pMeasure: A Tool for Measuring the Internet

Wenli Liu
School of Computer Science
University of Waterloo
Canada N2L 3G1
Email: w7liu@bbcr.uwaterloo.ca

Raouf Boutaba
School of Computer Science
University of Waterloo
Canada N2L 3G1
Email: rboutaba@bbcr.uwaterloo.ca

James Won-Ki Hong
Computer Science and Engineering
POSTECH
Korea
Email: jwkhong@postech.ac.kr

***Abstract*— This paper describes pMeasure, a tool for measuring large scale networks, particularly the Internet, from end to end. As the Internet grows in size and complexity, fine-grained, precise and timely measurements are needed to ensure its healthiness and ultimately benefit those mission-critical applications deployed on the Internet. The existing measurement infrastructures are unable to satisfy this need due to various reasons. pMeasure, on the other hand, is built on a Peer-to-Peer network substrate, namely Pastry, and can accomplish measurement tasks by utilizing only nodes at the edge of the Internet. The distinguishing characteristics of pMeasure, such as self-organizing and scalability, just to name a few, make it a measurement tool that can satisfy the measurement need for the Internet community efficiently and economically.**

I. Introduction

With decades of evolvement, the Internet has now become an indispensable constituent of human life and mission-critical applications are increasingly deployed on it. The loss resulting from any dysfunction of the Internet, even for a short period of time, would be unaffordable. Fortunately, researchers have started their efforts in investigating the characteristics of the Internet and in devising better network management strategies and traffic engineering approaches. All these efforts require timely, precise and fine-grained measures of the Internet and unfortunately, the existing measurement facilities by no means can satisfy this need due to the following. First of all, no single administrative organization owns the Internet and each manages only a part of it. Because of commercial factors, these organizations usually disallow any direct measurement of their infrastructure and are reluctant to share measurement data with others. Secondly, since the measurement facilities deployed so far are usually expensive and require a large amount of storage due to the high speed of the links monitored, the deployed measurement facilities are few in number and are far from providing fine-grained information. As a consequence, what the network administrators face is mostly an unsynchronized, limited, and partial view of the Internet, which is far from satisfying the need and imposes difficulties, more or less, to the already complicated task.

For these reasons, a Peer-to-Peer (P2P) based measurement system is promising. Like the other P2P applications, this measurement system is able to self-organize into a scalable P2P application that depends solely on computing resources at the edge of the network. Similarly, researchers can join the system by contributing their measurement functionalities from the edge and once in the system, they have the full control over what to collect, when to collect, how to collect and in what granularity. Since the measurement functionalities in the system are synchronized, the collected data are synchronized as well. Srinivasan and Zegura proposed such a measurement system [1] in which each node is responsible for one part of the Internet(Area Of Responsibility) and measurement tasks are carried out by nodes that are responsible for the areas involved. This system is similar to pMeasure in that they are both P2P applications and measurement tasks are carried out by cooperative nodes from the edge of the Internet. However, a number of major issues plague the system in [1]. First of all, the paper is unclear on how the peers are identified and located. More importantly, two query formats are required in [1] while P2P networks can support only one query format. Secondly, AORs are divided into smaller AORs when new peers enter the system and AORs merge to form a larger AOR after peers depart. This division and merging of AORs generates a considerable amount of overhead. Thirdly, [1] estimates the path characteristics between a source IP address and a destination IP address by measuring the path between the node whose AOR contains the source IP address and the node whose AOR contains the destination IP address. Apparently, the error in the estimation will be considerable when there are only a few nodes available in the system. Last but not the least, free riding and security are usually major issues in P2P applications, but they are not considered in [1]. pMeasure, on the other hand, is constructed on top

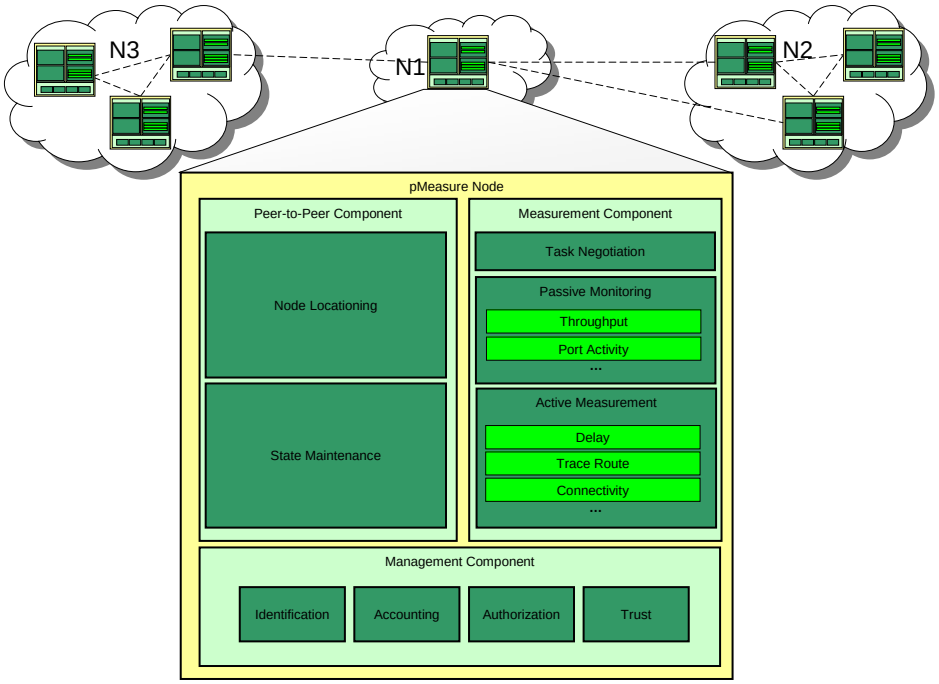

Fig. 1. pMeasure Architecture

of a P2P substrate, namely Pastry [2], and is able to take all the aforementioned issues into consideration, thus offering a feasible and practical solution to the measurement need.

The rest of this paper is organized as follows. The design of pMeasure is presented in the next section and section III discusses various aspects of its implementation. A brief introduction on how to use pMeasure, along with some added values of pMeasure, is given in section IV. This paper concludes with the envisioned improvements for pMeasure in the near future.

II. The Design of pMeasure

Architecturally, the entire pMeasure system consists of a collection of nodes, each running pMeasure both as a server and as a client. When running as a server, a pMeasure node receives measurement tasks from other nodes and if resources are available and various conditions are met, it carries out the tasks cooperatively. On the other hand, a pMeasure node running as a client can inject measurement tasks into the system, which automatically locates necessary and appropriate nodes for the tasks and monitors the execution of the tasks closely. Since all the nodes are equal in terms of functionalities they have and there is no need for centralized server or special support in the network infrastructure, pMeasure has the potential to connect millions of nodes and provide a measurement coverage that can satisfy the need for fine-grained, precise and timely measures of the Internet. Fig. 1 depicts a pMeasure system with a collection of seven pMeasure nodes that are scattered at the edge of three interconnected networks.

As Fig. 1 shows, each pMeasure node has three functional components. The measurement component is responsible for providing and requesting measurement tasks, while the Peer-to-Peer component maintains the overlay for the whole system. The management component, on the other hand, ensures the operation of the other two components in a secure, reliable and organized manner.

When a pMeasure node starts, the P2P component creates one Pastry [2] peer for each Network Interface Card(NIC) on the host machine. A Pastry peer is able to create and maintain a 40×15 state table, with each cell pointing to another Pastry peer in the system. The routing of messages in the system is then carried out by consulting the state tables along the path from source to destination [2]. In pMeasure, a Pastry peer's ID, instead of being randomized, is now constructed based on the IP address of the NIC in such a way that IDs and IP addresses can be computed from each other without difficulty. From now on in this paper, **ID**(ip) is used to represent the procedure which generates an ID from an IP address ip, while **NET**(x) is the procedure to retrieve

the network number from an ID x or an IP address x, and **HOST**(x) is the procedure to retrieve the host number from an ID x or IP address x. It is worth noting that IDs in pMeasure are used to identify the underlying Pastry peers. The unique identification of a pMeasure node is done through the node's public key. A node is supposed to have its public/private key pair setup properly before entering the system.

The messages routed in a pMeasure system are now the extended messages from Pastry, each denoting a measurement task to be carried out. The measurement component retrieves information from users, generates measurement tasks and instructs the P2P component to route the tasks to their destinations via the underlying Pastry peers. In case of a passive monitoring task involving an IP interface dst, the task will always be sent to a Pastry peer with **ID**(dst) as its ID. Should no such Pastry peer exist in the system, the passive monitoring task fails. In case of an active measurement task, the user specifies a network number, and the measurement component will explore the entire host space in an attempt to find a participating pMeasure node in the specified network. During its first attempt, the measurement component generates an IP address dst by combining the network number and a random host number. The task is routed to a Pastry peer with **ID**(dst) as its ID or having a closest ID to **ID**(dst). If the receiving peer id is outside the specified network, i.e. **NET**$(id) \neq$ **NET**(dst), or refuses to participate even if it is in the specified network, the measurement component computes the range of the host space[1] covered by the IP address dst and starts its second attempt with the remaining host space. This process repeats until a pMeasure node is found or the entire host space has been explored, and in the later case, the active measurement task is assumed as failed. Fig. 2 presents the overall procedure to compute the range $[s, e]$ covered by an IP address ip given the receiving peer id and the host space $[m, n]$ while Fig. 3 illustrates the computation of a particular coverage when the receiving peer resides in the specified network but refuses to participate in the measurement task, i.e., the third case in Fig. 2.

As mentioned before, the management component requires each pMeasure node to acquire a public/private key pair prior to entering the system. As the public keys issued by trusted authorities usually remain unchanged and seldom repeat themselves, they are used to identify pMeasure nodes uniquely[2]. Meanwhile, the management component requires all messages to be encrypted using

[1]Given a network number, the host space is a set of integers that can be legitimately used as host numbers.

[2]IP address is not used here since it changes from time to time for some edge users, e.g. ADSL users.

IF NET$(id) <$ **NET**(ip)
$[s, e] = [m,$ **MIN**$\{n,\ 2\times$**HOST**$(ip) - m\}];$
ELSE IF NET$(id) >$ **NET**(ip)
$[s, e] =$ [**MAX**$\{m,\ 2\times$**HOST**$(ip) - n\},\ n];$
ELSE IF HOST$(id) <$ **HOST**(ip)
$[s, e] =$ [**HOST**(id),
MIN$\{n,\ 2\times$**HOST**$(ip)-$**HOST**$(id)\}];$
ELSE IF HOST$(id) >$ **HOST**(ip)
$[s, e] =$ [**MAX**$\{m,\ 2\times$**HOST**$(ip)-$**HOST**$(id)\}$,
HOST$(id)];$

Fig. 2. Coverage Computation

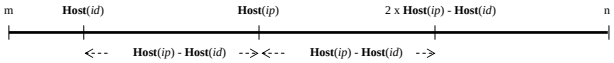

Fig. 3. The Computation of a Particular Coverage

the private keys before sent and be decrypted using the corresponding public keys after their arrival at the destination. This is done to provide security in a pMeasure system. Also, the management component requires that at the end of each measurement task, the task initiator send encrypted credentials to each participating node, specifying the task accomplished, the time, the initiator's public key, and the participating node's public key, etc. The management of all the credentials received is defined as accounting in pMeasure. With accounting, a pMeasure node will have a clear knowledge of whom it served and which tasks it participated in. Also, accounting provides an efficient method to prevent free riding in pMeasure. Upon the arrival of a measurement task, a node can decide to participate if the initiator's contribution is high or reject otherwise. A pMeasure system computes a node's contribution based on the number of different nodes it served and all these information can be retrieved from accounting. In addition, the management component enables a pMeasure node to compute the trustworthiness of a pMeasure node by validating the measurement results acquired by the node. In a passive monitoring task, the initiator can instruct a third node to send testing packets to the participating node. The participating node is deemed as trustable if the testing packets are reported and deemed as un-trustable otherwise. In an active measurement, the initiator can always find more than one node from an IP network to conduct the same task. While minimum and maximum results are discarded, the average of the remaining can be considered the ultimate result. Finally, the management component allows a pMeasure node to disable/enable

any of its measurement functionalities and for each enabled function, a black list of pMeasure nodes can be specified so that a pMeasure node can always refuse tasks from those on its black list. The pMeasure nodes that have low contributions according to accounting are one of the sources for the black list. Other sources include users judgements and those nodes labelled as un-trustable.

III. Implementation

This section first describes the capabilities of the current version of pMeasure and then explores three issues in detail, namely the traffic overhead, storage requirement, and computing resource consumption.

The passive monitoring component enables a pMeasure node to generate network traffic statistics on the host machine for the purpose of network monitoring and at the same time, to report the generated statistics to other pMeasure nodes upon request. A pMeasure node maintains statistics for every 15 minutes. The current 15 minutes statistics are updated in real-time and historical 15 minutes statistics are kept on secondary storage and can be retrieved when needed. The current version of pMeasure runs on Windows XP and WinPcap [3] is used to capture Ethernet frames, which are then decapsulated to retrieve IP packet header information. In case of a frame that contains the first fragment of an IP packet and the protocol used is TCP/UDP, TCP/UDP header information is retrieved as well. Based on the header information, data such as the number of packets sent/received, the number of bytes sent/received, and the Min/Max packet size sent/received are maintained for each active port and for each active NIC in every 15 minutes statistics. The active measurement component is able to measure not only the path characteristics between the task initiator and a participating node, but the path characteristics between two participating nodes as well. Table I depicts the capabilities of the active measurement component for the current version of pMeasure. In each active measurement task, node A and node B are synchronized to an Internet Time Server and probe packets are sent to each other at a fixed frequency within the measurement period. The measurement result is transmitted back to the task initiator immediately after the task is completed. In the current version of pMeausre, the possible combinations of $\langle$Node A, Node $B\rangle$ are $\langle$The task initiator, A participating node$\rangle$ and $\langle$A participating node, Another participating node$\rangle$.

Compared with manually negotiated measurement facilities, Pastry peers under the management of the P2P component have to ping the 600 peers that appear in its routing table every 15 minutes in order to detect failed or departed peers. This systematic pinging of others forms the majority of the traffic overhead in a pMeasure system. However, the overhead is tolerable according to the following calculation. A ping message in a pMeasure system has a length of 32 bytes, and with the overhead added at the transport layer, the IP layer and the link layer, a frame of size 74 bytes is transmitted for each ping message. Given a pMeasure system with 1 million nodes, each having one Pastry peer, the bandwidth consumed by a single pMeasure node is about 400**bps** and the total bandwidth consumed by the entire system is about 400**Mbps**.

In addition to the traffic overhead injected to the network, a pMeasure node consumes secondary storages for its historical 15 minutes statistics. A 15 minutes statistics consist of a list of elements, each of which records the activities on a distinct port and has a length of 25 bytes. The length of a 15 minutes statistics is thus dependent on the number of distinct ports used in the 15 minutes period. Table II outlines the secondary storage requirements for a pMeasure node to run continuously for 30 days as the number of the active ports increases from 1000 to 7000 in each statistical interval. According to Table II, the storage required for every 1000 distinct ports is only 72**MB** for a period of 30 days.

The performance of a pMeasure node is further evaluated in terms of the CPU time consumed and the main memory allocated. In the experiment, a number of tasks are sent to a pMeasure node and YourKit Java Profiler [8] is used to capture the total CPU time and the total main memory needed by the pMeasure node to accomplish these tasks. All the tasks are designed in such a way that the pMeasure node, which is running on a Dell Inspiron notebook running Windows XP with a 2.8**GHz** CPU and

TABLE I
Active Measurement Tasks

Measurement Tasks	**Description**
One-way Delay	Measures One-way Delay [4], Round-Trip Delay [5], Connectivity [6], and Trace Route [7] from pMeasure node A to pMeasure node B and from pMeasure node B to pMeasure node A simultaneously.
Round-Trip Delay	
Connectivity	
Trace Route	

TABLE II
Secondary Storage Requirements

Number of Active Ports	**Storage Required**	**Number of Active Ports**	**Storage Required**
1000	72**MB**	5000	360**MB**
2000	144**MB**	6000	432**MB**
3000	216**MB**	7000	504**MB**
4000	288**MB**	8000	576**MB**

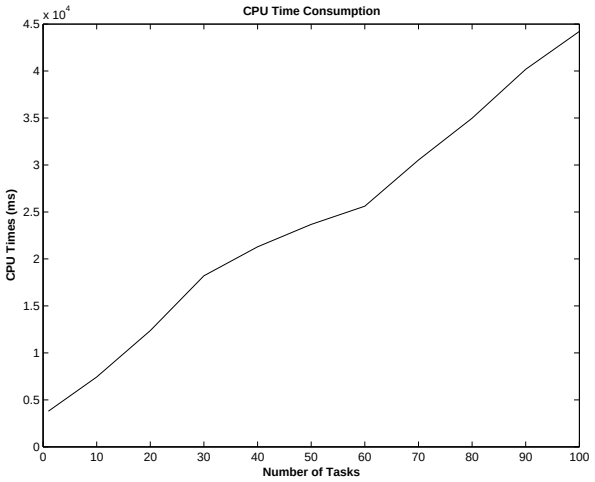

Fig. 4. CPU Time Consumption

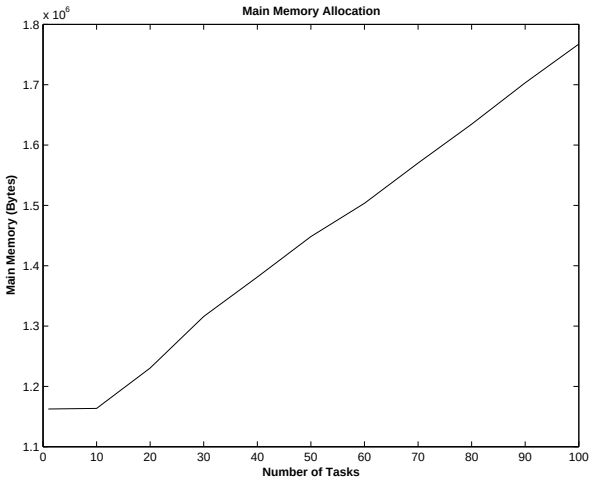

Fig. 5. Main Memory Allocation

512**MB** RAM, measures the round trip delay every five minutes between itself and another pMeasure node and for a duration of 10 minutes. As Fig. 4 depicts, the CPU time consumption increases linearly as the number of tasks increases, at a speed of about 4000**ms** for every 10 tasks. The main memory allocation, which is illustrated in Fig. 5, increases linearly as well, but at a speed of about 60**KB** for every 10 tasks.

IV. Using pMeasure

To convey a concrete feeling of how pMeasure can be used, this section presents two brief examples on how to measure port activities on an interface and on how to measure round trip time between two networks. In addition, two scenarios where pMeasure can be useful are presented as well.

To monitor port activities on an interface, the IP address of the interface, the start time and the end time have to be specified. In case that the activity on a particular port is required, the port number has to be given as well. A passive monitoring task is constructed from the inputs and immediately sent to the pMeasure node running on the host which contains the specified interface. The status of the submitted and received passive monitoring tasks can be checked as well. The screen snapshot in Fig. 6 depicts the status of an in-processing passive monitoring task, which shows the activities on interface 129.97.34.180 and port 21.

To measure the round trip time between a source network and a destination network, the start time, the end time, the network numbers and the measurement frequency have to be specified. An active measurement task is constructed subsequently. The pMeasure system then locates one pMeasure node that is running in the source network[3] and another in the destination network. The active measurement task is then sent to the found nodes for processing. In case that no pMeasure node is running in at least one of the specified networks, the task is labelled as failed. Similarly, the status of the submitted and received active measurement tasks can be checked. Fig. 7 depicts the status of an active measurement task that measures the round trip time from a pMeasure node in UWNET(129.97.0.0/16) to another pMeasure node in DOC-1-7-1-1-KTGC-1(69.193.90.0/23) every 5 minutes.

Being a P2P measurement application and conducting measurement from end-to-end, pMeasure offers capabilities and potentials that existing measurement facilities cannot compete with in certain situations. In Fig. 8, a pMeasure system is established to support SLAs between an ISP and its subscribers in which the ISP sets up some pMeasure nodes inside its network while each of its subscribers can set up one pMeasure node at the edge. With the pMeasure system, the ISP can collect measurement data to generate performance reports required by the SLAs, to evaluate its network performances, etc. And more importantly, the same system can be used by the subscribers to validate what they actually get is what they paid for. As the competition increases in the Internet service market and subscribers become more selective, this cooperative measurement of network performance can indisputably attract more subscribers on board. Fig. 9 depicts a pMeasure system with each node deployed in the subnetworks of a large campus network.

[3]In the current version of pMeasure, the source network is the network in which the task initiator resides, and the found node in the source network is the task initiator.

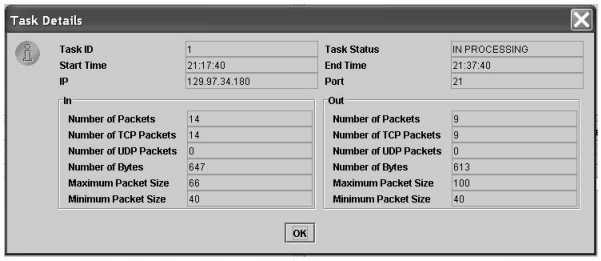

Fig. 6. Measurement of Interface Port Activities

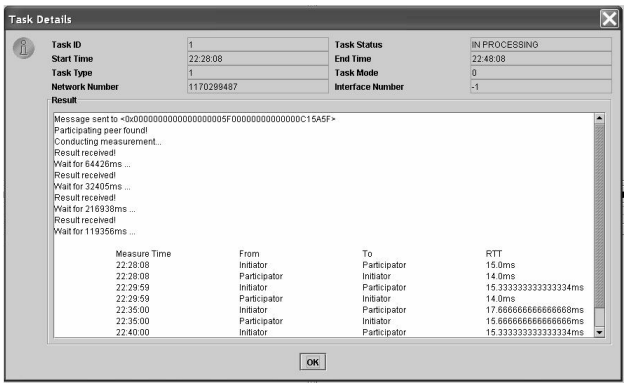

Fig. 7. Measurement of Path Characteristics

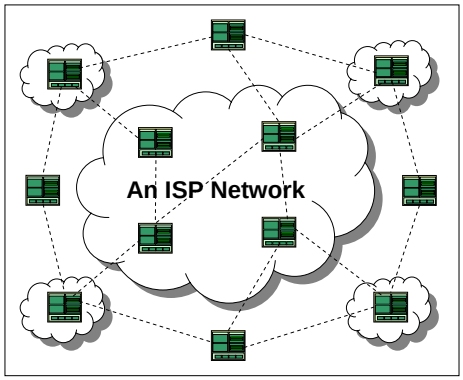

Fig. 8. A pMeasure System That Supports SLAs within an ISP Network

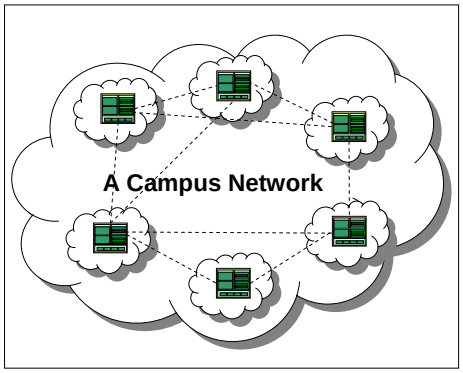

Fig. 9. pMeasure Systems That Monitors a Campus Network

Compared to the monitoring functionalities provided by the commonly used SNMP protocol, each pMeasure node can potentially act as the console, sending tasks to other pMeasure nodes so that the activities of each subnetwork, as well as the path characteristics between each subnetworks, can be monitored regularly. With this pMeasure system, a more comprehensive view of the campus network can be created and faults can be tracked down more efficiently.

V. Conclusions

pMeasure is a measurement tool that is built on top of Pastry, a Peer-to-Peer network substrate. This paper describes its design and evaluates it in terms of the injected traffic overhead, the required secondary storage, the consumed CPU cycles and the allocated main memory. A overview of how the current version of pMeasure can be used, along with the added values, is given subsequently. The current version of pMeasure can accomplish active measurements such as delay, connectivity, and trace route. Some primitive passive monitoring

tasks can be conducted too. In future versions, tools such as RRDTool [9] will be employed to generate graphical representations of the measurement results and head information for other protocols such as ICMP will be captured as well. In addition, future versions will support other popular platforms such as Linux and Solaris.

REFERENCES

[1] S. Srinivasan and E. Zegura, "Network measurement as a cooperative enterprise," in *Lecture Notes In Computer Science*, Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, Eds. March 2002, pp. 166 – 177, Springer-Verlag London, UK.

[2] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[3] Loris Degioanni, *Development of an Architecture for Packet Capture and Network Traffic Analysis*, Ph.D. thesis, Politecnico Di Torino, Turin, Italy, March 2000.

[4] G. Almes, S. Kalidindi, and M. Zekauskas, "A one-way delay metric for ippm," RFC 2679, September 1999.

[5] G. Almes, S. Kalidindi, and M. Zekauskas, "A round-trip delay metric for ippm," RFC 2681, September 1999.

[6] J. Mahdavi and V. Paxson, "Ippm metrics for measuring connectivity," RFC 2678, September 1999.

[7] V. Jacobson, "Traceroute software," Available from ftp://ftp.ee.lbl.gov/pub/traceroute.tar.Z, December 1988.

[8] YourKit LLC, "Yourkit java profiler, version 2.5," Available from http://www.yourkit.com/home/index.jsp, May 2004.

[9] Tobi Oetiker, "Rrdtool: Round robin database tool," Available from http://people.ee.ethz.ch/ oetiker/webtools/rrdtool/, 1999.