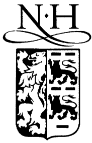

ELSEVIER

Performance Evaluation 36–37 (1999) 233–247

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

A server-based non-intrusive measurement infrastructure for enterprise networks

Yingfei Dong [a,*], Yiwei Thomas Hou [b], Zhi-Li Zhang [a], Tomohiko Taniguchi [a]

[a] University of Minnesota, Department of Computer Science and Engineering, Minneapolis, MN 55455, USA
[b] Fujitsu Laboratories of America, Sunnyvale, CA, USA

Abstract

In this paper we propose a server-based measurement approach that can be deployed in replicated servers within a *wide-area* enterprise network to provide distributed service to a large number of clients across the Internet. In our approach, each server performs traffic measurement and exchanges the collected metrics with peer servers. This server-based measurement approach consists of three steps: *non-intrusive* data collection, data analysis and performance metrics generation, and exchange of performance metrics among peer servers. As part of an experimental system we have built, we describe a performance metrics generation tool called *Woodpecker*, which is designed based on a non-intrusive passive packet capturing mechanism. Experimental results obtained using *Woodpecker* demonstrate that it is indeed feasible to employ non-intrusive measurement tools to generate the majority of desired performance metrics for dynamic server selection.
© 1999 Elsevier Science B.V. All rights reserved.

Keywords: Internet; Enterprise networks; Traffic measurement; Measurement infrastructure; Quality of service

1. Introduction

Server replication (or mirroring) is a common technique that has been used to provide scalable distributed service over the Internet. If done appropriately, server replication can avoid server overload and congested paths, and significantly reduce client access latency. In order to select a server to process a client request so as to provide the ‘best service’ for clients, measurement of server loads and network performance is critical in a replicated server system.

In this paper we propose a server-based measurement approach to facilitate dynamic server selection. This approach is developed in particular for an enterprise network environment, where a number of replicated servers are strategically deployed in geographically dispersed locations to provide distributed services to a large number of clients across the Internet. As typically is the case, we assume that these replicated servers are connected via a wide-area enterprise network, whereas clients access to

* Corresponding author. E-mail: dong@cs.umn.edu

0166-5316/99/$ – see front matter © 1999 Elsevier Science B.V. All rights reserved.
PII: S0166-5316(99)00023-1

the services provided by these servers through the Internet. Our proposed server-based measurement approach consists of three steps. First, each server conducts network measurement and collects network statistics in a fully independent and distributed manner. This step is referred to as *data collection* step. Second, each server analyzes the collected network statistics and generates relevant performance metrics that are useful for dynamic server selection. This step is referred to as *data analysis and performance metrics generation* step. Lastly, the network performance metrics generated by individual servers are exchanged among the peer servers. This step is referred to as *information exchange* step. Based on the exchange information, each server will build a performance metrics database. which will be used as an input to dynamic server selection algorithms.

A salient feature of our proposed server-based measurement approach is *client-transparency*. Because the network performance measurement as well as dynamic server selection are performed on the server side, a server-based measurement infrastructure based on our approach can be readily deployed without the need to install special hardware or software at clients and/or at network routers. Another important feature of our approach is the use of *non-intrusive* measurement techniques (see Section 2) to generate the majority of network performance metrics that are useful for dynamic server selection. Active probing based intrusive measurement techniques are used only for those performance metrics that cannot be obtained through non-intrusive measurement. These performance metrics such as path bandwidth (i.e., the bottleneck link capacity along a path) are typically *static*, therefore intrusive measurement is only invoked occasionally. As a result, our approach minimizes the extra network load incurred by measurement traffic injected into the network.

Since our server-based non-intrusive measurement approach is targeted for replicated server systems within an enterprise network environment, the communication overhead of performance metrics exchange can be limited and controlled. To see why this is the case, we first note that the number of replicated servers with a single enterprise is typically small. For example, currently Yahoo, Lycos, American Online, Alta Vista and Infoseek have, respectively, 15, 12, 8, 7 and 3 replicated web servers. Second, for the purpose of server selection, it suffices to generate network performance metrics on a subnetwork basis, instead of per client application instance, or per host. To further reduce the amount of performance information generated and exchanged, an enterprise replicated server system can also provide 'differentiated services' to clients by only keeping track of network performance for 'preferred' clients who regularly access to its service. In addition, exchange of performance metrics through a wide-area enterprise network can also reduce the impact of the system overhead on the perceived performance at clients.

To demonstrate the feasibility and advantages of the proposed server-based measurement approach, we have built an experimental measurement system. The non-intrusive data collection mechanism employed in this system is based on a passive packet capture tool, BSD Packet Filter. In addition, `tcpdump` is used to collect data traffic for each TCP flow between servers and clients. To analyze the collected raw data and generate relevant performance metrics for dynamic server selection, we design and implement a prototyping metrics generation software tool called *Woodpecker*. Through experiments, we demonstrate that *Woodpecker* is capable of generating a range of performance metrics that are useful for dynamic server selection. These performance metrics include throughput, goodput, packet loss rate and round trip delay for TCP flows. Static performance metrics such as path bandwidth are obtained by occasionally invocation of intrusive measurement tools such as `pathchar` [10]. We also design a simple SNMP-like request/response protocol for periodic exchange of performance metrics. The frequency of these exchanges can be controlled to maintain validity of the distributed performance

metrics database at each server while at the same time minimizing unnecessary traffic load within an enterprise network.

The remainder of this paper is organized as follows. In Section 2 we provide a brief overview of the background and related work. In Section 3 we first introduce a number of performance metrics that are useful for dynamic server selection, and then describe the experimental system built based on the proposed server-based non-intrusive measurement approach, in particular, the performance metric generation tool *Woodpecker*. The paper is concluded in Section 4.

2. Background and related work

In this section we briefly give an overview of the background for our work and provide a brief survey of related work.

2.1. Active probing versus passive watch

Network measurement and data collection techniques can be generally classified into two categories: *intrusive measurement* (also referred to as *active probing*) and *non-intrusive measurement* (also referred to as *passive watch*).

Active probing requires injection of a sequence of test packets into the network and obtain network performance information by analyzing the behavior of the feedback probing packets. Examples of active probing measurement techniques include the `pathchar` [10], `bprobe` and `cprobe` [5], `TReno` [12], and `traceroute` [9]. A major advantage of active probing is that it is controllable. For example, the time when network measurement is performed and the part of the network to be measured can all be determined and controlled. Furthermore, the desired level of measurement accuracy can also be controlled by appropriately adjusting the number of probing packets injected. The major drawback of active probing is that it needs to inject extra measurement traffic into the network, the amount of which may be substantial at times. Consequently, the extra measurement traffic may affect the normal network traffic behavior, and can potentially cause unnecessary congestion.

On the contrary, passive watch (or non-intrusive measurement) can infer network status by passively observing network traffic traversing through measurement points [3,13]. Using passive watch, no probing traffic is injected into the network for measurement. Instead traffic flowing through a measure point is captured and measured as it is. As a result, not all interested performance metrics can be obtained through passive watch. Furthermore, the measurement accuracy may also be limited, because performance measurement cannot be actively controlled.

2.1.1. Packet capturing and BSD packet filter

Many versions of Unix provide facilities for user-level packet capturing, making it possible to perform network traffic monitoring using general-purpose workstations. Captured packets are usually first stored in a dump file and then analyzed off-line. Such a process decouples the tasks of data collection and data analysis. A kernel agent, called a *packet filter*, can be used to selectively copy captured packets across the boundary of kernel space and user space.

A commonly accepted packet filter is the BSD Packet Filter (BPF) [13]. BPF is comprised of a network tap and a number of packet filters. The network tap collects copies of packets from network

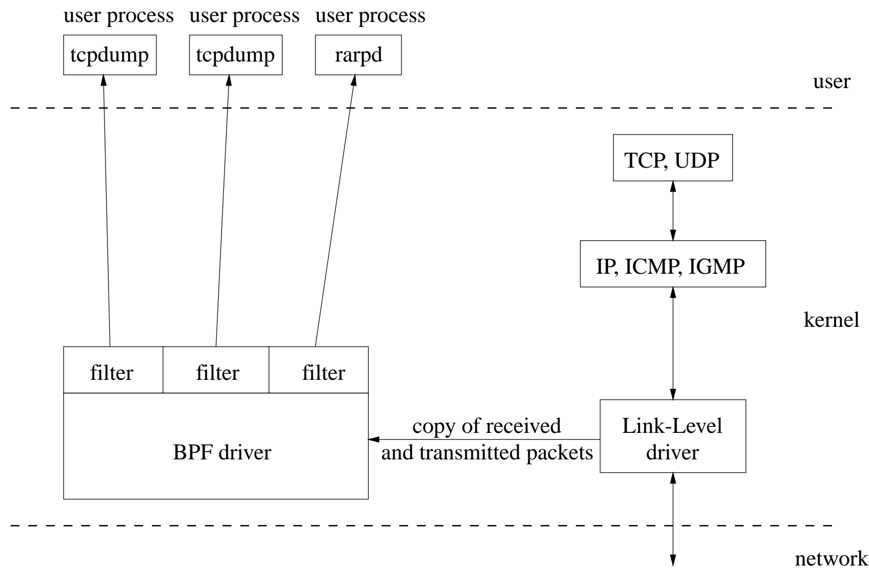

Fig. 1. BSD Packet Filter (BPF) architecture.

device drivers and delivers them to packet filters of listening applications (which perform network measurement and performance monitoring). A packet filter decides if a packet should be accepted, and, if so, the number of bytes of the packet should be copied to a listening application.

Fig. 1 illustrates how BPF works and its relationship with application measurement processes and link-level device drivers. In normal protocol processing, when a packet arrives at a network interface, the link-level device driver sends it up to the system protocol stack. But when BPF is listening on this interface, the driver invokes BPF first whenever a packet arrives. BPF feeds the packet to each participating process's filter. This user-defined filter decides whether a packet is to be accepted and how many bytes of the packet is to be copied. For each filter which accepts the packet, BPF copies the requested amount of data to the buffer associated with the filter. The device driver then regains control after BPF finishes processing the packet. If the packet is not addressed to the local host, the driver returns from the interrupt. Otherwise, normal protocol processing proceeds.

As typically only a small subset of network traffic is wanted by a measurement application process, and a dramatic performance gain can be realized by filtering out unwanted packets in an interrupt context. To minimize memory traffic, BPF filters packets 'in place' (where the network interface DMA engine put it) rather than copying the packets to some other kernel buffer before filtering. Thus, if a packet is accepted, only those bytes (e.g., the headers) which are needed by the filtering process are referenced by the host machine.

Studies in [13] show that BPF is an efficient tool for packet capture. It outperforms SunOS Network Interface Trap (NIT) in its buffer management mechanism, and Carnegie Mellon University (CMU)/Stanford Packet Filter (CSPF) in its filtering mechanism. Its programmable pseudo-machine model makes it extensible, as any knowledge of a particular protocol is factored out of the kernel. It is also portable, as it can work with various data link layers. Furthermore, the whole BPF system is small and easy to implement. Because of these advantages, in the prototyping server-based measurement system we build, we employ BPF as our non-intrusive data collection mechanism. Another reason we

use BPF is that its source code is readily available in public domain [13], whereas the source codes for many other packet capturing software tools (such as Solaris *snoop* [4]) are proprietary.

2.2. Internet measurement infrastructure

In recent years several researchers have proposed to develop a global Internet performance measurement infrastructure [1,7,8,11,17]. Such a global Internet performance measurement infrastructure is envisioned to provide a variety of network performance related services, including, among others, network performance monitoring and diagnostics and dynamic service/server selection. However, due to both the scale and the complexity of the Internet, such a global Internet measurement infrastructure is unlike to be fully deployed and operational in the near future. In contrast, with its modest scale and scope, our proposed server-based measurement infrastructure is designed in particular for replicated server systems within an enterprise network environment, and is readily deployable with relative low overhead.

3. Server-based measurement approach and *Woodpecker*

In this section we first identify a few performance metrics that are useful in dynamic server selection. We then present the server-based non-intrusive measurement approach, using the experimental system we have built as an example. In particular, we describe the organization of the performance metrics generation tool *Woodpecker* as well as the data structures and algorithms used in the tool. Sample experimental results obtained using *Woodpecker* are also presented. Various other issues such as implementation complexity of *Woodpecker* and the use of intrusive measurement tools for generating certain static performance metrics are also discussed. Finally, we provide a short overview of the performance metrics exchange mechanism implemented in our experimental system.

3.1. Performance metrics

Before describing the data collection process in our experimental system, we first introduce a few performance metrics which are considered useful for dynamic serve selection.

Path bandwidth: measured in bytes per second, is the minimum physical link capacity of all the links traversed from a server to a client.

Throughput: is the average number of bytes of data transferred per second between a server and a client, as experienced by a TCP flow.

Goodput: is measured as the average ‘useful’ data transfer rate (in bytes per second) between a server and a client, as experienced by a TCP flow. Due to network congestion, some packets of a traffic flow may be dropped. As a result, the goodput of TCP may be less than its throughput along a path. In most cases, the goodput of a flow dominates the overall elapsed time, i.e. the performance perceived by a user. Note that quantitatively, we have:

$$\text{path bandwidth} \geq \text{throughput} \geq \text{goodput} \geq 0.$$

Packet loss rate: is defined as the ratio of the number of packets lost to the total number of packets transmitted for a TCP flow. Packet loss rate reflects the congestion status of a particular path.

Round trip delay: is defined as the latency between the time when a packet is sent from a server and the time when the acknowledgment of the packet is received by the server. The minimum round-trip delay of a packet provides an upper bound for Round Trip Time (RTT). RTT is the delay due to propagation and transmission cost, and it reflects the minimum delay that is likely experienced by a packet when the path traversed is lightly loaded. When the round trip delay is much larger than RTT, it is an indication that network congestion may have occurred somewhere along the path.

The above performance metrics are generated using our tool in each measurement interval, which is in the scale of minutes. This time scale is chosen based on the studies of network stability [2,16].

We classify performance metrics into two classes, namely, *static* metrics and *dynamic* metrics. A performance metric is referred to as static if it is unlikely to change within a measurement interval. For example, the path bandwidth metric is considered as a static metric. Although it is possible that path bandwidth may change due to link upgrade, such a change occurs very infrequently, whose time scale is much larger than minutes. A performance metric is referred to as dynamic if it is tended to change within a measurement interval. Throughput, goodput, packet loss rate, and round trip delay metrics are considered as dynamic metrics.

3.2. Data collection

The data collection tool used in our system is `tcpdump`, which is a popular network monitoring and data acquisition tool [15]. It uses `libpcap`, which is a system-independent interface for user-level packet capture [14]. `libpcap` supports a filtering mechanism based on the architecture in BPF (see Section 2.1.1). Although most packet capturing interfaces support in-kernel filtering, `libpcap` utilizes in-kernel filtering only for the BPF interface [1].

The `tcpdump` tool operates by setting a network interface card in *promiscuous* mode. It is therefore capable of capturing every packet going across the wire. For generating performance metrics, we use filtering options on `tcpdump` which allows us to capture the packets of interest, namely data packets from a server to clients and their acknowledgments. Furthermore, we have made a revision on `tcpdump` so that we can periodically perform packet capturing and filtering every measurement interval.

3.3. Data analysis and metrics generation

In this section, we describe the data analysis and performance metrics generating tool in detail.

3.3.1. The organization of Woodpecker

Fig. 2 shows the three steps involved in packet processing of *Woodpecker*: capturing packets using `tcpdump`, buffering packets, and analyzing data.

Packet capture using `tcpdump`: As mentioned above, `tcpdump` is used to capture packets over a link. The output of `tcpdump` contains complete unprocessed information about packets. To extract the performance metrics of our interest, these ‘raw’ data must be further analyzed. *Woodpecker* processes the output of `tcpdump` and generates performance metrics on-line in memory, thereby avoiding disk accessing overhead.

[1] On the systems which do not have BPF, all packets are read into user-space and the BPF filters are evaluated in the `libpcap` library, incurring additional overhead (especially, for selective filters).

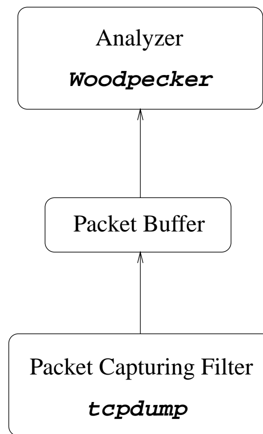

Fig. 2. Packet process flow in *Woodpecker*.

Packet buffer: A packet buffer is employed between `tcpdump` and *Woodpecker*.

Performance metrics generation: At the end of each measurement interval, *Woodpecker* uses the information in IP and TCP headers of the captured packets to build performance metrics for each TCP connection.

3.3.2. Data structures in Woodpecker

Due to different TCP connection status, appropriate data structures and algorithms must be employed to process packets captured during a measurement interval. As shown in Fig. 3, during a measurement interval, packets belonging to a flow may fall into the following four cases:

- *Case 1:* A connection starts and ends within the same interval, e.g., Connection 1 in Interval 1 and Connection 5 in Interval 2 (Fig. 3).
- *Case 2:* A connection starts in current interval and ends in another interval, e.g., Connection 3 in Interval 1 and Connection 4 in Interval 2 (Fig. 3).
- *Case 3:* A connection starts in a previous interval and terminates in the current interval, e.g., Connection 4 in Interval 3 (Fig. 3).
- *Case 4:* A connection starts from a previous interval, crosses the current interval, and terminates in a future interval, e.g., Connection 2 in Interval 2 (Fig. 3).

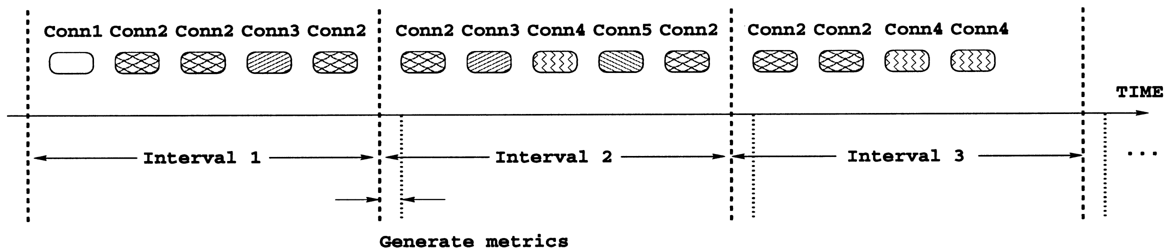

Fig. 3. An example of packet capturing over multiple intervals.

To handle the four cases appropriately, the following data structures are introduced.

Active connection table: We maintain an 'active' connection table to keep track of all connections that are currently active (i.e., not terminated yet) in the current measurement interval. We use one entry in the active connection table for each active connection. An entry is created upon the initiation of a connection, and is removed in response to the termination or time-out of the connection. An entry contains the following information:

Identifiers: are used to keep track of addresses for a connection, and the connection number. Each connection is identified by its source IP address, destination IP address, source port number, and destination port number. After identifying a packet by its addresses, a unique connection number will be assigned to identify such connection.

Performance metrics for current interval: are used to track of the network behavior during the current measurement interval. Since we are only interested in QoS perceived by a client, we focus on data sent from a server to clients and acknowledgments or requests from clients to the server. The following metrics are kept for current measurement interval.

- *Maximum packet delay* is the largest packet delay of the connection.
- *Minimum packet delay* records the minimum packet delay of the connection, which is the upper bound of RTT.
- *Average packet delay* is calculated by algebraic averaging of packet delays for all packets dumped in current interval.
- *Elapsed time* is the time in current interval that the first packet of a connection is recorded until the time within the same interval the last packet of the connection is collected.
- *Throughput* is calculated by dividing the total number of bytes transferred in this interval by the elapsed time. The total transferred bytes include the data part of each packet, whose size can be obtained from the dumped packet headers.
- *Goodput* is calculated by dividing the difference between the total transferred bytes in a measurement interval and the total retransmitted bytes in the same interval by the elapsed time.
- *Packet loss rate* is calculated by dividing the number of retransmitted packets by the total number of transferred packets in a measurement interval.

For connections that are terminated in the current measurement interval (i.e., Cases 1 and 3), we maintain the status about these connections in a separate data structure. Therefore, they will not appear in the active connection table in the future measurement interval.

For connections which cross over multiple intervals, both the metrics for the current interval and the metrics for all previous intervals since the initiation of the connection will be maintained as follows:

Accumulated metrics: record metrics for connections across multiple intervals.

- *Accumulated maximum packet delay*, *accumulated minimum packet delay*, and *accumulated average packet delay* are the maximum packet delay, the minimum packet delay, and the average of packet delay among all packets recorded so far for a connection.
- *Accumulated throughput*, *accumulated goodput*, and *accumulated packet loss rate* are similarly defined as throughput, goodput, and packet loss rate, respectively, measured during the time period from the initiation of the first packet of a connection till the current measurement interval.
- *Accumulated elapsed time* maintains the time of the connection from the time when it transmits the first packet to the time when its last packet is captured.

Time-out for an entry: Since there are packets lost along a path, some TCP connection may never see their closing flag FIN. A *keep-alive* timer is employed to solve this problem [18]. A similar idea

is also used in our data analysis. In particular, if we find that a connection is inactive for more than T_{alive}, we assume it is closed. We set the value of T_{alive} as the same as the value of TCP keep-alive timer (i.e., 2 hours) [18].

Frequently-visited subnets: We also maintain a set of records for frequently-visited subnets, instead of one for each individual client from these subnets. To decide whether to create an entry for a particular subnet, we use a threshold. When the number of connections or the sum of packets from a specific subnet is above the threshold, a frequently-visited subnet entry will be created.

3.3.3. Algorithms for performance metrics generation

In this section, we describe the algorithm for metrics generation. Fig. 4 shows the flow-chart of the algorithm used for data analysis and performance metrics generation. As shown in Fig. 4, *Woodpecker* repeatedly reads packets from the buffer during a measurement interval. Based on the information in the IP and TCP headers of the individual packet, the status of the respective connection is updated. At the end of a measurement interval, *Woodpecker* generates a new set of metrics for each connection.

As shown in Fig. 4, the packet processing in *Woodpecker* is performed in a two-branch loop. The upper right-hand branch controls what needs to be processed at the end of a measurement interval. The algorithm examines the time stamp on a packet and checks to see if it is beyond the current measurement interval. Once the time stamp on a packet indicates that the current measurement interval is ended, *Woodpecker* generates a new metrics report and updates the accumulated metrics set. Then *Woodpecker* starts the next interval.

If the time stamp on a packet is within the current measurement interval (the upper left branch in Fig. 4), the packet will be processed as follows. The algorithm first searches the *active connection table* to see whether this packet belongs to an existing connection. If a match is found, the corresponding entry for this connection is updated with the information in the packet. If a match is not found, the packet is

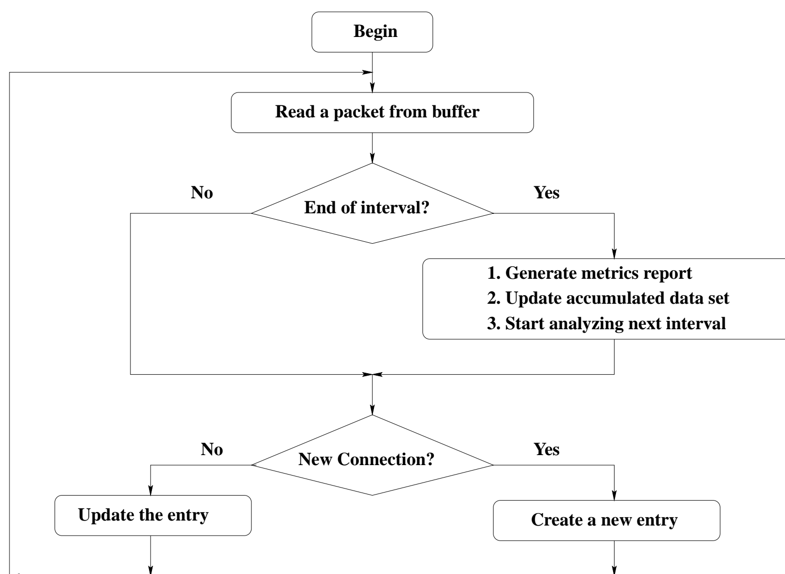

Fig. 4. Performance metrics generation algorithm in *Woodpecker*.

the first packet of a new connection. The algorithm creates a new entry corresponding to this connection in the *active connection table*. Note that a connection is identified by its source IP address, source port number, destination IP address, and destination port number.

3.3.4. Sample results

In this section, we present some sample experimental results obtained using *Woodpecker*. We installed *Woodpecker* on a SUN Ultra 2 workstation, `endeavor.net.fla.fujitsu.com`, which runs as a server. The operating system on this machine is Solaris 2.6. Two clients are set up. One is on an Ultra 1, `julius.cs.umn.edu`, with Solaris 2.5.1, and the other is on a SPARC 20, `napali.net.fla.fujitsu.com`, with Solaris 2.5.1. The server endeavor and client napali are located at Fujitsu Labs of America in Sunnyvale, CA, while client julius is located at Computer Science Department of University of Minnesota in Minneapolis, MN. In the following, we list the set of performance metrics for a few selected connections during three consecutive measurement intervals. These results shows the various cases of connection status of these connections during each measurement interval. Connections 1 and 2 in Interval 1 is the simplest case (Case 1), which are initiated and terminated within the same measurement interval (1 min). Connection 3 in Intervals 1, 2 and 3 is the most complicated case, which starts in Interval 1 and terminates in Interval 3.

```
Woodpecker Version 1.0, August 15, 1998

(Interval 1)

     ================================
     TCP Connection No.1 ---
            From the server:  endeavor.net.fla.fujitsu.com:47320
         ===> To the client:  julius.cs.umn.edu:telnet
            Elapsed time:  5.776208 seconds
                  for      1488 Bytes in       24 packets
         avg segm size         62 Bytes
         Throughput           258 Bytes/sec
         Goodput              238 Bytes/sec
         Packet loss rate     0.076923
         Min Round-trip Delay 87.2 ms
         Max Round-trip Delay 102.7 ms
         Avg Round-trip Delay 93.3 ms
         STD Round-trip Delay 4.8 ms
         ......

     ================================
     TCP Connection No.2 ---
            From the server:  endeavor.net.fla.fujitsu.com:672
         ===> To the client:  napali.net.fla.fujitsu.com:2049
            Elapsed time:  0.107327 seconds
                  for     6105 Bytes in       37 packets
         avg segm size        165 Bytes
         Throughput         56882 Bytes/sec
         Goodput            56882 Bytes/sec
         Packet loss rate   0.000000
         Min Round-trip Delay      1.0 ms
         Max Round-trip Delay      1.5 ms
         Avg Round-trip Delay      1.1 ms
```

```
        STD Round-trip Delay      0.1 ms
        ......

    ================================
    TCP Connection No.3 ---
           From the server:  endeavor.net.fla.fujitsu.com:47323
        ===> To the client:  julius.cs.umn.edu:telnet
           Elapsed time:  47.690798 seconds
                 for     2450 Bytes in       43 packets
        avg segm size         57 Bytes
        Throughput            51 Bytes/sec
        Goodput               50 Bytes/sec
        Packet loss rate  0.027778
        Min Round-trip Delay     87.0 ms
        Max Round-trip Delay     770.1 ms
        Avg Round-trip Delay     146.5 ms
        STD Round-trip Delay     118.1 ms

        Accumulated Throughput  51 Bytes/sec
        Accumulated Goodput     50 Bytes/sec
        Accumulated Packet loss rate 0.027778
        Accumulated Min Round-trip Delay 87.0 ms
        Accumulated Max Round-trip Delay 770.1 ms
        Accumulated Avg Round-trip Delay 146.5 ms
        Accumulated Elapsed Time 47.690798 seconds
        ......

(Interval 2)

    ================================
    TCP Connection No.3 ---
           From the server:  endeavor.net.fla.fujitsu.com:47323
        ===> To the client:  julius.cs.umn.edu:telnet
           Elapsed time: 59.608097 seconds
                 for      452354 Bytes in       553 packets
        avg segm size          818 Bytes
        Throughput            7588 Bytes/sec
        Goodput               7053 Bytes/sec
        Packet loss rate      0.070520
        Min Round-trip Delay 40.3 ms
        Max Round-trip Delay 109.3 ms
        Avg Round-trip Delay 51.6 ms
        STD Round-trip Delay 24.7 ms

        Accumulated Throughput  4238 Bytes/sec
        Accumulated Goodput     3954 Bytes/sec
        Accumulated Packet loss rate 0.067114
        Accumulated Min Round-trip Delay 40.3 ms
        Accumulated Max Round-trip Delay 770.1 ms
        Accumulated Avg Round-trip Delay 58.4 ms
        Accumulated Elapsed Time 107.298895 seconds
        ......

(Interval 3)
```

```
================================
TCP Connection No.3 ---
       From the server:  endeavor.net.fla.fujitsu.com:47323
    ===> To the client:  julius.cs.umn.edu:telnet
       Elapsed time: 57.096218 seconds
             for      462551 Bytes in       550 packets
    avg segm size          841 Bytes
    Throughput            8101 Bytes/sec
    Goodput               7459 Bytes/sec
    Packet loss rate      0.080000
    Min Round-trip Delay 43.1 ms
    Max Round-trip Delay 98.2 ms
    Avg Round-trip Delay 53.3 ms
    STD Round-trip Delay 14.9 ms

    Accumulated Throughput 5580 Bytes/sec
    Accumulated Goodput    5171 Bytes/sec
    Accumulated Packet loss rate 0.073298
    Accumulated Min Round-trip Delay 40.3 ms
    Accumulated Max Round-trip Delay 770.1 ms
    Accumulated Avg Round-trip Delay 55.9 ms
    Accumulated Elapsed Time 164.395113 seconds
    ......
```

3.3.5. Implementation complexity

Packet capturing and analyzing overhead. In our metrics generation scheme, certain computing power is consumed for capturing packets and analyzing those packets.

It has been shown in [13] that BPF is much more efficient than several other packet capturing tools in both packet capturing and filtering. In our implementation, the whole processing time for metrics generation for a typical dumping interval (1 min) is usually in milliseconds. Even in the worst case, it is no more than a few seconds.

The overhead of data analysis includes reading each packet from the packet buffer, searching in the active connection table, and updating the metrics for the respective connection. Since we only dump the header of each packet, which has fixed size of 54 bytes, the time to read a packet from the packet buffer is, therefore, almost in constant time. Furthermore, since the number of connections that can be set up simultaneously at a server is limited, the overhead of searching in the connection table is also quite small. Finally, the computation overhead is also insignificant since updating the performance metrics only requires several arithmetic operations and memory accesses.

Quantitatively, we measure that the average cost of processing a packet is about 30 to 70 microseconds in our testing system. As a result, in a 1-minute interval, at least 1 million packets can be processed by our *Woodpecker*. It is much more than sufficient for a 100Base–T Ethernet interface. Furthermore, the server used here is only a SUN Ultra 2 workstation. If a powerful machine or extra hardware is used, there should not be any problem to process packets captured over a higher bandwidth link for the measurement.

Server load. Although *Woodpecker* will incur some additional load on a server, such additional load can be effectively controlled since the server is within an enterprise network. For example, we can reserve certain computing power on the server for *Woodpecker* by limiting the maximum number of client jobs that a server can accept. If the maximum number of such client jobs are reached, we either direct new

request for another replicated server or add more servers in our enterprise network. Moreover, additional hardware can be dedicated for measurement purpose. All these alterations on a server is possible since it is within our enterprise networks. In practice, a special hardware based measurement tool can be used to reduce the overhead of performance metric generation imposed on a server.

3.4. Using intrusive measurement tools to get path bandwidth

In our experimental system, we also employ intrusive measurement tools to obtain static performance metrics that cannot be generated by *Woodpecker*. For example, we use `pathchar` [10] to measure path bandwidth of a route between a server and a client. It is important to point out that intrusive measurement tools are used infrequently because of the nature of static performance metrics such as path bandwidth.

3.5. Metrics exchanging among networked servers

Our *Woodpecker* software is performed independently at each individual replicated server within a wide area enterprise network. As a final step to build a measurement infrastructure within such an enterprise network, we let the servers exchange performance metrics periodically among themselves.

A simple SNMP-like, request/response protocol is used for periodic exchange of performance metrics among peer servers. In our experimental system, the frequency of performance metrics exchange is set to a multiple of the length of measurement intervals to prevent the management traffic overload on the enterprise network. To further reduce the communication overhead incurred by performance metrics exchange, a selective push mechanism is implemented to allow a server to dynamically inform other servers of *significant* changes in performance metrics.

4. Concluding remarks

In this paper we presented a server-based measurement approach for building a (primarily) non-intrusive measurement infrastructure in a wide-area enterprise network environment. This server-based measurement approach was designed with the specific objective of facilitating dynamic server selection in a replicated server system. Our approach has three salient features. (1) It is client-transparent and readily deployable. (2) It uses non-intrusive measurement techniques to generate the majority of performance metrics that are useful for dynamic server selection, thereby minimizing the extra network load incurred by the measurement infrastructure. (3) It can be implemented in a scalable manner with relative low communication overhead. To demonstrate the feasibility and advantages of the proposed server-based measurement approach, we built an experimental measurement system. In particular, we described a performance metrics generation tool called *Woodpecker*, which was designed based on a non-intrusive passive packet capturing mechanism. Experimental results obtained using *Woodpecker* demonstrated that it is indeed feasible to employ non-intrusive measurement tools to generate the majority of desired performance metrics for dynamic server selection.

Our current work has primarily focused on generating performance metrics for TCP applications. With the emergence of multimedia streaming applications, which are typically implemented using UDP, it is also necessary for our system to be able to measure UDP traffic as well. This will be one direction of our future work. Another direction of our future work is to study dynamic server selection algorithms

using the performance metrics collected by the measurement infrastructure. Some initial studies along this direction has been conducted and is reported in [6].

References

[1] G. Almes, IP performance metrics: metrics, tools, and infrastructure, http://io.advanced.org/surveyor/, 1997.
[2] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, M. Stemm, R.H. Katz, Analyzing stability in wide-area network performance, Proc. ACM SIGMETRICS, 1997.
[3] N. Brownlee, C. Mills, G. Ruth, Traffic Flow Measurement: architecture, RFC 2063, Internet Engineering Task Force, Jan. 1997.
[4] B. Callaghan, R. Gilligan, Snoop Version 2 Packet Capture File Format, RFC 1761, Internet Engineering Task Force, Feb. 1995.
[5] R.L. Carter, M.E. Crovella, Measuring bottleneck link speed in packet-switched networks, Performance Evaluation 27 & 28 (1996) 297–318.
[6] Y. Dong, Z. Zhang, Y.T. Hou, Server-based dynamic server selection algorithms, Technical Report, Department Computer Science and Engineering, University of Minnesota, March 1999.
[7] P. Francis, S. Jamin, V. Paxson, L. Zhang, D.F. Gryniewicz, Y. Jin, An architecture for a global Internet host distance estimation service, Proceedings IEEE INFOCOM, New York, 1999.
[8] C. Huitema et al., Project Felix: independent monitoring for network survivability, ftp://ftp.bellcore.com/pub/mwg/felix/index.html, 1997.
[9] V. Jacobson, Traceroute, ftp://ftp.ee.lbl.gov/traceroute.tar.Z, 1989.
[10] V. Jacobson, pathchar — a tool to infer characteristics of Internet paths, http://ee.lbl.gov/nrg-talks.html, April 1997.
[11] C. Labovitz et al., The Internet performance and analysis project, http://www.merit.edu/ipma/docs/team.html, 1997.
[12] M. Mathis, J. Mahdavi, Diagnosing Internet congestion with a transport layer performance tool, Proceedings INET, 1996, Montreal, Canada.
[13] S. McCanne, V. Jacobson, The BSD packet filter: a new architecture for user-level packet capture, Proceedings USENIX, 1993, San Diego, CA.
[14] S. McCanne, C. Leres, V. Jacobson, libpcap, ftp://ftp.ee.lbl.gov/libpcap.tar.Z.
[15] S. McCanne, C. Leres, V. Jacobson, tpcdump, ftp://ftp.ee.lbl.gov/tcpdump.tar.Z.
[16] V. Paxson, Measurements and analysis of end-to-end Internet dynamics, Ph.D. dissertation, University of California at Berkeley, 1997.
[17] V. Paxson, J. Mahdavi, A. Adams, M. Mathis, An architecture for large-scale Internet measurement, IEEE Commun. Mag. (August 1998) 48–54.
[18] W.R. Stevens, TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley Longman, 1994.

Yingfei Dong received his B.S. and M.S. degrees in Computer Science from Harbin Institute of Technology, China, in 1989 and 1992, and a Doctorate degree in Engineering from Tsinghua University, China, in 1995, respectively. Currently he is a Ph.D. candidate in the Department of Computer Science, University of Minnesota, Minneapolis, MN. He received the ‘Ying Xue’ Scholarship from Harbin Institute of Technology in 1990, a Motorola Scholarship from Tsinghua University in 1994, and a Graduate Research Fellowship Award from University of Minnesota in 1998, respectively. He spent the summer of 1998 at Fujitsu Laboratories of America, Sunnyvale, CA, working on network performance measurement and optimal sever selection algorithms. His current research interests include computer communication networks and storage systems, especially network performance measurement, QoS guarantee issues, electronic commerce, multimedia networking and distributed locking mechanism in file systems. He is a student member of IEEE and ACM.

Yiwei Thomas Hou obtained his B.E. degree (Summa Cum Laude) from the City College of New York in 1991, the M.S. degree from Columbia University in 1993, and the Ph.D. degree from Polytechnic University, Brooklyn, New York, in 1997, all in Electrical Engineering. He was awarded a National Science Foundation Graduate Research Traineeship for pursuing Ph.D. degree in high speed networking, and was recipient of Alexander Hessel award for outstanding Ph.D. dissertation in 1998 from Polytechnic University. While a graduate student, he worked at AT&T Bell Labs, Murray Hill, New Jersey, during the summers of 1994 and 1995, on implementations of IP and ATM inter-networking; he also worked at Bell Labs, Lucent Technologies, Holmdel, New Jersey, during the summer of 1996, on network traffic management. Since September 1997, Dr. Hou has been a Research Staff Member at Fujitsu Laboratories of America, Sunnyvale, California. His current research interests are in the areas of next generation Internet architecture, protocols, and implementations for differentiated and integrated services. Dr. Hou is a member of the IEEE, ACM, and Sigma Xi.

Zhi-Li Zhang received his B.S. degree in Computer Science from Nanjing University, China, in 1986 and his M.S. and Ph.D. degrees in Computer Science from the University of Massachusetts, Amherst, in 1992 and 1997, respectively. In 1997 he joined the Computer Science and Engineering faculty at the University of Minnesota, where he is currently an Assistant Professor. From 1987 to 1990, he conducted research in Computer Science Department at Århus University, Denmark, under a fellowship from the Chinese National Committee for Education. Dr. Zhang's research interests include computer communication and networks, especially the QoS guarantee issues in high-speed networks, multimedia and real-time systems, performance evaluation, queueing theory, applied probability theory, theory of computation and algorithms. He is a member of IEEE, ACM and INFORMS Telecommunication Section. Dr. Zhang received the National Science Foundation CAREER Award in 1997.