Measurement Architectures for Network Experiments with Disconnected Mobile Nodes

Jolyon White, Guillaume Jourjon, Thierry Rakatoarivelo, and Maximilian Ott

NICTA[⋆]
Australian Technology Park
Eveleigh, NSW, Australia
firstname.lastname@nicta.com.au

Abstract. Networking researchers using testbeds containing mobile nodes face the problem of measurement collection from partially disconnected nodes. We solve this problem efficiently by adding a proxy server to the Orbit Measurement Library (OML) to transparently buffer measurements on disconnected nodes, and we give results showing our solution in action. We then add a flexible filtering and feedback mechanism on the server that enables a tailored hierarchy of measurement collection servers throughout the network, live context-based steering of experiment behaviour, and live context-based control of the measurement collection process itself.

Keywords: measurement, testbeds, mobile, OML, disconnected measurement.

1 Introduction

Distributed networking experiments require distributed measurement collection systems. Approaches to remote network measurement collection range from ad-hoc methods used in academia through to large, commercial systems deployed by network operators. Ad-hoc methods are typically sub-optimal, error-prone, and time consuming, but available measurement and monitoring frameworks [11,12] tend to be prohibitively complex for use in many research projects. A measurement framework for network research should be simple to use and administer, but must be flexible enough to match the heterogeneous, dynamic needs and environments that usually characterize it.

Mobile networking research is a good example. Indeed, for static testbeds, a simple client/server measurement collection architecture is adequate [5], as long as the rate of measurement output does not influence the studied phenomena, and does not overload the collection server. If a testbed network includes mobile nodes, some nodes may not always be connected to the network. In that

⋆ NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

case, what should happen to the measurements that the disconnected nodes are generating?

On the other hand, in an experiment where all nodes are always connected, the rate of generation of measurement data by even a single node may congest either the network, the measurement collection server, or the client applications generating the measurements. This can result in lost measurements; worse still, it can lead to the measurement collection activity influencing the behaviour of the network under observation, and with it, the results of the experiment.

These two different problems can both be solved by making a single but important change to the architecture: namely, the addition of a proxy server on the experiment node, effectively a queue, to act as an intermediary between the client applications and the measurement collection server. Once the measurement architecture contains such proxy servers on the experimental nodes themselves, a further innovation of the architecture becomes apparent, that to our knowledge has not been attempted before. We extend the proxy server to implement a measurement database instead of just a queue, which allows us to perform measurement-based feedback to the experiment applications themselves in what we term *distributed, context-based experiment steering.* This leads to a flexible hierarchy of measurement servers for future advanced testbed networks.

In this paper, we consider the architecture of measurement collection frameworks in detail:

– We describe the two problems of mobility (Section 3.1) and measurement bandwidth constraints (Section 3.2).
– We show how both of these problems can be solved by introducing a proxy server to buffer measurements on the local node before sending them to the central measurement server (Section 3.3).
– We give some quantitative measurements to demonstrate the benefits of this approach (Section 3.4).
– We discuss extensions to the proxy server to allow measurement-based experiment steering (Section 4).
– We compare and contrast our architecture to existing measurement frameworks (Section 5).

The measurement architectures described in this paper are embodied in OML2, the second generation Orbit Measurement Library, which we have developed and made freely available at [3]. OML is a generic measurement framework capable of instrumenting the entire software stack, and is not just limited to network-specific measurements.

2 Background

To set the scene for this paper, we begin with a description of the testbed network environments that we are considering, and a simple, naïve client/server architecture for measurements that we use as our starting point.

2.1 Testbed Architectures

Fig. 1 shows a general testbed architecture. The *experiment nodes* participating in the experiment use one or more *Experiment Networks* (EN) to perform the networking tasks that comprise the experiment itself. Meanwhile, the *control nodes* communicate with the experiment nodes using a separate *Control Network* (CN) to perform tasks such as imaging the nodes with an operating system at the start of the experiment, bringing the experiment nodes up, starting the applications that participate in the experiment on the experiment nodes, logging status and error information, and ensuring orderly shutdown of the experiment once it is complete.[1]

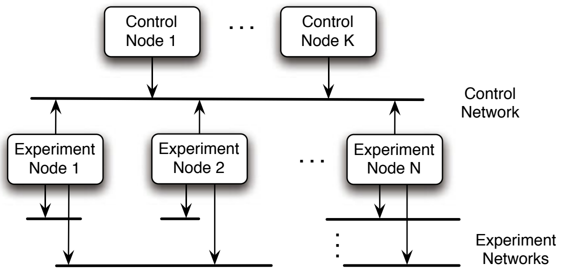

Fig. 1. Generic testbed network architecture

The architecture in Fig. 1 can contain heterogeneous experiment nodes, each node connected to different experiment networks. In practice, there will be variations in the hardware capabilities and available interfaces on each experiment node. With mobile nodes, the connectivity may even change mid-experiment.

The separate control network minimizes the impact of control tasks on the behaviour of the experiment tasks, so that the underlying protocols, algorithms, and applications can be studied in as much isolation as possible. This improves quality of results and repeatability. However, sometimes we do not have the luxury of a separate control network. The node hardware might not support enough interfaces of the right type, or the separate infrastructure required for a control network might not be available for some nodes. Mobile nodes often have these properties.

We have drawn the control and experiment networks as single network segments, but this is just for simplicity: the actual network topology of each network could be more complex. Also note that infrastructure nodes such as routers could also be participating in the experiment and generating measurements.

2.2 Experiment Node Architecture

Each experiment node runs a number of applications and services (i.e., daemons) that execute the tasks required to run the experiment itself, as shown in Fig. 2.

[1] We use OMF, a control framework that we have developed, to perform these tasks on our own testbed networks [16].

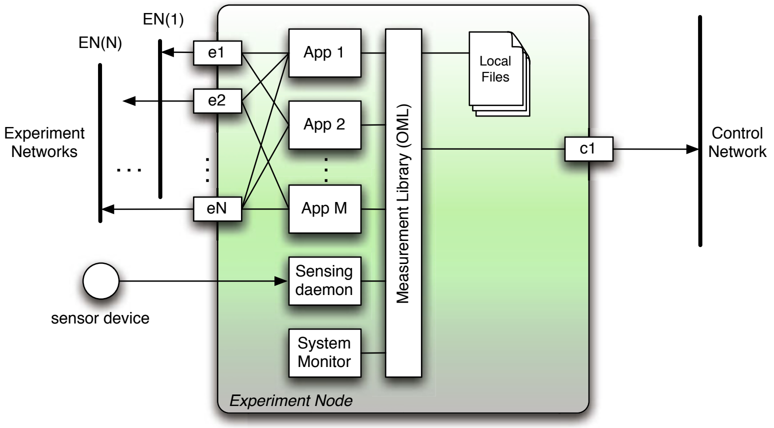

Fig. 2. Architecture of an experiment node, showing some applications, a system monitor daemon generating measurements from information provided by the operating system, and a sensor daemon generating measurements from an input sensor device

These applications and services can communicate with other experiment nodes using the interfaces $e1$–eN. They can also access and monitor operating system information and local devices, such as GPS receivers, temperature sensors, and pressure sensors.

The applications and services perform measurements of the system under study, measuring quantities such as:

– network characteristics and impairments (e.g., bandwidth, packet loss rate);
– local context information (e.g., RAM or CPU usage); and
– device-generated data (e.g., GPS coordinates, temperature, pressure),

for example. They use functions provided by the OML measurement library to send their measurements either to a file on the local filesystem, or to a measurement server on the control network via $c1$. OML is flexible enough that the applications can send measurement data to multiple measurement servers if desired. Fig. 2 shows the general case. The node may have only one experiment network interface ($N = 1$) and it may have to send and receive control and measurement data over the experiment interface if no separate control interface is available.

The experiment nodes may have a wide variety of hardware, operating systems, attached peripheral devices, and networking interfaces. Thus, the measurement architecture must be portable, flexible, and efficient enough to cope with such a wide range of platforms.

2.3 Client/Server Measurement Architecture

The simplest distributed measurement architecture, which is our starting point, is a client/server architecture. OML operates in this fashion in its most basic configuration. Fig. 3 depicts the data path from a single experiment application to the server in OML.

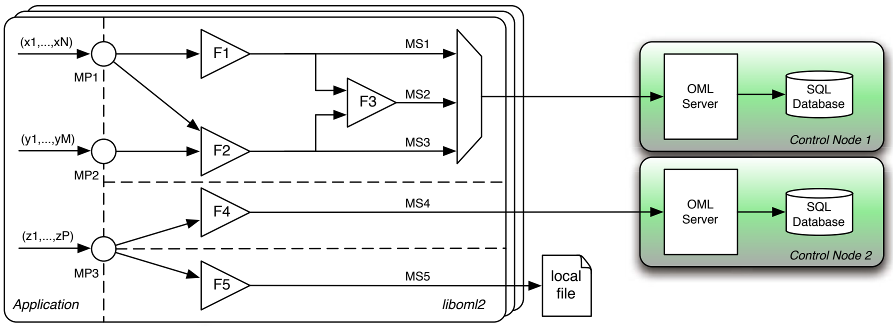

Fig. 3. Measurement data path in OML. The application illustrated defines three measurement points, and the user has configured the library to generate five measurement streams.

The application defines a number of *measurement points* (MP) into which it injects a stream of typed measurement tuples. Each MP is an interface to the client library, `liboml2`. The client library creates a number of *measurement streams* (MS), based on the run-time configuration specified by the user in an XML file, to match the needs of the experiment. Each MS filters the MP inputs in a configuration defined by the XML file. OML supports built-in and user-defined filters. Fig. 3 shows that an MP can be a source of data for multiple MS's (MP1 participates in streams MS1, MS2, and MS3), and that filter outputs can be combined to form new streams (filters F1 and F2 are inputs to F3, which generates stream MS2). Measurement streams can be sent to an OML server or a local file (also configurable via the XML file) and different measurement streams can be sent to different destinations, including potentially multiple OML servers. The filter outputs are also typed tuples.

Currently OML supports integer, floating point, and string data, and we have plans to add support for more data types such as blobs. OML uses a one-way protocol initiated by the client. Both text and binary versions of the protocol are available, and we are currently evaluating adopting IPFIX [6].

The server collects data from each experiment and sends it to a storage backend. Because the measurement streams consist of sequences of typed tuples, they are well suited to be stored in tables in a relational database; currently the concrete backends supported by OML are SQL databases. OML imposes very little structure on the measurements collected to remain as flexible as possible and support an evolving research context. The SQL database allows us to offload the problem of devising our own measurement storage format, and provides easy and efficient result querying. OML currently supports SQLite directly, but some OML users have added support for PostgreSQL. We plan to extend OML to directly support multiple database backends in the future.

There is a table in the database for each measurement stream in each client application; the same application can be running as part of the same experiment on multiple nodes, in which case all of their measurement outputs will be stored in the same table.

3 Measurement in Dynamic Networks

We now describe two scenarios where the assumptions underlying the client/server architecture are broken, and we further show how our proxy-based architectural enhancement addresses these problems. These examples are informed by the previous experiments of users evaluating their own research prototypes on wireless testbeds, such as the ORBIT or NICTA testbeds [17]. Thus, they represent real problems that users had to overcome to advance their research agendas.

3.1 Mobile Nodes

Sometimes when users perform experiments involving mobile nodes, they are explicitly interested in studying the behaviour of networking technologies and algorithms when the mobile nodes move outside the testbed network's normal wireless coverage. For example, smart phones typically have multiple radio interfaces, such as WiFi, 3G, and WiMAX. We may be interested in the behaviour of a distributed algorithm that preferentially favours a low-cost radio interface (WiFi) when available, but falls back on a more expensive interface (3G, WiMAX) if no other networks are available in the mobile handset's vicinity [14]. They may even go out of range of all wireless networks for a period.

Such experiments could be done with real mobile handsets, or they could be done with mid-range hardware emulating the mobile handsets. In either case, this configuration causes two problems for measurement collection.

The first problem is that if measurements are sent during the experiment, then the measurement traffic must often be sent over one of the experiment network interfaces, which may interfere with the experiment itself and taint subsequent measurements. Depending on the testbed configuration, the measurement server may not even be reachable from any of the experiment networks, and this situation could even extend beyond the duration of the experiment. This leads to the second problem: what should the mobile node do with the measurements that it generates while it is out of range of the control network?

We have two options: either drop measurements while the control network is not reachable, or buffer them until the mobile node reconnects to the control network. Discounting the first option as undesirable, we must buffer.

3.2 Throughput-Constrained Measurement

Even in networks with a static topology, we still sometimes need to buffer measurement data on the local node. If the experiment involves high traffic rates on high bandwidth interfaces, then the rate of generation of measurements can also be very large, and the datapath to the measurement server can become congested, risking either loss of measurement data or changes in the behaviour of the experiment applications due to delays in the measurement datapath.

One of the primary aims of the filtering facility of `liboml2` is to allow reduction of the measurement data that needs to be transmitted to the server, for instance, using averaging. However, in some circumstances the experimenter might want to observe effects that the filtering would discard. In that case another solution is required: buffering measurements on the experiment node.

3.3 Proxy Servers

Recalling that we want our measurement framework to be as convenient as possible for researchers to use, we want to ensure that buffering measurements does not force complicated modifications to the client applications. Our solution is to create a separate *proxy server* on the experiment node. The proxy server acts as a FIFO queue, but allows the experimenter to gate the FIFO output.

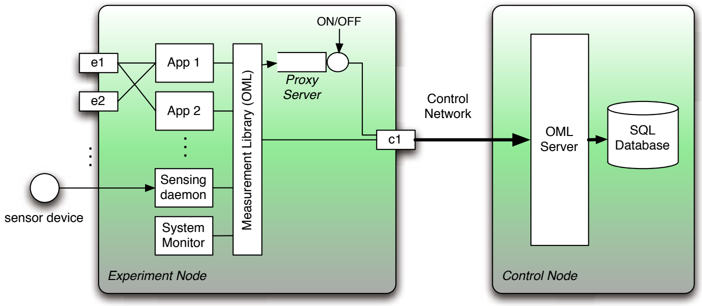

Fig. 4. Measurement architecture with proxy servers

Figure 4 shows the proxy server architecture. The proxy server presents an interface to the client applications that is identical to the regular OML server: it supports TCP or UDP socket connections using the same protocol as the server. It is thus transparent to the client applications, which do not need to be modified or re-compiled. The measurement server protocol (TCP/UDP), address, and port number are run-time configuration parameters, specified on the client application's command line or through an XML configuration file.

In Fig. 4, the proxy server is shown running on the same node as the experiment applications, but it can be hosted on a separate node if the situation requires. However, for the two use-cases we presented in the preceding sections, running the proxy server on the experiment node is exactly what we want, because it makes the measurement collection independent of the network. In both of those cases, the proxy server is configured to buffer all measurement data in memory until the end of the experiment. At the end of the experiment, the experimenter instructs the proxy server to turn 'ON' the output stream, whereupon the proxy server connects to the upstream full OML server and transmits the stored measurements to it.

The proxy server implementation is simple, as it does not have to process any data on its input stream. Furthermore, since it is a one-way stream, the proxy server can simply store the raw octets from the experiment applications in memory, and then replay them out to the OML server verbatim. It also has the option to write the measurement data to file to provide a backup and limit its memory usage.

3.4 Results

We now provide some experimental data that demonstrate the management of the disconnection. This experiment was originally presented at the 4th GENI Engineering Conference [4]. In this experiment, two mobile nodes exchange UDP traffic over a WiFi ad-hoc network. One node is stationary, and the other moves along a short circuit as illustrated in Fig. 5. Both nodes run an OML-enhanced version of `iperf` [2]. In addition, the roaming node also runs a GPS application collecting location information. The UDP traffic and GPS measurements are collected using the OML framework.

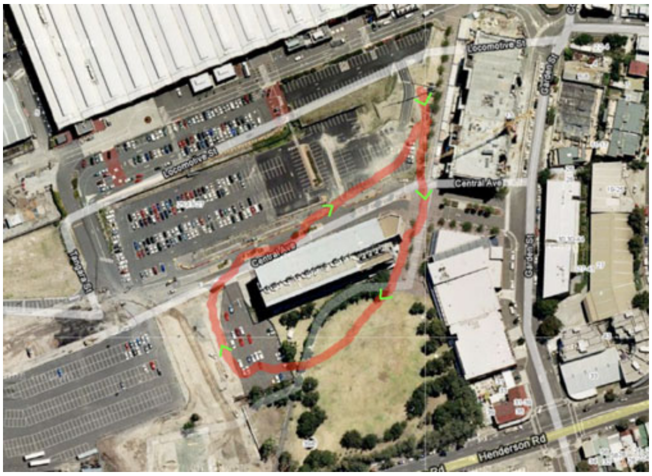

Fig. 5. Path of the roaming node from GPS data (aerial photo from Google Maps [8])

To demonstrate the proposed proxy scheme, the UDP traffic measurements are collected via OML over the WiFi network, which will become unavailable as the roaming node moves away from the static one. However, to allow real-time visualization during a live demonstration, a second permanent WiMax network is used to continuously collect GPS information. The duplication of the GPS measurement stream is completely transparent to the application.

We have run this experiment numerous times, with a typical result shown in Fig. 7. In this figure, the x-axis represents time; the OML server automatically time stamps all samples received throughout the experiment. The distance was computed based on the GPS localisation, and the bandwidth was computed using fixed windows of one second on the receiver side.

In Fig. 7, we can observe the correlation between the distance and the achieved bandwidth. This can be explained by the fact that during this time the two nodes are disconnected. During this disconnection period, all the measurements are stored by the proxy. Once the roaming node gets closer to the static node and to the OML server, the proxy is set to resume sending the buffered measurements to

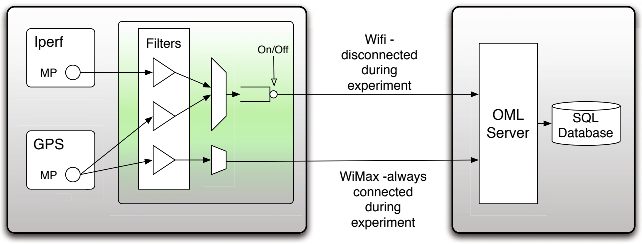
Iperf
MP
GPS
MP
Filters
On/Off
Wifi - disconnected during experiment
WiMax -always connected during experiment
OML Server
SQL Database

Fig. 6. OML Internal Configuration of the two Nodes and Server

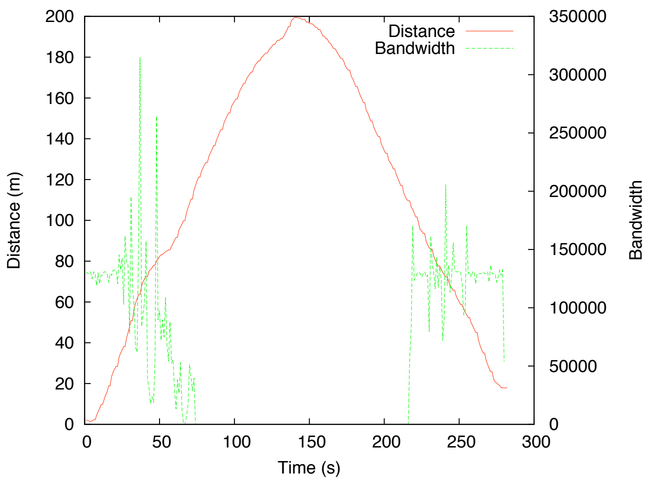

Fig. 7. Packet Loss Rate and Distance in function of the time

the OML server. Another advantage of OML in this experiment is the automatic time stamping, which allows the time evolution of different quantities to be put in perspective.

4 Further Extensions

Once the measurement architecture contains a processing element beyond the client applications on the experiment node, it is natural to ask what further sorts of processing could be done on the node itself. This line of thought takes the measurement architecture away from a basic client/server architecture. In the following sections, we describe two architectural enhancements we have developed as a result of discussions with users of OML whose needs were not met by the standard existing OML facilities.

4.1 Hierarchical Measurement Collection

In a measurement application that generates large volumes of data, it may be either too expensive or impracticable to store every sample collected. This may

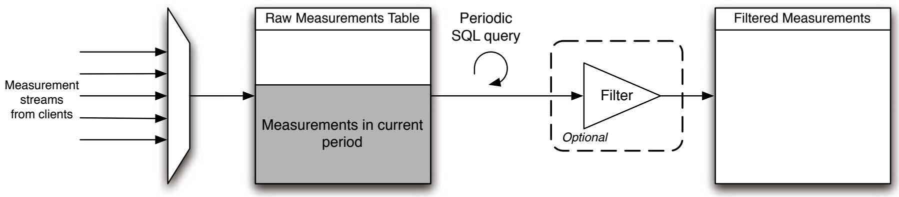

Fig. 8. Server architecture for hierarchical measurement collection

not be a problem if the utility of the collected samples decreases over time. For instance, in a server load-monitoring application, high resolution measurements for the last hour might be interesting and useful, but the same for a period six months ago might be useless. An average over a coarser timescale might suffice for historical records of that age or older, so that full-rate, high resolution data does not need to be stored in its entirety.

The basic architecture described in the previous section does not permit this type of volume-thinning. To support it, we augment the server architecture as shown in Fig. 8. The server includes a query mechanism that periodically executes an SQL query on any measurement table. The results of the query are appended to another table. The query can, e.g., compute an average of numeric quantities stored in the table, then cull the rows that were averaged, to prevent the table size from increasing beyond a set bound. This gives a compromise between the storage requirements and availability of high resolution data, and is similar to stream databases [7] and round-robin databases [18].

We can extend this idea in three ways. First, we can compose a hierarchy of measurement timescales to suit the requirements of various different users of the collected data. For instance, we could store high resolution measurements for the last ten minutes, medium resolution for the last hour, and low resolution for the last six months.

Secondly, the destination table for the periodic query results does not have to be hosted on the same machine. We can instead transmit the aggregate measurements to another OML server, using the same measurement protocol that the client applications use. The hierarchy of timescales is then reflected in the hierarchy of collection servers. This flexibility can be put to several uses, such as server load management or multi-site redundant storage, but we will describe what we think are some of the most interesting ones in the following subsection.

Thirdly, if SQL does not provide enough expressive power to compute the desired summary metric for a particular measurement table, we can augment the server with a configurable filtering mechanism, identical to the one available to the client applications in OML. This is the filter block shown in Fig. 8.

We can use the measurement stream architecture in the client application to implement very flexible measurement collection configurations. For instance, we can create two streams from the same MP and send one stream to a local high-resolution server on the experiment node and the other to a lower-resolution server elsewhere on the network.

4.2 Context-Driven Experiment Steering

We now have an architecture with an OML server that can do periodic computations on the received data, and send results of the queries to an upstream server. The local OML server can be running on the experiment node itself. We could also use this mechanism to periodically check for particular events that might be reflected in the measurement data. If we add a feedback mechanism, then we can use this event detection facility to modify the course of an experiment while it is running. We call this capability *context-driven experiment steering*.

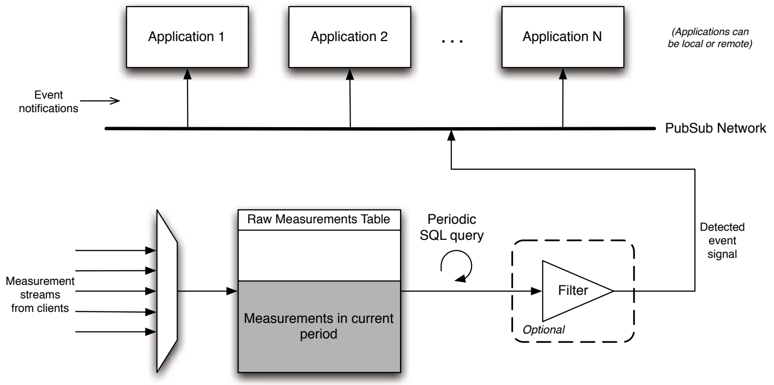

Fig. 9. Context-driven experiment steering. Measurements are used to detect conditions that trigger pre-defined behaviour in the client applications, using a publisher/subscriber notification mechanism.

Fig. 9 illustrates this architecture. The feedback loop can be contained entirely within one node if the OML server is running on the same node as the applications. More generally, the feedback mechanism is distributed, so that a remote OML server in the measurement hierarchy can give steering feedback to one or many nodes participating in the experiment.

One example of such an application would be to detect when the quality of service to a node becomes too degraded, and remove the node from the experiment. Another approach might be to only start some of the experiment applications after a condition has been met, for example, in a peer-to-peer download experiment, only starting the peers once the seeder has downloaded enough of the file from a central server. The idea of trip lines, where a mobile node crossing from one geographic region to another causes some action to be performed, is a third example of experiment context that can be implemented using our measurement architecture [9].

We are considering a publisher-subscriber framework to implement the feedback mechanism, such as Dbus or XMPP. The client applications must subscribe to and listen for particular events, and the server must have a mechanism for specifying what events are published and how they are detected. This could use a combination of SQL queries and filtering, with the final stage of the filter being a predicate function. The management framework can also play a part in the feedback mechanism, e.g., for starting and stopping experiment applications.

This extension opens up a range of new possibilities for experiment design and measurement applications.

4.3 Context-Driven Measurement

If we have a feedback mechanism that detects events in the measured environment, why not then allow detected events to influence the measurement process itself? This is the third extension. We reflect the feedback mechanism back onto the OML server, so that we can tailor the measurement strategy to the current conditions. For instance, suppose we are only interested in low-resolution measurements of a particular quantity most of the time, but when an alarm occurs, we want to start recording high-resolution measurements. In this case we can add a second query/filter path to Fig. 8, and switch between them based on a feedback signal, as in Fig. 10.

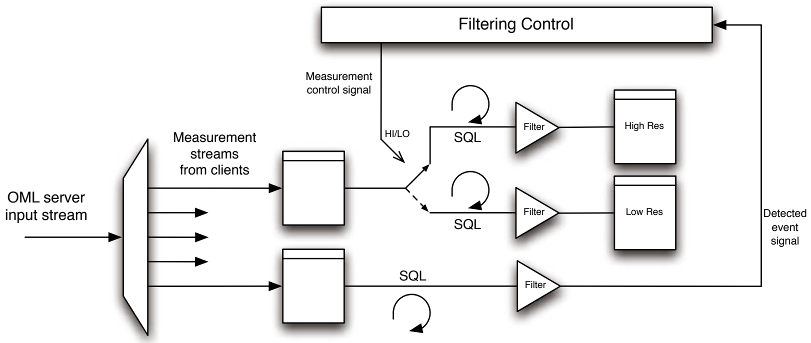

Fig. 10. Context-driven measurement. A measured event feedback signal is used to influence the measurement capture process itself.

With this third extension, we have outlined our architecture for distributed measurement collection. We now go on to compare our architecture against other work in the field.

5 Related Work

There are various existing measurement frameworks, some of them open source and some of them proprietary. Some of them are geared towards network monitoring for system administration, whereas others are more useful in research contexts.

CoMo (*Co*ntinuous *Mo*nitoring) [11,12] is a network measurement system based around measurement of packet flows. It has core processes that are linked in stages, namely packet capture, export, storage, and query. These processes capture, filter, measure, and store properties of packets and packet traces. The core processes are linked by user-defined modules that are used to customize the measurement system and implement filtering functions. The query process provides an interface for distributed users to run queries on the captured packet traces.

The core processes are designed for speed and efficiency and are in charge of data movement operations. One of the overriding goals in CoMo is to make querying as efficient as possible[11], because CoMo can operate on very large data sets ($\sim$ 1 TB). CoMo modules essentially pre-compute the answers to queries, speculatively. Queries identify traffic with specific properties, such as finding flows that match certain criteria. As the CoMo system itself, including captured packet storage, can be distributed across the network, CoMo introduces the notion of "distributed indices" to speed up the process of finding the locations of packet traces of interest to a query.

CoMo is a highly tailored tool designed for efficient packet trace capture and analysis. OML, by contrast, is a generic framework that can instrument the whole software stack, and take input from any sensor with a software interface. One of CoMo's great strengths is its query architecture, and OML does not include a comparable mechanism, relying instead on its SQL database storage substrate to provide the experimenter with a query interface to her data.

One could also compare OML to network adminstration monitoring tools such as SNMP (covered by numerous IETF RFC's, starting with RFC 1155, 1157, for instance). SNMP has a high overhead compared to OML. Monitoring in SNMP is based around OID's—object identifiers—that identify measurement items of interest and are essentially numeric and not human-readable. A central management information base (MIB) must be maintained to map OID's to human-readable strings. This is at odds with the needs of research, which is by nature dynamic and often not centralized enough to permit the maintenance of an MIB, which also adds unnecessary cost. In OML, a user wanting to measure a new quantity simply defines a new measurement point in the relevant client application and configures the filters for his experiment run to filter it into the database. There is no central organization needed. From our survey, there do not appear to be suitable open source implementations that could be easily adapted to the needs of research.

Of all the measurement architectures we surveyed, MINER [1,5] appears to be the closest to our architecture. MINER is not available as open source software, but [5] describes its architecture. MINER is Java-based and comprises a measurement architecture as well as elements of what we refer to as the management framework. A *client library* provides an API for defining and running experiments, which consist of invocations of *tools*. A *core* component is the server component of the infrastructure and the mediator between the client library and the measured network. A *tool proxy* component acts as a mediator between the core component and the MINER tools. A tool proxy executes a scenario request on a network node, starts the requested MINER tools, and then grooms the measurement results back to the core.

The MINER tools are Java components that may provide measurement results directly, or may be wrappers around external libraries or applications that do the actual measurements. MINER tools can be defined by the user.

Our management framework (OMF [16]) is decoupled from the measurement aspect of experimentation. This makes each component more generic and flexible. The MINER approach of providing a wrapper interface for existing tools is a great idea. The main method to instrument existing applications with OML is to directly modify their source code (e.g., `iperf` in Section 3.4). When these sources are not available, it is easy to develop a short program to process the application's outputs and collect the resulting measurements using OML.

Emulab [19] is a large network emulator based on a set of computers that can be configured into various topologies through emulated network links. Many experimenters currently use Emulab testbeds to evaluate their research schemes. It allows them to monitor and capture network traffic (packet headers or full payload) on links and LANs within their experimental topologies. The capture points, equivalent to OML measurement points, are either on the resource that emulates a link, or on end-point resources. In both cases, the captured data are stored as a local file on that resource. To analyse the experimental results, the user has to retrieve the resulting file from all the used resources at the end of an experiment. This simple scheme is limited to the measurement of only network traffic, and does not allow the monitoring of any experiment's contextual variables (e.g., node location) or application integrated data (e.g., download/upload statistics for a peer-to-peer application).

PlanetLab [15] is a global research platform based on more than 1000 distributed computers, which are hosted by independent organisations. It is the primary large-scale testbed used for experimental overlay and service oriented systems (e.g., distributed storage, peer-to-peer content distribution). Multiple services are currently deployed on Planetlab, which provide users with measurements of their experimental slices and the whole testbed, such as CoMon [13], or PlanetFlow [10]. CoMon provides different statistics (e.g., memory, disk usage) at a node or a slice granularity. However, it does not support collection of application or experiment generated measurements. CoMon uses a client/server design like the basic OML architecture. The processed measurements are made available to the entire experimenter community via a distributed content delivery system. PlanetFlow also uses a client/server scheme. On each node, a client entity captures all outgoing packet headers, then aggregates and classifies them into flows. This process is akin to the OML client filtering. However PlanetFlow does not provide any other client-side flow processing. These flow measurements are centrally collected in a MySQL database accessible via a Web interface.

6 Conclusions

In this article we presented extensions to the versatile measurement library OML. The new features that we presented allow the experimenter to extend the range of possible measurements. In particular, we detailed the transparent integration of a proxy server on the experiment node allowing measurements in a disconnected environment. This solution has been made possible by the addition of a measurement proxy server on the mobile node within the existing measurement framework. The first goal of this proxy is to buffer the measurement stream without losing any information. We identified two main fields of application for this measurement feature, a disconnected experiment and a shared control and experiment network. We demonstrate the benefit of this new feature in the context of a simple disconnected experiment during the 4^{th} GEC and presented the results in this article. Finally we extended this architecture with hierarchical measurement collection and server-side filtering, which allows us greater control over the measurement collection process, and with a feedback mechanism that allows us to steer both experiments and the measurement process itself while the experiment is running, based on the current measured context.

Acknowledgements

This work was achieved in the context of the Onelab2 projects funded by the E.U. 7^{th} Framework Program, and the GENI (Global Environment for Network Innovations) initiative funded by the U.S. National Science Foundation.

References

1. MINER: The Measurement Infrastructure for Network Research, `http://miner.salzburgresearch.at/index.php`
2. NLANR/DAST: Iperf - the TCP/UDP bandwidth measurement tool, `http://dast.nlanr.net/Projects/Iperf/`
3. OML: The OMF Measurement Library, `http://omf.mytestbed.net/projects/show/oml`
4. The 4th GENI engineering conference (March 2009)
5. Brandauer, C., Fichtel, T.: MINER – a measurement infrastructure for network research. In: International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities, pp. 1–9. IEEE Computer Society, Los Alamitos (2009)
6. Claise, B.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard) (January 2008)
7. Franklin, M.J., Krishnamurthy, S., Conway, N., Li, A., Russakovsky, A., Thombre, N.: Continuous analytics: Rethinking query processing in a network-effect world. In: Fourth Biennial Conference on Innovative Data Systems Research (CIDR 2009), Asilomar, CA (January 2009)
8. Google. Google maps, `http://maps.google.com/`
9. Hoh, B., Gruteser, M., Herring, R., Ban, J., Work, D., Herrera, J.-C., Bayen, A., Annavaram, M., Jaconbson, Q.: Virtual trip lines for distributed privacy-preserving traffic monitoring. In: ACM MobiSys (2008)
10. Huang, M., Bavier, A., Peterson, L.: PlanetFlow: Maintaining Accountability for Network Services. Operating Systems Review 40(1) (2006)

11. Ianaccone, G., Diot, C., McAuley, D., Moore, A., Pratt, I., Rizzo, L.: The CoMo white paper. Technical Report IRC-TR-04-17, Intel Research Cambridge (September 2004)
12. iANNACCONE, G.: CoMo: An open infrastructure for network monitoring—research agenda. Technical report, Intel Research Cambridge (February 2005)
13. Park, K., Pai, V.S.: CoMon: A Mostly-Scalable Monitoring System for Planet-Lab. ACM SIGOPS Operating Systems Review (2006)
14. Petander, H.: Energy aware network selection using traffic estimation. In: Proc. of MITCN 2009 Workshop in ACM Mobicom Conference (September 2009)
15. PlanetLab Consortium. Planetlab: An open platform for developing, deploying, and accessing planetary-scale services, `http://www.planet-lab.org/`
16. Rakotoarivelo, T., Ott, M., Seskar, I., Jourjon, G.: OMF: a control and management framework for networking testbeds. In: SOSP Workshop on Real Overlays and Distributed Systems (ROADS 2009), Big Sky, USA, p. 6 (October 2009)
17. Raychaudhuri, D., et al.: Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols. In: Proc. IEEE Wireless Communications and Networking Conference, WCNC (2005)
18. Oetiker, T.: RRDtool, `http://oss.oetiker.ch/rrdtool/`
19. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. SIGOPS Oper. Syst. Rev. 36(SI), 255–270 (2002)