A Testbed for Agent-Based Multi-Purpose Extensible Active Measurement

Marat Zhanikeev*, Yoshiaki Tanaka*†

* Global Information and Telecommunication Institute, Waseda University
1-3-10 Nishi-Waseda, Shinjuku-ku, Tokyo, 169-0051 Japan
Email: maratishe@asagi.waseda.jp

† Advanced Research Institute for Science and Engineering, Waseda University
17 Kikuicho, Shinjuku-ku, Tokyo, 162-0044 Japan

Abstract— **Requirements for a measurement platform nowadays have advanced to the level at which a number of different in principle measurement techniques have to be performed simultaneously. Quite common are hybrids of passive data collections and event-driven active measurements. This calls for a highly extensible measurement platform, in which design of the probe and parameters of probing could be accessible in realtime while the measurement itself is being performed. In this paper, we introduce a testbed application that was developed with the initial requirement to be fit for a number of different applications, such as bandwidth measurement, performance prediction, tomography, and others, without a loss of performance characteristics.**

I. Introduction

In recent years, a number of new-generation measurement platforms were brought into existence. The need for a new generation of tools and platforms came from the fact that the previous generation of tools would often offer only a single measurement functionality. For example, a tool would only measure bulk transfer capacity or available bandwidth. Therefore, when a set of different measurement targets was required for overall network performance analysis, one would have to use a number of different tools from different providers.

The main objective in the new generation is flexibility of measurement objectives, higher precision and, generally, a higher level of intelligence on the part of the measurement tool itself.

Besides, some measurement objectives by definition require a complex set of measurement activities. Those are available bandwidth measurement [1], performance anomaly analysis [2], recently emerged tomography, i.e. network topology discovery through active measurement, and others.

On the top of the development of measurement technology under growing research interest, there is another incentive which is a number of new high-performance networks, such as Internet2 and GEANT in Europe, or Japan Gigabit Network, which raised the issue of online performance monitoring through active measurement. A number of projects came to life as a product of management experience. One example is NCC RIPE in Europe [3], which is involved in management of GEANT.

Other projects that are currently active in the field are Bandwidth of the World project [4], skitter [5], SCAMPI [6], and Surveyor [7].

However, all of the mentioned projects exhibit a low level of flexibility of their measurement frameworks. Some of them offer the ability to upload user-defined code for special probing, however, the internals of the platform itself are mainly hidden from user.

In this paper, we present a testbed for a distributed extensible platform of end-to-end probing. It employs client-server paradigm and round-trip probing, which allows us not to use GPS synchronization between nodes. One-way measurements with proper synchronization can also fit into the proposed framework, but are not discussed in this paper. The proposed platform allows extensions in probe design, probing parameters, measurement data collection management and flexible result processing functionality. Highly extensible prototype allows to perform online analysis of measurement results, generate events, and make decisions based on these events. The closest to the present research is currently being under development ETOMIC project [8].

Section 2 contains detailed description of the proposed measurement platform, while all the extensibility advantages are additionally described in Section 3. In Section 4, we display several case studies with different targets

1-4244-0106-2/06/$20.00 ©2006 IEEE

and probing procedures to display advantages of chosen design. Some additional considerations can be found in Section 5, which is followed by a conclusion in Section 6.

II. Agent-Based Testbed Design

The proposed platform implements a fully distributed design which is based on active agents of two types: servers and clients. Clients are the main part of measurement functionality, as they require detailed configuration files and perform all online calculations, while servers only respond to probing packets by sending UDP acknowledgments back to the source. As was mentioned earlier, unlike many existing projects, the proposed platform does not depend on precise time synchronization. Such a synchronization is normally very costly to implement and requires a GPS receiver and a sophisticated interface card for each node. This imposes limitations upon flexibility of measurement topology and its size.

On the other hand, measurement projects have already crossed borders among ISPs and are already used on paths that span multiple domains. Such a scale of measurement makes it hard to provide synchronization with precision required for active measurements. As of today, whenever a probing tool needs to scale, it has to abstain from tightly coupled synchronization mechanisms.

A. *Measurement Topology Initial Setup*

Considering the issue of scalability, registration and call-back paradigm of measurement topology creation was adopted by the proposed platform. The initial setup procedure for a single server-client pair is displayed in Fig.1. Both probing server and client initially register their IP addresses with the network operating center (NOC). NOC collects registration messages and creates a measurement topology based on user preferences. Based on these preferences, all servers and clients in the network are configured with detailed information on their further operation.

Action diagram for this process is depicted in Fig.2. Setup messages that are distributed to clients and servers in the network also contain description and conditions for generation of alarms. After NOC finishes the setup process, it goes into idle state, at which it is ready to accept alarms from clients in measurement topology.

As event-driven decisions is an important part of the proposed platform, alarms that are received from clients are processed immediately. As NOC is in the idle state most of the time, prompt processing of alarms can be easily provided. Decisions that NOC can make in realtime include change of overall probing topology, reconfiguration of client/server pairs, creation of new probing paths, etc.

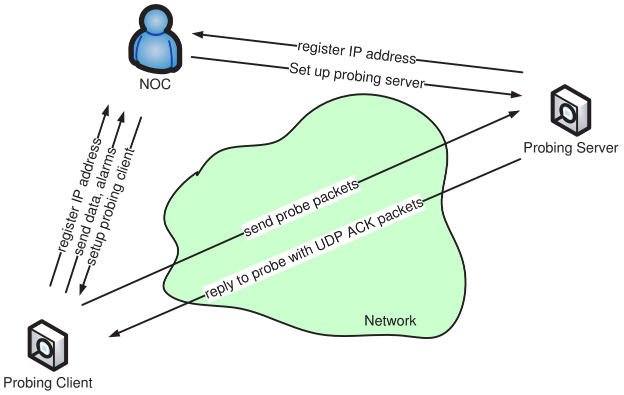


Fig. 1. Actions taken for the initial setup of the measurement network.

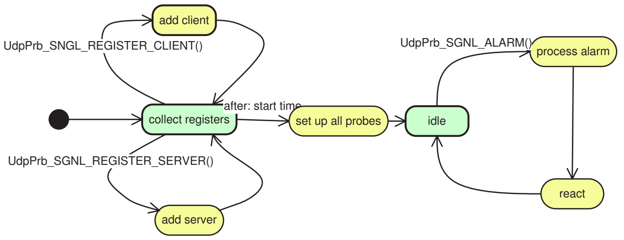


Fig. 2. Action diagram for the measurement topology setup.

B. *Probing Client*

The most important part of the proposed platform is the client, module diagram of which is depicted in Fig.3. Gray modules denote areas that are implemented by default in the operating system, and green modules are developed by us.

Probing client functionality in Fig.3 is implemented in three main modules : **UDP client**, **Generator**, and **Manager**.

Manager is responsible for all communications with NOC and transit of internally generated alarms to the outside. Its functionality is quite simple and is limited to learning of node's own IP address, registering it along with node description to NOC, and entering idle state, which is interrupted only by arrival of an alarm, which would be relayed to NOC. Setup file received from NOC is handed over to **Generator** for handling.

Generator is much more complicated and is a complex module. The complexity is on the part of the need

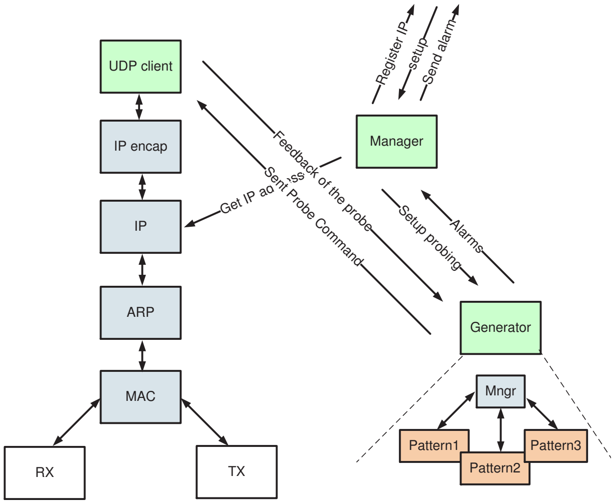


Fig. 3. Module structure of probing client.

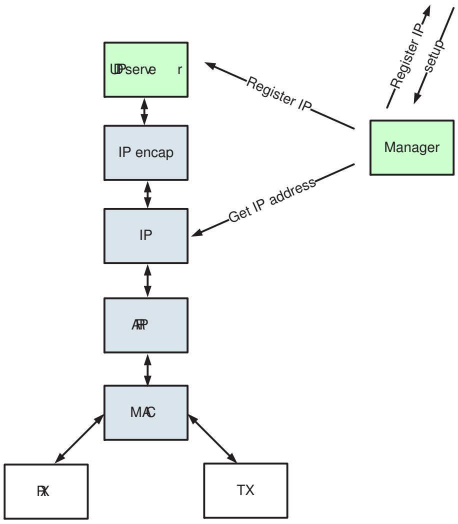


Fig. 4. Module structure of probing server.

to spawn a separate thread for each probing pattern. **Generator** also parses configuration files and creates all required resources as per the description of each probing pattern. It creates and manages result data pools and is responsible for garbage, which is left after a process has completed its job. Apart from spawning a thread for the actual probing, **Generator** also contains a pool of custom data processors, such as histogram, kernel density, variance, etc., which implement a certain statistics processing mechanism and can be linked to a data pool. Alarms can be generated by the spawned probing thread or by processing thread, which are not necessarily in sync with each other.

UDP client also has a simple structure, which communicates directly with spawn **Generator**'s threads by passing messages. The description of each probe is received from a thread under **Generator**, and is parsed, setting up interrupts for each packet in the probe. For each packet, an interrupt should wake it and make it transmit another packet. **UDP client** also contains listeners, which listen to a port defined in the probing pattern from NOC, and notify **UDP client** upon arrival of a packet on the specified port. The packet is processed and when all ACK packets for each probe are collected, all timings are passed to the thread under **Generator** that initially scheduled probe for transmission.

C. Probing Server

Internal structure of a server is much simpler than that of the client and is depicted in Fig.6. It lacks Generator module, which was present in the client.

Manager in the server performs the same task as the **Manager** in the client. It also registers its internal setup to NOC and receives setup configuration file from it. However, in the server **Manager** does a little extra work. It parses setup file and identifies which remote IP addresses should be considered as source of probing and, therefore, should be replied with UDP acknowledgment packets. Remote IP addresses and ports that are not registered by NOC, are ignored.

UDP server operates slightly differently from **UDP client** and works on its own. Once all remote clients are registered by NOC, all measurement traffic from them can be served by **UDP server** in a standalone manner.

According to the proposed design, no alarms are generated in the server, and no processing of data is performed. Probing servers are simply bouncers for measurement traffic with ability to distinguish measurement traffic from other traffic in the network.

D. Probing Setup and Operation Datatypes

Fig.5 contains data structures used between NOC and client and by client for internal interfaces. The main structure in the diagram is `Probing_Pattern`, which contains all the data required by both the client and the server, while a part of this structure it dedicated for storing descriptions of alarms. We can also see from the diagram that both `ClientChild_Setup` and `Client_Setup` are included in `Probing_Pattern`, but their contents are different. `Client_Setup` is used by **Generator** to prepare all resources and spawn a probing thread, which name is identified by `child_process_name`, which is then configured by a set of parameters contained in `ClientChild_Setup`.

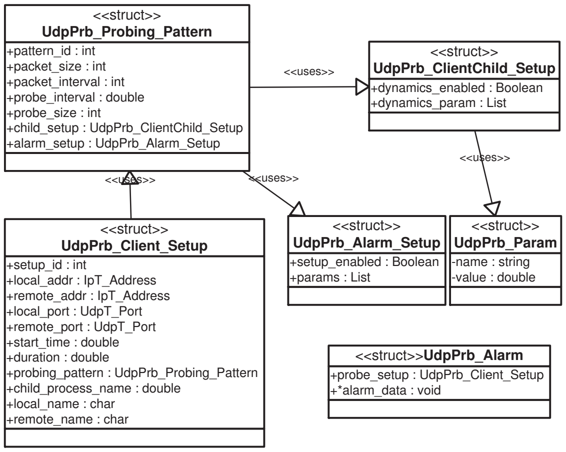


Fig. 5. Data structures on the client-NOC side.

Server data structure in Fig.6 is much simpler and contains only details of the source and destination ends of the measurement path. The destination address and port are verified against those of the server and source address and port are registered inside of the server after which probing traffic from this location is expected.

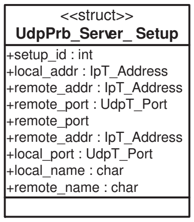


Fig. 6. Data structures on server-NOC side.

III. Extensibility of Probe and Result Analysis Design

A. Data Pools and Processing

Internal design of the probing client is shown in Fig.7 with the stress to its extensibility features. **Probe Manager** can spawn any of the available probing threads, which are located in the pool of **Probe Clients**. The pool can contain implementations of any number of tools, which do not interfere with each other as one traffic pattern can specify only one measurement logic. Parallel threads with two different probing methods are allowed and take place in the case studies in this paper, but there is still no interference as probing results are collected in a separate **Results/State Pool** block. **Results/State Pool** creates separate storage structures for each registered probing thread.

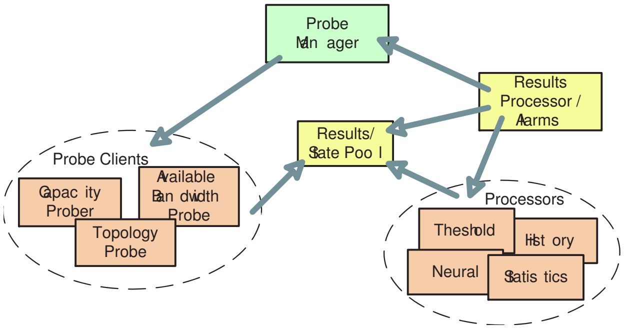


Fig. 7. Flexibility and extensible design of probing client.

B. Custom Results Processing Tools

It is obvious from Fig.7 that **Results Processor/Alarms** thread is the most important in the client. It can both access **Results** directly in case a simple result polling is required in setup, or it can spawn a thread from a pool of **Processors**, that would perform a statistical processing task on the results. The pool of **Processors** contains implementations of various statistical methods, some of which are displayed in the diagram, such as threshold rule, history processing, neural learning, and statistic tools. Some examples of statistic processors will be mentioned in the case studies.

Any **Processor** has access to the data pool of a particular probing thread, and can also create new pools within the space of a particular probing thread. These newly created pools may be used for communication between processors and probing threads, when input by a processor may affect the behavior of a probing thread.

Specification of **Alarms** are given in the initial setup from NOC. They are generated by **Results Processor** block, and contain basic rules. The rules that we use are threshold, basic more/less/equal check performed on certain variables in the **Results Pool**, etc. If a certain condition is met, an alarm is created based on the metric that caused the alarm, time of alarm occurrence, and its ID, after which the alarm is transmitted over to NOC for processing.

IV. Case Studies

A. Test Network Setup

For all our case studies, we used the network among campuses within Waseda University, some of which are over 150km away from each other. The simplified diagram of the network topology is shown in Fig.8.

Dashed lines in the diagram represent hidden topology, which means that there are more routers on those lines, that were skipped for simplicity of presentation.

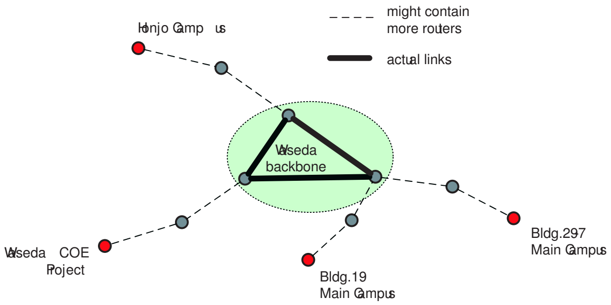


Fig. 8. Inter-campus network used for case studies.

Location of probes in Fig.8 are marked with red dots. We have four separate locations used in case studies, and the total of six probes, as some of probes were installed in pairs at one location to be able to do measurement in parallel with different setup parameters but identical measurement environment. The results obtained in such a fashion are easy to compare and analyze based on the comparison of results.

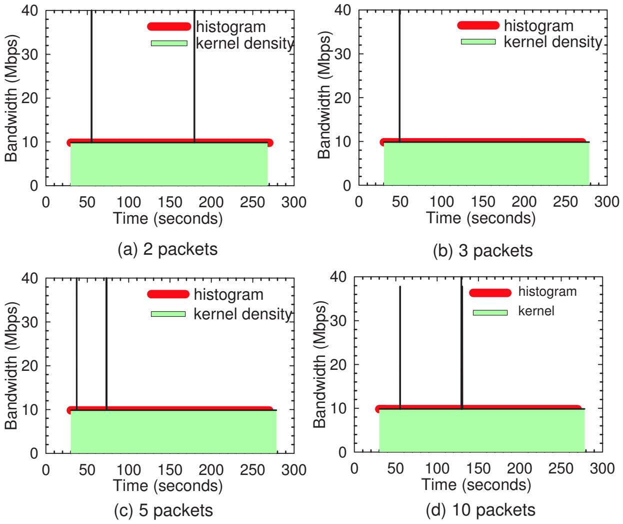
Each probe is a PC with Linux operating system installed. We used 2.4.18 kernel, that, by default, allows the lowest interrupt granularity of 10ms in the user space. We had an option of moving the application into the kernel space, but, instead, used a patch in the kernel that improved interrupt performance to a few microseconds. The patch enables APIC hardware timer in the kernel space, thus, providing another interrupt stream with virtually hard realtime characteristics. Such patching is commonly done for realtime-related research and for audio-video processing, which is also very harsh on time constraints.

NOC, probing client and probing server are written in C++ and compiled with GCC version 2.95, which is a little old, but fits very well into the applied kernel, for which we managed to make the APIC patch work.

All the setup of measurement topology was done from a single location in the same network, but location is not shown as it is not related to measurement topology. Most of Waseda Network is 100Mbps with 155Mbps outside link, and a few locations inside of the university, which are connected over 10Mpbs links. We deliberately used the route through one of such links in order to perform the case study on bulk capacity measurement.

B. Bulk Capacity Estimation

The first case study in this paper is done for bulk capacity measurements and is the easiest in terms of setup. The measurement was performed from Waseda COE to Bldg.19 Main Campus as per Fig.8.



(a) 2 packets

(b) 3 packets

(c) 5 packets

(d) 10 packets

Fig. 9. Results of simultaneous probing performed from two neighboring probes to a single destination with two different processing methods and several probe lengths.

Two probes were installed at the source with identical setup with exception for measurement start time, which was shifted for half-a-second for one of probes, and statistics processing method used in result data pool to calculate the estimate.

Measurement itself was performed using basic packet-pair property as described at length in [9]. The reason why two probes were used with different estimation methods is that calculating an estimate online is the main part of any bulk capacity measurement method. Most currently existing tools are using some sort of statistical pre-filtering or processing that is meant to filter out unwanted samples in measurement data. It has long been established that measurements in the Internet offer multi-modal results that require sophisticated statistical processing to separate the modes and find the correct one that corresponds to bulk capacity.

In Fig.9, the results are displayed for different lengths of the probe. A probe of each length was transmitted each second, separated by quarter of second from the probe of another length. In such a fashion we were able to measure roughly unchanged network conditions with probes of different lengths.

Results in Fig.9 is a very good exhibit of prevalence of more complex statistics processing over simple methods, such as histogram analysis, that is used in this case study versus kernel density, which is also used in the Nettimer tool in [9]. From the results, the probe with kernel density processing was able to identify spikes in data, which stand for the period of minor congestion in the network, which jammed the probe and minimized space between the packets, thus depriving the sample from the basis property of packet-pair measurement. Therefore, on the plot this is represented as a very high spike at near 50 and 180 seconds. As bulk capacity is calculated as the size of the packets to the space between them obtained in the bottleneck, i.e. $B = S/T$, the nature of the spike is easily understood.

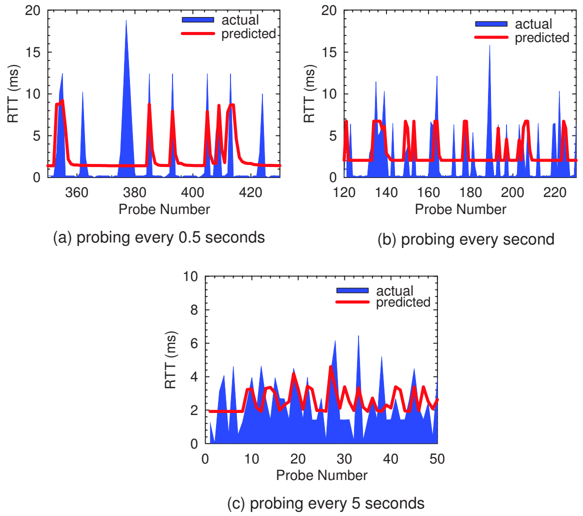
Again, quite naturally, longer trains (5 and 10 packets) are much less stable even on broadband networks, and congestion spikes in these cases are stronger and more vivid. Longer trains are much more susceptible to even minor fluctuations in probing traffic, which is proven by the results.

C. Online Neural Performance Predictions

This case study required a more complicated setup procedure, while the measurement itself became simpler. The case study addresses the problem of neural predictions of network performance. Predictions were made based on simple RTT measurements from *Waseda COE* and *Honjo Campus* on network diagram in Fig.8, which is not only the longest available path within Waseda (9 hops), but also is partially optical, which offers higher end-to-end delays and more interesting measurement material for predictions.

Although the probe structure in pattern setup is simple and specifies regular single packets sent to a single location, the neural prediction part is more complex. Totally 500 probes were transmitted, and when the 500th result entered the pool, it triggered Backprop algorithm implementation of neural network, that processed a setup number of first samples as learning material, after which it would predict the final, hidden part. When the prediction is made, results processor block of the client comprises actual and predicted results and pass the data over to NOC for display.

Results of this case study are displayed in Fig.10. Three probing intervals were used in different probing sessions, from half-a-second to 5 seconds in the last case. Predictions on plots are represented as red thick lines over the area-filled blue-colored actual measurement results.



(a) probing every 0.5 seconds

(b) probing every second

(c) probing every 5 seconds

Fig. 10. Online neural network (Backprop) predictions based on a single-packet RTT measurement results (zoomed view into the most interesting match regions).

Visual analysis of results brings forth a conclusion that higher-granularity measurements should be preferred over sparse probing if the main objective is the precision of predictions. In Fig.10, measurement each harf-a-second offers much higher correct prediction hits than any other probing frequency. The worst case at 5 seconds almost never is correct in its predictions, which means that backprop algorithm failed to find any worthy patterns in measurement data.

It should be noted that this case study is completely different from bulk capacity measurements in the previous case, however, it could be handled by the same probe client, which operation was defined by a different pattern setup. The probing thread within client itself was not aware of neural calculations, and was simply collecting results from single-packet probes, while a different block, which was responsible for neural calculations, was not aware of probing client, and was only concerned with the samples collected in the pool.

D. Unconventional Probe Structure

While the previous case study stressed on complicated data processing techniques, this case study stresses on probe structure. As was mentioned earlier, some methods may require unconventional probe structures for various reasons. To start with, it could be the need to learn of dependence of interference with cross-traffic on packet size, parallel collection of results from two separate parts of the same probe, etc. As two vivid examples of this consideration, we offer two separate tests for different-size packet pairs in Fig.11 and packet size stairs in Fig.12. Opposite to the previous case study, in the present case there is no processing done, and the plotted data is raw.

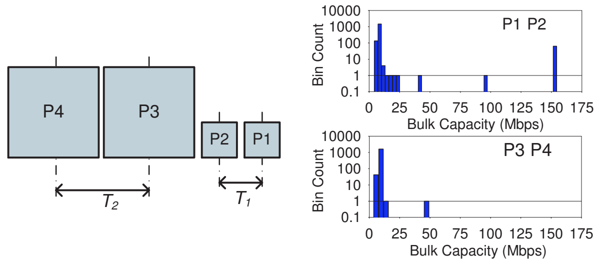


Fig. 11. Study of simultaneous traffic impact on different packet sizes used for packet pairs within a single probe.

A probe structure in Fig.11 could be used in cases when network impact dependence on packet size is desirable. The front of the probe consists of a small packet size pair and the back part contains a pair of large packets (limited by MTU). Small packets by definition traverse the network at a higher speed, therefore, no interference between pairs in the probe is expected unless the network is extremely congested, which is not the case with Waseda University.

From the results in Fig.11, it is obvious that the assumption was true, as distribution of modes in results obtained from the small-sized front of the probe is far richer than that of the other packet pair. The main mode at 10Mbps, nevertheless, is overwhelmingly high, which is natural given the packet-pair probing method. However, the presence of additional modes may help in statistical processing. In fact, there are several researches which use data collected from measurements by various packet sizes to create a pool of modes from which the correct modes are selected based on statistics.

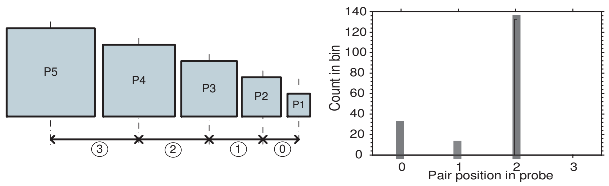


Fig. 12. Study of susceptibility of different packet sizes to cross-traffic interference, which can serve as light indicator to bandwidth utilization.

Second case of unconventional probe structures is displayed in Fig.12. It is a simple staircase of packet sizes comprised within a single probe. The target of this measurement was to identify a pair of different-size probes which suffers the highest interference from traffic at any particular point of time. This idea is very close to self-loaded stream, which means that the probe loads the network to the point at which it creates a small congestion (later packets catch up with the preceding part of the train). Based on this mechanism, such a train might serve as a light indicator of network utilization. In the present case, the network is only mildly utilized, as highest response is registered in P3-P4 pair.

Client setup in the two above cases are extremely simple, as they only have to identify packet sizes in the probe. The design of packet size setup is such that it can both specify a static packet size, or separate packet sizes for each packet the probe.

E. Tomography Measurements

Tomography, or, actually, a small part of it, which is a task to identify whether two paths are shared or completely separate, is the most complicated of the presented case studies. It comprises both complicated probe structure and transmission timings, as well as tricky calculation procedure.

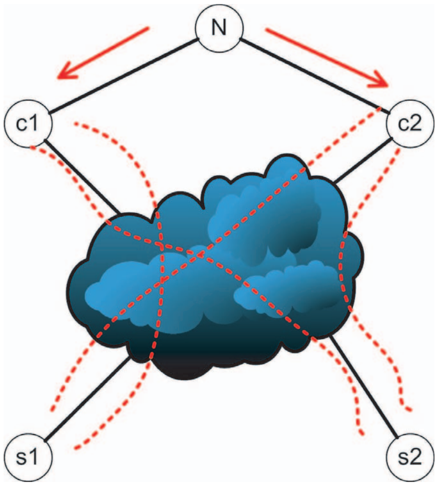


Fig. 13. Probe layout for tomography measurement.

All the details on tomography can be found in [10], which describes the algorithm used in this case study. The probing network topology is displayed in Fig.13. As synchronization between two source probes is required, we created a two-step communication, one is the preparation stage between N and $c1$ with $c2$ and the actual tomography measurements between $c1$ with $c2$ and $s1$ with $s2$.

The measurement algorithm is as follows :

1) $c1$ and $c2$ send probing packets to N.
2) N expects probing packet from two locations, when they arrive, after a certain timeout ACK packets are sent to $c1$ and $c2$, thus, implementing the synchronization mechanism.
3) $c1$ and $c2$, upon reception of a probe from N, trigger tomography measurements, which is when two packets back to back are sent to separate locations, with a small interval between transmission of probe. Random intervals between probes are decided at $c1$, while $c2$ has a fixed interval. This way, probe from $c2$ may be queued both before and after the probe from $c1$, which is the main requirement of tomography.
4) When the ACKs from $s1$ and $s2$ are received back at $c1$ and $c2$ their order may be different from the order in which they were sent. To indicate this, $c1$ and $c2$ would immediately send a 100-byte packet in case the order changed, and 60-byte packet in case the order did not change.
5) N makes digital results from the packet sizes. If 100-byte packet is received, it assigned -1 as the result, and $+1$ otherwise.

The actual implementation of the measurement in probes is done in two separate measurements :

1) Measurement from N to $c1$ and $c2$, when the probing thread sends the measurement probe to each of the servers, which, in this case, are $c1$ and $c2$. When the tomography measurement itself is completed, N receives packets from $c1$ and $c2$, which are treated at ACK packets for the measurement, and are analyzed for their size.
2) The actual tomography measurement, details on which can be found in [10].

Results of the case study are displayed in Fig.14 and consist of both signs as they are received from $c1$ and $c2$, as well as the logical XOR performed on them, which leaves only those samples at which the signs did not match. All the data is plotted on horizontal axis of interval between probes, which could be both negative and positive, which speaks for the order of probes transmitted from $c1$ and $c2$.

As both paths in our case study contain shared segment, the lower plot in Fig.14 is roughly empty. Random spikes in it can be explained by the imperfection of the synchronization mechanism that we are using as well as precision problem, which is natural with very small gaps between probe transmissions. Those spikes are scarce and random, which allows us to disregard them and declare the fact that the topology of both paths is, in fact, shared. Please, refer to [10] to find out the look of tomography result plot in cases when the topology is not shared.

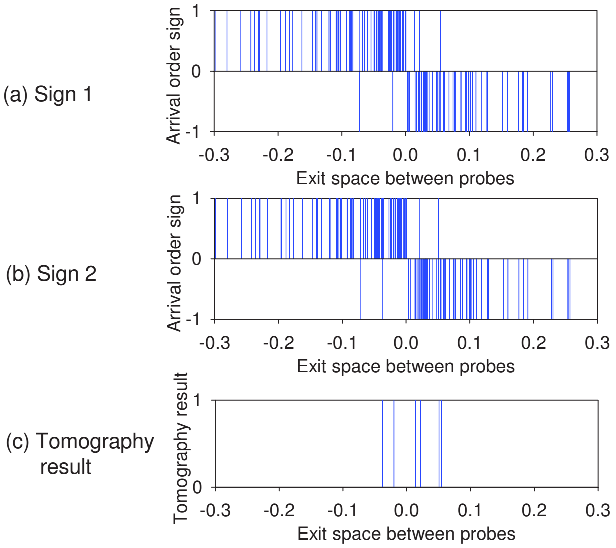


Fig. 14. Results of tomography test, which proves that with the exception of a few spikes at small space between probes, the results prove the fact that topology is in fact shared.

V. Additional Considerations

A. *Self-Loaded Agents Based on Multi-Kernel Machines*

In this paper nodes are either clients or servers, and do not contain both. Of course, it is possible to run a client at the node, which operates as a server, which will make this node both a client and a server, but this is manual work and seriously harms flexibility of measurement topology.

Another approach could be a self-loading of clients and servers into measurement nodes, which would otherwise be simple containers for measurement agents. This would replace the registration/callback communication paradigm used in this paper and, instead, would be completely agent-based, when agents would be uploaded to nodes at runtime.

This, however, in its perfect implementation, requires changes in operating system, or, to be specific, a multi-kernel structure. Projects like this already exist, and one of them is called UML Linux. Currently it is primarily used for research, where it's possible to create an actual network with interfaces, real communication, etc., within a single machine. We are planning to use this concept to be able to upload agents at remote nodes at runtime.

Although multi-kernel functionality is not so much required for self-loaded agents, it is a must for the next important prospective feature, - self-monitoring.

B. Self-Monitoring Features for Parallel Operation

The present paper did not perform monitoring of CPU usage or memory consumption during measurement. It is acceptable for the cases with a single agent per node, which was our case, but it will not work in complicated measurement topologies, where one node could host a few probing clients and servers.

This is another reason for adopting multi-kernel approach, that would allow to move self-monitoring features into the coordination base beyond the scope of kernels. In other words, self-monitoring would exist in the master process, and its indicators could be used to make decisions on whether or not to host another agent. Some of these indicators could be memory and CPU consumption, number of connections created on the same link, and others.

This feature would allow use of a research-based measurement topology by multiple users. Currently, most of such projects allow its registered users to schedule a test, which is then verified subject to time collisions with other tests. A multi-kernel self-monitored approach could be a feasible solution to this problem.

Both of additional features that are mentioned above are scheduled to be implemented in the next stage of the development of the platform.

VI. Conclusions

This paper presented an extensible agent-based platform for active measurement. The main objective of the platform is its extensibility, as well as the ability to use various measurement techniques without major changes in design and code. In order to implement that we adopted client/server based approach, and registration/callback paradigm of distributed computing. Servers and clients register their locations and parameters with NOC, and when, based on this information, measurement topology is decided, NOC configures each client-server pair in the network.

The importance of server and client are not the same. Servers are very simple and are only able to distinguish between cross traffic and measurement probes. Clients, on the other hand, contain all the functionality logic and are quite complicated multi-thread message-passing applications, which are able to host any number of probing threads, running in parallel.

The clear separation of data processing, alarm generation, and probe transmission blocks in the design of the probing client, a fairly high level of functional flexibility was achieved. This was verified on a number of case studies with completely different objectives and complexity of data processing requirements. In each of those case studies, all the work was done at probing clients and NOC received only final results. In the case study for tomography we even managed to use probing clients for two-level probing without changes in probe design, which, again, proves the advantage of such a open functionality approach and separation of functionality within the probing client.

However, there are still issues that are being looked into at the current stage of the project. The two we listed above are self-loading mechanism for distribution of agents over measurement topology, and node-based self-monitoring features.

References

[1] J. Navratil and R. Les.Cottrell, "ABwE :a practical approach to available bandwidth estimation," in *Passive and Active Measurement Workshop*, La Jolla, California, USA, April 2003.

[2] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *2nd ACM SIGCOMM Workshop on Internet Measurement*, New York, NY, USA, November 2002, pp. 71 – 82.

[3] "RIPE NCC measurement project," available at : http://www.ripe.net/ttm.

[4] "Internet end-to-end performance monitoring - bandwidth to the world (IEPM-BW)," Available at : http://www-iepm.slac.stanford.edu/bw/.

[5] "Skipper/skping measurement tool," available at : http://www.caida.org/tools/measurement/skitter/skping.

[6] "SCAMPI measurement project," available at : http://www.ist-scampi.org/overview.html.

[7] "Surveyor project," available at : http://www.advanced.org/csg-ippm.

[8] D. Morato, E. Magana, M. Izal, J. Aracil, F. J. Naranjo, P. Astiz, U. Alonso, I. Csabai, P. Haga, G. Simon, J. Steger, and G. Vattay, "The european traffic observatory measurement infraestructure (ETOMIC): A testbed for universal active and passive measurements," in *TridentCom*, 2005, pp. 283 – 289.

[9] K. Lai and M. Baker, "Nettimer: A tool for measuring bottleneck link bandwidth," in *3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, USA, March 2001, pp. 123–134.

[10] T. Bu, N. Duffield, F. L. Presti, and D. Towsley, "Network tomography on general topologies," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 1, pp. 21–30, 2002.