# Traffic Trace Engineering

Pham Van Dung[1], Marat Zhanikeev[2], and Yoshiaki Tanaka[1,3]

[1] Global Information and Telecommunication Institute, Waseda University
1-3-10 Nishi-Waseda, Shinjuku-ku, Tokyo, 169-0051 Japan
[2] School of International Liberal Studies, Waseda University
1-6-1 Nishi-Waseda, Shinjuku-ku, Tokyo, 169-8050 Japan
[3] Research Institute for Science and Engineering, Waseda University
17 Kikuicho, Shinjuku-ku, Tokyo, 162-0044 Japan
dungpv@fuji.waseda.jp, maratishe@aoni.waseda.jp,
ytanaka@waseda.jp

**Abstract.** Traffic traces captured from backbone links have been widely used in traffic analysis for many years. By far the most popular use of such traces is *replay* where conditions and states of the original traffic trace are recreated almost identically in simulation or emulation environments. When the end target of such research is detection of traffic anomalies, it is crucial that some anomalies are found in the trace in the first place. Traces with many real-life anomalies are rare, however. This paper pioneers a new area of research where traffic traces are engineered to contain traffic anomalies as per user request. The method itself is non-intrusive by retaining the IP address space found in the original trace. Engineering of several popular anomalies are shown in the paper while the method is flexible enough to accommodate any level of traffic trace engineering in the future.

**Keywords:** Traffic, Analysis, Trace, Replay, Simulation, Emulation, Anomaly.

## 1 Introduction

Many traffic traces captured from backbone links are available for network research [1]. In fact, there are many research papers and tools [2] that can analysis traffic content of such traces. On the practical side, one of the most popular uses of traffic traces is *replay* where conditions and states of the original traffic are recreated almost identically in simulation or emulation environments. When the *replay* aims to detect traffic anomalies, however, reality is such that it is not always easy to find traces which inherently contain anomalies one expects to find and detect. Having studied the content of a traffic trace, one might want to modify the trace in some way to increase the share of a certain traffic type or add traffic from malicious attacks.

On the other hand, several studies recently were dedicated to classification of traffic types [3], [4]. To the best of authors' knowledge, the common aim of such classification methods is to identify all traffic types present in a trace, namely to find out which applications generate which traffic type and what is the share of traffic held by each type in total traffic soup.

This paper goes one step further by not only allowing to classify traffic in order to discover the traffic content, but also by facilitating various engineering routines among various traffic types. The paper uses IP aggregation to split a trace into traffic entities where each is composed of a traffic node and a number of connections coming from its communication parties. In this paper, traffic node is defined as a collection of IP addresses which share a common prefix. Traffic classification is done by using basic properties of a traffic entity. Six particular types of traffic each representing a popular traffic type in real life are defined while the method can also fit other traffic types if needed.

When properly classified into individual traffic types, traffic trace can also be subjected to engineering in order to obtain more traffic of a given type. In this paper, the keyword "traffic engineering" stands for changing traffic of some type(s) into another, a targeted type. The modification process is implemented within a well controlled environment with a set of rigid rules created in order to retain sanity in the morphing process.

Engineering of several traffic types are presented in this paper as an example, while the method itself is capable of dealing with other types as long as those are well defined. The paper describes in detail how to manipulate each possible type in turn as well as all of them at once to obtain more traffic of some typical types. Results in terms of number of major occurrences of each traffic type in the trace prove the fact that a traffic trace can be easily engineered to obtain more of target traffic.

The remainder of this paper is organized as follows. Six traffic entities (occurrences of traffic types in trace) are defined in Section 2. Section 3 stipulates a set of regulations used for the process of engineering. Section 4 presents the proposed algorithm of discovering communication parties and describes how to manipulate traffic in order to obtain more of target traffic. Results of engineering of three typical traffic types as well as evaluation of performance are shown in section 5. Finally, the conclusion and directions of future work are drawn in Section 6.

## 2 Traffic Entity

The concept of traffic entity is created when IP aggregation is used to cluster a number of IP addresses which share a common prefix into a *traffic node*. While traffic node is simply the prefix itself, traffic entity is characterized by a number of properties including the number of IP addresses inside the prefix, number of *destinations* in connections originated from the prefix, number of *sources* in connections terminated at the prefix, number of *incoming/outgoing connections* and incoming/outgoing *traffic volume*. As a major concept throughout the paper, *traffic entity* is defined as a traffic node and a number of incoming and outgoing connections to or from it. By applying the concept of traffic entities, it is possible to successfully split traffic into a number of traffic entities interacting with each other on a timeline.

The properties of a traffic node bring out an image of users inside a prefix by indicating how crowded they are and how actively they interact with the rest of network community. Correspondingly, this paper uses combinations of properties as criteria to classify traffic entities. After classifying entities, basic traffic units such as *packets*, *flows*, *connections*, and *nodes* are thereof labeled by the type of entity they belong to.

In addition, the remaining traffic which does not belong to any types defined above is considered as unclassified traffic. Among many types the method works with six particular entity types as illustrated in Fig. 1.

In Fig. 1, each traffic entity is composed of a traffic node depicted by a clouded shape on the left with one or several IP addresses inside and connections depicted by lines with arrows to or/and from other cluster of users on the right. The size of the cloud on the left indicates the density of a traffic node. Similarly, the arrows show the connection direction while the thickness of lines indicates how much traffic in bytes the connections transfer. In order to determine whether a property is small or large in the classification process, this paper sets two thresholds, *min* and *max* for each property. The thresholds are inferred from real values of the property of all the nodes. The thresholds are picked as follows. Given that there are one hundred different values of a property sorted in an ascending list, the *min* threshold is chosen as value of the 30th element in the list while the *max* is value of the 70th element. A property of a node is considered as very small if its value is less than the *min* and very large if its value is greater than the *max* threshold.

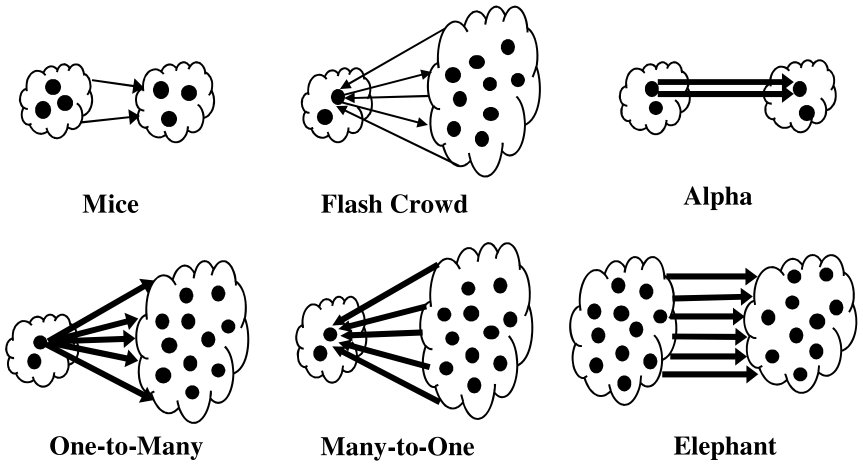

**Fig. 1.** Traffic entity definitions

Each traffic entity type in Fig. 1 maps closely to a popular type of traffic in real life. Mice entity represents the case where a small group of users creates a small amount of traffic and sends to few other users over Internet. Flash Crowd entity stands for a famous social phenomenon which occurs when a web server suffers a surge in number of requests within a short period of time due to large social events, thus, rendering the server unavailable [5]. Alpha entity represents high-rate transfers between a pair of a single source and a single destination [6]. One-to-Many and Many-to-One traffic entities are the cases where a large amount of traffic exchanged between a single server and a group of users. These might be content distribution where many users together download files from a server or content aggregation where users together upload files to a server. Finally, Elephant entity represents the case where a large group of users actively join and create much traffic in network [7].

## 3 Rules of Modification

The ultimate goal of this paper is to modify traffic among types that is to change one or several traffic types to a given target type. Obviously, before performing the modification, reasonable boundaries for the process have to be defined. This section stipulates which operations are permitted during modification as well as sets a number of related constraints. The modification process itself from this point on will be referred to as *morphing*.

***Constraints:*** This paper performs morphing in order to get more of a given traffic type while trying to be as unobtrusive as possible. Below is the list of rules governing the morphing process:

1. It is not allowed to generate new IP addresses. Instead, the method should do within the IP space extracted from the original traffic trace.
2. The modification process must guarantee that the total number of packets and flows stay unchanged no matter how they are manipulated.
3. Based on a fact that source and destination of a packet in a packet trace are always physically located on opposite sides of the link, new packets, if any must also conform strictly to this rule.
4. When changing traffic type X to type Y only the traffic belonging to traffic entity X is to be modified.

***Possible operations:*** This paper limits possible operations to modifying packets and flows. In the paper, packets and flows are seven-tuple units (time, source IP address, destination IP address, source port, destination port, protocol, size). The modification process is allowed to change the time and the size of a packet. However, new packet time must be in the range of duration time of the trace and new packet size must be a standard size of the current protocol. It is also possible to replace source or destination IP or even both of them. In case both are replaced, new source IP address must be the old destination IP address, and vice versa, to ensure that no new communication pairs are generated out of thin air. Depending on each target type, one or several operations will be used to manipulate to obtain expected results.

## 4 Traffic Type Morphing

The morphing process can be divided into two phases. The former is to discover all IP space related to the traffic node within the entity. For a given target traffic type of morphing, the goal of this phase is to determine sides for every IP address of the type. In plain words, all IP addresses must be attributed only to side A or side B provided that A and B are two sides of the backbone link.

The second stage is to select all traffic entities suitable for morphing and manipulate their connections by using the rules and guidelines defined above.

## 4.1 Pre-morphing IP Space Discovery

This paper uses a simple procedure to discover the entire IP space related to a given traffic entity. From this point on this process will be referred to as *Round Trip IP Space Discovery* (RTSD) as illustrated in Fig. 2. The paper exploits IP aggregation to diminish overhead of the algorithm by operating with traffic nodes instead of individual IP addresses. More specifically, the RTSD algorithm attempts to determine how many IP addresses can be used in morphing itself as well as which side relevant to the source traffic node they are found at.

This paper assumes that a traffic trace has the most popular traffic node, a node which has the largest number of total incoming and outgoing connections. General idea of the algorithm is to track out side for sources based on the communication pairs extracted from the original trace. The most popular node is selected as a starting point in attempt to make the overall process completed in the smallest number of round trips with the maximum number of nodes localized. The procedure is performed in round trips where each performs two similar tasks. The former is to discover nodes in side B from pairs where either source or destination node has already been known as located in side A. In a similar manner, the latter is to discover nodes in side A. The algorithm completes when no more nodes can be discovered, which normally happens when the entire IP space within the trace has been successfully located and attributed to the either side.

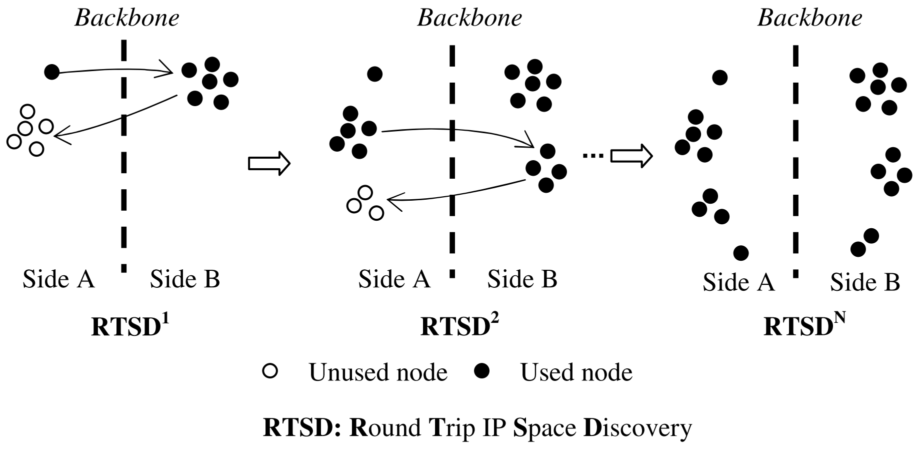

**Fig. 2.** Process of discovering side of sources

In regard to performance of the RTSD algorithm, although most sources are either directly or indirectly connected to the most popular node, there are always isolated communities where their member nodes only communicate with each others. In fact, these isolated nodes can be forced to get involved in the process of discovering by decreasing length of the prefix used in the aggregation. In addition, multiple round trips also help to improve the performance of the algorithm. The more round trips the algorithm takes to perform, the more traffic nodes are localized.

## 4.2 Traffic Morphing

Once the RTSD algorithm has completed its work, the morphing process can begin. Due to a fact that each entity type has its own characteristics, the process of morphing must be implemented individually for each target type. The differences among cases of different target types are indicated by operations used to morph the source entities. The set of operations is determined based on the definitions of both *src* (source) and *dst* (destination) entity types. However, there is a generic four-step algorithm which can be applied to every case of modification as follows:

1. Select all entities of *src* type from total traffic entities.
2. Use **RTSD**(*src*) to discover IP space that will be later used for morphing.
3. From the entities whose sources are localized in the step 2, select suitable ones based on the definition of the *dst* entity type.
4. Based on the definitions of both the *src* and *dst* type, determine proper operations and morph *n* source entities into *m* target ones using the operations to create a number of new *src* entities according to the user's requirement, where *n* != *m* in most cases.

After selecting all entities of the *src* type, the generic algorithm invokes RTSD(*src*) to discover IP space of *src* entities. Note that only the *src* nodes are used in the run of RTSD. This certainly limits the performance of the RTSD, yet guarantees that the algorithm does not waste time on localizing side for an enormous number of superfluous sources. In Step 3, among localized entities only ones which have the *potential* to become *dst* entities are selected for further morphing. The algorithm compares properties of current entities with those in definition of *dst* entity to make decision for such selection. Next, the algorithm selects a certain set of operations to morph *src* into *dst* entities. This part is different for each *src* and *dst* types.

**Table 1.** Interchangeability with priorities among traffic types

**MI:** Mice **AL:** Alpha **FC:** Flash Crowd
**OM:** One-to-Manny **MO:** Many-to-One **EL:** Elephant **UN:** Unclassified

| *src* / *dst* | **MI** | **AL** | **FC** | **OM** | **MO** | **EL** | **UN** |
|---|---|---|---|---|---|---|---|
| **MI** | 0 | 2 | 0 | 0 | 0 | 0 | 1 |
| **AL** | 2 | 0 | 3 | 4 | 5 | 0 | 1 |
| **FC** | 2 | 0 | 0 | 4 | 3 | 0 | 1 |
| **OM** | 0 | 0 | 2 | 0 | 0 | 0 | 1 |
| **MO** | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **EL** | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Not all traffic types can be subject for morphing. Based on experience, for each *dst* this paper recommends a priority list for *src* traffic types. Table 1 shows the interchangeability among types with priorities in range from 0 to 5 where 0 means it is impossible to change *src* to *dst* traffic, 1 means the *src* is the best choice for creating more *dst* traffic and 5 means *src* is the worst case. The table shows that unclassified traffic (UN) is the best choice for any *dst*. This is because changing unclassified traffic not only leaves other types intact, but also helps to convert an amount of abundant traffic to one of the types.

Because of space limitations, this paper offers the morphing examples for two cases of *dst* traffic, Elephant and Flash Crowd. Creating more Elephant entities is quite simple because no RTSD is required and only a single operation is needed while creating more Flash Crowds is the most complicated case in all target traffic types. Henceforth, the paper also uses shortened names of traffic types as in Table 1 for the purpose of limiting space.

**More El:** Table 1 indicates that only UN traffic can be used for creating more EL traffic. Morphing UN into EL does not require any operation that can violate the link-side constraint. Therefore, it is not necessary to run RTSD for this case. Next, to perform the selection in the step 3, only UN nodes that have many IP addresses aggregated inside and many outgoing connections are able to become EL entities therefore are chosen. Each of these entities easily becomes an EL entity by increasing sizes of packets of outgoing connections. More concretely, size of a selected UN packet is converted to a random size in the list of existing EL packets of the current protocol. Thus, changing packet sizes is the only operation used to manipulate in the step 4 of the general algorithm.

**More FC:** Unlike the case of EL, creating more FC traffic is complicated because of the nature of flash events. It has been stated that FC packets often arrive at servers with a rule of time [8]. This paper creates new FCs based on the flash crowd generation model proposed in [8]. According to the model, a FC has three phases in sequence, *ramp-up*, *sustained* and *ramp-down*, where each is accompanied by a function of time to calculate a number of requests to server. Hence, to create a new FC, this paper makes requests from selected sources to a selected server with request counts as well as packet arrival times are modified to conform to the model in [8]. In addition, for some cases of *src* types, many entities rather than one are used to construct a new FC. In those cases, the algorithm needs to invoke the RTSD algorithm for discovering sides for the *src* entities.

Let us turn back to the four-step procedure. Potential types in descending priority for making more FC traffic are UN, MI, MO and OM as in Table 1. Among the types, it is easiest to change MO traffic to FC. Certainly, not all MO nodes can become FC entities. Only those who have little outgoing traffic are usable. Each selected MO node can be modified to become a new FC by changing its incoming connections into incoming requests to its server. Besides, arrival times of requests must also be changed to conform to the rule in the model. Changing OM to FC can also be accomplished similarly to the case of MO, with an additional task of reversing the direction of connections. Here, reversing direction means replacing source IP address of every packet of a connection with destination address and vice versa.

Changing UN and MI traffic to FC involves more operations because an individual entity of each case is not eligible to become a new FC. Fortunately, the procedures of changing these two traffic types to FC are similar to each other. To change UN entities to new FCs, only UN nodes that have very few sources inside and little incoming traffic are eligible. Each new FC will include a server node and a list of user nodes. Then, a unique IP address inside a server node will be determined to become a real

FC server. Concerning the link-side constraint, user nodes must be selected from different sides of the server. Namely, RTSD is mandatory in these cases. Eventually to create a new FC, all outgoing connections of the server node are converted to incoming requests to the server address. Meanwhile, outgoing connections of user nodes are converted to requests destined to the server by replacing destinations of their packets and changing their packet sizes. Again, the packet arrival times need to be rearranged to conform to the time rule in the model.

## 5 Results and Performance

This paper uses 30 seconds of a *WIDE* packet trace [9] as a playground to analyze traffic content and modify traffic types. The modification results are shown in Table 2 in the form of traffic content distribution in terms of entity counts of traffic types before and after morphing. In regard to the original traffic content, the trace contains 3693 traffic entities. Table 2 shows that most entities are MI as its entity count is an order of magnitude greater than that of other entity types. Originally, there are 6 FCs in the input traffic. There are 4327 traffic nodes extracted from 5792 communication pairs of the nodes in the trace. Among those, 634 nodes are discovered as unclassified, corresponding to about 14% of total.

For each of three typical target types, EL, MI and FC, the table shows results of changing each of possible types separately as well as result of changing all of them at once. As mentioned before, unclassified traffic is not counted as an entity type. However, in Table 2, it is the first choice in all three cases of modification.

**Table 2.** Distribution of traffic content before and after morphing

| Type | Before | More EL | More MI | | | More FC | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | UN | UN | AL | All | UN | MI | MO | OM | All |
| MI | 3246 | 3017 | 3593 | 3328 | 3718 | 2929 | 1042 | 3018 | 3112 | 1124 |
| FC | 6 | 4 | 4 | 5 | 4 | 13 | 24 | 9 | 10 | 36 |
| AL | 157 | 151 | 145 | 2 | 1 | 152 | 148 | 162 | 146 | 138 |
| OM | 38 | 31 | 35 | 42 | 36 | 35 | 34 | 41 | 0 | 0 |
| MO | 22 | 21 | 20 | 23 | 22 | 22 | 21 | 0 | 25 | 1 |
| EL | 224 | 526 | 218 | 244 | 221 | 205 | 207 | 243 | 203 | 191 |

The Table 2 shows that 302 new EL entities are created by morphing UN into EL nodes. Surprisingly, not only EL traffic is affected by the modification process but all types of traffic are. In fact, morphing process affects properties of traffic nodes which results in changes in their thresholds. In case of EL, morphing UN into EL increases outgoing traffic of some UN nodes, which consequently increases incoming traffic of some other nodes as well. As a result, *max* threshold of outgoing traffic increases while its *min* threshold decreases. Similar reasoning can be applied to incoming traffic property. Eventually the changes of the thresholds explain why the entity counts of all other types decrease. This side effect, however, is diminished by the success of creating a larger number of additional traffic entities.

One can get more MI traffic by morphing UN into AL traffic. While the UN can be used to make more 347 MI entities, the AL can only help to make more 82 ones. Besides, changing UN makes entity counts of all other types decreased while changing AL only takes one FC away, yet increases OM, MO and EL counts. In case even more MI traffic is required, the maximum of 3718 entities can be created, among them 472 new entities, by choosing to morph both UN and AL into MI.

With 6 existing FCs in traffic, one can get more FCs by several morphing processes. UN traffic can be used to make 7 new FCs, MI helps to make 18 more, MO makes 3 and OM makes 4. However, if one individual source type alone does not make enough new FCs, one can pick entities from other types and use them for morphing. In case of 30 new FCs, all of possible types are operated at once. Especially, changing UN and MI both decrease all remaining types; changing MO makes more AL, OM and EL traffic but reduces MO one and changing OM reduces MI, AL and EL traffic but makes more MO.

With regard to the performance, the engineering method is implemented using database tables. For this reason, this paper evaluates the performance of the method in terms of amount of time required to complete the process, the number of times the algorithm required to read an entry or update/write it back to the database as well as the ratio of localized nodes to total nodes the RTSD algorithm can discover.

**Table 3.** Evaluation of performance in terms of database operations

| **Metrics** | **More EL** | | **More MI** | | **More FC** | |
|---|---|---|---|---|---|---|
| | RTSD | Morphing | RTSD | Morphing | RTSD | Morphing |
| % Nodes picked | 0 | 31% | 0 | 34% | 77% | 37% |
| Reads | 0 | 256 | 0 | 284 | 4048 | 792 |
| Updates/Writes | 0 | 1820 | 0 | 1488 | 2998 | 5413 |
| Time (s) | 0 | 51 | 0 | 47 | 154 | 143 |
| Total Time (s) | 51 | | 47 | | 297 | |

This paper uses 12-bit mask (20-bit prefix) for IP aggregation. Table 3 shows the evaluation of performance with various metrics for three cases of engineering. First, RTSD is only required in the case of FC where 77% of sources are discovered in the trace. As for the ratio of nodes selected in the step 3 of the generic algorithm presented earlier, 37% of *src* nodes are selected to create FC traffic while that of the EL is 31% and of the MI is 34%.

The morphing algorithm reads from database when it searches for nodes and updates (writes) whenever a new entity is created with a number of packets and flows are modified. From the table, number of updates is much greater than that of the reads indicating that the algorithm mainly spends time on updating data. Meanwhile, instead of updating, the RTSD algorithm writes new discovered nodes to database. With total 4327 traffic nodes, using 4 round trips to run, the RTSD can discover side for 2998 nodes among all nodes of possible types as seen in Table 3.

Finally, as for the processing time, without running RTSD, the processes of engineering EL and MI traffic only took less than 1 minute of actual time to complete. On the contrary, in the case of FC, the algorithm needs 154 seconds for the RTSD and 143 seconds for the morphing, totaling to almost 5 minutes to complete.

## 6 Conclusion

This paper proposes a new method which can be used to morph one or more types of traffic into one or more other types within the commonly known traffic types. The model enables to engineer traffic trace by manipulating entities under predefined regulations. The morphing process happens at packet lever but guarantees that the original IP space of the traces remains intact. In order to deliver on this promise, the process conforms to number of rules presented in this paper and used both as guidelines and as constraints in the morphing process. The results of using the proposed method in practice were also shown for a few well understood traffic types.

Results of morphing in terms of entity count for each type and its share in total count indicated that target traffic of a given type can be easily created as per user request with the help of the proposed method. The evaluation of performance showed that the engineering method can be completed in a reasonably short amount of time on fairly large traffic traces. For future work, authors plan to define new types of entities in order to allocate the share of unclassified traffic present in each trace. Secondly, while traffic entity count is the only metric mentioned in the discussion, other basic metrics like IP address count, packet count, flow count and byte count will be included in future research works in order to get a deeper insight into traffic types prior to performing the morphing process proper.

## References

1. Internet Traffic Archive (ITA), `http://ita.ee.lbl.gov/html/traces.html`
2. Pham Van, D., Zhanikeev, M., Tanaka, Y.: Effective High Speed Traffic Replay Based on IP Space. In: ICACT, Phoenix Park, Korea, Feburary 2009, pp. 151–156 (2009)
3. Karagiannis, T., Papagiannaki, K., Faloutsos, M.: BLINC: Multilevel Traffic Classification in the Dark. In: ACM SIGCOMM, Philadelphia, USA, August 2005, pp. 229–240 (2005)
4. Szabó, G., Szabó, I., Orincsay, D.: Accurate Traffic Classification. In: IEEE WOWMOM, Finnland, June 2007, pp. 1–8 (2007)
5. Jung, J., Krishnamurthy, B., Rabinovich, M.: Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In: WWW Conference, Hawaii, USA, May 2002, pp. 532–569 (2002)
6. Lakhina, A., Crovella, M., Diot, C.: Characterization of Network-Wide Anomalies in Traffic Flows. In: Internet Measurement Conference, Italy, October 2004, pp. 201–206 (2004)
7. Estan, C., Varghese, G.: New Directions in Traffic Measurement and Accounting. In: ACM SIGCOMM, San Francisco, USA, November 2001, pp. 75–80 (2001)
8. Pham Van, D., Zhanikeev, M., Tanaka, Y.: Synthesis of Unobtrusive Flash Crowds in Traffic Traces. In: IEICE General Conference, Ehime, Japan, March 2009 BS-4-19 (2009)
9. WIDE Project, `http://tracer.csl.sony.co.jp/mawi/`