at a client from an interfering AP is reduced with a directional beam compared to Omni and indirectly contributes to spatial reuse. Ideally, a highly directional beam in one direction at an AP should cause very low interference power in other beam directions. In practice, the effect of sidelobes and indoor scattering might result in several directions receiving strong interference from an AP.
iii. Spatial reuse factor: Aggregate rate by itself does not directly capture spatial reuse since it is impacted by two factors namely, link (array) gain and interference reduction (spatial reuse) gain. While the link gain improves the signal power S, spatial reuse gain comes from the reduction of interference I to jointly impact the SINR of the receiver, making it hard to isolate their contributions. However, we note that the link gain is common to both TDMA and joint strategies, while spatial reuse gain is specific to joint strategies, and hence define the following metric to better capture spatial reuse:

$$\beta = \frac{\text{Sum rate of concurrent links}}{\text{Average rate of isolated links}} \tag{1}$$

Thus, when $\beta > 1$, we have throughput benefits that are directly contributed due to interference reduction and hence spatial reuse.

4 Spatial Reuse with Directional Antennas

4.1 Aggregate Rate of Directional Links

We first study the performance improvement that concurrent directional links provide over other competing strategies. We perform experiments over all possible two link pairs (sixty pairs) in our testbed. We plot the aggregate rate CDF results (using 802.11g SINR-rate table) for each of the four strategies: Omni-Joint, Omni-TDMA, Dir-Joint, Dir-TDMA in Figure 3 along with the ideal two link rate. While directional strategies provide good throughput improvements compared to Omni strategies as expected, there are two important insights that the figure reveals: (1) Dir-Joint performs worse than Dir-TDMA for around 20%

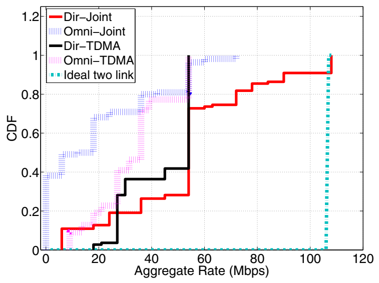


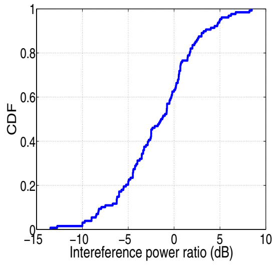
Fig. 3. Aggregate rate of directional links

of the link pairs, indicating that joint use of links is not always good. (2) For around 40% of the cases, Dir-TDMA performs better than Dir-TDMA. However, only in 10% of the cases does Dir-Joint achieve the ideal two link rate (108 Mbps). This indicates that a only a small fraction of the gains of directional beams is contributed by interference suppression. This is also confirmed by the fact that the median gain of both Dir-Joint and Dir-TDMA is the same (54 Mbps).

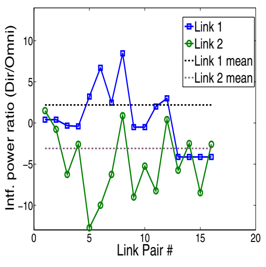
4.2 Analyzing the Performance Degradation

Interference power reduction: To understand the reasons behind marginal spatial reuse, we focus on the interference powers at the receivers. We first present the CDF of the interference power ratio over different two link configurations. By configuration, we mean each of the two APs communicates simultaneously with one of its clients using one of 16 directional beams. In the ideal case, all the beams (except that beam which points from the interfering AP to the client) should lead to very low interference power at the client. Consequently, one would expect that most of the configurations yield an interference power ratio less than 1 (0 dB). Results in Figure 4(a) reveal the surprising observation that for more than 40 % of the configurations, the use of directional beams *increases* the amount of interference caused (compared to omni) rather than decreasing it. This clearly shows that unlike expected in an ideal free space environment, most beams cause significant interference powers to be radiated towards unintended clients. This is shown in greater detail in Figure 4(c) which plots the interference power reduction for a single two-link pair across beam configurations alone. This non-ideal interference reduction occurs due to a combination of factors such as side lobes, scattering, varying channel gains across different elements, etc. However, the net impact is that *directional antennas do not provide the assumed interference power reduction as expected.*

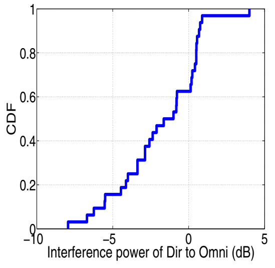
We further analyze the interference power reduction assuming that an ideal beam selection algorithm is used. i.e the best combination (pair) of beams that maximizes the aggregate throughput of the two concurrent links is used. The



(a) Intf. reduction

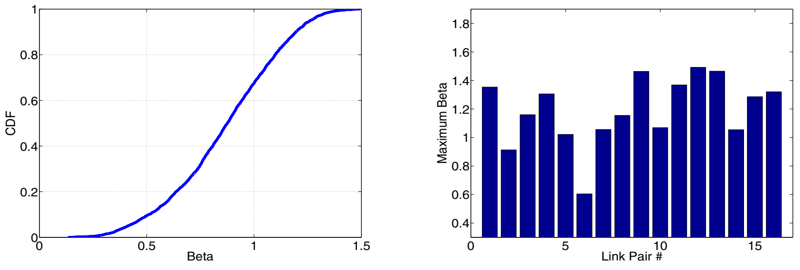


(b) Intf. - best rate beam



(c) Intf. - fixed link

Fig. 4. Interference power reduction of directional beams



(a) cdf of β (b) Best β across link pairs

Fig. 5. Spatial reuse factor β

resulting interference power for this case is plotted for link 1 and link 2 for 16 pairs of client locations in Figure 4(b). From the figure, we observe that while some of the link pairs do indeed have a reduced interference power compared to omni, most of the pairs have increased (> 0 dB) interference power. The mean interference is increased by 3 dB and reduced by -3 dB for link 1 and link 2 respectively indicating that the *interference suppression is minimal even with ideal beam adaptation.*

Distribution of spatial reuse factor β: For each pair of links, the signal and interference powers are measured at the clients while changing the beam pattern on the two APs chosen as described in Section 3 for the Joint and TDMA directional strategies. The CDF of β over all link pairs and beam configurations is presented in Figure 5(a). The results reveal that: (1) Unlike a free space ideal environment, where the expected improvement is $\beta = 2$, the maximum improvement is much less indoors (i.e around $\beta = 1.45$). (2) In an ideal environment where directionality is preserved, the CDF of β is expected to indicate a sharp rise near 2. However, in practice, the CDF is distributed from 0.3 to 1.45 indicating that the expected directionality and consequent interference suppression are not obtained in practice for a majority of link and beam pair configurations. (3) While the median improvement is expected to be close to $\beta = 2$ with good interference suppression, the actual median improvement is 0.9! Further, more than 70% of the configurations have $\beta < 1$, indicating that for a majority of scenarios, there is no benefit from spatial reuse with directional antennas, where TDMA using link gain performs better.

Performance with optimal beam selection: We also fix two APs, AP1 and AP2, and determine the best beam combination (pair) for concurrent operation (Dir-Joint) and best directional beams for isolated operation (Dir-TDMA) for the resulting 16 link pairs, results of which are presented in Figure 5(b). Note that this Dir-Joint is an exhaustive version of the greedy algorithm in [3] and thus upperbounds the gains from spatial reuse. The maximum value of β for the 16 link pairs is plotted in Figure 5(b). The figure indicates that the best improvement using concurrent directional beams is distributed from 0.6 to 1.45. Additionally, there are link pairs (namely 2,5,6) for which Dir-Joint performs worse than Dir-TDMA. More interestingly, the average β across link pairs is just 1.17, meaning that the contribution from spatial reuse improves performance by only around 17%.

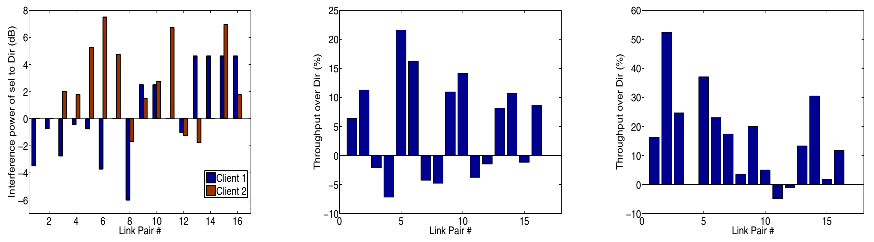
Thus, our experiments highlight that *while directional antennas provide gains over omni in indoor environments, the benefits are contributed largely by link gain and not by spatial reuse.*

5 Alternate Strategy for Spatial Reuse

To gain a deeper understanding, we study the indoor multiple antenna channel. Our experiments reveal that the channel gain varies drastically (over 10 dB) even for antennas which are closely spaced due to multipath fading. This drastic variation in channel gain motivates adapting the power transmitted from each antenna. This unequal gain across antennas also makes directional beams sub-optimal because directional beams are typically created by splitting the transmit power equally across the antenna elements. To verify this proposition, we consider a strategy called *antenna selection*, where the antenna element with strongest gain is alone chosen for transmission and allocated all the power.

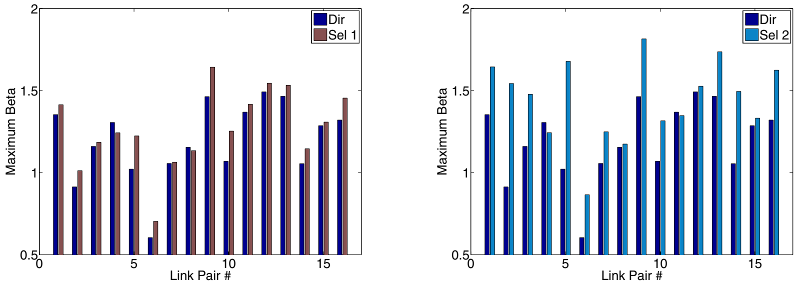
We experiment with fifty links (each activated in isolation) and measure the Signal to Noise Ratio at the receiver for both directional and selection of the best of eight elements. We observe that for more than 80% of the locations, antenna selection *outperforms* directional beamforming. Given the highly varying nature of the channel, a strategy that uses the higher gain channels (antennas at the Tx) would lead to higher SNR at the receiver. However, in a multi-link scenario, the transmitter must also minimize the interference it causes to other receivers. Hence, we study how antenna selection performs in a two link configuration with AP1 and AP2 in our testbed. We try out all possible combinations of single element selection. i.e AP1 and AP2 select antenna elements i and j respectively, $\forall i, j \in [1, 8]$. We then compute the pair of antenna elements (one for each link) which yields the highest aggregate throughput for each link pair. The corresponding interference powers at the two clients is noted. We then compute the difference in interference power reduction compared to Dir-Joint for each link and plot the results in Figure 6(a). It can be seen that, while selection performs well from a single link standpoint, it's performance in a multi-link context is not as significant. i.e it does not provide a large reduction in interference power compared to Dir-Joint.

Link gain - Interference reduction trade-off: The previous experiments highlight the fact that the multi-link scenario is more constrained and both the desired and interfered clients must be considered when choosing the transmission beam pattern. Consequently, from a multiple antenna usage standpoint, there is a trade-off between the link (diversity gain) and the interference suppression gain. Using single element selection leverages the link to the desired client best, because it transmits all the power on the highest gain channel. However, this



(a) Selection - two links (b) 1 element selection (c) 2 element selection

Fig. 6. Antenna selection rate improvements



(a) 1 element selection (b) 2 element selection

Fig. 7. Spatial reuse factor β for antenna selection

may not correspond to least interference to other clients. Similarly, selecting all available elements with equal power split also provides no flexibility in interference suppression and reduces array gain. On the contrary, selecting a small subset of elements with equal power split (antenna subset selection), provides the flexibility to tradeoff some array gain to choose antenna elements that also provide good interference suppression. To verify whether the trade-off occurs in practice, we evaluate the aggregate throughput improvements of both one element and two element selection. The throughput gain of one element selection compared to Dir-Joint is plotted in Figure 6(b). The figure reveals that single element selection provides throughput gains over directional for some link pairs but not all. We then plot the throughput gains of two element selection in Figure 6(c). This figure reveals two important facts: (1) The maximum throughput gain with one element selection is around 23% whereas the gain of two element selection is 53%. (2) More locations have a positive gain over directional. These results indicate that antenna subset selection is a promising approach.

Further, we evaluate the spatial reuse factor β for the single element and two element selection strategies for each of the link pairs. The results plotted in Figure 7 indicate that the improvements are more pronounced with two element selection reaching up to a factor of 1.6 (which is closer to the maximum β of 2 for two links). Additionally, the spatial reuse factor averaged over links increases from 1.19 with directional to 1.26 with single element selection and 1.42 using two element selection, thus confirming that antenna subset selection is a *better spatial reuse strategy* compared to directional.

6 Conclusion

The ability of directional antennas to reduce interference and consequently improve the spatial reuse in indoor wireless networks is impacted significantly by multipath scattering and fading. Additionally, indoor wireless channels have large gain variations across multiple antennas which motivates strategies that adapt the power transmitted on each antenna element. Selection of a subset of antennas enables such adaptation with low complexity. Intelligent algorithms to identify the right subset of antenna elements and comparison of its performance with closed loop MIMO strategies is an interesting avenue for future work.

References

1. Phocus Array, `http://www.fidelity-comtech.com`
2. Blanco, M., et al.: On the Effectiveness of Switched Beam Antennas in Indoor Environments. In: Claypool, M., Uhlig, S. (eds.) PAM 2008. LNCS, vol. 4979, pp. 122–131. Springer, Heidelberg (2008)
3. Liu, X., et al.: DIRC: increasing indoor wireless capacity using directional antennas. In: ACM SIGCOMM 2009 (2009)
4. Ruckus Wireless Inc., `http://www.ruckuswireless.com`
5. Cisco Inc., `http://www.cisco.com`
6. Subramanian, A.P., Lundgren, H., Salonidis, T.: Experimental Characterization of Sectorized Antennas in Dense 802.11 Wireless Mesh Networks. In: ACM Mobihoc 2009 (2009)
7. Paulraj, A., Nabar, R., Gore, D.: Introduction to space-time wireless communications. Cambridge University Press, Cambridge (2003)
8. Navda, V., et al.: MobiSteer: Using Steerable Beam Directional Antenna for Vehicular Network Access. In: ACM MobiSys 2007 (2007)
9. Choudury, R.R., Yang, X., Ramanathan, R., Vaidya, N.: On designing MAC protocols for wireless networks with directional antennas. IEEE Transactions on Mobile Computing 5, 477–491 (2006)
10. Das, S.M., et al.: DMesh: Incorporating practical directional antennas in multichannel wireless mesh networks. IEEE Journal on Selected Areas in Communication 24(11), 2028–2039 (2006)

Influence of the Packet Size on the One-Way Delay in 3G Networks

Patrik Arlos and Markus Fiedler

Blekinge Institute of Technology
Karlskrona, Sweden
{patrik.arlos,markus.fiedler}@bth.se

Abstract. We currently observe a rising interest in mobile broadband, which users expect to perform in a similar way as its fixed counterpart. On the other hand, the capacity allocation process on mobile access links is far less transparent to the user; still, its properties need to be known in order to minimize the impact of the network on application performance. This paper investigates the impact of the packet size on the minimal one-way delay for the uplink in third-generation mobile networks. For interactive and real-time applications such as VoIP, one-way delays are of major importance for user perception; however, they are challenging to measure due to their sensitivity to clock synchronisation. Therefore, the paper applies a robust and innovative method to assure the quality of these measurements. Results from measurements from several Swedish mobile operators show that applications can gain significantly in terms of one-way delay from choosing optimal packet sizes. We show that, in certain cases, an increased packet size can improve the one-way delay performance at best by several hundred milliseconds.

1 Introduction

Increasingly many devices use mobile connectivity for the exchange of data. Users expect the emerging mobile broadband to perform in a similar way as its fixed counterpart, no matter to which extent the medium is shared. In the third generation of mobile communications, represented by WCDMA and HSDPA, the per-user capacity allocation depends amongst others on the radio conditions, the user density, the mobility pattern, the offered traffic, etc. It is, however, not communicated explicitly towards user and the applications that might need this information for yielding the best performance, given the specific allocation. This imposes the need for end-to-end measurements with the goal to highlight network impact on the performance parameters of interest.

Given this background, this paper investigates the impact of the packet size on the minimal one-way delay (OWD) for the uplink in third-generation mobile networks, which is an important performance parameter for interactive and real-time applications. In particular, the minimal OWD provides information about the best-possible performance with given settings, undisturbed by congestion, radio problems, etc. However, we should not omit importance for the uplink's

OWD behaviour as it will affect the TCP performance accross mobile networks, as the acknowledgement packets utilizes this link. However, in this paper, we will not investigate the average and maximum OWD values for the uplink, as these are more likely to exhibit temporal and spactial artifacts, varying from test to test. As OWD measurements are very sensitive to clock synchronisation issues, the paper also presents and demonstrates a robust and innovative method to assure the quality of these measurements. We focus on the minimal OWD as our goal is to investigate the best-possible system behaviour, instead of the statistical behaviour of the system. Hence, our results can be seen as best case with regards to the OWD perceived by an application.

The paper is organized as follows. First, we describe and verify the measurement method in Section 2. Section 3 describes and discusses the experimental setup and analysis procedure. In Section 4, minimal OWD in several Swedish networks are evaluated as a function of the packet size. Section 5 concludes the paper and points out future work.

2 Method

The fundamental problem when evaluating the OWD is how to handle the clock synchronization. The OWD is, as such, simple to calculate. The OWD of the i^{th} packet, d_i, is calculated as:

$$d_i = T_{i,b} - T_{i,a} \tag{1}$$

where $T_{i,a}$ is the arrival time of the i^{th} packet at location a; correspondingly $T_{i,b}$ is the arrival time of the same packet at location b. In the general case, the time stamps ($T_{i,x}$) are obtained from two different clocks. To get an unbiased OWD estimate, these clocks should be synchronized. In [1] the authors investigate the three main synchronisation methods NTP, GPS and IEEE1588 used for OWD measurements. Usually the Network Time Protocol (NTP) [2] is used. This enables the clocks to be synchronized within $10 - 20$ ms for WAN, and < 1 ms for LAN. If the synchronisation needs to be better, then a GPS solution is needed. Together with NTP, this allows a synchronisation in the order of 1 μs. The current state of the art is to use Endace [3] DAG cards together with a GPS, then the theoretical synchronisation is in the range of 60 ns. However, according to our own experience [6], this is difficult to obtain in practice, as we still have two independent clocks. In [4] the author described the internal functioning of the time-keeping in side of the DAG cards, and in [5] the authors describe a method to synchronize clocks accross the Internet. Regardless of what method or technique used for synchronisation, the OWD estimations can at the worst be twice that of the synchronisation level [6].

Our method uses wiretaps and a special wiring in conjunction with DAG cards to obtain the time stamps from the same clock. In Figure 1 a schematic of the wiring is shown. When a packet is sent from SRC to DST it will travel across the upper wire (dashed line). As is passes the first wiretap A, a copy of the packet is made and is sent to the interface dag00, where it arrives it at time $T_{1,\mathrm{A}}$. At the same time the original packet makes its way across the network and eventually

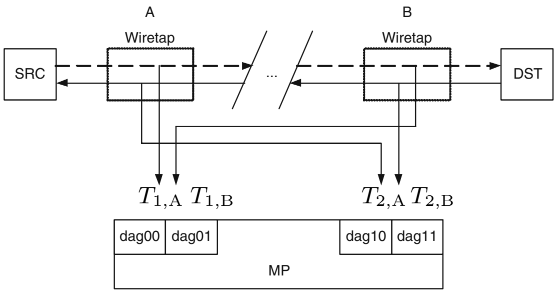

A
B
Wiretap
Wiretap
SRC
DST
...
$T_{1,\mathrm{A}}$ $T_{1,\mathrm{B}}$
$T_{2,\mathrm{A}}$ $T_{2,\mathrm{B}}$
dag00
dag01
dag10
dag11
MP

Fig. 1. Wiring method

reaches wiretap B. Here, a copy is sent to interface dag01, where it is received at $T_{1,\mathrm{B}}$. Similarly, if a packet is sent from DST to SRC the packets are duplicated by the wiretaps and made available to the dag1x interfaces. The main drawback with this wiring is that we require close proximity between SRC and DST. The actual distance is determined by the technology that carries the traffic from the wiretap to the DAG cards. The main benefit with this wiring, is that the packet will be time stamped by the same clock, thus subject to the same drift/skew if present.

Let t_0 be the time when the packet actually passes wiretap A, and t_1 when it passes wiretap B. Then $T_{1,\mathrm{A}} = t_0 + L_a/P_s$, where L_a is the cable length between wiretap A and dag0, and P_s is the propagation speed in that cable. Similarly, we define $T_{1,\mathrm{B}} = t_1 + L_b/P_s = t_0 + L/P_s + L_b/P_s$, where L_b is the cable distance from wiretap B to dag0, and L is the cable distance between wiretap A and B. The OWD is then obtained as:$\Delta = T_{1,\mathrm{B}} - T_{1,\mathrm{A}} = L/P_s + \frac{L_b - L_a}{P_s}$. So if we select $L_b = L_a$ we cancel the second factor and obtain the desired OWD between the wiretaps.

To verify the method, we conducted two experiments using the setup as shown in Figure 1. In the first experiment the network in between the wiretaps was replaced by a 10 m CAT5e cable, in the second a 25 m cable was used instead. The CAT5e cable has a P_s of $0.59c \sim 0.64c$ [7,8], where $c = 299\,792\,458$ m/s. The theoretical propagation time (P_t) is then between 52.1 ns and 56.6 ns for the first experiment, and 130.3 ns and 141.4 ns for the second. In Table 1 we show the corresponding results. As we are limited by the DAG 3.6 card resolution of approximately 60 ns [4,6], our OWD values are multiples of this. Our results are within the span given by the theoretical results. Obviously, the method allows us to accurately detect changes in the OWD on the scale of a few nanoseconds.

Table 1. Summary statistics from the verification experiments

Exp	Minimum	Mean	Max	Std.dev	Theoretical
	[ns]	[ns]	[ns]	[ns]	[ns]
1	0.0	54.7	119.3	17.4	$52.1 \sim 56.6$
2	119.0	139.2	179.0	28.2	$130.3 \sim 141.4$

3 Setup

To evaluate the mobile networks, we used the setup shown in Figure 2. Here SRC is sending traffic to DST. This is done via a Gateway (GW) that uses a Huawei E220 USB modem to connect to the mobile network. In-between the SRC and GW, we placed wiretap A. The other wiretap B is placed just in front of DST. The SRC and DST (both are P2-400 MHz with Linux 2.4 kernels) are connected with 10 Mbps full-duplex Ethernet cards (3Com). The GW is a Dual AMD Athlon 64 with 2 Gbytes of RAM (Windows XP SP2). The GW was configured for Internet sharing of the mobile network and no firewall was active. The SRC computer connected directly to the built-in Ethernet card (Broadcom) of the GW. The wiretaps feed into a Distributed Passive Measurement Infrastructure [9] enabled Measurement Point (MP) that stored the packet trace to file. Furthermore, the DAG cards were synchronised using both NTP and GPS.

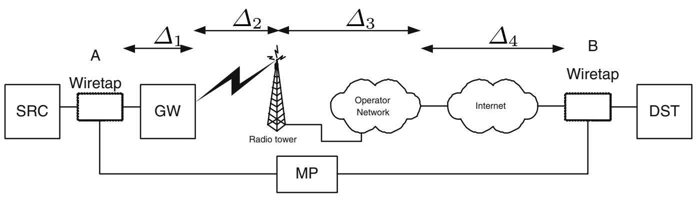


Fig. 2. Setup used in experiments

3.1 Traffic Generation

To generate data we used a C++ program that sends UDP datagrams and allows us to control packet sending rate and datagram size. Furthermore, the progam uses an application layer header with three fields. These fields allow us to separate experiments (experiment id), experiment run (run id), as well as packets within a particular experiment run (sequence number). The sequence number starts at zero and is incremented by one for each transmitted datagram. Based on these three fields, we can uniquely identify each packet, thus avoiding any ambiguities associated with hashing.

During the evaluation of the mobile networks we used two streams running in parallel. The first sends one packet of size K bytes every T_s second, which is done 200 times. It then waits for a fixed amount of time, and then starts to send another batch of 200 packets, this time with a larger packet size. The procedure is then repeated for all the packet sizes we wish to investigate. The second stream runs continously throughout the evaluation of all the different packet sizes, sending one 48 byte packet every 10 second. The purpose with this stream is to detect any time-of-day based variations in the network. As the purpose is to find the OWD, we do not want to stress the system so that it needs to queue our traffic. Using the two streams we will at most inject $(1468 + 48)$ bytes during one second.

3.2 Delay Calculation

As we are in control of both sender and receiver, we can easily identify both sending and receiving IP address as well as UDP port numbers. We then use the application header for the individual packet identification. Once identified, we can calculate the OWD for the individual packets. Using the same notation as before, the delay would be calculated as defined in Equation 1. However, due to numerical issues [6], this is not recommended. It is better to use the following equation:

$$d_i = \widetilde{T_{i,a}} - \widetilde{T_{i,b}} \quad \widetilde{T_{i,x}} = T_{i,x} - \lfloor T_1 \rfloor, \tag{2}$$

where T_1 is the arrival time of the first packet leaving the sender in that experiment. This will avoid having the time stamps truncated by the precision of the analysis tool.

3.3 Delay Components

The OWD that we will calculate has four contributors, see Figure 2. Δ_1 is the delay contribution by the GW, Δ_2 that of the radio network, Δ_3 that of the core network of the operator, and the last contribution Δ_4 comes from the Internet. Out of these four, we cannot measure or estimate Δ_2 and Δ_3 alone, as this means entering the domain of the operator.

We can estimate Δ_4 by using ICMP ping to the operator's Internet exchange. From our vantage point in the Internet, the operators are between five or six hops away, and between us and them we have the Swedish University Network (SUNET) [11] with optical multi-gigabit links. Hence the impact of this will not be negligble, but it will be quite small and stable, the average RTT between DST and the operator Internet exchange is 15 ms for all three operators. Hence, as the links are symetrical, the OWD contribution will be around 7.5 ms. Furthermore, we can ignore the packet size as the links have such high capacity that the serialisation delay is neglible [10].

In order to quantify Δ_1, we designed a special experiment. Instead of using the E220 USB, we replaced it with a D-link DUB-E100 FastEthernet USB adaptor that allows us to connect directly to the destination through Ethernet. As we are using a USB NIC, the packets travelling across this NIC will receive the same treatment as those that are sent across the modem.

From the collected data, the second stream did not detect any time-varying behavior. In fact, 37% of the packets experienced a delay less than 0.11 ms, and the maximum delay was 1.2 ms. The mean was 0.4855 ms with a standard deviation of 0.3362 ms. Apart from this, the GW seems quite stable in its handling of the packets. In Figure 3 we show the OWD through the GW for different packet sizes. We see that the OWD increases linearly, as expected [10]. The peaks noticable for the maximum values, are the result of single packets experiencing larger delays. Furthermore, the largest minimum OWD is just above 1.3 ms. If we use the minimum delay as a base, we can construct a rough model for Δ_1 given in ms:

$$\Delta_1(L) = 8.354\text{e}{-5} \cdot L + 0.078 \tag{3}$$

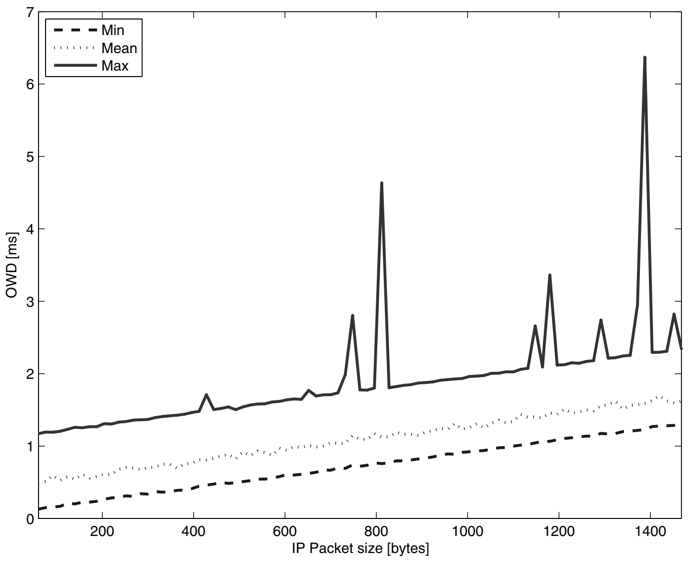


Fig. 3. Minimal OWD through the GW for different packet sizes

Here L represents the IP packet length in bytes. The 0.078 ms represents the minimal time through the GW, and the constant (8.354e−5) corresponds roughly to the capacity of the interface, i.e. 10 Mbps.

As both Δ_1 and Δ_4 turned out to be significantly smaller than the OWD measured in the subsequent experiments, they are neglected from now on.

4 Evaluation of Mobile Networks

We conducted experiments on three different Swedish operators, the experiments were conducted at the end of September and early October 2009. Two of them (A and B) share the radio access (RA), while the third (C) uses a different RA. The experiments focus on the OWD of the uplink; furthermore, all experiments were done while the sender was stationary. We focus on the uplink as this is believed to be the narrowest part in the end-to-end chain. Furthermore, to minimize the impact of daily patterns, congestion, radio problems, queueing, etc., we focus our analysis on the minimum OWD obtained in the experiments. However, just to give an indication on the environments, mean OWD was in the range of 140 ms to 700 ms, while the maximum was between 200 and 3870 ms, and the standard deviation was found between 7 ms and 480 ms.

In the first experiment we investigated IP packet sizes starting at 60 bytes, and incrementing 16 bytes for each run, up to 1468 bytes, at a sending rate of one packet per second. We start by looking on the variations over time in the OWD of individual packets (one 48 byte packet every 10 second), shown in

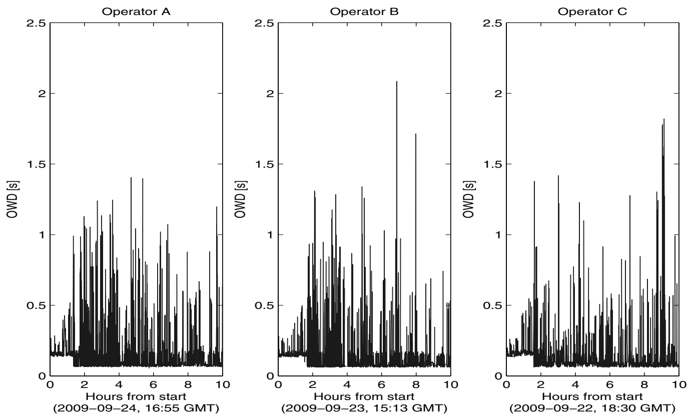


Fig. 4. Long term evaluation of the operators

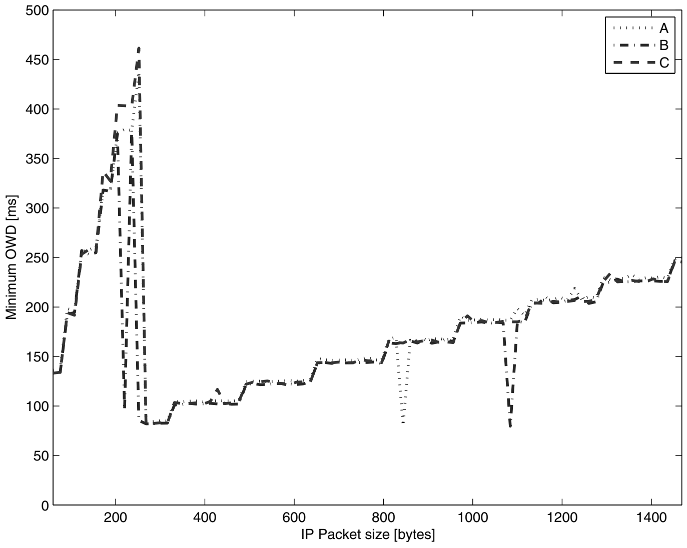


Fig. 5. Minimum OWD accross different operators

Figure 4. We see that all graphs look quite similar, and around 1–2 hours into the experiment, the smallest OWD drops to a new, stable level. The peak values are typical for mobile networks and originate from the ARQ mechanisms of the RA.

Now, we turn our attention to the minimum OWD obtained when varying the packet sizes, which is illustrated in Figure 5. There are two clear regions, one from 60 to 252 bytes and the other from 252 bytes and above. From the E220 GUI (MobilePartner), we get an indication of what service the operator

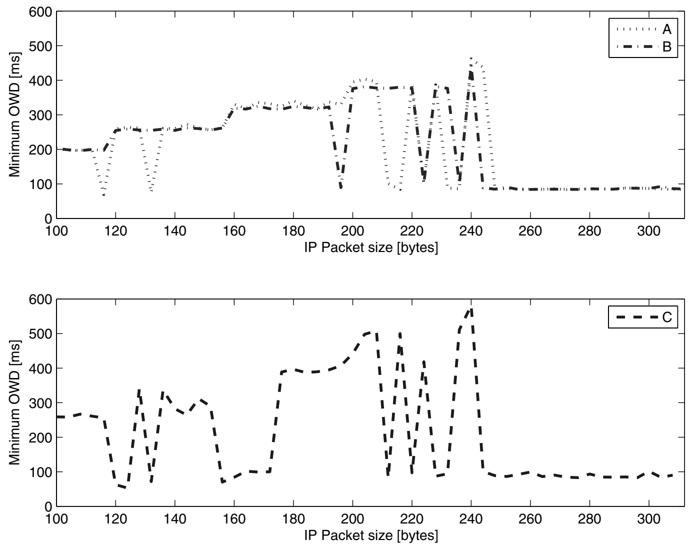


Fig. 6. Minimal OWD for packet sizes from 100 to 300 bytes

is providing the modem with. However, the GUI only reports WCDMA or HSDPA as the service. When the experiments were conducted, all operators started by giving a WCDMA service. As the packet sizes increased they started to alternate between WCDMA and HSDPA, and eventually changed permanently to a HSDPA service. The point where the change was made varied a little bit from operator to operator, but it happened around 236 to 256 bytes, matching nicely with the drop of the OWD in Figure 4. Looking at the second region, we clearly see a staircase pattern. The steps are 144 bytes wide, and the step height is approximately 18 ms for all operators. This indicates that somewhere in the end-to-end path, some entity sends the data in blocks of approximately 144 bytes. In case of operator A and B, different behaviours were seen for two packet sizes, 844 bytes for operator A and 1084 bytes for operator B. The reason for this behaviour merits further investigation. It is also worth noting that both operators exhibited a very small minimum OWD around 80 ms, which is the smallest value in the HSDPA region.

In order to gain more insight into the behaviour of the minimal OWD at the transition between WCDMA and HSDPA regions, we conducted a second and more detailed experiment. The packet size was incremented from 100 bytes to 300 bytes with an increment of 4 bytes. The result is shown in Figure 6. The upper graph holds operator A and B, while the lower holds operator C. We have separated the graphs to highlight the patterns for operator A and B. For those operators, the staircase pattern is present, however not as pronounced as before. The width of the step is around 36 bytes, and the step height is approximately 60 ms. During the evaluation of the packets, both operators started by providing WCDMA service, then operator A changed (temporarily) to HSDPA for the 116 and 132 byte packets, while operator B didn't offer HSDPA until we reached a packet size of 196 bytes. But subsequently, both operators tend to offer HSDPA more frequently, and after 248 bytes both only offered HSDPA service. Looking at the measurements taken from operator C, that pattern is much less clear, the operator had started switching to HSDPA earlier. Due to this, the same staircase pattern for packet sizes up to 188 bytes is not as clearly visible as for operators A and B.

5 Conclusions and Outlook

Based on quality-assured measurements from three Swedish mobile operators, this paper provides insights into the relationship between packet sizes and one-way delays, revealing the corresponding operator's resource allocation policy on the WCDMA/HSDPA uplink. Hereby, the quality of the one-way delay (OWD) measurements has been assured by a specific set-up and cabling scheme between Endace DAG cards that avoids common clock synchronisation problems, and by a quantification of the delay contribution of the gateway feeding the mobile link.

The most surprising result is that short packets might need more time to reach the receiver than long packets. Short packets experience a rather steep increase of OWD as their size is growing. For packet sizes in the range of 100 to 250 bytes, the minimal OWD varies heavily. For larger packets, it grows starting from quite small values (less than 100 ms) with a quite decent gradient. In both cases, the minimal OWD is increased approximately stepwise as a function of the packet size, which means that the maximal packet size per step allows for maximizing the throughput without paying for it in terms of extra delay. Thus, our method and results deliver guidelines for application programmers to make the best out of mobile connectivity w.r.t. delay and throughput by choosing optimal packet sizes. In our case, packet sizes of at least 250 bytes avoid the potentially large and strongly varying miminal OWDs associated with smaller packets. Once again, the study shows the necessity to investigate the characteristics of network connectivities if these are not explicitly known. The outliers detected for operator A and B also merit further investigation.

Of course, it has to be observed that due to radio and network conditions, the actual OWD is likely to exceed the minimal value under consideration. Nevertheless, the minimal OWD indicates the best performance that can be expected, given the chosen packet size. The examination of further OWD statistics is left for future work.

The results obtained so far motivate the use of the proposed measurement method on the downlink and comparison of the results with each other and with measured roundtrip times. The quite significant delay for small packets might affect the effective throughput of downloads using TCP, as the acknowledgements are small packets carried on the uplink. Thus, further work will include the study of the impact of the discovered allocation policy onto TCP performance.

Acknowledgements

We would like to thank Ravichandra Kommalapati for conducting the experiments.

References

1. De Vito, L., Rapuano, S., Tomaciello, L.: One-Way Delay Measurement: State of the Art. IEEE Transactions on Instrumentation and Measurement 57(12), 2742–2750 (2008)
2. Mills, D.: RFC1305 Network Time Protocol (Version 3), Specification, Implementation and Analysis
3. Endace Measurement Systems, http://www.endace.com (verified in January 2010)
4. Donnelly, S.: High Precision Timeing in Passive Measurements of Data Networks, Ph.D. Thesis, The University of Waikato (2002)
5. Veitch, D., Babu, S., Pásztor, A.: Robust Synchronization of Software Clocks Across the Internet. In: Proc. Internet Measurement Conference (2004)
6. Arlos, P.: On the Quality of Computer Network Measurements, Ph.D. Thesis, Blekinge Institute of Technology (2005)
7. Draka: SuperCat OUTDOOR CAT 5e U/UTP, http://communications.draka.com (verified January 2010)
8. Messer, J.: Ethernet FAQ, http://www.networkuptime.com/faqs/ethernet (verified January 2010)
9. Arlos, P., Fiedler, M., Nilsson, A.: A Distributed Passive Measurement Infrastructure. In: Proc. Passive and Active Measurement Workshop (2005)
10. Constantinescu, D., Carlsson, P., Popescu, A.: One-way Transit Time Measurements, Research Report, Karlskrona, Sweden (2004)
11. High Level Design Description for Sunet, http://basun.sunet.se/aktuellt/Opto-sunetDesignv10.pdf (verified 2009-10-09)

An Experimental Performance Comparison of 3G and Wi-Fi

Richard Gass[1] and Christophe Diot[2]

[1] Intel Labs
[2] Thomson

Abstract. Mobile Internet users have two options for connectivity: pay premium fees to utilize 3G or wander around looking for open Wi-Fi access points. We perform an experimental evaluation of the amount of data that can be pushed to and pulled from the Internet on 3G and open Wi-Fi access points while on the move. This side-by-side comparison is carried out at both driving and walking speeds in an urban area using standard devices. We show that significant amounts of data can be transferred opportunistically without the need of always being connected to the network. We also show that Wi-Fi mostly suffers from not being able to exploit short contacts with access points but performs comparably well against 3G when downloading and even significantly better while uploading data.

1 Introduction

Wireless communication is an important part of everyday life. It allows people to stay connected with their jobs, family, and friends from anywhere there is connectivity. The two dominant wireless technologies are Wi-Fi and third generation cellular (3G) networks.

IEEE 802.11, commonly known as Wi-Fi, refers to a set of standards which operate in the unregulated ISM band[1]. They are very well known for providing wireless connectivity in homes, offices, and hot-spots. They provide throughput of up to 600 Mbits/s[2] with a coverage area in the hundreds of meters. Wi-Fi is easy and inexpensive to deploy, and is ubiquitous in urban areas. Despite access controls being deployed and newer access points (APs) being configured with security enabled by default, many Wi-Fi APs remain open[9]. In addition, the growing popularity of community networks such as FON[1] and the growing list of large cities providing free wireless makes opportunistic communication a realistic scenario in urban areas.

Due to the sparse and non-coordinated deployment of APs, Wi-Fi is not an "always connected" technology. It is designed primarily for the mobile user that accesses the network while relatively stationary. It provides high data rates between locally connected clients but is limited by the capacity of the link between the AP and the Internet.

[1] www.fon.com

3G is based on technology that has evolved to fill the growing need for data in wireless voice networks. 3G provides seamless connectivity across large coverage areas with advertised data rates of 2 to 14 Mbits/s, shared among all users connected to any given base station. 3G network operators charge either based on consumption or have flat rate monthly plans. These networks are expensive to deploy and the performance experienced by users is sensitive to the number of users in a cell due to the large coverage areas.

For data applications, one could argue that persistent connectivity may not be necessary. Instead, being connected "frequently enough" should be acceptable if applications and communications protocols could take advantage of short, but high bandwidth contact opportunities.

We present results of a side by side, Wi-Fi vs 3G face-off. We show that with default access point selection (greatest signal strength), unmodified network setup methods (scan, associate, request an IP address with DHCP), and off the shelf equipment with no modifications or external antennae, opportunistic Wi-Fi performance is comparable to 3G. Despite only connecting to open or community access points in a typical urban residential area, Wi-Fi throughput surpasses 3G at walking and driving speed while uploading data and is nearly equivalent to 3G while downloading.

The remainder of this paper is organized as follows: We first explain how the experiments were conducted and describe the equipment and software setup in Section 2. Next, in Section 3, we show the results of the experimental runs with the comparisons of 3G vs Wi-Fi under driving and walking conditions as well as look at the effects related to the uploading or downloading of data. Finally, we discuss related work in Section 4 and conclude the paper in Section 5.

2 Experiment Description

The experiments consist of two mobile clients and a server that is always connected to the Internet. One mobile client uses its Wi-Fi interface to transmit and receive data to/from the server and the other uses 3G. Experiments are performed both on foot and in a car following the same route. Wi-Fi and 3G tests are run simultaneously for a true side-by-side comparison. While downloading, the data originates at the servers and is streamed down to the mobile clients. Conversely, when uploading, the data originates on the mobile clients and is streamed to the servers.

We investigated the potential of using the 3G device for collecting both 3G and Wi-Fi data but discovered that stationary Wi-Fi transfers in the uplink direction were capped around 6 Mbits/s, well below the advertised rates of an 802.11G enabled interface. We also saw variations in the Wi-Fi throughput while running simultaneous 3G and Wi-Fi experiments on the same mobile device. Due to these limitations, we chose to use a separate platform for each technology.

2.1 Server Setup

The servers run the Ubuntu distribution of Linux (version 8.04.1 with a 2.6.24-19-server kernel) and are publicly accessible machines on the Internet that are the source or sink for the clients. The servers are virtual machines running on the Open Cirrus cluster[11] hosted at Intel Labs Pittsburgh (ILP). The dedicated Internet connection to ILP is a 45Mbit/s fractional T3 and did not pose any restrictions in these experiments. We ran extensive tests of the code on the virtual machines and saw no performance related issues with the system or the network.

The 3G server runs the `apache` web server and hosts large, randomly generated data files that can be downloaded by the client. The Wi-Fi server runs a simple socket program that generates data with `/dev/random` and streams it down to the Wi-Fi client. When data is being uploaded from the client, both the 3G and Wi-Fi server run our socket program that receives the data and sends it to `/dev/null`. The network interfaces for both servers are monitored with `tcpdump` and the resulting data traces are stored for off-line analysis.

2.2 Wi-Fi Client

The Wi-Fi client setup consists of an IBM T30 laptop with a default install of the Ubuntu distribution of Linux (version 8.04 with a 2.6.24-21-server kernel). The internal wireless device is the Intel 2915ABG network card using the unmodified Intel open source Pro/Wireless 2200/2915 Network Driver (version 1.2.2kmprq with 3.0 firmware). No external antenna is connected to the laptop for the experiments.

The laptop attempts to connect to the Internet by first scanning the area for available open or community APs (excluding those with encryption enabled and those we have marked as unusable[2]) and chooses the one with the strongest signal strength. Once the AP is selected, it begins the association process followed by IP acquisition via DHCP. If the AP allocates an IP address to the client, it attempts to ping a known server to confirm connection to the Internet. Once Internet connectivity is verified, the Wi-Fi client begins either downloading or uploading data from/to the server via our simple socket program. After the client travels out of range of the AP, it detects the severed connection by monitoring the amount of data traversing the network interface. Once the client stops seeing packets for more than a configured time threshold, the current AP is abandoned and the search for another available AP begins. We choose 5 seconds in our experiments to allow ample time to make sure we do not attempt to reconnect to an AP that is at the trailing edge of the wireless range.

All experimental runs utilize a USB global positioning system (GPS) receiver that is plugged into the laptop capturing speed, location, and time once per second. The GPS device is also used to synchronize the time on the laptop. The laptop captures all data that is transmitted or received over the wireless interface with `tcpdump`.

2.3 3G Client

The 3G experiments employ an out of the box Apple iPhone 3G with no modifications to the hardware. The iPhone connects via the AT&T 3G network,

[2] An example entry is CMU's public Wi-Fi that is open but only allows registered MAC addresses to use the network.

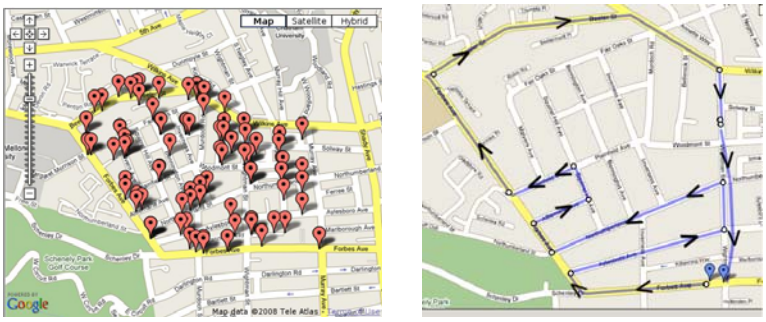

(a) (b)

Fig. 1. Maps of an area in Pittsburgh showing (a) all available open access points and (b) the route followed for the experiments

uses a jail-broken version of the firmware (2.2, 5G77), and its modem baseband firmware is at version 02.11.07.

The 3G client begins by first synchronizing its clock with NTP. Once the clock has been synchronized, it launches `tcpdump` to monitor the 3G wireless interface. After the monitoring has started, the client begins either downloading or uploading data. To download data, we use an open source command line tool for transferring files called `curl`. The `curl` program downloads a large file from the server and writes the output to `/dev/null` to avoid unnecessary CPU and battery consumption on the mobile device. This also allows us to isolate only network related effects. If the client is uploading data, the `dd` command continuously reads data out of `/dev/zero`. The output is piped into `netcat` and the data is streamed to the server.

2.4 The Experiment Route

The experiments are performed in a residential area of Pittsburgh, Pennsylvania near the campus of Carnegie Mellon University (CMU). Figure 1(a) is a map of the area where we focused our measurement collection. This area lies between the CMU campus and a nearby business district where many students frequently travel. Each red tag in the figure represents an open Wi-Fi AP found from our wireless scans[3]. The area is also covered by 3G service allowing us to compare the two access technologies. We believe this area to be representative of typical Wi-Fi densities found in most European or US urban areas[4].

Figure 1(b) shows the route selected in this area for our experiments. The experiment starts at the leftmost tag at the bottom right hand corner of the figure and follows the indicated route until the destination (same as start position) is reached. The total distance of the route is about 3.7 miles. For the walking experiments, we maintain a constant speed (2.4 MPH) throughout the course of

[3] Our scan logs reveal 511 APs in the area with 82 that appear open.

[4] http://wigle.net

the route. While driving, we obeyed all traffic laws and signs and remained as close to the speed limit (25 MPH) as possible.

3 Results

Table 1 summarizes the results of the experiments which are based on 16 runs from different days performed in the afternoon and late evening.

3.1 3G vs Wi-Fi Downloads

Figure 2 shows the instantaneous throughput achieved for a single, representative experiment for 3G and Wi-Fi at driving speeds of up to 30 MPH. The 3G

Table 1. 3G vs Opportunistic Wi-Fi

Radio	Speed	Data-flow	Usable contact time	Throughput	Total transfer
3G	driving	download	760 seconds	579.4 kbits/s	55 MB
Wi-Fi	driving	download	223 seconds	1220 kbits/s	34 MB
3G	walking	download	3385 seconds	673 kbits/s	285 MB
Wi-Fi	walking	download	1353 seconds	1243 kbits/s	210 MB
3G	driving	upload	866 seconds	130 kbits/s	14 MB
Wi-Fi	driving	upload	118 seconds	1345 kbits/s	20 MB
3G	walking	upload	3164 seconds	129 kbits/s	51 MB
Wi-Fi	walking	upload	860 seconds	1523 kbits/s	164 MB

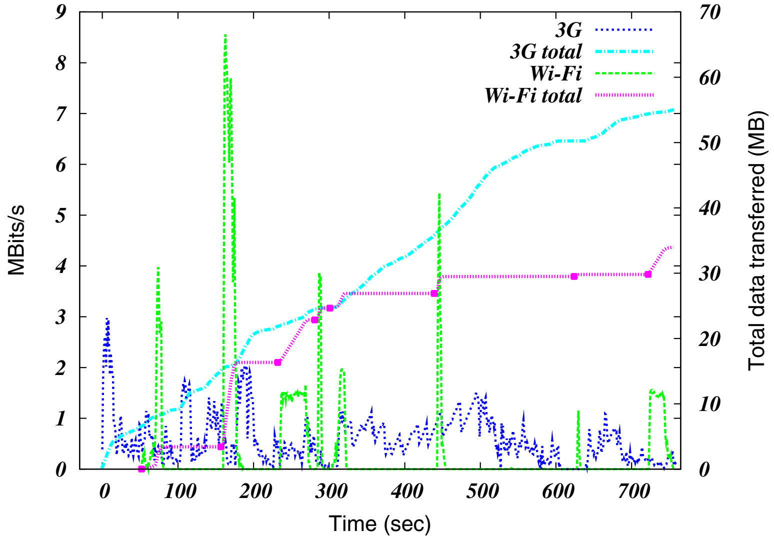


Fig. 2. Instantaneous throughput (Mbits/s) for 3G vs Wi-Fi downloads at driving speeds and total data transferred (MB)

device is able to transfer around 55 MB of data for the 760 seconds of the experiment duration. During this time, the Wi-Fi client connects opportunistically to APs along the route and manages to spend 223 seconds connected, transferring 34 MB. These "in the wild" results clearly show the potential of this untapped resource of open Wi-Fi connectivity and have a similar behavior to the isolated and controlled experiments in [5,6,14].

Deeper investigation of our logs shows that the majority of contacts were initiated while the client was either stopped, slowing down, or accelerating after a stop. This meant that the client stayed within the range of a single AP for longer durations and allowed more time to perform the steps needed to setup a connection and begin a data transfer. Since our AP selection algorithm was to always select the AP with the strongest signal, while moving, this was generally not the optimum choice. When the client approaches a potential AP, it would be best to select the AP that would be just coming into range to maximize the usable connection duration. We found that many connection attempts succeeded but when the data transfer was about to begin, the connection was severed. This does not mean that opportunistic contacts cannot happen at speeds, but instead brings to light the need for faster AP association and setup techniques similar to QuickWiFi[4] and better AP selection algorithms for mobile clients. Both of these would allow better exploitation of opportunistic transactions for in-motion scenarios.

Also plotted in Figure 2 is the total amount of data transferred for each access technology. The 3G connection is always connected throughout the entire experiment and shows a linear increase in the total bytes received. Wi-Fi, is represented by a step function which highlights how each connection opportunity benefits the overall amount of data received. Each point on the "Wi-Fi total" line represents a successful contact with an AP. Even though Wi-Fi contacts show large variability due to the intermittent nature of the contact opportunities, there is still a significant amount of data transferred because of the higher data rates of the technology. This is more apparent in the walking experiments where the speeds are much slower and the use of the sidewalks brings the client physically closer to the APs, allowing the client to remain connected for longer durations.

Figure 3 shows throughput results of a single, representative experiment for 3G and Wi-Fi at walking speeds. The walking experiments last around 3385 seconds and 3G is able to transfer around 285 MB of data. Wi-Fi, on the other hand, is only connected for 1353 seconds of the experiment and downloads 210 MB. Again, each Wi-Fi contact is able to exploit the opportunity and take advantage of very short, high throughput contacts, transferring significant amounts of data.

3.2 3G vs Wi-Fi Uploads

Figure 4 shows the instantaneous throughput of 3G and Wi-Fi uploads at walking speeds. It has a similar behavior to that of the downloads described previously but this time, the total data transferred for Wi-Fi exceeds 3G by 2.6 times. This is due to poor upload performance of 3G on the mobile device.

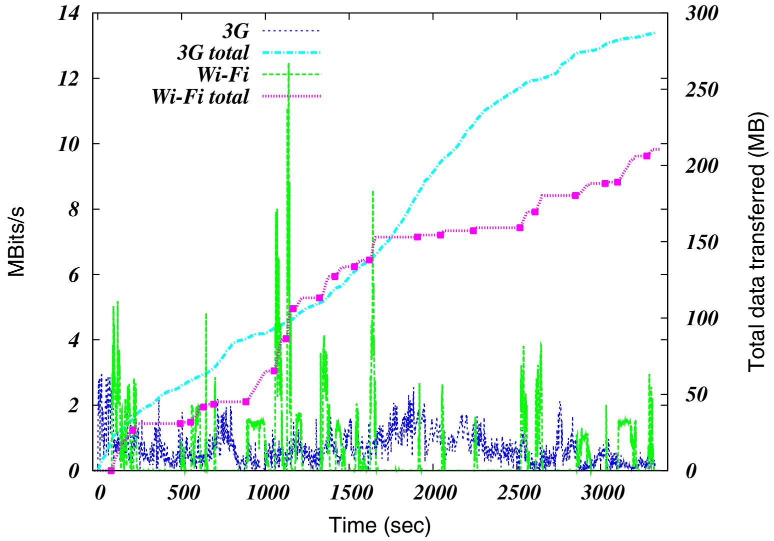


Fig. 3. Instantaneous throughput (Mbits/s) for 3G vs Wi-Fi downloads at walking speeds and total data transferred (MB)

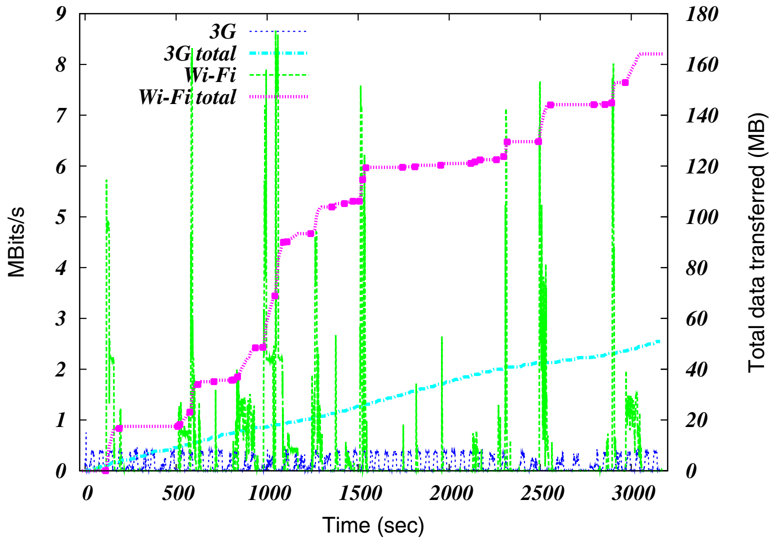


Fig. 4. Instantaneous throughput (Mbits/s) for 3G vs Wi-Fi uploads at walking speeds and total data transferred (MB)

The instantaneous 3G traffic pattern shows transitioning between idle states and periods of data transfers that result in throughput much less than that of the downloads (averaging at 130 kbits/s). In order to understand this phenomenon, we performed additional experiments with a stationary laptop (Lenovo T500 using the iPhone SIM card) and the iPhone with updated software and baseband firmware, 3.0 (7A341) and 04.26.08 respectively. We found that this periodic pattern is no longer evident. The new traces exhibit more consistent, albeit lower, throughput throughout the entire duration of an upload. The total amount of data transferred for a similar experiment did not change. We conjecture that these are due to improvements in the iPhone baseband software which allow more efficient buffering of data, eliminating the burstiness of the traffic egressing the device. Further upload experiments with the laptop show that it is able to transfer data at twice the rate of the iPhone. These results suggest hardware limitations on the iPhone and/or an artificial software limitation placed on the device[5].

One of the side observations from our experiments that impact the mobile client throughput is that residential Internet service rates are much higher than shown in [7]. Upon further investigation, we discovered that Verizon FIOS[6] has recently become available in this area and our experiments show that some homes have upgraded to this higher level of service. This is hopeful for utilizing opportunistic communications since more data can be transferred during these very short contact opportunities. It is also important to note that during these experiments, the full potential of the Wi-Fi AP was not reached and instead was limited to the rate of the back-haul link the AP was connected to. Even though the cost of higher throughput links are dropping in price for residential service plans, affordable service provider rates are still well below the available wireless rates of 802.11. This will always place the bottleneck for this type of communication at the back-haul link to the Internet[7].

4 Related Work

This work compares two dominant access technologies, namely 3G and Wi-Fi, in the wild. Despite many works related to the performance of 3G and Wi-Fi networks, this is the first work to publish a side-by-side comparison while in motion. This work highlights the potential of Wi-Fi as a contender for high throughput in-motion communication.

The performance of communicating with stationary access points has been studied in a variety of different scenarios. There have been experiments on a high speed Autobahn[14], in the Californian desert[5], and on an infrequently travelled road in Canada[6] where the environment and test parameters were

[5] http://www.networkperformancedaily.com/2008/06/3g_iphone_shows_bandwidth_limi.html

[6] http://www22.verizon.com

[7] http://www.dslreports.com/shownews/Average-Global-Download-Speed-15Mbps-101594

carefully controlled. These works showed that a significant amount of data can be transferred while moving by access points along the road.

The authors of [3] took this idea into the wild and reported on 290 drive-hours in urban environments and found the median connection duration to be 13 seconds. This finding is very promising for in-motion communications. This could potentially allow large amounts of data to be transferred over currently under-utilized links without the use of expensive 3G connections.

Previous work investigating performance of HSDPA (High Speed Data Packet Access), and CDMA 1x EV-DO (Code Division, Multiple Access, Evolution-Data Optimized) networks show similar findings with variability in these data networks[10,12,8]. We also see this behavior in our experiments run on a HSDPA network.

5 Conclusion

In this paper, we perform a comparison of two popular wireless access technologies, namely 3G and Wi-Fi. 3G provides continuous connectivity with low data rates and relatively high cost while Wi-Fi is intermittent with high bursts of data and comes for free when they are open. We experimentally show that with default AP selection techniques, off-the-shelf equipment, and no external antennae, we are able to opportunistically connect to open or community Wi-Fi APs (incurring no cost to the user) in an urban area and transfer significant amounts of data at walking and driving speeds. Intermittent Wi-Fi connectivity in an urban area can yield equivalent or greater throughput than what can be achieved using an "always-connected" 3G network.

Wi-Fi could be easily modified to increase the number of successful opportunities. (1) Reduce connection setup time with APs, especially with community networks like FON that have a lengthy authentication process. (2) Clients could take advantage of Wi-Fi maps and real time location updates in order to choose which APs will provide the most benefit to the in-motion user[13]. Finally, Wi-Fi is bottlenecked by the ISP link and (3) caching data on the AP (both for upload and download) would eliminate the Internet back-haul link bottleneck. We are currently testing an improved in-motion Wi-Fi architecture that exhibits significantly higher transfer rates than 3G at all speeds.

References

1. IEEE Standard 802.11: 1999(E), Wireless LAN Medium Access Control (MAC) and Physical Layer Specifications (August 1999)
2. IEEE 802.11n-2009, Wireless LAN Medium Access Control (MAC) and Physical Layer Specifications Enhancements for Higher Throughput (June 2009)
3. Bychkovsky, V., Hull, B., Miu, A., Balakrishnan, H., Madden, S.: A measurement study of vehicular internet access using in situ wi-fi networks. In: MobiCom 2006: Proceedings of the 12th annual international conference on Mobile computing and networking, Los Angeles, CA, USA, pp. 50–61. ACM Press, New York (2006)

4. Eriksson, J., Balakrishnan, H., Madden, S.: Cabernet: Vehicular content delivery using wifi. In: MobiCom 2008: Proceedings of the 14th ACM international conference on Mobile computing and networking, pp. 199–210 (2008)
5. Gass, R., Scott, J., Diot, C.: Measurements of in-motion 802.11 networking. In: WMCSA 2006 (HotMobile) Proceedings of the Seventh IEEE Workshop on Mobile Computing Systems & Applications, Semiahmoo Resort, Washington, USA, pp. 69–74. IEEE Computer Society, Los Alamitos (2006)
6. Hadaller, D., Keshav, S., Brecht, T., Agarwal, S.: Vehicular opportunistic communication under the microscope. In: MobiSys 2007: Proceedings of the 5th international conference on Mobile systems, applications and services, San Juan, Puerto Rico, pp. 206–219. ACM, New York (2007)
7. Han, D., Agarwala, A., Andersen, D.G., Kaminsky, M., Papagiannaki, K., Seshan, S.: Mark-and-sweep: Getting the "inside" scoop on neighborhood networks. In: IMC 2008: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement, Vouliagmeni, Greece. ACM, New York (2008)
8. Jang, K., Han, M., Cho, S., Ryu, H.-K., Lee, J., Lee, Y., Moon, S.: 3G and 3.5G wireless network performance measured from moving cars and high-speed trains. In: ACM Workshop on Mobile Internet through Cellular Networks: Operations, Challenges, and Solutions (MICNET), Beijing, China (October 2009)
9. Jones, K., Liu, L.: What where wi: An analysis of millions of wi-fi access points. In: Proceedings of 2007 IEEE Portable: International Conference on Portable Information Devices, May 2007, pp. 25–29 (2007)
10. Jurvansuu, M., Prokkola, J., Hanski, M., Perälä, P.H.J.: HSDPA performance in live networks. In: ICC, pp. 467–471 (2007)
11. Kozuch, M., Ryan, M., Gass, R., Scholsser, S., O'Hallaron, D., Cipar, J., Stroucken, M., Lopez, J., Ganger, G.: Tashi: Location-Aware Cluster Management. In: First Workshop on Automated Control for Datacenters and Clouds (ACDC 2009), Barcelona, Spain (June 2009)
12. Liu, X., Sridharan, A., Machiraju, S., Seshadri, M., Zang, H.: Experiences in a 3G network: Interplay between the wireless channel and applications. In: ACM MOBICOM, San Francisco, CA (September 2008)
13. Nicholson, A.J., Chawathe, Y., Chen, M.Y., Noble, B.D., Wetherall, D.: Improved access point selection. In: MobiSys 2006: Proceedings of the 4th international conference on Mobile systems, applications and services, pp. 233–245. ACM Press, New York (2006)
14. Ott, J., Kutscher, D.: Drive-thru internet: IEEE 802.11b for automobile users. In: INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, March 2004, vol. 1, p. 373. IEEE, Los Alamitos (2004)

Extracting Intra-domain Topology from `mrinfo` Probing

Jean-Jacques Pansiot[1], Pascal Mérindol[2], Benoit Donnet[2], and Olivier Bonaventure[2,⋆]

[1] Université de Strasbourg, Strasbourg, France
[2] Université catholique de Louvain, Louvain-la-Neuve, Belgium

Abstract. Active and passive measurements for topology discovery have known an impressive growth during the last decade. If a lot of work has been done regarding inter-domain topology discovery and modeling, only a few papers raise the question of how to extract intra-domain topologies from measurements results.

In this paper, based on a large dataset collected with `mrinfo`, a multicast tool that silently discovers all interfaces of a router, we provide a mechanism for retrieving intra-domain topologies. The main challenge is to assign an AS number to a border router whose IP addresses are not mapped to the same AS. Our algorithm is based on probabilistic and empirical IP allocation rules. The goal of our pool of rules is to converge to a consistent router to AS mapping. We show that our router-to-AS algorithm results in a mapping in more than 99% of the cases. Furthermore, with `mrinfo`, point-to-point links between routers can be distinguished from multiple links attached to a switch, providing an accurate view of the collected topologies. Finally, we provide a set of large intra-domain topologies in various formats.

1 Introduction

The *Internet topology* discovery has been an extensive subject of research during the past decade [1]. While topological information can be retrieved from passive monitoring (using, for instance, BGP dumps in the case of AS level topology), router level topology is usually obtained from active measurements based on *traceroute*.

Nevertheless, if traceroute has been largely deployed in the last few years, it comes with some important drawbacks. Traceroute provides a partial view of the network as it is routing dependent. For instance, backup links (high IGP weighted links for intra-domain and low BGP local preference links for inter-domain) are rarely captured by traceroute. Furthermore, the alias resolution problem is a complex issue to fix [2]. This leads thus to an incomplete and biased view of the

⋆ This work is partially funded by the European Commission funded Trilogy ICT-216372 project. B. Donnet's work is supported by the FNRS/FRS (Fonds National de la Recherche Scientifique, rue d'Egmont 5 – 1000 Bruxelles, Belgium.).

network. Obtaining complete intra-domain topologies is further a daunting task, requiring extensive probing campaigns [3].

Recently, we used `mrinfo` [4], a management multicast tool, in order to collect topology information [5]. `mrinfo` has the advantage of sweeping out many of traceroute's limitations as it is able to silently discover all interfaces of a router. However, it requires multicast being enable within ISPs' networks and no filtering policies, limiting so its applicability range. Indeed, only IPv4 multicast enabled routers reply to `mrinfo`. Also, some ISPs filter the IGMP messages used by `mrinfo` (i.e., they do not propagate them).

In this paper, we take advantage of the `mrinfo` dataset [6] for extracting intra-domain router level topologies. Obtaining real data concerning intra-domain topologies is of the highest importance. Indeed, it allows one to study actual network characteristics (e.g, degree distribution, network connectivity, ...) and to obtain insights on the way operators build their network. Furthermore, real topologies are crucial inputs for network simulations in order to consider complex and realistic scenarios. By modeling the collected topologies characteristics, it can also contribute to building better topology generators.

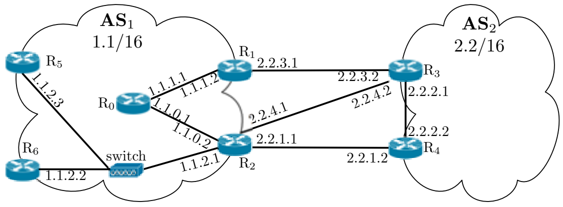
The contributions of this paper are twofold. We first describe how to extract intra-domain topologies from raw `mrinfo` data. While it is pretty easy to map IP addresses to an autonomous system number (ASN), the challenge is to mark the boundary of a given autonomous system (AS). Then, it is necessary to assign the right ASN to an AS border router (ASBR) whose IP addresses are not mapped to a single AS. In this paper, we provide an efficient algorithm, called *router-to-AS mapping*, for fixing this issue. We evaluate our algorithm and show that it provides a consistent mapping in more than 99.5% of the cases. In addition, an interesting feature of `mrinfo` is that point-to-point links between routers may be distinguished from multiple links attached to a switch. On average, we discover that roughly 11% of the nodes, in probed networks, are actually switches. As depicted in Sec. 3, this is a fundamental issue to correctly analyze network characteristics. Second, based on our router-to-AS mapping, we provide a set of intra-domain topologies under various formats. Our set of topologies is composed of three kind of networks: Tier-1 (such as Sprint), Transit networks (such as TDC), and Stub networks (such as UNINETT).[1] An extended version of this paper provides more results and discussions [7].

The remainder of this paper is organized as follows: Sec. 2 discusses how we collected topology data using `mrinfo`; Sec. 3 explains and evaluates our router-to-AS algorithm; Sec. 4 positions our work regarding the state of the art; Finally, Sec. 5 concludes this paper by summarizing its main achievements and discussing further works.

2 Collection Methodology and Dataset

`mrinfo` messages use the Internet Group Management Protocol (IGMP [8]). IGMP was initially designed to allow hosts to report their active multicast groups

[1] See `http://inl.info.ucl.ac.be/content/mrinfo`



```
1.1.0.2 [version 12.4]
    1.1.0.2 → 1.1.0.1 [1/0/pim/querier]
    1.1.2.1 → 1.1.2.2 [1/0/pim/querier]
    1.1.2.1 → 1.1.2.3 [1/0/pim/querier]
    2.2.4.1 → 2.2.4.2 [1/0/pim/querier]
    2.2.1.1 → 2.2.1.2 [1/0/pim/querier]
```

Fig. 1. `mrinfo` example with R$_2$ output

to a multicast router on their LAN. Upon reception of an IGMP `ASK_NEIGHBORS` message, an IPv4 multicast router replies with an IGMP `NEIGHBORS_REPLY` message providing the list of all its local interfaces with some information about their state. Fig. 1 shows an example of the usage of `mrinfo` to query the router R$_2$ (`1.1.0.2` is the responding interface of R$_2$). `mrinfo` reports that this router is directly connected to R$_0$ (through interface `1.1.0.1`) and two ASBRs, R$_3$ (through the interface `2.2.4.2`) and R$_4$ (through interface `2.2.1.2`). We can also notice that R$_2$ is connected to routers R$_5$ and R$_6$ through a switch because the interface `1.1.2.1` appears twice in R$_2$'s reply. This information is obtained by sending a single IGMP message. In practice, `mrinfo` provides similar information to a `show` command on the router's command line interface.

Based on `mrinfo`, we build a recursive probing scheme, `mrinfo-rec`, to scan connected components of networks. Initially, `mrinfo-rec` is fed with a single IP address corresponding to the first router attached to the `mrinfo-rec` vantage point. `mrinfo-rec` probes this router and recursively applies its probing mechanism on all the collected IP addresses. These recursive queries stop at unresponsive routers or when all known routers have been queried. The same process is run every day. It is worth noticing that a router not replying to an `mrinfo` probe during a given day is not queried again afterwards except if it appears again in a list of captured addresses.

To illustrate this behavior, let us apply it on the topology depicted in Fig. 1. `mrinfo-rec` receives, as input, the IP address of router R$_0$. From R$_0$, `mrinfo-rec` collects a set of neighbor IP addresses, i.e., {1.1.1.2, 1.1.0.2}. For all IP addresses in this set that were not previously probed, `mrinfo-rec` sends an IGMP `ASK_NEIGHBORS` message and, if the probed router replies, it again runs through the set of neighbor IP addresses collected.

Since May 1st, 2004, we have been collecting the `mrinfo` data from a host located in the University of Strasbourg, France. In this paper, we consider the data collected until the end of December 2008. The entire dataset is publicly available [6]. During this period, on average, `mrinfo-rec` was able to daily discover roughly 10,000 different routers while scanning 100,000 interfaces. Note that we remove interfaces with non-publicly routable IP addresses, i.e., the special-use IPv4 addresses described in RFC 3330. We also remove all tunnel and disabled interfaces. The IP-to-AS mapping is done using the last daily BGP table dump of the given day from the Routeviews dataset. We remove from our dataset IP addresses that cannot be mapped to an AS (0.5% on average) as well as those that are labeled to multiple origin ASes [9] (between 2 and 3% of IP addresses discovered each day by `mrinfo-rec`). We roughly identify between 400 and 650 different ASes every day of `mrinfo-rec` probing and we capture more than 850 ASes during the whole period of probing. Those ASes are distributed among Tier-1, Transit, and Stub networks, Transit being the most represented.

3 Router-to-AS Mapping

If it is easy to determine the ASN of a core router (each IP address of such a router is mapped to the same ASN[2]), the challenge is to accurately identify a router as an ASBR and assign it the right ASN. Fig. 2 and 3 illustrate the basics of our *router-to-AS* algorithm. The label attached on each link is the result of the IP-to-AS mapping (we assume that the two IP addresses on each directed link are necessarily mapped to the same AS). First, there are ASBRs whose IP addresses *do not all belong* to the same AS. In such a case, identifying them as ASBRs[4] is straightforward but assigning them an ASN is more difficult. This situation is illustrated in Fig. 2 where router R_1 has two interfaces mapped to AS_1 while the remaining two interfaces are mapped to AS_2. This *Shared Addressing Space* case must be solved to perform the router-to-AS mapping. As soon as all routers are mapped to their right ASN, it is possible to extract intra-domain topologies without falsely cutting between ASes. We denote $\mathbb{SAS}$ the subset of ASBRs falling into the Shared Addressing Space case.

Second, there are ASBRs whose all IP addresses are mapped to the same ASN. If the router-to-AS mapping is obvious, identifying them as ASBRs is a different ball game: their detection essentially relies on their relationships with ASBRs belonging to $\mathbb{SAS}$. This is illustrated in Fig. 2 with routers R_4 and R_5 as all their interfaces are mapped to AS_2. If R_1 is correctly assigned to AS_1, then R_4 and R_5 are ASBRs mapped to the AS corresponding to the address space of links $R_1 \leftrightarrow R_4$ and $R_1 \leftrightarrow R_5$. This issue is thus trivial if the $\mathbb{SAS}$ case has been previously correctly solved.

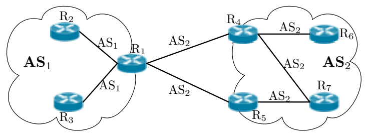
At this point, it is already worth noticing (see Sec. 3.2 for further details) that the vast majority (almost 90%) of routers are directly mapped to the right ASN because they do not belong to the $\mathbb{SAS}$ set. Thus, our router-to-AS algorithm is applied to only 10% of routers ($\sim$ 1,000 routers on average per day).

3.1 Router-to-AS Algorithm

Our router-to-AS algorithm is based on two families of rules: *probabilistic* and *empirical* rules. The main idea behind our algorithm is to quickly converge to a single and consistent mapping for each router. For that purpose, our algorithm verifies the consistency of the results returned by each rule.

We start by assigning a candidate ASN to any router. This is done using our first probabilistic rule (called *global election*, or *elec*). It works as follows: each

[2] Note that there exists specific cases more difficult to solve (see [7]).



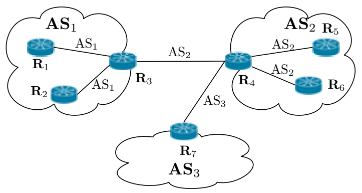


Fig. 2. Shared Addressing Space case on R_1 **Fig. 3.** Neighborhood empirical rule, N

router is mapped to the ASN assigned to the largest number of its IP addresses (with an IP apparition order for tie-breaking equality cases). Let S_r be the set of the occurrence of each IP-to-AS mapping computed on addresses belonging to the router r. If r is initially mapped to AS n, it means that n appears $max(S_r)$ times in the IP-to-AS mapping of r. We attribute a confidence level to the ASN mapped to r in such a way: $c(r) = 1 - \frac{max(S_r \setminus \{max(S_r)\})}{max(S_r)}$. Closer to one, higher the confidence in the mapping. Note that $r \in \mathbb{SAS}$ if $c(r) < 1$. For instance, regarding the link between R_3 and R_4 in Fig. 3, it means that IP addresses involved in both directions of the link $R_3 \leftrightarrow R_4$ belong to AS_2. Then, looking at Fig. 2, $c(R_4) = 1$ whereas $c(R_1) = 0$, we know that R_1 is an ASBR because it belongs to the set $\mathbb{SAS}$, but we have to figure out whether it belongs to AS_1 or to AS_2 and respectively whether R_2, R_3 or R_4, R_5 are ASBRs. In contrast, in Fig. 3, $c(R_3) = 0.5$ meaning that R_3 seems to belong to AS_1. The *elec* rule is the primary block of all our analysis: other rules aim at confirming or disproving it when $c < 1$.

The second probabilistic rule relies on the detection of LAN interfaces. The *lan* rule concerns a subset of $x \rightarrow 0.0.0.0$ interfaces. Those 0.0.0.0 interfaces usually describe leaf LAN without transit and multicast capabilities (more details are given in [7]). We assume that intra-domain LANs are more frequent than inter-domain LANs, and considering that a local LAN interface uses the address space of the internal domain, a router has a higher probability of belonging to the AS assigned to LAN interfaces. Note that we only consider cases where all LAN interfaces are mapped to the same AS. Furthermore, we do not take into account $x \rightarrow 0.0.0.0$ interfaces if the AS to which IP x belongs is not multicast enabled (e.g., an AS where `mrinfo-rec` does not obtain replies). We observe that, probabilistically talking and, on average, $x \rightarrow 0.0.0.0$ interfaces produce fewer *Shared Addressing Space* cases than *elec*: fewer than 4% compared to the 10% produced by the global *elec* rule. This observation reinforces our hypothesis on intra-domain LAN detection.

As already mentioned, we also use empirical rules consisting in a set of usual rules. The first empirical rule takes into account the loopback interface of a router (*lb* rule). When configuring a loopback interface, an ISP uses an address belonging to its own IP address space. Thus, identifying loopback interfaces using their DNS name and performing a standard IP-to-AS mapping on this address resolves the router-to-AS mapping.

We also use an empirical rule consisting of a neighborhood analysis (N rule): we assume that inter-domain links are mapped to the address space of one of the ASes it interconnects. This rule can be applied when a $\mathbb{SAS}$ is mapped to a given AS thanks to another rule. For instance, in Fig. 3, let us assume that R_3 has been mapped to AS_1 with the *lb* rule, then R_4 necessarily belongs to AS_2 and iteratively R_7 is then mapped to AS_3. In practice, we apply this rule iteratively until no more new AS assignment is recorded and use it at each step of the router-to-AS algorithm. We also apply this rule iteratively during the last steps of the router-to-AS algorithm.

As already mentioned by claffy et al. [10], a provider generally allocates IP addresses from its own address space to its customers links (*c2p* rule). In a simple case, this means that if two routers denoted R_1 and R_2 are connected through an inter-domain link mapped to AS_x, and such that R_1 also uses the address space allocated by another AS_y ($y \neq x$) while AS_y is a customer of the AS_x, then R_1 is mapped to AS_y (and R_2 to AS_x). In Fig 2, if we know that AS_1 is a customer of AS_2, the c2p rule allows us to map R_1 to AS_1. To perform such a relationship mapping, we use the AS ranking data set provided by Caida [11]. Note that this rule seems relatively consistent with the *elec* rule: on average, in more than 70% of the cases, this rule is verified when focusing on routers with a confidence level superior to 0.5. This is our penultimate rule, so that it tie-breaks remaining equality cases when the rule can be used (c2p or p2c relationships between involved ASes).

Finally, to perform the router-to-AS mapping we need a global order between our pool of rules to characterize the confidence we attribute to each of them. We use the following order: $elec > lb > N_0 > lan > N_1 > H_{0.9} > N_2 > H_{0.8} > N_3 > \ldots > N_{10} > c2p > N_{11}$ where H_β stands for a β-confident assignment rule. According to the confidence threshold $0 < \beta < 1$, if $c(r) > \beta$ (for a given router $r \in \mathbb{SAS}$ mapped to the ASN n with the *elec* rule), H_β maps definitively r to n by attributing to r a confidence level of 1. In order to take advantage of AS assignments produced by the decreasing level of confidence of our set of rules, we apply the neighboring rule between each other rules' application.

Moreover, we use a threshold $0 < \alpha < 1$ to decide whether other rules can overwrite the candidate assignment (the result of *elec*). For all routers $r \in \mathbb{SAS}$, if a given rule is not in concordance with the *elec* rule (i.e., the ASN returned by the given rule differs from the candidate one given by *elec*), we select the ASN returned by the tested rule only if $c(r) \leq \alpha$. Otherwise we ignore the result provided by the tested rule. In practice we choose $\alpha = 0.5$[3]. Sec. 3.2 describes the consistency of the mapping we obtain using this ordered set of rules.

3.2 Evaluation

From our four years daily dataset, we arbitrarily select the largest `mrinfo` raw data file of each month, leading to 56 files. We then evaluate our router-to-AS algorithm on those files.

[3] It means that there are at least two more IP addresses mapped to the candidate ASN compared to any other ASN.

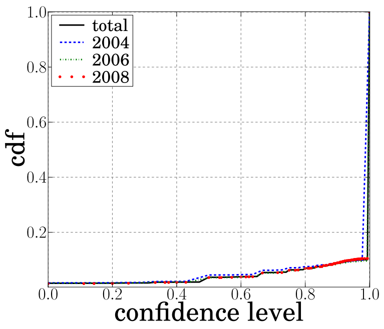


Fig. 4. *elec* rule efficiency

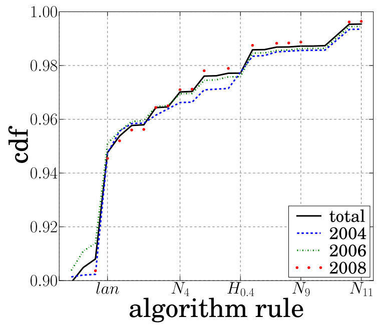


Fig. 5. Algorithm convergence time

Fig. 4 provides the cumulative distribution of the confidence level c (the horizontal axis) assigned during the first step (*elec*) of the router-to-AS mapping. The first observation is that, on average, 90% of the routers have addresses mapped to a single AS. We identify roughly that only 1.5% of the routers have a confidence level equal to 0, whereas more than 95% of routers have a confidence level superior to 0.5. According to our threshold $\alpha = 0.5$, only 5% of the router assignments are really problematic.

Fig. 5 shows the cumulative distribution of the required number of rules (the horizontal axis) to converge to a positive decision, i.e., $c = 1$, in the router-to-AS algorithm. In this figure, we observe that the decision process quickly converges: more than half of $\mathbb{SAS}$ cases are already treated after the *lan* rule application (i.e., after the 4th rule). This means that a large subset of the critical cases are treated at the beginning of our set of rules which are ordered depending on the confidence we attribute to each of them. At the end of the process, using the N_{11} rule, we can notice that fewer than 0.46% of the routers remain unmapped.

Fig. 6 gives a more detailed overview of the actions taken during the convergence of our algorithm. Each point represents the mean over the 56 files we consider. We determine 95% confidence intervals for the mean but intervals are typically too tight to appear on Fig. 6.

We can see that in most cases, our pool of rules confirms the candidate assignment of the global election, *elec*: at the end of the process, 88.9% of the candidate assignment are confirmed with one or another rule. We also count the number of contradictions produced by our set of rules against *elec*, and divide them in two categories according to the threshold α that we use. We notice that on average, 12% of the AS assignments are concerned, and mainly when $c > 0.5$ (6.4% compared to 5.6% when $c \leq 0.5$). However, using our threshold $\alpha = 0.5$, we only effectively record the 5.6% of changes that are not strongly inconsistent ($c \leq 0.5$) to stay consistent with the *elec* rule. We have also noticed that less than 6% of routers among the $\mathbb{SAS}$ set are subject to real inconsistencies (see [7]).

To summarize, we have seen that 90% of the routers are assigned, directly at the first step of the algorithm, to an ASN with the highest level of confidence. The remaining 10% of routers belongs to $\mathbb{SAS}$ and represent thus critical cases.

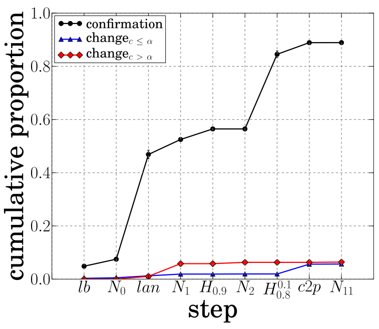


Fig. 6. A closer look at each step

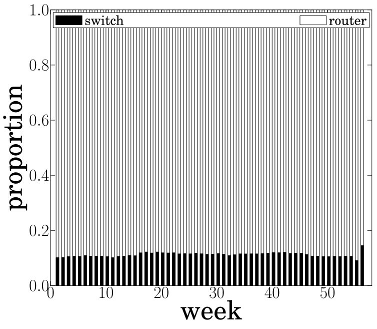


Fig. 7. Switches and routers proportion

Our algorithm is able to quickly solve a large subset of those cases. At the end of the process, only 0.46% of the routers remain unmapped.

A more detailed discussion on the algorithm evaluation as well as on particular cases may be found in [7]. In particular, we have empirically verified that the AS where our probing host is located (AS2259) is correctly and fully discovered by our algorithm.

3.3 Point to Point Links and Switches

In addition to our router-to-AS algorithm, we also provide a way to distinguish point-to-point links from switch inter-connections. As previously mentioned (see Fig. 1), replies collected with `mrinfo-rec` allow us to easily discover switch pseudo-nodes and extract them from our raw data.

This point is of the highest importance since it provides accurate information on the real network connectivity. Using traceroute-like probing, switches are not easily detectable and this bias leads to produce false interpretations: a set of nodes may appear to be fully meshed whereas they are actually connected through a simple switch. Identifying switches in `mrinfo` output is straightforward as it is enough to capture outgoing IP addresses appearing several times on the same router (see interface `1.1.2.1` on router R_2 in Fig. 1).

Note that a switch inter-connection discovered with `mrinfo-rec` can hide a switch cascade, i.e., several switches might be connected together. It can also hide some other types of level 2 inter-connections. Moreover, when possible, we verify that all routers connected through a switch share the same vision of the inter-connection (e.g., one IP address pointing towards the same set of addresses).

Fig. 7 provides the distribution of switches and routers over the 56 weeks. On average, we identify that 11% of inter-connection points discovered in the networks are switches (or cascade of switches), while the remaining 89% are actual routers. Note that the same distribution occurs when we distinguish inter-domain from intra-domain connections. Only 1% of the whole set of discovered nodes are inter-domain switches (Internet exchange points, IXPs) whereas 9% of them are ASBR. Note that we do not apply the neighboring rule N for IXPs.

4 Related Work

A tool like *Rocketfuel* [3] has been used to infer ISP topologies. However, inferring topologies in a non-cooperative and heterogeneous environment has proven to be extremely difficult, and results obtained have to be carefully evaluated in terms of validity [12,13]. The recently introduced *DisCarte* [14] pushes the accuracy of collected data a few steps further but it requires the "record route" option being enable and does not entirely sweep out standard traceroute limitations.

Mao et al. provide mechanisms for improving the IP-to-AS mapping [15,16]. Their techniques are based on several information sources: traceroute, BGP update, BGP table dumps, and reverse DNS lookup. In addition, they propose heuristics for identifying IXPs, sibling ASes as well as ASes sharing address space. Their work differs from ours as they focus only on IP-to-AS mapping and not on router-to-AS mapping.

The recent work done by claffy et al. is probably the most relevant compared to this paper [10]. For assigning ASes to routers, claffy et al. assume that a provider always gives IP addresses belonging to its own address space for connections to their customers [10]. Given that assumption, the router-to-AS mapping becomes straightforward when focusing on customer-to-provider links (and reciprocally). Otherwise, the router is assigned to the AS with the smallest outdegree. Note that no evaluation of this technique has yet been made in [10].

5 Conclusion

We provide a mechanism for extracting intra-domain topologies from raw data collected by `mrinfo`, a multicast based tool that is able to silently discover all interfaces of a router. The main challenge is to mark the boundaries of each AS. The goal of our algorithm is to assign an AS number to a router, performing the so called *router-to-AS mapping*. We demonstrate that our router-to-AS mapping is able to efficiently assign an AS number to a router with a high confidence level. In addition, our AS extraction mechanism is able to discover connections through layer-2 switches, providing a more accurate view of the topology than with traceroute probing. Finally, we provide, in various format, several intra-domain topologies for Tier-1, Transit, and Stub networks all along the four years of collected data.

We believe the technique described in this paper as well as the whole `mrinfo` dataset are valuable for the research community. Indeed, the next steps of this work would be to deeply study intra-domain topologies and improve `mrinfo` based probing using complementary topology discovery methods.

References

1. Donnet, B., Friedman, T.: Internet topology discovery: a survey. IEEE Communications Surveys and Tutorials 9(4), 2–15 (2007)
2. Gunes, M.H., Sarac, K.: Importance of IP alias resolution in sampling Internet topologies. In: Proc. IEEE Global Internet Symposium (May 2007)
3. Spring, N., Mahajan, R., Wetherall, D.: Measuring ISP topologies with Rocketfuel. In: Proc. ACM SIGCOMM (August 2002)
4. Jacobson, V.: Mrinfo (1995), http://cvsweb.netbsd.org/bsdweb.cgi/src/usr.sbin/mrinfo/?only_with_tag=MAIN
5. Mérindol, P., Van den Schriek, V., Donnet, B., Bonaventure, O., Pansiot, J.J.: Quantifying ASes multiconnectivity using multicast information. In: Proc. ACM USENIX Internet Measurement Conference (IMC) (November 2009)
6. Pansiot, J.J.: Mrinfo dataset, http://svnet.u-strasbg.fr/mrinfo/
7. Pansiot, J., Mérindol, P., Donnet, B., Bonaventure, O.: Internet topology discovery through mrinfo probing. TR 2009-01, Université catholique de Louvain (UCL), (October 2009), http://inl.info.ucl.ac.be/content/mrinfo
8. Deering, S.: Host extensions for IP multicasting. In: RFC 1112, Internet Engineering Task Force (August 1989)
9. Zhao, X., Pei, D., Wang, L., Massey, D., Mankin, A., Wu, S.F., Zhang, L.: An analysis of BGP multiple origin AS (MOAS) conflicts. In: Proc. ACM SIGCOMM Internet Measurement Workshop (IMW) (October 2001)
10. Claffy, K., Hyun, Y., Keys, K., Fomenkov, M., Krioukov, D.: Internet mapping: from art to science. In: Proc. IEEE Cybersecurity Applications and Technologies Conference for Homeland Security CATCH (March 2009)
11. CAIDA: AS relationships (2009), http://www.caida.org/data/active/as-relationships/index.xml
12. Zhang, M., Ruan, Y., Pai, V., Rexford, J.: How DNS misnaming distorts internet topology mapping. In: Proc. USENIX Annual Technical Conference (May/June 2006)
13. Teixeira, R., Marzullo, K., Savage, S., Voelker, G.: In search of path diversity in ISP networks. In: Proc. ACM SIGCOMM Internet Measurement Conference (IMC) (October 2003)
14. Sherwood, R., Bender, A., Spring, N.: DisCarte: A disjunctive Internet cartographer. In: Proc. ACM SIGCOMM (August 2008)
15. Mao, Z.M., Rexford, J., Wang, J., Katz, R.H.: Towards an accurate AS-level traceroute tool. In: Proc. ACM SIGCOMM (August 2003)
16. Mao, Z., Johnson, D., Rexford, J., Wang, J., Katz, R.: Scalable and accurate identification of AS-level forwarding paths. In: Proc. IEEE INFOCOM (April 2004)

Quantifying the Pitfalls of Traceroute in AS Connectivity Inference

Yu Zhang[1], Ricardo Oliveira[2], Hongli Zhang[1], and Lixia Zhang[2,⋆]

[1] Harbin Institute of Technology, Harbin, 150001, China
{yuzhang,zhanghongli}@hit.edu.cn
[2] University of California, Los Angels, CA 90024, USA
{rveloso,lixia}@cs.ucla.edu

Abstract. Although traceroute has the potential to discover AS links that are invisible to existing BGP monitors, it is well known that the common approach for mapping router IP address to AS number (IP2AS) based on the *longest prefix matching* is highly error-prone. In this paper we conduct a systematic investigation into the potential errors of the IP2AS mapping for AS topology inference. In comparing traceroute-derived AS paths and BGP AS paths, we take a novel approach of identifying mismatch fragments between each path pair. We then identify the origin and cause of each mismatch with a systematic set of tests based on publicly available data sets. Our results show that about 60% of mismatches are due to IP address sharing between peering BGP routers in neighboring ASes, and only about 14% of the mismatches are caused by the presence of IXPs, siblings, or prefixes with multiple origin ASes. This result helps clarify an argument that comes from previous work regarding the major cause of errors in converting traceroute paths to AS paths. Our results also show that between 16% and 47% of AS adjacencies in two public repositories for traceroute-derived topology are false.

Keywords: AS topology measurement, traceroute, BGP.

1 Introduction

The Internet is a vast distributed system formed by a myriad of networks called Autonomous Systems (ASes) that exchange routing information using the Border Gateway Protocol (BGP). There have been two basic approaches to measuring AS-level connectivity: (1) passive measurement through collecting BGP routing updates, and (2) active measurement using traceroute. In the BGP-based measurement, AS adjacencies can be directly extracted from the ASPATH attribute in BGP updates collected from the monitors/routers by Routeviews [4] and RIPE-RIS [3]. But because of policy filters and best path selection, each BGP monitor only provides a limited partial view of the topology. Most monitors in

⋆ This work was partially supported by the National Basic Research Program of China (973 Program) under grant No.2005CB321806 and by the US National Science Foundation under Contract No CNS-0551736.

traceroute measurement projects, such as CAIDA's Ark [1] and DIMES [16], are placed in different ASes than BGP monitors, thus ideally they can complement the topology inferred from existing BGP sets. It is also easier to deploy a traceroute monitor than to obtain a new BGP feed [16].

However converting the router IP addresses on traceroute paths to AS numbers, termed *IP2AS mapping*, is a difficult problem. Typically this conversion is done by finding the origin AS of each IP address in the traceroute path from the BGP routing table using *longest prefix matching* (LPM). Unfortunately this approach is known to generate potentially false AS links, and the following question emerges: *what's the impact of inference errors of traceroute-derived AS paths on the AS topology map when using LPM?*

Several previous efforts have studied the problem of traceroute-derived measurement and articulated possible causes for the mismatch between the traceroute-derived path and the BGP path [7,13,9,10,12]. However, these previous efforts did not provide answers to our question because of the following reasons: (1) They quantified mismatch causes in the unit of *path*, *e.g.* either there was a match in the converted path or not, which does not pin down all individual points on the topology that the two paths differ; and (2) They did not investigate the accuracy of traceroute-derived topology.

In this paper we conduct a systematic and exhaustive investigation into the impact of pitfalls of LPM-based traceroute measurement on topology inference. Our contributions can be summarized as follows. (1) We identify differences in pairs of traceroute and BGP paths systemically. This allows us to pinpoint multiple mismatches in the same AS path pair and to identify each mismatch point shared by multiple path pairs. (2) We collect a comprehensive set of publicly available data and develop a set of tests to infer the cause of each mismatch more systematically than before. (3) Our results show that about 60% of mismatches occur because of IP address sharing between neighbor routers. This result is a departure from previous work [13,10] that attributed the causes of errors mainly to Internet eXchange Points (IXPs), sibling ASes under the same ownership, and prefixes originated from multiple origin ASes. (4) We find that between 16% and 47% of the traceroute-derived adjacencies in public data sets widely used by the community may be bogus.

2 BGP vs. Traceroute

Generally speaking, the data path inferred from traceroute and BGP control path should match. There are however some scenarios where the two paths differ, either because the data path is not completely aligned with the control path, or because of IP2AS shortfalls in converting IP addresses to AS numbers. We describe different reasons why the BGP AS paths may differ from the AS paths measured by LPM-based traceroute method.

(1) There may be divergence between data path and control path due to BGP aggregation, multi-hop sessions, tunneling, layer-2 switching, and abnormal routing. (2) The traceroute path may be incomplete because of non-responsive

hops. In addition, the BGP routing tables may not tell exactly the original ASN of a given prefix, *e.g.* (3) an unannounced prefix or (4) a Multiple Origin ASes (MOAS) prefix.

The IP addresses announced by a given AS X may be used by another AS Y. We call those addresses the *foreign addresses* of Y. (5) A typical case is that one prefix is shared by multiple participants in the IXP, which is a shared infrastructure where multiple networks peer with each other publicly. (6) ASes under the same ownership, *i.e.* siblings, may also share the same IP address space. (7) Another typical case is *IP address sharing between neighbor ASes*, where a border router owned by AS Y replies to a traceroute using one of its interfaces whose IP address is borrowed from the neighboring AS X to enable the point-to-point connection. For example, when Y has a private peering with X, two incident routers' interfaces are typically numbered from a /30. If the /30 is coming from X, then routers at Y may reply with X's address range.

According to our measurements, 63∼88% of path pairs had a match (no extra links in traceroute path). In the remaining cases, at most 3.7% of mismatch path pairs are originated by divergence of control path and traceroute path, and for the rest we provide evidence for their occurrence due to errors in IP2AS mapping. Therefore, we believe it makes sense to use BGP paths as the reference by default as [13,10,12]. If in the vast majority of cases the data path would not be align with the control path (or BGP), there would be a significant number of mismatch cases we could not explain, which is not the case.

3 Related Work

Measuring Internet AS-level connectivity from traceroute data has attracted many research efforts over recent years. One of the first such studies was done by Chang *et al.* [7], which alerted for possible errors in AS topology inferred from traceroute data using the LPM approach. They presented a technique to identify the ownership of border routers based on IP alias resolution, and presented some heuristics to fill the holes of unmapped hops in traceroute paths. This work was probably the first that pointed out potential errors in traceroute-derived AS paths because of IP address sharing between neighbor ASes.

In a later work, aiming at an accurate AS-level traceroute tool, Mao *et al.* [13] compared BGP paths with traceroute paths launched from the same AS where the BGP table was extracted. They investigated a comprehensive set of possible causes of mismatch and developed heuristics to correct the IP2AS mapping. In a following work [12], they presented a dynamic programming algorithm to reassign /24 prefixes to ASes to minimize the number of mismatched path pairs. The main outcome of this work was a method to correct the mismatches due to unmapped hops, MOAS prefixes, IXPs and siblings.

At the same time, Hyun *et al.* [10] presented a path pair comparison and quantified the mismatched pairs due to IXPs and siblings. They adopted the algorithm for the longest common subsequence (LCS) problem to describe the pattern of unexplained mismatches. Later[9], they presented the concept of *third-party addresses*, but its definition does not clearly address the issue of IP address sharing between BGP neighbors.

As far as we can tell, our paper is the first to propose a systematic method of identifying mismatches between each traceroute-derived path and BGP path pair. This method allows us to pinpoint multiple mismatch points in the same AS path pair and align mismatched portions of a pair of paths, giving local context to the comparison and explaining the cause of the mismatches. Our result asserts that the main cause of mismatch is the IP address sharing between neighbor ASes in accordance with [7] and departing from [13,10] that attributed the mismatches to the presence of IXPs, siblings and MOAS. Note that we do not use the LCS algorithm to describe the unexplained mismatches (as [10]), but we enhanced it to identify the mismatches.

4 Data Sets

4.1 AS Path Pair Data

We collect traceroute raw data and the corresponding BGP routing updates from 4 ASes. Table 1 lists the number of destination IP addresses and prefixes probed by the traceroute vantage points, as well the corresponding BGP information.

UCLA: From a host located at UCLA, we performed probes targeting all /24 blocks in the BGP routing table, using the traceroute tool *scamper* (`http://www.wand.net.nz/scamper/`) with ICMP-paris [6]. At the same time, we collected BGP updates and tables from a backbone router at UCLA.

CAIDA Ark: There are 3 CAIDA Ark monitors that happen to be located in ASes which provide a BGP feed to either RouteViews or RIPE-RIS collectors. For each /24 block, the latest traceroute result is picked.

The traceroute AS paths are generated by the LPM-based IP2AS mapping on the BGP routing table of the AS where the traceroute is launched from. Since there is no guarantee that BGP routers will have consistent tables inside a large AS (*e.g.* different routers in same AS can have different tables), we only collect the path pairs where the next-hop AS is the same in order to reduce ambiguities. A traceroute path is paired with its corresponding BGP path to the same prefix, only if there is no change observed in the local BGP route to the destination prefix during the traceroute probe, otherwise the paths are discarded.

Table 1. Information of AS path pair data sources

Monitor	ASN	#pair	#prefix	Collector	Orgnization	Date
ucla	52	7.6M	272K	ucla	UCLA	2009-02-22∼03-10
ams-nl	1103	5.2M	218K	ris-rrc03	SURFnet	2009-02-01∼03-12
nrl-jp	7660	4.9M	212K	rv2-oix	APAN	2009-02-01∼03-12
she-cn	4538	5.2M	218K	rv-wide	CERNET	2009-02-01∼03-12

4.2 AS Adjacencies

To evaluate the accuracy of traceroute-derived AS adjacencies, we collect data from CAIDA Ark, DIMES, and UCLA IRL [5]. The data from IRL is also used to explain the mismatch with the assistance of data from Internet Routing Registry (IRR) [2] and iPlane [11].

CAIDA Ark: Two traceroute-derived AS topologies are obtained by merging all snapshots in Feb. 2009 [1]: (1) The topology with only direct links, in which every consecutive pair of ASes have a pair of contiguous hops in the traceroute path; and (2) The topology with both direct links and indirect links, in which two IP addresses in different ASes may be separated by one or more unmapped or non-responsive hops.

DIMES: We also collect DIMES' monthly traceroute-derived AS topology in Feb. 2009 [16]. This graph include the AS links which are observed at least once in the given month and at least twice considering all period.

BGP: We use the BGP-derived AS adjacencies available at UCLA IRL [5], which is extracted from RouteViews and RIPE-RIS. For the sake of completeness, the data is accumulated over a period of 5 months ending at March 2009, following the methodology in [15].

IRR: The Internet Routing Registry (IRR) [2] is a central repository where ISPs explicitly insert information such as routing policies and BGP adjacencies. We are able to extract 28,700 total AS numbers and 156,094 total AS adjacencies from all available IRR databases as of 2009-03-05.

iPlane: iPlane [11] project provides a list of routers' alias, *i.e.* a set of interface IP addresses belonging to the same router. This information can be used to explain mismatches due to IP address sharing between BGP neighbors, since we can look up each interface alias in BGP tables and estimate which ASes have BGP sessions in a same router. We extracted a total of 286,043 IP interface addresses on 67,430 routers on 2009-03-05.

4.3 IXP and Sibling Lists

To help identify ASNs used by IXPs and ASes with sibling relationship, we extracted the name/description of each AS from all WHOIS databases.

IXPs: We compiled a list of 404 /24 prefixes belonging to IXPs by crawling three websites, peeringDB.com, PCH.net and euro-ix.net, on 2009-03-09. Additionally, we search a list of ASNs associated with IXP names (from the previous websites) and the common words "internet exchange", "exchange point", "access point" and "gigapop", carefully filtering the false IXP records, *e.g.* a description "peering at an IXP". We end up with a total of 323 ASNs belonging to IXPs.

Siblings: We look for similarities in AS names/descriptions of a given pair of ASes using approximate string matching except in the cases where the name is a word appearing in an English dictionary. The acquisition history of all Tier-1

ISPs from wikipedia is also used to group ASes. After computing the transitive closure of sibling relationships and cleaning up the candidate sibling groups with size greater than 20 manually, we get 3,490 sibling groups with 13,639 ASes.

5 Mismatch Analysis: Breaking Paths into Fragments

In this section we develop a technique for comparing BGP paths and traceroute-derived AS paths obtained in Sec. 4.1. We use the classic file comparison command `diff`-like method to find the *longest common subsequence* (LCS) that is present in both traceroute path and BGP path [8]. The LCS solution is described as a minimum array of binary operations needed to transform the BGP AS path into the traceroute AS path: insertion '+', deletion '−', or unmodified '='. The consecutive '=' operations represent the common segments, while the '+' and '-' operations indicate the difference.

To pinpoint multiple mismatches in the same AS path pair and describe a mismatch in its local context, we define a *mismatch fragment* between two AS paths as a sequence of '−' and/or '+' wrapped around by two '='. For example, Figure 1(a) shows the solution (F1) and mismatch fragments (F2) of a one-to-one substitution case. Note that the same mismatch fragment at the AS level may have the different IP-level fragment. We develop five additional steps to detect the mismatch fragment systematically:

1. We add 4 special tokens to the traceroute AS path: '∗' representing consecutive non-responsive hops, '?' representing consecutive unmapped hops; ' ∧' and '$' represent the beginning and end of a path, respectively. See examples in Figure 1(b) and (c).
2. When there are multiple alternative solutions with the same number of operations, the one whose '=' operator appears earlier in the BGP path is picked. The goal of this tie-break is to concentrate the errors in the least number of original hops as possible. In Figure 1(d), F1 is picked from two solutions.
3. The mismatch fragment is replaced with its inside loop, since loops describe differences more properly as shown in Figure 1(e).
4. Among more than one '−' operations in the substitution at the end of path, only the first is kept, *e.g.* F1 is replaced with F2 in Figure 1(f).
5. Mismatch fragments whose modifying operations include only '+∗' or '+?', or only deletions ('−') at the end of path are discarded, because our interest is in extra links brought by traceroute.

We obtained a total of 39K unique mismatch fragments (15∼20K per monitor) including 44% *extra* ('+'-only), 20% *missing* ('−'-only), and 36% *substitute*

	(a) substitute	(b) end-extra	(c) non-responsive	(d) tie-break	(e) loop	(f) end-substitute
BGP:	A B C D	A B	A B C	A B C D	A B	A B C
Traceroute:	A E C D	A B C	A ∗ C	A C B D	A C A B	A D
F1:	=A -B +E =C =D	=B +C =$	=A -B +∗ =C	=A +C =B,=B -C =D	=A +C +A =B	=A -B -C +D =$
F2:	=A -B +E =C			=A -B =C,=C +B =D	=A +C =A	=A -B +D =$

Fig. 1. Examples of AS path pairs and their mismatch fragments

(both '−' and '+') patterns. Among the *extra* mismatch fragments, 39% are *loops*. Overall, there are 12∼37% of path pairs containing one or more mismatch fragments. And we also observe that the appearance frequency of mismatch fragments follows a heavy-tailed distribution, which means that there are a small number of mismatch fragments shared by a large number of the path pairs.

6 Inferring the Causes of Mismatch

In this section we look into causes of mismatch between BGP paths and traceroute derived paths, and classify them into 7 types as described in Sec. 2. Our classification algorithm follows an if-then-else process, *i.e.* it starts by checking whether the mismatch is of type 1, if yes then the classification stops, otherwise it continues and checks for type 2, and so on until it's put in *Unknown* bin.

1. **Divergence:** These are cases where the control plane is not aligned with the traceroute path. We detect these cases whenever the edit distance of the mismatch fragment (*i.e.* the number of modifying operations), has a high value, *i.e.* greater than 3 for *substitute* and *loop* patterns, or greater than 2 for *missing* and *extra* patterns. Below are two examples we find in our data:
   (a) **Tunneling:** The Amateur Packet Radio Network (AMPR.org) uses the prefix 44/8 announced by AS7377 (UCSD) and an overlay network on the Internet to tunnel traffic (including ICMP) between different parts of the network. So the BGP path to 44/8 ends at AS7377, while the traceroute path comes in AS7377, travels cross the overlay network on other ASes, and then ends in AS7377.
   (b) **Routing Dynamics:** We observed some *substitute* and *extra* mismatch fragments at the end of paths, where AS2512 (CalREN), the provider of UCLA, is appended. An example is '$= AS12969$, $-AS43571$, $+AS2512$, $=\$$', where AS12969 is more than 2 AS hops away from AS2152. This happened because some of our traceroute probes were actually falling in a routing loop within AS2152 immediately after reaching AS12969.
2. **Unannounced Prefixes:** In a *substitute* fragment, only '?' is inserted.
3. **Non-responsive Hops:** In a *substitute* fragment, only '$*$' is inserted.

For each following cause, given a mismatch fragment M, we conduct the specific tests on one or more AS pairs which are adjacent operands in M. Once one AS pair pass the test, the corresponding cause for M is determined. Let the AS pair be X and Y. For *extra* pattern, the pair is '$+X$' and '$= Y$'. For *missing* pattern, the pair is '$= X$' and '$-Y$'. For *substitute* pattern, the pair is '$+X$' and '$-Y$', or '$+X$' and '$= Y$'.

4. **MOAS Prefixes:** The matching prefix is announced by both X and Y.
5. **IXPs:** X is an IXP ASN, or the IP addresses mapped to X are used in IXPs.
6. **Siblings:** X and Y belong to the same sibling group.

Table 2. Taxonomy of causes of mismatch as measured in units of paths and fragments

%	Paths				Fragments			
	ucla	ams-nl	nrt-jp	she-cn	ucla	ams-nl	nrt-jp	she-cn
1 Divergence	0.58	3.34	0.34	2.00	6.25	4.09	6.11	4.46
2 Unannounced	1.49	0.83	7.22	3.49	2.02	1.81	2.31	2.97
3 Non-responsive	14.88	2.77	22.65	24.60	7.28	4.35	3.66	5.08
4 MOAS	9.22	0.50	0.90	1.12	2.42	2.61	2.09	1.97
5 IXP	32.56	1.77	10.86	34.66	6.45	3.21	3.19	4.22
6 Siblings	8.38	3.43	5.25	7.20	5.61	6.99	6.96	6.83
7 Neighbors	37.85	91.72	63.53	29.64	60.52	62.91	62.84	61.84
8 Unknown	0.63	0.36	0.66	1.12	9.45	14.05	12.84	12.65

7. **Neighbors:** Three types of tests are conducted: 1) According to the iPlane's alias list, the IP address mapped to X belongs to a router that has another interface mapped to Y. 2) X and Y are neighbors in BGP topology. 3) X and Y are neighbors in the topology from the IRR. The contributions of these three tests are 18%, 77% and 5%, respectively.

Table 2 shows the fraction of cases in each class, measured in the percentages of paths and fragments. The path values are relevant for comparison with previous work [13,10]. We can see that the majority of mismatch cases are the result of foreign addresses including IXPs, siblings and BGP neighbors. And over half of mismatch fragments are due to IP addresses sharing between BGP neighbors, that supports the view of [7]. The contribution of IXPs, siblings and MOAS only sum up to nearly 14%, although the previous work [13,10,12] considered them as the major causes. In addition, 6∼9% of mismatch cases are due to holes, *i.e.* unannounced or non-responsive hops, in traceroute paths.

Comparing the results in units of path and fragment side by side, we note that the fragment-based results is more robust to the monitor location and the possible flaws in cause inference than the path-based. And there are two types of bias in path-based counting: (1) Overestimating the influence of some causes, such as *IXPs* in *ucla* and *she-cn*. This is mainly because multiple paths may often share a single point of mismatch close to the monitor. (2) Underestimating the difficulty to infer causes. In *Unknown* bin, only about 1% of paths contain 9∼14% of fragments. About 93% of fragments in *Unknown* bin are at the end of path. Most of these cases may either correspond to BGP sessions not visible in the current BGP topology or due to misclassified *Divergence* cases.

7 Accuracy of Traceroute-Derived AS Connectivity

In this section we assess the accuracy of traceroute-derived AS adjacencies. According to our previous work [14], the BGP table of a monitor should reveal almost all its AS neighbors over time. The BGP AS graph from UCLA IRL is denoted by G_{bgp}. There are about 180 monitors providing full tables residing

Table 3. Inaccuracy of traceroute-derived AS topology by causes

%	$L_{\overline{bgp}}/L$				L_{bogus}/L			
	ucla	ams-nl	nrt-jp	she-cn	ucla	ams-nl	nrt-jp	she-cn
1 Divergence	1.85	2.19	2.61	2.26	0.68	0.65	0.82	0.78
2 Unannounced	0.76	0.78	1.09	1.31	0.42	0.37	0.55	0.29
3 Non-responsive	2.67	1.88	1.67	2.27	1.01	0.84	0.58	1.01
4 MOAS	0.62	0.68	0.50	0.54	0.10	0.09	0.09	0.08
5 IXP	2.29	1.19	1.32	1.51	0.35	0.35	0.27	0.40
6 Siblings	1.08	1.42	1.61	1.47	0.22	0.34	0.39	0.28
7 Neighbors	14.64	13.28	13.92	12.89	5.24	3.95	4.36	4.07
8 Unknown	3.22	5.68	5.63	5.50	0.65	1.05	1.01	0.97
Total	24.96	24.24	25.39	24.94	6.19	4.30	4.22	4.61

Table 4. Inaccuracy of public traceroute AS topology data sets

	L	$L_{\bar{bgp}}$	L_{bogus}	$L_{\bar{bgp}}/L$	L_{bogus}/L	$L_{bogus}/L_{\bar{bgp}}$
DIMES	77358	32159	11204	41.6%	14.5%	34.8%
Ark$_{direct}$	56014	14515	4510	25.9%	8.0%	31.1%
Ark$_{indirect}$	69962	25215	9059	36.0%	12.9%	35.9%
Total	104844	49731	17062	47.4%	16.3%	34.3%

in 112 different ASes connected to 13.6K unique ASes through a total of 43K links. Let G_{truth} denote this set of AS adjacencies. A traceroute-derived AS link $X - Y$ is bogus, if either X or Y is in our set of 112 ASes but the link $X - Y$ dose not exist in G_{truth}.

To evaluate the inaccuracy of traceroute-derived AS links, we inspect two values: $L_{\overline{bgp}}/L$ and L_{bogus}/L, where L is the number of links discovered by traceroute; $L_{\overline{bgp}}$ is the number of extra links not in G_{bgp}; L_{bogus} is the number of bogus links. The value $L_{\overline{bgp}}/L$ can be considered as a upper bound of the error rate, while the value L_{bogus}/L should be seen as a lower bound of inaccuracy.

To understand how the extra links and bogus links were created, we search for these links in our mismatch fragments and group them in the causes described in the previous section. Table 3 shows the inaccuracy of traceroute-derived AS topology by causes. The results from different monitors are similar. We see that the cause *Neighbors* are responsible for most of the links not seen in BGP, and contribute to the highest chunk of bogus links.

We also verify the accuracy of the AS adjacency sets provided by Ark and DIMES in Table 4. About 47% of AS adjacencies in the traceroute-derived topologies are not seen in BGP. In addition, we verify that about 16% of the traceroute AS adjacencies are false. Discarding the indirect links, that is caused by non-responsive hops or unannounced prefixes, in CAIDA's data can reduce the fraction of bogus links from 13% to 8%. However still 31% of the extra links not seen in BGP are actually bogus ($L_{bogus}/L_{\bar{bgp}}$).

8 Conclusion

In this paper we develop a systematic approach to identify and classify errors in AS paths inferred from traceroute using the LPM method. Our results shed light into the major pitfalls of traceroute-based AS topology measurement and show the limitations of publicly available AS topologies derived from traceroute. Since most of the inconsistencies originate from IP address sharing between BGP neighbors, we believe that building an accurate database of router interface aliases can bring significant improvement to the accuracy of the router path to AS path conversion process, and this is part of our future work.

References

1. Archipelago Measurement Infrastructure, http://www.caida.org/projects/ark/
2. Internet Routing Registry, http://www.irr.net/
3. RIPE routing information service project, http://www.ripe.net/
4. RouteViews routing table archive, http://www.routeviews.org/
5. UCLA IRL Internet topology collection, http://irl.cs.ucla.edu/topology/
6. Augustin, B., Cuvellier, X., Orgogozo, B., Viger, F., Friedman, T., Latapy, M., Magnien, C., Teixeira, R.: Avoiding traceroute anomalies with paris traceroute. In: IMC 2006 (2006)
7. Chang, H., Jamin, S., Willinger, W.: Inferring AS-level Internet topology from router-level path traces. In: SPIE ITCom (2001)
8. Hunt, J.W., Mcllroy, M.D.: An algorithm for differential file comparison. Tech. rep., Bell Laboratories (1976)
9. Hyun, Y., Broido, A., Claffy, K.C.: On third-party addresses in traceroute paths. In: Proc. of Passive and Active Measurement Workshop, PAM (2003)
10. Hyun, Y., Broido, A., Claffy, K.C.: Traceroute and BGP AS path incongruities. Tech. rep., CAIDA (2003)
11. Madhyastha, H., Isdal, T., Piatek, M., Dixon, C., Anderson, T., Krishnamurthy, A., Venkataramani, A.: iPlane: an information plane for distributed services. In: Proc. of OSDI (2006)
12. Mao, Z.M., Johnson, D., Rexford, J., Wang, J., Katz, R.H.: Scalable and accurate identification of AS-level forwarding paths. In: INFOCOM 2004 (2004)
13. Mao, Z.M., Rexford, J., Wang, J., Katz, R.H.: Towards an accurate AS-level traceroute tool. In: Proc. of ACM SIGCOMM (2003)
14. Oliveira, R., Pei, D., Willinger, W., Zhang, B., Zhang, L.: In search of the elusive ground truth: The Internet's AS-level connectivity structure. In: Proc. ACM SIGMETRICS (2008)
15. Oliveira, R., Zhang, B., Zhang, L.: Observing the evolution of Internet AS topology. In: ACM SIGCOMM (2007)
16. Shavitt, Y., Shir, E.: DIMES: Let the Internet measure itself. ACM SIGCOMM Computer Comm. Review, CCR (2005)

Toward Topology Dualism: Improving the Accuracy of AS Annotations for Routers*

Bradley Huffaker, Amogh Dhamdhere, Marina Fomenkov, and kc claffy

CAIDA, University of California, San Diego
{bradley,amogh,marina,kc}@caida.org

Abstract. To describe, analyze, and model the topological and structural characteristics of the Internet, researchers use Internet maps constructed at the router or autonomous system (AS) level. Although progress has been made on each front individually, a *dual graph* representing connectivity of routers with AS labels remains an elusive goal. We take steps toward merging the router-level and AS-level views of the Internet. We start from a collection of traces, i.e. sequences of IP addresses obtained with large-scale traceroute measurements from a distributed set of vantage points. We use state-of-the-art alias resolution techniques to identify interfaces belonging to the same router. We develop novel heuristics to assign routers to ASes, producing an **AS-router dual graph**. We validate our router assignment heuristics using data provided by tier-1 and tier-2 ISPs and five research networks, and show that we successfully assign 80% of routers with interfaces from multiple ASes to the correct AS. When we include routers with interfaces from a single AS, the accuracy drops to 71%, due to the 24% of total inferred routers for which our measurement or alias resolution fails to find an interface belonging to the correct AS. We use our dual graph construct to estimate economic properties of the AS-router dual graph, such as the number of internal and border routers owned by different types of ASes. We also demonstrate how our techniques can improve IP-AS mapping, including resolving up to 62% of false loops we observed in AS paths derived from traceroutes.

1 Introduction

There is growing scientific interest in the structure and dynamics of Internet topology, primarily at the router and Autonomous System (AS) levels. Substantial progress over the last decade toward understanding and improving the integrity and completeness of router and AS-level topologies *separately* (reviewed in Section 4) has inspired us to seek a graph construction that *merges router and AS-level views of the Internet.* Such a view would capture administrative boundaries while providing sufficient detail about the geography and internal structure of each AS. Inherent limitations and inaccuracies of existing techniques for alias resolution, IP-to-AS mapping, and router-to-AS assignment (not to mention validation of any of them) render this goal challenging.

In this work we take initial steps toward merging router and AS-level views into a *dual graph* representation of the Internet. We start from active measurement (traceroute-like) datasets collected using CAIDA's Archipelago distributed measurement infrastructure (Ark) [17]. We then apply state-of-the art alias resolution techniques [19] to infer

* Support for this work is provided by DHS N66001-08-C-2029 and NSF 05-51542.

which interfaces belong to the same router, creating a router-level Internet map. Finally, we propose heuristics to assign routers to ASes, using information derived from the interfaces that we infer belong to a particular router. We evaluate our AS assignment heuristics by validating against ground truth data from tier-1 and tier-2 ISPs and five research networks. We successfully assigned 80% of multi-AS routers, i.e, routers whose interfaces map to different ASes. When we include single-AS routers (routers whose interfaces all map to the same AS), the accuracy drops to drops to 71%, due to the 24% of total inferred routers for which our measurement or alias resolution fails to find an interface belonging to the correct AS. We also demonstrate how our techniques can be used to study the statistical properties of the resulting AS-router dual graph, and can improve IP-AS mapping of state-of-the-art AS-level traceroute tools.

2 Datasets and Methodology

We briefly describe three components of our methodology: gathering a large set of Internet path data; resolving IP address aliases to create a router-level graph; and designing heuristics to map annotated routers to ASes. All CAIDA data sets and tools developed to support this work will be publicly available.

2.1 Datasets

Active Measurements
We collected our active measurements using CAIDA's Archipelago (Ark) Measurement infrastructure [17], using 37 monitors in 28 countries. The Ark monitors used Paris traceroute [6] to randomly probe destinations from each routed /24 seen in BGP dumps from Routeviews over a 28-day collection period in September and October 2009. We call the resulting set of 268 million traceroute paths our *traceroute* dataset, which we used to infer which IP interfaces belong to the same router (Section 2.2).

BGP Data
To assign IP addresses to ASes, we used publicly available BGP dumps provided by Routeviews [26] and one of RIPE NCC's collectors (RCC12) [25]. BGP (Border Gateway Protocol) is the protocol for exchanging interdomain routing information among ASes in the Internet. A single origin AS typically announces ("originates") each routable prefix via BGP. We perform IP-to-AS mapping by assigning an IP address to the origin AS of the longest matching prefix for that IP address. We also used this BGP data to annotate each interdomain link with one of three (over-simplified) business relationships: customer-provider (the customer pays the provider); settlement-free peer (typically no money is exchanged); and sibling (both ASes belong to the same organization) – using the classification algorithm in Dimitropolous *et al.* [10].

Ground Truth Dataset
Our ground truth datasets includes private data from a tier-1 ISP (ISP_1) and a tier-2 ISP (ISP_2). In addition we use public data from the following research networks: CANET (ISP_C)[1], GEANT (ISP_G)[2], Internet2 (ISP_T)[4], I-Light (ISP_L)[3], and National LambdaRail (ISP_N)[5]. ISP_1 and the five research networks provided the full list

of interfaces. ISP_1 and ISP_2 provided their hostname conventions, which allowed us to identify interfaces in their address space, but not on their routers. We thus have two sets of interfaces for each network i: $\mathcal{I}_i$ (interfaces on routers that belong to network i) and $\bar{\mathcal{I}}_i$ (interfaces in i's address space, but on routers that do not belong to network i). For each network we then generate a list of AS numbers known to belong to that network: $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_C$, $\mathcal{A}_I$, etc., and the set of ASes that are not in each $\mathcal{A}_i$, denoted $\bar{\mathcal{A}}_i$.

2.2 Alias Resolution

For alias resolution, we rely on CAIDA's alias resolution tools MIDAR and *kapar* [19]. MIDAR expands on the IP velocity techniques of RadarGun [8], and kapar expands on the analytical techniques of APAR [14]. We first use the *traceroute* dataset as input to MIDAR, the output of which is fed into *kapar*. *kapar* heuristically infers the set of interfaces that belong to the same router, and the set of two or more routers on the same "IP link" (which could either be a point-to-point link, or LAN or cloud with multiple attached IP addresses). *kapar* produces two datasets corresponding to inferred nodes (routers) and links. Each node in the router dataset has a set of *known interfaces* and *inferred interfaces*. Known interfaces were measured directly; inferred interfaces result from *kapar* determining that a router r_1 has a link to interface i_2 on router r_2, but we did not see an actual interface on router r_1. The interfaces on an IP link are typically assigned IP addresses from the same prefix, so we assume that router r_1 must have an interface *from the same prefix as* i_2. The link dataset contains, for each link, the set of routers and router interfaces that we inferred as sharing that link. *kapar* correctly identified 66% of the true aliases from among the set of ISP_1's observed interfaces (our largest set of ground truth data), with a 5% false positive rate.

At least three limitations of our alias resolution techniques may affect the AS assignment process. First, a large number of interfaces and links between them are never observed, either because they do not respond to ICMP, or because none of the traceroutes encounter those interfaces. Second, some interfaces that respond to ICMP have addresses belonging to private address space, which makes them indistinguishable from other interfaces using the same private address space. Third, even when all of a router's interfaces are discovered, we may have insufficient information to infer that they belong to the same router. For example, we inferred 1390 routers as having interfaces from a single AS in $\mathcal{A}_1$, which our method would infer to mean these routers are in ISP_1. But our ground truth dataset refutes this inference; these routers do not belong to ISP_1, and likely have an interface (which we either did not observe or did not resolve accurately) from at least one other AS in $\bar{\mathcal{A}}_1$.

2.3 AS Assignment Methods

The goal of the AS assignment process is to determine the AS that owns each router. For each router r, we create an *AS frequency matrix* that counts the number of interfaces (known and inferred) from each AS that appears on r. The ASes in this frequency matrix represent the set of possible owner ASes of r. Next, we describe the heuristics we designed to determine r's ownership from among the candidates present in r's AS frequency matrix. Figure 1 illustrates the five heuristics examined in this paper.

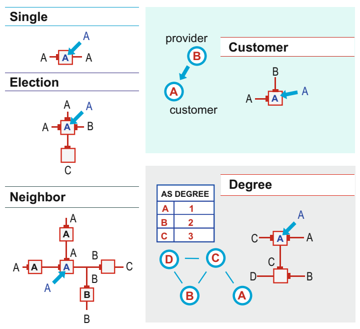


Fig. 1. Depiction of five evaluated heuristics for assigning AS labels to routers: **Single** (only one choice); **Election** (assign to AS with largest number of interfaces); **Neighbor** (assign to AS with most neighbors); **Customer** (assign to customer AS); **Degree** (assign to smallest degree AS).

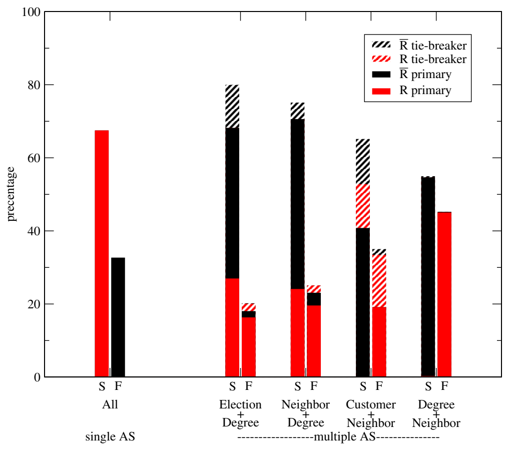


Fig. 2. Success (S) and failure (F) rates of AS assignment primary heuristics, and the best tie-breaking heuristics for each primary, for single-AS and multi-AS routers in $\mathbb{R}$ and $\bar{\mathbb{R}}$

Single: This heuristic is used for the case where a single AS is present in r's AS frequency matrix. In this case, we (trivially) assign r to this AS.

Election: This heuristic assigns a router r to the AS with the highest frequency in r's AS frequency matrix, assuming routers tend to have more interfaces in the address space of their owner. **Election** produces an ambiguous assignment when multiple ASes have the same (highest) frequency, which occurred for 14% of the multi-AS routers in our set.

Neighbor: For this heuristic, we first determine the set of single-AS routers to which r is connected (its single-AS neighbors). We create a new AS frequency matrix that counts the number of single-AS neighbors of r from each AS. The **Neighbor** heuristic assigns r to the AS with the largest frequency (most single-AS neighbors), based on the intuition that a router is connected to a larger number of single-AS routers in its owner AS. **Neighbor** produces an ambiguous assignment when multiple ASes have the same (highest) frequency.

Customer: This heuristic uses the AS relationship dataset to assign relationships to each pair of ASes from r's AS frequency matrix[1]. **Customer** assigns r to the AS inferred to be a customer of every other AS in r's AS frequency matrix. This heuristic is based on the common practice that customer and provider routers typically interconnect using addresses from the provider's address space. Consequently, a router with interfaces from both the customer and provider address spaces is assigned to the customer.

[1] Not every possible AS pair in r's frequency matrix has a known relationship; many AS pairs have no link between them in the original BGP AS graph, so no defined relationship.

Degree: For this heuristic, we first generate an AS-level graph by assuming full-mesh connectivity among ASes from each router's AS frequency matrix. We then use this graph to generate an AS degree for each AS. **Degree** assigns router r to the smallest-degree AS from r's AS frequency matrix, i.e., the AS most likely to be the customer AS, based on similar intuition as the **Customer** heuristic.

2.4 Evaluation of AS Assignment Heuristics

We next evaluate our AS assignment heuristics by comparing our AS assignment with our ground truth datasets. We classify each router inferred by *kapar* into the following sets. If a router r_0 has at least one interface in $\mathcal{I}_i$, then we assign r_0 to the set $\mathcal{R}_i$ (the set of routers owned by ISP$_i$). If a router r_1 has at least one interface from the set $\bar{\mathcal{I}}_i$, then we assign r_1 to the set $\bar{\mathcal{R}}_i$ (inferred routers not owned by ISP$_i$). We found 39 routers (0.6% of the total analyzed) with interfaces in both $\mathcal{I}_i$ and $\bar{\mathcal{I}}_i$ or $\mathcal{I}_i$ and $\mathcal{I}_j$, which contradicts the meaning of these data sets (describing mutually exclusive routers). These discrepancies are due to false positives in our alias resolution process, so we discard them for the purpose of evaluating our AS assignment heuristics. All but three routers in $\mathcal{R}_i$ have a single AS in $\mathcal{A}_i$ ($\mathcal{A}_i$ is the set of ASes owned by ISP$_i$), which means there is a single successful assignment for most routers. For the three routers with multiple ASes in A_i, successful assignment is ambiguous, and we omitted these routers from the evaluation, leaving us with $|\mathcal{R}_1|$ = 3,405 and $|\bar{\mathcal{R}}_1|$ = 2,254, $|\mathcal{R}_2|$ = 241 and $|\bar{\mathcal{R}}_2|$ = 86, $|\mathcal{R}_G|$ = 37 and $|\bar{\mathcal{R}}_G|$ = 0, $|\mathcal{R}_L|$ = 32 and $|\bar{\mathcal{R}}_L|$ = 0, $|\mathcal{R}_T|$ = 17 and $|\bar{\mathcal{R}}_T|$ = 0, $|\mathcal{R}_N|$ = 16 and $|\bar{\mathcal{R}}_N|$ = 0, and $|\mathcal{R}_C|$ = 8 and $|\bar{\mathcal{R}}_C|$ = 0. We call the combined set of all routers $\mathbb{R} = \cup\mathcal{R}_i$, those owned by some network in our ground truth dataset, and the set $\bar{\mathbb{R}} = \cup\bar{\mathcal{R}}_i$ those we know not to be owned by a specific network in our ground truth datasets. Using our knowledge of interface ownership, we derive $|\mathbb{R}|$ = 3,795 and $|\bar{\mathbb{R}}|$ = 2,340 routers on which to test AS assignment heuristics. We consider $H(r)$, the AS to which a certain heuristic assigns router r, as a *successful assignment* if $((r \in \mathcal{R}_i)\&\&(H(r) \in \mathcal{A}_i))||((r \in \bar{\mathcal{R}}_i)\&\&(H(r) \in \bar{\mathcal{A}}_i))$, i.e., if the router is in $\mathbb{R}$ and $H(r)$ selects an AS owned by the same ISP as the router, or the router is in $\bar{\mathbb{R}}$ and $H(r)$ selects an AS not owned by the ISP known to not own router.

Section 2.3 outlined the cases for which each heuristic provides an ambiguous assignment. To resolve ambiguous assignments, i.e., break ties, we paired each heuristic with a second one. We tested all combinations of pairs of heuristics to find the best tie-breaker[2] for each primary heuristic, resulting in the following combinations: **Election + Degree**, **Neighbor + Degree**, **Customer + Neighbor**, and **Degree + Neighbor**.

Figure 2 shows the fraction of routers we assigned successfully (bars labeled "S"), and the fraction that were failures (bars labeled "F"), determined using the ground truth datasets. Figure 2 presents these results separately for routers in $\mathbb{R}$ and $\bar{\mathbb{R}}$, and for different assignment heuristics. We found that for single-AS routers, all heuristics are either successful for the 67% in $\mathbb{R}$ or failures for the 33% in $\bar{\mathbb{R}}$. The explanation is straightforward: All routers in $\mathcal{R}_i$ or $\bar{\mathcal{R}}_i$ have at least one interface in ISP$_i$'s address space (not necessarily being used by ISP$_i$), and by extension an AS in $\mathcal{A}_i$. For single-AS routers in

[2] The best tie-breaker is the heuristic that produced the largest number of successful assignments for routers where the primary heuristic resulted in an ambiguous assignment.

$\mathcal{R}_i$, the AS must belong to $\mathcal{A}_i$, and the assignment is a success. For single-AS routers in $\bar{\mathcal{R}}_i$, assigning it to that single AS results in failure. For these single-AS routers in $\bar{\mathcal{R}}_i$, we have most likely failed to either see or accurately resolve the alias for the router's interface in address space not owned by ISP$_i$.

Figure 2 shows that when a router has interfaces from multiple ASes, the most effective stand-alone heuristic was **Neighbor**, which successfully assigned 70% of these routers. **Election** + **Degree** was the most successful combination of heuristics (mainly due to fewer failures on routers from $\bar{\mathbb{R}}$), with a success rate of 80%.

3 Applications of AS Assignment

In this section, we use the AS assignment heuristics described in Section 2.3 to produce a *dual graph* that merges router and AS-level topologies. We then describe two applications of this dual graph construct – producing representative dual topologies of the Internet, and improving the accuracy of AS-level traceroute tools.

3.1 Toward Representative Dual Topologies of the Internet

Previous work [11,21] has focused on generating AS-level graphs of arbitrary size, while preserving the correlation structure seen in real Internet topology, e.g., correlations between the number of customers, providers and peers of an AS, or between degrees of ASes at each end of an inter-domain link. We seek to extend this previous work by designing a graph generator that can produce Internet-like dual topologies, i.e., AS annotated router-level graphs, of arbitrary size, preserving the statistical properties of the Internet's dual graph. Another application is to security-related situational awareness objectives, which require knowledge of the internal structure of ASes. We focus on two questions: How many inferred single-AS (internal) and multi-AS (border) routers do ASes own (with the aforementioned caveat that we may mis-characterize routers as single-AS if we undersample or mis-resolve interfaces)? Is there a correlation between an AS's degree and the number of routers it owns? We use the heuristics from Section 2.3 to assign routers to ASes, and measure the router ownership properties of resulting ASes. Our results do not represent the actual number of routers owned by an AS, only the number observed in our data samples.

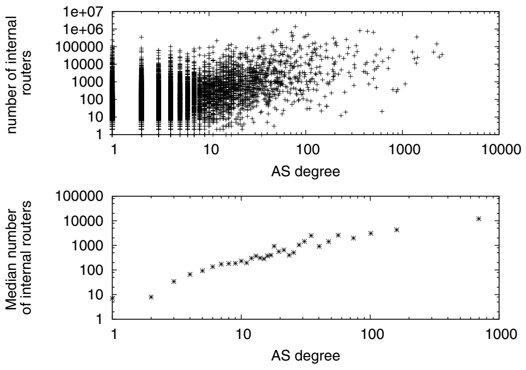


Fig. 3. The number of single-AS routers per AS vs degree (top) and the median number of single-AS routers per AS vs degree (bottom)

We first examine the number of single-AS routers owned by an AS, which does not depend on the assignment heuristic we used, since every heuristic assigns a single-AS router to the same AS. The top graph in Figure 3 shows a scatter plot of the number of single-AS routers inferred per AS versus the AS degree as observed in BGP data (from Routeviews2 and RIPE's RRC12). We confirmed the expected positive correlation, where ASes with larger degrees (which typically represent larger transit providers) tend to have more single-AS routers. Several outliers have many single-AS routers and relatively low AS degrees (1 or 2). The top 10 such outliers corresponded to ASes that were either regional networks of a larger transit provider, or smaller administrative domains within a large transit provider. Consequently, these ASes had just one or two observed AS links, to the backbone AS of the larger transit provider. It is plausible that such regional transit networks or access provider networks have a large number of single-AS routers.

The bottom graph in Figure 3 shows the median number of single-AS routers per AS as a function of the AS degree. We bin ASes according to their degree, ensuring a minimum bin size of 50 ASes. We see a strong positive correlation between the number of single-AS routers and the inferred AS degree, which is expected since ASes with larger AS degrees typically represent transit providers, which need many routers. ASes with lower degrees are typically stub networks with less internal routing infrastructure.

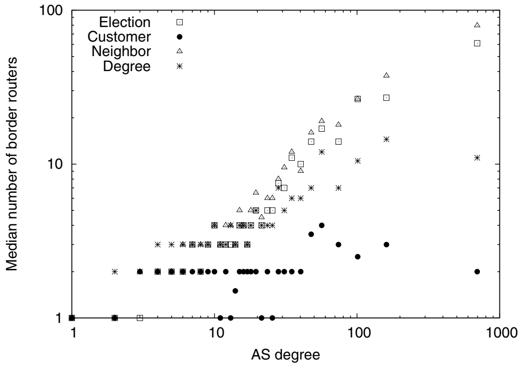


Fig. 4. Number of multi-AS routers per AS using **Election**, **Neighbor** and **Degree** heuristics shows strong correlation with AS degree

Figure 4 shows the number of multi-AS routers owned by an AS as a function of AS degree, for different AS assignment heuristics. We found similar results with the **Election**, **Neighbor** and **Degree** assignment heuristics, and a strong positive correlation between the number of multi-AS routers of an AS and its degree. The **Customer** heuristic shows a much weaker correlation between the number of multi-AS routers and AS degree. **Customer** favors lower (BGP) degree ASes, since customer ASes tend to have smaller degrees than their providers, and **Customer** assigns a multi-AS router to the customer AS, decreasing the number of multi-AS routers for ASes with larger BGP degrees. We found that the **Neighbor** heuristic favors higher (BGP) degree ASes, inflating the number of multi-AS routers for higher degree ASes.

3.2 Toward Accurate AS-Traceroute

As a second application of the dual graph construct, we outline an approach to designing a more accurate AS-traceroute tool, a problem first studied by Mao *et al.* [23]. Mao *et al.* concluded that an accurate router-level map of the Internet would help to resolve anomalies seen in AS paths derived from traceroutes. Here, we investigate whether our AS assignment heuristics can improve AS-traceroute accuracy, by resolving anomalies such as missing AS hops, extra AS hops and AS loops. Identifying missing and extra

AS hops requires BGP feeds from the vantage points used for traceroute measurements, which the Ark infrastructure does not yet have. However, we can identify traceroutes that have AS loops, by performing an IP-to-AS mapping using BGP dumps collected from Routeviews and RIPE. Mao *et al.*. [23] noted two possible explanations for false loops in traceroute paths: the presence of Internet Exchange Point (IXP) infrastructure, and sibling ASes. We investigated whether our router-to-AS assignment alone can help to resolve these loops. In future work, we plan to incorporate IXP data collected by Augustin *et al.*. [7] to identify false loops due to IXP infrastructure, and WHOIS data to identify false loops due to sibling ASes.

By applying IP-to-AS mapping on the sequence of interfaces seen in each traceroute, we found that most Ark monitors yielded fewer than 5% of inferred AS paths that had loops. However, traces from one particular monitor yielded 75% of inferred AS paths with loops, which we discovered was caused by a single incorrectly mapped interface traversed by most traces from that monitor. We removed these traces for the remainder of our analysis. We then assigned an AS to each inferred router on the path using the AS assignment heuristics from Section 2.3. We replaced the loop segment in the traceroute AS path with an AS path segment derived from the router assignment heuristic. We measured the fraction of paths with traceroute loops resolved, i.e., removed, via this procedure. Figure 5 shows the fraction of traces with AS path loops that we could resolve using each of the AS assignment heuristics. We found that the **Customer** heuristic performed poorly. The **Neighbor** heuristic, which was the most accurate stand-alone AS assignment heuristic (Section 2.4) was able to resolve 62% of AS path loops. The combination **Election+Degree**, which was the most accurate combination AS-assignment heuristic, was able to resolve just over 61% of AS path loops.

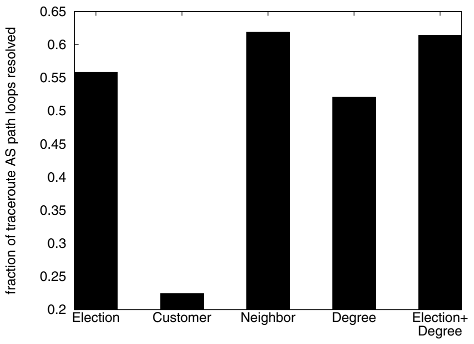


Fig. 5. Fractions of traceroute loops resolved by each heuristic

4 Related Work

There has been significant interest in studying structural properties of the Internet at the router and AS-levels for over a decade [12]. Several measurement studies have since highlighted the incompleteness of topologies inferred from publicly available routing data [9,16,22,24,32]. Much work has gone toward capturing as much of the Internet's AS-level topology as possible, most notably Zhang *et al.* [32] and He *et al.* [16]. Several large-scale active measurement projects, including Ark [17], iplane [20], and DIMES [27], use traceroutes from distributed vantage points to a large set of destinations across the IPv4 Internet. The resulting datasets have been used to reconstruct router and AS-level topologies, but merging the two views has received less attention.

A major challenge in deriving topologies from traceroute measurements is *alias resolution*, i.e., determining which interfaces belong to the same router. Tangmunarunkit *et* *al.* [13] proposed Mercator, a tool that attempted alias resolution by observing response packets sent from different interfaces than those probed. Spring *et al.* [29] used Ally to detect when two candidate interfaces likely shared the IP ID counter. Follow up work on alias resolution [8,15,28] used techniques such as IP ID counter velocities, DNS hostname conventions, and bi-directional traceroutes. Keys [19] recently documented CAIDA's attempt to expand and combine these techniques into a unified system.

There has been relatively little work on assigning routers (inferred by the previous alias resolution techniques) to the ASes that own those routers. Tangmunarunkit *et al.* [31] used a simple heuristic based on longest prefix matching to assign routers (inferred using Mercator) to ASes. Due to a lack of ground truth data, they were not able to validate their router-to-AS assignment heuristic. Tangmunarunkit *et al.* [30] was the first to study the properties of ASes in terms of the number of routers per AS. They found that ASes show high variability in the number of routers, and the number of routers per AS is highly correlated with BGP AS degree. Our work on improving AS-traceroute is inspired by the work of Mao *et al.* [23], who studied the discrepancies between traceroute-derived AS paths and BGP AS paths, and Hyun *et al.* [18], who measured the presence of third-party addresses in traceroute paths.

5 Conclusions

We have presented an approach to merge router and AS-level views of the Internet, creating a *dual graph of the Internet*. We proposed new heuristics for assigning routers from traceroute-derived graphs to ASes. We validated the success rates of our heuristics against ground truth data from a set of commercial ISPs and research networks. For multi-AS routers, the most successful heuristic was a combination of **Election** (assign the router to the AS with the largest number of interfaces) followed by **Degree** (assign the router to the AS with the smallest degree), with a success rate of 80%. For 32% of inferred single-AS routers, we either missed or mis-resolved some interface that belonged to the true owning AS, reducing our overall AS assignment accuracy to 71%. We also showed how our AS assignment techniques could be used to quantify statistical properties of ASes, as well as to improve on current state-of-the-art AS-traceroute techniques, resolving up to 62% of false loops observed in traceroute-derived AS paths.

References

1. Canet4 topology data, `http://dooka.canet4.net/`
2. Geant topology data, `http://stats.geant2.net/lg/`
3. I-light topology data, `http://routerproxy.grnoc.iu.edu/ilight/`
4. Internet2 topology data, `http://vn.grnoc.iu.edu/Internet2`
5. National lambdarail topology data, `http://routerproxy.grnoc.iu.edu/nlr2/`
6. Augustin, B., Cuvellier, X., Orgogozo, B., Viger, F., Friedman, T., Latapy, M.: Avoiding Traceroute Anonmalies with Paris Traceroute. In: Proc. Internet Measurement Conference, IMC (2006)
7. Augustin, B., Krishnamurthy, B., Willinger, W.: IXPs: Mapped?. In: Proc. Internet Measurement Conference, IMC (2009)

8. Bender, A., Sherwood, R., Spring, N.: Fixing Ally's Growing Pains with Velocity Modelling. In: Proc. Internet Measurement Conference, IMC (2008)
9. Cohen, R., Raz, D.: The Internet Dark Matter - On the Missing Links in the AS Connectivity Map. In: Proc. IEEE Infocom (2006)
10. Dimitropoulos, X., Krioukov, D., Fomenkov, M., Huffaker, B., Hyun, Y., Claffy, K., Riley, G.: AS Relationships: Inference and Validation. In: ACM SIGCOMM CCR (2007)
11. Dimitropoulos, X., Krioukov, D., Vahdat, A., Riley, G.: Graph annotations in Modeling Complex Network Topologies. ACM Transactions on Modeling and Computer Simulation 19(4) (2009)
12. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On Power-law Relationships of the Internet Topology. In: Proc. ACM SIGCOMM (1999)
13. Govindan, R., Tangmunarunkit, H.: Heuristics for Internet Map Discovery. In: Proc. IEEE INFOCOM (2000)
14. Gunes, M.H.: APAR tool, http://itom.utdallas.edu/data/APAR.tar.gz (accessed 2008-07-02)
15. Gunes, M.H., Sarac, K.: Analytical IP Alias Resolution. In: Proc. IEEE International Conference on Communications, ICC (2006)
16. He, Y., Siganos, G., Faloutsos, M., Krishnamurthy, S.V.: A Systematic Framework for Unearthing the Missing Links: Measurements and Impact. In: Proc. USENIX/SIGCOMM NSDI (2007)
17. Hyun, Y.: Archipelago Infrastructure, http://www.caida.org/projects/ark/
18. Hyun, Y., Broido, A., Claffy, K.: On Third-party Addresses in Traceroute Paths. In: Proc. Passive and Active Measurement Conference, PAM (2003)
19. Keys, K.: Internet-Scale IP Alias Resolution Techniques. In: ACM SIGCOMM CCR (2010)
20. Madhyastha, H.V., Katz-Bassett, E., Anderson, T., Krishnamurthy, A., Venkataramani, A.: iPlane: An Information Plane for Distributed Services. In: Proc. USENIX OSDI (2006)
21. Mahadevan, P., Hubble, C., Krioukov, D., Huffaker, B., Vahdat, A.: Orbis: Rescaling Degree Correlations to Generate Annotated Internet Topologies. In: Proc. ACM SIGCOMM (2007)
22. Mahadevan, P., Krioukov, D., Fomenkov, M., Huffaker, B., Dimitropoulos, X., Claffy, K., Vahdat, A.: The Internet AS-Level Topology: Three Data Sources and One Definitive Metric. In: ACM SIGCOMM CCR (2005)
23. Mao, Z.M., Rexford, J., Wang, J., Katz, R.H.: Towards an Accurate AS-level Traceroute Tool. In: Proc. ACM SIGCOMM (2003)
24. Oliveira, R., Pei, D., Willinger, W., Zhang, B., Zhang, L.: In Search of the Elusive Ground Truth: The Internet's AS-level Connectivity Structure. In: Proc. ACM SIGMETRICS (2008)
25. RIPE NCC. Rcc12 bgp collector, http://www.ripe.net/projects/ris/rawdata.html
26. University of Oregon RouteViews Project, http://www.routeviews.org/
27. Shavitt, Y., Shir, E.: DIMES: Let the Internet Measure Itself. In: ACM SIGCOMM CCR (October 2005)
28. Spring, N., Dontcheva, M., Rodrig, M., Wetherall, D.: How to Resolve IP Aliases. Technical Report UW-CSE-TR 04-05-04 (2004)
29. Spring, N., Mahajan, R., Wetherall, D.: Measuring ISP Topologies with Rocketfuel. In: Proc. ACM SIGCOMM (2002)
30. Tangmunarunkit, H., Doyle, J., Govindan, R., Willinger, W., Jamin, S., Shenker, S.: Does AS Size Determine Degree in AS Topology? In: ACM SIGCOMM CCR (2001)
31. Tangmunarunkit, H., Govindan, R., Shenker, S., Estrin, D.: The Impact of Routing Policy on Internet Paths. In: Proc. IEEE INFOCOM (2001)
32. Zhang, B., Liu, R., Massey, D., Zhang, L.: Collecting the Internet AS-level Topology. In: ACM SIGCOMM CCR (2005)

The RIPE NCC Internet Measurement Data Repository

Tony McGregor[1,2], Shane Alcock[1], and Daniel Karrenberg[2]

[1] University of Waikato, Hamilton, New Zealand
[2] RIPE NCC, Amsterdam, The Netherlands
(tonym,salcock)@cs.waikato.ac.nz,
daniel.karrenberg@ripe.net

Abstract. This paper describes datasets that will shortly be made available to the research community through an Internet measurement data repository operated by the RIPE NCC. The datasets include measurements collected by RIPE NCC projects, packet trace sets recovered from the defunct NLANR website and datasets collected and currently hosted by other research institutions. This work aims to raise awareness of these datasets amongst researchers and to promote discussion about possible changes to the data collection processes to ensure that the measurements are relevant and useful to the community.

1 Introduction

A core requirement of any Internet measurement project is to acquire appropriate measurement data. However, privacy and security concerns often prevent researchers from being able to collect the data themselves. It is very important, therefore, that organisations that collect useful measurement data are able to share it with the research community. Time that would otherwise be spent conducting measurements can instead be dedicated to the analysis of existing data. Shared access to measurement resources also promotes collaboration between researchers and allows validation studies to be performed.

One common problem when sharing Internet measurement data is cultivating awareness of data sets amongst the research community. At present, publicly available data is typically scattered amongst a large number of hosting locations, meaning that it can be difficult for researchers to locate suitable datasets and keep informed of new datasets as they are released. DatCat [1] has helped in this regard, but it is not yet a comprehensive resource.

Maintenance of repositories hosted by research groups that depend on competitive grants for funding is also a significant concern. This was evidenced by the recent disappearance of the NLANR website which had hosted many passive trace sets, including the popular Auckland and Abilene traces. In this instance, it was fortunate that the contents of the site were salvaged by the University of Waikato with the support of CAIDA before they became inaccessible. However, the data could easily have been lost to the research community permanently if that intervention had not taken place.

As a Regional Internet Registry (RIR), the RIPE Network Coordination Centre (RIPE NCC) has the ability to collect a large quantity of measurement data that would be extremely difficult for researchers based in academic institutions to acquire themselves. Some of the RIPE NCC data is already publicly available, but each RIPE NCC project shares data independently in a variety of ways. Therefore, the RIPE NCC is developing a common and consistent platform for hosting and sharing Internet measurement data. While the primary goal is to streamline the mechanisms by which the RIPE NCC datasets can be accessed, the data repository will be open to other collectors who wish to share their measurement data with the research community. By grouping the datasets into a single repository, finding and accessing appropriate measurement data will be easier and awareness of the datasets that are available to researchers will be increased. One advantage of the RIPE NCC founding and operating a measurement data repository is that the continued existence of the repository does not depend solely on research grants and the likelihood of the repository disappearing is much smaller.

One of the most significant issues that arises when sharing Internet measurement data is that of anonymisation. Datasets that are being published for the first time will need to be anonymised in some fashion and agreements with users must be developed in order to prevent inappropriate disclosure of personal and commercial information. Such decisions will need to be made on a case-by-case basis, as each dataset can contain different types of sensitive information, e.g. policies for protecting IP addresses will not be applicable to personal information such as names and contact details.

Similarly, the structure and scope of a system for providing useful metadata and annotation of the shared datasets is yet to be completely determined. We expect that entries for datasets shared through the RIPE repository will be added to existing sites such as DatCat [1] and WITS [2], as well as a site that is developed and hosted as part of the RIPE repository itself. We hope to elicit ideas and thoughts from the wider research community with regard to the information that should be provided through such a system and the best format in which to present it. The RIPE NCC also plans to identify all users of the repository and keep contact with them during their research. It is hoped that this will encourage researchers to engage with the data collectors about how the measurements can be improved to be more useful and relevant.

The remainder of this paper describes the datasets that are currently under consideration for sharing through the RIPE repository. For each dataset, a brief overview of the dataset, its associated research project and the measurement techniques employed is presented.

2 RIPE Datasets

2.1 K-root

The K-root service is an Internet root name service operated by the RIPE NCC [3]. The server consists of seventeen nodes located both inside and outside of

Europe. Six of the nodes are global instances and are announced with an anycast 23-bit prefix. The remaining eleven nodes are local instances announced with a 24-bit prefix using the Border Gateway Protocol (BGP) no-export community tag [4]. Each node operates three distinct data collection systems.

Firstly, tcpdump [5] is used to capture passive traces of incoming traffic on port 53, i.e. Domain Name Server (DNS) queries. The trace files are rotated hourly and retained on disk for five days. Each node generates between 300 and 500 megabytes of compressed traces per hour. The total amount of data produced daily through this system is approximately 300 gigabytes.

As part of the "Day in the Life of the Internet" (DITL) project organised by CAIDA [6], traces for a 50 hour period have been fetched and archived each year. The 2008 DITL traces contain 1.46 billion packets and are 600 gigabytes in size [7]. At present, this data is hosted by DNS-OARC [8] and is available under their terms and conditions. Meta-data about the traces, including query rates per node and known issues that may affect analysis such as clock skew, has been documented publicly on DatCat, a measurement data catalogue [1].

Secondly, each node operates a DNS Statistics Collector (DSC) [9] that captures DNS traffic and summarises it into one minute bins. This data is used to generate graphs that are shown on the K-root website. In addition, the raw DSC output is transferred to RIPE NCC and archived indefinitely. The archive extends back to the beginning of 2008. The amount of data collected is estimated to be approximately 1 megabyte per day, with the entire 2008 dataset being less than 200 MB. The raw data has also been exported to DNS-OARC where it can be accessed by members.

Finally, Simple Network Management Protocol (SNMP) statistics are collected from the last-hop router serving each K-root node. The SNMP queries originate from RIPE NCC in Amsterdam. If connectivity to a queried router is lost, data will not be collected and the resulting dataset will not be contiguous. The statistics are summarised and exported into a round-robin database (RRD) where they can be queried and analysed using Cacti [10]. The RRD is configured to retain the SNMP data for a year.

2.2 Reverse DNS

RIPE NCC also hosts reverse DNS (rDNS) services for its delegated address space. There are four servers providing rDNS and other associated services. These servers process approximately 50 thousand queries per second which is more than triple the load of the K-root server. Because of the high query rate, it is not feasible to regularly collect passive traces directly on these servers. Occasional tcpdump traces have been collected when there was a specific need, such as during an attack, but the traces are short and irregular. However, if there was sufficient need, it may be possible to collect a sample or summary of the traffic using a dedicated collector on a mirrored switch port.

DSC is used on each of the rDNS servers and the raw data is kept indefinitely. At present, this data and the DSC graphs are only available internally at RIPE

NCC but could be made available to researchers if there was a need amongst the research community.

2.3 AS112

RIPE NCC hosts an Autonomous System (AS) number 112 [11] reverse DNS and dynamic DNS update server for the RFC 1918 private address space [12]. The server processes about 2000 transactions per second. As anyone can announce the AS112 prefix, there is no definitive list of AS112 servers. There are more than 50 servers listed at [11] but there are almost certainly others.

A passive tcpdump trace is collected from the RIPE NCC AS112 server annually and contributed to the DITL project [6]. More frequent passive captures could be scheduled from this server if required. In addition, DSC data is collected and used to generate graphs that are publicly available from the RIPE NCC AS112 website [13].

2.4 RIS

The Routing Information Service (RIS) is a set of 16 route collectors running quagga [14] that peer with approximately 600 BGP routers. Most of the collectors are located within the RIPE region, but there are a few elsewhere including the United States and South America. The routes learned are not used for routing traffic but are instead collected and published to provide a resource for understanding Internet routing and diagnosing routing problems. Around 100 of the peers provide a complete routing table and the others provide partial tables. The BGP sessions include both IPv4 and IPv6.

The collectors export route updates every five minutes and perform a full table dump every eight hours. It is normal that, at any one time, not all of the peers are actively peered with a collector. There are many causes explaining these peering gaps including configuration changes, equipment failure, network failure and human error. There are also some gaps in the dataset due to system failures. To reduce the impact of these errors, an automated system detects results that are much smaller than expected and informs an administrator who will investigate the fault. Also, it should be noted that some Interior Gateway Protocol (IGP) routes are leaked to RIS, meaning that there will be some single-bit prefixes advertised in the data.

The data collected from the RIS is stored in the Multi-threaded Routing Toolkit (MRT) format [15]. All data collected since the system started in 2000 is retained indefinitely. A month of data is approximately 22 gigabytes compressed and the entire dataset is close to one terabyte. The last three months of data is exported to a MySQL database which is also one terabyte in size. Quagga logs and a selection of meta-data is supplied with the RIS data. The logs show when peers have started or ended a BGP session and when timers expired, for example.

The raw data is currently publicly available through the RIS website. This is accessed by around 1000 distinct hosts each month, including the BGPmon

[16] and Cyclops [17] websites which use the data to offer route announcement and bogon notification services. In addition, RIPE NCC also publish weekly statistical reports, have released a variety of tools for querying and visualising the RIS data and have enabled Looking Glass queries to be sent directly to the collectors. Users can also subscribe to a notification service that will inform them of interesting events, such as a change in the advertisement for a particular AS.

2.5 Hostcount

The Hostcount project [18] generates statistics from a monthly DNS scan of approximately 100 top level domains (TLDs) within the RIPE region. The scan is performed by conducting a zone transfer on the DNS tree rooted at each TLD. During the walk, counts of A and PTR records are maintained for both forward and reverse IPv4 addresses as well as forward AAAA (but not reverse) IPv6 addresses.

As public zone transfers are disabled by most DNS administrators, the scan is not exhaustive. Some administrators permit the RIPE scan, but this is often under the condition that only statistics, rather than raw data, are published. The blocking of zone transfers has increased over time so the data from earlier years is a better reflection of the total number of hosts at that time.

Currently, the statistics are published via the Hostcount website [18]. These include the number of distinct hosts found at different levels of the DNS tree for each TLD, the number of zones discovered and the number of zone transfers that were permitted and successful. The raw data from between 1990 and 2007 has been archived and is currently in off-line storage. However, the present policy is to discard the raw data after statistics have been extracted. This could be reversed if there was sufficient need amongst the research community.

2.6 TTM

Test Traffic Measurements (TTM) is an active measurement system consisting of 105 operational probes [19] that has been operating since 1999. The majority of probes are located at ISPs and academic institutions within the RIPE region. Other probe locations include the United States, South America, Asia, the Middle East, Australia and New Zealand. The clocks on the probes are synchronised using GPS to give a 10 microsecond accuracy to real-time. With the exception of some probes that are in private meshes, the TTM project conducts full mesh measurements.

The probes regularly perform a series of active tests including a UDP one-way delay test, traceroute, a multicast performance test (limited to sites that have enabled multicast measurements), DNSMON measurements (see Section 2.7 below) and IPv6 pMTU discovery. In addition, the TTM probes support conducting ad hoc measurements that are initiated by authorised users. The ad hoc tests include a ping test and an HTTP page fetch. It is also possible to develop and run arbitrary ad hoc tests. There are limits on the range of destinations and the probing rate for ad hoc tests and the results are not released to other sites or to the public. To maintain the integrity of the system, the probes are managed solely by the RIPE NCC. However, it may be possible to request special tests be performed using the TTM framework, provided there is no significant impact on the existing measurements.

At present, performance graphs are available via the TTM website to users who accept an electronic license agreement. Bulk data is also published using the CERN ROOT format [20] to researchers who sign a paper license agreement. The total dataset is approximately 0.7 terabytes in size, with the dataset from 2008 being 110 gigabytes.

2.7 DNSMON

The DNS Monitor project (DNSMON) collects data regarding the reachability and latency for some of the top levels of the DNS system [21]. The data is collected using 60 TTM probes that are not located in private meshes. The root domain, `.com`, `.net`, `.org`, `e164.arpa` and 24 country code TLDs (mostly from within the RIPE region) are measured. Performance over both IPv4 and IPv6 is measured for probes that have IPv6 connectivity. Name Server Identifier [22] information returned in the DNS response is retained and may be used in the future to generate per anycast instance graphs.

Summary statistics are available dating back to the commencement of the project in April 2003, although not all servers have been monitored since this time. Graphs are publicly available to anyone, but only paying subscribers can access graphs for the last two hours. Raw data is retained for approximately a year. Each probe collects between 10 and 20 megabytes of uncompressed data per day and the total dataset is 121 gigabytes in size. The raw data is available on request to country code TLD administrators and researchers, although non-subscribers are restricted from accessing the most recent two days of data.

2.8 RIPE DB

The RIPE DB is an open shared database that is maintained by the RIPE community [23], containing 3.2 million public object records. There are three classes of data stored in the database: Internet number registration objects, reverse DNS domain objects and route registry objects.

Internet number registration objects store details about IP addresses and AS numbers including references to administrative and technical contact information. The RIPE DB only contains records for the RIPE region, not the entire Internet. The vast majority of the 2.6 million registration objects are for IPv4 addresses, but there are also 24,000 IPv6 address and 18,000 AS number objects in the database.

At present, there are 410,000 reverse DNS objects and eight thousand forward domain objects in the RIPE DB. Historically, both forward and reverse domain information was stored but forward domains are no longer encouraged except as community support for small, emerging domains. It is likely that all forward domain support will cease in the future. The reverse DNS records are used to create the zone files for the RIPE NCC reverse DNS service.

The route registry objects are used to provide an Internet Routing Registry (IRR), enabling organisations that participate in Internet routing to store and publish their routing policy. The RIPE DB contains approximately 100,000 route registry objects. The structure of the routing data in the database conforms to the Routing Policy Specification Language [24] and the structure and use of the route registry conforms to RFC 2650 [25]. Standard tools exist that can be used to check the policies stored in the IRR data for consistency and to generate router configurations from the IRR records. Unlike the other classes of object in the RIPE DB, the route registry is synchronised to other IRRs and copies of the information from the other IRRs is retained.

Public queries of the RIPE DB are supported through the use of both command-line and web `whois` tools. A daily limit is imposed on the number of queries that include personal information attributes. Bulk data is also available via FTP. The bulk data files are generated daily, including both a file with the complete database and files split by object type. Personal details, such as the person and maintainer objects, are not included in these files. The complete database file is approximately 150MB in size. In addition, it is possible to subscribe to a near real time mirror feed of the database for an annual fee.

Access to personal data within the RIPE DB is restricted for both legal and practical (e.g. limiting abuse) reasons. At present, the restrictions are applied at a very broad level which sometimes results in limitations that are inappropriate. For example, an ISP that has entered a large number of person objects may not be able to access all the objects that they have created. This problem will be resolved as part of the development of the common RIPE data sharing platform.

3 External Datasets

3.1 Auckland

The Auckland dataset consists of a series of trace sets collected by the WAND group at the University of Waikato. The traces were collected at the University of Auckland in New Zealand, measuring the link between the University and the Internet. All of the traces were captured using DAG hardware capture cards [26], although the card model was upgraded on several occasions. Each of the Auckland trace sets is briefly described in Table 1, which is based on detailed summaries provided by the Waikato Internet Trace Storage (WITS) project [2].

There have been some significant variations in the capture configuration between each trace set. Some of the changes were necessitated by the network infrastructure being upgraded, meaning that the measurement point was no longer capable of capturing all of the traffic it had previously. In other cases, the amount of packet header data that was retained is varied, e.g. the Auckland VI traces captured the first 64 bytes of every TCP, UDP and ICMP packet whereas the Auckland VII traces are limited to only the ATM cell header for all packets regardless of protocol.

Most of the Auckland traces were publicly released by NLANR [27] and frequently feature in measurement literature such as [28] and [29]. With the recent

Table 1. The Auckland trace sets

Name	Format	Year	Duration	Packets	Bytes	Size
I	ERF	1999	7 days	169 M	8 GB	2 GB
II	Legacy ATM	2000	24 days	996 M	359 GB	26 GB
IV	Legacy ATM	2001	45 days	3,157 M	1,269 GB	64 GB
V	ATM Cell	2001	7.5 hours	2,710 M	133 GB	8 GB
VI	Mixed Legacy	2001	4.5 days	844 M	345 GB	17 GB
VII	ATM Cell	2001	15.5 hours	6,040 M	297 GB	19 GB
VIII	ERF	2003	13 days	1,654 M	698 GB	68 GB

demise of the NLANR site, the traces have become difficult for researchers to acquire. In addition, the RIPE repository will include the Auckland I and Auckland V trace sets, which were not available from NLANR.

3.2 Waikato

The Waikato dataset is a collection of six very long-duration trace sets captured at the border of the University of Waikato network by the WAND group. The capture point is located between the University network infrastructure and the commodity Internet, allowing access to all traffic entering and exiting the University but excluding any internal traffic. All of the traces were captured using software that was specifically developed for the Waikato capture point [30] and a DAG 3 series hardware capture card [26]. All IP addresses within the traces are anonymised using Crypto-Pan AES encryption [31], with the encryption key being changed on a weekly basis. A brief description of each of the trace sets is shown in Table 2.

The location and hardware of the capture point has remained unchanged since the first capture began in 2003. The software has been upgraded between each trace set, resulting in some minor variations. For example, in Waikato I packets are truncated at the end of the transport header but subsequent trace sets retained four bytes of unanonymised application payload for all packets except in the case of DNS packets where twelve bytes were kept.

Table 2. The Waikato trace sets

Name	Format	Years	Duration	Packets	Bytes	Size
I	ERF	2003-2005	620 days	53,263 M	21,434 GB	1,329 GB
II	ERF	2005-2006	301 days	34,712 M	15,789 GB	839 GB
III	ERF	2006-2007	160 days	21,984 M	9,144 GB	545 GB
IV	ERF	2007	56 days	10,128 M	4,588 GB	255 GB
V	ERF	2007	99 days	19,710 M	9,740 GB	491 GB
VI	ERF	2007-2008	135 days	20,886 M	11,092 GB	495 GB

The Waikato I trace set is currently available for public download from the WITS archive [2]. The other Waikato trace sets will be made available through the RIPE data repository.

3.3 NLANR Datasets

The NLANR project [27] collected both active and passive datasets. These data sets have been the focus of a significant amount of research and many papers have been published based on them. Although the project is complete, the datasets collected are still in demand. They have been preserved by the WAND network research group and are available from the WITS [2] repository. The traces will also be hosted on the RIPE repository.

4 Conclusion

This paper catalogues and describes the large quantity of Internet measurement data that will be shared with the research community through a data repository hosted by the RIPE NCC. While much of the data described here is already publicly available, it is scattered amongst a variety of hosting organisations or, in the case of the Auckland traces, is no longer available from the original source. By creating a common portal for sharing and accessing all of the data, it will become easier for researchers to locate and download suitable measurement data for their particular project.

The primary aim of the repository is to bridge the gap between the organisations capable of conducting Internet measurements and the researchers who analyse the measurement data. This is evidenced by the partnership with the University of Waikato to enable the Auckland and Waikato datasets to be mirrored on the RIPE repository. However, the relationship must function in both directions to ensure the collected data is relevant and useful. As a result, we encourage submissions from the community regarding the collection and format of any of the aforementioned datasets that would improve their utility to researchers.

References

1. Cooperative Association for Internet Data Analysis (CAIDA): DatCat: Internet Measurement Data Catalog, http://imdc.datcat.org/
2. WAND Network Research Group: WITS: Waikato Internet Traffic Storage, http://www.wand.net.nz/wits/
3. RIPE NCC: K-root, http://k.root-servers.org/
4. Chandra, R., Traina, P., Li, T.: RFC 1997 - BGP Communities Attribute (August 1996)
5. Jacobson, V., Leres, C., McCanne, S.: Tcpdump, http://www.tcpdump.org/
6. Cooperative Association for Internet Data Analysis (CAIDA): A Day in the Life of the Internet, http://www.caida.org/projects/ditl/

7. Nagele, W., Buddhdev, A., Wessels, D.: K-root DNS traces DITL (2008) (collection), http://imdc.datcat.org/collection/1-0690-J=K-root+DNS+traces+DITL+2008
8. DNS-OARC: Domain Name System Operations, Analysis and Research Center, https://www.dns-oarc.net/
9. The Measurement Factory: DSC: A DNS Statistics Collector, http://dns.measurement-factory.com/tools/dsc/
10. Cacti: http://www.cacti.net/
11. The AS112 Project: http://www.as112.net/
12. Rekhter, Y., Moskowitx, B., Karrenberg, D., de Groot, G.J., Lear, E.: RFC 1918 - Address Allocation for Private Internets (February 1996)
13. RIPE NCC: RIPE NCC AS112, http://www.ripe.net/as112/
14. Quagga: http://www.quagga.net/web/quagga.html
15. Blunk, L., Karir, M., Labovitz, C.: MRT routing information export format (IETF Draft), http://tools.ietf.org/html/draft-ietf-grow-mrt-04
16. BGPmon: http://bgpmon.net/
17. Oliveira, R.V., Lad, M., Zhang, L.: Cyclops, http://cyclops.cs.ucla.edu/
18. RIPE NCC: Hostcount, http://www.ripe.net/is/hostcount/stats
19. RIPE NCC: Test Traffic Measurements, http://www.ripe.net/ttm/
20. C.E.R.N.: Root, http://root.cern.ch/drupal/
21. RIPE NCC: RIPE NCC DNS Monitoring Services, http://dnsmon.ripe.net/dns-servmon/
22. Austein, R.: RFC 5001 - DNS Name Server Identifier (NSID) Option (August 2007)
23. RIPE NCC: RIPE Database, http://www.ripe.net/db/
24. Alaettinoglu, C., Villamizar, C., Gerich, E., Kessens, D., Meyer, D., Bates, T., Karrenberg, D., Terpstra, M.: RFC 2622 - Routing Policy Specification Language, RPSL (June 1999)
25. Meyer, D., Schmitz, J., Orange, C., Prior, M., Alaettinoglu, C.: RFC 2650 - Using RPSL in Practice (August 1999)
26. Endace Measurement Systems, Ltd: http://www.endace.com
27. McGregor, A., Braun, H.W., Brown, J.: The NLANR NAI Network Analysis Infrastructure. IEEE Communication Magazine: Special Issue on Network Measurement, 122–128 (May 2000)
28. Erman, J., Arlitt, M., Mahanti, A.: Traffic Classification Using Clustering Algorithms. In: MineNet 2006: Proceedings of the 2006 SIGCOMM workshop on Mining network data, pp. 281–286. ACM, New York (2006)
29. McGregor, A., Hall, M., Lorier, P., Brunskill, J.: Flow Clustering Using Machine Learning Techniques. In: Passive and Active Measurement, pp. 205–214 (2004)
30. WAND Network Research Group: WDCap, http://research.wand.net.nz/software/wdcap.php
31. Fan, J., Xu, J., Ammar, M.H., Moon, S.B.: Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme. Computer Networks 46(2), 253–272 (2004)

Enabling High-Performance Internet-Wide Measurements on Windows

Matt Smith and Dmitri Loguinov*

Department of Computer Science and Engineering
Texas A&M University, College Station, TX 77843, USA
{matt,dmitri}@cse.tamu.edu

Abstract. This paper presents analysis of the Windows kernel network stack and designs a novel high-performance NDIS driver platform called IRLstack whose goal is to enable large-scale Internet measurements that require sending billions of packets and managing millions of outstanding connections on inexpensive commodity hardware available to any research lab. Our results show that with just 75% of one modern CPU core, IRLstack can saturate a gigabit link with SYN packets (i.e., 1.48M pps) and achieve 3.52 Gbps (i.e., 5.25 Mpps) with a quad-core CPU. IRLstack's transmission performance exceeds that of Winsock by a factor of 92-174, batch-mode WinPcap by a factor of 4.7-6.7, and the latest optimized PF_RING/TNAPI Linux kernel by up to 30%.

1 Introduction

With the expansion in size and popularity of the Internet, many distributed applications now require high-performance network stacks to sustain the scalability demands of their users. Traditional domains that exhibit a significant network burden in terms of bitrate and packets per second (pps) are massive Internet services with hundreds of millions of active users (e.g., Google, Facebook, Blogspot, root DNS servers, CDNs), whose main approach to solving scalability issues has been to acquire vast server clusters and distribute incoming requests across multiple geographic datacenters. While scaling the server side of network applications in commercial applications has a well-established solution, researchers often face scalability problems from the *client* side (i.e., issuing rather than receiving requests) and often do not have the resources to deploy dedicated clusters to conduct their Internet measurements. To overcome this problem, we investigate scalability issues arising during Internet-wide experimental studies, explore network-stack bottlenecks in the most-commonly deployed OS in the Internet (i.e., Microsoft Windows), and propose a solution that enables large-scale network measurements using a single inexpensive Windows host.

Due to its ever-growing size, diversity, decentralized nature, and enormous amount of information, the Internet is becoming more of a mystery every day (e.g., even Google does not know how big the web is [16]). Many Internet studies

* Supported by NSF grant CNS-0720571.

aim to shed light on its structure [2], [13], user behavior [14], [17], host availability [6], [11], and web content [7], [10]; however, accurately capturing Internet-wide metrics has long remained a challenging research problem. The main tradeoff involves the amount of available hardware and the delay the user is willing to tolerate. In many cases, measurements over a longer period of time are less desirable as they skew the obtained result, delay the corresponding analysis, and potentially impede future research. To provide additional motivation for developing large-scale measurement platforms, we next outline several of our projects that have experienced network-stack bottlenecks and then present our solution.

Our first project [17] involves measurement of P2P networks and modeling of various system properties (e.g., churn, lifetimes, topology) based on the obtained results. This process relies on a Gnutella crawler that contacts all alive ultra-peers in the system and obtains their neighbors via special requests. In order for the measurement to be unbiased [15], it is highly beneficial to capture Gnutella snapshots instantaneously; otherwise, a crawl of duration T samples a superposition of multiple Gnutella networks that exist during interval $[0, T]$. Given approximately 1.2M ultra-peers, connection rates on the order of 200K/sec are needed to guarantee cover times that would approximate an instantaneous snapshot (i.e., 10 seconds or less). While [17] was over 18 times faster than any previous P2P crawler, its coverage delay of 3 minutes could use a lot of improvement; however, various bottlenecks inside the Windows kernel leave no room for much speedup.

Our second project [7] is a high-performance web crawler IRLbot, whose main requirement has been keeping CPU utilization of the network stack close to 0% in order to leave room for computationally expensive processing related to spam control, HTML parsing, page decompression, calculation of domain reputation, and checking for duplicates. With one CPU core almost entirely dedicated to networking, IRLbot is usually CPU-limited during its crawls. Since Winsock does not scale very well to multiple cores (see below), achieving very high download rates is almost infeasible with a single host.

Our third project studies the DNS infrastructure for Internet-wide delay measurements [8] and various botnet-related anomalies, which requires traversing the DNS tree with over 650M DNS requests. Sending such a large number of small packets presents a problem for Winsock and limits the duration of the measurement to days instead of minutes. A slightly different, but related, measurement goal that requires high pps sending rates is discovery of open services using horizontal scanning [1], [3], [6], [11], where each IP address in the IANA (3.3B destinations) or BGP (2.1B) space is probed with a packet on a given port. Instead of using months to scan the Internet as in prior work [1], [3], [6], [11], our goal in another ongoing project is to accomplish this activity in several hours/days.

Other applications that are enabled by a scalable network stack are various Intrusion Detection Systems (IDS), firewalls, software routers, and network monitoring tools, all of which require line-rate capture of incoming packets and sometimes certain processing on the fly. Leaving as much CPU as possible for processing and not dropping any packets are both of critical importance.

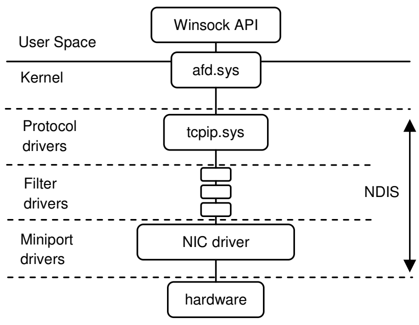
The novelty of this work lies not only in our approach to designing a high-performance *client-side* rather than server-side network stack, but also in our tackling of this problem in *Windows*, which has not been attempted before (see [4], [5], [12] for Linux approaches). The benefit of using just the client side of TCP is that it requires minimal functionality of performing the SYN handshake and sending one request packet, without tedious congestion-control functionality, management of complex timers and buffers, and retransmission overhead. As a result, a well-designed TCP stack can function at wire speed. The benefit of using Windows lies in its wide range of powerful APIs, outstanding support (in terms of software and hardware), and more ubiquitous deployment opportunities (i.e., finding a Windows host to conduct measurements is simpler than a Linux host, especially at remote locations). As there is a general perception that Windows is too slow for serious high-performance research work, we aim to dispel this myth and provide researchers with an additional platform option.

2 Overview of Windows and Linux Network Stacks

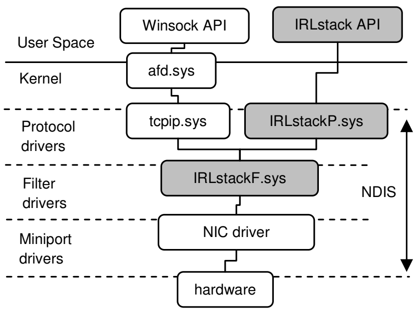
The structure of the Windows networking stack is illustrated in Fig. 1(a). Application packets are transmitted through a Winsock API into the kernel driver afd.sys whose main purpose is to manage the socket interface and interact with *protocol drivers* inside NDIS. Most normal Winsock exchange takes place with the default TCP/IP protocol driver tcpip.sys. Packet buffers created by TCP/IP are then sent down the stack to any *filter drivers* that are registered in the stack, which may do additional processing and/or filtering. The last step of this chain are *miniport drivers*, which are specific to each NIC and whose purpose is to directly interface with the hardware, set up DMA transfers, process interrupts, and manage the assigned adapter. Once the miniport has sent the frame (or queued it internally) and no longer needs the structure it received, it issues a callback up the stack indicating completion of the request, which causes the corresponding protocol driver to notify the user-space caller of the completion of their request. This process, described in terms of the synchronous send path, is similar on the receive side and for asynchronous operations.

Besides Winsock, network applications can use WinPcap [9], which is a popular tool for network capture and transmission on Windows. It is implemented as a filter driver with an API directly exported to a user-space library. Since it is located below tcpip.sys inside NDIS, it handles raw link-layer frames and bypasses most of the Windows network stack, which in theory should enable it to perform significantly faster than standard Windows sockets. However, as we will show in Section 3, this is not the case in practice.

The third alternative is a highly optimized Linux network stack such as the one developed by the ntop project [5] (the default Linux performance is lower and not studied here). Ntop makes use of a custom Linux kernel and modified network adapter drivers to exploit the features of the latest NICs. The first



(a) Winsock



(b) IRLstack

Fig. 1. Windows network stack, NDIS, and IRLstack

main modification is PF_RING, whose primary contribution is using DMA in combination with technologies such as Intel's I/OAT to directly expose kernel memory buffers (into which incoming packets have been transferred) to user-space processes. The second modification is TNAPI, which deserializes receive operations by exposing multiple RX (receive) queues as virtual adapters that can be used concurrently in user-space. This distributes load across several processors and allows the stack to scale in multi-core systems.

3 Performance of Winsock and WinPcap

Most of the issues discussed in the introduction arise from the poor *small-packet* performance of default Windows and Linux kernels. Our goal then is to achieve wire-rate transmission of arbitrary packets from user-space to many unique destinations, which translates into high rates of outgoing TCP connections/sec, fast horizontal scanning of the Internet, and low-overhead management of millions of concurrent connections to numerous remote servers (e.g., using multiple IPs aliased to the same interface with 64K ports each).

To calculate the target pps rate, we focus on gigabit Ethernet as one common example. The IEEE 802.3 Ethernet standards define the minimum frame size as 64 bytes, with smaller packets padded by the adapter as necessary. Taking into account the inter-grame gap (12 bytes), preamble (7 bytes), and the "start of frame" delimiter (1 byte), 84 bytes (672 bits) must be transmitted per minimum-size frame including overhead. We thus arrive at $1,000,000,000/672 = 1,488,095$ frames per second as the absolute upper limit for gigabit Ethernet, which is our performance goal. Using three handshake packets (SYN, SYN-ACK, ACK) and one RST for terminating connections, the absolute best performance of any TCP stack is 371K connections/sec. In applications that require graceful termination with four FIN packets, this number is 212K/sec.

3.1 Raw Packets

We now examine the performance of the Windows network stack to measure the maximum send rate of TCP SYN packets on a raw socket (ICMP and UDP results are nearly identical and thus omitted). All Windows tests in this paper are run on a dual AMD Opteron 2427 (2.2GHz, six cores per socket) system with 32GB of DDR2-667 RAM. The NIC is an Intel Pro/1000 PT Quad-Port Gigabit PCI-E NIC, and the OS is Windows Server 2008 SP2. We dedicate a single CPU core to each gigabit port and restrict the OS kernel, all drivers, and user-space programs to run on as many cores as there are ports being used during the test. All reported CPU utilization numbers later in the paper are relative to the number of active cores.

As shown in the first row of Table 1, Winsock can send packets to a single destination at rates between 116 Kpps (single core, single port) and 193 Kpps (quad-core, quad-port) at 100% CPU utilization. Winsock additionally drops its performance by a factor of 7 when each packet targets a unique IP address (demonstrated in the next line of the table). In order to alleviate the CPU overhead, we experimentally found that completely disabling (not just turning off) certain default Windows services (e.g., firewall and network list service) allowed Winsock to achieve a 25-80% speedup for a single destination and a five-fold rate increase for multiple destinations as shown in the next two rows of the table. However, this performance is still quite poor compared to the line rate of 1.48 Mpps and far from desirable in practice as no other processing can be done on the server due to the high CPU utilization. Furthermore, disabling critical Windows services (such as the firewall) causes installation of certain OS updates to fail and potentially leaves the host vulnerable to attack, which is undesirable. Another interesting result, shown in the last two rows of the table, is that WinPcap performs no better (and sometimes worse) than Winsock with disabled services.

As CPU usage is extremely high for the number of packets sent for both approaches above and multi-core scaling is rather poor due to various bottlenecks in the kernel, one must conclude that Winsock and WinPcap are unsuitable for truly high-performance applications.

Table 1. Server 2008 SP2 raw SYN transmission performance

Method	Destinations	Rate in pps (link utilization)		CPU
		1 port / 1 core	4 ports / 4 cores	
Winsock (default)	single	116,037 (7.7%)	193,142 (3.2%)	100%
	all unique	16,290 (1%)	30,110 (0.5%)	100%
Winsock (services off)	single	207,041 (14%)	244,196 (4.1%)	100%
	all unique	75,687 (5%)	153,232 (2.5%)	100%
WinPcap 4.1	single	49,349 (3.3%)	152,467 (2.5%)	75%
	all unique	49,493 (3.3%)	152,297 (2.5%)	75%

Table 2. Server 2008 SP2 TCP connection performance to a single destination

Method	Rate (conn/sec)		CPU
	1 port / 1 core	4 ports / 4 cores	
connect/closesocket	16,656	39,462	100%
connectEx/disconnectEx	20,801	45,277	100%
WSK (kernel mode)	31,389	54,783	100%

3.2 TCP Connections

TCP connection performance to a single destination is summarized in Table 2. The standard approach using the Unix BSD socket interface (i.e., connect/closesocket) achieves between 16K and 39K connections/sec, which is slightly surpassed by the new Winsock APIs connectEx/disconnectEx with their 20 and 45K connections/sec, respectively. The performance gain is related to the fact that these APIs keep sockets open between subsequent connections. Finally, the new (i.e., Vista/Server 2008) kernel-level Winsock interface WSK is measurably faster at 31 and 54K connections/sec, but its multi-core scalability is again quite poor. Connection rates to multiple unique destinations are much worse and not shown here due to limited space.

4 IRLstack: Overcoming the Bottlenecks

Kernel stack traces indicate that the performance drop when sending raw packets to many unique destinations occurs in afd.sys and tcpip.sys in Fig. 1(a). Bypassing them completely and generating raw SYN packets entirely from within the kernel brings performance up to 289 Kpps (single-core) and 652 Kpps (quad-core), regardless of firewall settings and how many destinations are used. Nevertheless, this solution is hardly acceptable as it still consumes 100% of the CPU, stays well below link capacity, and requires writing kernel-level code for each high-performance application, which is cumbersome and prone to crashing the system.

Further profiling of NDIS shows that its path from protocol to miniport drivers in Fig. 1(a) has another major bottleneck in synchronization spinlocks and DMA transfers to the NIC. To overcome this problem, we developed a general-purpose suite of network drivers called IRLstack that accepts buffers of packets from user-space (using standard Windows API calls such as WriteFile) and transmits them in a single call to the miniport driver. Multiple outstanding asynchronous requests are supported via overlapped I/O. The buffer consists of multiple raw link-layer frames, each preceded by an IRLstack-specific header. Link-layer, IP, and TCP/UDP checksums may be omitted as they are calculated by the NIC using checksum offloading.

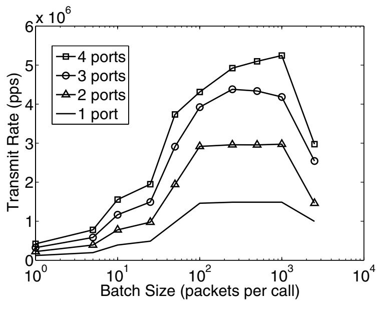
At the kernel level, the protocol driver scans through the buffer creating the appropriate auxiliary data structures for each encountered packet and proceeds to send the entire batch in a single call as allowed by NDIS. Batching multiple packets maximizes useful work between acquisitions of kernel spinlocks, ensures that the send path remains zero-copy, and allows the NIC to perform large DMA transfers directly from user-space. In Fig. 1(b), protocol driver IRLstackP.sys handles raw application-layer packets through a special IRLstack API, while filter driver IRLstackF.sys intercepts return packets and channels those destined to IRLstack applications back to IRLstackP.sys. The remaining packets are sent to tcpip.sys as before. This is accomplished by redirecting any incoming traffic destined to non-default IP addresses on the NIC (assumed to be allocated for IRLstack's use) to our protocol driver.

4.1 Sending

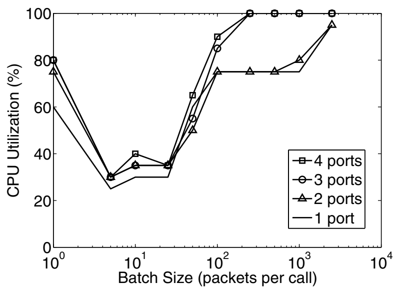
The first issue we investigate is the transmit performance of IRLstack and the optimal batch size needed to saturate the link. Results of our testing can be seen in Fig. 2(a). All transmission rates are far from optimal until the burst size starts to exceed 50 packets, at which point IRLstack achieves 50-66% (depending on the number of cores) link utilization. For single and dual-core cases, full wire speed is reached with any batch size between 128 and 1,024 packets, while the 3-core setup has a unique peak at 256 packets and the 4-core case maxes out at 1,024. Interestingly, for very large batch sizes, performance actually drops due to bottlenecks in the Intel miniport driver, which for some reason is unable to efficiently handle large bursts of packets.

As seen in Fig. 2(b) at batch sizes 128-512, IRLstack can saturate a 1 Gbps link with just 75% CPU utilization of a single 2.2GHz core and two gigabit links using 75% of two cores. With multiple NIC ports, IRLstack scales much better than Winsock and achieves 5.25 Mpps as shown Table 3. This scaling is less than linear due to synchronization bottlenecks stemming from the common (single-threaded) miniport driver controlling all four ports, though the main sublinear dropoff only occurs when increasing from 3 to 4 ports.

While WinPcap also exports a batch-mode interface to *user-space*, it does not fully utilize the interfaces provided in NDIS 5.x and later (e.g., NdisSendNetBufferLists in 6.x) for batching within the kernel. This makes its multi-packet



(a) TX Performance



(b) CPU Utilization

Fig. 2. IRLstack transmission performance and CPU utilization using 40-byte SYN packets

Table 3. Send performance with SYN packets using optimal batch size (2.2 GHz Opteron 2427)

Method	Rate in pps (link utilization)			
	1 port / 1 core	2 ports / 2 cores	3 ports / 3 cores	4 ports / 4 cores
IRLstack	1,487,298 (100%)	2,973,933 (100%)	4,379,999 (98%)	5,245,458 (88%)
WinPcap	319,814 (21%)	485,617 (16%)	648,370 (14%)	815,921 (14%)

performance significantly lower than it could be as also seen in Table 3. We thus note that IRLstack's in-kernel batching techniques could be easily implemented in WinPcap as well, benefitting those who seek higher pps performance in WinPcap-based tools on commodity PC platforms.

4.2 Receiving

While most of our projects have required high sending rates, additional research can be enabled by a network stack that allows high capture rates as well. We now turn our attention to receive performance in Table 4, where we only focus on IRLstack, with Linux PF_RING/TNAPI numbers [5] provided as a reference. (Winsock/WinPcap results are again vastly suboptimal and are thus omitted.) Observe in the table that the receive path in IRLstack is approximately 20-50% slower than the send path, which can be explained by two factors. First, our receive path is *not* zero-copy as it was during transmission, because IRLstack is able to directly export user-space buffers for DMA transfers *into* the NIC; however, no reverse functionality (i.e., *from* the NIC) is provided by NDIS unless specialized hardware is used. Second, the interrupt frequency is higher along the receive path than the send path since the miniport driver controls the former and IRLstack controls the latter. With the maximum miniport batch size equal to 64 packets, it is no wonder that it is unable to sustain the wire speed along the receive path. If future versions of Intel drivers remove this limitation, much higher receive rates are to be expected.

Nevertheless, IRLstack's receive performance compares quite favorably to the latest Linux numbers from a custom PF_RING/TNAPI kernel [5]. Specifically, both solutions achieve close to 3 Mpps with quad-cores and four independent RX queues (we use four gigabit ports, while [5] uses a single 10 GE adapter with four hardware queues). This is despite IRLstack's receive path not being zero-copy (which it is in [5] using Intel I/OAT), its use of rather frequent

Table 4. Receive performance with SYN packets (IRLstack on a 2.2 GHz Opteron 2427 vs. Linux on a 2.4 GHz Xeon 54xx)

Method	Rate (pps)			
	1 port / 1 core	2 ports / 2 cores	3 ports / 3 cores	4 ports / 4 cores
IRLstack	1,232,745 (82%)	1,526,460 (51%)	2,282,554 (51%)	2,946,707 (50%)
Linux [5]	$\sim 920,000$ (61%)	–	–	$\sim 3,000,000$

64-packet interrupts, standard Intel NIC drivers, default NIC settings (e.g., adaptive interrupt moderation), and no kernel modifications (i.e., all drivers are loaded at run-time). Furthermore, while [5] posts the highest throughput numbers we've seen on Linux, it is meant for capture only and does not have a transmit path for general-purpose traffic.

4.3 TCP Connections

IRLstack implements the client side of TCP in user space, which simultaneously allows for easy debugging and high-performance management of numerous outstanding connections – hiding the work of constructing link-layer frames that would otherwise be required of the user. All supported operations are performed using batching and include issuing outgoing connections with three handshake packets, ability to send requests in regular or ACK packets of the handshake (which is however not always supported by remote servers), and standard SACK TCP receiver functionality (i.e., selective ACKs, large windows, etc.). To avoid keeping the server in the time-wait state, the application has an option of terminating connections using RST packets, in which case the useful connection throughput in Gnutella-like applications is close to 250K/sec (i.e., four control packets, one request packet, one reply packet).

4.4 Latency

It should be noted that the receive-path interrupt batching provides notification from the miniport to NDIS every 64 packets and is not under control of IRLstack. The batching delay along the send path, however, is user-selectable based on the batch size passed down to IRLstack. Thus, applications that require accurate timestamps might need to trade off pps performance for lower latency by changing the miniport interrupt moderation and reducing the batch size during transmission.

5 Conclusions and Future Work

We have shown that while Windows is often overlooked as a platform on which to conduct serious networking research, perhaps due to impressions of inefficiency or low performance, this need not be the case. With a well-designed NDIS 6.x network stack, it is possible to achieve wire-rate transmission (and near wire-rate reception) on gigabit Ethernet using inexpensive commodity hardware. IRLstack achieves a nearly 100-fold increase in transmission performance over Winsock (when unique destinations are used), with lower CPU usage. Moreover, it can coexist with the default network stack on a single adapter so that other network applications may run as usual.

Future work involves expanding IRLstack's receive performance (e.g., using DMA remapping, multiple hardware queues) and evaluating its performance on 10 GE hardware.

References

1. Benoit, D., Trudel, A.: World's First Web Census. Intl. Journal of Web Information Systems 3(4), 378–389 (2007)
2. Chang, H., Jamin, S., Willinger, W.: To Peer or not to Peer: Modeling the Evolution of the Internet's AS-level Topology. In: Proc. IEEE INFOCOM (April 2006)
3. Dagon, D., Provos, N., Lee, C.P., Lee, W.: Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. In: Proc. NDSS (February 2008)
4. Degioanni, L., Varenni, G.: Introducing Scalability in Network Measurement: Toward 10 Gbps with Commodity Hardware. In: Proc. ACM IMC, October 2004, pp. 233–238 (2004)
5. Deri, L., Fusco, F.: Exploiting Commodity Multicore Systems for Network Traffic Analysis (July 2009), http://ethereal.ntop.org/MulticorePacketCapture.pdf
6. Heidemann, J., Pradkin, Y., Govindan, R., Papadopoulos, C., Bartlett, G., Bannister, J.: Census and Survey of the Visible Internet. In: Proc. ACM IMC, October 2008, pp. 169–182 (2008)
7. Lee, H.-T., Leonard, D., Wang, X., Loguinov, D.: IRLbot: Scaling to 6 Billion Pages and Beyond. In: Proc. WWW, April 2008, pp. 427–436 (2008)
8. Leonard, D., Loguinov, D.: Turbo King: Framework for Large-Scale Internet Delay Measurements. In: Proc. IEEE INFOCOM, April 2008, pp. 430–438 (2008)
9. WinPcap: The Windows Packet Capture Library, http://www.winpcap.org/
10. Najork, M., Heydon, A.: High-Performance Web Crawling. Compaq Systems Research Center, Tech. Rep. 173 (September 2001), http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-173.pdf.
11. Pryadkin, Y., Lindell, R., Bannister, J., Govindan, R.: An Empirical Evaluation of IP Address Space Occupancy. USC/ISI, Tech. Rep. ISI-TR-2004-598 (November 2004)
12. Schneider, F., Wallerich, J., Feldmann, A.: Packet Capture in 10-Gigabit Ethernet Environments Using Contemporary Commodity Hardware. In: Proc. PAM, April 2007, pp. 207–217 (2007)
13. Spring, N., Mahajan, R., Wetherall, D.: Measuring ISP Topologies with Rocketfuel. In: Proc. ACM SIGCOMM (August 2002)
14. Stutzbach, D., Rejaie, R.: Understanding Churn in Peer-to-Peer Networks. In: Proc. ACM IMC, October 2006, pp. 189–202 (2006)
15. Stutzbach, D., Rejaie, R., Duffield, N., Sen, S., Willinger, W.: On Unbiased Sampling for Unstructured Peer-to-Peer Networks. In: Proc. ACM IMC, April 2006, pp. 27–40 (2006)
16. The Official Google Blog, We knew the web was big (July 2008), http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html
17. Wang, X., Yao, Z., Loguinov, D.: Residual-Based Estimation of Peer and Link Lifetimes in P2P Networks. IEEE/ACM Trans. Networking 17(3), 726–739 (2009)

MOR: Monitoring and Measurements through the Onion Router

Demetris Antoniades[1], Evangelos P. Markatos[1], and Constantine Dovrolis[2]

[1] Institute of Computer Science
Foundation for Research & Technology Hellas
{danton,markatos}@ics.forth.gr
[2] College of Computing, Georgia Institute of Technology
dovrolis@cc.gatech.edu

Abstract. A free and easy to use distributed monitoring and measurement platform would be valuable in several applications: monitoring network or server infrastructures, performing research experiments using many ISPs and test nodes, or checking for network neutrality violations performed by service providers. In this paper we present MOR, a technique for performing distributed measurement and monitoring tasks using the geographically diverse infrastructure of the Tor anonymizing network. Through several case studies, we show the applicability and value of MOR in revealing the structure and function of large hosting infrastructures and detecting network neutrality violations. Our experiments show that about 7.5% of the tested organizations block at least one popular application port and about 5.5% of them modify HTTP headers.

1 Introduction

A common request of researchers, administrators and simple users, is easy access to a number of geographically distributed machines. Such access would facilitate experimentation and better understanding of several network configurations, or checking for network neutrality violations. In this extent a freely available, distributed monitoring and measurement platform would be of great value.

For many years now researchers have been using the Planetlab [10] infrastructure for conducting distributed experiments. Planetlab offers access to machines located in many different educational institutions around the world. Through a Unix-like system it allows its users to run experimental code on these machines. In this way, researchers are able to run distributed programs in many different locations, and check the network communication of their applications in real network environments. Being a (mostly) educational infrastructure, Planetlab omits commercial networks and Internet Service Providers (ISPs) with very different policies, configurations and infrastructures. Furthermore, access to Planetlab is limited to researchers. Administrators wanting to check recent configuration changes, and end users aiming at checking the quality of the service they pay for, lack a geographically distributed system freely available for this kind of daily experiments.

In this paper we present MOR, a technique for performing geographically distributed monitoring and measurement experiments. MOR utilizes the infrastructure of the Tor anonymity network [12]. Tor is a free software aiming to protect the privacy of its users, by directing users' traffic through a distributed infrastructure. This infrastructure is built by voluntarily deployed nodes in organizations, institutions and homes. To support our idea we provide a proof-of-concept implementation of such a monitoring and measurement technique. Using our technique we examine a number of case studies that show the applicability and value of MOR. The provided case studies range from examining the structure and function of large hosting infrastructures and detecting network neutrality violations. Our main contributions can be summarized as follows:

- *We propose a technique for performing large-scale distributed measurements using the Tor anonymity network.*
- We *demonstrate the feasibility of our technique* by providing a *proof-of-concept implementation*, and provide several different *use cases*.
- We explore the extent of *port blocking* by various organizations over the Globe. Our results show that 11 out of 149 tested organizations (7.5%) block outgoing connections on ports of widely used services such as ftp(21), ssh(22) and telnet(23).
- We explore the extent to which organizations *alter HTTP headers*, and find that 9 out of 166 tested organizations (5.5%) alter, suppress or add HTTP headers.
- We explore the extent of *Skype blocking* by organizations. Interestingly enough, we find one ISP which consistently blocks Skype.

The rest of the paper is organized as follows: Section 2 gives a small introduction to Tor. Section 3 shows how we can use Tor in order to perform distributed network monitoring. We provide a number of case studies for our proposed technique in Section 4. Finally, we place our work in the appropriate context by presenting the related work in Section 5 and we conclude the paper with a discussion on the advantages and disadvantages of our approach in Section 6.

2 The Tor Network

Tor [12] is currently the most widely deployed anonymous communication system, with an estimation of more than 100,000 daily users around the globe [16]. These users range from ISP clients, military and company employees, to journalists and law enforcement officers [15]. Used mainly for web traffic, Tor is carefully designed in order to provide anonymity for low latency services.

Tor is based on the idea of Onion Routing [14], with the approach having its roots in the idea of Mix Networks proposed by Chaum [9] in early 1980s. Onion Routing is built on the concept that a message from a source to a destination will first travel via a sequence of arbitrary selected proxies (*Onion Routers*). In Tor this sequence (*circuit*) is selected at random when a connection request is received (*stream*). The last node in the circuit, called an *Exit node*, is the one that will perform the actual communication with the service of interest on behalf of the user. Before the source node transfers any message to the system, it will first encrypt it with the public key of all the intermediate proxies, creating a succession of layers like an *"onion"*. Any intermediate node will then decrypt the message, with its private key, and pass it on to the next node of the circuit. When the Exit node decrypts the message, it will have the actual request data and will be able

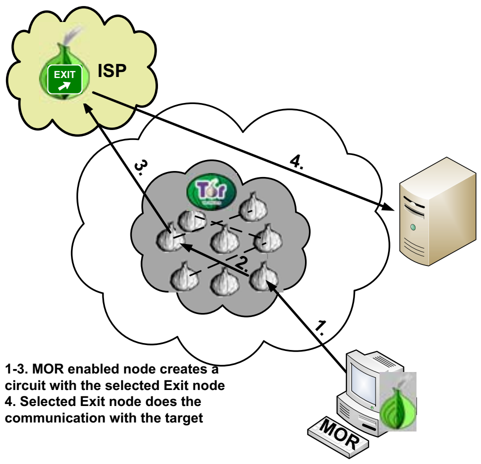

ISP
EXIT
1.
2.
3.
4.
MOR
1-3. MOR enabled node creates a circuit with the selected Exit node
4. Selected Exit node does the communication with the target

Fig. 1. Basic steps for routing experimental traffic through the Tor network

to proceed by communicating with the requested service. The response will follow the same procedure (onion creation and "un-peeling") and reverse path back to the client.

In order for Tor to succeed in providing acceptable and efficient anonymity for its users, it needs to create an overlay with a large number of proxies. In this way, latency is minimized due to the even distribution of the load to the proxies, and anonymity is improved due to the wide combination of nodes for building circuits [11]. From the information provided by the directory servers of Tor about 1600 nodes were connected to the overlay in mid September'09. Almost half of the nodes (660) were accepting to forward traffic to the outer Internet, functioning as *Exit nodes*. These nodes were located in 48 different countries all over the word and registered by 312 different Autonomous Systems (AS). Such a large and geographically diverse overlay implicitly offers free access to the Internet through a large number of different operational networks in an easy and publicly available form. The work presented in this paper gives a first, to our knowledge, approach of using Tor for performing monitoring and measurements tasks.

3 Using Tor as a Monitoring and Measurement Platform

Participation in the Tor network is freely available and minimal effort is needed to install and configure a proxy. Though, to be able to perform experiments using Tor's infrastructure one needs to properly instrument circuit creation and connection attaching. Fortunately, the Tor community provides extended documentation for the proxy control protocol and a python library for instrumenting the proxy [22].

For any given application we want to run through Tor a basic sequence of steps has to be followed. First we select the "experimental-node-set", a proper set of Exit nodes fitting the requirements of the application. Consider a Web-based application, the experimental-node-set will exclude all Exit nodes blocking outgoing traffic to port 80 in their configuration policy. After we have this experimental-node-set we follow the sequence of steps shown in Figure 1, iterating over each different Exit node in the set. First we create a circuit with the selected Exit node (steps 1-3). Since Tor does not allow for single-node circuits, this circuit includes at least one additional proxy. This intermediate proxy can be arbitrary selected, though we prefer to select a stable (based on its given status) router, to minimize any interference to our experiment. After the circuit is properly created, we proceed with creating the data socket for handling the required communication stream. On the first packet, the stream is attached to the previously created circuit. After these two steps are successfully completed, all traffic of the data stream is routed through the selected Exit node and our monitoring application can proceed as it would in the absence of Tor. Thus, it will send any data and wait for the response. The target host can be any online host in the Internet able to respond to our request, or a controlled host in a laboratory that would be instrumented to respond to our requests and/or log any incoming traffic from the Tor network for post analysis.

4 Case Studies

In this section we present a number of applications, that show the applicability and value of using Tor as an experimental platform. In our case studies we use MOR to reveal the structure and function of large hosting infrastructures and to detect network neutrality violations.

4.1 Examining Content Replication in a One-Click Hosting Service

Our first case of interest comes from the need to understand how files are replicated and served by *www.rapidshare.com*: one of the largest One-Click Hosting Services (OCH). OCH services enable users to upload and download very large (100-200 MBytes) files for a very low cost. Their low cost, dependable service and very high capacity, make OCH services very popular within file sharing communities. Thus, recently, OCH services have been used as an alternative to peer-to-peer file-sharing systems. From a technical point of view, OCH services can be considered as Content Distribution Networks (CDNs). However, OCH services host mostly large files, while CDNs, in addition to large files, host lots of small files, such as objects in web pages.

Since the files shared by users of *rapidshare.com* are several MBytes large, our interest is to understand how the service replicates and serves each file to its users. To explore this, we access the same file from many different locations (Tor Exit nodes) in order to derive the actual server(s) that provide the file each time. In our experiments, described in detail in [2], we build a list with more than 20,000 *Rapidshare* URLs, available on the Web, and repeatedly requesting them for download, by a group of 421 different *Exit nodes*. In this way, if server selection is done based on the client's IP address the address of the Exit node would be used by the decision algorithm.

Our experiments show that, although we observe more than 5,000 Rapidshare server IP addresses in total, *each* file is hosted only by *exactly* 12 *Rapidshare* servers. Our download attempt is always redirected to a single indexing server providing us with download URLs that point to 12 different servers hosting the actual file. Furthermore, we observed no ISP specific policy decisions, since all download requests for a specific URL where (almost) equally distributed among the 12 download servers.

Table 1. Checked port numbers

Description	Port	Blocked (%)	Description	Port	Blocked (%)	Description	Port	Blocked (%)
Sun-RPC	111	13.16	MS-SQL	1434	7.97	Telnet	23	7.38
IMAP	143	6.70	MySQL	3306	6.52	Unreal-Game	7777	6.52
Netmeet	1503	6.47	Shiva-VPN	2233	6.47	SNMP	161	6.43
FW1-VPN	259	6.43	Netmeet	1720	6.43	Bay-VPN	500	6.38
SSH	22	6.36	Skype	5060	6.34	FTP	21	6.33
DNS-Xfer	53	6.25	IMAPS	993	5.98	HTTP	80	5.83
HTTPS	443	5.73	POP3	110	5.43			

4.2 Network Neutrality

An important discussion regarding human rights on network access is the one related to network neutrality. Internet users expect neutral treatment of their traffic from their provider, regardless the application protocol, port number or content they aim to access. In that extent, we use MOR to present a number of use cases able to infer whether a user is receiving neutral treatment from her network provider. Note, though, that use of the described experiments is not limited to end users, but an administrator can also exploit the same setup to test network or firewall configurations.

Port Blocking. Our first case study, regarding network neutrality, explores whether an organization blocks outgoing traffic from specific port numbers to the global Internet. We use a MOR client issuing access requests (TCP-SYN) to a controlled machine, located in our organization, for a number of different TCP ports. The controlled machine logs all incoming traffic using *tcpdump*, and is set out of the firewall of our organization, thus able to receive and respond to any incoming request.

In our experiments we use the port numbers defined as *"Ports of Interest"* in [7]. These ports span a large number of applications (web, p2p, e-mail, games, chat etc.). The full list of ports we use, and the description of each port, are depicted in Table 1. We probe each port 10 times from 236 different Tor *Exit nodes*. We consider a port to be blocked by an organization only if no TCP-SYN was received in any of the 10 tries.

Column *"Blocked"* of Table 1 shows the percentage of nodes that seem to be blocking each tested port number. Note that we currently have no indication whether the blocking is done by the Exit node hosting organization or somewhere in the path between that organization and our controlled machine. This kind of identification is left for future work. In all cases at least 5% of the nodes employ port blocking. From the results, we can see the largest percentage of blocking (more than 7%) to be for ports that are prone to Internet attacks, like MS-SQL and MySQL default ports (1434 and 3306 respectively) and Sun-RPC (111). Furthermore, it is interesting to see that a number of the tested organizations (more than 6%) block access to widely used services such as ftp (21), ssh (22), and telnet (23). We speculate that port blocking is done both for security considerations (ssh, telnet) and traffic discrimination (skype). Figure 2 plots the number of organizations (ASNs) exploiting port blocking per country.

As a further step, we examined the same port numbers using the Planetlab infrastructure [21, 10]. We use 191 different Planetlab nodes, again sending 10 TCP-SYN

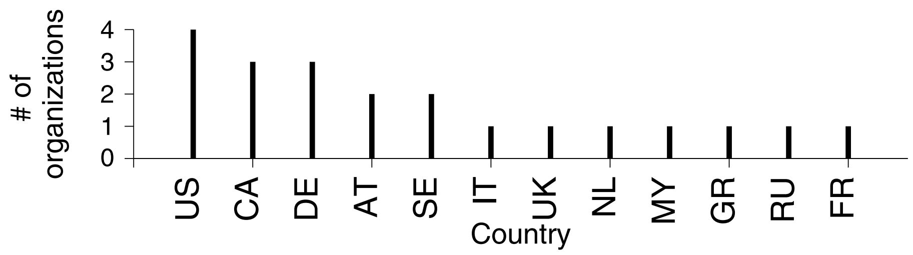


Fig. 2. Number of port blocking organizations per country

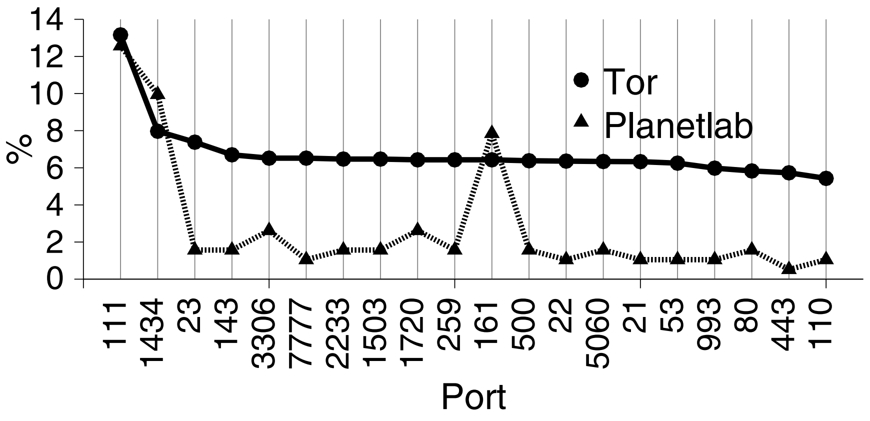


Fig. 3. Port blocking comparison between Tor and Planetlab

connection requests for each port. Figure 3 shows the comparison between the two infrastructures. We can observe a similar percentage of blocking only in three port numbers (111, 1434 and 161), which are considered prone to Internet attacks. In all other cases we observe a larger percentage of blocking in the case of the Tor overlay. Comparing the autonomous system numbers (ASN) hosting Planelab and Tor nodes we found only 10 common ASNs. Most of the organizations hosting the planetlab node were universities and research institutions. On the other hand, the percentage of academic organization in the Tor ASNs was less than 2%. Thus, MOR, by utilizing the Tor infrastructure, provides access to nodes located in commercial providers.

HTTP Header Suppression. In the next use case, we use MOR to study the suppression of *HTTP Headers* performed by different organizations. As shown in [5] most of the time *HTTP Header* suppression comes from the network and not the browser, since some organizations may use intermediate proxies, and add or remove some headers, for caching, security and privacy reasons.

In our setup, we use a monitored machine, running an Apache HTTP server on port 80. Our MOR client issues HTTP requests for a simple web page hosted in our server. The initial *HTTP Header* contained in the request is shown in Figure 4. The HTTP

```
'Accept-language': 'en-us'
'Accept-encoding':
  'gzip, deflate, compress;q=0.9'
'Host': '139.91.70.22'
'Accept': 'text/html,
  application/xhtml+xml,
  appplication/xml;q=0.9,*/*;q=0.8'
'User-agent': 'Mozilla/5.0 (X11; U;
  Linux i686; en-US; rv:1.9.0.3)
  Gecko/2008092510 Ubuntu/8.04
  (hardy) Firefox/3.0.3'
'Accept-charset':
  'iso-8859-1,utf-8;q=0.7,*;q=0.7'
'Connection': 'Close'
'Referer': '139.91.70.81'
'Cache-control': 'max-age=0'
```

Fig. 4. Actual Request HTTP Header

```
'Content-Length': '175'
'Accept-Ranges': 'bytes'
'Server': 'Apache/2.2.3 (Debian)
  DAV/2 SVN/1.4.2
  mod˙python/3.2.10 Python/2.4.4
  PHP/5.2.0-8+etch13
  mod˙ssl/2.2.3 OpenSSL/0.9.8c
  mod˙perl/2.0.2 Perl/v5.8.8'
'Last-Modified':
 'Sat, 11 Apr 2009 10:05:05 GMT'
'Connection': 'close'
'etag': '"44540-9e-9d84b640"'
'Date':
 'Sat, 18 Apr 2009 12:34:24 GMT'
'Content-Type': 'text/html;
 charset=UTF-8'
```

Fig. 5. Actual Response HTTP Header

server always responds with the header shown in Figure 5.[1] We used tcpdump to capture both the traffic sent and received from our client to the Tor proxy and also the traffic to and from the Web server.

Our experimental-node-set contains 166 different exit nodes. In the largest percentage of Exit Nodes we observed no difference in both request and response headers. In 5.5% of the cases, though, we had suppressed headers, addition of extra header fields and in some cases responses without even accessing the Web server, probably due to caching of a previous identical request from another Exit Node in the same network.

In most cases the altered, added or suppressed field reveals the existence of a proxy, either used as a centralized HTTP access point for an organization or for content caching purposes. In such cases we observe altering of the *"Connection"* header field in the request header, and addition of the *"Via"* header field in the response header. Furthermore, we observe cases where the organization completely removes the *"Referer"* or *"User-agent"* header, probably due to privacy policies. The organizations for which we observe alteration in the HTTP header fields are located in several countries, namely France, Germany, Argentina, USA, Canada and China.

As a further step we run our Web server on an arbitrary port number (20000). This experiment investigates whether an organization uses Deep Packet Inspection (DPI) to recognize any HTTP traffic, or whether the identification is based only on well known port numbers. In our results no altering or suppression was observed when accessing the server in a different port. Thus we can say that all used organizations do not employ DPI for HTTP traffic altering. As future work we plan to extend this study to identify traffic discrimination for file-sharing applications, like BitTorrent, through DPI.

Skype Censorship. Another issue of interest, regarding network neutrality violations, arises when an organization is restricting access or limiting the performance of an Internet application, based on the port numbers used, employing DPI techniques or specific host blocking. As an example, in this case study, we explore the extent to which organizations block access to the Skype IP telephony application.

[1] Note that in case of a difference in the HTTP Request the server will also respond differently.

Skype utilizes a p2p infrastructure to connect its clients. When the Skype application starts it first communicates with a centralized login server (`ui.skype.com`) which will verify the user's credentials and allow her to log in to the p2p network. After the log in phase, the user's Skype traffic, either chat, voice or video, is transferred through the application's p2p overlay [6].

The login phase is done over the HTTP protocol by requesting a URL from the server. The URL contains the hashed user credentials and information about the running version of the application. We use this login method in order to identify organizations that block Skype users from logging into the system. For our experiments we extracted the URL from the latest version of a Linux Skype client.[2] We request the URL from Skype's login server through Tor, using 171 different Exit nodes. For every Exit node we request the URL 10 times and log the response result. We consider the organization not to be blocking access to Skype if we receive at least one valid response from the server. Our connection requests were 100% unsuccessful in only one case. This Exit node is hosted by a Kuwait ISP which has also been reported by others to block Skype.[3]

4.3 Further Possible Use Cases

Web-Page Censorship: Recent work has shown that a number of web clients receive altered pages during their browsing sessions [19]. These alterations may include advertisements, extra javascript code and even malware, that are either annoying (in the best case) or harmful to the user. Using our technique one can compare the page she receives from a Web server, with the pages received when the server is visited from a different geographical location and/or ISP.

Network Problems Diagnosis: Using MOR one can easily detect if an administered or desired service is working properly. For example a user can check if a service is non responsive also from other organizations or only from it's own network in order to report this to her administrators. Also online service administrators can check the visibility and correct functioning of their service when viewed from external networks.

DNS Update Speed: Using a SOCKS4a proxy one can direct DNS queries through the Tor network. Combined with our technique one can measure the time needed for a domain name update to become visible by the rest of the Internet.

5 Related Work

With Tor being increasingly popular during the last years, a number of researchers examined the network, targeting its performance [20], attacking the system [8, 18]. or trying to compromise its users' anonymity through traffic analysis [13, 17].

[2] http://ui.skype.com/ui/2/2.0.0.72/en/getlatestversion?ver=2.0.0.72&uhash=1074a31ab9146cc11ab149c86a32dc920

[3] http://www.248am.com/mark/kuwait/skype-blocked-by-qualitynet/

The goal of performing a variety of distributed experiments, led a number of researchers to use overlay networks from a different aspect than the one initially intended. Athanasopoulos *et al.* in [3] used the Gnutella overlay network to perform Distributed Denial of Service attacks on third party services. In a subsequent work the same authors illustrated the use of Gnutella in anonymously downloading a Web file [4]. Close to our work, Beverly *et al.* in [7] used the Gnutella Network to quantify the prevalence of port blocking from ISPs and institutions. In their setup, a super-peer in the Gnutella network was instrumented to redirect each contacting client to a specific port of a measurement host controlled by the authors. Recently, Barth *et al.* in [5] used advertisement networks to study the use and suppress of the *HTTP Referer* field. They used two advertisement networks to display custom advertisements to the users for 3 days. When the advertisement was displayed in the user's Web browser, it issued a number of HTTP requests to two servers controlled by the authors. Their observation showed that most of the times, the *Referer* HTTP field was suppressed in the network and not in the browser.

6 Discussion, Limitations and Conclusions

In this paper we propose a new technique for performing measurement and monitoring tasks. We propose the use of the Tor anonymizing network as a geographically distributed infrastructure. Using our proof-of-concept implementation, MOR, we present a number of case studies that demonstrate the applicability and value of our approach. Our experiments show that about 7.5% of the tested organizations block at least one popular application port and about 5.5% of them modify HTTP headers.

While our work actually leverages the Tor network, the applications we propose make careful use of the network adding limited overhead (i.e. single TCP-SYN packets and small Web requests). We expect MOR to act as a motivation for a number of users, interested in measurement and monitoring tasks, in adding more relays to the network. In this case, Tor will benefit from MOR users, since more proxies will increase the network's geographic diversity, improve anonymity and Tor's overall performance [1].

Limitations: Unfortunately, with the current state of the Tor network, a number of interesting tasks can not be implemented. Two main limitations are the lack of relaying non-TCP traffic and the limited throughput performance. Tor does not, for the time being, support relaying non-TCP traffic. In this extent a number of programs (i.e. traceroute) that use other IP protocols, can not be relayed through Tor. Due to this, a number of experiments based on this type of tools are not feasible. Furthermore, though Tor tries, and succeeds, to provide low latency anonymization, it is still not able to support high throughput applications. In this case experiments targeting on identifying path delay, loss and average/peak throughput are not guaranteed to provide accurate results. It is highly possible that the measured metric will be affected by the system itself and will not correspond to the actual value from the targeted network. Though these are fundamental limitations on the number of possible use cases of our technique, we believe that our work will encourage exploration for integrating the aforementioned metrics.

References

1. Acquisti, A., Dingledine, R., Syverson, P.: On the Economics of Anonymity. In: Wright, R.N. (ed.) FC 2003. LNCS, vol. 2742, pp. 84–102. Springer, Heidelberg (2003)
2. Antoniades, D., et al.: One-Click Hosting Services: A File-Sharing Hideout. In: Proceedings of the ACM/SIGCOMM conference on Internet Measurements (November 2009)
3. Athanasopoulos, E., Anagnostakis, K., Markatos, E.: Misusing Unstructured P2P Systems to Perform DoS Attacks: The Network That Never Forgets. In: Proceedings of the 4th International Conference on Applied Cryptography and Network Security (June 2006)
4. Athanasopoulos, E., et al.: GAS: Overloading a File Sharing Network as an Anonymizing System. In: Proceedings of the 2nd International Workshop on Security (2007)
5. Barth, A., Jackson, C., Mitchell, J.: Robust Defenses for Cross-Site Request Forgery. In: Proceedings of the 15th ACM conference on Computer and Communications security (2008)
6. Baset, S., Schulzrinne, H.: An Analysis of the Skype Peer-To-Peer Internet Telephony Protocol. In: IEEE International Conference on Computer Communications (2006)
7. Beverly, R., et al.: The Internet's Not a Big Truck: Toward Quantifying Network Neutrality. In: Proceedings of the 8th Passive and Active Measurement Workshop (April 2007)
8. Borisov, N., Danezis, G., Mittal, P., Tabriz, P.: Denial of Service or Denial of Security? How Attacks on Reliability can Compromise Anonymity. In: Proceedings of the 14th ACM Conference on Computer and Communication Security (October 2007)
9. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 24(2) (1981)
10. Chun, B., et al.: Planetlab: an overlay testbed for broad-coverage services. SIGCOMM Comput. Commun. Rev. 33(3), 3–12 (2003)
11. Dingledine, R., Mathewson, N.: Anonymity Loves Company: Usability and the Network Effect. In: Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006) (June 2006)
12. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the Second-Generation Onion Router. In: Proceedings of the 13th conference on USENIX Security Symposium (2004)
13. Evans, N., Dingledine, R., Grothoff, C.: A Practical Congestion Attack on Tor Using Long Paths. In: Proceedings of the 18th USENIX Security Symposium (August 2009)
14. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Hiding Routing Information. In: Proceedings of Information Hiding: First International Workshop (May 1996)
15. K. Loesing. Measuring The Tor Network: Evaluation of Client Requests to the Directories (2009), http://git.torproject.org/checkout/metrics/master/report/dirreq/directory-requests-2009-06-26.pdf
16. McCoy, D., et al.: Shining Light in Dark Places: Understanding the Tor Network. In: Proceedings of the 8th International Symposium on Privacy Enhancing Technologies (July 2008)
17. Murdoch, S.J., Danezis, G.: Low-Cost Traffic Analysis of Tor. In: Proceedings of the 2005 IEEE Symposium on Security and Privacy (May 2005)
18. Pappas, V., et al.: Compromising Anonymity Using Packet Spinning. In: Proceedings of the 11th Information Security Conference (September 2008)
19. Reis, C., et al.: Detecting In-flight Page Changes with Web Tripwires. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (2008)
20. Snader, R., et al.: A Tune-up for Tor: Improving Security and Performance in the Tor Network. In: Proceedings of the Network and Distributed Security Symposium (February 2008)
21. Spring, N., Peterson, L., Bavier, A., Pai, V.: Using PlanetLab for Network Research: Myths, Realities, and Best Practices. SIGOPS Oper. Syst. Rev. 40(1), 17–24 (2006)
22. The Tor Project. TorCtl., https://svn.torproject.org/cgi-bin/viewvc.cgi/torctl/

Evaluating IPv6 Adoption in the Internet

Lorenzo Colitti, Steinar H. Gunderson, Erik Kline, and Tiziana Refice

Google, Inc.
{lorenzo,sesse,ek,tiziana}@google.com

Abstract. As IPv4 address space approaches exhaustion, large networks are deploying IPv6 or preparing for deployment. However, there is little data available about the quantity and quality of IPv6 connectivity. We describe a methodology to measure IPv6 adoption from the perspective of a Web site operator and to evaluate the impact that adding IPv6 to a Web site will have on its users. We apply our methodology to the Google Web site and present results collected over the last year. Our data show that IPv6 adoption, while growing significantly, is still low, varies considerably by country, and is heavily influenced by a small number of large deployments. We find that native IPv6 latency is comparable to IPv4 and provide statistics on IPv6 transition mechanisms used.

1 Introduction

The network protocol that has been used in the Internet since its inception is IPv4 [1], which provides 2^{32} distinct addresses. Its successor IPv6 [2] provides 2^{128} addresses, but IPv6 adoption has not proceeded as quickly as its designers expected [3]. Since IPv6 is not backward-compatible with IPv4 both clients and servers have to deploy IPv6 to make use of it; since IPv6 provides little immediate benefit apart from larger address space the operational community has seen little motivation to deploy it. However, as IPv4 address exhaustion approaches [4], a number of networks have deployed IPv6 or are preparing for deployment.

One of the problems faced by an organization, especially a Web site operator, wanting to deploy IPv6 is the lack of information on IPv6 adoption and the quality of service provided by the IPv6 Internet. Thus, it is difficult to predict the adoption and impact to existing services of even a small-scale IPv6 roll-out. Conventional wisdom in the operational community is that low adoption leads to lower quality of service, since problems are less likely to be noticed, and lower quality of service leads to low adoption, as a network with lower quality of service is less desirable. However, little data is available to validate these assumptions. The situation is further complicated by the wide variety of mechanisms currently used to provide IPv6 connectivity. To work around an initial lack of IPv6 deployment, various transition mechanisms were developed to provide IPv6 connectivity on IPv4-only networks, often by encapsulating IPv6 traffic in IPv4. Examples are configured tunnels [5], 6rd [6] and ISATAP [7], which require operator-controlled intermediate nodes; and 6to4 [8] and Teredo [9], which can use third-party relay nodes that may be anywhere on the Internet. An IETF survey [10] lists over 30 documents covering a multitude of scenarios.

Attempts have been made to quantify IPv6 adoption in various ways. Past work has counted IPv6 addresses observed at 6to4 relays [11,12] or proxies [13], though this can only observe the users of the relay examined. Arbor Networks [14] sampled IPv6 traffic on backbones, but since current routers generally do not support IPv6 packet sampling, they could only observe tunneled traffic and were unable to conclude how much native traffic was present. Karpilovsky et al.[15] compared IPv6 address allocations published by Internet registries to BGP routing tables, finding that allocations were growing, but that many prefixes were used either long after allocation or not at all. They found that IPv6 traffic within an undisclosed tier-1 ISP was negligible, and mostly consisted of DNS and ICMP. However, tier-1 status in IPv4 does not necessarily imply a substantial IPv6 deployment; in fact, many IPv4 tier-1 ISPs currently only have a handful of IPv6 customers—for example, on 6 October 2009, AT&T (AS7018) only had four IPv6 BGP customers according to publicly-available BGP tables [16,17]. They also looked at Teredo traffic, but as we note later, this is prone to undercounting since many implementations prefer IPv4 over Teredo. Zhou et al. [18] compared IPv6 and IPv4 one-way delay between 26 measurement points, finding that IPv6 latency exhibited larger variation, and was significantly higher for 36% of paths. Huston [3] analyzed logs from the RIPE and APNIC Web sites, finding that IPv6 connectivity was available to about 1% of their users. These Web sites are primarily targeted at network operators, who are likely to have more access to IPv6 connectivity than general Internet users. Kevin Day [19] used 1x1 pixel images on a Web page to track working IPv6, finding growth from 0.014% to 0.084% between December 2005 and March 2008, and broken IPv6, which varied between 0.2% and 0.4%. Wikimedia [20] conducted similar measurements between June 2008 to July 2009. A software package [21] is also available for Web site operators to measure working and broken IPv6 for their user base. However, to our knowledge there is no comprehensive study that examines IPv6 deployment from the perspective of a Web site operator or that can serve to quantify the impact of adding IPv6 support to a large Web site with a broad user base.

In this paper we present a methodology for characterizing IPv6 adoption, connectivity, and latency from the perspective of a Web site operator (Section 3). We apply the methodology to the Google Web site to conduct a large-scale study of IPv6 deployment. Our data (Section 4) helps answer questions such as: what percentage of users would use Google's services over IPv6 if it were enabled? What would be the impact on reliability and latency? What is the extent of IPv6 deployment in various countries and networks, and which transition mechanisms are used?

2 Browser Behaviour

Predicting the impact of enabling IPv6 on a Web site requires an understanding of browser behaviour in a multi-protocol environment, as the availability of two protocols creates a choice as to which protocol to use. Since IP addresses are inconvenient to remember, network applications such as Web browsers typically allow users to input names (e.g. `www.google.com`), and use the Domain

Name System (DNS) to resolve names to IP addresses. The DNS uses A and AAAA *Resource Records* (RRs), respectively, to specify IPv4 addresses (e.g., `74.125.19.99`), and IPv6 addresses (e.g., `2001:4860:b006::68`). It also employs distributed caching of RRs based on the *Time-To-Live* (TTL) of each RR; when the TTL expires, a new RR must be requested. Web browsers supporting both IPv4 and IPv6 generally request both types of RRs; most recent operating systems allow limiting the requests to only those protocols that are available at query time. Browsers then typically attempt to connect to all the returned addresses in order, retrying with the next address if an attempt fails or times out [22]. The exact order is application and operating-system dependent [23], but virtually all IPv6-capable browsers will prefer native IPv6 to IPv4; some implementations will prefer IPv4 to tunneling mechanisms, some will not. Although this fallback behaviour can be used as a crude high-availability solution, content providers typically use *Virtual IP addresses* (VIPs), backed by multiple Web servers, instead of relying on browser behaviour.

3 Measurement Methodology

Our measurement methodology is based on asking Web clients to send HTTP requests to either an IPv4-only host or a dual-stack host and comparing the results. For this purpose, we use two hostnames (the *ExpHostNames*):

- `dualstack.ipv6-exp.l.google.com`: a dual-stack hostname, i.e. a hostname with both AAAA and A DNS RRs, (the ExpHostName$_D$), and
- `ipv4.ipv6-exp.l.google.com`: an IPv4-only hostname, i.e. a hostname with an A DNS RR only (the ExpHostName$_4$).

The ExpHostNames correspond to a set of VIPs (the *ExpVIPs*), which are load-balanced among a set of Web Servers, (the *ExpWSes*) in geographically distinct datacenters. Each datacenter has a pair of ExpVIPs, one IPv4 and one IPv6. To allow comparison of IPv4 and IPv6 results, we configure the Google DNS servers so that DNS requests from a given resolver (and thus from a given client) will return ExpVIPs in the same datacenter. We set a low TTL value (5 seconds) on the DNS records for both ExpHostNames. This ensures that most requests require DNS lookups, minimizing latency differences due to caching. In September 2008, we set up ExpVIPs in two datacenters (one in the US, one in Europe). Between March 2009 and June 2009, we added three more (one in the US, one in Europe, and one in Asia). In June 2009, the Asian ExpVIPs were turned down for unrelated reasons. Since then we have been using four datacenters.

To cause Web clients to connect to the ExpVIPs, we modify the results returned for a small, randomly-selected fraction of Google search requests, which we name *SearchRequests*. Fig. 1 describes the measurement process. Specifically, when a SearchRequest is processed by a Web Server (the *SearchWS*):

1. SearchWS adds to the search results a JavaScript fragment which instructs the browser to fetch a URL (the *ExpURL*) containing an ExpHostName.

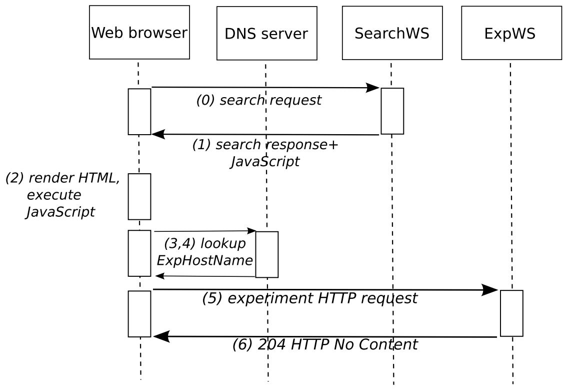


Fig. 1. Measurement Request Flow

2. If the browser executes the Javascript code, it will:
   (a) Execute a DNS lookup for the ExpHostName (either A or AAAA+A)
   (b) Receive an IPv4 and/or an IPv6 ExpVIP (depending on which ExpHostName and which DNS records were requested)
   (c) Send an HTTP request (the *ExpRequest*) to the ExpVIP
3. If ExpWS receives the ExpRequest, it logs it and returns to the browser a HTTP `204 No Content` response.

The URL loaded by the JavaScript contains the following elements:

ExpHostName. 10% of the SearchRequests are sent to ExpHostName$_4$, while 90% are sent to ExpHostName$_D$. We arbitrarily chose the 90/10 split because we expected the vast majority of hosts to connect over IPv4, and wanted to collect as much IPv6 data as possible.
SearchClientIP. The IPv4 or IPv6 address of the client, as seen by SearchWS.
SearchTS. The timestamp of the SearchRequest, as measured by SearchWS.
HashCode. A cryptographic hash of the other elements which can be used to verify that the ExpRequest is genuine.

The ExpURL is constructed as follows:
http://*ExpHostName*/gen_204?ip=*SearchClientIP*&ts=*SearchTS*&auth=*HashCode*

4 Data Analysis

Data collection started in September 2008 and is still ongoing. In this paper we present data collected up to September 2009. Each ExpRequest logged by an ExpWS is termed a *hit*. We receive order of millions of hits per day. Hits received on ExpHostName$_D$ are termed *dual-stack hits*; of these, hits received over IPv6 are termed *IPv6 hits*. In every hit, we examine the following fields in addition to the fields in the ExpURL:

ExpClientIP. The IP address which sent the ExpRequest.
ExpTS. The timestamp of when ExpWS processed the ExpRequest.
UserAgent. The User-Agent header in the ExpRequest; identifies the browser.

Logs are processed once per day, using MapReduce [24] and Sawzall [25] for efficiency. Before performing data analysis, we exclude hits from Google's own networks as well as invalid and duplicate hits, as follows. First, in order to reduce the likelihood of users altering their or others' requests (e.g., by arbitrarily delaying IPv6 requests, or by sending IPv6 and IPv4 requests through different hosts), we exclude all hits with an invalid HashCode. The number of such hits is negligible. Second, to avoid duplicate hits due, for example, to replay attacks or users clicking on the browser's back button and causing a second ExpRequest to be issued, we exclude all but the first hit with a given HashCode in a given day. To avoid duplicates across days, we discard all hits which are logged more than *MaxResponseTime* = 5 minutes after the corresponding SearchRequests. If a request does not succeed within 5 minutes, we consider that client not to have working IPv6, as we consider this is not an acceptable response time for a Web page. This does not significantly bias our data, because – as can be seen in Fig. 8 – about 94% of all hits are received in the first 3.5 seconds.

By analyzing the hits in our dataset, we infer various statistics such as the clients that can or cannot connect to dual-stack Web servers and their request latency. The remainder of this section describes these statistics.

IPv6 Connectivity Ratio. To measure the availability of IPv6 connectivity among Google users, we count the number of IPv6 hits. Every IPv6 hit implies that the client that sent it has *working IPv6*. A SearchRequest will not result in a hit in at least the following cases:

1. The browser has JavaScript disabled or does not accept third-party images.
2. The user navigates away from the result page, loses connectivity, or closes the browser before the ExpURL has been loaded.
3. The browser attempts to contact the dual-stack host ExpHostName$_D$, but cannot reach it within MaxResponseTime. In this case, we say that the client has *broken IPv6*. Since our measurement methodology does not allow us to distinguish this case from the others, we do not present data on broken IPv6.

We define the *working IPv6 ratio* W as the number of IPv6 hits divided by the total number of hits to the dual-stack host: $W = \frac{n_{D6}}{n_{D4} + n_{D6}} = \frac{n_{D6}}{n_D}$, where n_{D4} and n_{D6} are, respectively, the number of IPv4 and IPv6 hits to the dual-stack host and n_D is their total. Since W is computed by sampling a constant value (the percentage of IPv6 connectivity) using a fixed number of independent trials, we treat it as being binomially distributed. Also, as shown by Fig. 2, n_D is very large compared to n_{D6}. Thus, we can approximate the binomial as a normal distribution and calculate the standard deviation of W as: $\sigma_W = \sqrt{W(1-W)/n_D}$.

As an example, during the week of 2009-09-20, about 0.252% of all dual-stack hits were IPv6, and $\sigma_W = 0.001\%$. σ_W is of the same order of magnitude for every week in our dataset. In the following, we use weekly numbers (i.e. we aggregate all the hits for one week) unless otherwise noted. If we assume the average number of searches per day made by a given user does not depend on whether the user has IPv6 connectivity or not, we can use W as an estimator of the ratio of Google users with working IPv6.

Connectivity Over Time. Fig. 2 plots the value of W over time since we started collecting data. Although the percentage is still low, it has grown significantly in the last 12 months. For example, for the week of 14 September, year-over-year growth is approximately 35%. We attribute the dip in IPv6 connectivity in August to seasonal variations: since – as we see in Fig. 6 – IPv6 deployment is heavily influenced by a small number of countries, IPv6 traffic is more prone to the effects of holiday seasons than IPv4 traffic. W varies during the course of a week and is substantially higher during the weekends. An example is Fig. 3, which shows September 2009. This suggests that IPv6 is more available to users at home than in their workplace.

Connectivity by Type. To infer the type of IPv6 connectivity used by clients, we examine the ExpClientIP address of each IPv6 hit, as described in [13,15]. This allows us to distinguish 6to4, Teredo, and ISATAP hits. Other hits, which include both native traffic and other transition mechanisms such as configured tunnels, cannot be distinguished based on IP address alone. As shown in Fig. 4, 6to4 is the most common connectivity type, while ISATAP and Teredo are comparatively rare. Note that this is not an indication of what type of connectivity is available to users, but what type of connectivity would be used to connect to dual-stack Web sites. For example, implementations such as Windows Vista prefer IPv4 over Teredo and 6to4, and thus we will not observe them (Section 2).

Connectivity by Operating System. We infer the operating system of the client based on the HTTP User-Agent header of each IPv6 hit. We only take into account OSes that, during September 2009, had a significant number of IPv6 hits (i.e. $W^{os} \geq 1\%$ of all IPv6 hits). As shown in Fig. 5, most IPv6 hits are from MacOS and Windows Vista clients. This highlights the importance of operating system defaults. In fact, the number of IPv6 hits from Windows XP (which is not IPv6-enabled by default) is approximately three times lower than those from Windows Vista (which is IPv6-enabled by default), even though Windows XP's market share is approximately three times higher [26]. Further analysis of the data shows that most MacOS IPv6 hits use 6to4 ($\simeq 90\%$, while all the other OSes have $< 50\%$). We do not know the reason for this. We are unable to infer the operating system for 0.082% of IPv6 hits.

Connectivity by Country. To determine IPv6 availability and connectivity types in different countries, we geolocate the ExpClientIP using internal geolocation databases. We then consider the ratio $W_{ct}^{cc} = n_{6ct}^{cc}/n_D^{cc}$ between IPv6 hits using connectivity type ct from country cc and all dual-stack hits from cc. We only take into account countries accounting for at least 1% of all IPv6 hits in September 2009. As shown in Fig. 6, there are both significant differences in IPv6

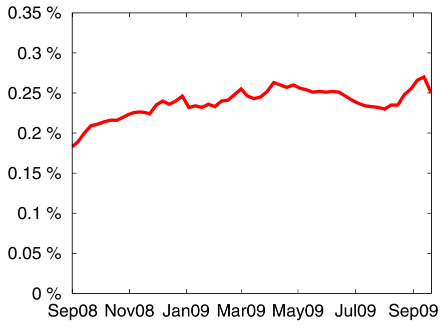


Fig. 2. Working IPv6 over time

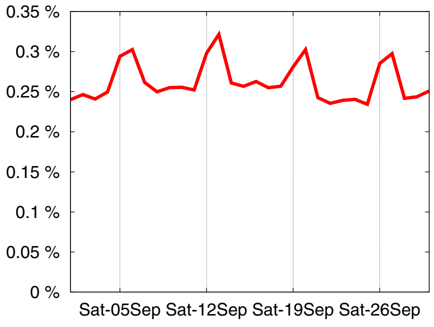


Fig. 3. Daily working IPv6 in Sep 2009

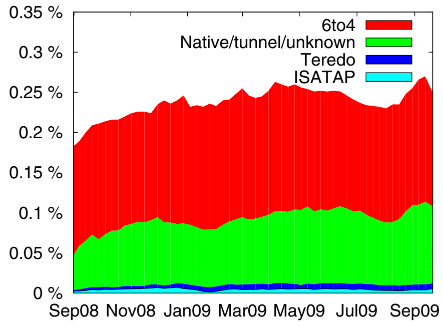


Fig. 4. Working IPv6 by connectivity type

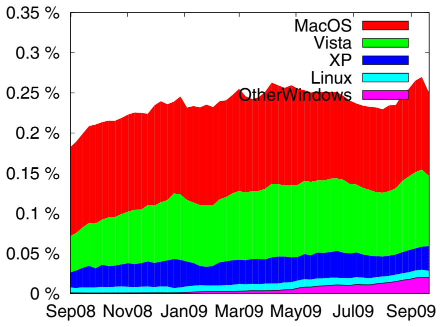


Fig. 5. Working IPv6 by operating system

adoption (almost an order of magnitude) and in connectivity types between countries. We note that availability of IPv6 connectivity in a given country does not necessarily correlate with IPv6 deployment, since relayed transition mechanisms such as 6to4 and Teredo can be enabled by users in the absence of IPv6 network infrastructure. A better measure of the deployment of IPv6 in a given country can be obtained by removing relay mechanisms. Fig. 7 shows that the most significant deployments of IPv6 are in France and China. The high proportion of 6to4 in Russia and Ukraine may be due to the Opera browser, which prefers 6to4 over IPv4; our internal data shows that it is popular in those countries.

Connectivity by AS. We then determine the Autonomous Systems which originate IPv6 hits by looking up the ExpClientIP in the BGP routing tables of Google's routers. Then, for each AS, we compute the working IPv6 ratio $W^{AS} = n_{D6}^{AS}/n_{D}^{AS}$, where n_{D}^{AS} and n_{D6}^{AS} are, respectively, the number of dual-stack hits and IPv6 hits (excluding 6to4 and Teredo) coming from IP addresses originated by that AS. Table 1 presents the top ASes with higher W^{AS} (during September 2009). We filter out ASes that contribute only marginally to the experiment, i.e. with $n_{D6}^{AS} < 1\%$ of all the IPv6 hits in the same period. Only

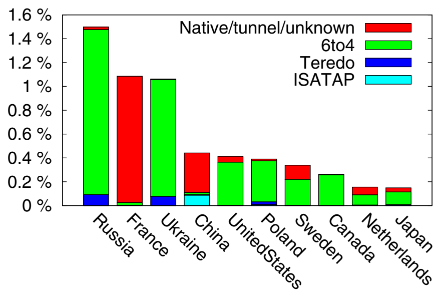


Fig. 6. Working IPv6 ratio for top-10 countries by connectivity type

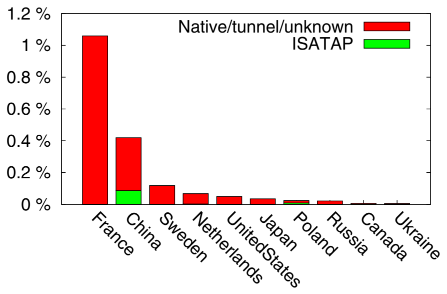


Fig. 7. Working IPv6 ratio for top-10 countries, non-relayed only

Table 1. IPv6 connectivity per origin AS vs dual-stack connectivity for the same AS

ASN	AS Name	Country	W^{AS}
37944	CSTNET	CN	100.000%
23910	CERNET2	CN	99.962%
19782	Indiana Uni.	US	89.325%
1312	Virginia Tech	US	51.743%
6122	ICN	US	13.449%
12322	Free	FR	5.131%
4538	CERNET-BKB	CN	2.650%

Table 2. IPv6 connectivity per origin AS versus overall IPv6 connectivity

ASN	AS name	Country	W^{AS}_{D6}
12322	Free	FR	54.956%
23910	CERNET2	CN	7.160%
1312	Virginia Tech	US	2.472%
4538	CERNET-BKB	CN	2.001%
19782	Indiana Uni.	US	1.428%
6122	ICN	US	1.178%
37944	CSTNET	CN	1.038%

7 ASes match this criterion. Moreover, in order to identify ASes with higher impact on overall IPv6 traffic, we also compute $W^{AS}_{D6} = n^{AS}_{D6}/n_{D6}$, where n_{D6} is the total number of IPv6 hits (excluding 6to4 and Teredo) in September 2009. Table 2 shows the top ASes with higher W^{AS}_{D6} (during September 2009). Note the ASes in both tables are the same. 6 out of 7 ASes shown in Table 1 and 2 are universities or research institutions. The only exception - Free (AS12322) - contributes to most of the French IPv6 native hits (presented in Fig. 7).

IPv6 vs IPv4 Latency. In order to compare the latency of HTTP queries performed over IPv4 and IPv6, we consider the latency of an ExpRequest (the *ExpLatency*) as Δt = ExpTS − SearchTS and we aggregate hits into 50 ms buckets. Fig. 8 plots latency computed since 18th June 2009. The graphs in Fig. 8 account for approximately 94% of all hits (the rest are received after the 3.5 second cut-off). As we can see, ExpRequests sent to ExpHostName$_4$ and to ExpHostName$_D$ on IPv4 have almost identical latency. Either these clients did not attempt to request an IPv6 record or any added latency of doing so was insignificant. Thus, assigning a Web site an IPv6 address in addition to an IPv4 address did not affect the latency of IPv4-only clients in any measureable way.

We also see that *IPv6 latency* (i.e. latency of requests to ExpHostName$_D$ over IPv6) is in general slightly higher than IPv4 latency. However, if we exclude

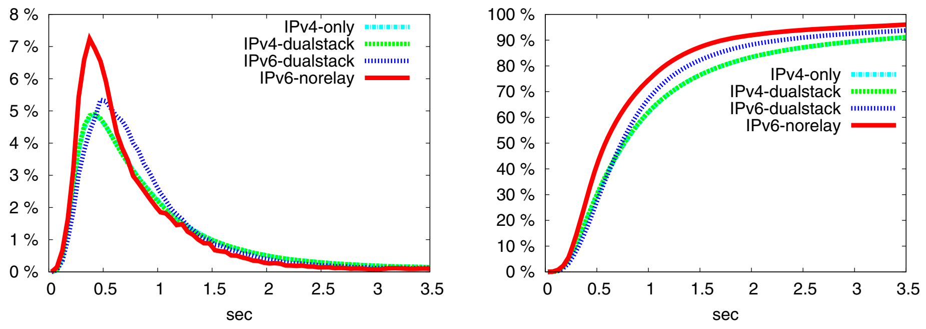


Fig. 8. PDF and CDF of hit latency. Time granularity of 50 ms. The IPv4-only and IPv4-dualstack plots are indistinguishable. The latency data are not indicative of ordinary Google service latency.

relayed IPv6 connectivity (i.e., 6to4 and Teredo), IPv6 latency is actually lower than IPv4 latency. This is likely due to the fact that, as shown in Table 2, IPv6 deployment is heavily dominated by research and education networks and one large broadband ISP. We would expect these networks to have higher-bandwidth, lower-latency connections than average.

5 Conclusions and Future Work

We have presented methodologies that allow us to characterize several aspects of IPv6 adoption, connectivity, and quality. We have applied them to a large data set, and shown that IPv6 deployment is small but growing steadily, and that adoption is still heavily influenced by a small numer of large deployments. While we see IPv6 adoption in research and education networks, IPv6 deployment is, with one notable exception, largely absent from consumer access networks. We find that native IPv6 latency is not significantly worse than IPv4, but transition mechanisms such as 6to4 and Teredo can add noticeable latency, perhaps because relays can be very far away from the users they serve. Finally, we have shown that the default settings of operating systems and applications factor strongly in the level of IPv6 adoption seen on those platforms.

We believe that our methodology usefully characterizes properties of connectivity in the IPv6 Internet and intend to continue our measurements to provide a baseline as adoption grows. We also plan to conduct further measurements to quantify the incidence of broken IPv6 and its causes.

References

1. Information Sciences Institute: Internet Protocol. RFC 791 (1981)
2. Deering, S., Hinden, R.: Internet Protocol, Version 6 (IPv6). RFC 2460 (1998)

3. Huston, G.: IPv6 Transition, http://www.potaroo.net/presentations/2009-09-01-ipv6-transition.pdf
4. Huston, G.: IPv4 Address Report, http://www.potaroo.net/tools/ipv4/
5. Nordmark, E., Gilligan, R.: Basic Transition Mechanisms for IPv6 Hosts and Routers. RFC 4213 (2005)
6. Després, R.: IPv6 Rapid Deployment on IPv4 infrastructures (6rd), http://tools.ietf.org/html/draft-despres-6rd
7. Templin, F., Gleeson, T., Thaler, D.: Intra-Site Automatic Tunnel Addressing Protocol (ISATAP). RFC 5214 (2008)
8. Carpenter, B., Moore, K.: Connection of IPv6 Domains via IPv4 Clouds. RFC 3056 (2001)
9. Huitema, C.: Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380 (2006)
10. IPv6 Operations Working Group charter, http://ietf.org/dyn/wg/charter/v6ops-charter.html
11. Savola, P.: Observations of IPv6 Traffic on a 6to4 Relay. In: SIGCOMM CCR (2005)
12. Hei, Y., Yamazaki, K.: Traffic Analysis and Worldwide Operation of Open 6to4 Relays for IPv6 Deployment. In: Symposium on Applications and the Internet (2004)
13. Malone, D.: Observations of IPv6 Addresses. In: Claypool, M., Uhlig, S. (eds.) PAM 2008. LNCS, vol. 4979, pp. 21–30. Springer, Heidelberg (2008)
14. Arbor Networks: Tracking the IPv6 migration. Technical report (2008)
15. Karpilovsky, E., Gerber, A., Pei, D., Rexford, J., Shaikh, A.: Quantifying the Extent of IPv6 Deployment. In: Moon, S.B., Teixeira, R., Uhlig, S. (eds.) Passive and Active Network Measurement. LNCS, vol. 5448, pp. 13–22. Springer, Heidelberg (2009)
16. RIPE NCC: Routing Information Service, http://www.ris.ripe.net/
17. Hurricane Electric: IPv6 BGP table, http://ipv6.he.net/bgpview/bgp-table-snapshot.txt
18. Zhou, X., Mieghem, P.V.: Hopcount and E2E Delay: IPv6 Versus IPv4. In: Dovrolis, C. (ed.) PAM 2005. LNCS, vol. 3431, pp. 345–348. Springer, Heidelberg (2005)
19. Day, K.: Working vs. Broken IPv6 Clients (2008), http://your.org/v6clients.png
20. Wikimedia: IPv6 Deployment Status, http://wikitech.wikimedia.org/view/IPv6_deployment
21. Ward, N.: IPv6 WWW Test, http://www.braintrust.co.nz/ipv6wwwtest/
22. Hagino, J.: Implementing AF-Independent Application (1998), http://www.kame.net/newsletter/19980604/
23. Draves, R.: Default Address Selection for Internet Protocol version 6 (IPv6). RFC 3484 (2003)
24. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI (2004)
25. Pike, R., Dorward, S., Griesemer, R., Quinlan, S.: Interpreting the Data: Parallel Analysis with Sawzall. Scientific Programming Journal (2005)
26. Wikipedia: Usage Share of Desktop Operating Systems Retrieved on 2009-10-02

Internet Usage at Elementary, Middle and High Schools: A First Look at K-12 Traffic from Two US Georgia Counties

Robert Miller[1], Warren Matthews[2], and Constantine Dovrolis[1]

[1] Georgia Institute of Technology
robert.miller@gatech.edu,dovrolis@cc.gatech.edu
[2] JANET
warren.matthews@ja.net

Abstract. Earlier Internet traffic analysis studies have focused on enterprises [1,6], backbone networks [2,3], universities [5,7], or residential traffic [4]. However, much less is known about Internet usage in the K-12 educational system (elementary, middle and high schools). In this paper, we present a first analysis of network traffic captured at two K-12 districts in the US state of Georgia, also comparing with similar traces collected at our university (Georgia Tech). An interesting point is that one of the two K-12 counties has limited Internet access capacity and it is congested during most of the workday. Further, both K-12 networks are heavily firewalled, using both port-based and content-based filters. The paper focuses on the host activity, utilization trends, user activity, application mix, flow characteristics and communication dispersion in these two K-12 networks.

1 Introduction

K-12 networks are unique for several reasons. First, they are used primarily by a very specific part of the population: children and adolescents. Second, these networks are mostly used for educational purposes, as opposed to business, research or entertainment. Third, the conventional wisdom at least is that K-12 networks are often under-provisioned in terms of Internet access capacity, experiencing congestion during most of the working day. Fourth, again based on conventional wisdom, K-12 networks are tightly controlled in terms of allowable applications and downloadable content.

Our objective in this paper is to analyze K-12 Internet traffic so that we can better understand how the Internet is used in these unique networks. Which are the dominant applications? What is the diurnal utilization pattern? Are there significant differences between say elementary schools and high schools? How does congestion affect usage, and in particular, the flow size distribution or the per-flow throughput? Further, we would like to examine the previously mentioned conventional wisdoms and understand the differences between K-12 traffic and the more often studied university traffic.

The Schools: We have collected data from two K-12 districts (counties) in the state of US Georgia: Barrow and Walton. These districts are geographically close and of similar size but they have very different Internet access capacities. Barrow is connected through a 150Mbps link to PeachNet (the education network of the state of Georgia), while Walton has a 20Mbps connection to a commercial provider. Barrow's access link has plenty of available capacity, while Walton is heavily congested during the school day.

The Barrow network is used by approximately 12,000 students and teachers at 16 schools (3 high, 4 middle and 9 elementary schools). Barrow also has 3 administrative facilities that use its network. The Walton network is used by approximately 13,000 students and teachers at 13 schools (2 high, 3 middle and 8 elementary schools). In both counties, the networks are subnetted based on schools and as such we can identify the school that each IP flow belongs to. Both networks are NAT-ted. In Barrow, our monitor is located inside the private network, and so we can identify individual hosts. In Walton, on the other hand, our monitor is located after the NAT and so we cannot identify individual hosts (even though we can still identify individual schools because each school uses a different public IP address).

To compare K-12 traffic with the more often studied university traffic, we have also collected network traces at Georgia Tech. The Georgia Tech network has several 1Gbps access links, and it is used by approximately 20,300 students and faculty. Further, Georgia Tech, as most US universities, does minimal filtering of application ports or content. While we are able to compare the K-12 traffic to the Georgia Tech data in some instances, in others we could not do so. This is mostly due to limitations on our data collection at Georgia Tech.

The Data: Data was captured using port-mirroring at the central switch of both K-12 networks. The data was stored in *nfdump* files rotated every 5 minutes.[1] In this paper, we only present nfdump data from the week of April 14-20 2008, which was a typical week for both counties in terms of usage and school operation. For Georgia Tech, we analyzed netflow data from the access router collected on September 8, 2008.

We also used a packet sniffer to extract various HTTP headers and the DNS-query field of DNS requests. That data was collected on September 8, 10 and 21, 2008 only at Walton county.

The structure of the paper is as follows. In section 2 we describe the broad characteristics of each network. In section 3 we give the breakdown of captured traffic in terms of protocols and applications. In section 4 we compare flow-level characteristics between the schools. We conclude in section 5.

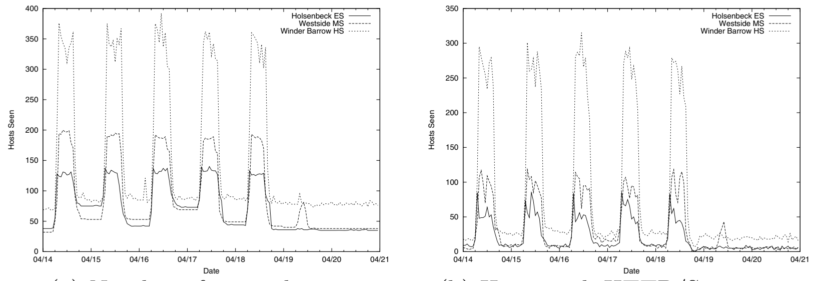
2 Network Characteristics

Host Count and Activity: We first estimate the number of network-connected hosts at Barrow County. We cannot do the same for Walton because of the

[1] nfdump is a tool that collects and processes netflow data via the command line. It is part of the NfSen project: http://nfsen.sourceforge.net/

Table 1. Maximum number of hosts seen at each subnet. ES stands for elementary school, MS for middle school and HS for high school.

School	Max Hosts Seen	School	Max Hosts Seen
Barrow County	2970	ELC	38
Auburn ES	108	Russell MS	481
Bethlehem ES	143	Westside MS	205
Bramlett ES	139	Window Barrow MS	201
County Line ES	186	Apalachee HS	583
Holsenbeck ES	159	Winder Barrow HS	465
Kennedy ES	172	PLC	143
Statham ES	209	Others	32
Yargo ES	162		



(a) Number of active hosts (b) Hosts with HTTP/S activity

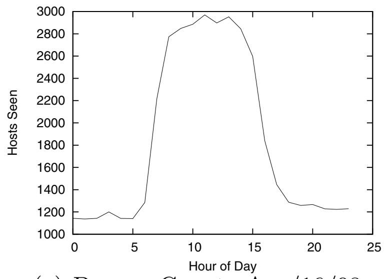
Fig. 1. Hourly activity of Barrow hosts during a week

previously mentioned NAT issue. Table 1 shows the maximum number of distinct hosts seen at each subnet (school) in a single day.[2] Note that the two high school networks tend to be larger in terms of hosts than middle and elementary schools.

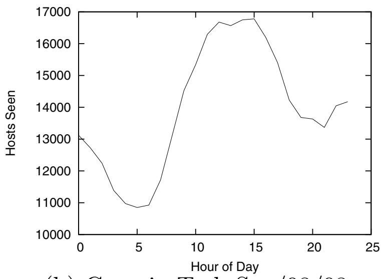
Next, we focus on the diurnal pattern of the number of hosts that are turned-on. We assume that such hosts will be generating/receiving some traffic (e.g., for network management reasons) if they are connected to the network. Figure 1(a) shows the hourly progression of hosts over the course of the week for three representative schools. We chose to display one school of each type (ES, MS, HS). As expected, the number of active hosts increases during school days, from about 7am to about 5pm. What is also interesting, however, is that a significant fraction of hosts (20% to 40%) are turned-on during evenings and weekends. Most likely, not all of these machines are servers. This indicates that these machines have been left on during off hours and weekends, probably without reason.

We are also interested in the number of visible hosts due to user-initiated activity. It is not easy to infer whether a machine is currently used by a human or

[2] Russell Middle School has a few special machines (mostly servers) assigned to its subnet, causing some unusual results. The "Others" category represents machines that do not belong to any school in particular.



(a) Barrow County Apr/16/08



(b) Georgia Tech Sep/08/08

Fig. 2. Number of active hosts

not. The heuristic that we use is to detect whether a host generates or receives any HTTP or HTTPS (denoted as HTTP/S) traffic during that time period, assuming that most (but clearly not all) HTTP traffic is due to user-initiated web browsing. Figure 1(b) shows the number of HTTP/S-active hosts during a week, in hourly intervals. Note that the activity at Holsenbeck ES and Westside MS is close to zero during the evening hours, but the same is not true for Winder Barrow HS. This may be due to hosts running Web sessions with periodic page-refreshes.

It is interesting to compare host activity between a K-12 and a university network. Figures 2(a) and 2(b) show the number of hosts seen at Barrow county and at the Georgia Tech network in hourly intervals over the course of a day. We see a very different pattern. At Georgia Tech, the number of active hosts remains at high levels during the evening hours, until midnight or so, as many students and faculty work during after-hours at the school or from home. Also, we see again that a large fraction of hosts (about 65%) remains turned-on and network-active during the evenings. This is an issue that large organizations will have to address if we are to reduce power dissipation and energy demands.

Network Utilization: Figure 3 shows how the utilization varies over the course of a weekday at Walton, Barrow and Georgia Tech, in five-minute intervals. We show two curves in each graph, for incoming and outgoing traffic. Walton is congested, with over 90% utilization of its access link, from about 8am until about 3-4pm. Further evidence of Walton's congestion is evident in the RTTs (not shown here) between Georgia Tech and the monitoring machine at Walton. During peak hours, the RTT reaches 300ms, up from around 8ms during off hours. The peak load at Barrow is about 45Mbps, much below the 150Mbps capacity. During peak hours in Barrow county we do not see significant RTT fluctuation, with the RTT staying around 2ms over the course of the day. Also, the two counties are mostly consumers of Internet traffic; the outgoing traffic rate (mostly DNS and HTTP requests as well as outgoing email) is only 11% of the incoming rate.

Georgia Tech's diurnal usage pattern, on the other hand, is very different. It generates an almost symmetric traffic load between the incoming and outgoing directions. As we will see in the next section, this is probably because Georgia

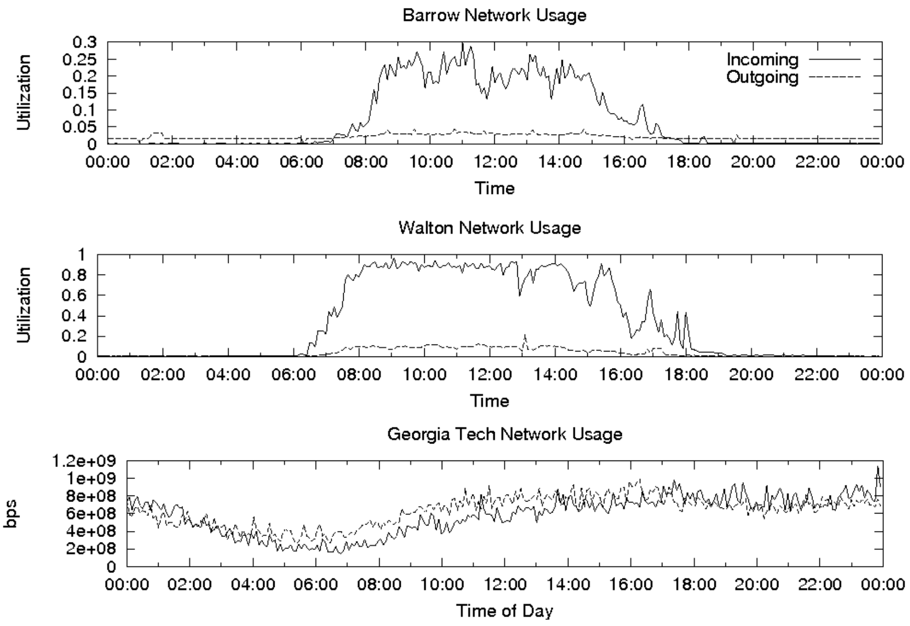


Fig. 3. Network load variations during a working day

Tech acts as a significant HTTP content provider (through multiple research and software distribution servers) and because it allows the activity of peer-to-peer (p2p) applications. Further, the load variations at Georgia Tech during the day are much smaller than in K-12 networks: the minimum traffic load (200Mbps at about 6am) is 20% of the maximum traffic load (1Gbps in afternoons and evenings)[3]. On the other hand, the traffic load at K-12 networks is almost zero in the late evening hours.

3 Traffic Characteristics

Protocol and Application Breakdown: We first examine the breakdown of traffic in terms of transport protocol. The main protocols in both K-12 networks are TCP and UDP covering together more than 99% of the bytes. TCP dominates the transport layer, with almost 95-96% of the packets and 97-100% of the bytes at Walton county. The UDP percentages are much higher at Barrow, but this is due to a single IP address at Russell MS that multicasts a CNN video stream using UDP. If we exclude Russell from the analysis, Barrow is also dominated by TCP traffic, with similar numbers as Walton. On the other hand, the data from Georgia Tech shows a significantly larger fraction of UDP traffic, about 11-13% of packets and 4-5% of bytes.

[3] Georgia Tech's true capacity was undisclosed. Therefore we present only the measured bps instead of utilization.

Table 2. Application layer breakdown (outgoing data/incoming data)

	HTTP	HTTPS	RTMP	SMTP	RTSP	Unknown
Barrow Packets	74.1%/79.5%	8.1%/6.9%	5.0%/5.4%	6.5%/3.2%	2.4%/2.5%	2.4%/1.7%
Barrow Bytes	66.5%/82.9%	12.2%/4.6%	1.6%/6.7%	11.4%/0.9%	0.8%/2.7%	3.4%/1.4%
Walton Packets	75.2%/79.6%	6.8%/6.1%	2.7%/1.9%	6.7%/4.2%	1.4%/1.9%	6.6%/5.6%
Walton Bytes	74.2%/85.9%	10.9%/5.3%	1.0%/4.2%	8.9%/0.3%	0.5%/2.2%	4.1%/1.9%

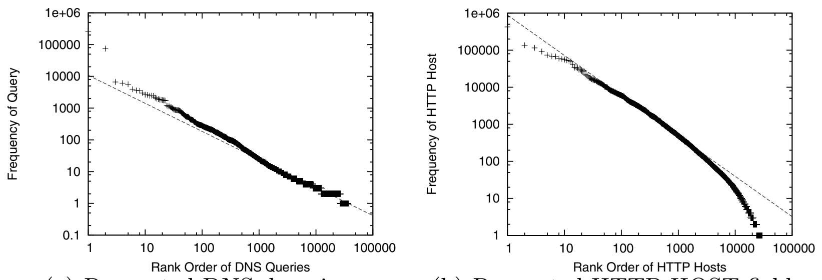
	Unknown	HTTP	rsync	DNS	NNTP	RTMP
Georgia Tech Packets	42.8%/37.5%	36.2%/37.3%	5.8%/3.6%	3.1%/4.9%	1.0%/1.8%	1.0%/2.0%
Georgia Tech Bytes	43.9%/33.2%	36.4%/49.3%	10.5%/0.6%	0.5%/0.7%	0.1%/3.9%	0.1%/3.82%

We next examine the application breakdown. We use a simple port-based classifier. It is well-known that this classifier is inaccurate because it fails to detect p2p or other applications that do not use well-known port numbers [5]. However, as will be shown next, this is not an issue for these K-12 networks because they block most traffic, excluding traffic from well-recognized port numbers such as HTTP or HTTPS.

Table 2 shows the application breakdown at the two K-12 networks, as well as at Georgia Tech, both for outgoing and incoming traffic. The major application-layer protocols at Barrow and Walton are HTTP/S, SMTP (email), RTMP (Real-Time Messaging Protocol, a proprietary Adobe protocol for media streaming using a Flash player) and RTSP (Real-Time Streaming Protocol, used by media clients to control remote media servers with VCR-like capabilities). HTTP/S dominates, with about 80-90% of the packets and bytes in both directions. Of course we should be aware that some applications use the HTTP/S port numbers today to "disguise" as Web browsing. Unfortunately, we have no way to detect such applications. It is interesting that the RTMP/RTSP percentages are higher for Barrow than Walton. This may be due to the heavy congestion at Walton. Streaming is more sensitive to congestion, and if streaming applications do not perform well, people would use them less frequently. Finally, note that the percentage of unidentified traffic is quite low, typically less than 5% of the bytes. This is not surprising, given that the administrators at the two K-12 networks block all ports except those that are explicitly white-listed.

The application breakdown is very different at Georgia Tech. In that case, the percentage of "Unknown" traffic is significant, 35-45% of the packets/bytes in both directions, with slightly more outgoing traffic. Although we cannot be certain using port-based classification, we expect that most of that traffic is generated by p2p applications such as BitTorrent. HTTP/S generates roughly the same traffic volume as "Unknown" even though the fraction of incoming traffic in bytes is significantly higher than outgoing traffic. Other significant protocols at Georgia Tech are rsync (synchronization of remote file systems), DNS, NNTP and RTMP.

DNS Requests: We have also captured DNS-requests (only at Walton county) in order to characterize the domains that K-12 hosts request most frequently. Two domains were the most popular by far: walton.k12.ga.us (about 43% of



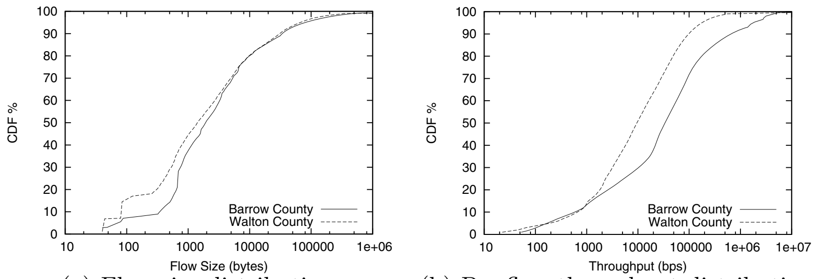
(a) Requested DNS domains (b) Requested HTTP HOST fields

Fig. 4. Rank-order distributions on log-log scale with the associated regression curves

the requests) and akamai.net (about 12%). This is not surprising. The Walton domain is so popular because it is probably the default web page at many hosts in that network. Akamai is the largest CDN and the web pages of their customers includes objects with Akamai DNS names. Other popular DNS domains are Google, Yahoo, AOL, MSN, Photobucket, the advertising domains llnwd.net and doubleclick.net, and nsatc.com, a domain that powers many of Microsoft's services. The top-10 domains requested capture 61% of the total requests, while the top-100 domains capture 72% of the requests. This implies that a significant fraction of the requested domains are at the tail of the distribution. Indeed, Figure 4(a) shows the frequency-versus-rank plot, in log-log scale, for the requested DNS domains. The linear trend indicates a Zipf distribution, with exponent -0.88 and R^2=96%.

HTTP Headers: We also collected HTTP headers at Walton, focusing on the Host, Content-Length and Content-type fields of the HTTP header. When we examined the median content length downloaded at Walton over the course of a day, we noticed a significant increase in the size of downloaded HTTP objects shortly after midnight. This may be due to automated software updates. In terms of Content-type, the most popular types are gif and jpeg images, followed by html, javascript and flash. We plan to compare these measurements with Barrow, when we become able to collect HTTP headers from that network.

The HTTP Host field allows us to measure the most popular Web servers the users of these K-12 networks request. The three most popular servers are Windows updates, Trend Micro, and Google. Trend Micro is an anti-virus company. The top-10 servers in the list make up 25% of all the HTTP requests, while the top-100 hosts make up 52%. We have also examined how these distributions differ between elementary schools and high schools. In the former, we see a strong presence on websites hosting educational games. In high schools, Google and its various services dominate. We have also examined the popularity distribution of HTTP servers (see Figure 4(b)) and it also follows a Zipf distribution, but with a truncated tail. If we only consider the top 2000 HTTP servers, we get a much better fit to the Zipf model (exponent=-1.09, R^2=99%).



(a) Flow size distribution (b) Per-flow throughput distribution

Fig. 5. Flow size and throughput distributions at Barrow and Walton

4 Flow Characteristics

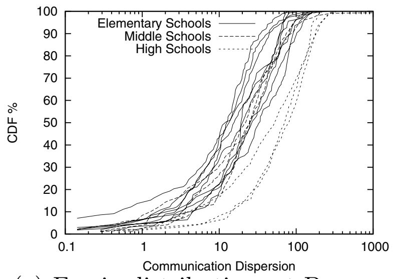
In this section, we focus on the flow size distribution and the per-flow throughput in the two K-12 networks. In particular, we are interested in examining how congestion at Walton affects these two important flow characteristics.

Flow Sizes and Throughputs: Figure 5(a) shows the flow size distribution for Barrow and Walton during a working day. It is interesting that the two distributions are very similar, especially for larger flow sizes (more than 10KB) despite the fact that Walton experiences severe congestion. This observation implies that users do not react to congestion by downloading smaller files, as one may expect. Instead, it seems that they download the same files that they would download if they had more capacity.

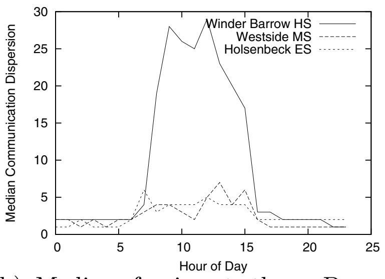
The difference between the two distributions in smaller flow sizes, however, may be a result of congestion. In detail, we observe that the fraction of small flows (less than 10KB) is higher at Walton. This may be due to aborted flows: users often abort a transfer when it takes too long to start (due to packet losses in the TCP connection establishment or slow-start phase, for instance). The increased frequency of very small flows, compared to Barrow, is an interesting difference that we plan to further investigate.

We also examined how Walton's flow size distribution varies during the busy hours of a 9-hour working day (from 7am till 4pm). However, this distribution showed that Walton had very similar flow sizes over the course of the work day, which is not surprising given that the network is congested during this entire 9-hour period.

Another interesting characteristic is the per-flow throughput distribution. Of course we expect much lower throughput at Walton than Barrow. Indeed, Figure 5(b) shows the corresponding distribution functions. Note that the median throughput is 33.8 kbps at Barrow and 9.5 kbps at Walton. The 90-th percentile of the throughput distribution, which may be more indicative of large-transfers, is 665.8 kbps at Barrow and only 97.7 kbps at Walton. Looking at Walton's throughput variation over the course of a day again showed us that the distribution changes very little over the course of a school day.



(a) Fan-in distributions at Barrow



(b) Median fan-in at three Barrow schools on April 16, 2008

Fig. 6. Communication dispersion measured based on HTTP fan-in

In summary, the results of this section indicate that congestion affects the per-flow throughput but not the size distribution of downloaded files. It is possible that users adapt to congestion by downloading files in the background, or by simply being more patient as they browse the Web. We plan to investigate this issue in more depth in the future.

Communication Dispersion: Another interesting aspect of traffic analysis is the number of outside hosts that each internal host at these K-12 networks communicates with. Here, we are primarily interested in HTTP traffic and in the incoming direction of traffic (mostly downloads). Specifically, for each host at Barrow we count the number of external HTTP servers that send traffic to that host. We refer to that number as the "fan-in" of that host. We cannot do the same for Walton due to the previously discussed NAT issue.

Figure 6(a) shows the fan-in distribution for each school within Barrow county. We label the curves based on the type of school. It is interesting that high schools have significantly higher fan-in than middle or elementary schools. This difference may be indicative of the larger diversity of content and sites that high school students (mostly adolescents) prefer, compared to the younger students (mostly children) at elementary and middle schools. This difference is further illustrated in Figure 6(b), where the median fan-in of three representative schools is shown as function of time. Note that there are no statistically significant differences between the ES and the MS, but the HS fan-in is 5-6 times larger.

5 Ongoing Work

We have recently started collecting data from more Georgia counties. We plan to expand our analysis in terms of the number of schools and the duration of the study. We are also trying to further understand the effects of congestion on user behavior and application performance.

Acknowledgments

The authors would like to thank Ed Morrison, John St.Clair and Kevin McCage at Barrow County Schools and Todd Antwine and Jon Graves at Walton County Schools for hosting the monitoring servers. We'd also like to thank Claudia Huff and the F3 group for their ongoing support and assistance. The Georgia Tech data used en this work were made available by the Georgia Tech Research Network Operations Center (www.rnoc.gatech.edu).

References

1. Aiello, W., Kalmanek, C., McDaniel, P., Sen, S., Spatscheck, O., Van der Merwe, J.: Analysis of communities of interest in data networks. In: Passive and Active Network Measurement (2005)
2. Fomenkov, M., Keys, K., Moore, D., Claffy, K.: Longitudinal study of Internet traffic in 1998-2003. In: ACM International Conference Proceeding Series (2004)
3. Fraleigh, C., Moon, S., Lyles, B.: Packet-level traffic measurements from the Sprint IP backbone. IEEE Network (2003)
4. Fukuda, K., Cho, K., Esaki, H.: The impact of residential broadband traffic on Japanese ISP backbones. In: ACM SIGCOMM (2005)
5. Karagiannis, T., Broido, A., Faloutsos, M., Claffy, K.C.: Transport layer identification of P2P traffic. In: ACM SIGCOMM (2004)
6. Pang, R., Allman, M., Bennett, M., Lee, J., Paxson, V., Tierney, B.: A first look at modern enterprise traffic. In: Internet Measurement Conference (2005)
7. Smith, F.D., Campos, F.H., Jeffay, K., Ott, D.: What TCP/IP protocol headers can tell us about the web. In: ACM SIGMETRICS (2001)

A First Look at Mobile Hand-Held Device Traffic

Gregor Maier, Fabian Schneider, and Anja Feldmann

TU Berlin / Deutsche Telekom Laboratories
Ernst-Reuter-Platz 7, 10589 Berlin, Germany
{gregor,fabian,anja}@net.t-labs.tu-berlin.de

Abstract. Although mobile hand-held devices (MHDs) are ubiquitous today, little is know about how they are used—especially at home. In this paper, we cast a first look on mobile hand-held device usage from a network perspective. We base our study on anonymized packet level data representing more than 20,000 residential DSL customers. Our characterization of the traffic shows that MHDs are active on up to 3 % of the monitored DSL lines. Mobile devices from Apple (i. e., iPhones and iPods) are, by a huge margin, the most commonly used MHDs and account for most of the traffic. We find that MHD traffic is dominated by multimedia content and downloads of mobile applications.

Keywords: Mobile Devices, iPhone, Traffic Characterization.

1 Introduction

Today advanced mobile hand-held devices (MHDs, e. g., iPhones and BlackBerrys) are very popular. MHDs have evolved rapidly over the years—from pure offline devices, to cell phones with GSM data connectivity, to 3G devices, and universal devices with both cellular as well as WiFi capabilities. Their increased graphics and processing power makes these devices all-in-one PDAs and media centers. Today's MHDs can be used to surf the Web, check email, access weather forecast and stock quotes, and navigate using GPS based maps—to just name some of the prominent features. This increase in flexibility has caused an increase in network traffic. Indeed, cellular IP traffic volume is growing rapidly and significantly faster than classic broadband volume [15].

We, in this paper, cast a first look at Internet traffic caused by mobile hand-held devices. We use anonymized residential DSL broadband traces, spanning a period of 11 month, to study MHD behavior and their impact on network usage. We are thus able to observe the behavior of MHDs when they are connected via WiFi at home and compare their traffic patterns to the overall residential traffic characteristics. Some devices (most notably iPod touch and iPhone) require WiFi connectivity rather than cellular connectivity for some services. Other services are more likely to be used via cellular connectivity due to user mobility, e. g., looking up directions on Google Maps, while walking around town or driving. Although, we in this paper only focus on residential MHD usage and not MHD usage in cellular networks, our analysis gives first insights into what kind of services users are interested in when they are at home and have access to all services. This information is crucial for 3G cellular providers to anticipate usage patterns and future traffic growths.

The remainder of this paper is structured as follows. In Sec. 2 we present our data sets and methodology, Sec. 3 presents our results. In Sec. 4 we discuss related work before we conclude our paper in Sec. 5.

2 Data and Methodology

In this section we describe the anonymized data sets of residential DSL connections and our methodology for analyzing them.

2.1 Data Sets

We base our study on multiple sets of anonymized packet-level observations of residential DSL connections collected at aggregation points within a large European ISP. The monitor, using Endace monitoring cards, operates at the broadband access router connecting customers to the ISP's backbone. Our vantage point allows us to observe more than 20,000 DSL lines. The anonymized packet-level traces are annotated with the anonymized DSL line card port id. This enables us to uniquely distinguish DSL lines since IP addresses are subject to churn and as such cannot be used to identify DSL lines [7]. While we typically do not experience any packet loss, there are several multi-second periods (less than 5 minutes overall per trace) with no packets due to OS/file-system interactions.

We use several 24 h traces collected over a period of 11 months which gives us the the opportunity to track changes in mobile device usage over time. Table 1 summarizes characteristics of the traces, including their start, duration, size, and number of observed MHDs. We note that while the number of observed DSL lines remains about the same in each trace, the number of observed MHDs has increased significantly.

The data anonymization, classification, as well as application protocol specific header extraction is performed immediately on the secured measurement infrastructure using the Bro NIDS with dynamic protocol detection [3].

2.2 Identifying MHDs

To understand how MHDs are utilized we need to identify not only their presence in our traces but also their contributions. This is non-trivial as MHD users commonly do not just operate the MHD over their DSL-line but also/mainly computers or set-top boxes. Note, that all devices active via one DSL-line usually share a single IP address.

Table 1. Overview of anonymized packet traces

					MHD HTTP Traffic	
Name	Start date	Duration	Size	# MHDs	Volume	% of HTTP
SEP08	Thu 18 Sep'08 4am	24 h	>4 TB	>200	>2 GB	0.1 %
APR09	Wed 01 Apr'09 2am	24 h	>4 TB	>400	>9 GB	0.4 %
AUG09a	Fri 21 Aug'09 2am	24 h	>6 TB	>500	>15 GB	0.6 %
AUG09b	Sat 22 Aug'09 2am	24 h	>5 TB	>500	>15 GB	0.7 %

Therefore, we rely on network signatures which we gather by observing and recording MHD behavior in a controlled environment.

Among the currently popular MHD devices are Symbian based phones, BlackBerrys, iPhones and iPods, Windows Mobile based phones, and Google Android phones [12]. We collected manual traces using tcpdump for all device types except BlackBerrys[1]. With each device we performed the following set of actions using a wireless access-point for data collection: connecting to the access-point, accessing several Web sites, watching videos on YouTube, using other mobile applications like Weather and Stocks, checking and sending emails, using Facebook, and updating/installing mobile applications on the MHD.

Analyzing these manual traces reveals that HTTP dominates the protocol mix and that most mobile applications, including Weather, Stock quotes, AppStore, and YouTube, use HTTP. From our manual traces we extract a list of HTTP user-agent strings for each device and OS combination.[2] We further augment this list by well-known strings from other mobile devices, e.g., BlackBerrys. This captures the strings of the standard applications. However, it is not possible to compile a list of all user-agent strings that MHD application writers may use. However, since most rely on standard libraries, we can add patterns for these. For example, most applications for Apple devices use the Apple CFNetwork library for communication and CFNetwork usually adds its name and version number to the end of user-agent strings. While Mac OS X also uses CFNetwork, the version numbers used by the iPhone and Mac OS X are disjoint and we can distinguish them. Based on this collection of user-agent strings we create patterns for *(i)* identifying DSL lines that "host" MHDs and *(ii)* identifying and classifying MHD usage of HTTP.

2.3 Application Protocol Mix

Finding signatures for identifying non-HTTP traffic caused by MHDs is more difficult since most other application protocols, e. g., POP, do not add device related information to their user-agent strings. Furthermore, they may use encryption.

One obvious approach for overcoming this limitation is to assume that MHDs and regular computers are used consecutively, i. e., not used at the same time at the same DSL line. Based upon this assumption one can classify all traffic after a HTTP request from a MHD on a DSL line as MHD traffic (relying on a timeout). However, we show in Sec. 3.1 that the underlying assumption is incorrect. A majority of the lines shows contemporaneous activity from MHDs and regular computers.

Therefore, we take advantage of another characteristic of network devices—their IP TTLs. The default IP TTLs of popular MHDs differ from those of the most commonly used home OSs. The default TTL of iPhones/iPods and Macs is 64, Symbian uses 69, while Windows uses 128. This enables us to separate MHD usage from regular PC usage for some combinations of OSs. While we cannot distinguish iPhones/iPods from

[1] Manual trace collection was performed with Google's G1 (Android 1.5), Apple's iPod touch (iPhone OS 2 & iPhone OS 3), HP's iPaq (Windows Mobile), HTC Touch 3G (Windows Mobile), Nokia 810 (Maemo Linux), and Nokia E61 (Symbian). Thanks to all device owners.

[2] We note that these MHD user-agent strings differ from user-agent strings used by PCs/Macs.

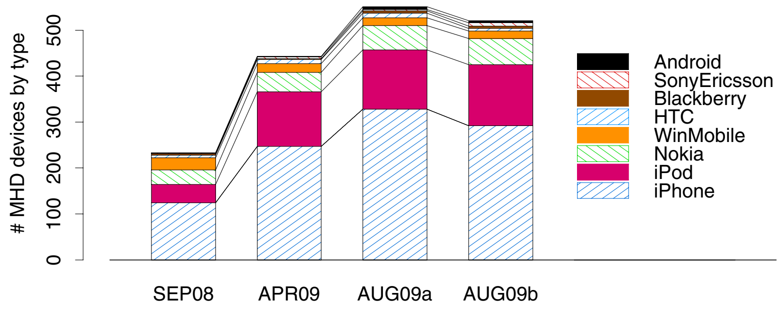


Fig. 1. Popularity of MHD device types

Macs or Windows Mobile from Windows we can use IP TTLs to separate the other combinations. Our observations show that the majority of home OSs is Windows while the majority of MHDs are iPods or iPhones. In order separate those, we first select all DSL lines for which *every* HTTP request with a TTL[3] of 64 or 69 is originated by a MHD (as identified via the user-agent). The assumption is that all traffic on these lines with TTL 64/69 is then caused by a MHD. Thus, we can then use Bro's DPD [3] on this traffic to get a first impression of the application protocol mix of MHDs. Since this approach excludes lines with certain combinations of MHDs and regular computers we are left with 54–59 % of the lines with MHDs. In addition, if the activity of the regular computer does not include HTTP we might misclassify its traffic. We note that we use this heuristic only for analyzing the application protocol mix, we use user-agent strings for all other analyses.

3 Results

After reporting on the pervasiveness of MHDs we focus on their protocol mix. Then we characterize MHDs' HTTP traffic, analyze mobile application usage, and present results on iTunes and AppStore usage.

3.1 MHD Pervasiveness

On a significant number of the DSL lines we observe traffic from MHDs (see Table 1). Indeed, in the most recent trace, AUG09, 3 % of active lines have MHD activity. Moreover, the contribution of MHDs to the observed HTTP traffic is also substantial (up to 0.7 % of HTTP bytes). This indicates that some MHD users may find it more convenient to use their mobile devices at home even if they have a regular computer as well. Note, HTTP's share of overall traffic volume is 50–60 % [4,7].

There is a strong temporal trend underlined by the rapid growth in the number of lines with MHDs' activity and in the MHDs' HTTP traffic volume. The number of lines with MHDs almost doubled between SEP08 and AUG09. The HTTP traffic volume

[3] We take NAT devices and our hop distance to the end system into account.

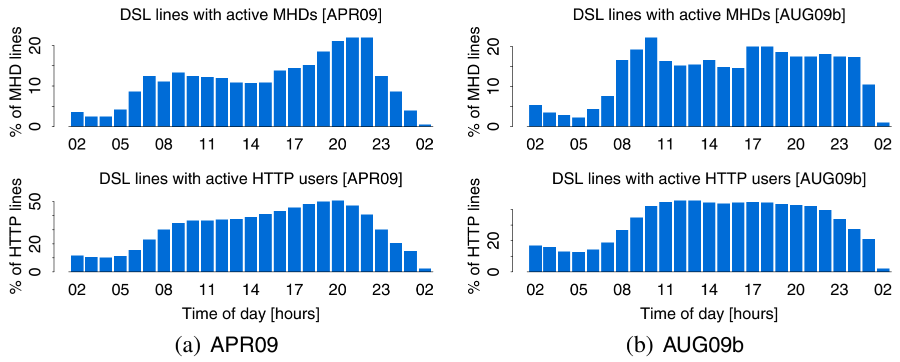


Fig. 2. Number of lines with MHD activity (top) vs. Number of lines with HTTP activity (bottom)

from MHD grew sixfold while the overall traffic volume increased only slightly and the overall HTTP volume increased by 22 % at our vantage point.

Fig. 1 shows the distribution of active devices types for all traces. We observe that Apple devices (iPhone and iPod touch) clearly dominate, both in terms of number of lines and traffic volume (not shown). They account for 86–97 % of MHDs' HTTP traffic and 71–87 % of the devices. This is in contrast to the market shares of the devices [12]. Possible explanations are that Apple users *(i)* find their device very convenient even for home use and/or *(ii)* are looking for a multimedia device that "also works as a phone". Indeed, the iPod Touch is an iPhone without phone capability. We note that starting from APR09 the number of lines with iPods outnumber the number of lines with all non-Apple MHDs combined.

We already pointed out that we have a substantial number of DSL lines "hosting" MHDs. Now we want to illustrate how the use of MHDs is distributed over the course of a day. To determine how the use of MHDs is distributed across time we plot the relative number of lines with active MHDs per hour (top) and the percentage of lines with HTTP traffic per hour for APR09 and AUG09b in Fig. 2. We see that MHDs are used throughout the day. While we see a similar behavior when looking at overall HTTP traffic, we see that MHD usage has a stronger pick-up in the morning (AUG09b even shows a peak). Overall HTTP traffic on the other hand slowly ramps up during the day. Again the convenience of using the mobile device may be a possible explanation. Users can use them to check their emails or the weather when "starting their day". The low byte contribution of mobile devices in the morning hours supports this claim (figure not shown).

Next, we examine if MHDs and regular computers are used consecutively or whether they are used contemporaneously. To asses this, we compute for each DSL line and for any two subsequent HTTP requests their inter-request-times (IRTs) and label them as *(i)* both from MHDs, *(ii)* both from non-MHDs, or *(iii)* from MHD and non-MHD. Using this information and timeouts of one second, one minute, and five minutes we compute the number of DSL lines with mixed activity (MHD and non-MHD). We find

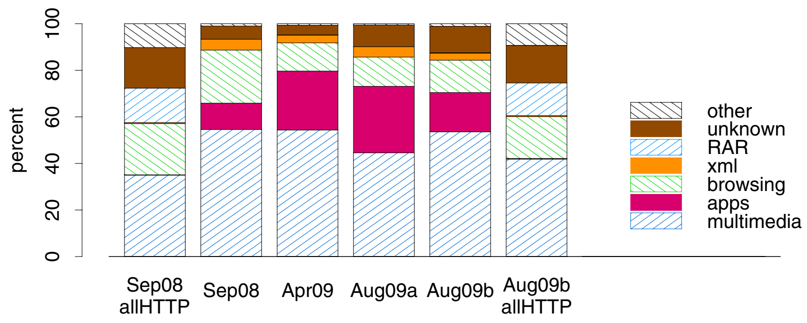


Fig. 3. HTTP content type categories by volume. Comparing MHD traffic all HTTP traffic.

that 33–39 % of MHD lines exhibit mixed MHD/non-MHD activity with IRTs of less than one second. For IRTs of less than one minute (five minutes) up to 62 % (72 %) of the lines have mixed activity.

3.2 Application Protocol Mix

While our approach for analyzing the application protocol mix of MHDs is limited (see Sec. 2.3), it still gives us a first impression of MHDs' traffic composition. We find that HTTP clearly dominates across all of our traces. HTTP contributes 80–97 % of all MHD bytes. Email related protocols account for more than 9 % of the bytes in SEP08, 2.3–2.5 % in APR09 and AUG09a. However, it drops to 0.2 % in AUG09b most likely due to a different usage patterns on weekends. In general, no other protocol has a traffic share of more than 1.5 % with the exception of 13 % unclassified traffic in APR09, and 15 % RTMP streaming in AUG09a, caused by only a handful of MHDs.

3.3 MHD Web Traffic

Given that HTTP traffic accounts for the vast majority of MHD traffic we now examine it more closely to characterize its usage and how it differs from overall HTTP usage. We use anonymized HTTP headers and identify HTTP requests from MHDs using user-agents strings as discussed in Sec. 2.2.

To identify the content-type of each transfered HTTP object we join information from the Content-Type HTTP header field and an analysis of the initial part of the HTTP body using libmagic, see [7]. We then group these into a handful of categories. We classify downloads of mobile applications as apps, video and audio content as multimedia, and images as web-browsing since the latter are usually an integral part of Web pages.

Fig. 3 shows the HTTP content type categories for MHDs and compares them with all HTTP traffic. We find that multimedia content is the most voluminous MHD content-type across all traces followed by application downloads. Interestingly, XML objects are also common. They account for 2–5 % of the transfered HTTP bytes. XML is used by many applications for status and data updates, e.g., weather forecasts, stock quotes, and sport results. Surprisingly, Web surfing itself (text based content-types and images) is

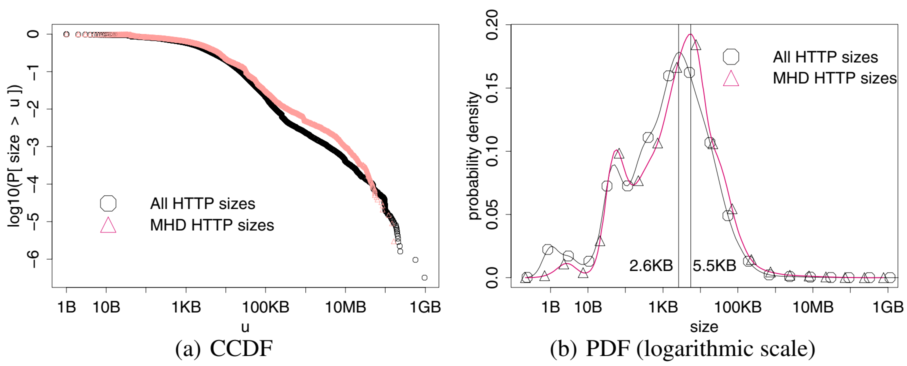


Fig. 4. Size of HTTP objects for all traffic and MHD traffic for trace APR09

only the third largest category contributing less than 14 % in the 2009 traces (23 % in SEP08).

Comparing these results to all HTTP traffic [7] we find that downloads of mobile applications and XML contribute a significantly smaller fraction to the content type mix. In contrast the volume contributed by RAR archives to all HTTP traffic is significantly larger. Browsing is a bit more prevalent in all HTTP traffic (18–22 %). Multimedia content is the biggest contributor for both. However, for all HTTP traffic flash-video is the most popular video codec, while MHDs use MPEG coding.

The volume share per DNS domain reflects the distribution of MHD content-types. Apple's `apple.com` is responsible for most of the traffic due to application downloads. Note, only the AUG09a trace shows a significant number of iPhone application downloads from third-party sites rather than the Apple's AppStore. YouTube and Stream.fm are the next most popular domains. For overall HTTP traffic One-Click-Hosters and video portals are among the top domains by volume.

To answer the question if MHD HTTP traffic characteristics differ from overall HTTP traffic we compare the distribution of HTTP object sizes. See Fig. 4 for a plot of the Cumulative Complementary Distribution Function (CCDF) and Probability Density Function (PDF) for APR09[4]. The results for the other traces are similar. We find that both distributions are consistent with a heavy-tailed distribution (see Fig. 4(a)). While the dominating mode of objects sizes downloaded by MHDs is larger (see support lines in Fig. 4(b)) the tail is heavier for all HTTP traffic.

3.4 Mobile Applications

Fig. 5 shows the popularity of the top MHDs' applications. The most popular application is Apple's browser Safari. Up to 62 % of all devices are using it. This is followed by iTunes (up to 37 %) and Weather (up to 32 %). For non-Apple MHDs we observe

[4] Coupled with a logarithmic scale on the x-axis, plotting the density of the logarithm of the data facilitates direct comparisons between different parts of the graphs based on the area under the curve.

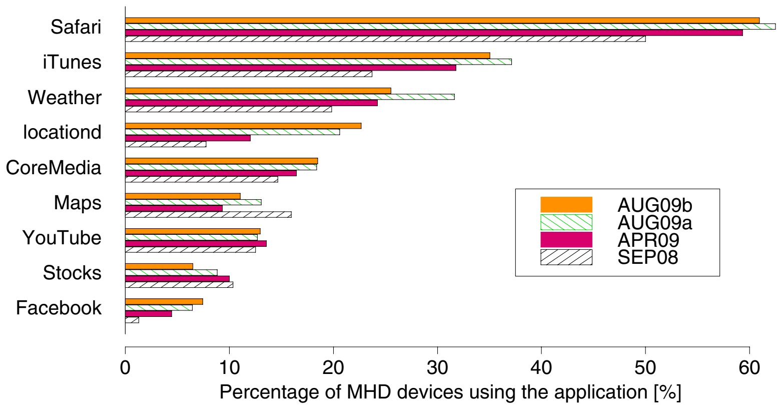


Fig. 5. Application popularity by number of MHD devices using this application

that the browser is also the most popular application. Overall we find that Apple's default applications clearly dominate. Surprisingly, given our own usage, the popularity of Maps is relatively low. One possible explanation is that one rarely needs directions while at home. CoreMedia, the media player of iPhones and iPods, is also quite prevalent. This application is e. g., responsible for playing videos accessed via the YouTube application or the browser. The YouTube application itself is only used for searching videos, tagging, and navigating within YouTube. Locationd is the wireless positioning system used on Apple devices.

To understand if users take advantage of specialized applications available for popular Web services we select two Online Social Networks that are popular in our user base: Facebook and StudiVZ. For both OSNs there are specialized applications available for the iPhone/iPod MHDs. We find that roughly half of the users ($50\,\% \pm 10\,\%$) use the specialized applications while the other half continues to use the built-in browser. This relationship is stable throughout our 11 month observation period.

3.5 Application and Media Downloads

Given that we are observing traffic from residential DSL lines we have the ability to evaluate if users use their mobile devices or their regular computer to download mobile applications and/or multimedia content. Due to the prevalence of Apple devices in our dataset we now focus on Apple iTunes store and Apple AppStore.

We find that applications are predominantly downloaded directly to the MHD (see Table 2), e.g., more than 70 % of downloads for the 2009 traces. Surprisingly, we see that for AUG09a and AUG09b the volume of application downloads in terms of bytes is almost the same for regular computers and MHD, i. e., the mean application size is larger for applications downloaded by PC/Macs. A detailed analysis reveals that this is caused by outliers; the median application size is the same for both.

We see a vastly different behavior for media downloads or purchases from Apple's iTunes store. Downloads are almost exclusively done via the regular computers. We see

Table 2. Downloads from AppStore

	# Apps	by PC/Mac		by MHD	
Trace	available	Volume	# Req	Volume	# Req
SEP08	3,000	<1 GB	<100	<1 GB	<100
APR09	7,500	<1 GB	>100	>2 GB	>250
AUG09a	70,000	>2 GB	>150	>3 GB	>450
AUG09b	70,000	>3 GB	>150	>3 GB	>400

several thousand media files being accessed in the 2009 traces. However, only a handful of downloads are via MHDs which results in a small byte contribution.

4 Related Work

Only a small number of studies have focused on Internet traffic in 3G mobile or cellular networks. Svoboda et al. [8] analyze various aspects of GPRS and UMTS traffic using anonymized header traces from 2004 and 2005. They study traffic volume per user and protocol mix. In terms of protocol mix, they find that HTTP is the dominant protocol with 40–60 % of traffic. Heikkinen et al. [5] analyze P2P usage from passive UMTS header traces in Finland from 2005–2007. Web traffic accounts for 57–79 % of bytes from mobile hand-held devices, email for 10–24 %, and P2P is not noticeable. Williamson et al. [13] analyze packet/data call event traces from a CDMA2000 network from 2004. They focus their analysis on link-layer behavior, session properties, and user mobility.

Several studies have analyzed TCP performance and low-level traffic characteristics in GPRS and CDMA data networks [2,6,14]. Other studies analyze the content requested or available for mobile devices. Using data from 2000, Adya et al. [1] analyze the Web server logs of a major commercial site and study the requests of mobile clients. They find that stock quotes, news, and yellow pages were the most commonly accessed content in their traces. Timmins et al. [9] use active measurements to crawl the Web for sites offering specialized content for mobile devices. Verkasalo [11] studies how Symbian phone features are used by instrumenting the handset. He finds that the camera feature and games are the most common multimedia applications.

Trestian et al. [10] analyzes mobility and web-application usage in a 3G network from a metropolitan area. We on the other hand, focus on stationary usage when MHDs are connected at home via WiFi. Trestian et al. characterize web-application usage by counting the number of HTTP request and find that social networking, music, and e-mail are the most common web. They do not asses who many *users* utilize a particular application, which is the approach we use to characterize application usage.

5 Conclusion

Our analysis of residential broadband DSL lines of a large European ISP shows that there is a significant and increasing number of active MHDs. We find that iPhones and iPods are by far the most commonly observed MHDs. This has an impact on the most popular mobile applications: Safari (Apple's browser), iTunes, and Weather. The largest fraction by volume of MHD HTTP content is multimedia. Comparing HTTP object sizes of overall and MHD traffic we find that MHD HTTP objects are on average larger. The contribution of MHDs to the overall traffic volume is still small, but rapidly growing, especially compared to the overall traffic growth. In future work we plan a more detailed analysis of non-HTTP protocols and refine our methodology for protocol classification. In addition, we plan to extend our analysis to traces from cellular data networks.

References

1. Adya, A., Bahl, P., Qiu, L.: Characterizing alert and browse services of mobile clients. In: Proc. Usenix Annual Technical Conference (2002)
2. Benko, P., Malicsko, G., Veres, A.: A large-scale, passive analysis of end-to-end TCP performance over GPRS. In: Proc. IEEE INFOCOM, vol. 3 (2004)
3. Dreger, H., Feldmann, A., Mai, M., Paxson, V., Sommer, R.: Dynamic application-layer protocol analysis for network intrusion detection. In: Proc. Usenix Security Symposium (2006)
4. Erman, J., Gerber, A., Hajiaghayi, M.T., Pei, D., Spatscheck, O.: Network-aware forward caching. In: Proc. World Wide Web Conference (2009)
5. Heikkinen, M., Kivi, A., Verkasalo, H.: Measuring mobile peer-to-peer usage: Case Finland 2007. In: Proc. Passive and Active Measurement Conference (2009)
6. Lee, Y.: Measured TCP performance in CDMA 1x EV-DO network. In: Proc. Passive and Active Measurement Conference (2006)
7. Maier, G., Feldmann, A., Paxson, V., Allman, M.: On dominant characteristics of residential broadband internet traffic. In: Proc. Internet Measurement Conference (2009)
8. Svoboda, P., Ricciato, F., Pilz, R., Hasenleithner, E.: Composition of GPRS, UMTS traffic: snapshots from a live network. In: IPS-MOME: Workshop on Internet Performance, Salzburg Research Forschungsgesellschaft (2006)
9. Timmins, P., McCormick, S., Agu, E., Wills, C.: Characteristics of mobile web content. Hot Topics in Web Systems and Technologies (2006)
10. Trestian, I., Ranjan, S., Kuzmanovic, A., Nucci, A.: Measuring serendipity: connecting people, locations and interests in a mobile 3g network. In: Proc. Internet Measurement Conference (2009)
11. Verkasalo, H.: Empirical observations on the emergence of mobile multimedia services and applications in the U.S. and Europe. In: Proc. Conference on Mobile and ubiquitous multimedia (2006)
12. Wikipedia. Smartphone: Operating systems (September 2009), http://en.wikipedia.org/w/index.php?title=Smartphone&oldid=315621107#Operating_systems
13. Williamson, C., Halepovic, E., Sun, H., Wu, Y.: Characterization of CDMA2000 cellular data network traffic. In: Proc. IEEE Conference Local Computer Networks (2005)
14. Won, Y.J., Park, B.-C., Hong, S.-C., Jung, K.B., Ju, H.-T., Hong, J.W.: Measurement analysis of mobile data networks. In: Proc. Passive and Active Measurement Conference (2007)
15. Wood, N.: Mobile data traffic growth 10 times faster than fixed over next five years. Total Telecom (September 2009), http://www.totaltele.com/view.aspx?ID=448681

A Learning-Based Approach for IP Geolocation

Brian Eriksson[1], Paul Barford[1], Joel Sommers[2], and Robert Nowak[1,⋆]

[1] University of Wisconsin - Madison
[2] Colgate University
bceriksson@wisc.edu, pb@cs.wisc.edu, jsommers@colgate.edu,
nowak@ece.wisc.edu

Abstract. The ability to pinpoint the geographic location of IP hosts is compelling for applications such as on-line advertising and network attack diagnosis. While prior methods can accurately identify the location of hosts in some regions of the Internet, they produce erroneous results when the delay or topology measurement on which they are based is limited. The hypothesis of our work is that the accuracy of IP geolocation can be improved through the creation of a flexible analytic framework that accommodates different types of geolocation information. In this paper, we describe a new framework for IP geolocation that reduces to a machine-learning classification problem. Our methodology considers a set of lightweight measurements from a set of known monitors to a target, and then classifies the location of that target based on the most probable geographic region given probability densities learned from a training set. For this study, we employ a Naive Bayes framework that has low computational complexity and enables additional environmental information to be easily added to enhance the classification process. To demonstrate the feasibility and accuracy of our approach, we test IP geolocation on over 16,000 routers given ping measurements from 78 monitors with known geographic placement. Our results show that the simple application of our method improves geolocation accuracy for over 96% of the nodes identified in our data set, with on average accuracy 70 miles closer to the true geographic location versus prior constraint-based geolocation. These results highlight the promise of our method and indicate how future expansion of the classifier can lead to further improvements in geolocation accuracy.

1 Introduction

There are many ways in which the structural and topological characteristics of the Internet can be considered. One way that has significant implications for advertisers, application developers, network operators and network security analysts is to identify the *geographic location* of Internet devices (*e.g.,* routers or

⋆ This work was supported in part by the National Science Foundation (NSF) grants CCR-0325653, CCF-0353079, CNS-0716460 and CNS-0905186, and AFOSR grant FA9550-09-1-0140. Any opinions, findings, conclusions or other recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the NSF or the AFOSR.

end hosts). Geographic location can mean the precise latitude/longitude coordinates of a device or a somewhat more coarse-grained location such as within a zip code, city, county or country.

There are a number of challenges in finding the geographic location of a given Internet device. The most obvious is that there is no standard protocol that provides the position of any device on the globe (although DNS entries can include a location record). Furthermore, Internet devices are not typically equipped with location identification capability (*e.g.,* GPS, although this may change in the future), and even if they did, some would consider this information private. Prior methods have focused on identifying the geographic location of an Internet device based on its position relative to a set of active measurements from landmarks with known positions. While these methods have been shown to be capable of producing relatively accurate geographic estimates in some areas, inaccuracies remain for a variety of reasons. Principal among these is the fact of inconsistent density of specific measurements across the globe.

The goal of our work is to broadly improve IP geolocation accuracy over prior methods. Our hypothesis is that the large estimation errors caused by imperfect measurements, sparse measurement availability, and irregular Internet paths can be addressed by expanding the scope of information considered in IP geolocation. The estimation framework that we develop to test this hypothesis is to cast IP geolocation as a *machine learning-based classification* problem. This extensible approach enables information from multiple datasets to be fused such that areas that have low information content from one measurement can be compensated with better information content from other measurements.

To flesh out this framework in order to test our hypothesis, we must select both a classification method and a set of measurements that can be used to estimate IP geolocation. We develop a *Naive Bayes* estimation method that assigns a given IP target to a geographic partition based on a set of measurements associated with that IP target. Given the potentially large number of measurements to an IP target, probability likelihood estimation is simplified by a Naive Bayes approach. The network measurement data considered in this framework includes latency and hop count from a set of landmarks to an IP target. We also include population density in the framework as a demonstration of a non-network measurement that can help refine the estimates. The selection of this classifier/measurement combination was made to demonstrate the potential of this new approach, but is not meant to be definitive nor comprehensive.

To test and evaluate the capabilities of this initial instance of our learning-based approach, we consider geographic partitioning at the level of counties in the continental United States[1]. While considerable Internet topology lies outside the continental United States, the initial validation on this dataset will motivate future work on end hosts located outside the United States. We identified a target set of 114,815 spatially diverse nodes in the Internet through full

[1] Finer-grained partitioning on the order of zip codes or city blocks is certainly feasible in our framework, but county-level was selected due to the availability of data for test and evaluation.

mesh `traceroute` probing from Planetlab nodes, supplemental data from the iPlane [1] project, and careful alias resolution. For ground truth on the geographic location of these target nodes, we used the Maxmind database [2] as a validation set for our methodology. Of the 114,815 IP target nodes identified in our measurements, 16,874 were identified in the Maxmind database as being within the United States with known city locations. Due to its use as a commercial product, the exact underlying methodology for the Maxmind database is not available, although extensive use of user-survey geolocation information is known to be used.[2] For that set of 16K target nodes, we then gathered hop count and latency measurements from 78 PlanetLab nodes located in the United States, which were the starting point for our assessment.

We selected a subset of target nodes[3] for training our classifier, with the training set nodes having both known measurements to the monitors and known geolocation. With the remaining nodes, we compare the geolocation estimates of both our learning-based approach and Constraint-Based Geolocation (CBG) [3] (the current state-of-the-art geolocation algorithm using ping measurements) validated against the locations found using the Maxmind database. We find that our estimator is able to provide better location estimates than CBG for 96% of the nodes and on average provide an estimate that is 70 miles closer to the true location. We believe that these results make a compelling case for future development of learning-based methods for IP geolocation.

2 Learning-Based IP Geolocation

Given a single target IP address, *can we determine the geographic location of the target IP?* Consider a single target IP address with a set of measurements from a set of monitors with known geolocation to this target IP address. For the purposes of this work, the measurement set $\mathcal{M}$ $(= \{m_1, m_2, ..., m_M\})$ is the collection of both latency and hop count values going from the monitor set. Without loss of generality, now consider a set of possible counties in the continental United States ($\mathcal{C}$), such that the target is located in some county $c \in \mathcal{C}$. This changes the underlying problem to, *Given the measurement set* $\mathcal{M}$*, can we estimate which county* $c \in \mathcal{C}$ *the target IP is located in?* The best classifier would choose the county ($\widehat{c}$) that the target is most probably located in given the measurement set, $\widehat{c} = \arg\max_{c \in \mathcal{C}} P(c \mid \mathcal{M})$. Using Bayes Theorem [4] ($P(A|B) = \frac{P(B|A)P(A)}{P(B)}$), therefore we can restate the classifier as $\widehat{c} = \arg\max_{c \in \mathcal{C}} P(c \mid \mathcal{M}) = \arg\max_{c \in \mathcal{C}} P(\mathcal{M} \mid c) P(c)$. Where the value $P(\mathcal{M})$, the probability of observing the set of measurements, can be ignored due to this value being constant across any choice of county c.

[2] Due to its dependence on user generated data, updating the Maxmind database requires extensive user surveying that is not needed with our learning-based methodology.

[3] We consider IP addresses and nodes to be equivalent in this paper since even if alias resolution on routers is imperfect, it should not affect our empirical results.

Next, we expand our estimation framework to consider features other than measurements from monitors to IP targets. Given that the targets in this paper are routers, we can use the work in [5] to inform where these routers should be geographically located. Specifically, the value $P(c)$, the probability of classifying a target in county c, will be chosen using the results showing that the number of routers in a specific geographic location is strongly correlated with the population of that geographic location. Therefore, we can estimate the probability of classifying into a given county to be the population of that county divided by the total population in all the counties under consideration.

$$\widehat{P}(c_i) = \frac{\text{Population of } c_i}{\sum_{j \in \mathcal{C}} \text{Population of } c_j} \tag{1}$$

How can we estimate the value $P(\mathcal{M} \mid c)$, the probability likelihood of a measurement set $\mathcal{M}$ being observed given the target is located in county c? Given a set of training data, a set of IP addresses with known measurement sets $\mathcal{M}$ and locations c, we could use off-the-shelf techniques (kernel density estimators, histograms, etc.) to estimate the multivariate likelihood density $P(\mathcal{M} \mid c)$. A problem is that the set $\mathcal{M}$ is most likely of high dimensions (with dimensionality equal to the number of hop count and latency measurements observed to this target, in this case, on the order of 100), and most density estimator techniques have an error rate that increases quickly with the dimension of the problem [4].

If all of the values of M were statistically independent from each other, then the likelihood density could be restated as: $P(\mathcal{M} \mid c) = P(\{m_1, m_2, ..., m_M\} \mid c) \approx P(m_1 \mid c) P(m_2 \mid c) ... P(m_M \mid c)$ This converts the problem from estimating one M-dimensional density to estimating M one-dimensional densities. However, it should be assumed that there is a large degree of correlation between measurements, with prior work in [6] showing correlation between hop count measurements, and work in [7] showing correlation between latency measurements. The risk of assuming statistical independence between measurements is informed by empirical studies on highly dependent data in [8]. That work shows that for classification, there is little penalty for assuming statistical independence even when the measurements are highly statistically dependent. This is due to classification performance depending only on the most-probable class (in this case, county region) likelihood probability being greater than other class likelihood probabilities, not the goodness-of-fit of our estimated likelihood probability to the true likelihood probability.

The next step in our learning-based framework is to estimate the one-dimensional densities, $P(m_i \mid c)$, the probability of the measurement value m_i being observed given that the target is located in county c. Consider a set of training data, where for each training target, both the measurement set $\mathcal{M}$ and the geolocation county c is known. Given the known monitor placement, for the entire training set we can determine the distance vector $\mathbf{d} = \{d_1, d_2, ..., d_M\}$, where d_i is the distance between the monitor associated with measurement m_i and county c. These measurements with distance ground truth can then be used to learn the density (the probability of observing measurement m_i given that the target is located d_i distance away from the monitor associated with measurement m_i).

Simple density estimators, such as histograms, can be used and will assure that measurement outliers do not significantly contribute to the density estimation. One drawback to histogram estimators is that the lack of smoothness in the estimated density can hurt performance. Instead, we will look to use Kernel Density Estimators [4], which use the summation of smooth kernel functions to estimate the density. This smoothness in the estimated density allows improved estimation of the true density given the limited size of our training set.

For hop count measurements, a one-dimensional density will be estimated at each hop count value ranging from one hop away from a monitor to ten hops away (it is assumed that any distance longer than ten hops will not help in estimating distance). For latency measurements, due to the limited amount of training data, the measurements are aggregated together separated by 10ms, with a single estimated one-dimensional density for 0-9ms, a separate one-dimensional density for 10-19 ms, 20-29ms, etc. An example of a kernel estimated density for latency measurements can be seen in Figure 1 along with the resulting probability distribution across the US counties for observing this latency measurement to a monitor with known geolocation.

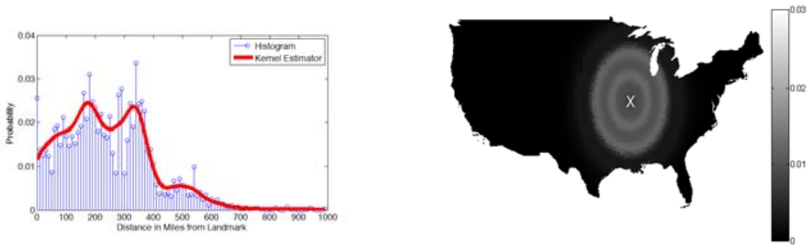


Fig. 1. (Left) - Probability for latency measurements between 10-19ms being observed given a target's distance from a monitor. Stem plot - Histogram density estimation, Solid line - Kernel density estimation. (Right) - The kernel estimated probability of placement in each county given latency observation between 10-19ms from a single monitor marked by 'x'.

The amount of location information from latency measurements is likely to be of more use than the location information derived from hop count measurements or population data. Therefore we introduce two weights λ_{hop} and λ_{pop} as the weights on the hop count measurements and the population density data respectively. Informed by the geolocation improvement by using measurement weights in the Octant framework [9], the ordering of the measurements should also imply some degree of importance, as the location of the monitor with the shortest latency measurement to the target should inform the classifier more than the monitor with the 30-th closest latency measurement. Therefore, we will also weight the ordering of measurement values by an exponential, such that the i-th latency measurement is weighted by $\exp(-i \cdot \gamma_{lat})$ and the j-th hop count measurement is weighted by $\exp(-j \cdot \gamma_{hop})$. The weight parameter values $(\lambda_{hop}, \lambda_{pop}, \gamma_{lat}, \gamma_{hop})$ will be found by the weight values that minimize the sum of squared distance errors between the training set of IPs known locations and the Naive Bayes estimated locations.

2.1 Methodology Summary

Dividing the measurement set $\mathcal{M}$ into the set of latency measurements $\{l_1, l_2, ..., l_m\}$ and the set of hop count measurements $\{h_1, h_2, ..., h_m\}$ (where the total number of measurements $M = 2m$), our learning-based classifier using the independence assumption can be restated using the kernel density estimators (where instead of the true likelihood $P(m_i \mid c)$ we have the kernel estimated $\widehat{P}(m_i \mid c)$), the weight terms, and the monotonic properties of the logarithm function as

$$\widehat{c_i} = \arg\max_{c \in \mathcal{C}} \left(\lambda_{pop} \log \widehat{P}(c) + f_{hop} + f_{lat} \right) \tag{2}$$

Where $f_{hop} = \lambda_{hop} \sum_{j=1}^{m} \exp(-j \cdot \gamma_{hop}) \log \widehat{P}(h_j \mid c)$, and $f_{lat} = \sum_{j=1}^{m} \exp(-j \cdot \gamma_{lat}) \log \widehat{P}(l_j \mid c)$, and the term $\widehat{P}(c)$ for the 3,107 counties in the continental United States is found using Equation 1.

Algorithm 1. Naive Bayes IP Geolocation Algorithm

Initialize:

- Measure the hop-count and latency from every monitor to a training set with known geographic locations.
- Using a population density database, find $\widehat{P}(c)$ for all $c \in \mathcal{C}$ using Equation 1.
- Using kernel density estimators, estimate the one-dimensional distribution $\widehat{P}(m|c)$ for every measurement $m \in \mathcal{M}$.
- Find the optimal values for $\lambda_{hop}, \lambda_{pop}, \gamma_{lat}, \gamma_{hop}$ that minimize the sum of squared distance errors over the training set.

Main Body

1. For each target IP with unknown geography, estimate the location $\widehat{c_i}$ using Equation 2.

A summary of the complete methodology is seen in Algorithm 1. Note that all the computational complexity of this algorithm is on training the parameters $(\lambda_{hop}, \lambda_{pop}, \gamma_{lat}, \gamma_{hop})$. Each target is geolocated using only $O(M |\mathcal{C}|)$ number of multiplications, where $|\mathcal{C}|$ is the total number of location classes under consideration (in this paper, the number of counties in the continental United States), and M is the total number of measurements to the current target IP. The computational complexity being linear in both the number of locations and the number of monitors demonstrates the feasibility of future large-scale Internet studies using this method.

3 Experiments

To assess our geolocation algorithm, we sought a large set of IP addresses of routers with as much spatial diversity as possible within the continental United States. Starting with the spatially diverse set of Planetlab [10] node locations, the full mesh `traceroute` probing between these nodes will find a very large set of router IP addresses with high spatial diversity. Existing data were provided by the iPlane project [1], which performs a `traceroute` from all available Planetlab hosting sites to a set of target prefixes obtained through the Routeviews project [11]. We used four weeks of iPlane data collected over the period of 12 December 2008 to 8 January 2009. In addition to the iPlane data, we collected `traceroute` data between a full mesh of Planetlab hosting sites, of which there were 375 at the time we collected these data. For performing traceroutes, we used the Paris traceroute tool [12], using it once in UDP mode and a second time in ICMP mode in order to discover as many routers as possible [13]. Options were set in the Paris traceroute tool so that it produced a low level of probes while taking somewhat longer to complete a given traceroute. We collected a full mesh of Planetlab traceroute measurements three separate times between December 11, 2008 and January 6, 2009. For these measurements, we were able to use about 225 Planetlab sites due to maintenance and other issues.

Using these two data sets, we were able to discover 125,146 unique router IPv4 addresses. A standard problem with traceroute-based studies is IP interface disambiguation, also known as *alias resolution*. Interfaces on a given Internet router are typically assigned separate IP addresses; identifying which addresses correspond to the same physical router is the challenge in alias resolution. To de-alias our data set, we used the alias database published by the iPlane project. This database builds on prior work in alias resolution, including the methods used by the Rocketfuel project [14]. Upon de-aliasing our set of router IP addresses, we identified 114,815 routers.

To construct the measurements used in our analysis (as described below), we required the hop counts and latency measurements to each identified router from all available Planetlab sites. In order to limit the overhead of probing for this hop count and latency data, we used the following approach. For each IP address, we sent a direct ICMP echo request packet (*i.e.*, a ping). In other work, it was observed that a majority of Internet hosts respond to ICMP echo request packets [15]; we also found this to be true. Indeed, more than 95% of all router IP addresses we identified responded. This should not be surprising considering the fact that these addresses were initially identified through active probing. For computing the hop count, we use the methodology of [16] on the echo response (note that this is the hop count of the reverse path). For geolocation ground truth, we use the Maxmind database [2], which is rated to be 82% accurate within 25 miles for IPs located within the US. From our dataset of 114,815 disambiguated routers, Maxmind identified 16,874 routers located in the continental United States with known county location. Using 5-Fold Cross Validation [4], we test the performance of the methodology five times using 20%

of the routers as our training set, leaving the remaining 80% of the routers to test the accuracy of our methodology.

We compare the geolocation results from our learning-based method to Constraint-Based Geolocation (CBG). To generate CBG geolocation estimates, we implemented the algorithm described in [3]. CBG is the current state-of-the-art IP geolocation methodology using only ping-based measurements. The basic intuition behind CBG is that each latency measurement to a set of monitors with known location can be considered a series of constraints, where given speed-of-light in fiber assumptions and self-calibration using a set of training data, we can determine a feasible geographic region given each latency measurement. Given a series of latency measurements, the possible geographic placement is considered the intersection of many constraint regions, with the estimated location behind the centroid of this intersection region.

To assess performance of both geolocation algorithms, we will consider the error distance to be the distance in miles between the centroid of our estimated classified county and the centroid of the ground truth (Maxmind) county. Performance of our learning-based Naive Bayes framework and the CBG method with respect to the empirical cumulative probability can be seen in Figure 2-(left). As seen in the figure, the geolocation estimates produced by our learning-based framework are more accurate than CBG for 96% of the routers. On average the Naive Bayes location estimates are 70 miles closer to the true location than the CBG estimates.

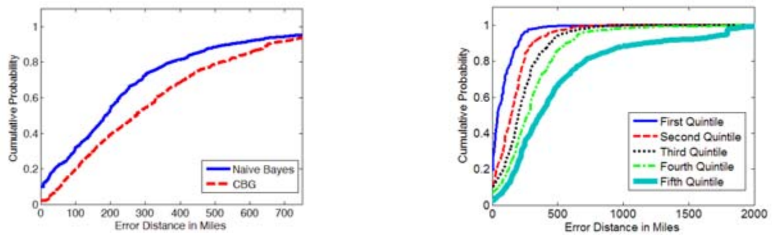


Fig. 2. (Left) - Empirical cumulative probability of error distance. (Right) - Breakdown of each quintile empirical cumulative probability error distance for our learning-based methodology.

To analyze the impact of using multiple features in our learning-based framework, we generate geolocation estimates when both population density information is removed (setting the weight of using the population density to zero, $\lambda_{pop} = 0$) and when hop count information is removed (setting the weight of using the hop count data to zero, $\lambda_{hop} = 0$). These two conditions resulted in an average error distance of 261.89 and 277.29 miles, for missing population data and missing hop count data respectively. These results indicate that both the hop count data and the population density information significantly contribute to the improved performance of the methodology. Using only latency information, the Naive Bayes methodology still outperforms the CBG method (278.96 mile average error vs. 322.49 mile average error) due to the more accurate multiple latency density estimates used to classify the location of each end host instead of simply using the intersection of feasible latency regions as in the CBG methodology.

Using Equation 2, the Naive Bayes framework can find $\widehat{P}(\widehat{c} \mid \mathcal{M})$, the estimated probability of each target being classified correctly by our learning-based framework given the set of measurements. This can be considered a level of confidence in the classification of each target IP. Using this confidence level, we can sort into quintiles and form quintile sets containing the 20% of the target IPs with the largest $\widehat{P}(\widehat{c} \mid \mathcal{M})$ values (*e.g.,* the targets we are most confident in accurately geolocating), to a quintile set containing the 20% of target IPs with the smallest $\widehat{P}(\widehat{c} \mid \mathcal{M})$ values (*e.g.,* the targets we are least confident in). Figure 2-(right) shows how this confidence level accurately predicts the quality of our classification, with the most confident 20% of the targets being classified far more accurately than any other quintile set. Therefore, in addition to estimating the geolocation of each target IP, we also have a level of confidence that directly corresponds to the accuracy of our prediction.

4 Related Work

The main prior work in IP geolocation that we compared and contrasted our learning-based methodology with is Constraint-Based Geolocation [3]. More recent geolocation work in [9],[17] has found improvements over Constraint-Based Geolocation, but both methodologies require `Traceroute`-based measurements to the targets along with location hints acquired by unDNS [14] probes. One potential disadvantage of these methodologies is the dependency on DNS naming conventions, which have been shown to not always be reliable [18]. This requires sophisticated location validation and reweighting mechanisms to be developed and maintained. The focus of this work was to introduce our elegant learning-based geolocation framework and validate its performance using simple ping-based measurements. We leave the extension of our learning-based framework to these newer `Traceroute`-based methodologies as future work. To the best of our knowledge, this is the first work to frame IP geolocation as a machine learning problem.

5 Conclusions and Future Work

The goal of our work is to improve the accuracy of estimates of the geographic location of nodes in the Internet. Our work is based on the hypothesis that the ability to zero in on the geolocation of nodes is improved by considering a potentially broad set of features including both active measurements and more static characteristics associated with locations. To consider this hypothesis, we introduce a learning-based framework that enables geolocation estimates to be generated efficiently, and is flexible in the feature space that can be considered. In this initial study, we employ a Naive Bayes classifier and generate estimates from two types of empirical measurements in our framework (latency and hop counts) and one societal characteristic (population density). We then test the feasibility of our learning-based approach using an empirical dataset of over 16K target routers, and latency and hop count data to 78 monitors with known geographic locations. We show that our geolocation estimates are more accurate for 96% of the routers in our test set versus the estimates generated by a current state-of-the-art constraint-based geolocation method. We also show how the use of multiple features does indeed enhance the overall estimation accuracy. In future work, we plan to investigate additional features that improve the accuracy of our estimates, and the possible use of a multi-scale classification framework that narrows the classification region given classification confidence levels.

References

1. Madhyastha, H., Isdal, T., Piatek, M., Dixon, C., Anderson, T., Krishnamurthy, A., Venkataramani, A.: iPlane: An Information Plane for Distributed Services. In: USENIX OSDI 2006 (November 2006)
2. Maxmind geolocation database, http://www.maxmind.com
3. Gueye, B., Ziviani, A., Crovella, M., Fdida, S.: Constraint-based geolocation of internet hosts. IEEE/ACM Transactions on Networking (December 2006)
4. Wasserman, L.: All of Nonparametric Statistics (May 2007)
5. Lakhina, A., Byers, J., Crovella, M., Matta, I.: On the Geographic Location of Internet Resources. IEEE Journal on Selected Areas in Communications (August 2003)
6. Eriksson, B., Barford, P., Nowak, R.: Network Discovery from Passive Measurements. In: ACM SIGCOMM 2008 (August 2008)
7. Ng, E., Zhang, H.: Predicting Internet Network Distance with Coordinate-baseed Approaches. In: IEEE INFOCOM (April 2002)
8. Rish, I.: An Empirical Study of the Naive Bayes Classifier. In: Workshop on Empirical Methods in Artificial Intelligence (2001)
9. Wong, B., Stoyanov, I., Sirer, E.G.: Octant: A comprehensive framework for the geolocation of internet hosts. In: USENIX NSDI 2007 (April 2007)
10. Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T., Wawrzoniak, M.: Operating System Support for Planetary-Scale Network Services. In: USENIX NSDI 2004 (March 2004)
11. Oregon Route Views Project, http://www.routeviews.org/
12. Augustin, B., Cuvellier, X., Orgogozo, B., Viger, F., Friedman, T., Latapy, M., Magnien, C., Teixeira, R.: Avoiding traceroute anomalies with Paris traceroute. In: ACM IMC 2006 (October 2006)
13. Luckie, M., Hyun, Y., Huffaker, B.: Traceroute Probe Method and Forward IP Path Inference. In: ACM IMC 2008 (October 2008)
14. Spring, N., Mahajan, R., Wetherall, D.: Measuring ISP Topologies with Rocketfuel. In: ACM SIGCOMM 2002 (August 2002)
15. Heidemann, J., Pradkin, Y., Govindan, R., Papadopoulos, C., Bartlett, G., Bannister, J.: Census and Survey of the Visible Internet. In: ACM IMC 2008 (October 2008)
16. Wang, H., Jin, C., Shin, K.: Defense against spoofed IP traffic using hop-count filtering. IEEE/ACM Transactions on Networking 15(1), 40–53 (2007)
17. Katz-Bassett, E., John, J.P., Krishnamurthy, A., Wetherall, D., Anderson, T., Chawathe, Y.: Towards IP Geolocation Using Delay and Topology Measurements. In: ACM IMC 2006 (October 2006)
18. Zhang, M., Ruan, Y., Pai, V., Rexford, J.: How DNS Misnaming Distorts Internet Topology Mapping. In: USENIX Annual Technical Conference (2006)

A Probabilistic Population Study of the Conficker-C Botnet

Rhiannon Weaver

CERT, Software Engineering Institute
rweaver@cert.org

Abstract. We estimate the number of active machines per hour infected with the Conficker-C worm, using a probability model of Conficker-C's UDP P2P scanning behavior. For an observer with access to a proportion δ of monitored IPv4 space, we derive the distribution of the number of times a single infected host is observed scanning the monitored space, based on a study of the P2P protocol, and on network and behavioral variability by relative hour of the day. We use these distributional results in conjunction with the Lévy form of the Central Limit Theorem to estimate the total number of active hosts in a single hour. We apply the model to observed data from Conficker-C scans sent over a 51-day period (March 5th through April 24th, 2009) to a large private network.

Keywords: Botnets, Conficker, Population Estimation, Probability Models, Central Limit Theorem.

1 Introduction

When new botnets emerge, the classic question is, "How big is it?" In the statistical literature, population estimation is based on mark-recapture models and their extensions to a wide class of generalized linear models [6]. In network analysis, simple mark-recapture techniques, which reduce to counting intersections among overlapping sets, have been applied to study botnet populations [3,8,9], as well as to other phenomena such as peer-to-peer file sharing networks [10,16]. But the "overlapping sets" method is valid only for closed populations with direct observation of individuals of interest, and equal probability of capture for all individuals. Internet phenomena often violate these assumptions, resulting in the need for more sophisticated modeling techniques.

Extending mark-recapture models to open populations is widely addressed in the literature (eg. [17]), but network phenomena often admit a specific complication of direct observation: we would like to express population sizes in terms of the number of infected machines, but we view botnets through a filter of IP space. The existence of NAT, proxies and DHCP leases complicates the "one IP address, one host" model.

Applying mark-recapture models to machines, as opposed to IP addresses, requires averaging aggregations and distributions of activity across possible configurations of disambiguated hosts. On the other hand, applying mark-recapture models directly to IP addresses introduces heterogeneity among individuals; for example, a NAT is observed if at least one of its underlying hosts is observed, whereas a DHCP address is observed only if the single host to which it is allocated is observed. [4] present a solution for the case when heterogeneity can be modeled as a series of observable, nominal classes. But heterogeneity in botnet behavior often arises from variations in underlying rates of observed counts.

This leads us to consider a method for measuring the active size of a botnet, based on the observable behavior of a single infected host. Our dynamic measurement of active machines per hour cannot be used to track a botnet's overall "footprint" size [1] across days or weeks. But it is simpler to implement than a fully specified heterogeneous mark-recapture model. A single-host behavioral model is also a necessary component of the generalized mark-recapture methodology, so this work provides a stepping stone toward applying more complicated models.

As an example, the Conficker-C variant that emerged within the Conficker botnet in March 2009 introduced a specific pattern of peer-to-peer (P2P) activity. When a host infected with Conficker-C comes online, it searches for peers by randomly generating a set of destination IP addresses across most of IPv4 space, and attempting UDP connections to these hosts. Connection ports use an algorithm based on the source IP address and date, which was cracked by several independent researchers [5,13]. As a result, Conficker-C P2P traffic can be observed with high reliability in the large-scale summary information contained in network flow data, making it a good candidate for behavioral modeling.

We model the hourly number of UDP P2P connection attempts that an observer monitoring a proportion δ of IP space would see from a single infected host. Rather than inferring a time series of UDP scan activity for each machine, disambiguated from NAT or DHCP addresses, our model represents the "typical host" by averaging across reasonable probability distributions for many unobservable parameters. This marginal model is used in conjunction with the Central Limit Theorem to estimate the total number of active hosts per hour, with confidence intervals that account for measurement uncertainty, stochastic elements in the Conficker-C protocol, and random variation across network activity.

Section 2 discusses the stochastic components of the Conficker-C P2P protocol and network behavior that inform the marginal model, and formalizes this information into a probability model. Section 3 introduces a version of the Central Limit Theorem that lets us describe the distribution of aggregate scan attempts of all active hosts per hour. Section 4 presents results of applying the method to data collected over a 51-day period from a large network. Section 5 summarizes.

2 Modeling Conficker-C

We develop our model in two steps. First, we use information from published reports and studies of the Conficker-C P2P protocol to specify the distribution of the number M_h of hourly UDP connection attempts made by an infected host. Next, we specify the conditional distribution of the number y_h of observed hits in the monitored space given M_h. We use $\pi(a)$ to denote the prior distribution of quantity a, and $\pi(a \mid b)$ to denote the conditional distribution of a given b. We use μ_a or $E(a)$ to denote the mean of quantity a, and σ_a^2 or $\mathrm{Var}(a)$ to denote the variance of a.

Protocol and Network Behavior. In September 2009, [15] provided a de-obfuscated reverse engineering of the image of the Conficker-C P2P binary image as it appeared on March 5, 2009. We use this information to determine the protocol-specific variations in the distribution of UDP scan attempts per hour for an active, infected host.

When initiated, the P2P module spawns a global UDP scanning thread for each valid network connection discovered, in order to bootstrap a peer list of up to 2048 peers. Up to 32 threads can run simultaneously. Each thread alternates between a 5-second sleep cycle and a scan phase where it randomly generates a list of up to 100 IP addresses to contact. At each selection, the host chooses an IP address from its list of n peers with probability equal to

$$\gamma_n = \left(1000 - \left\lfloor \frac{950n}{2048} \right\rfloor\right)^{-1} . \tag{1}$$

This expression is taken directly from the C code in Conficker-C's P2P module. If the choice is not to select a peer, the host pseudo-randomly generates an IP address, which is added to the list only if it satisfies the following connection criteria:

1. the IP address is not a DHCP or broadcast address;
2. the IP address is not a private (RFC1918) subnet address;
3. the IP address is not on a Conficker-C filtered address range.

When a generated IP address fails to meet these criteria, the value for the contact list is not updated. The host will try to fill its list slots in order, using up to 100 attempts.

The speed at which UDP packets are sent out over the wire depends on the hardware and network capabilities of the infected host, as well as the amount of bandwidth, drop percentage, etc. of the network. The P2P protocol has a maximum of 1200 scanning connection attempts per minute, but observed accounts of Conficker-C P2P scan activity cite lower numbers. McAfee [11] reported seeing "roughly 2-3 UDP queries per second" ($\approx$ 130 per minute) during the 24 hours leading up to April 1, 2009. A Sophos technical report [7] notes that batches of 100 probes are generated on the wire, and that "probes in each batch are separated by small fixed intervals (2-5 seconds)". [14] performed a sandbox test of an infected Conficker-C host with a single network interface and observed scanning rates that start at approximately 1000 to 2000 IP addresses per 5 minute interval, and decrease over the first two hours of activity to a steady rate of approximately 200 IP addresses per 5 minutes. We base our model roughly on the SRI results, as they most thoroughly explain the time-dependence in scanning rate.

UDP connections. We model M_h as a Poisson process, which is a reasonable model for small-packet scanning activity programmed at regular intervals. [12] note that self-similarity is more common in packet inter-arrival times once connections have been established. Also, in their sandbox experiment, [14] show relatively smooth scanning rates, within both 30-minute and 6-hour time frames.

The marginal model is constructed to minimize dependencies between parameters from hour to hour, so that each population estimate can be calculated using an aggregated count from that hour alone. The goal for this model is not to track individual hosts, but to average over a wide range of possible behaviors in each hour. To that end, we take a simplified approach to the time-dependency of the UDP scanning rate, by defining a latent class η_h as one of three states that an active, infected host can be in for a particular hour h:

1. $\eta_h =$"Start-up" (S): The host comes online and initiates P2P scanning in this hour. This state is characterized by a high scan rate per minute and a small peer list, with activity commencing at some point t within the hour.
2. $\eta_h =$"Running" (R): The host has initiated start-up and is actively scanning for the entire hour. This state is characterized by a low scan rate per minute, a middle-sized to large peer list, and scans occurring throughout the hour.
3. $\eta_h =$"Shut-down" (D): The host has been actively scanning and goes offline during this hour. This state is characterized by a low scan rate per minute, a large peer list, and scans terminating at some point t within the hour.

Each of these states depends on three quantities: scan rate (ϕ), active minutes (t), and number of peers (n), that vary from hour to hour. We describe this variability mathematically using the prior distributions in Table 1. We suppress the index h for ease of notation.

Table 1. Prior distributions by active state

s	$\pi(\phi)$	$\pi(t)$	$\pi(n)$
Start-Up	$\Gamma(\mu_{\phi S} = 130, \sigma_{\phi S} = 20)$	Unif$(1, 60)$	TrGeo$(2048, \alpha_S = 0.950)$
Running	$\Gamma(\mu_{\phi R} = 40, \sigma_{\phi R} = 15)$	$t = 60$ w.p. 1	TrGeo$(2048, \alpha_R = 0.999)$
Shut-Down	$\Gamma(\mu_{\phi D} = 40, \sigma_{\phi D} = 10)$	Unif$(1, 60)$	TrGeo$^*(2048, \alpha_D = 0.999)$

$(*)$ this prior is defined as $\pi(2048 - n)$

Figure 2 shows the Gamma prior $\pi(\phi)$ for each state. Gamma distributions are often used to model the mean of a Poisson process, as they have strictly positive ranges. The mean rates $\mu_{\phi s}$ decrease from Start-Up through Shut-Down, and the standard deviations $\sigma_{\phi s}$ decrease to account for less stable behavior in Start-Up that gradually settles down to the more stable Running and Shut-Down states. Discrete Uniform priors on t represent the total number of active minutes in the Start-Up or Shut-Down states. Truncated Geometric distributions (geometric distribution restricted to a minimum and maximum value) are used for peer list counts. For the shut-down state, the $(*)$ indicates that the truncated geometric distribution is defined on the range $2048 - n$. The hyperparameters

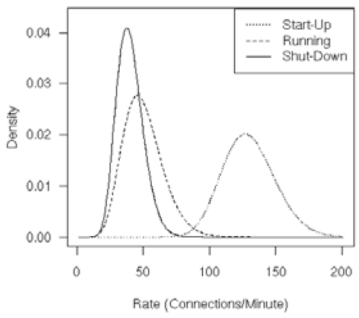


Fig. 1. Prior distributions for UDP scan rates by active state

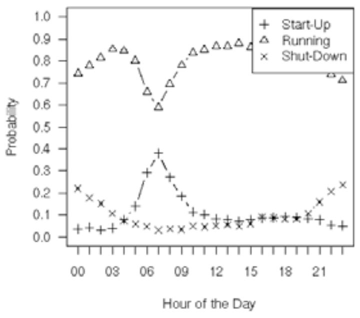


Fig. 2. Prior probabilities π_{ks} by relative hour of the day

$\pi(n)$ correspond to mean peer list sizes of approximately 20, 700, and 1350 for Start-Up, Running, and Shut-Down states.

We assume that the number of network connections (w) for an infected host does not change between states; based on elicitation from experts we choose a truncated geometric distribution between 1 and 32 connections, with a mean value $\mu_w = 1.67$ network connections per active host. When these quantities are fixed or known, it follows that M has a Poisson distribution with conditional mean μ_M equal to $\phi t w$.

Again to minimize dependencies between hours, temporal trends in the scan rates are not instituted by a time series component in the single-host model, but by the prior probability of the active state, $\pi_{ks} = \pi(\eta_k = s), s \in \{S, R, D\}, k \in [0, 1, \cdots, 23]$, which varies with the time-zone corrected hour of the day. Intuitively, π_{ks} is an estimate of the proportion of active hosts in the population that are in each state at each time-zone corrected hour. There are 48 free parameters in this distribution, arising from two free parameters per relative hour to estimate the probability of $\{S, R, D\}$ under its sum-to-one constraint. Figure 2 summarizes the values used for π_{ks}. We describe the empirical method used to set these values in Section 4.

Observed Connections. Each of the M connection attempts either hits the monitored proportion δ of IP space, or it does not. Since scan connections are independent, identical events, the distribution $\pi(y \mid M)$ is Binomial with parameters $M$and p, where p is the probability a connection attempt falls within the monitored space.

To determine p, we assume that the monitored space resides completely within the connection criteria from Section 2, and that it does not contain any infected peers listening on Conficker-C's designated ports. The monitored space must be free of infected machines to ensure that the only way of reaching the monitored space is through a completely random selection of an IP from all of IPv4 space; infected hosts may also reside on peer lists, which we do not choose to model.

With a set of n existing peers, the probability that each connection attempt is generated randomly is $1 - \gamma_n$. If the connection attempt is generated randomly, the probability that it falls in the monitored region is equal to δ/C, where $C = 0.995$ is the approximate proportion of IP space covered by Conficker-C's connection criteria, under the assumption of 1 broadcast address per 256 addresses in the space outside of Conficker-C's internal blacklist and ignored space. From these calculations p is equal to:

$$p_{n\delta} = \frac{(1-\gamma_n)\delta}{C}, \tag{2}$$

where δ is the proportion of monitored IP space.

From a well-known distributional result (see e.g. [18], ch 5, thm 1.2), the marginal distribution of y over all values of M, holding other unknown quantities fixed, is Poisson:

$$\pi(y \mid t, \phi, w, p_{n\delta}) = e^{-\phi t w p_{n\delta}} \frac{(\phi t w p_{n\delta})^y}{y!} \ . \tag{3}$$

Though we have described the model in hierarchical stages, in practice we are interested only in the unconditional distribution of y, which is difficult to express analytically. But, the hierarchical structure of the model makes it easy to obtain a large sample $(y)_1, \cdots (y)_B$ from this distribution, using simulation. For $b = 1$ to B, we do the following:

1. Draw a state η_b from $\{S, R, D\}$ using the prior probabilities π_{ks}, and draw a network connection w_b from $\pi(w)$.
2. Draw ϕ_b, n_b, and t_b using the prior distributions for η_b.
3. Draw y_b from the Poisson distribution with rate equal to $\phi_b t_b n_b w_b$.

We then use the observed proportions in $y_1, ..., y_B$ as Monte Carlo estimates of the marginal probability of y, accounting for prior uncertainty in the underlying parameters. This simulation can be performed easily using statistical packages for languages such as R, C, or Python.

3 Lévy's Central Limit Estimator $\hat{H}$

Suppose $y_1, ..., y_H$ are independent with distribution $\pi(y \mid \pi_{ks}, \mu_{\phi s}, \sigma_{\phi s}, \alpha_w, \alpha_n, \delta, k)$ as defined in Section 2. We will suppress the dependence on hyper-parameters in notation for this section. The population size H is unknown, and only $Y = \sum_{i=1}^{H} y_i$ is observed. We define the population estimator:

$$\hat{H} = \frac{Y}{\mu_y} \ . \tag{4}$$

We call $\hat{H}$ the Central Limit Estimator of H. This estimator has the following properties:

1. $E(\hat{H}) = \frac{1}{\mu_y} E(\sum_{i=1}^{H} y_i) = \frac{1}{\mu_y} H\mu_y = H$;
2. $\text{Var}(\hat{H}) = \frac{1}{\mu_y^2} \text{Var}(\sum_{i=1}^{H} y_i) = H \left(\frac{\sigma_y}{\mu_y}\right)^2$;
3. (Lévy result): The distribution of $\hat{H}$ is approximately Normal when H is sufficiently large.

The Lévy form of the Central Limit Theorem (see e.g. [2], Ch. 5, p 243) outlines conditions under which the sum of independent and identically distributed variables converges to a Normal distribution. Using this result, an approximate 95% confidence interval for H is:

$$\hat{H} \pm 1.96\sqrt{\hat{H}}\frac{\sigma_y}{\mu_y} \tag{5}$$

When $y_{k1}, ..., y_{kH_k}$ are identically distributed when grouped within relative hour of the day, $k \in [0, ..., 23]$, then an approximate 95% confidence interval of $\hat{H} = \sum_{k=0}^{23} \hat{H}_k$ is:

$$\hat{H} \pm 1.96\sqrt{\sum_{k=0}^{23} \hat{H}_k \left(\frac{\sigma_{yk}}{\mu_{yk}}\right)^2} \; . \tag{6}$$

We estimate μ_y and σ_y from simulations $y_1, \cdots y_B$ for $B = 1,000,000$, with the formulas:

$$\mu_y \approx \frac{1}{B}\sum_{b=1}^{B} y_b, \qquad \sigma_y^2 \approx \frac{1}{B-1}\sum_{b=1}^{B}(y_b - \mu_y)^2 \; . \tag{7}$$

In practice, B can be set large enough that the Monte Carlo sampling error in these estimates has little effect on the variance of $\hat{H}$. Alternatively, an approximation method such as the Delta method ([2], ch 7) can be used to account for this variability.

4 Analysis and Results

Data Collection. The monitored space in our experiment consists of a large private network comprising approximately 21000 /24 net blocks. To account for uncertainty in this size estimate as well as network availability, we also set an additional prior for δ, $\pi(\delta) = \text{Beta}(15, 13000)$, and add an extra simulation step in the calculation of $\pi(y \mid \pi_{ks}, \mu_{\phi s}, \sigma_{\phi s}, \alpha_w, \alpha_n, \delta, k)$. This corresponds to a mean μ_δ of 0.0012.

Using the SiLK Conficker.C Plug-In [5], we obtained historical records of UDP connection requests with the Conficker-C signature sent into the monitored network space from external hosts over the period from March 5th through April 24th, 2009. We recorded the total number of incoming UDP connection attempts for each external IP address per hour, and aggregated these counts to the /24 level to attempt to account for ephemeral DHCP leases within subnets. A total

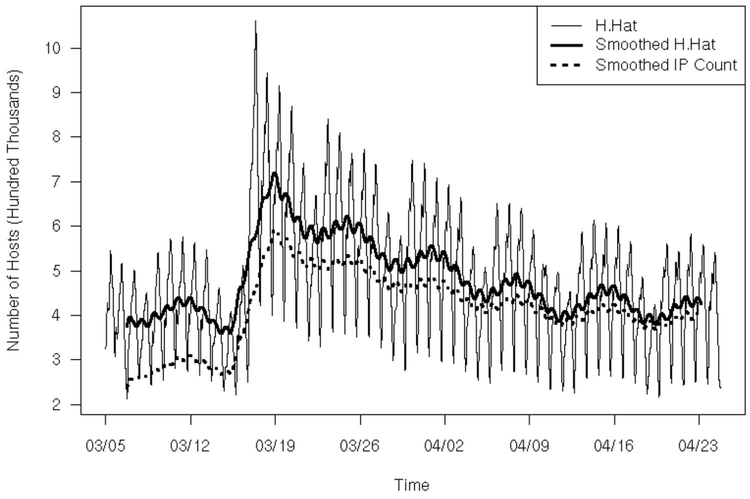


Fig. 3. $\hat{H}$ per hour over the 2-month span

of 1091013 external /24 net blocks were observed performing Conficker-C UDP scans.. Each net block was assigned roughly to a time zone based on the country code associated with that block, with 1% of blocks remaining unassigned due to satellite locations or unavailable country codes.

Population Estimates. Figure 3 shows the estimates of H_h for the two-month span starting on March 5th, and ending April 2 4th. 95% confidence bands for the hourly counts, calculated using equation 6, are on the order of under ±10000 and are too tight to be seen on the figure. The large jump occurs on March 17th and corresponds to a binary update released into the Conficker-C botnet. The largest host count associated with the botnet is 1.06 million active hosts. Numbers decline steadily through the month of April, but appear to stabilize toward the end of the month. The overall decline occurs because Conficker-C infections spread only among previously infected machines, with had no means of infecting new hosts.

The heavy lines correspond to a smoothed plot of both host count estimates (solid line), and observed unique IP address counts (dotted line). These lines show a trend that as the botnet ages, it "spreads out" among IP space. The ratio of hosts per IP–observable as the space between the two lines– is large prior to the update in mid-March, but declines steadily afterward. This decline makes sense; large infected networks (often behind proxies) would propagate local infections quickly, while isolated hosts would take longer to reach with P2P bootstrapping. The effect would also appear as larger corporate networks work to clean up enclaves of local infections, suggesting that the persistent infections of Conficker-C are among more isolated machines in IPv4 space.

Hyperparameters and Prior Sensitivity. The prior values π_{sk} can be estimated empirically using a random sample of infected *hosts* from the Conficker-C population. To approximate such a sample, we sampled a set of 1000 /24 blocks from the observed set of 1.09 million, each with probability proportional to the observed average scan rate. In 71% of the sample, the activity behind the sampled net block was sparse enough to roughly equate one block with one active host, and to estimate active from non-active hours. An active hour following two inactive hours, or an active hour preceded and followed by two inactive hours, was classified as "Start-Up". An active hour preceded by at least one active hour in the past two, and followed by two inactive hours was classified as "Shut-Down". All other active hours were classified as "Running". The counts of observed blocks in each state, normalized over hour by time zone, were used as the values for π_{sk}. This method used simple heuristics as opposed to formal models for estimating an active state, but the resulting prior probabilities display a reasonable and intuitive pattern in Figure 2.

The scaling factor $\frac{\sigma_y}{\mu_y}$ was close to 1 for all hours, with the highest value of 1.08 occurring at 8am, relative time. This result indicates that the $\sqrt{H}$ term dominates the confidence interval for $\hat{H}$. The value $1.96\sqrt{H}$ is a very tight bound relative to the size of the population estimate, but its precision is predicated on an unbiased model for $\pi(y \mid \pi_{ks}, \mu_{\phi s}, \sigma_{\phi s}, \alpha_w, \alpha_n, \delta, k)$. Small shifts in the hyperparameters may have a large influence on $\hat{H}$. This suggests that measurement of the uncertainties and behavioral quantities making up a single-host model should be well-informed and precise to take advantage of this simple estimator. We used a reasonable and informed set of 67 hyperparameters $(\pi_{ks}, \mu_{\phi s}, \sigma_{\phi s}, \alpha_w, \alpha_n, \delta, k)$ in this model, and we opted not to model any further levels of uncertainty with probability distributions. In the future, the model can be easily adapted to include another hierarchical level of priors for these hyperparameters, allowing us examine the sensitivity of population estimates to the choice of hyperparameters.

5 Summary and Discussion

A marginal probability model of single-host behavior provides a way of measuring populations based on the number of active infected machines, as opposed to counting net blocks or IP addresses. The model is based on a set of hyperparameters that can be independently measured or assessed based on protocol and network activity profiles. By characterizing this distribution precisely, and applying the Central Limit Theorem, we obtain both a point estimate of the population, and a confidence interval that accounts for variability arising from both the stochastic elements of the protocol and from uncertainty across multiple measurements. In the future we hope to expand the estimation methodology to a fully Bayesian scheme that incorporates priors for the chosen hyperparameters and that allows for the calculation of posterior distributions of the current model hyperparameters given observed data y, making the model more robust to parameter misspecification. We also hope to develop a full mark-recapture model for comparison with the expanded marginal model.

References

1. Abu Rajab, M., Zarfoss, J., Monrose, F., Terzis, A.: My botnet is bigger than yours (maybe, better than yours): Why size estimates remain challenging. In: Proceedings of the First Annual Workshop on Hot Topics in Botnets (March 2007)
2. Casella, G., Berger, R.: Statistical Inference. Duxbury Press, Boston (1990)
3. Chan, M., Hamdi, M.: An active queue management scheme based on a capture-recapture model. IEEE Journal on Selected Areas in Communications 21(4), 572–583 (2003)
4. Dupuis, J., Schwarz, C.: A Bayesian approach to the multistate Jolly-Seber capture-recapture model. Biometrics 63, 1015–1022 (2007)
5. Faber, S.: Silk Conficker. C Plug-in (2009), CERT Code release, http://tools.netsa.cert.org/wiki/display/tt/SiLK+Conficker.C+Plugin
6. Fienberg, S., Johnson, M., Junker, B.: Classical multilevel and bayesian approaches to population size estimation using multiple lists. Journal of the Royal Statistical Society: Series A 162(3), 383–405 (1999)
7. Fitzgibbon, N., Wood, M.: Conficker.C: A technical analysis (March 2009), Sophos white paper, http://www.sophos.com/sophos/docs/eng/marketing_material/conficker-analysis.pdf
8. Horowitz, K., Malkhi, D.: Estimating network size from local information. Information Processing Letters 88, 237–243 (2003)
9. Li, Z., Goyal, A., Chen, Y., Paxson, V.: Automating analysis of large-scale botnet probing events. In: ASAICCS 2009 (March 2009)
10. Mane, S., Mopuru, S., Mehra, K., Srivastava, J.: Network size estimation in a peer-to-peer network. Tech. Rep. TR 05-030, University of Minnesota Department of Computer Science and Engineering (2005)
11. McAfee: Conficker.C over the wire. McAfee Network Security blog publication (March 2009), http://www.avertlabs.com/research/blog/index.php/2009/04/01/confickerc-on-the-wire-2
12. Paxson, V., Floyd, S.: Wide-area traffic: The failure of poisson modeling. IEEE/ACM Transactions on Networking 3(3), 226–244 (1995)
13. Porras, P., Saidi, H., Yegneswaran, V.: Conficker C Actived P2P scanner. SRI international Code release/document (2009), http://www.mtc.sri.com/Conficker/contrib/scanner.html
14. Porras, P., Saidi, H., Yegneswaran, V.: Conficker C analysis. Tech. rep., SRI International (2009)
15. Porras, P., Saidi, H., Yegneswaran, V.: Conficker C P2P protocol and implementation. Tech. rep., SRI International (2009)
16. Psaltoulis, D., Kostoulas, D., Gupta, I., Briman, K., Demers, A.: Decentralized schemes for size estimation in large and dynamic groups. Tech. Rep. UIUCDCS-R-2005-2524, University of Illinois Department of Computer Science (2005)
17. Schwarz, C., Arnason, A.: A general methodology for the analysis of capture-recapture experiments in open populations. Biometrics 52(3), 860–873 (1996)
18. Taylor, H., Karlin, S.: An Introduction to Stochastic Modeling. Academic Press, London (1998)

Network DVR: A Programmable Framework for Application-Aware Trace Collection

Chia-Wei Chang[1], Alexandre Gerber[2], Bill Lin[1],
Subhabrata Sen[2], and Oliver Spatscheck[2]

[1] University of California, San Diego, La Jolla, CA
[2] AT&T Labs-Research, Florham Park, NJ

Abstract. Network traces are essential for a wide range of network applications, including traffic analysis, network measurement, performance monitoring, and security analysis. Existing capture tools do not have sufficient built-in intelligence to understand these application requirements. Consequently, they are forced to collect *all* packet traces that might be useful at the finest granularity to meet a certain level of accuracy requirement. It is up to the network applications to process the per-flow traffic statistics and extract meaningful information. But for a number of applications, it is much more efficient to record packet sequences for flows that match some application-specific signatures, specified using for example regular expressions. A basic approach is to begin memory-copy (recording) when the first character of a regular expression is matched. However, often times, a matching eventually fails, thus consuming unnecessary memory resources during the interim. In this paper, we present a programmable *application-aware* triggered trace collection system called Network DVR that performs precisely the function of packet content recording based on user-specified trigger signatures. This in turn significantly reduces the number of memory copies that the system has to consume for valid trace collection, which has been shown previously as a key indicator of system performance [8]. We evaluated our Network DVR implementation on a practical application using 10 real datasets that were gathered from a large enterprise Internet gateway. In comparison to the basic approach in which the memory-copy starts immediately upon the first character match without *triggered-recording*, Network DVR was able to reduce the amount of memory-copies by a factor of over 500x on average across the 10 datasets and over 800x in the best case.

1 Introduction

Accurate trace collection of network traffic is the foundation of a wide range of network monitoring tasks. Traditionally, packet capture tools (e.g., TCPdump [1], Ethereal [2], Libpcap [3], and WinPcap [4]) are primarily focused on collecting and reconstructing packet sequences for flows by matching packets against simple packet header rules, such as the source/destination IP addresses, the port numbers, or the transmission protocol. The collected packets are often delivered to remote servers where monitoring and management applications can perform post-processing, as depicted in Fig. 1. Although the traditional network monitoring architecture has had some success in offering comprehensive insights about network traffic, the scalability of this architecture is limited

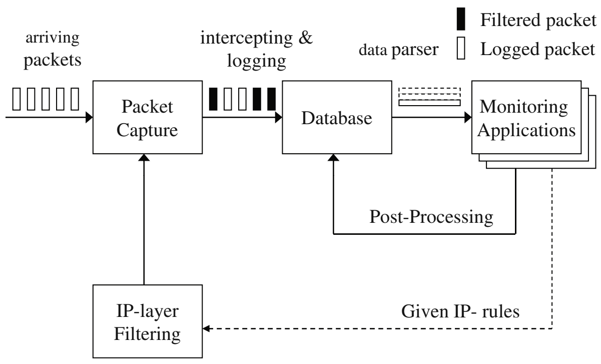


Fig. 1. Simplified traditional network monitoring architecture

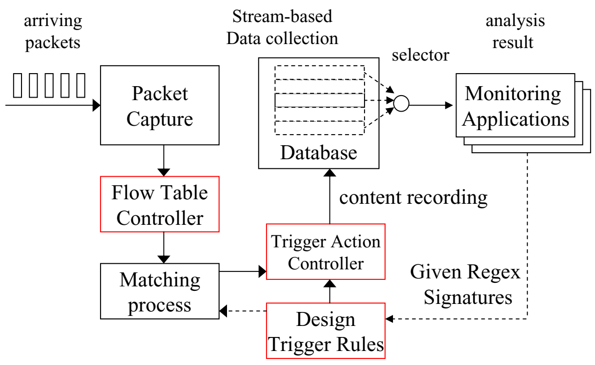


Fig. 2. Programmable network monitoring architecture

in practice. In today's high-speed network, especially in core networks, the amount of traffic can be immense, with possibly millions of flows. Therefore, recording all packet traces in details is prohibitive in most cases.

In this paper, we present a programmable *application-aware* trace collection architecture called *Network DVR* that can perform the *recording* of *packet contents* highly selectively by matching captured packets against application-specific signatures. This is depicted in Fig. 2. The name Network DVR is loosely analogous to a digital video recorder for television that can be intelligently programmed to record discriminately. Our design of Network DVR is based on a novel concept of *triggered-recording* that allows a user to flexibly define rules for triggering the *start* and *termination* of packet content recordings. In particular, Network DVR can be programmed to only begin recording when some *start rule* has been matched, which can significantly reduce the likelihood of recording matchings that will eventually fail. We have applied our Network DVR approach on a practical example application by using 10 real datasets that were gathered from a large enterprise Internet gateway. Our evaluations show that

Network DVR can dramatically reduce the amount of memory-copies by a factor of over 500x on average across the 10 datasets and over 800x in the best case.

The rest of the paper is organized as follows. Section 2 presents a high-level overview of our proposed triggered trace collection approach called Network DVR. Section 3 presents evaluation results. Section 4 discusses related work. Finally, Section 5 concludes the paper.

2 Proposed Triggered Trace Collection Concept

In this section, we present the concept of application-aware *triggered* trace collection, which aims to record packet contents based on application-specific signatures that control when the recordings should *start* and *abort*. These signatures are captured as regular expressions. Although regular expression matching has been widely used for intrusion detection, regular expression matching is used differently in our context to trigger the start and termination of packet content recordings. Specifically, we define three trigger rulesets, one that defines when the recording module should *start* recording, one that defines when the recording should *abort*, and one that defines if a recording is a valid *final match*. These three regular expression rulesets are respectively called the *start* ruleset (Ω_α), the *abort* ruleset (Ω_β), and the *final match* ruleset (Ω_γ)). Incoming traffic can be matched against these trigger rulesets by means of a deterministic finite automaton (DFA). In particular, each *accept state* will correspond to one or more rules matched from these three rulesets. Accordingly, the corresponding *start*, *abort*, and/or *finalized* messages would be triggered to control the recording behavior, depending to which rulesets the matching rules belong. To simplify the presentation, we will assume a conventional DFA representation for these rules. However, in general, our framework can make use of any state-of-the-art DFA variants [21, 11, 20, 10, 7, 16, 9] for efficient representation and matching.

In the following, Section 2.1 describes how trigger rulesets are constructed, Section 2.2 presents the high-level triggered trace collection procedure, and Section 2.3 presents our memory management scheme.

2.1 Trigger Rulesets Construction

An essential part of trigger trace collection is the trigger ruleset construction (i.e., to build the *start* (α), *abort* (β) and *final match* (γ) rules, which control when to activate, abort, or finalize the trace collection). Naturally, these trigger rules can be defined directly by the monitoring applications, but these rules can also conceivably be automatically generated in a systematical way from a set of application signatures.

Consider the examples shown in Fig. 3. The first monitoring application is for tracking valid URLs in the form of "http://.*\.edu" to find say the "top 100" sites of educational institutions on the web, excluding non-educational ".com" and ".org" sites. The second monitoring application aims to identify the characters except carriage returns or new line characters between the two sub-patterns "Del" and "ATT" (this example is truncated from a Snort ftp rule). Each monitoring application signature generates its individual trigger rule set and each signature is mainly split into two parts, a prefix part and a suffix part. Generally, the prefix part serves as a start condition for the recording

Application-Specific Signatures

Monitor Application 1
Match `http://.*\.edu` Don't match `http://.*\.com` or `http://.*\.org`

Monitor Application 2
Match `Del[^\r\n]*ATT`

Design Trigger Rules

Monitor Application 1
$MA_1 = \{ \alpha_1, \beta_{11}, \beta_{12}, \gamma_1 \}$ $\alpha_1 =$ `http://` $\beta_{11} =$ `\.com` $\beta_{12} =$ `\.org` $\gamma_1 =$ `\.edu`

Monitor Application 2
$MA_2 = \{ \alpha_2, \beta_{21}, \beta_{22}, \gamma_2 \}$ $\alpha_2 =$ `Del` $\beta_{21} =$ `\r` $\beta_{22} =$ `\n` $\gamma_2 =$ `ATT`

Corresponding Rulesets

Ruleset	Rules
Start Ω_α	$\alpha_1 =$ `http://` $\alpha_2 =$ `Del`
Abort Ω_β	$\beta_{11} =$ `\.com` $\beta_{12} =$ `\.org` $\beta_{21} =$ `\r` $\beta_{22} =$ `\n`
Final Ω_γ	$\gamma_1 =$ `\.edu` $\gamma_2 =$ `ATT`

Fig. 3. Construction of trigger rulesets

while the suffix part serves as a finish condition for the recording. In addition, there may be conditions that will allow us to abort a recording. For example, the complement syntax, "[^]", is used to indicate the characters that should be *excluded* from a match. In such cases, these exclusive characters can serve as abort conditions for the recording (e.g., "\r" and "\n" in application 2). In other cases, the abort conditions may be explicitly specified (e.g., "\.com" and "\.org" in application 1).

2.2 Triggered Trace Collection Procedure

Given the trigger rulesets, we construct a DFA to perform the matching. The DFA for the example depicted in Fig. 3 is shown in Fig. 4. The *accepting* states are shown in the shaded ovals (i.e., States 7, 11, 14, 17, 20, 23, 24, 25) with the corresponding matching rules specified. In order to implement cross-packet inspection, the matching module needs to remember for each flow the *last* state that the matching module visited in the DFA to serve as the starting state for the next packet for the same flow. To remember the last state visited for each flow, we use a flow state table where each entry corresponds to a flow, and the value of the entry is the "last visited" state of the flow in DFA. The flow state table can be implemented as a hash table and a new hashed flow can be initialized to the initial state (e.g., State 0).

With every incoming packet, the matching module will find its corresponding last visited state in the flow state table and adjust the matching procedure to start at the right position of the packet payload based on the header information. Each symbol is read sequentially from the packet payload and transferred to the next state by following the DFA structure. If it matches a *start* rule, the symbol-copy/recording behavior will be activated, also the matching index and the recording-begin memory position will be logged. If this flow is under the recording process, for each symbol, the memory allocator will unlink one memory cell from the free memory cell list, copy the symbol, link it to the temporary recording strings. After a corresponding *abort* rule is matched, this temporary recording string will be recycled and connected back to the free list

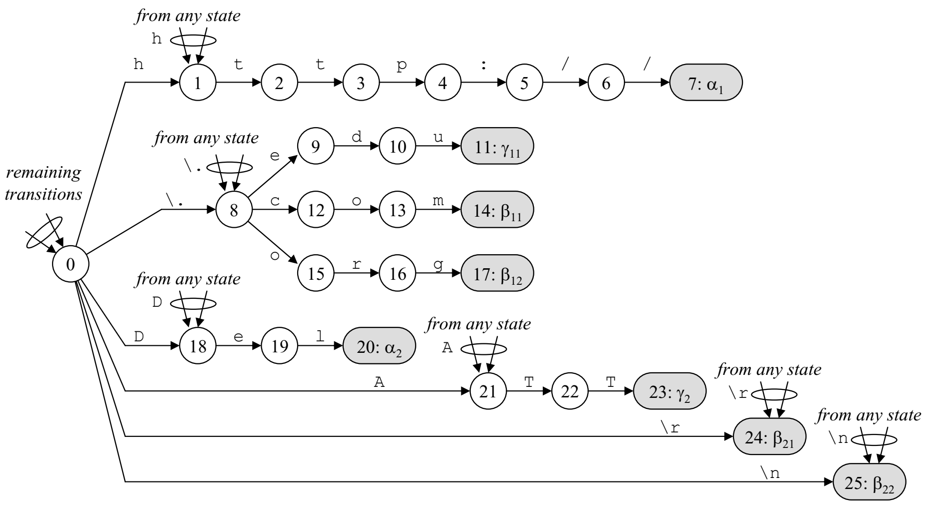


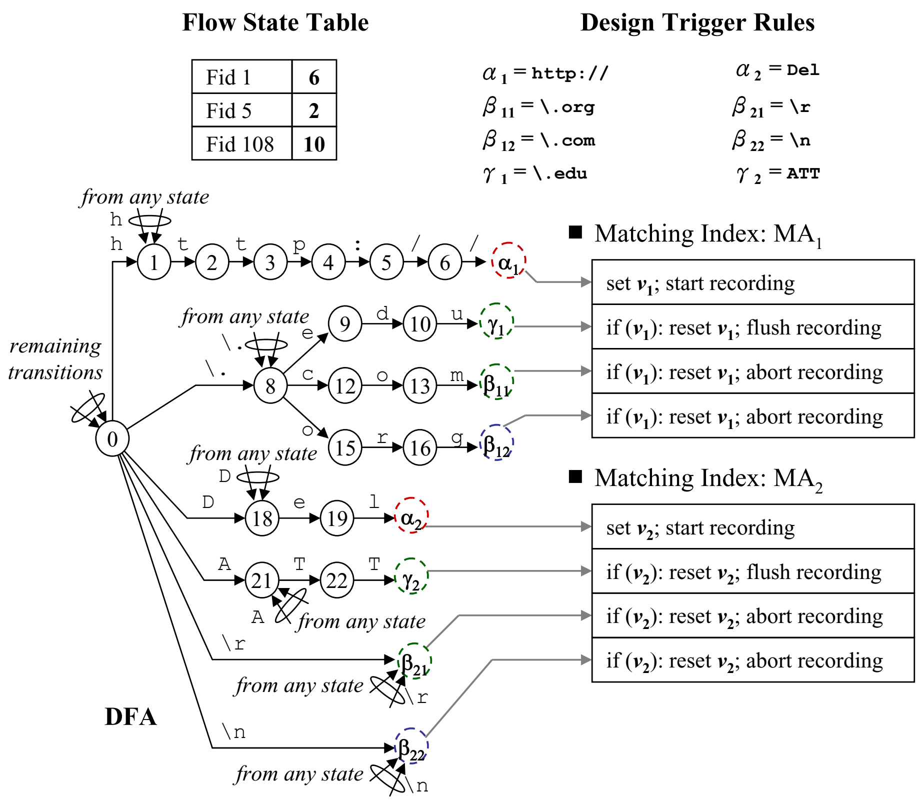
Fig. 4. Compiled DFA

of memory cells. On the other hand, if a corresponding *final match* rule is matched, the temporary recording string will be appended to a flush queue for writing to disk, and its recording-end memory position will be logged. Here, "corresponding" means for the trigger rules either in *start*, *abort*, or *final match* ruleset that come from the same application-specific signature (e.g., have the same matching index).

In particular, for each application-specific signature MA$_i$ = $\{\alpha_i, \beta_{i1} ... \beta_{ij}, \gamma_{i1} ... \gamma_{ik}\}$, there is a corresponding variable v_i that gets set where its corresponding start rule has been matched. This is depicted in Fig. 5. Upon encountering an abort or a final match rule, we check if there is an active recording for this rule by testing v_i. If it is set, then the corresponding actions for abort or final match are to reset v_i, and the recording is either aborted or flushed, respectively. Since we support recordings on a per-flow basis, we must keep track of a separate set of these v_i variables for each flow. This can be dynamically managed using a hash table. In particular, to test if v_i is set for flow f, we can perform a hash lookup on the key f:v_i, where the key is constructed by combining the flow ID and the variable name. To set v_i for flow f, we can perform a hash insert (or lookup-then-insert) with the key f:v_i. Finally, to reset v_i for flow f, we can perform a hash delete with the same key f:v_i. Since we are not storing a value in this hash table, this hash table can as well be efficiently using a counting Bloom filter [22].

Note that multiple recordings may be triggered for a single flow if multiple *start* rules have been matched. Therefore, the worst-case bound on memory bandwidth/processing time is $O(N)$, where N is the number of given total application-specific signatures[1]. Fortunately, instead of having multiple recording strings for each matched pattern in the *start* ruleset, we use a single aggregated recording string for each flow to guarantee there is always one memory copy for each incoming symbol. By logging the

[1] Here the memory bandwidth requirement is expressed in terms of the number of memory operations (e.g., copies) to be performed for each input character processed.



Flow State Table

Fid 1	**6**
Fid 5	**2**
Fid 108	**10**

Design Trigger Rules

α_1 = `http://`	α_2 = `Del`
β_{11} = `\.org`	β_{21} = `\r`
β_{12} = `\.com`	β_{22} = `\n`
γ_1 = `\.edu`	γ_2 = `ATT`

■ Matching Index: MA_1

set v_1; start recording
if (v_1): reset v_1; flush recording
if (v_1): reset v_1; abort recording
if (v_1): reset v_1; abort recording

■ Matching Index: MA_2

set v_2; start recording
if (v_2): reset v_2; flush recording
if (v_2): reset v_2; abort recording
if (v_2): reset v_2; abort recording

Fig. 5. Flow state table, DFA, and actions

recording-begin/end memory positions of each valid matching result in the aggregated recording string, the system can output all recorded matching strings for each application-specific signature.

2.3 Constant Time Memory Allocation Structure

Initially, the memory allocator maintains a free-list of memory cells as a linked list, and each unit of memory cell can hold one byte (for a character) and a pointer. Depending on the implementation, it may be more memory efficient to have the memory allocation granularity be multiple bytes for each memory cell instead of one. For simplicity of presentation, we will assume each memory cell holds one symbol. For recording, the recording module stores the string being recorded as a linked list of symbols as well. When the recording string needs more memory, the memory allocator unlinks cells at the front of the free list sequentially and puts them to the tail of the linked list corresponding to the recording string (constant time memory allocation). It can connect the entry of the bitmap table to the recording string by setting a pointer at the bitmap table entry to the head of the corresponding linked list.

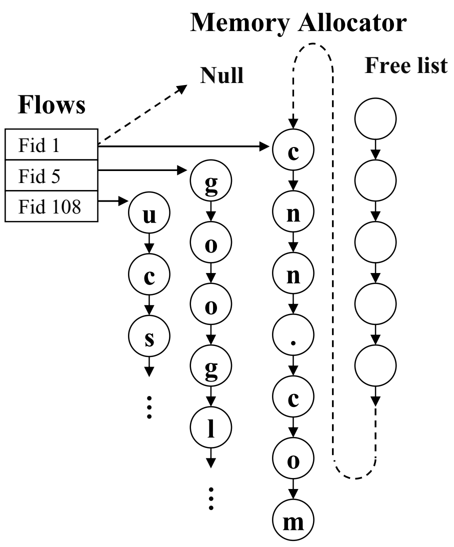


Fig. 6. Trace collection module

Suppose this recording string is confirmed as a valid recording. The memory allocator can simply put the head of this linked list to an output queue for flushing to disk and reset the flow state entry pointer to NULL. After written to disk, it can be linked back to the tail of the free-list (constant time recovery of the memory cells). If a recording is aborted (e.g. Fid 1 records "cnn.com" which is not ended by ".edu" in Fig. 6), the memory allocator will directly set the flow state entry pointer to NULL, and link this linked list to the tail of the free-list, which again is a constant time operation.

3 Evaluation

In this section, we evaluate the proposed Network DVR framework. We show the scalability of the modified regular expression extraction module, which uses an application-aware trigger mechanism to control the recording process. We then demonstrate the performance that Network DVR can achieve by measuring the number of memory copies consumed during the recording process. Indeed, for an initial evaluation, the number of memory copies is a good proxy for the future performance of a real time system.

3.1 Simulation Setup

We gathered real data traces from a couple of servers for a day by using Gigascope [8]. These traces were collected on a trunk with four 1-Gigabit Ethernet links at a large enterprise Internet gateway by using IP-filtering for specific IP-groups in which we are interested, and we used these traces to verify our application-aware data collection approach. The collected traces were then partitioned into 10 datasets based on the packets collected every 60 minutes. Each of these datasets has approximately 3,500 flows. For each dataset, given the application-specific signatures, we replay the complete trace

Table 1. Evaluate the number of memory copies needed for Snort rules

	`data`	`basic`	`netDVR`	`actual`	`reduction`	`overhead`
Set 1	7.00×10^8	3.13×10^7	7.06×10^4	4.69×10^4	444.04	1.51
Set 2	6.50×10^8	2.89×10^7	6.39×10^4	4.38×10^4	451.64	1.46
Set 3	6.60×10^8	3.09×10^7	7.22×10^4	4.63×10^4	427.74	1.56
Set 4	6.30×10^8	2.84×10^7	6.96×10^4	4.68×10^4	407.39	1.49
Set 5	6.10×10^8	2.74×10^7	6.83×10^4	4.50×10^4	401.05	1.52
Set 6	7.90×10^8	3.60×10^7	5.07×10^4	3.16×10^4	709.49	1.60
Set 7	7.60×10^8	3.53×10^7	7.77×10^4	5.12×10^4	453.73	1.52
Set 8	7.60×10^8	3.25×10^7	8.24×10^4	6.44×10^4	394.35	1.28
Set 9	8.20×10^8	3.44×10^7	5.66×10^4	4.08×10^4	607.21	1.39
Set 10	9.00×10^8	3.77×10^7	4.66×10^4	3.09×10^4	808.92	1.51
				average	510.56	1.48
				max	808.92	1.60

and calculate the number of memory copies that Network DVR needs and compare the result to a basic approach in which the recording starts when the first character of a regular expression is matched. We have employed an efficient public domain DFA variant provided by Becchi and Crowley [7] to serve as the matching module in Network DVR, but we note that other efficient DFA variants [21, 11, 20, 10, 16, 9] may be used in our framework as well.

For our evaluation, we considered a practical application in which Network DVR is used to record the details of matched results for an intrusion detection system (IDS). To perform this recording task without our trigger concept, current IDSs would be forced to begin copying symbols to memory when the first symbol of a regular expression is matched. Here, we chose one category of signatures from the Snort 2007 signature set [12] for our evaluation. In particular, we considered the ftp signatures (consisting of 58 regular expressions) for our experiment to evaluate the amount of unnecessary memory copies that Network DVR can reduce by using the proposed triggered-recording concept. For each signature, we manually decomposed the signature into start, abort, and final match rules. Specifically, we used the prefix of each signature as a start rule, the suffix of each signature as a final match rule, and any exclusion characters as abort rules.

3.2 System Performance Comparison on Memory Copy Times

Table 1 summarizes the results for the 10 datasets considered in terms of memory copies. The column labeled `data` shows the size of the total incoming symbols which is replayed by using the real traces collected by Gigascope [8] using IP-filtering only (unaware of the application). The column labeled `basic` shows the results for a basic approach that begins copying symbols to memory when the first character of a regular expression is matched. The column labeled `netDVR` shows the results using our triggered-recording approach. The column labeled `actual` shows the total number of times actual memory copies were needed for successful matches. The column labeled `reduction` shows the reduction factor in memory copies that `netDVR` can achieve relative to the `basic` approach (i.e., `reduction` = `basic` / `netDVR`). The last column labeled `overhead` shows the overhead factor in memory copies that `netDVR` incurs relative to the `actual` memory copies for successful matches (i.e., `overhead` = `netDVR` / `actual`).

As can be seen in Table 1, our Network-DVR approach can achieve a reduction factor of over 500x across the 10 datasets and over 800x in the best case (i.e., for Set 10). This reduction is achieved by only activating the recording when a start rule has been matched. However, even with this start ruleset pre-filtering, it may still be possible that a matching eventually fails. This accounts for the difference in results between the `netDVR` column and the `actual` column. However, this overhead is only a factor of 1.48x on average with 1.6x in the worst case.

4 Related Work

Packet recordings gathered by core routers provide valuable coarse-granularity traffic information for a variety of measurement-related applications. However, because of the large volumes of traffic, filtering unnecessary data becomes an important issue. Packet filtering at the IP-header level is among the first techniques applied to solve this issue [13, 14]. BLINC [15], which observes and identifies patterns of host behavior at the transport layer, is designed for traffic classification. Deep packet inspection (DPI) [7, 9] methods, which identify and classify traffic based on a signature database that includes information extracted from the data part of a packet, allows for finer control than classification based only on header information. These approaches are what we describe as the "basic" implementations, which start copying symbols to memory when the first symbol of a regular expression is matched in order to output the details of the matching results. Our approach performs much better by triggering the memory copies later.

ProgME [17] introduces flowsets based on packet headers for defining aggregates in the context of network measurements (specifying counting). ATMEN, a triggered measurement infrastructure to communicate and coordinate across various administrative entities, is proposed in [18] to reduce wasted measurements by judiciously reusing measurements along three axes: spatial, temporal, and application. Another approach, in which any particular application can express the classes of traffic of its interest, is proposed in [19] for application-specific flow sampling in many monitoring applications, such as SNORT [12], BLINC [15]. Although the above methods make flow recording feasible by using traffic classification, faster deep packet inspection, hardware pre-matching, concept of flowsets or application-aware packet sampling, none of them perform content-state-aware filtering of packet content for a given monitoring application.

5 Conclusion

In this paper, we presented Network DVR, a framework for a programmable content-aware packet trace collection system. Our main contribution is programmable recording framework that can be programmed to record packet traces that are relevant to a given monitoring application. Our framework employs a trigger recording concept that defers recording until some start conditions have been matched, which significantly minimizes the number of memory copies that the system consumes for valid trace collection (a key indicator of performance). Modules with knowledge about application requirements enable Network DVR to collect traffic data in accordance to the application at hand. Our evaluation using real datasets from a large enterprise Internet gateway shows that Network DVR can collect relevant packet traces effectively in a concise manner and also reduce the amount of memory copies dramatically. Given these encouraging results, we plan to develop a real-time implementation of Network DVR with larger sets of application signatures to better quantify the actual performance improvement of our proposed approach.

References

[1] Jacobson, V., Leres, V.C., Mccanne, S.: The TCPdump Manual Page. Lawrence Berkeley Laboratory, Berkeley (1989)
[2] Orebaugh, A.D., Ramirez, G.: Ethereal Packet Sniffing. Syngress Publishing (2004)
[3] Libpcap, http://www.tcpdump.org/#documentation
[4] WinPcap: The Windows Packet Capture Library, http://www.winpcap.org/
[5] Estan, C., Varghese, G.: New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. ACM Trans. Computer Systems (2003)
[6] PCRE: Perl Compatible Regular Expressions, http://www.pcre.org/
[7] Becchi, M., Crowley, P.: A hybrid finite automaton for practical deep packet inspection. CoNEXT (2007)
[8] Cranor, C., Johnson, T., Spataschek, O., Shkapenyuk, V.: Gigascope: a stream database for network applications. ACM SIGMOD (2003)
[9] Smith, R., Estan, C., Jha, S., Kong, S.: Deflating the big bang: fast and scalable deep packet inspection with extended finite automata. ACM SIGCOMM (2008)
[10] Yu, F., Chen, Z., Diao, Y., Lakshman, T.V., Katz, R.H.: Fast and memory-efficient regular expression matching for deep packet inspection. ACM ANCS (2006)
[11] Kumar, S., Dharmapurikar, S., Yu, F., Crowley, P., Turner, J.: Algorithms to accelerate multiple regular expressions matching for deep packet inspection. ACM SIGCOMM (2006)
[12] Snort., http://www.snort.org
[13] Moore, A.W., Zuev, D.: Traffic classification using bayesian analysis techniques. SIGMETRICS (2005)
[14] Kundu, S., Pal, S., Basu, K., Das, S.: Fast classification and estimation of Internet traffic flows. In: Uhlig, S., Papagiannaki, K., Bonaventure, O. (eds.) PAM 2007. LNCS, vol. 4427, pp. 155–164. Springer, Heidelberg (2007)
[15] Karagiannis, T., Papagiannaki, K., Faloutsos, M.: BLINC: multilevel traffic classification in the dark. ACM SIGCOMM (2005)
[16] Becchi, M., Cadambi, S.: Memory-efficient regular expression search using state merging. INFOCOM (2007)
[17] Yuan, L., Chuah, C.N., Mohapatra, P.: ProgME: towards programmable network measurement. ACM SIGCOMM (2007)
[18] Krishnamurthy, B., Madhyastha, H.V., Spatscheck, O.: ATMEN: a triggered network measurement infrastructure. ACM WWW (2005)
[19] Madhyastha, H.V., Krishnamurthy, B.: A generic language for application-specific flow sampling. ACM CCR (2008)
[20] Kumar, S., Turner, J., Williams, J.: Advanced Algorithms for Fast and Scalable Deep Packet Inspection. ACM ANCS (2006)
[21] Tuck, N., Sherwood, T., Calder, B., Varghese, G.: Deterministic memory-efficient string matching algorithms for intrusion detection. INFOCOM (2004)
[22] Bonomi, F., Mitzenmacher, M., Panigrahy, R., Singh, S., Varghese, G.: An improved construction for counting Bloom filters. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 684–695. Springer, Heidelberg (2006)

OpenTM: Traffic Matrix Estimator for OpenFlow Networks

Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali

Department of Computer Science
University of Toronto, Toronto, ON, Canada
{amin,monia,yganjali}@cs.toronto.edu

Abstract. In this paper we present OpenTM, a traffic matrix estimation system for OpenFlow networks. OpenTM uses built-in features provided in OpenFlow switches to directly and accurately measure the traffic matrix with a low overhead. Additionally, OpenTM uses the routing information learned from the OpenFlow controller to intelligently choose the switches from which to obtain flow statistics, thus reducing the load on switching elements. We explore several algorithms for choosing which switches to query, and demonstrate that there is a trade-off between accuracy of measurements, and the worst case maximum load on individual switches, *i.e.*, the perfect load balancing scheme sometimes results in the worst estimate, and the best estimation can lead to worst case load distribution among switches. We show that a non-uniform distribution querying strategy that tends to query switches closer to the destination with a higher probability has a better performance compared to the uniform schemes. Our test-bed experiments show that for a stationary traffic matrix OpenTM normally converges within ten queries which is considerably faster than existing traffic matrix estimation techniques for traditional IP networks.

1 Introduction

A traffic matrix (TM) represents the volume of traffic between origin-destination (OD) pairs in a network. Estimating the point-to-point TM in a network is an essential part of many network design and operation tasks such as capacity planning, routing protocol configuration, network provisioning, load balancing, and anomaly detection.

Direct and precise measurement of TM in large IP networks is extremely difficult, if not infeasible, due to the large number of OD pairs, the high volume of traffic at each link, and the lack of measurement infrastructure [1]. Previous works infer the TM (a) indirectly from link loads [2,3], (b) directly from sampled flow statistics (*e.g.*, using Cisco NetFlow) [4,5], or (c) using a combination of both [1]. Indirect methods are sensitive to the statistical assumptions made in their models and are shown to have large errors [6]. Direct methods can be quite attractive due to their high accuracy levels. However, the lack of required measurement infrastructure and the prohibitively large overhead imposed on the network components are two main drawbacks of direct measurements.

In this paper, we revisit the TM estimation problem using direct measurements in the context of OpenFlow-based networks [7]. OpenFlow is an open standard that makes any changes to the network control plane very easy by separating the data and control planes. An OpenFlow network consists of OpenFlow switches (data plane) managed by a logically centralized OpenFlow controller (control plane) which has a network-wide view. OpenFlow's unique features remove the prohibitive cost of direct measurements for TM estimation. Unlike commodity switches, flow level operations are streamlined into OpenFlow switches which lets us query for flow statistics, enabling access to accurate flow statistics.

Taking advantage of these features, we have designed OpenTM, a TM estimator for OpenFlow networks. OpenTM reads byte and packet counters kept by OpenFlow switches for active flows and therefore incurs a minimal overhead on network elements. At the same time the highest level of accuracy is preserved, because the TM is derived directly without making any simplifying mathematical and statistical assumptions. Our work shows the possibility of direct measurement of TM with the least overhead as long as the infrastructure (OpenFlow here) provides the appropriate feature set for measurements. We note that the scope of our work is limited to the networks where OpenFlow can be deployed, *i.e.*, where maintaining per-flow counters is likely tractable.

For different flows, OpenTM can query any switch along the flow path. This choice, however, can affect the accuracy of the measurement as well as the load on individual switches. We present several strategies for choosing which switch to query at any point of time. Even though all these schemes result in TM estimations with very small errors, our analysis and experiments show that there is a trade-off between the accuracy of the TM measurements and the maximum query load on individual switches, *i.e.*, the perfect query distribution sometimes results in the worst estimate, and the best estimation can lead to the worst query distribution amongst switches.

We have implemented OpenTM as an application for NOX [8], an open-source OpenFlow controller. We study OpenTM's performance on a small testbed of OpenFlow switches. Even though using a small testbed for evaluation has its own shortcomings, we believe that most results would not be significantly different in larger networks. Our results show that in a system with a stationary traffic matrix, OpenTM normally converges within 10 queries to a value within 3% of the average rate which is notably faster than existing techniques for traditional IP networks.

The contributions of this work are two-fold. First, we present the design and implementation of OpenTM for OpenFlow-based networks. Based on the evaluation, we argue that low-overhead accurate TM estimation is feasible using direct measurements in a setting where the switches keep track of flow statistics. Second, we explore the idea of constructing the TMs from switch-level measurements, where the choice of which switch to query can be decided at runtime. To the best of our knowledge, this is in contrast to the existing techniques that usually instrument all ingress/egress links leading to an very uneven measurement load on the boundary switches or routers (switches internal to the network have a very little measurement load).

2 Design

Direct measurements in large traditional IP networks is prohibitively costly due to the processing required to handle the large volume of traffic at each interface [1]. On the other hand, OpenFlow switches keep track of active flows in the network and update per flow counters. The measurement infrastructure that OpenFlow provides enables direct and precise flow measurements without packet sampling or incurring any prohibitive overhead on switches. We take advantage of these features to present OpenTM's design in this section.

OpenTM's logic is quite simple. It keeps track of all the active flows in the network, gets the routing information from the OpenFlow controller's routing application, discovers flow paths, and periodically polls flow byte and packet-count counters from switches on the flow path. Using the routing information, OpenTM constructs the TM by adding up statistics for flows originated from the same source and destined to the same destination[1]. Using the information available to an OpenFlow controller, OpenTM can create different types of TMs with different aggregation levels for sources and destinations. Our implementation of OpenTM computes the TM for switches, but the implementation can be easily augmented to derive other TM types described in [9].

The total number of queries generated by OpenTM during each querying interval is bounded by the number of active flows in the network. It is commonly believed that the number of concurrently active flows in large enterprise IP networks is small. According to the data from the 8000-host network at LBNL, the total number of active flows in their network never exceeds 1200 in any second [10]. The data from the Stanford Computer Science and Electrical Engineering network with 5500 active hosts shows that their number of active flows stays well below 10000 [11]. Currently, our system generates a single query for a single source-destination IP pair. As an improvement, a single query can be generated for all flows sharing the same path, as long as the IP addresses could be aggregated.

Different switches on the path may observe different rates for a given flow due to packet loss. We consider the last switch on the flow path to be the point of reference since this is what is seen by the receiver. Consequently, we query the last switch on the path for the most accurate TM. However, this strategy imposes an uneven and substantially high amounts of load on the first/last switches and does not scale well. We expect to get close statistics if other switches on the flow path are queried since packet loss is negligible in enterprise networks (where OpenFlow is designed for). Based on this observation, we propose different switch querying strategies: *(a)* querying the last switch, *(b)* querying switches on the

[1] Multipath routing, routing changes or hot potato routing do not affect the correctness of OpenTM, because OpenTM coordinates with the controller routing application to discover any change in flow paths.

flow path uniformly at random, *(c)* round-robin querying, *(d)* non-uniform random querying that tends to query switches closer to the destination with a higher probability, and *(e)* querying the least loaded switch.

Querying the last switch results in the most accurate TM, but imposes a substantial load on edge switches. Uniform random querying of switching elements of a given flow's path evenly distributes the load amongst switches as long as all switches are equally capable. The price, however, is losing some accuracy. Round-robin querying deterministically queries switches on a round-robin fashion. On average, we expect both uniform random querying and round-robin querying to behave similarly, but round-robin querying may result in synchronization in querying, because the same switch might be queried by several flows simultaneously. Using a non-uniform distribution for querying switches gives us control over the accuracy and the load of OpenTM. A distribution which chooses last switches in the path with a higher probability, results in a more accurate TM but imposes more load on those switches. In our experiments, for non-uniform querying, we randomly select two switch along the flow path and query the one closer to the destination. Querying the least loaded switch evenly distributes queries among all switches in the network, contrary to the uniform random querying method which only distributes queries among switches on individual flow paths. In Section 5, we compare these methods with each other.

The frequency at which OpenTM queries switches for statistics is another factor that directly affects the accuracy and overhead of TM estimation. Querying more frequently results in a more accurate TM but with the cost of added overhead. Here we only consider fixed length intervals for querying different switches for all the flows. Switch querying interval can be adaptively adjusted for each source-destination IP pair based on the flow and network dynamics (*e.g.*, round trip time, available bandwidth). The relation between an efficient querying frequency and flow and network dynamics is outside the scope of this work.

3 Implementation

We implemented OpenTM as a C++ application for NOX [8], an open-source OpenFlow controller designed to simplify developing network applications. A NOX application can get notified of all network events (*e.g.*, flow initiation and termination), has access to the routing information, and can interact with the switches in the network. NOX also lets applications interact with each other[2].

In each querying interval, OpenTM queries the network for the statistics of all active IP pairs. Element (i, j) in the TM is then computed by summing up the flow rates that are originated from switch i and are destined to switch j. We note that the flow statistic queries do not hit switches at the same time, because flow initiation among OD-pairs are not synchronized.

OpenTM starts querying for statistics periodically once it sees the first flow between an OD-pair and stops querying once all the flows between an OD-pair are expired. To keep track of the number of active IP pairs in the network,

[2] For instance, OpenTM exposes the real-time traffic matrix to other applications.

OpenTM counts the number of TCP/UDP flows between IP pairs. OpenTM increments the mentioned counter upon receiving a `Flow_in` event and decrements it upon receiving the corresponding `Flow_expired` event[3].

Once the IP pair's flow count becomes one, OpenTM fetches the flow path (a list of switches) from the routing application and sends an aggregate statistics query to a switch in the flow path according the desired querying strategy. OpenTM updates the TM when it gets the aggregate query statistics reply back from the network. At this time, if the IP pair flow count is non-zero, OpenTM registers a callback function to query the network for the flow statistics again after a certain period of time[4]. An implicit assumption here is that all packets flowing from the same source to the same destination take the same path. This enables us to query the same set of routers to get statistics for all flows between an IP pair.

OpenTM keeps track of switch loads, so it can choose the least loaded one and optimally balance the queries among all of them. We use the number of outstanding queries on each switch as the load metric. When a switch is queried, a counter that keeps track of the number of outstanding requests on each switch is incremented. Upon receiving the reply back, that counter is decremented. This simple method captures the difference in the processing power of switches. More capable switches can handle more requests and should get more queries compared to the less capable ones. In the following section, we present the results of our empirical evaluation based on our implementation.

4 Experiments and Results

In this section, we present real-time traffic measurements in a testbed to evaluate OpenTM. We study the performance and convergence time of OpenTM. We also compare different switch querying schemes introduced in Section 2.

For our experiments, we use HP DL320 G5p servers equipped with an HP NC326i PCIe dual-port gigabit network card running Debian Lenny and OpenFlow-enabled NEC IP8800/S3640 switches. In all our experiments, we use TCP cubic with the maximum advertised TCP window size set to 20MB. The path MTU is 1500 bytes and the servers send maximum-sized packets. We use the NetEm [12] to emulate network delay and loss, and use Iperf to generate the input traffic.

Our testbed topology is illustrated in Figure 1(a), where H_i, $1 \leq i \leq 10$, are host machines, S_j, $1 \leq j \leq 10$ are OpenFlow switches, and L_k, $1 \leq k \leq 3$ are loss emulator machines. Five OD pairs H_i-H_j are created in which host H_i sends TCP traffic to host H_j. Specifically, we create 10 TCP flows between each OD pair H_1-H_{10}, H_2-H_9, H_3-H_4, H_5-H_6, and H_7-H_8. We add 100ms of delay on the forward path of each flow. The delay emulators are also configured to each

[3] Both `Flow_in` and `Flow_expired` events are fired by the NOX's authenticator application upon flow initiation and termination, respectively. A flow expires when the switch does not see any packets belong to that flow after a specific timeout.

[4] The querying interval which is set to five seconds in our current implementation.

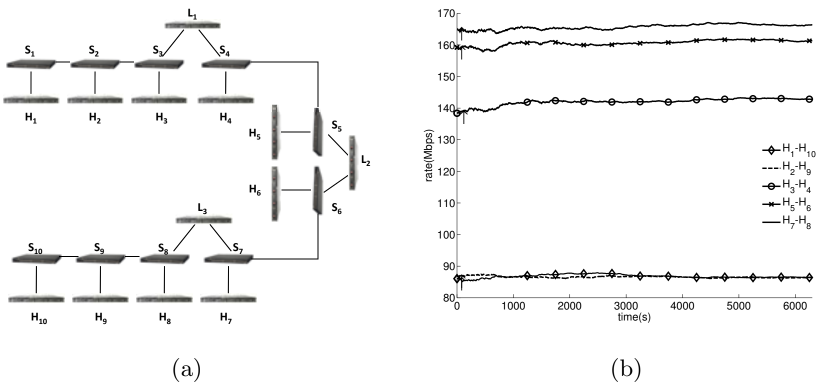


Fig. 1. (a) Testbed topology consisting of H_i: host machines, S_i: OpenFlow switches, and L_i: loss emulators. (b) Measured average flow rate for each of the five OD pairs in the network.

emulate 1% packet loss. For the purpose of evaluation, OpenTM logs the flow statistics of all switches every five seconds for a duration of two hours; we then analyze the data offline. Note that this is not the case in the real application where the switches are queried according to the querying scheme chosen by the network operator. Since we are interested in studying the system in equilibrium, we remove the first 15 minutes of our data as the warm up period.

We start with a very basic question: How fast does OpenTM rate measurements converge? For an element f in the TM, let us assume we have t queries. The i-th query, r_i, is the average rate from the beginning of the measurement up to that point in time. We define the *convergence point* c as the first query for which all the proceeding queries are within 3% of the overall mean rate. We note that since the rate is assumed to be stationary in this system[5], such an average exists. Also, by a simple application of the Central Limit Theorem, we can show that the queries will converge to the real average as we increase the number of queries. Roughly speaking, the convergence point is when the estimated rate becomes and remains very close to the overall average rate that we are trying to estimate.

Figure 1(b) shows the average rate over time for each of the five OD pairs. The measurement is performed at the last switch in each path with a querying interval of only five seconds. We can see that in all cases convergence point, marked by vertical arrows in the graph, is within 50 seconds, or just 10 queries. Note that OD pairs H_1-H_{10} and H_2-H_9 receive the least rate as they are traversing the longest path in the topology through three loss emulators hence experiencing

[5] This assumption does not break the generality of our results. We make the stationarity assumption in order to have a well-defined notion of convergence time. However, as long as the changes in system are slower than our convergence rate, OpenTM gives a close estimate of the TM.

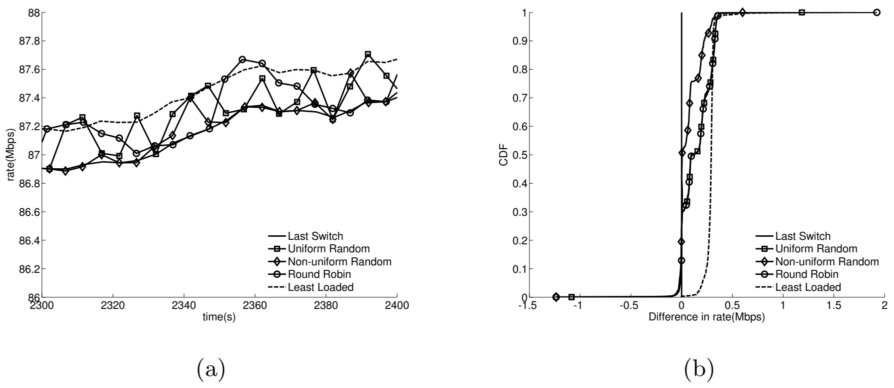


Fig. 2. (a) Comparing the querying strategies methods for traffic between H_1 and H_{10} with a querying interval of 5 seconds. (b) CDF of difference between querying strategies and the last switch's traffic.

3% drop rate. On the other hand, H_3-H_4, H_5-H_6, and H_7-H_8 only experience 1% drop rate and thus have higher rates.

In the next set of experiments, we compare different querying strategies proposed in Section 2. Figure 2(a) shows the measured average throughput versus time for traffic between H_1 and H_{10} when each of the querying strategies are used and the querying interval is five seconds. To make it more visible, the plot is zoomed in and only shows 100 seconds of the measurement. From the figure, it appears that the least loaded method is almost always reporting a rate higher than the last switch method and having the largest estimation error. This is because in our setup, the least loaded switches are mostly the first switch in a flow's path which is before all three drop emulators and hence this method suffers from the most inaccuracy. This is not necessarily the case in other network topologies and it is dependent on the traffic load over the switches.

To better illustrate the difference between querying strategies, Figure 2(b) shows the CDF of differences between each querying strategy and the last switch's rate for traffic between H_1 and H_{10} when each of the querying strategies are used and the querying interval is five seconds. The ideal case is to always measure the last switch's rate and hence having zero difference and it is shown by a vertical line at zero in the graph. As expected from the analysis presented in Section 5, the figure shows that the non-uniform random querying method has the best performance, as it tends to query switches closer to the destination with higher probability. Both the round-robin and uniform random querying methods are performing very close to each other, and worse than the non-uniform querying method. As mentioned above, the least loaded method is performing the worse in this case, since in our setup the first switch on the path is almost always the least loaded switch.

Finally, we note that the overall difference between all these schemes is relatively small. In fact, the maximum difference between the best and worst querying schemes is about 2 Mbps, which is about 2.3% of the actual rate (86 Mbps) between H_1 and H_{10}. This observation suggests that when we do not need extremely accurate TM, any of these querying schemes can be used. Clearly, in this case the least loaded scheme might be the preferred scheme as it minimizes the maximum load among switches. For higher-accuracy requirements, however, one might want to use schemes which favor the last few switches on the path.

5 Analysis

In this section we analytically compare the querying strategies proposed in Section 2 in terms of their accuracy in estimating the flow rates between source and destination. Intuitively, as long as there are no packet drops in the network, all measurements from switches should be the same[6], and thus all querying strategies should be very close to each other; our experiments confirm this. However, when there are packet drops in the system, we expect to see differences in the various querying schemes proposed before.

Let us consider a topology similar to Figure 1(a). We are interested in finding the expected value of rate of a given flow f. We denote the link between switches S_i and S_{i+1} by e_i and the measured rate corresponding to f over e_i by r_i. If e_i has a drop rate d, then the rate measured at e_{i+1} will be $\leq r_i \times (1-d)$. Assuming there are M uniform randomly distributed congestion points in the network, each with a drop rate of d, we can find the expected rate for each querying strategy as follows. Note that here for simplicity we assume that all links have equal drop probability of d.

Querying the last switch before the destination. We define the rate between an OD pair as the rate seen by the destination. Assuming negligible packet drops on the link connecting the last switch to the destination node, querying the last switch must give us the rate as seen by the destination regardless of network conditions. We use this rate as the baseline for comparing with randomized querying techniques presented below.

Uniform random querying. We first consider the simple case where there is only one congested link in the network and call the measured rate by this method at a time slot i as $R_r(i)$ and the rate at the last switch by $R_t(i)$. There are two possible cases: (1) if the randomly selected switch is between the congested link and the last switch before the destination, then rate scene by the selected switch is same as the rate at the last switch; $R_r(i) = R_t(i)$ (2) if the selected switch is between the source and the congested link, then rate at the selected switch is higher than the rate at the last hop switch before the destination. In particular, $R_r(i) = \frac{R_t(i)}{1-d}$. Hence, $R_t(i) \leq R_r(i) \leq \frac{R_t(i)}{1-d}$. Assuming that the congested link is placed uniformly random over the path then each of the above cases has an

[6] We ignore the difference caused by the delay between switches.

equal probability of one half. If we take the average rate over N queries, the expected rate $R_r = \sum_{i=1}^{N} R_r(i)/N$, will lie exactly in between the two cases; *i.e.*, $R_r = 0.5 \times (R_t + R_t/(1-d))$.

Similarly, if there are M congestion points in the network then we have $R_t(i) \leq R_r(i) \leq R_t(i)/(1-d)^M$ and if we assume that the congestion points are distributed uniformly over the path, then the probability of $R_r(i) = \frac{R_t(i)}{(1-d)^m}$ is $\frac{1}{M+1}$, where $0 \leq m \leq M$ is the number of congestion points that the flow has traversed before reaching the querying switch. Hence,

$$R_{ur} = \frac{R_t}{M+1} \sum_{m=0}^{M} \frac{1}{(1-d)^m} = \frac{R_t}{M+1} \times \frac{1-(1-d)^{M+1}}{d(1-d)^M} \tag{1}$$

Non-uniform random querying. In this method, we generate two random numbers i and j, $1 \leq i, j \leq N$, where N is the number of switches in a flow's path and query the switch with ID equal to $\max(i, j)$, assuming the switch with larger ID is the one closer to the destination. With same assumptions as the above and in the case that there are M congestion points in the network we have $(M+1)^2$ cases and if we take the average over N queries for large N, the expected average rate will be:

$$R_{nr} = \frac{R_t}{(M+1)^2} \left(1 + 2 \sum_{m=0}^{M-1} \frac{M-m}{(1-d)^m} + \frac{1}{(1-d)^M}\right) \tag{2}$$

Round-Robin querying. The expected value of average rate for the round-robin querying method, R_{rr}, is similar to the uniform random method since on average $\frac{1}{M+1}$ of queries will have rate $R_t(i)$, $\frac{1}{M+1}$ of queries will have rate $\frac{R_t(i)}{1-d}$ and so on.

Least-loaded switch querying. The performance of least-loaded switch querying highly depends on packet processing power of network switches, as well as how network load is distributed amongst them. If switches have equal processing power and load this scheme will perform very similar to uniform random querying. However, in the worst case, the least loaded switch might be the first switch on the path in which case it will lead to the worst case estimation of the rate.

6 Conclusion

This paper presents OpenTM, a traffic matrix estimator for OpenFlow networks. OpenTM derives the TM of an OpenFlow network in real-time with high accuracy using direct measurements without packet sampling. OpenTM evenly distributes the statistic queries among all the switches in the network and thus imposes the least overhead on the network. Our evaluation in a testbed using OpenTM implemented as a NOX application shows that OpenTM derives an accurate TM within 10 switch querying intervals, which is extremely faster than existing TM estimation techniques. Despite the limitations of our evaluation and the need for more comprehensive evaluation, we believe OpenTM can be deployed in OpenFlow networks with a very negligible overhead.

Acknowledgments

This work was partly supported by Cisco Systems and a grant from NSERC. NEC Corporation kindly provided us with the OpenFlow switches. We would also like to thank Bianca Schroeder and the anonymous reviewers for the their valuable feedback.

References

1. Zhao, Q., Ge, Z., Wang, J., Xu, J.: Robust traffic matrix estimation with imperfect information: Making use of multiple data sources. SIGMETRICS Performance Evaluation Review 34(1), 133–144 (2006)
2. Vardi, Y.: Network tomography: Estimating source-destination traffic intensities from link data. Journal of the American Statistical Association 91(433), 365–377 (1996)
3. Nucci, A., Diot, C.: Design of IGP link weight changes for estimation of traffic matrices. In: Proceedings of the 2004 Conference on Computer Communications (2004)
4. Feldmann, A., Greenberg, A., Lund, C., Reingold, N., Rexford, J., True, F.: Deriving traffic demands for operational IP networks: Methodology and experience. IEEE/ACM Transactions on Networking 9(3), 265–280 (2001)
5. Papagiannaki, K., Taft, N., Lakhina, A.: A distributed approach to measure IP traffic matrices. In: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (2004)
6. Medina, A., Taft, N., Salamatian, K., Bhattacharyya, S., Diot, C.: Traffic matrix estimation: Existing techniques and new directions. SIGCOMM Computer Communication Review 32(4), 161–174 (2002)
7. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: OpenFlow: Enabling innovation in campus networks. SIGCOMM Computer Communication Review 38(2), 69–74 (2008)
8. Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S.: NOX: Towards an operating system for networks. SIGCOMM Computer Communication Review 38(3), 105–110 (2008)
9. Medina, A., Fraleigh, C., Taft, N., Bhattacharrya, S., Diot, C.: A taxonomy of IP traffic matrices. In: SPIE ITCOM: Scalability and Traffic Control in IP Networks II, Boston (August 2002)
10. Pang, R., Allman, M., Bennett, M., Lee, J., Paxson, V., Tierney, B.: A first look at modern enterprise traffic. In: Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement, Berkeley, CA, USA, p. 2 (2005)
11. Naous, J., Erickson, D., Covington, G.A., Appenzeller, G., McKeown, N.: Implementing an OpenFlow switch on the NetFPGA platform. In: Franklin, M.A., Panda, D.K., Stiliadis, D. (eds.) ANCS, pp. 1–9. ACM, New York (2008)
12. Hemminger, S.: Network emulation with NetEm. In: Linux Conference, Australia (April 2005)

Web Timeouts and Their Implications*

Zakaria Al-Qudah[1], Michael Rabinovich[1], and Mark Allman[2]

[1] Case Western Reserve University, Cleveland, Ohio 44106
[2] International Computer Science Institute, Berkeley, CA 94704

Abstract. Timeouts play a fundamental role in network protocols, controlling numerous aspects of host behavior at different layers of the protocol stack. Previous work has documented a class of Denial of Service (DoS) attacks that leverage timeouts to force a host to preserve state with a bare minimum level of interactivity with the attacker. This paper considers the vulnerability of operational Web servers to such attacks by comparing timeouts implemented in servers with the normal Web activity that informs our understanding as to the necessary length of timeouts. We then use these two results—which generally show that the timeouts in wide use are long relative to normal Web transactions—to devise a framework to augment static timeouts with both measurements of the system and particular policy decisions in times of high load.

1 Introduction

One of the historic tenets of networking that has served the Internet well over the past 30 years is that components of the system should be both conservative and liberal at the same time. That is, actions should only be taken as they are strictly necessary—therefore acting *conservatively*. Furthermore, wide tolerance for a range of behavior from other components in the system is also desirable—or acting *liberally*. Another fundamental notion within the Internet is that the only thing we can absolutely count on is the passage of time. This notion naturally led to timeouts as a fundamental fallback mechanism to ensure robust operation. Adhering to the above stated principles tends to make timeouts long such that we can tolerate a range of behavior and the timer only expires when a gross anomaly occurs as opposed to when some task simply happens slower than expected.

Unfortunately, the above narrative becomes muddled in the presence of malicious actors as it creates an opening for the so-called *claim-and-hold* denial of service attacks [13], where an attacker can claim server resources without using them thus preventing the server from utilizing these resources on legitimate activities.

In this paper, we consider the issue of timeouts in the modern Internet within the context of the Web. We conduct an empirical investigation that seeks to understand (*i*) how timeouts are currently set on Web servers and (*ii*) how those settings relate to normal user-driven Web traffic. Our key finding is that

* This work is supported in part by NSF grants CNS-0615190 and CNS-0433702.

timeout settings are extremely conservative relative to actual traffic patterns and expose Web servers to easy DoS attacks. While this suggests that servers could take a more aggressive posture with respect to timeouts, doing so would run counter to the general tenet mentioned earlier (i.e., would result into dropping legitimate anomalies even at times when enough resources are available to serve them). Instead, we propose an adaptive approach whereby the timeouts are only reduced at times of measured stress. In fact, we observed a small number of Web sites that exhibit a behavior which indicates that they might be already varying their timeouts dynamically. We believe other sites, large or small, would benefit from similar reactions in the face of claim-and-hold attacks. Unfortunately, such timeout adaption is not available out-of-the-box in popular Web servers. As part of this project we have implemented and make available a simplified adaptive mechanism as a modification of the Linux kernel and Apache Web server [1].

2 Related Work

Qie, et.al. [13] studied, verified, and classified DoS attacks into *busy attacks* and *claim-and-hold attacks*. Web server administrators have reported encountering claim-and-hold attacks [7,6] and server software vendors seem cognizant of these attacks and typically recommend tuning Web server timeouts [4,8]. However, as we show in this paper, a large number of Web sites use default timeout values. Barford et. al. observed the negative effect of excessive persistent connections on busy Web servers and recommended an *early close* policy whereby Web clients close persistent connections after downloading a page and all its embedded objects [5]. Rabinovich et. al. suggested adaptive management of persistent connections at Web servers, where a server closes idle connections once it runs out of the connection slots [14]. We argue for a similar but more general approach in Section 4. Park, et.al. also point out the danger of inactive or slow Web clients and propose an independent component to filter and condition external connections for the Web server [12]. In contrast, we suggest an adaptive timeout strategy on the Web server itself.

3 Timeout Measurements

In this section, we assess timeout periods in operational Web servers and compare them with the time needed by Web clients to perform the corresponding activities that these timeout periods control. To this end, we probe two groups of Web servers: (*i*) Alexa's top 500 sites [3] denoted as "high volume" sites and (*ii*) 15,445 sites collected using the Link Harvester tool [15] denoted as "regular" sites. The list of these sites is available from [1]. In the high volume group, 53% of sites reported some version of Apache Web server in the "Server:" response header, 12% Microsoft-IIS, 10% GWS (Google), and the rest reported some other server or nothing at all. Among the regular sites, 68% were Apache, 19% Microsoft-IIS, and the rest other/unknown. As described below, we actively probe these sites for various timeouts. Inevitably for each experiment a small

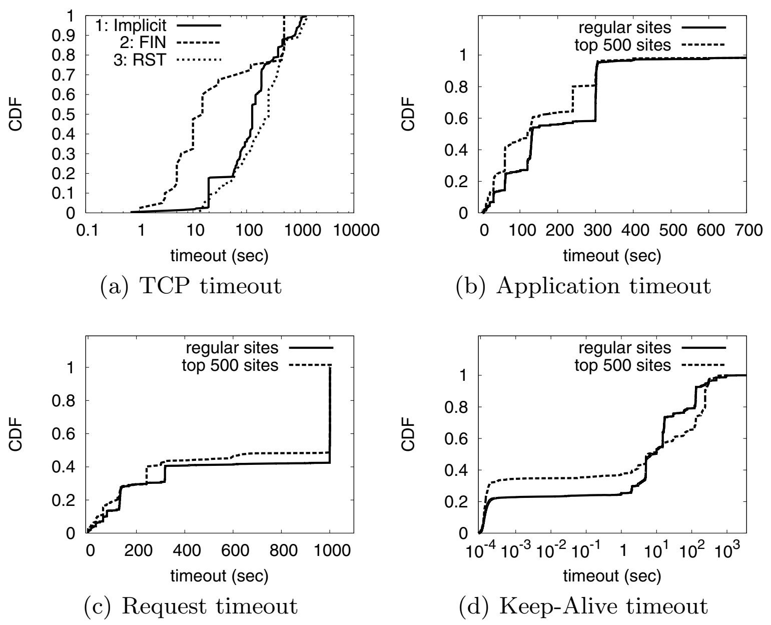


Fig. 1. Distribution of Web server timeouts

number of sites are unavailable and so the precise number of sites used varies and is reported for each experiment.

To assess the time consumed by Web clients to perform various normal activities, we analyze a week long packet trace of Web traffic collected at the border of International Computer Science Institute (ICSI) captured between August 11–18 2009. The trace contains nearly 1.6M HTTP connections involving nearly 14K servers and 25K clients. We note that Web clients in our trace are generally well-connected. While it would be desirable to verify our results directly in a qualitatively different environment, we do not expect dial-up clients to affect our findings (as discussed later).

TCP Timeout: The *TCP timeout* represents the length of time a TCP implementation will attempt to retransmit data before giving up on an unresponsive host. We assess this timeout by opening a TCP connection to a given server, sending an HTTP request and disappearing—i.e., sending no further data, ACK, FIN or RST packets. Some sites respond with an HTTP redirection and a FIN (either with the data or closely thereafter). We exclude these sites from further analysis because the timeout we experience in this case is the FIN_WAIT state timeout, not the retransmission timeout. This reduces the number of sites—437 high volume and 13,142 regular sites—involved in this experiment compared to other experiments.

We monitor the server's retransmissions and find three distinct ways for connections to end: (*i*) implicitly with the retransmissions eventually ceasing, (*ii*) explicitly with the server sending a FIN or (*iii*) explicitly with a server sending a RST. We measure the TCP timeout as the interval between the arrival of the first data transmission and the arrival of either the last retransmission or a packet with a FIN or RST bit set (note, this FIN case is distinct from the redirection case discussed above). Figure 1(a) shows the distribution of timeouts measured for each termination method for the high volume sites (the regular sites are omitted due to space constraints but show the same general behavior). In case (*i*)—encompassing 61% of the servers in both sets—the observed timeout is over 100 seconds for two-thirds of the servers. Note that in this case there is no wire event indicating the server has dropped a connection, and we expect that the server waits for some time for an ACK after the last retransmission. Therefore, the measurements represent a lower bound. In case (*ii*)—encompassing 9% of the servers in each set—we believe the FIN transmission is generally triggered by the overall application giving up on the connection rather than TCP terminating the connection itself. Therefore, at best these measurements also represent a lower bound on the length of the TCP timeout, which is illustrated by the order of magnitude difference between cases (*i*) and (*ii*) in Figure 1(a). In case (*iii*)—encompassing 30% of the servers in each set—we observe that servers that send a RST show the longest timeouts by a small margin over servers that silently terminate. We believe this is likely the best representation of the TCP timeout as it encompasses both the entire retransmission process and the additional waiting time that goes unseen in case (*i*).

We contrast the above determined lower bounds with data from our previous work [2]. In that work, we set up two servers: one configured with a normal TCP timeout (default Linux timeout of 15 retransmissions, or ≈13–30 minutes) and one with a quick TCP timeout (3 retransmissions or roughly 600 msec). We then used 59 Keynote [9] clients around the world to download a 2 MB file from each server every 15 minutes for over a week. Reduced retransmissions increased dropped connections due to timeouts by 0.16%, suggesting that continuing to retransmit for long periods of time is often futile.

In summary, while Figure 1(a) shows that—excluding cases where we do not believe TCP terminated the connection—80% of the surveyed servers have TCP timeouts exceeding 57 seconds, and nearly two-thirds of the servers have TCP timeouts exceeding 100 seconds, our preliminary data indicates that most Web interactions would succeed with a sub-second TCP timeout.

Application Timeout: The *application timeout* is the time a server allows between completing the TCP connection establishment and the arrival of the first byte of an HTTP request. To measure the application timeout in operational Web sites, we open a TCP connection to a server without sending an HTTP request using *nc6* [11]. We then measure the time from the completion of the TCP connection establishment until the connection is closed by the server (giving up after 20min). We use 492 high volume sites and 14,985 regular sites in this experiment. We find that just under 36% of sites in both groups do not end the connection after 20min. Potential reasons for this behavior include sites using the TCP_DEFER_ACCEPT Linux TCP option [16] (or like option on other systems). With this option, TCP does not promote a connection from the SYN_RCVD state to ESTABLISHED state—and thus hand it over to the application—until data arrives on the connection. Therefore, the notion of application timeout is not applicable for these sites. (Note however that these sites can still accumulate pending connections in the SYN_RCVD state, which may present a different attack vector.) Another explanation is these sites have an application timeout which is longer than 20min.

Figure 1(b) shows the distribution of measured application timeouts for the remaining ≈64% of sites in the two groups. The figure shows significant modes in both groups around 120s and 300s—the well-known defaults for IIS and Apache respectively. We also observe that high volume sites generally have shorter application timeouts than regular sites. Presumably these sites have determined that shorter timeouts are better for resource management without disrupting users. The figure also has a mode around 240s for the high volume sites which is mostly due to Google's sites (e.g., google.com, google.fr, google.co.uk, gmail.com, etc.). Similarly, we find that the high volume sites responsible for the mode around 30s to be mostly Akamai-accelerated sites. Finally, we find a mode around 60s which we cannot readily explain. Overall, around 54% of high-volume sites and 74% of regular sites have application timeouts of over 100s.

We now turn to our packet trace and measure the time between the last ACK in TCP's three-way handshake and the first packet with the client's HTTP request. We find that 99% of the requests were sent within one second of completing the TCP connection establishment. However, the longest time a client in our trace took to start sending the request after completing the TCP connection establishment is 586 seconds.

Request Timeout: The *request timeout* is the time a Web server allots to a request to completely arrive at the server after the first byte of the request has arrived. To measure the request timeout we drip a 1000 byte request over the network at a rate of one byte/sec and note when (or if) the server terminates the connection. Transmitting the request at a byte/sec factors out a possible effect of another timeout commonly applied to *poll()/select()* calls—which is usually greater than one second. This experiment involves 492 high-volume and 15,033 regular sites.

Figure 1(c) shows the distribution of the measured request timeouts. The plot indicates that 58% of the regular sites and 51% of the high volume sites keep the connection open for the entire 1,000 seconds it took our client to send its request, suggesting that the server does not impose a request timeout. Among the sites that do set a smaller request timeout, high volume sites have generally shorter timeouts than regular sites. Overall, 93% of the high volume sites and 96% of the regular sites have a request timeout period of over 30 seconds.

To assess how long Web clients normally take to transmit their requests, we measure the time between the first and last packets of HTTP requests in our trace. When the entire request fits in one packet, we report the time as zero.

Group	Impose Limit	IIS with limit	IIS without limit
Top 500	24.1%	32.7%	5.1%
Regular	23.5%	59.0%	7.2%

Fig. 2. Response timeout

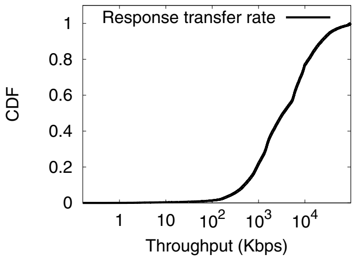


Fig. 3. Response rate

We concentrate on HTTP GET requests in this experiment as they are typically small. HTTP POST requests on the other hand could be arbitrarily large and take longer to send. This suggests that these two request types should be handled with different timeouts. We find that 85% of the requests fit into one packet. Further, 99.9% of the requests are completed within 1 second. Still, the longest time taken by a client in the trace is 592 seconds.

Response Timeout: The *response timeout* is the amount of time the server allocates to delivering an HTTP response. This timeout guards against a client that is alive (i.e., responds with TCP ACKs) but consumes data at a slow rate, by either acknowledging few bytes at a time or advertising a small (or at the extreme, zero) window. Since the client is responding the connection can only be closed by the application and not by TCP.

We are aware of only one major Web server that enforces a response timeout—alternatively presented as a minimum transfer rate—which is Microsoft's IIS. The default minimum transfer rate in IIS is 240 bytes/sec. Even though IIS notationally imposes a minimum *rate-based* limit, internally this is converted to a *time-based* limit. Specifically, IIS divides the response size by the minimum transfer rate with the result used to arm a timer. If the timer fires and the client has not fully consumed the response, IIS will close the connection [8]. This mechanism is efficient in that progress is checked only once. However, an attacker can leverage this mechanism by finding a large object and retrieving it at a low rate—which IIS will only detect after a long time.

To measure the response timeout, we open a connection to a Web server, send a request for the home page, and consume the response at a low rate. Given the IIS's default rate limit of 240 bytes/sec, in our experiments we consume the response at a lower rate of 100 bytes/sec. A site that delivers the entire response at this rate is assumed to not impose a limit, otherwise a limit is in place. This experiment involves 494 high volume sites and 15,034 regular sites. The table in Figure 2 shows our results. We find that less than 25% of sites—regardless of group—impose a limit on the transfer rate. Furthermore, 59% of the regular sites that impose limits identify themselves as IIS, as expected. However, only 33% of the high-volume sites that impose response time limits identify themselves as IIS servers. There could be a myriad reasons that can explain the remaining sites, including IIS servers obscuring their identities, servers behind transparent TCP proxies that keep their own timers, custom built servers, intrusion prevention systems impacting communication, etc. Interestingly, as shown in the last column of the table, there is a small percentage of sites that identify themselves as IIS servers and yet do not impose any response timeout. This could be caused by site administrators disabling the response timeout or transparent TCP proxies that obscure the actual Web server behavior.

We now consider the time needed by normal Web clients to consume responses. This time is determined mainly by the round trip time for small responses and by the available end-to-end bandwidth for large responses. Therefore, while a low limit on the transfer rate such as IIS's 240 byte/sec might be appropriate for small responses (although whether one could tighten this limit at times of stress is an interesting question for future work), we aim at assessing whether such a low limit is appropriate for large responses, especially that attacks against this timeout are particularly dangerous for large responses. To assess that, we consider responses in the ICSI trace with size of at least 50 KB. We approximate the end-to-end transfer rate as the response size divided by the time between the first and last packet of the response. Figure 3 presents the distribution of response transfer rates. The figure shows that nearly 99% of the responses (whether the response was originated from an ISCI server or an external server) were transferred at over 10 Kbps (that is 1,250 bytes/second compared to the default of 240 bytes/second of IIS).

HTTP Keep-Alive: We next turn to persistent HTTP, which attempts to make Web transfers efficient by keeping TCP connections open for more than one HTTP request. The *HTTP keep-alive timeout* is defined as the time the server will keep an idle connection open after successfully satisfying all requests. We start by issuing requests for the home pages of the Web sites using *nc6*. We then measure the time between receiving the last packet of the response and receiving a FIN or RST from the server. This experiment involves 490 high volume and 14,928 regular sites Figure 1(d) shows the distribution of these times. The problem of finding a cut-off point before which we assume servers do not maintain persistent connections is relatively easy in this figure. Indeed, selecting the cut-off point at 100ms or at 1 second produces similar results. Roughly, 65% of the high volume sites and 76% of the regular sites maintain persistent connections. These numbers indicate that the overall support of persistent connections has not changed appreciably since Fall of 2000 [10]. Surprisingly, regular sites seem to have shorter keep-alive timeouts than high volume sites. For instance, nearly 61% of the high volume sites that use persistent connections use a timeout over 30s while it is roughly 32% for the regular sites. We speculate that this is due to the higher incidence of Apache with default configuration of 15s keep-alive timeout among regular sites than it is among high volume sites.

Timeout Adaption: To get a preliminary intuition as to whether Web sites currently vary their timeouts over time, we performed periodic probing of the

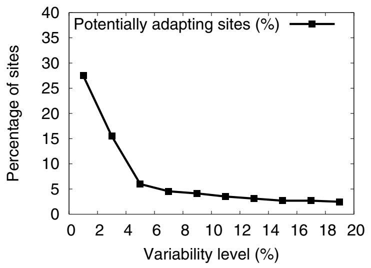


Fig. 4. Variability of request timeouts

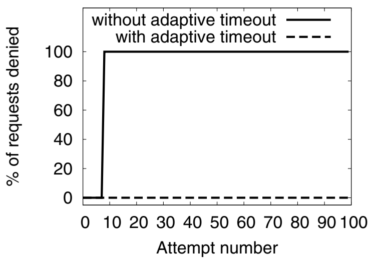


Fig. 5. Performance of adaptive timeouts

request timeout for the high volume sites. Specifically, we probed each site every 12 minutes for a week. We define a site as having an adaptive timeout if at least $m\%$ of the measurements to the server are at least $m\%$ different from the mean timeout to the given site (i.e., m is an experimental parameter). This procedure is clearly not conclusive given that we may simply not have observed adaption for a particular site because there was no reason for the site to adapt during our measurements. Further, a timeout change could be caused by reasons other than adaptability such as different requests arriving at different servers with various timeout configurations or a server crash during a connection lifetime. The percentage of sites found to be using an adaptive timeout as a function of m is shown in Figure 4. We find that roughly 3% of the sites tested exhibit behavior suggestive of timer adaption, as shown by the range of m values for which this finding holds.

Summary: our measurements indicate that normal web clients perform their activities quickly as compared to the time allowed by Web servers.[1] Long timeouts leave a server vulnerable to claim-and-hold attacks. These attacks have been reported in practice [7,6], and we will demonstrate a simple attack utilizing these timeouts in the next section. Short of complex external intrusion detection mechanisms, a naive way to counter these attacks would be to increase the number of allowable concurrent connection slots at the server. But this may cause performance degradation in case the slots are consumed by legitimate connections, since the number of concurrent connections is driven by the server capacity. Furthermore, although our measurements show that current long timeouts are generally unneeded by normal Web clients, slashing them blindly would run counter to the general networking tenet of allowing liberal client behaviors. Therefore, we suggest slashing these timeouts only at the time of

[1] While clients in our trace are generally well-connected, the characteristics of dial-up connections should not affect this finding. Indeed, dial-up connections offer a last-mile bandwidth of 30-40 Kbps—well within the 99th percentile we observe in our trace and also well above the 240 bytes/sec IIS requires. Furthermore, the few hundreds of milliseconds these connections add still leave the time needed by these connections to perform activities much shorter than allowed by Web servers.

stress.[2] While our measurements suggest that a small fraction of sites might already be varying their timeouts, popular Web servers such as Apache and IIS do not offer such mechanisms—which would limit the spread use of these mechanisms.

4 Adaptive Timeouts

We now present our implementation of an adaptive timeout mechanism and demonstrate its usefulness. Our implementation involves changes to the Linux TCP stack and Apache web server (version 2.2.11). The kernel extension allows an application to specify a target response transfer rate and to toggle the kernel between a conservative (current behavior) and aggressive (close any connection below the target transfer rate) modes. The kernel monitors the transfer rate of connections only during periods of non-empty TCP send queue to avoid penalizing a client for the time the server has no data to send. Our modified Apache sets the target transfer rate parameter (500 bytes/second in our experiments) and monitors the connection slots. Once allocated slots reach a certain level (90% of all slots in our experiments), it (a) reduces its application timeout from its current default of 300s to 3s and (b) toggles the kernel into the aggressive mode. While a complete implementation of our framework would consider all timeouts, our current implementation covers application timeout, TCP timeout, and response timeout.

To demonstrate how such a simple mechanism can protect sites from claim-and-hold attacks, we set up a Web site with Linux OS and Apache Web server, both using out-of-the box configurations except with Apache configured to allow a higher number of concurrent connections (256 vs. default 150). We then set up a machine that launches an attack targeting the response timeout. In particular, it attempts to keep 300 concurrent connections by requesting a 100 KB file and consuming it at a rate of 200–300 bytes/second on each of these connections. Another machine simulates legitimate traffic by probing the server once every 10 seconds by opening 100 connections to the server with a 5 second timeout period (i.e., a request fails if not satisfied within 5 seconds). This process repeats 100 times. The solid line on Figure 5 shows the results. The attack starts around probe number five. After a short delay (due to Apache's gradual forking of new processes) the attacking host is able to hold all the connection slots and thus completely deny the service to legitimate connections. Further, the attacker accomplishes this at the cost of consuming less than 1Mbps (300 connections with at most 300 bytes/s each) of its own bandwidth—available to a single average residential DSL user let alone a botnet. The dashed line in Figure 5 shows the

[2] One can imagine applications, as possible with AJAX, where HTTP connections could have long idle periods. Our trace accounts for all HTTP interactions including AJAX, and as discussed, did not encounter such connections in large numbers. We note that the content provider controls both ends of the connection in these applications. Therefore, these connections could either be treated differently by the server or the applications can be written to handle possible interruptions gracefully.

results of repeating the attack on our modified platform. As seen, our simple mechanism allows the server to cope with the attack load without impinging on legitimate connections by quickly terminating attack connections which leaves open slots for legitimate traffic. Our intent in this experiment is to show that a simple system can perform well. We consider a full study of a range of decision heuristics out of scope for this paper. Further, such decisions can be a policy matter and therefore cannot be entirely evaluated on purely technical grounds.

5 Conclusions

In this paper we study Internet timeouts from two perspectives. We first probe the timeout settings in two sets of operational Web sites (high volume and regular sites). We then study the characteristics of normal Web activity by analyzing passively captured Web traffic. The major finding from these two measurements is that there is a significant mismatch between the time normal Web transactions take and that which Web servers allow for these transactions. While this reflects a conservativeness on the Web server's part it also opens a window of vulnerability to claim-and-hold DoS attacks whereby an attacker claims a large fraction of connection slots from the server and prevents their usage for legitimate clients.

Rather than reducing servers' timeouts to match normal Web activity—a solution that could reduce the tolerance of the server to legitimate activity—we suggest a dynamic mechanism that is based on continuous measurements of both connection progress and resource contention on the server. A decision to reduce the timeouts and drop connections accomplishing little or no useful work is only taken when the server becomes resource constrained. We demonstrate how this simple mechanism can protect Web servers. Our mechanism is implemented in a popular open source Web server and is available for download [1].

References

1. Project Downloads, `http://vorlon.case.edu/~zma/timeout_downloads/`
2. Al-Qudah, Z., Lee, S., Rabinovich, M., Spatscheck, O., der Merwe, J.V.: Anycast-aware transport for content delivery networks. In: 18th International World Wide Web Conference, April 2009, p. 301(2009)
3. Alexa The Web Information Company, `http://www.alexa.com/`
4. Apache HTTP server - Security tips, `http://httpd.apache.org/docs/trunk/misc/security_tips.html`
5. Barford, P., Crovella, M.: A performance evaluation of hyper text transfer protocols. SIGMETRICS, 188–197 (1999)
6. `objectmix.com/apache/672969-re-need-help-fighting-dos-attack-apache.html`
7. `http://www.webhostingtalk.com/showthread.php?t=645132`
8. Microsoft TechNet Library, `http://technet.microsoft.com/en-us/library/cc775498.aspx`
9. Keynote, `http://www.keynote.com/`

10. Krishnamurthy, B., Arlitt, M.: PRO-COW: Protocol compliance on the web: A longitudinal study. In: USENIX Symp. on Internet Technologies and Sys. (2001)
11. nc6 - network swiss army knife, `http://linux.die.net/man/1/nc6`
12. Park, K., Pai, V.S.: Connection conditioning: architecture-independent support for simple, robust servers. In: USENIX NSDI (2006)
13. Qie, X., Pang, R., Peterson, L.: Defensive programming: using an annotation toolkit to build DoS-resistant software. SIGOPS Oper. Syst. Rev. 36(SI) (2002)
14. Rabinovich, M., Wang, H.: DHTTP: An efficient and cache-friendly transfer protocol for web traffic. In: INFOCOM, pp. 1597–1606 (2001)
15. SEOBOOK.com, `http://tools.seobook.com/link-harvester/`
16. TCP protocol - Linux man page, `http://linux.die.net/man/7/tcp`

A Longitudinal View of HTTP Traffic[⋆]

Tom Callahan[1], Mark Allman[2], and Vern Paxson[2,3]

[1] Case Western Reserve University
[2] International Computer Science Institute
[3] University of California, Berkeley

Abstract. In this paper we analyze three and a half years of HTTP traffic observed at a small research institute to characterize the evolution of various facets of web operation. While our dataset is modest in terms of user population, it is unique in its temporal breadth. We leverage the longitudinal data to study various characteristics of the traffic, from client and server behavior to object and connection characteristics. In addition, we assess how the delivery of content is structured across our datasets, including the use of browser caches, the efficacy of network-based proxy caches, and the use of content delivery networks. While each of the aspects we study has been investigated to some extent in prior work, our contribution is a unique long-term characterization.

1 Introduction

In this paper we study logs of web traffic collected at the border of a small research institute over a three and a half year period (2006–mid-2009). There are an average of 160 active users per month in our dataset. While this is a relatively small population, we gain insight into the evolution of web traffic by taking a longitudinal view of the traffic. This investigation serves to re-appraise and update previous results. We believe our contribution has utility in informing the community's mental models about myriad aspects of how the modern web works—including things like transaction types and sizes, as well as how web content delivery is accomplished through content delivery networks, browser caches and the like. In addition, a multi-faceted view of web content delivery is useful in setting up realistic testbeds and simulations to accurately reflect the make-up and structure of today's web.

Our methodology employs web traffic logs from our intrusion detection system collected over three and a half years to study various aspects of the web. We describe our data collection and analysis methodology in § 2. We then characterize a number of facets of the traffic at the transaction-level in § 3. We next consider various aspects of user-driven behavior, such as object popularity and the impact of caching in § 4. Finally, we consider the structure of the web page delivery process, including the use of CDNs in § 5. We briefly touch on related work in § 6 and summarize in § 7.

2 Data and Methodology

For this work we use logs of web traffic taken at the border connecting the International Computer Science Institute (ICSI) with its ISP. We use the Bro intrusion

[⋆] This work is supported in part by NSF grants CNS-0831535 and CNS-0831780.

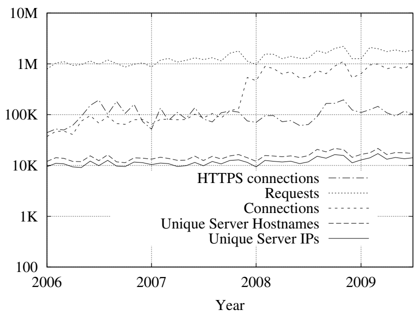


Fig. 1. Dataset Summary

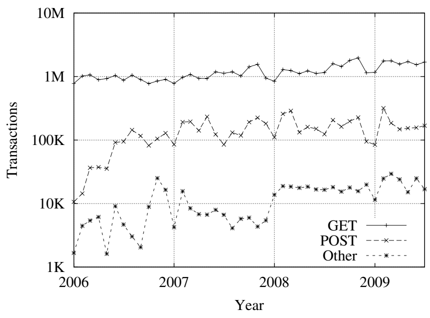


Fig. 2. HTTP Transaction Types

detection system [12] to reconstruct HTTP [7] sessions from the observed packet stream. These sessions are then logged using Bro's standard HTTP logging policy (found in `http.bro` in the Bro package). The logs include timestamps, involved IP addresses, URLs, HTTP transaction types and sizes, hostnames and HTTP response codes. The dataset used in this paper runs from January 2006 through July 2009. Due to the size of the dataset we analyze only the first seven days of each month for logistical reasons. We do not believe this biases our results. The original logs include all incoming and outgoing HTTP traffic. However, we winnowed to only the outgoing connections (i.e., ICSI clients) as we do not wish to bias our results by the particular characteristics of the few server instances at ICSI. Of the 28.8 million total connections from the first seven days of each month in our dataset we retain 16.9 million as initiated by ICSI clients. In figure 1 we show the high-order characteristics of the dataset. The overall number of web object requests, HTTP connections, HTTPS connections, server hostnames and server IP addresses show general stability over time.[1] Since the HTTPS connections are encrypted we cannot further analyze them in this work. We identify "web servers" in two different ways: by IP address and by hostname. Due to the use of content delivery networks (CDNs) a particular IP address may host content for multiple distinct hostnames. In fact, in the figure we see this effect as there are more server hostnames than server IP addresses (this is studied in more detail in § 5). However, note that the opposite is also true: that a given hostname could have multiple IP addresses (e.g., to serve content from a close source or for load balancing). Finally, we note that the number of users is modest—an average of 160 per month with a standard deviation of 13—our contribution is the longitudinal tracking of this user population.

Finally, we note that there are two versions of Bro HTTP policy scripts used in gathering the data we employ in this study with one crucial difference. For the first ten months of 2006 the scripts gathered logical web sessions together as one logical entity under one identifier regardless of the number of underlying TCP connections used to obtain the components of the web pages. In these log files this process obscures the number of TCP connections used to transfer the data. Due to the onerous amount of state required to stitch together a web session from disparate TCP connections, starting

[1] The number of connections has a dramatic increase in December 2007. We delve into this in detail in § 5.

in mid-October 2006 the scripts were changed to simply gather together all activity on a per-TCP connection basis. For most of our analysis the difference in logging is not important, but for analysis that requires an understanding of the number of underlying TCP connections we start our analysis in November 2006 instead of January 2006. In figure 1 the reported number of connections for the first 10 months of the dataset is actually the number of web sessions (which is reported to give the reader context even though the precise number of connections is unknown).

3 HTTP Transaction Characterization

We first focus on characterizing client HTTP transactions. First, figure 2 shows the transaction type breakdown over time. Over the course of our dataset the majority of observed transactions—approaching 90% in most months—are requests for data (`GET` transactions). Most of the remainder of the transactions—around 10%—involve the user uploading data to the web server (`POST` transactions). A small number of additional transaction types are also observed (`HEAD`, `PROPFIND`, etc.). Together these additional types account for less than 1% of the transactions in most months. We note that in absolute terms the number of `GET`s and `POST`s have a slight increasing trend over our observation period (note, the figure is plotted on a log scale and therefore the increase is less readily apparent). Further, the number of `POST`s observed increases quickly at the beginning of our dataset. This is caused by a dramatic uptick in the use of GMail during early 2006. Non-GMail `POST` requests are more steady and only slowly increasing during this period.

Figures 3 and 4 show the average and median size of `GET` and `POST` transactions over time. Both transaction types show a generally increasing average transaction size over time which is likely explained by users both increasingly downloading richer content and participating in so-called web 2.0 sites that host user-provided content. The median results for `POST` transactions are interesting as they remain small and fairly constant over the study period. This indicates that simple form input that only results in the transmission of a small amount of data is prevalent throughout. For the `GET` requests we find the medians are generally an order of magnitude less than the averages. This is expected due to many previous studies that show most responses are short and a few responses carry most of the bytes—i.e., web traffic is heavy-tailed [5]. Figure 5 shows the distribution of `GET` response sizes for July 2 2007 as a typical example of the per-day distribution (this date was chosen arbitrarily as a weekday roughly in the middle of the study period). Finally, we note that the average `GET` response size in December 2006 is four times the size of the surrounding months. This anomaly is caused by a single client fetching a large series of big files. We removed this client from our analysis and plot a point on the graph to show the average size of `GET` responses without this particular client. Without the energetic client the average is similar to the surrounding months.

Figure 6 shows the median duration of HTTP connections, as well as the median time between establishing a connection and the client issuing an HTTP request. We note that the median connection duration is reduced between November and December 2007 which is explained by a reduction in the use of persistent HTTP connections

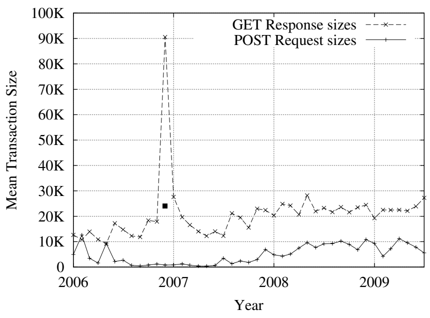


Fig. 3. Average transaction sizes

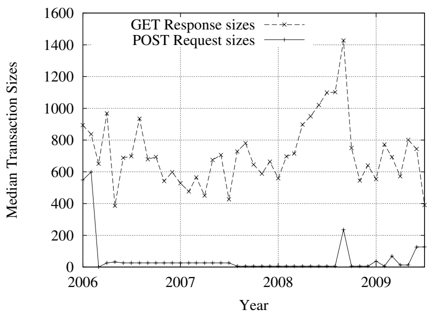


Fig. 4. Median transaction sizes

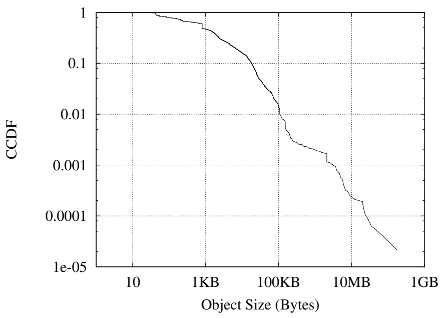


Fig. 5. CCDF of `GET` sizes for July 2 2007

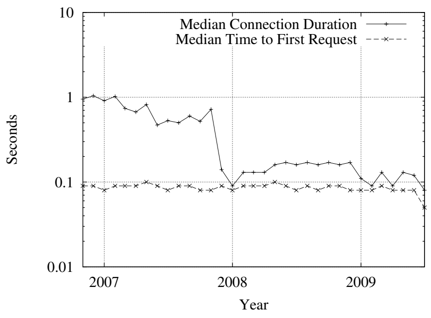


Fig. 6. Connection duration and request time

(see § 5). As connections are used for fewer objects their duration drops. Before December 2007 the median connection duration was around 1 second and after this point the median duration falls to 100–200 msec. The short duration of connections suggests that seemingly small changes to the delivery process that save modest amounts of wall-clock time may ultimately benefit the user experience more than one might think at first blush—e.g., Early Retransmit [2] and reducing TCP's traditional exponential backoff between retransmissions [10].

Figure 6 also illustrates the time between establishing a connection and sending an HTTP request. In related work [1] we study claim-and-hold attacks on web servers whereby a malicious client opens a connection and does not send an HTTP request to force the server to allocate resources that can then not be used for legitimate traffic. In figure 6 we show that the median time before an HTTP request is issued is roughly constant—at just under 100 msec—in our dataset, which agrees with the results in [1]. However, we also note that we find successful transactions whereby the time between connection establishment and transmission of the HTTP request is quite a bit longer. In particular, we find this with GMail. The 99^{th} percentile interval is roughly 246 seconds for GMail in each year, while the interval ranges from 14 seconds in 2006 to 55 seconds in 2009 for non-GMail traffic. This indicates that expecting short intervals may not be the right model for newer web application-driven web pages.

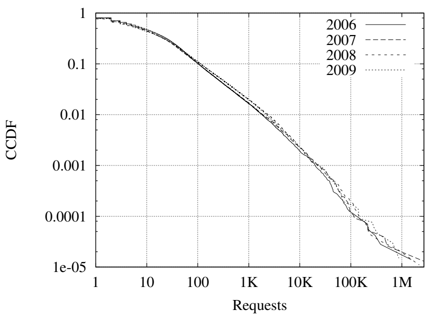


Fig. 7. Requests per hostname

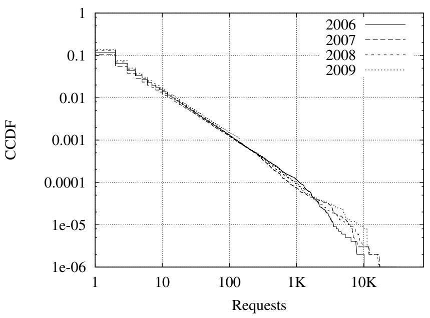


Fig. 8. Requests per object

4 User Behavior

Figure 7 shows the distribution of requests per server hostname. As discussed in § 5 all requests to a server name such as "www.cnn.com" are grouped together in this plot no matter what IP address handles the request. We see that across the years in our study the distribution of requests per hostname is similar. Many server hostnames are accessed infrequently—with the median being less than 10 requests per year. However, a small number of server hostnames are quite popular—with the maximum number of requests per hostname exceeding one million per year. We also observe that there is change in the top hostnames over time. We determined the top ten hostnames per year and find that four hostnames are within the top ten for all four years of our survey, one hostname is in the top ten in three of the years, two hosts appear in the list for two years and 17 hostnames appear only once across the four years.

Next we turn to object popularity. Figure 8 shows the distribution of the number of requests per object. The distributions are similar across the years. As seen in the plot, request popularity fits exceedingly well to a Zipf-like distribution. Across more than 3 orders of magnitude, the fall-off in the distribution matches closely to a Pareto distribution with $\alpha = 1$. Around 90% of the objects are accessed only one time.[2] Further, only a small number of objects are fetched more than 10 times. That said, there are some popular objects that are requested thousands of times over the course of a year.

Next, figure 9 shows the distribution of the unique number of objects (determined by URL) per hostname over time. We again observe stability across the years of our study. This plot shows that one-third of the hostnames we encounter (regardless of year) serve only a single unique object. Further, two-thirds of the hostnames serve ten or fewer objects. Similar to many other aspects of web traffic a small number of hostnames serve many web objects. For instance, roughly 5% of the hostnames provide more than 100 objects and hostnames top out at providing over one million objects.

Object popularity has a direct bearing on the usefulness of caching. We use our data to investigate both visible end-host caching and also potential savings from a network-based cache with our results illustrated in figure 10. First, we look at the number of

[2] As unique sets of parameters at the end of a URL generally yield distinct outputs, we consider the entire URL, including parameters, as a distinct object.

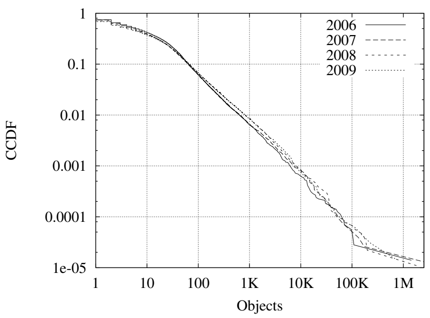


Fig. 9. Unique objects per hostname

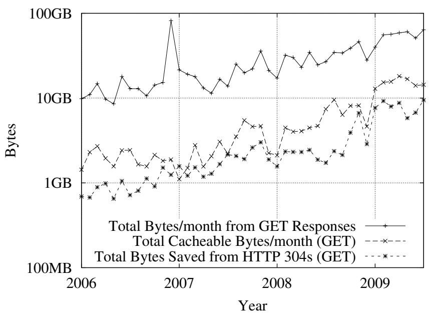


Fig. 10. Cond. GET and caching savings

bytes visibly saved by end-host caches using conditional GET requests. These requests call for a client re-requesting an object in the browser cache to include the timestamp of this cached object in the GET request. If the object has been updated the server will re-send it and otherwise will send a 304 ("not modified") response to the client, signaling that the version the client has is correct. As shown in the figure the number of bytes fetched would increase by roughly 10% without these conditional GET requests. The use of conditional GETs across the dataset increases in roughly the same manner as the overall number of bytes downloaded.

We next investigate the number of additional bytes that could benefit from caching at the institute-level—i.e., with some shared proxy cache at the border shared by all users. We first identify unique objects using the URL (including hostname) and object size. Ideally we would include a stronger notion such as a hash of the content itself, but our logs do not contain such information. We crunched 5.75 days worth of full-payload packet traces from January 2010 to assess the accuracy of our heuristic. We form tuples of URLs and sizes, (u, s), from the packet traces. In addition, for each we compute an MD5 hash of the object. For each (u, s) we record the number of MD5s observed. We find that 1.4% of the 846K (u, s) pairs in the trace have multiple MD5s. Further, we find this represents 6.7% of the bytes fetched over the course of the trace. Therefore, while we assume that if the object size stays the same the object has not changed this is wrong in a small number of cases. In addition, we calculate the amount of cachable data for each month in isolation and without any notion of timing out the objects or imposing a limit on the number or size of objects held in the cache. Therefore, our results are an upper bound on how an actual institute-wide cache would work with real-world constraints and a better understanding of the uniqueness of objects. As shown in figure 10 the percentage of bytes that could be cached by a network-wide proxy is 10–20% across most of our study. The number of cachable bytes does increase in 2009—when generally more than 25% of the GET requests could plausibly be handled by a cache. In 2009 to get the caching benefit suggested in the plot would require over 10 GB of storage space—which is quite modest in modern servers. We note that our caching results may be quite different if the population of users was larger.

5 Server Structure

Web traffic is composed of a series of HTTP transactions that in turn utilize TCP for reliable data transfer. HTTP uses one transaction per web object. Early HTTP used a separate TCP connection for each HTTP transaction, however with persistent HTTP connections an arbitrary number of HTTP transactions can be conducted over a single TCP connection. Figure 11 shows the total number of TCP connections we observe for the sampled week of each month of our dataset, as well as the average number of HTTP transactions per TCP connection. (Note, since these results depend on a solid understanding of connections, we do not include data before November 2006 as discussed in § 2.) The plot shows a fairly stable number of connections and average transactions per connection rate except for one impulse between November and December 2007. At that point we observe the re-use of TCP connections dropped by an order of magnitude. This results in a dramatic increase in the number of web connections observed. Note, figure 1 shows that the overall number of HTTP requests does not differ greatly across this event. That is, the same amount of content is being transferred, but the particulars of the underlying delivery has changed.

We believe this change in delivery pattern is due to a software change on the web clients. To verify this we determined the top 100 servers in each month by the overall number of requests (regardless of number of connections). For each server we then calculate the average number of transactions per connection. We find 66 servers to be common across the top 100 lists from the two months. We then calculate the difference in the average number of requests per connection for each of these 66 servers and find that in all but one instance the average drops in December. And, in over 70% of the cases the average requests per connection drops by at least 10 requests. This indicates that the use of persistent connections has dropped across the board and is not caused by some popular server curtailing support for persistent connections or some heavy-hitter client. We therefore conclude that this is a client policy change that is quite likely caused by a institute-wide web browser upgrade.[3]

We next turn our attention to how web sites are structured to serve content. Figure 12 shows the distribution of the number of hostnames each server IP address takes on over the course of each year in our study. CDN hosts can accommodate a wide range of logical hostnames using a server with a single IP address. The distributions are similar across the years with around 80% of the server IP addresses we encounter mapping to a single server hostname. While roughly 10% of the server IPs map to two hostnames we observe a small percentage (≈5%) of server IPs accommodating three or more IP addresses. Further, there are a handful of IP addresses that serve traffic for a large number of hostnames. Our data shows that the maximum number of hostnames observed for a single IP address is 477, 878, 1784 and 1353 for the years 2006–2009 respectively. This shows a definite increase over time. (Note, since the data only covers half of 2009 the number may well increase when the entire year is considered.)

[3] The ICSI system administrators report a minor version upgrade of Firefox (from 2.0.0.8 to 2.0.0.10) during this timeframe. While we find nothing in the Firefox change-log that indicates a difference in the use of persistent HTTP connections we believe the defaults more than likely changed in 2.0.0.10 given the observed behavior so dramatically changes at the time of the upgrade.

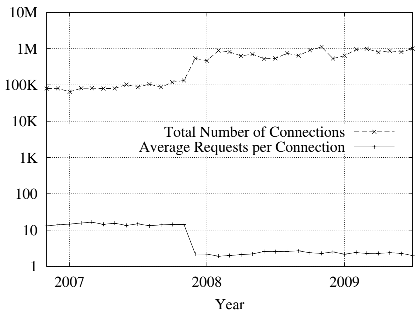


Fig. 11. Conns. / week and requests / conn

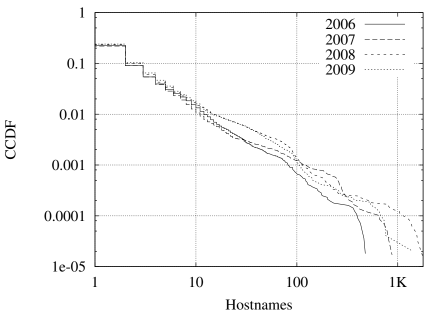


Fig. 12. Unique hostnames per IP

In addition to using one IP address to serve content from multiple server hostnames CDNs also use multiple IP addresses to serve content from a common server hostname. Generally this is done to transmit content from a server that is close to the requesting client and/or for load balancing purposes. Figure 13 shows the distribution of the number of IP addresses used for each hostname observed for each year in our dataset. The figure shows that 80–90% of the hostnames are served by one IP address. Another 5–10% of the server hostnames are handled by two IP addresses. Approximately 5% of the server hostnames are associated with three or more IP addresses over the course of the year. While relatively rare we find many hostnames that have dozens to hundreds of IP addresses over the course of a year. In particular, we note that we observe a maximum of 144, 183, 340 and 388 IP addresses for a single hostname in 2006–2009 respectively. This shows an increase in the number of hosts brought to bear to deliver content over the course of our study for some hostnames. While this trend is not general across all hostnames we believe it may indicate larger and more dynamic CDN behavior.

We next wanted to assess the degree to which content providers are relying on the content delivery networks (CDNs) to deliver their data. In this initial exploration we focus on the Akamai CDN but intend to broaden our consideration as part of our future work. A colleague provided us with a list of partial Akamai hostnames manually gathered for another project [15]. The partial hostnames in this set were determined from downloading known Akamai-based web pages from 300-400 locations around the world. The hostnames represent 12K Akamai IP addresses [15] which represents an undercount of Akamai's footprint (e.g., Akamai is cited as having 40K servers in [13]). Therefore, our accounting of Akamai traffic is highly likely to be an underestimate. We correlate the Akamai hostnames and the DNS logs produced in conjunction with the web logs by Bro. For each web log we use the corresponding DNS log to find resolutions for the Akamai names and record the associated IP addresses. We can then easily assign web traffic as Akamai traffic or non-Akamai traffic.

Figure 14 shows the percentage of the bytes fetched in response to `GET` requests that are handled by Akamai servers. Across the time period of our study we find that in general 15–30% of the bytes are delivered by Akamai.[4] However, recall that this is

[4] While we see a dip to 3% at the end of 2006, this is caused by the traffic spike at the same time—as shown in figure 3—which represents traffic caused by a single client to a non-Akamai server. This client excluded, 13% of the traffic involves Akamai.

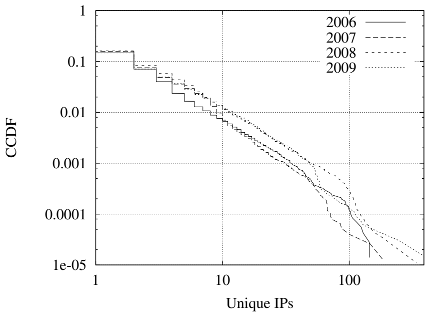


Fig. 13. Unique IPs per hostname

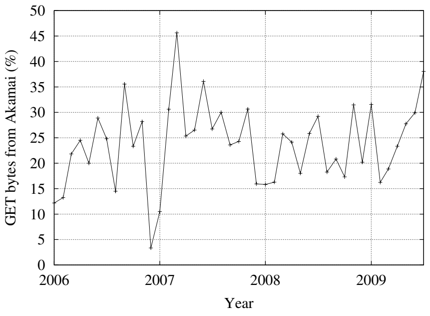


Fig. 14. Traffic using Akamai CDN servers

a lower bound due to our classification methodology. Also note that over the years of our study ICSI users accessed Akamai served content from over 9K distinct Akamai servers. The number of Akamai servers observed in a each year in our dataset is: 2.5K, 3.4K, 4.5K and 3K for 2006–2009 respectively. (Note, the 2009 count is for only half the year and so may well increase if the entire year is considered.)

6 Related Work

The large body of literature dedicated to empirical evaluations of web traffic is too vast to catalog in the space available here. A good overview, including pointers to much of the literature, appears in [9]. The topics that have been studied are diverse and numerous, with some *examples* being: characterization and modeling work [5,3], performance analysis [6], analysis of web applications [17], analysis of web technologies [14], assessments of web caching [16,4] and studies of the HTTP protocol itself [11,8]. Our work is quite similar, but serves to add additional longitudinal data to the community's body of work.

7 Summary

In this paper we have employed a three and a half year longitudinal dataset of web activity to assess web operations from numerous angles. This study represents both a reappraisal of previous work and a broadening of the viewpoint in the temporal dimension. We find that some aspects of web traffic have been fairly static over time (e.g., distribution of transaction types) while others have changed (e.g., average size of GET and POST transactions). We also develop a view of the structure of the web, including an initial understanding of the behavior of browser caches and the impact of content distribution networks, which we find to be more prominent as time progresses. While there are obviously more aspects of web operations to assess than could be fit in this initial paper, we believe our contribution will be useful in grounding the community's mental models and experiments in long-term empirical observation.

References

1. Al-Qudah, Z., Rabinovich, M., Allman, M.: Web Timeouts and Their Implications. In: Passive and Active Measurement Conference (April 2010)
2. Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J., Hurtig, P.: Early Retransmit for TCP and SCTP, Internet-Draft draft-ietf-tcpm-early-rexmt-01.txt (work in progress) (January 2009)
3. Arlitt, M., Williamson, C.: Internet Web Servers: Workload Characterization and Implications. IEEE/ACM Transactions on Networking (October 1997)
4. Barford, P., Bestavros, A., Bradley, A., Crovella, M.: Changes in Web Client Access Patterns: Characteristics and Caching Implications. In: Proc. WWW, vol. 2 (1999)
5. Barford, P., Crovella, M.: Generating Representative Web Workloads for Network and Server Performance Evaluation. In: ACM SIGMETRICS, July 1998, pp. 151–160 (1998)
6. Barford, P., Crovella, M.: Measuring Web Performance in the Wide Area. In: Performance Evaluation Review: Special Issue on Network Traffic Measurement and Workload Characterization (August 1999)
7. Fielding, R., Gettys, J., Mogul, J.C., Frystyk, H., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2068 (January 1997)
8. Krishnamurthy, B., Arlitt, M.: PRO-COW: Protocol compliance on the web: A longitudinal study. In: USENIX Symp. on Internet Technologies and Sys. (2001)
9. Krishnamurthy, B., Rexford, J.: Web Protocols and Practice. Addison-Wesley, Reading (2001)
10. Mondal, A., Kuzmanovic, A.: Removing Exponential Backoff from TCP. ACM Computer Communication Review 38(5) (October 2008)
11. Nielsen, H., Gettys, J., Baird-Smith, A., Prud'hommeaux, E., Lie, H., Lilley, C.: Network Performance Effects of HTTP/1.1, CSS1, and PNG. In: ACM SIGCOMM (September 1997)
12. Paxson, V.: Bro: A System for Detecting Network Intruders in Real-Time. Computer Networks 31(23–24) (1999)
13. Qureshi, A., Weber, R., Balakrishnan, H., Guttag, J., Maggs, B.: Cutting the Electric Bill for Internet-Scale Systems. In: ACM SIGCOMM (August 2009)
14. Schneider, F., Agarwal, S., Alpcan, T., Feldmann, A.: The New Web: Characterizing AJAX Traffic. In: Passive and Active Measurement Conf. (2008)
15. Triukose, S., Wen, Z., Rabinovich, M.: Content Delivery Networks: How Big is Big Enough? In: ACM SIGMETRICS poster (2009)
16. Wills, C., Mikhailov, M.: Studying the impact of more complete server information on Web caching. In: Proc. of the 5th Intl. Web Caching and Content Delivery Workshop (2000)
17. Zink, M., Suh, K., Gu, Y., Kurose, J.: Watch Global, Cache Local: YouTube Network Traces at a Campus Network - Measurements and Implications. In: Proc. Fifteenth Annual Multimedia Computing and Networking, ACMMCN (2008)

Author Index