Available online at www.sciencedirect.com
SCIENCE DIRECT
Computer Communications 29 (2006) 1665–1674
computer communications
www.elsevier.com/locate/comcom

pMeasure: A peer-to-peer measurement infrastructure for the internet

Wenli Liu*, Raouf Boutaba

School of Computer Science, University of Waterloo, 200 University Ave W, Waterloo, Ont., Canada N2L 3G1

Available online 7 September 2005

Abstract

While the Internet grows in size and complexity, its timely and fine-grained measurements are increasingly needed. Although, a lot of efforts have been carried out in recent years to monitor the Internet, an united and systematic measurement infrastructure that is appropriate for measuring the Internet has never been established. In this paper, we describe pMeasure, a peer-to-peer (p2p) system that is capable to create a large number of measurement nodes on the Internet and to provide a synchronized and cooperative measurement system. The system can accept measurement tasks, locate required measurement facilities and fulfill the tasks subsequently. A prototype of pMeasure has been implemented, analyzed and tested on three operational networks. The experiments and further simulations indicate that measurement resources can be located efficiently and p2p-based measurements are very promising.

Keywords: Peer-to-Peer (p2p) Systems; Service location; Active measurement; Passive measurement

1. Introduction

With decades of development, the Internet has evolved into a complex network that interconnects millions of heterogeneous computing devices and systems worldwide. Yet the service provided at the time being is only the so called best-effort service. The next generation Internet, however, is envisioned to provide differentiated services and to support emerging QoS sensitive applications such as VoIP and video conferencing. A much more complex Internet in the near future is indisputable. At the same time, mission-critical applications that are increasingly deployed on the Internet have reshaped almost all aspects of human life in business, communication, education, and entertainment. The loss would be priceless if the Internet degraded in performance or stopped functioning even for a short period of time.

A lot of efforts have been made recently to ensure the healthiness of the Internet via better network management and traffic engineering approaches [1]. These approaches, however, depend on timely, precise and fine-grained measures of the Internet. Existing measurement facilities appear limited in satisfying this need due to the following reasons.

(i) The Internet is co-managed by multiple administrative organizations who usually measure their own part and pay very little attention in cross domain measurement. The acquired data are mostly unsynchronized from one organization to another and are of very limited importance to the evaluation of cross domain trends.
(ii) The measurement facilities deployed in the Internet backbone are few in numbers and are far from providing fine-grained measurements [2].
(iii) Most of the measurement facilities are based on Client/Server architecture wherein clients carry out the actual measurements while the acquired raw data are transferred to the server side for further processing. Hence, these facilities are inevitably plagued by problems like denial of service, central point of failure, just to name a few.
(iv) Adding measurement capabilities such as keeping the flow-based statistics in key Internet elements (e.g. routers) will adversely affect the scalability of the Internet.

As a consequence, what the Internet research communities and the industry face is mostly an unsynchronized, limited and partial view of the Internet. This is far from

* Corresponding author
E-mail addresses: w7liu@bbcr.uwaterloo.ca (W. Liu), rboutaba@bbcr.uwaterloo.ca (R. Boutaba).

0140-3664/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2005.07.008

satisfying the measurement needs and poses difficulties to the already complicated task.

In this paper, we extend our earlier work published in [3] and provide a more complete description and further evaluations of pMeasure, a p2p system that is able to attract millions of measurement nodes on the Internet. The system can accept measurement tasks from member nodes, locate and synchronize necessary measurement resources, and fulfill measurement tasks, subsequently. The system also manages node identification, authorization, accounting, and trust, all of which ensure measurement tasks being carried out in a organized, reliable and secure manner. Being a p2p system itself, pMeasure can easily scale to tens of thousands of nodes, a coverage that no existing measurement infrastructure can compete with, thus paving the road for providing fine-grained performance data of the Internet. In addition, the same system can be applied to LANs and WANs without any modification. We envision pMeasure as a unique architecture that can satisfy measurement needs of network management, traffic engineering and capacity planning for LANs, WANs and the Internet.

The rest of this paper is organized as follows. Related works are presented in Section 2. Section 3 concentrates on the design of pMeasure, including its measurement component, management component and Peer-to-Peer component. We have implemented a prototype of pMeasure and conducted a series of evaluations and experiments on the prototype. Section 4 describes the prototype and presents the evaluation and experiment results in detail. This paper concludes with future improvements in Section 5.

2. Related work

AT&T began to monitor and analyze its ISP business, namely WorldNet, from 1997, with both passive measurement and active measurement components [2]. The passive measurement facility consists primarily of PacketScope and Cisco NetFlow. PacketScope is a proprietary tool developed to capture IP packet headers for all the packets present on the wire. A PacketScope node is usually equipped with a 10 GB striped disk array and a 140 GB tape robot. To our best knowledge, PacketScope were only deployed at three representative locations on WorldNet and each of them will produce around 90 GB data per week. Meanwhile, Cisco NetFlow is enabled on all border routers to provide flow-based summaries. The active measurement component consists of a group of dedicated machines at about 20 router centers, exchanging test traffic and measuring delay, loss, and connectivity throughout the day. The acquired data, as well as other configuration data, are then utilized to reflect network usage, to design better web caching strategies, to compute dial service prices and to study peering traffic. Similar measurement infrastructures from other backbone operators include IPMon from SPRINT, the global internet backbone SLA performance statistics from Cable and Wireless, etc.

In addition to backbone operators' efforts, various organizations have set up their own measurement facilities, to either monitor the global Internet or large proprietary networks. National Internet Measurement Infrastructure (NIMI) by NLANR is one such facility built with the scalability issue taken into consideration [5,6]. Three types of nodes exist in NIMI, namely probes, Configuration Point of Contact (CPOC) and the Measurement Clients (MC). Probes are based on the Vern Paxson's Network daemon concept [7] and are capable of conducting various measurements. The main role of CPOC is to provide the initial policies for each NIMI probe, and to update these policies over the time. MCs are the interfaces between users and the NIMI measurement infrastructure. Upon the arrival of a user's measurement requirements, a MC assembles these requirements and requests NIMI probes to actually carry out the measurement. The involved NIMI probes then schedule and execute the measurement requests and send the results back to the MC, which then relays the results to the user. Although NIMI was built with scalability taken into consideration, the manual negotiation of measurement sites is far behind the speed at which the Internet expands. Up to date, NIMI has only attracted less than 200 sites. Other similar measurement facilities include, but are not limited to the following: the MOAT project from NLANR [4], the CoralReef passive monitor from CAIDA [8], Surveyor from Advanced Network & Services Inc. [8], etc.

Since the aforementioned measurement facilities are mostly constructed with a central control in mind, they hardly offer a coverage that is appropriate for providing fine-grained performance data of the Internet. To this end, the p2p measurement system proposed in [9] appears promising in that volunteers can join the system at will and contribute their computing resources to measurement tasks. Once in the system, a volunteer can submit measurement tasks and retrieve the results. At the same time, a volunteer can help others accomplish measurement tasks. Since volunteers have the potential to provide a much wider coverage than the one obtained through the tedious and manual negotiation process, an Internet scale measurement overlay is technically and economically feasible.

The approach proposed in [9] is plagued with a number of issues though. First, the measurement capabilities in [9] have to rely on a p2p network for locating peers and executing measurement tasks. However, [9] is vague in whether to create its own p2p network or to utilize an existing p2p network. In both cases, no description is given on how to realize it. Secondly, each peer in [9] acquires an Area of Responsibility (AOR) from another peer upon entering the system and gives the AOR back to the original peer upon its leave. Since peers can enter and leave the system at will, this division and merge of AORs generates a considerable amount of overhead. Thirdly, [9] estimates the path characteristics between a source IP address

and a destination IP address by measuring the path between the peer whose AOR contains the source IP address and the peer whose AOR contains the destination IP address. Obviously, the error in the estimation will be considerable when there are only a few peers available in the system. Last but not the least, free riding and security are usually major issues in a p2p network, but no action is taken in [9]. Compared to [9], our approach combines seamlessly with the underlying p2p network so that measurement tasks and their results can be transmitted among nodes smoothly. Meanwhile, the node identification, authorization, accounting and trust management in our approach can effectively alleviate free riding and increase reliability and security in the system.

3. Design of pMeasure

A pMeasure system, as depicted in Fig. 1, consists of a collection of nodes, each running pMeasure as a server and as a client at the same time. When running as a server, a pMeasure node receives measurement tasks from other nodes and participates in the tasks cooperatively. A pMeasure node running as a client can submit measurement tasks and retrieve results from the system. It is the system's responsibility to automatically locate necessary and appropriate nodes for measurement tasks and to monitor and facilitate the execution of the tasks closely. Since all the nodes are equal in terms of functionalities and there is no need for centralized servers or special support in the network infrastructure, pMeasure has the potential to attract millions of nodes and provide a measurement coverage that can satisfy the need for fine-grained, precise and timely measures of the Internet.

The three functional components that each pMeasure node has, i.e. *the measurement component*, *the management component* and *the Peer-To-Peer component*, will be described in detail in the following sections.

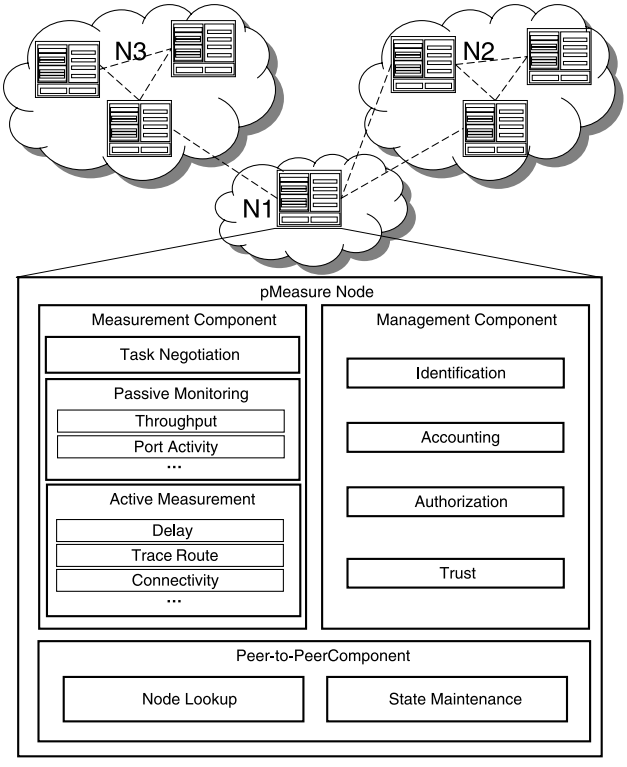

Fig. 1. pMeasure architecture.

3.1. The Peer-to-Peer component

A pMeasure node depends on its p2p component for submitting measurement tasks, receiving measurement results, and locating others when needed. Thanks to the plethora of research that has been conducted on p2p routing and location, this component can be accomplished based on any of the existing p2p networks. For the ease of explanation, Pastry [10] is used as the representative p2p network in this paper. Being a p2p network substrate, Pastry implements peer identification and query handling, and provides basic and important services on which other p2p applications can be built. To avoid flooding queries across the network, each peer in Pastry maintains a table of peers and given a query, a Pastry peer locates in its table the most closest peer and forwards the query to the peer subsequently.

Upon entering the system, a pMeasure node discovers the network interface cards(NIC) available on its host and creates a Pastry peer for each of them. Fig. 2 depicts a p2p component when four NICs are available on the host machine. As in Pastry, each peer inside a pMeasure node is identified by a 160 bit ID with the only difference that, instead of being randomized, an ID in pMeasure now consists of two meaningful parts. The first 80 bits house, the IP network number with leading zeros and the remaining 80 bits contain the host number with leading zeros. As such, IDs can be easily converted into IP addresses and vice versa. From now on in this paper, **ID**(*ip*) is used to represent the procedure that generates an ID from an IP address *ip*, while **NET**(*id*) or **NET**(*ip*) is the procedure to retrieve the network number from an ID *id* or an IP address *ip*, and **HOST**(*id*) or **HOST**(*ip*) is the procedure to retrieve the host number from an ID *id* or IP address *ip*. It worth noting that IDs in pMeasure are used for identifying the Pastry peers that are

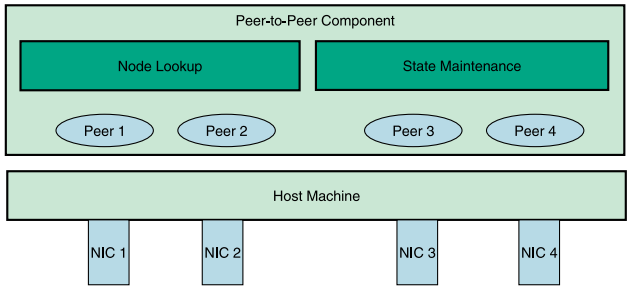

Fig. 2. The p2p component on a host with four NICs.

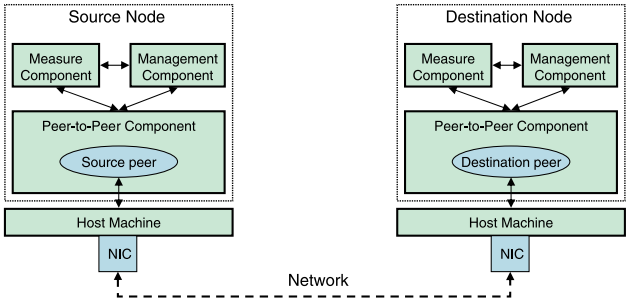

Fig. 3. Message forwarding in pMeasure.

housed in a pMeasure node. 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 overlay.

The messages routed in pMeasure are now the extended messages from Pastry, each denoting a measurement task to be carried out or a measurement result. Given a measurement task to be carried out on a host with an IP address *dst*, the source pMeasure node constructs a message of the form ($\langle \mathbf{ID}(dst) \rangle$, ⟨measurement task⟩) and sends the message via one of its Pastry peers. At the destination side, the receiving Pastry peer delivers the message to the pMeasure node for interpretation and execution. The result will be embedded in a new message targeting the Pastry peer at the source node and will be sent in a similar manner. Fig. 3 depicts the process in detail.

3.2. The measurement component

pMeasure has two categories of measurement tasks, namely the passive measurement tasks and the active measurement tasks. A passive measurement task measures the utilization of network resources and maintains statistics about the network traffic present on the machine. Compared with passive measurement tasks, an active measurement task focuses on the characteristics of a path between two or more participating nodes and cooperation among the participating nodes are mandatory. Further more, active measurement tasks are usually accomplished through generating testing traffic onto the path at the source node and observing the outcome at the destination node.

All measurement tasks in pMeasure are accomplished in two phases, the negotiation phase and the actual measurement phase. Fig. 4 describes these two phases in detail. During the negotiation phase, the source pMeasure node sends the task to the required pMeasure nodes and tries to find enough participating nodes for the task. For a passive measurement task involving IP address *dst*, the message will always be sent to a Pastry peer identified by $\mathbf{ID}(dst)$. The passive measurement task fails when the required Pastry peer does not exist in the system. For an active measurement task involving IP address *dst*, the message will be sent to a Pastry peer whose identifier *id* equals to $\mathbf{ID}(dst)$ or satisfies $\mathbf{NET}(id) = \mathbf{NET}(dst)$. The receiving pMeasure node notifies

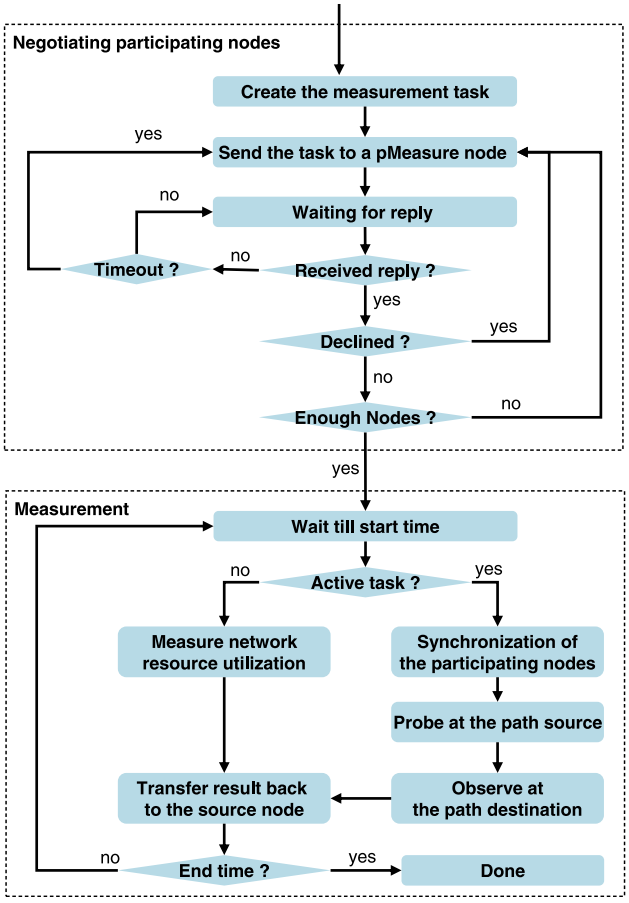

Fig. 4. The fulfillment of a measurement task.

the source node its decision, either to participate or not, and the source node can keep sending the task to other candidate pMeasure nodes until enough participating nodes have agreed to participate in the task or the source node itself has abandoned the task. A more detailed explanation on finding participating nodes for active measurement tasks will be presented in Section 3.2.2.

The second phase carries out the actual measurement and can start as specified by the task after a sufficient number of participating nodes have been negotiated. The details on the measurement will be presented in the next two subsections. Once a task has been fulfilled, the result is shipped back to the source node for further processing.

3.2.1. The passive measurement

The passive measurement functionalities are designed as such that a pMeasure node can be utilized to monitor the network traffic on the host. Specifically, each pMeasure node is equipped with a build-in passive measurement task, which maintains statistics about the network traffic present on the host for every 15 min. The current 15 min statistics are updated in real-time and saved on a secondary storage for future reference. In each 15 min statistics, a series of detailed counters are maintained, e.g. the number of packets and bytes from a port, the number of packets and bytes through a NIC,

etc. These statistics can be rendered to the host users and can be delivered to other pMeasure nodes when requested.

To request the statistics from a pMeasure node, other pMeasure nodes have to send passive measurement tasks to the pMeasure node. The message that denotes a passive measurement task is of format: (⟨*start_time*⟩, ⟨*end_time*⟩, ⟨*nic*⟩, ⟨*port*⟩, ⟨*direction*⟩, ⟨*client_id*⟩). The *start_time* and *end_time* specifies the time the task should be started and the task should be stopped. The next three fields serve as three filters in a row. The captured traffic is applied to the three filters in order and the portion coming out of the last filter is the traffic to be reported for this task. In particular, the *nic* field specifies a NIC and only traffic that has gone through this NIC will be reported. Similarly, the *port* field specifies the port and only traffic that uses the port will be reported. The *direction* field limits the traffic further to incoming or outgoing traffic only. The *client_id* field specifies the identifier of the Pastry peer to whom the measurement results should be sent once the task is done.

Fig. 5 depicts the internal design of this component. The packet capture engine is responsible for capturing packets from all the NICs. Details regarding the engine can be found in [11]. The captured packets, and more precisely the header information, are processed by the build-in passive measurement task. The current 15 min statistics are updated and saved on a secondary storage for future reference. The passive measurement tasks received from other pMeasure nodes are managed by the task pool, which accepts inputs from the build-in passive measurement task, filters the incoming packets and keeps statistics for each received task. When a task is done, the pool will send the results to the requesting node through the p2p component and then remove the task from the pool afterwards.

3.2.2. The active measurement

Unlike passive measurement tasks, active measurement tasks are carried out cooperatively among the participating

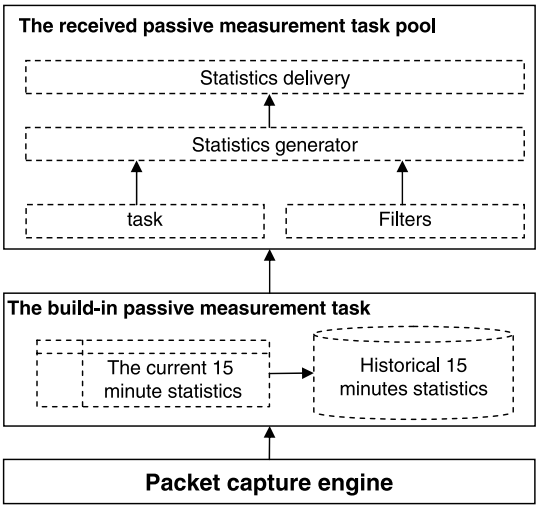

Fig. 5. Design of the passive component.

pMeasure nodes and the source node. During the measurement phase, a pMeasure node, which is either a participating node or the source node, sends probes to a second node. The second node processes each received probe and echoes back a new probe to the first node. By observing the outcomes of the probes shuttling back and forth between the two nodes, characteristics, such as one-way delay, round trip delay, connectivity, etc. regarding the path can be obtained.

The characteristics of the path between two networks can be measured as long as there is at least one participating pMeasure node running in each of the networks. In case multiple such nodes are present, each of them can potentially participate in the measurement. A random approach is taken in pMeasure to find a potential node from a network to balance the numbers of tasks that pMeasure nodes receive, i.e., the load, and to decrease cost, i.e. the number of attempts before a participating node can be found. To find some participating nodes from a network, the source pMeasure node randomly chooses an integer from the host space of the network and generates an IP address *ip* by concatenating the network number and the integer. An ID is then created using **ID**(*ip*). By targeting the encoded measurement task to this ID, the p2p component will always find a peer identified by *id* where $id = \mathbf{ID}(ip)$ or $|id - \mathbf{ID}(ip)|$ is the smallest number within the system [10]. However, chances are that the receiving node resides in a network other than the target network, i.e. $\mathbf{NET}(ip) \neq \mathbf{NET}(id)$, or the receiving node refuses to participate. In either case, the source node generates another random number from the remaining host space and repeats the process. This process is paused until enough participating nodes have been negotiated and is resumed when some participating nodes have departed or left the task. Fig. 6 details the procedure in finding one participating node from a network, and Fig. 7 illustrates the entire process.

Active measurement tasks are specified in extended Pastry messages as well. Depending on the nature of the task, the format of the message varies from one task to another. For tasks that measure one-way delay, round-trip delay, or loss ratio, the message is of the format (⟨*op_code*⟩, ⟨*start_time*⟩, ⟨*end_yime*⟩, ⟨*frequency*⟩, ⟨*packet_size*⟩). For tasks that measure connectivity or route information, the format of the message is (⟨*op_code*⟩, ⟨*start_time*⟩, ⟨*end_yime*⟩, ⟨*frequency*⟩). In these messages, *op_code* specifies the type of the task, *start_time* and *end_time* specify the time the task should be started and the time to be ended, *frequency* specifies the time in seconds between two adjacent measurements, and *packet_size* specifies the length of the IP packets that should be used in the measurement.

3.3. The management component

Like other p2p applications, pMeasure faces issues such as trust [12], free riding [13], etc. The design of pMeasure mitigates the effect of these issues by resorting to peer

Let N be the network number, and $[o, p]$ be the host number space. Let S be the set of portions of unexplored host number space. Initially $S = \{[o, p]\}$.

FindOneNode $(S)\{$

1. IF S is empty THEN return FAILED
2. Randomly choose an element $[m, n]$ from S, and $S = S - \{[m, n]\}$
3. Generates a random number k such that $m \leq k \leq n$
4. Create an IP address ip from N and k, and computes $\mathbf{ID}(ip)$
5. Issue a query targeting $\mathbf{ID}(ip)$, and waiting for reply
   Suppose the receiving peer's ID is id
6. IF $\mathbf{NET}(id) = N$ THEN
7. IF $\mathbf{HOST}(id) > k$, THEN
8. $S = S + \{[m, \mathbf{MAX}(2 \times k - \mathbf{HOST}(id), m)], [\mathbf{HOST}(id), n]\}$
9. ELSE
10. $S = S + \{[m, \mathbf{HOST}(id)], [\mathbf{MIN}(2 \times k - \mathbf{HOST}(id), \mathrm{n}), n]\}$
11. END
12. IF id agrees THEN
13. return id
14. ELSE
15. return **FindOneNode**(S)
16. END
17. ELSE
18. IF $\mathbf{NET}(id) < N$ THEN
19. $S = S + \{[\mathbf{MIN}(n, 2 \times k - m), n]\}$
20. ELSE
21. $S = S + \{[m, \mathbf{MAX}(2 \times k - n, m)]\}$
22. END
23. return **FindOneNode**(S)
24. END

$\}$

Fig. 6. Finding a participating node from an IP network.

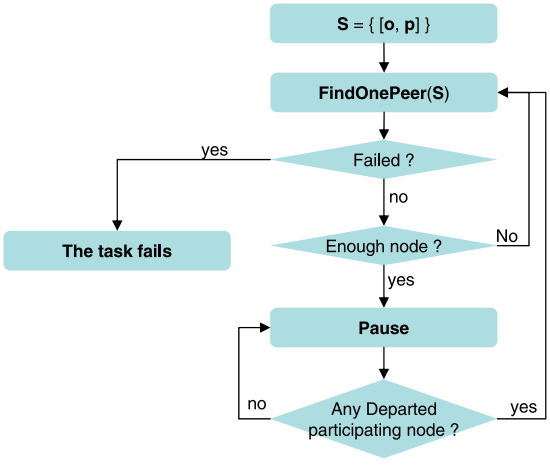

Fig. 7. Finding participating nodes.

identification, authorization and accounting, which are all provided by the management component.

3.3.1. Node identification

Since a pMeasure node can have more than one Pastry peers and IP addresses can change from time to time, it is infeasible to identify a pMeasure node using one of its Pastry peer's ID or simply an IP address. Instead, pMeasure nodes are identified by their public keys. Before entering the system, a pMeasure node is required to have its public/private key pair setup and signed by a trusted authority.

The public/private key pairs are used to encrypt and decrypt all the messages in pMeasure as well. In pMeasure, a source node encrypts a measurement task using its private key before distributing the task to other nodes. The receiving node, on the other hand, validates the source node's certificate and decrypts the measurement task subsequently using the source node's public key. In case of any false information, the receiving node can simply reject the measurement task. With encryption, node accountability and message integrity can be improved.

3.3.2. Authorization

In an Internet-scale measurement system, such as pMeasure, nodes cooperate with each other to smoothly accomplish measurement tasks. However, malicious nodes do exist. At the same time, a node may also want to restrict accesses to its measurement facilities to only a group of nodes. All these can be accomplished through authorization provided by the management component.

With authorization, a node can enable or disable any of its measurement functionalities and specify a list of non-authorized nodes for each enabled functionality. After having received a measurement task from a source node, the node checks whether the requested measurement functionality is enabled and whether the source node is in the corresponding non-authorized node list. In case that the requested functionality is disabled or the source node is in the list, the node can simply refuse to participate in the task.

3.3.3. Accounting

In pMeasure, a source node issues a credential to each participating node upon the accomplishment of a measurement task. A credential in pMeasure serves as a proof that a node once participated in a measurement task. A credential contains information such as the source node's public key, the participating node's public key, the description of the task accomplished, the time when the task is initiated and the source node's signature, etc. The management of all the credentials issued to a node in pMeasure is defined as accounting.

Accounting serves two purposes in pMeasure. First, it provides a clear knowledge of the nodes that a node has served and which measurement tasks the node once participated in. Secondly, accounting facilitates pMeasure

nodes in deciding whether to participate a measurement task or not. When making the decision, a node can always check the source node's contribution. In case that the source node's contribution is high, the receiving node can consider participating in the task. Otherwise, the receiving node may deny the task. pMeasure computes a node's contribution using the weighted sum of the number of different nodes the node has served and the number of tasks the node has participated in, as indicated in Eq. (1), where α_1 and α_2 are weights and $\alpha_1+\alpha_2=1$.

$$\alpha_1 \times \text{The number of tasks served} + \alpha_2 \times \text{The number of nodes served.} \quad (1)$$

3.3.4. Trustworthiness

Trust is a vital aspect that every Internet application has to deal with. The solution to this problem varies from one application to another [12]. This is particularly the case in pMeasure. Unlike the manually negotiated measurement sites, nodes in pMeasure have very limited knowledge about each other. The cultivation of trust in pMeasure, consequently, is difficult. However, we do believe that as nodes cooperate with each other over the time, smaller measurement communities will form as in other p2p applications and a full trustworthiness can be established inside a community afterwards.

In addition, pMeasure provides a source node with the capability to validate whether a measurement result is trustable. In a passive measurement task, the source node 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, while it is deemed as un-trustable otherwise. In an active measurement task, the source node can always instruct more than one node from a network to conduct the same task. While the minimum and the maximum results are discarded, the average of the remaining ones can be considered the final result.

4. Evaluation and experiments

Two problems are formulated to evaluate the efficiency of the algorithm in identifying participating nodes, as depicted in Fig. 6, in terms of cost and load balancing. First, suppose n pMeasure nodes are uniformly distributed in a host number space $[o, p]$, and each with the probability *prob* to participate in a measurement task. What is the average number of attempts required to find a cooperative node or to explore the entire host number space $[o, p]$? With this problem solved, we will be able to understand the overhead produced by the pMeasure system in locating participating nodes. Second, we distribute m measurement tasks to these pMeasure nodes and node i receives $load_i$ tasks. What is the maximum load difference among all the nodes on average? i.e. what is the value for the following formula: $Avg(Max(load_1, load_2,\ldots,load_n))$-$Min(load_1, load_2,\ldots, load_n)$)? Apparently, the smaller the value of the formula, the more balanced the load across pMeasure system will be.

Fig. 8 depicts our simulation results in terms of the cost with various combinations of the acceptance

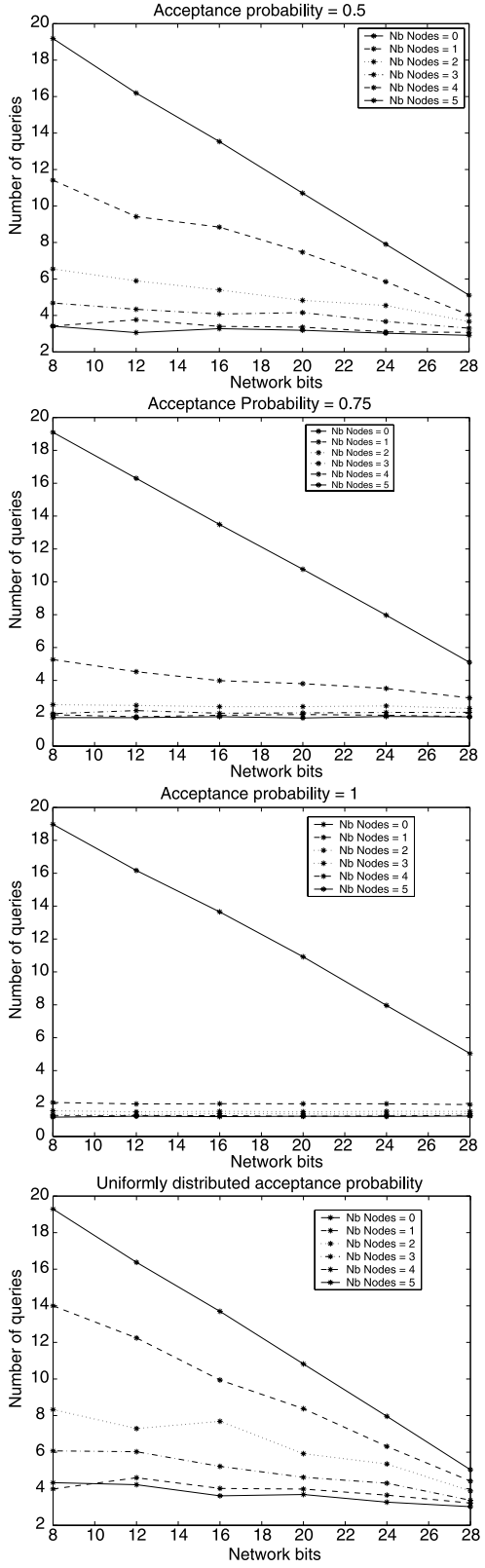

Fig. 8. Cost: the number of queries required.

probability, the number of nodes and the host number space(i.e. $[0, 2^{32\text{-NetworkBits}}]$. All nodes accept tasks with the same acceptance probabilities in each of the first three simulations and the probabilities are 0.5, 0.75 and 1, respectively. The fourth simulation no longer uses a fixed acceptance probability for all the nodes. Instead, each node accepts tasks according to the uniform distribution. Clearly, the cost decreases as the acceptance probability increases from 0.5 to 1. So does the cost as the number of nodes decreases and as the host space decreases. In addition, the cost decreases linearly with the host space in all the probability settings if the number of nodes is zero. Further more, only two queries are needed to traverse any host space when there is at least one node and all the nodes accept tasks with the probability of 1.

Fig. 9 presents our simulation results regarding the load distribution among pMeasure nodes. In our simulation, we uniformly allocate a certain number of pMeasure nodes in a host space and distribute 1000 tasks to the host space for 100 times. We record the maximum difference in the number of tasks received by nodes in each run and the average is used to produce the plots. In the first plot, each node accepts tasks with probability 1, and the acceptance probability is uniformly distributed in the second plot. As indicated in Fig. 9, the load tends to be more balanced when the number of nodes increases. The worst case happens when there are only two nodes available from a host space, with the maximum load difference being 34. Compared with the 1000 tasks we have distributed, this is negligible.

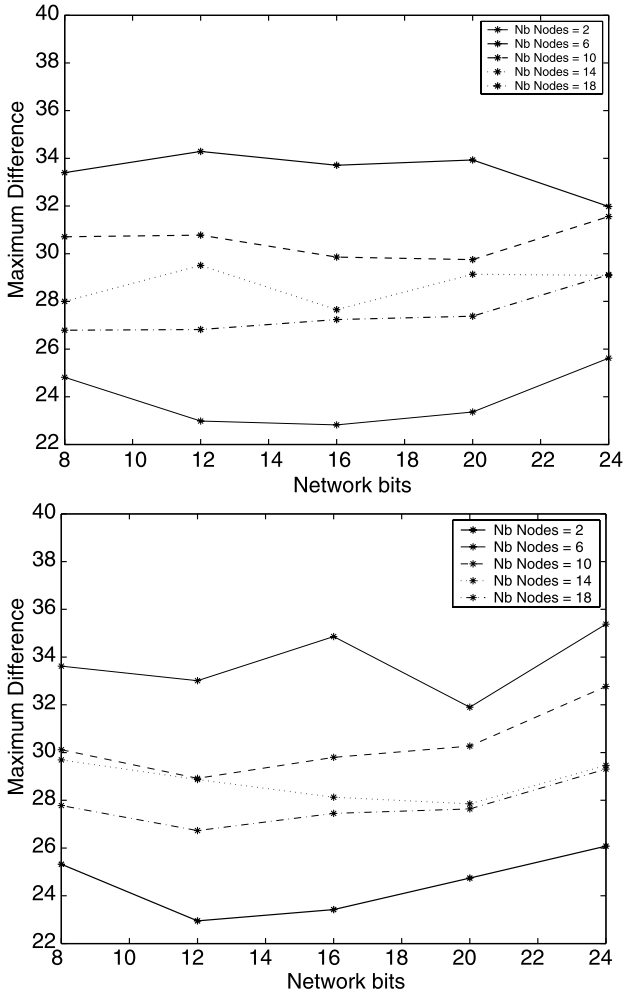

Fig. 9. Load distribution.

We have implemented a prototype pMeasure system using Java SDK 1.4 on Windows XP. With its passive measurement component, the prototype is able to capture frames present on Ethernet interface cards and to retrieve IP header, TCP header and UDP header information subsequently. The current 15 min statistics are then updated based on the captured information. In the prototype, each 15 min statistics maintains two entries for each active port, one for incoming traffic and another for outgoing traffic. Each entry in the statistics is of the following format: ⟨⟨*Port number*⟩, ⟨*Total number of packets*⟩, ⟨*Total number of bytes*⟩, ⟨*Total number of TCP packets*⟩, ⟨*Total number of TCP bytes*⟩, ⟨*Total number of UDP packets*⟩, ⟨*Total number of UDP bytes*⟩, ⟨*NIC used*⟩⟩. The active measurement aspect of the prototype is able to synchronize pMeasure nodes using Network Time Protocol (NTP) to an Internet time server and is able to measure round-trip delay, the number of hops and the connectivity between two pMeasure nodes. In addition, a user-friendly interface is created to facilitate task creation and management. The acquired measurements are rendered via the interface as well.

Compared with manually negotiated measurement facilities, Pastry peers in pMeasure have to probe the 600 peers that appear in their state table every 15 min in order to detect failed or departed peers. This systematic probing of others forms a considerable amount of overhead in a pMeasure system. However, the overhead is tolerable according to the following calculation. A probe message in a pMeasure system has a length of 32 b. Adding the overhead introduced at the transport layer, the IP layer and the link layer, a frame of size 74 b is transmitted for each probe 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. A pMeasure node consumes secondary storages for historical 15 min statistics as well. In the prototype, two entries are designated to each active port and each entry has a length of 31 b. The length of a 15 min statistics is thus dependent on the number of distinct ports used in the 15 min period. Our computation demonstrates that the secondary storage consumption is trivial as well. For a pMeasure node running continuously for 30 days, the secondary storage consumption is about 170 Mb for every 1000 distinct ports.

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 the total CPU time and the total main memory consumed by the node to accomplish these tasks

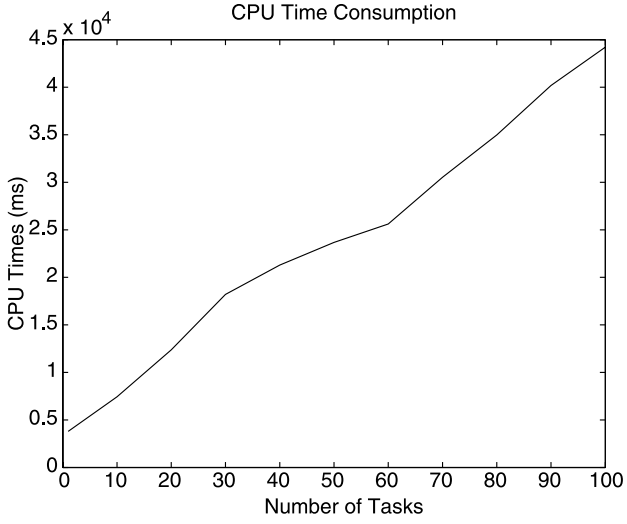

Fig. 10. CPU time consumption.

are captured. We conduct these experiments on a Dell Inspiron notebook with a 2.8 GHz CPU and 512 Mb RAM. The tasks are designed as such that the pMeasure node measures the round trip delay every 5 min between itself and another pMeasure node and for a duration of 10 min. As Fig. 10 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. 11, increases linearly as well, but at a speed of about 60 kb for every 10 tasks.

The prototype pMeasure system has been deployed in three different real world networks. In UWNET, two pMeasure nodes, named node 1 and node 2, are installed and connected to the Internet via a 100 Mbit Ethernet LAN. Node 3 is installed in Rogers-CAB-8 and node 4 is installed in SYMG021804-CA. Both nodes 3 and 4 are connected to the Internet using ADSL. A series of experiments have been conducted with the deployed pMeasure system. In the first

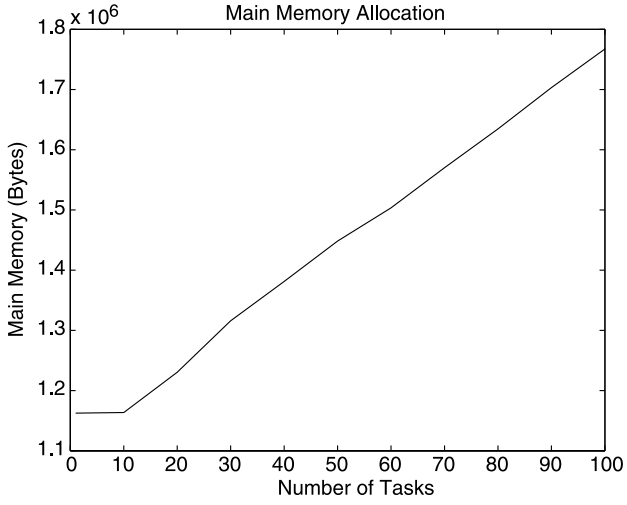

Fig. 11. Main memory allocation.

Table 1
Number of hops between nodes

From	To	Number of hops
Node 1	Node 3	16
Node 3	Node 1	12
Node 2	Node 4	15
Node 4	Node 2	11

experiment, node 1 sends 1000 active measurement tasks to Rogers-CAB-8 and node 2 sends another 1000 active measurement tasks to SYMG021804-CA. Nodes 3 and 4, who accepts tasks with probability 1, are found successfully for all the tasks. In the second experiment, node 1 sends 1000 active measurement tasks to UWNET and all the two nodes in UWNET record the number of tasks they have received. We conduct the second experiment for a hundred times and the result is very similar to the one obtained from simulation.

In the third experiment, node 1 sends an active measurement task to node 3. The active measurement task is designed as such that node 1 measures the round trip delay

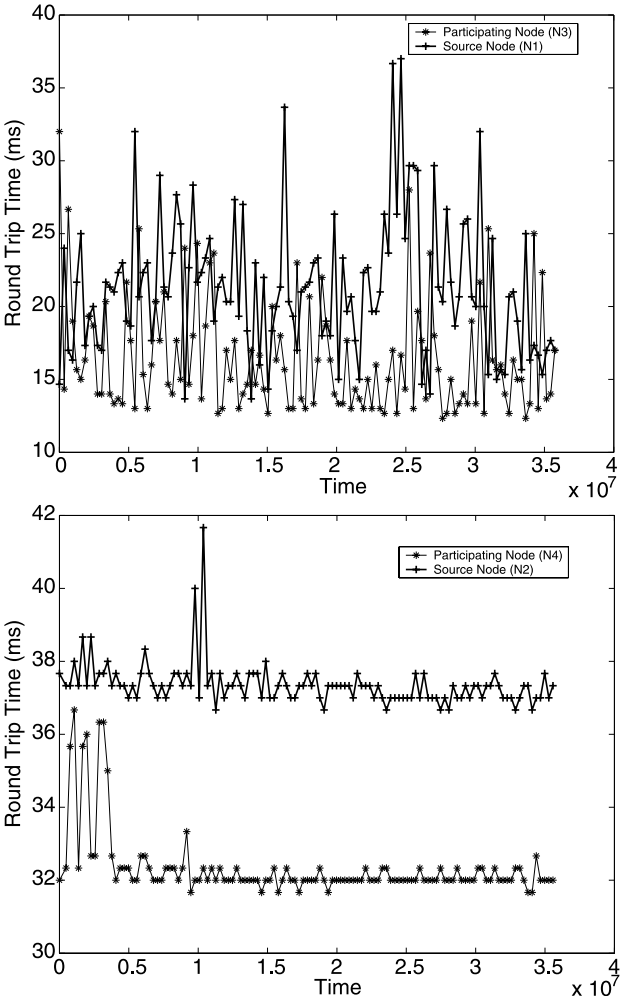

Fig. 12. Round trip delay between nodes.

and the number of hops to node 3 in every 5 min and node 3 measures the round trip delay and the number of hops to node 1 in very 5 min as well. At the same time, node 2 sends a similar active measurement task to node 4 so that nodes 2 and 4 measure the round trip delay and the number of hops to each other in every 5 min. In both cases, the tasks are kept running for 10 h.

Table 1 describes the number of hops from one node to another. From the result, we found that for each pair of nodes, the number of hops from the source node to the participating node is different from the number of hops from the participating node to the source node. In addition, the number of hops in all the paths remained unchanged during the time period of the experiment. An interesting aspect revealed by Table 1 is that both paths to the ADSL nodes have a few more hops than that of the reverse path.

Fig. 12 describes the round-trip delay obtained from each pair of nodes for the duration of 10 h. Similar to the number of hops, the round trip delays between a pair of nodes are asymmetric as well, with the round trip delay from the participating node to the source node slightly shorter than that from the source node to the participating node. In addition, the round trip delay varies from time to time.

5. Conclusion

In this paper, we described pMeasure, a p2p measurement infrastructure that can attract a large number of measurement nodes on the Internet. pMeasure can accept both passive and active measurement tasks from member nodes, locate and synchronize participating nodes for the task, and fulfill the tasks accordingly. Being a p2p system, pMeasure is able to self-organize into a large scale measurement infrastructure, thus satisfying various measurement needs.

We have implemented a prototype of pMeasure on top of Pastry, a p2p network substrate. We validated the efficiency of pMeasure in identifying participating nodes and evaluated it in terms of the introduced overhead, the required secondary storage, the consumed CPU cycles and the allocated main memory. In addition, the prototype was experimented on three real-world operational networks and the results are very promising. We will continue our efforts in improving the prototype system and in evaluating its performance via larger scale real-world deployment. Tools such as RRDTool will be employed to generate graphical representations of the measurement results and other popular platforms such as Linux and Solaris will be supported as well.

References

[1] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, X. Xiao, RFC3272: Overview and Principles of Internet Traffic Engineering, 2002.
[2] R. Caceres, N. Duffield, et al., Measurement and analysis of IP network usage and behavior, IEEE Communications Magazine (2000).
[3] Wenli Liu, Raouf Boutaba, James Won Ki Hong, pMeasure: A tool for measuring the Internet, The 2nd Workshop on End-to-End Monitoring Techniques and Services (E2EMON), 2004.
[4] A.J. McGregor, H.-W. Braun, J.A. Brown, The NLANR network analysis infrastructure, IEEE Communications Magazine (2000).
[5] V. Paxson, J. Mahdavi, A. Adams and M. Mathis, An Architecture for Large Scale Internet Measurement, IEEE Communications. 36 (8).
[6] V. Paxson, A.K. Adams, M. Mathis, Experiences with NIMI, The Symposium on Applications and the Internet, 2002.
[7] V. Paxson, End-to-end routing behavior in the Internet, SIGCOMM, 1996.
[8] M. Murray, K.C. Claffy, Measuring the immeasurable: global internet measurement infrastructure, The Passive and Active Measurement Workshop, 2001.
[9] S. Srinivasan, E. Zegura, Network measurement as a cooperative enterprise, First International Workshop on Peer-to-Peer Systems (IPTPS '01), 2002.
[10] A. Rowstron, P. Druschel, Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems, IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001.
[11] V. Jacobson, V. Paxson and C. Leres, Libpcap: the Library for Packet Capture, Available from http://www.tcpdump.org.
[12] T. Grandison, M. Sloman, A survey of trust in internet applications, IEEE Communication Surveys (2000).
[13] E. Adar, B.A. Huberman, Free Riding on Gnutella, First Monday, 5 (10).