Novel approaches to end-to-end packet reordering measurement

Xiapu Luo and Rocky K. C. Chang
Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong, SAR, China
Email: {csxluo|csrchang}@comp.polyu.edu.hk

Abstract

By providing the best-effort service, the Internet Protocol (IP) does not maintain the same order of packets sent out by a host. Therefore, due to the route change, parallelism inside a switch, and load-balancing schemes, IP packets can be received in an order different from the original one. Such packet reordering events could cause serious performance degradation in TCP and UDP applications. As a result, a number of measurement methods have recently been proposed to enable any Internet host to detect packet reordering from itself to another host. However, these methods have encountered a number of practical difficulties, such as rate-limiting and filtering imposed on ICMP and TCP SYN packets. Moreover, some of the methods cannot detect packet reordering in all scenarios. In this paper we present three new methods for end-to-end packet reordering measurement. Since these methods are based on the TCP data channel, the probing and response messages will not be affected by any intermediaries on an Internet path. We have validated and tested the methods in 20 most common systems and implemented them in a tool called POINTER. We also present measurement results obtained from 200 websites in the Internet.

1 Introduction

Packet reordering, which was first identified in the pioneering work of Paxson [1], is still a common phenomenon in the Internet today [2, 3]. The packet reordering is the result of parallelism in network components, e.g., routers and switches, route instability, and load balancing mechanisms [2]. It is well known that packet reordering can adversely affect the performance of TCP and UDP based applications [4, 5, 6]. For example, a TCP flow may enter the fast retransmit state or timeout state if it misinterprets out-of-order delivery as packet losses [4]. To alleviate the effect on the TCP throughput, several proposals have been made to make TCP more robust to packet reordering events [7, 8, 9, 10, 11]. Furthermore, malicious packet reordering can be used to launched Denial-of-Service attacks [12].

Therefore, it is important to measure packet reordering [17] and to quantify the degree of reordering [13, 14]. Previous works have studied various characteristics of packet reordering, such as its frequency, magnitude, and its relationship with other factors. They can be categorized into two main classes—*passive measurement* [15] and *active measurement* [1, 2, 3, 5, 6, 16, 17, 18]. In this paper, we concentrate on the active measurement approaches, which can be further classified into two groups. The first is referred to as *bulk-transport based measurement* that exchanges a relatively large amount of real application traffic between many participating sites [1, 3, 5, 6, 16]. The second is *packet-train based measurement* that sends a train of probing packets, and the feedback information is then used to infer the presence of packet reordering [2, 17, 18].

The main advantage of the bulk-transport based measurement is their ability to measure the impact of packet reordering on real applications and on different transport protocols. Moreover, they can observe the packet reordering phenomena in both the forward-path and backward-path by examining traffic traces in different directions. However, these approaches require coordinations among various participating sites, which obviously cannot be done easily for an arbitrarily large network scale. The packet-train based measurement, on the other hand, allows any Internet host to measure packet reordering on the paths between itself and any other host. Moreover, this approach usually requires only a small number of packets to conduct the measurement.

The focus of this paper is on the packet-train based measurement methods, which unfortunately still suffer from a number of practical limitations. First, the ICMP and TCP packets used in the ICMP-based approach [2] and the TCP SYN Test [17] are often rate-limited or even filtered by firewalls [19]. Second, both the Dual Connection Test [17] and Tulip [18] rely on the assumption that the ID field in the IP header (IPID) increases monotonically across TCP

connections for the same server. This assumption, unfortunately, does not hold in some popular systems, such as Linux and OpenBSD. Third, the Single Connection Test [17] may yield inaccurate results due to the delayed acknowledgment algorithm. The Dual Connection Test also suffers from inaccurate results in the presence of load balancing schemes. Fourth, both the TCP Data Transfer Test [17] and the ICMP-based method [2] cannot distinguish all packet reordering scenarios. Moreover, all the aforementioned methods detect *end-to-end* packet reordering. That is, two packets can arrive inorder even though they may be reordered in an even number of times on the path.

In this paper we propose three new methods to end-to-end packet reordering measurement, which can detect all four reordering cases: *no-reordering*, *forward-path reordering*, *backward-path reordering*, and *dual-path reordering*. Since they use TCP data as probing messages in a single connection, the probing messages and the responses will not be filtered by firewalls or routers. Moreover, each measurement is conducted in one TCP session; therefore, the measurement result will not be affected by a content-blind load balancing scheme. To reduce the impact on the normal network traffic, each method injects only a minimal number of packets into the network, and the size of the response packets is also small.

The three methods differ from each other only in the mechanisms to trigger the required responses from the remote host, catering for the diverse TCP implementations in the Internet. Another important feature of our methods is that they only rely on the TCP sliding window protocol, which is supported by all TCP variants (e.g. TCP Tahoe, TCP Reno, TCP NewReno, etc.) [21, 22]. Moreover, our approaches make use of customized receiving window size and MSS (Maximum Segment Size) to control the number and size of response packets. Furthermore, we have employed a timeout mechanism to discard suspicious samples as a result of packet losses. We have implemented the three methods in a tool called POINTER, and have thoroughly tested and verified the approaches in a test-bed and 200 websites.

The rest of the paper is organized as follows. In section 2, we introduce the principles behind the three new approaches, including a short review on the TCP sliding window protocol. Based on the general approach, we detail in section 3 the three measurement methods. In section 4, we present the measurement results obtained from a test-bed and the Internet for the validation of the methods. Moreover, we have analyzed the correlation of packet reordering events on a backward path from two websites, and discovered that they share the same path where packet ordering occurs. We finally conclude this paper in section 5 with current work.

2 The fundamental principles of the new approaches

In the next subsection we first discuss some general fundamental principles for the design of a reordering detection scheme. The results presented there will then lead to a general requirement for designing a comprehensive and effective scheme to detect packet reordering. After that, we sketch our approach of meeting this requirement based on the TCP sliding window mechanism. In the rest of this paper, we refer the host that performs packet reordering measurement to a *client*, and the host on the other side of the TCP connection to a *server*.

2.1 The fundamental principles

Proposition 1. *Assume that during each probing session a client can trigger k distinguishable responses out of a set of n possible ones from a server. Then, n must be at least 3 in order to discriminate all the 4 packet reordering scenarios.*

Proof. Note that the k responses received from the server are *ordered* according to their times of generation. Therefore, the client can trigger a set of $P(n,k) = n!/(n-k)!$ k-response. In order to differentiate the 4 reordering scenarios, clearly $n \geq 3$, because $P(2,k) < 4$, and $P(n,k) > 4$ for some $k > 0$ when $n \geq 3$. □

Proposition 2. *Given that the smallest n is used, the minimum number of distinguishable responses from a server required for a complete reordering detection during each probing session is 2. Therefore, if one probing message will trigger at least one distinguishable response, then the client only needs to send 2 probing messages during each session.*

Proof. There are two possible cases for $n = 3$: $k = 2, 3$ for which $P(3,2) = P(3,3) = 6 > 4$. □

Corollary 1. *Consider that a client sends 2 probing messages—$M1$ first and then $M2$—to trigger a generation of 2 normal, distinguishable responses from a server, denoted by $R1$ and $R2$. Then, the scheme can distinguish all 4 reordering cases if $M1$ and $M2$ can trigger another distinguishable response $R3$, and, by definition, $R3$ is not a normal response.*

Proof. This is a direct consequence of Proposition 2, which requires one more distinguishable response for a complete reordering detection. □

The result of Corollary 1 gives us a hint in designing a comprehensive and effective reordering detection scheme. First of all, consider that the server responds with $R1$ and $R2$ upon receiving $M1$ and $M2$ in order. That is, if there is no forward-path reordering, $R1$ will be generated first

Table 1: Variables used in the TCP sliding window mechanism.

Name	Description
SND.UNA	Oldest unacknowledged sequence number (SN)
SND.NXT	SN of the next segment to be sent
SND.WND	Size of the send window
RCV.NXT	SN of the next segment to be received
RCV.WND	Size of the receive window
SEG.ACK	Acknowledgment number (AN) of a segment
SEG.SEQ	First SN of a segment
SEG.LEN	The length of a segment in bytes

and then followed by $R2$. Therefore, the order of receiving $R1$ and $R2$ on the client side can differentiate between the cases of no-ordering and backward-ordering.

To trigger the third response $R3$, we need an "erroneous event" taken place at the server. The novelty here is that we use the reordering of $M1$ and $M2$ to serve as the erroneous event. That is, if there is forward-path reordering, the receiver will be compelled to send $R3$ and another normal response ($R1$ or $R2$). For instance, the receiver responds with $R3$ first and then $R1$. Then the order of receiving them by the client can differentiate between the cases of forwarding-reordering and dual-path reordering.

Based on the discussion above, we only need a pair of probing messages $M1$ and $M2$ to detect all four reordering cases. Note that we have so far assumed no packet losses and a reasonable amount of latency between the message arrivals at the server. Moreover, the server's responses to $M1$ and $M2$ only depend on the information inscribed in the messages and the relative order of the two messages received by the server. We will address the effect of packet losses and the solution to them in section 3.4. In the next section, we first present several possibilities of selecting $M1$ and $M2$ based on the TCP sliding window mechanism.

2.2 The TCP-based probing message pair

Since our methods are based on the TCP data channel, we first review the TCP sliding window algorithm. We adopt the notations in Table 1 [20] for the following discussion. A TCP sender uses a *send window* to control the transmission of segments, while a TCP receiver maintains a *receive window* for receiving segments from the sender. When a TCP receiver is in the ESTABLISHED state and a segment arrives, it will process the segment according to the following order [20]: (1) check the SN, (2) check the RST bit, (3) check security and precedence, (4) check the SYN bit, (5) check the AN, (6) check the URG bit, (7) process the segment text, and (8) check the FIN bit.

In our proposed methods, a client sends a probing message pair to induce two TCP data segments from the server when there is no packet reordering on the forward path, i.e., the probing message pair passes all eight steps. On the other hand, the reordering of the probing message pair, which is perceived as an erroneous event at the server, will induce the transmission of a pure ACK and a data segment. There are altogether three ways of generating the pure ACK.

First of all, a pure ACK can be generated as a result of not passing step 1 where a TCP receiver performs a *sequence number check* (SNC). The purpose of the SNC is to determines whether the received segment's SN is acceptable according to Eq. (1) or Eq. (2). If the segment fails the SNC (an erroneous event), some TCP implementations will drop the segment and return a pure ACK whose value is specified in the second part of Eq. (3).

A pure ACK can also be generated as a result of not passing step 5, where a TCP receiver performs an *acknowledgment number check* (ANC) based on Eq. (4). If the ACK is acceptable, the receiver will update its send window by setting SND.UNA = SEG.ACK. Otherwise, if SEG.ACK $<$ SND.UNA, i.e., a duplicate ACK, the receiver will ignore it. Moreover, if SEG.ACK $>$ SND.NXT (an erroneous event), i.e., acknowledging bytes that have not been sent, some TCP implementations will drop the segment and send back a pure ACK whose value is specified in the second part of Eq. (3).

Finally, if a TCP implementation does not respond to both failed SNC and failed ANC, a pure ACK can still be induced by sending a out-of-ordered segment. Accordingly, we have designed three different probing message pairs based on the server's responses discussed above. The corresponding methods are referred to as ACM (*ACknowledgment based Measurement*) for the response to a failed SNC, $SAM1$ (*Sequence number and ACK based Measurement*) for the response to a failed ANC, and $SAM2$ for the response to a out-of-order segment.

$$\texttt{RCV.NXT} \leq \texttt{SEG.SEQ} < \texttt{RCV.NXT} + \texttt{RCV.WND}. \quad (1)$$

$$\texttt{RCV.NXT} \leq \texttt{SEG.SEQ} + \texttt{SEG.LEN} - 1 < \texttt{RCV.NXT} + \texttt{RCV.WND}. \quad (2)$$

$$\texttt{SEG.SEQ} = \texttt{SND.NXT},\ \texttt{SEG.ACK} = \texttt{RCV.NXT}. \quad (3)$$

$$\texttt{SND.UNA} < \texttt{SEG.ACK} \leq \texttt{SND.NXT}. \quad (4)$$

We have tested 20 common systems to observe their responses to the failed SNC and failed ANC. As shown in Table 2, the majority of the tested systems responded to unacceptable ACKs. Therefore, the ACM method can be applied to detect packet reordering on the paths to these systems. Although the Linux systems do not respond to unacceptable ACKs, they respond to unacceptable SNs. Thus, the SAM1 method can be applied to them. Finally, the SAM2 method can be applied to the VM and HP-UX systems which ignore both unacceptable ACKs and unacceptable SNs.

Table 2: Popular operating systems and the new measurement methods.

Operating systems	Response to unacceptable SNs	Response to unacceptable ACKs	Measurement Methods
NT4/Win98, Win2000, WinXP, Win2003, MaxOSX, NetWare, SCO UNIX, NetBSD, AIX, OS/2, IRIX, Tru64, FreeBSD, Solaris 9 , Solaris 8, Solaris*, OpenVMS	Not tested	Send a pure ACK	ACM
Linux	Send a pure ACK	No response	SAM1
VM and HP-UX	No response	No response	SAM2

3 Three new measurement methods

The three measurement methods use different pairs of TCP data segments to induce from the server two data segments in the absence of reordering in the forward path, and one pure ACK and one data segment in the presence of reordering in the forward path. Moreover, they do not assume any specific TCP-based application running at the server. The only assumption is that the server replies with at least several hundreds of bytes of data in response to the client's probing messages, which is clearly possible for most popular TCP-based services. For example, the HTTP `GET` can be used to fetch a sufficiently large web object from a web server. The `GET` command in most FTP clients can be used to download a suitable file from a public FTP server. In the following, we illustrate the three methods using HTTP, but they can be easily adapted for FTP, POP3, or other TCP applications.

The probing session can be carried out at any time in the ESTABLISHED state. To keep the following discussion simple, however, we start the probing procedure immediately after the TCP three-way handshake is completed. The notations used in the remaining of this paper are summarized in Table 3, and the segments involved in the 3-way handshaking are given in Table 4. Note that these segment exchanges are the same for all three methods. C_0 and S_0 are the initial SNs for the client and server, respectively. In $TC1$, the client initiates a SYN segment with a *small* advertised window size of W_C. The main function of the small window size is to ensure that the server will adopt W_C for its send window size. Therefore, even before issuing the HTTP request in $TC2$, the client is able to predict correctly that the server will return one data segment and the size of the TCP payload (W_C), provided that the size of the requested document is larger than W_C. The packet sequence concerned is shown in (3)-(5) in Table 4. As we shall see later, the ANs in the probing message pair are the same for all three methods, because they all exploit the predicability of the server's payload through the small W_C. However, they differ in their SNs and the TCP payloads.

3.1 The ACM method

Recall that the *ACM* method is based on the server's response to the recipient of an unacceptable ACK. In this case, the server is expected to drop the corresponding segment and to respond with an ACK. Figs. 1(a) and 1(c) first show the packet sequences in the absence of packet reordering in the forward path. The probing message pair used in this method are two back-to-back, pure ACKs ($TC3$ and $TC4$). Notice from Table 5 that $TC3$ acknowledges the receipt of $DataS0$, whereas $TC4$ acknowledges a "yet-to-receive" data segment. Because of the predicability of the payload length in $DataS1$, the client is able to send $TC4$ with a correct AN immediately after $TC3$.

If these two ACKs arrive at the server in order (Figs. 1(a) and 1(c)), the server will expectedly transmit two data segments $TS4$ and $TS5$, one after the other, which contain the requested document. The order that $TS4$ and $TS5$ are received at the client can then be used to differentiate between the no-ordering and backward-path ordering cases.

However, if these two ACKs are reordered in the forward path (Figs. 1(b) and 1(d)), the server will discover that $TC4$ is an unacceptable ACK, because `SND.UNA` $= S_1$ and `SND.NXT` $= S_1 + W_C$ at the time of receiving the ACK. In this case, the server, which is implemented under the first group of systems in Table 2, is expected to drop $TC4$ and to send back a pure ACK $TS4$. When $TC3$, the first ACK, later arrives, the server will send out the requested data in $TS5$. Thus, the order of receiving $TS4$ and $TS5$ can be used to differentiate between the forward-path and dual-path reordering cases.

To ensure a proper working of this method, the size of the requested document must be at least $3 \times W_C$ bytes. Since the probing message pair used in the ACM method are pure ACKs, the active probings have a minimal impact on the normal network traffic. Furthermore, the server's response involves at most $3 \times W_C$ bytes of data ($DataS0$, $DataS1$ and $DataS2$) for differentiating between the no-reordering and backward-path reordering cases. For the other two cases, the amount of data required is even less: $2 \times W_C$ bytes ($DataS0$ and $DataS1$).

Table 3: Notations used in the description of the new measurement methods.

Notations	Description
$TCi, (i = 1, \ldots, 4)$	The ith segment dispatched by the client
$TSi, (i = 1, \ldots, 5)$	The ith segment dispatched by the server
SYN	The TCP SYN packet sent by the client
SYN-ACK	The TCP SYN/ACK packet responded by the server
$CmdCi, (i = 1, 2, 3)$	ith HTTP request sent by the client
$X_i, (i = 1, 2, 3)$	Size of $CmdCi$ in bytes
$ACKCi, (i = 1, 2)$	ith pure ACK sent by the client in the ESTABLISHED state
$ACKSi, (i = 0, 1, 2)$	ith pure ACK sent by the server in the ESTABLISHED state
$DataSi, (i = 0, 1, 2)$	ith data segment containing the HTTP response sent by the server
W_C	The size of the client's advertised window
W_S	The size of the server's advertised window

Table 4: The packet sequence common for all three methods.

No.	Segment	Sequence Number	Acknowledgment Number	Segment Type	Payload Length
1	$TC1$	C_0	0	SYN	0
2	$TS1$	S_0	C_1=C_0+1	SYN/ACK	0
3	$TC2$	C_1	S_1=S_0+1	HTTP request ($CmdC1$)	X_1
4	$TS2$	S_1	C_2=C_1 + X_1	Pure ACK ($ACKS0$)	0
5	$TS3$	S_1	C_2	HTTP reply ($DataS0$)	W_C

3.2 The SAM1 method

The second method is based on the server's response to unacceptable SNs which can be applied to Linux systems. In this case, a Linux server drops the corresponding segment and responds with an ACK. Figs. 2(a) and 2(c) show the packet traces when there is no packet reordering in the forward path. The probing message pair in this case are $TC3$ and $TC4$ which are a second HTTP request and a pure ACK, respectively. Similar to the case for the ACM method, $TC4$ acknowledges a "yet-to-receive" data segment from the server. The purpose of sending the second HTTP request message, on the other hand, will be clear from the following explanation.

From the parameters in Table 6, it is not difficult to see that the server will send two more data segments, each in the size of W_C, in response to the probing message pair when there is no forward-path reordering. Therefore, the order of receiving the two data segments can be used to differentiate between the no-ordering and backward-path reordering cases.

On the other hand, if the probing message pair is reordered (Figs. 2(b) and 2(d)), $TC4$ will fail the SNC on the server side, because the SN falls outside the legal range according to Eq. (2). Specifically, from Table 6, $TC4$'s SN is given by `SEG.SEQ` $= C_2 + X_2 + W_S - 1$ ($X_2 > 1$) and `SEG.LEN` $= 0$ (0 payload length). On the other hand, at the time of receiving $TC4$, the server's states as a TCP receiver are given by `RCV.NXT` $= C_2$ and `RCV.WND` $= W_S$. According to Eq. (2), $TC4$'s SN is therefore illegal, i.e., `SEG.SEQ` $>$ `RCV.NXT` $+$ `RCV.WND`. As discussed before, the server is therefore expected to discard this pure ACK and to respond with an ACK ($ACKS1$). When $TC3$ that contains a correct SN arrives, the server will clearly respond with $DataS1$. Therefore, the order of receiving $ACKS1$ and $DataS1$ can differentiate between the backward-path reordering and dual-path reordering cases. Thus, unlike the ACM method that sends out two pure ACKs, this method sends a second HTTP request, so that the TC4's SN is different from TC3's.

Same as the ACM method, the total size of the requested document must be at least $3 \times W_C$ to ensure a proper working of this method. In terms of the overhead, the size of the probing message pair for the $SAM1$ method is larger than that in the ACM method by X_2 bytes, the size of the second HTTP request message. However, the maximum amount of HTTP data returned by the server is the same for both methods, i.e., $3 \times W_C$ bytes.

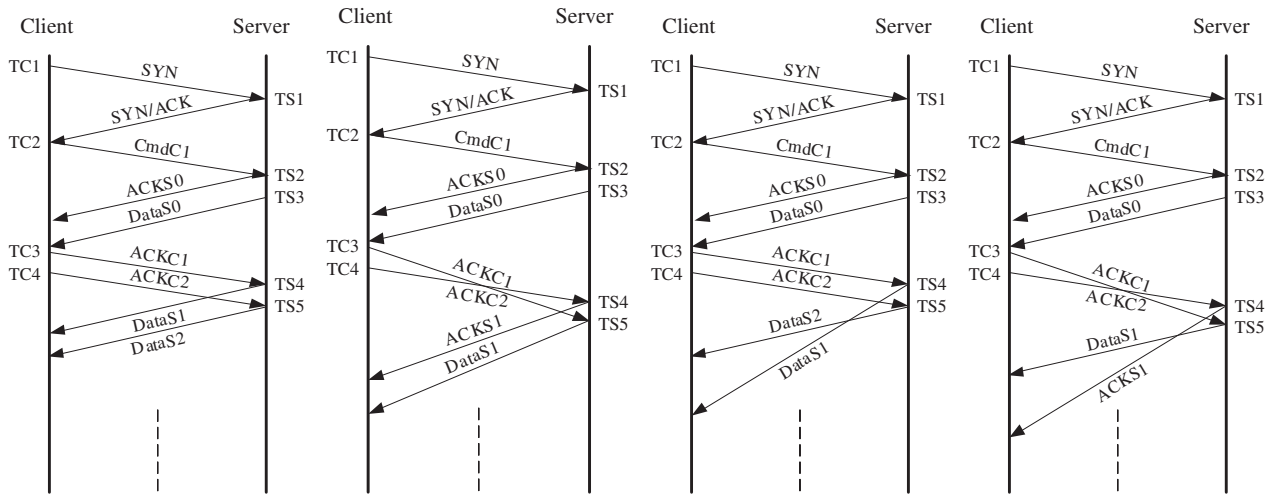


Figure 1: Packet sequences for the ACM method under the four packet reordering scenarios.

Table 5: The packet sequence for the ACM method (continued from Table 4).

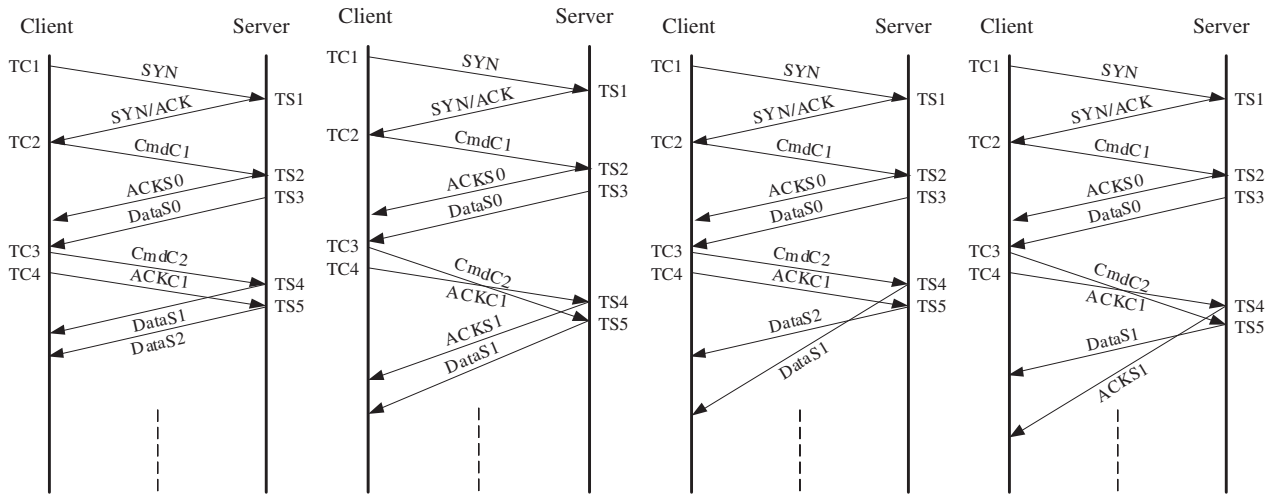
No.	Segment	Sequence Number	Acknowledgment Number	Segment Type	Payload Length
The probing message pair					
6	$TC3$	C_2	$S_2 = S_1 + W_C$	Pure ACK ($ACKC1$)	0
7	$TC4$	C_2	$S_3 = S_1 + 2W_C$	Pure ACK ($ACKC2$)	0
Server's responses in the absence of packet reordering in the forward path					
8	$TS4$	S_2	C_2	HTTP reply ($DataS1$)	W_C
9	$TS5$	S_3	C_2	HTTP reply ($DataS2$)	W_C
Server's responses in the presence of packet reordering in the forward path					
8'	$TS4$	S_2	C_2	Pure ACK ($ACKS1$)	0
9'	$TS5$	S_2	C_2	HTTP reply ($DataS1$)	W_C

3.3 The SAM2 method

Recall from the beginning of this section that the TCP implemented in HP-UX and VM do not respond to both failed SNC and failed ANC. The $SAM2$ method described in this section is designed to cater for this set of TCP implementations. Figs. 3(a) and 3(c) illustrate the packet sequences for this method when there is no packet ordering in the forward path. After receiving the first data segment $TS3$, the client sends out the probing message pair $TC3$ and $TC4$.

Similar to the $SAM1$ method, the probing message pair uses different SNs. However, notice from Table 7 that the SNs are offset by Off and $2 \times Off$ respectively, where Off is a small value. That is, both messages contain unexpected but *acceptable SN* (out-of-order segments). Therefore, the recipient of $TC3$ will enable the server to advance its send window and to send a data segment. When $TC4$ later reaches the server, this segment acknowledges all the outstanding data sent by the server (AN = `SND.NXT` = $S_1 + 2W_C$); therefore, the server will also advance its send window. As a result, the server will reply with another data segment $TS5$. The server therefore responds with a total of two data segments whose order of arriving at the client can be used to differentiate between the no-ordering and backward-path reordering cases.

If the two messages are reordered, as shown in Figs. 3(b) and 3(d), the message $TC4$ contains an *unacceptable ACK*. However, the HP-UX and VM server will ignore the illegal AN, but, similar to other TCP implementations, it will definitely respond to the out-of-ordered segment with a duplicate ACK, $TS4$. There is a minor difference between the HP-UX and VM systems though. $TS4$ contains a SN of S_1 for an HP-UX server) whereas $TS4$ contains a SN of $S_1 + W_C$ for a VM server, and both contain an AN of C_2. However, this difference does not affect our measure-



(a) No reordering (b) Forward-path reordering (c) backward-path reordering (d) Dual-path reordering

Figure 2: Packet sequences for the $SAM1$ method under the four packet reordering scenarios.

Table 6: The packet sequence for the $SAM1$ method (continued from Table 4).

No.	Segment	Sequence Number	Acknowledgment Number	Segment Type	Payload Length
The probing message pair					
6	$TC3$	C_2	$S_2 = S_1 + W_C$	HTTP request ($CmdC2$)	X_2
7	$TC4$	$C_2 + X_2 + W_S - 1$	$S_3 = S_1 + 2W_C$	Pure ACK ($ACKC1$)	0
Server's responses in the absence of packet reordering in the forward path					
8	$TS4$	S_2	$C_2 + X_2$	HTTP reply ($DataS1$)	W_C
9	$TS5$	S_3	$C_2 + X_2$	HTTP reply ($DataS2$)	W_C
Server's responses in the presence of packet reordering in the forward path					
8'	$TS4$	S_2	C_2	Pure ACK ($ACKS1$)	0
9'	$TS5$	S_2	$C_2 + X_2$	HTTP reply ($DataS1$)	W_C

ment method, and in fact it can be used to remotely identify these two systems [19]. On the other hand, the second message $TC3$ contains an *acceptable* AN of $S_1 + W_C$. In this case, the server will advance its `SND.UNA` and reply with a data segment $TS5$ with $SN = S_1 + W_C$ and $AN = C_2$. Therefore, the order of receiving the ACK and data segment from the server can be used to differentiate between the backward-path reordering and dual-path reordering cases.

The size of the requested document must again be at least $3 \times W_C$ to ensure a proper working of the method. It is not difficult to show that the maximum amount of data required by the $SAM2$ method is the same as that for the $SAM1$ method, which is only $3 \times W_C$ bytes. The size of the probing packets in the $SAM2$ method is larger than that of the ACM method by the size of $CmdC3$.

3.4 Packet losses control

Packet losses in the probing packets and responses will clearly affect the measurement results. Some of them can be easily detected by observing abnormal responses from the server, and they merely delay the observation period. On the other hand, other kinds of packet losses may introduce a bias in the measurement results, such as when the responses are lost and then retransmitted. Due to the limited space, we only describe the solutions for the ACM method under the second type of packet losses, which can be easily modified for the $SAM1$ and $SAM2$ methods.

To illustrate the problem, consider the scenario in Fig. 4(a) where there is backward-path reordering and the segment $TS5$ is lost. If the client interprets the segment $TS4$ and the retransmitted segment $TS5$ as the server's responses to its probing messages, it will arrive at a false conclusion that there is no packet reordering.

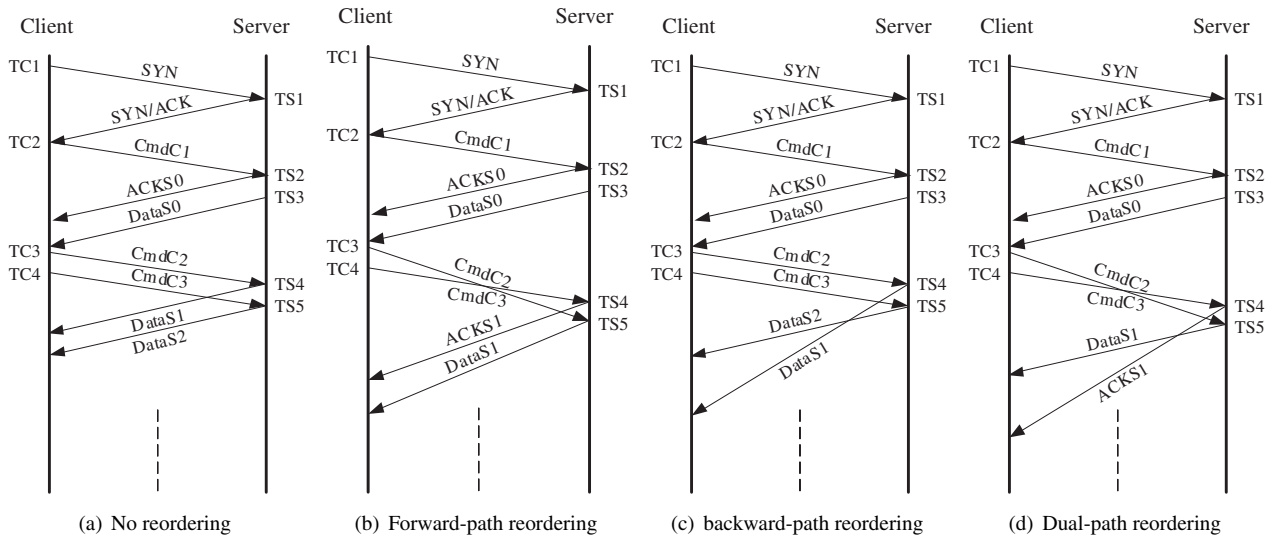


Figure 3: Packet sequences for the $SAM2$ method under the four packet reordering scenarios.

Table 7: The packet sequence for the $SAM2$ method (continued from Table 4).

No.	Segment	Sequence Number	Acknowledgment Number	Segment Type	Payload Length
The probing message pair					
6	$TC3$	$C_2 + Off$	$S_2 = S_1 + W_C$	HTTP request ($CmdC2$)	X_2
7	$TC4$	$C_2 + 2Off$	$S_3 = S_1 + 2W_C$	HTTP request ($CmdC3$)	X_3
Server's responses in the absence of packet reordering in the forward path					
8	$TS4$	S_2	C_2	HTTP reply ($DataS1$)	W_C
9	$TS5$	S_3	C_2	HTTP reply ($DataS2$)	W_C
Server's responses in the presence of packet reordering in the forward path					
8'	$TS4$	S_1 *or* $(S_1 + W_C)$	C_2	Pure and duplicate ACK ($ACKS1$)	0
9'	$TS5$	S_2	C_2	HTTP reply ($DataS1$)	W_C

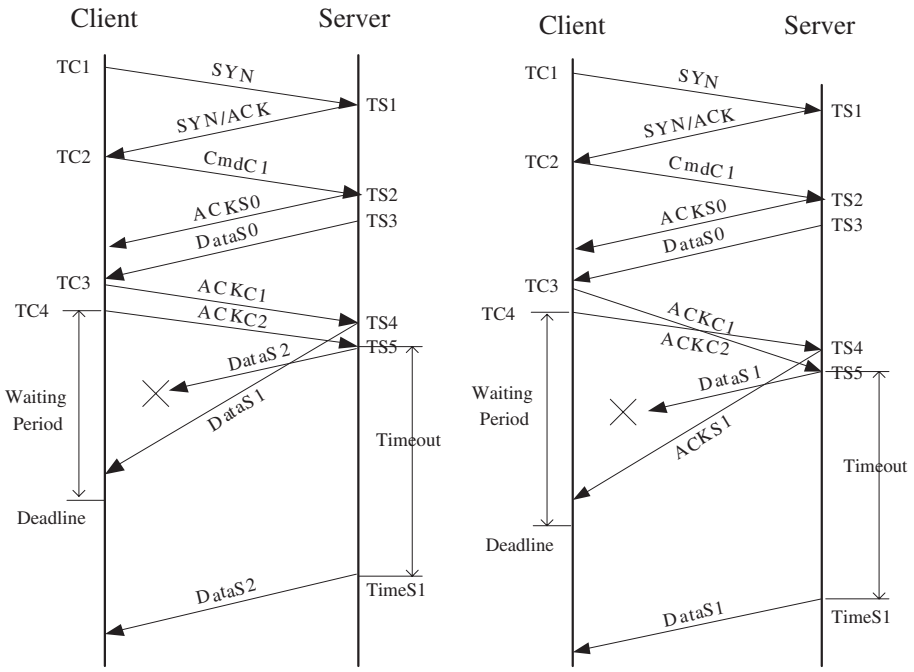
To remedy this problem, we impose a deadline for receiving responses from the server. As illustrated in Fig. 4(a), after the client sends $TC4$, it will not accept any response arriving after the deadline and declare the measurement unsuccessful. In another example, Fig. 4(b) depicts that the dual-path reordering would be mistaken for the forward-path reordering if the deadline mechanism were not used. It is obvious that the server can only use the timeout mechanism to retransmit the lost packet because it is impossible to trigger the fast retransmit mechanism. We therefore set the deadline to $1.5 \times \overline{RTT}$, where $\overline{RTT}$ is the mean value of RTT between the client and the server.

4 POINTER and the measurement results

We have developed a measurement tool called POINTER (*Packet reOrderINg tesTER*) that implements the ACM, SAM1, and SAM2 methods. The implementation uses the packet filter API available from [27] to block the TCP RST packets generated by the local host, and the WinPcap library [28] to generate customized TCP packets. With POINTER, we can validate the three measurement methods on a test-bed and in the Internet, and analyze the packet reordering statistics.

4.1 Measurement results from a test-bed

We have validated the ACM and $SAM1$ methods on a test-bed environment. As shown in Fig. 5, the test-bed consists of two servers, a POINTER client, and an Iperf host. The Iperf client is responsible for generating background traffic. Apache web servers are running on both the Linux server (Linux 2.4.20-8) and Windows 2000 server for validating the SAM1 method and the ACM method, respec-



(a) A packet loss in the presence of backward-path reordering

(b) A packet loss in the presence of dual-path reordering

Figure 4: Handling packet losses in the ACM method.

tively. Both the servers and the POINTER client use tcpdump or Windump to capture the probing packets and response packets. The servers and clients are connected by a FreeBSD-based router running Dummynet [31] to simulate multipath between a client and a server.

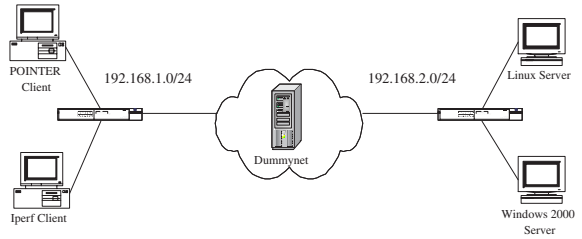


Figure 5: The network configuration of the test-bed.

By setting 20 pipes between the subset of 192.168.1.0/24 and 192.168.2.0/24 in the Dummynet, we can emulate the four packet reordering scenarios on the test-bed. We let each method keep on probing the server until it has detected 100 packet reordering events in the probing packet pairs. The measurement results are then compared with the packet traces. Our findings show that all the detection results produced by the ACM and SAM1 methods are correct for all test cases. Moreover, we have validated the two methods on other systems available in our department, including WinXP, Netware, FreeBSD, and Solaris. To validate the methods with the remaining systems listed in Table 2, we have conducted the experiments with the WWW servers in the Internet, to be presented next.

4.2 Measurement results from the Internet

We have made used of the service provided by NetCraft to identify the systems running in the WWW servers [29]. We have also obtained additional information, e.g. from [30], to ensure correct system identification. We have altogether selected around 200 websites, part of which are randomly chosen from Yahoo random URL database [32] suggested by [17] and part of which include the popular websites, e.g. Microsoft, Apple, Hotmail, Yahoo, Google, etc. For more popular systems, such as Linux and Windows, we have tested 10 servers for each system. However, for OS/2 and OpenVMS, we could just locate a handful of websites, such as `www.os2.org` and `www.openvms.compaq.com`. For both Solaris 2 and Solaris 7, NetCraft has identified them to be the same, and labelled them by Solaris. Therefore, our tests may not have covered both Solaris 2 and Solaris 7, which are marked with $*$ in Table 2.

For each web server, the validation process consists the following steps:

1. carry out the measurement process in no reordering case, i.e. sending out the segments $TC3$ and then

$TC4$, and recording the responses $TS4$ and $TS5$ from the server.

2. carry out the measurement process in forward reordering case, i.e. sending out the segments $TC4$ and then $TC3$, and recording the responses $TS4^{'}$ and $TS5^{'}$ from the server.

3. If the responses, $\{TS4, TS5, TS4^{'}, TS5^{'}\}$ fulfill the requirement of Corollary 1, then the packet reordering in the path from the monitoring point to remote host can be measured.

The measurement results obtained from these websites have confirmed the correctness of the three methods and produced the measurement results in Table 2. For an arbitrary accessible web server, we do not need to first identify the OS type of remote host before measuring packet reordering. We can conduct the validation process of ACM, SAM1 and SAM2 sequently to find out which method is proper.

Moreover, we have conducted more measurements from 100 WWW servers among the 200 websites used in the above validation process, and obtained 500 measurement results for each server. Fig. 6 shows the empirical cumulative distribution function (CDF) of the forward-path reordering rate and backward-path reordering rate. The measured dual-path reordering rate is zero for all cases. The rate is given by the percentage of measurements that indicate the presence of packet reordering. Then we rank all the servers according to a nondecreasing order of their reordering rates, and obtain the CDF. The results show that more than 35% of the paths experienced forward-path reordering at least once, and backward-path reordering was observed on about 10% of the paths. Moreover, the forward-path reordering was more prevalent than the backward-path reordering in terms of the percentage of reordering events. The forward-path reordering rate could go up as high as 0.4, while that for the backward-reordering was only 0.3.

4.3 Correlation of packet reordering events

In this section we analyze the correlation of packet reordering events. To this end, we have conducted measurements on `www.applecomputer.com` and `kidsafe.apple.com` over a 2-month period. These two sites have been found to have more than 10% forward-path reordering rate according to [17]. In each measurement, the client first sends out the probing pair to `www.applecomputer.com` and then another probing pair to `kidsafe.apple.com`. The time gap between sending the two probing pairs is very short. There is a random delay (between 1 second and 2 seconds) between two consecutive measurements. We have conducted 59 sets of measurements and each set consists of 100 samples.

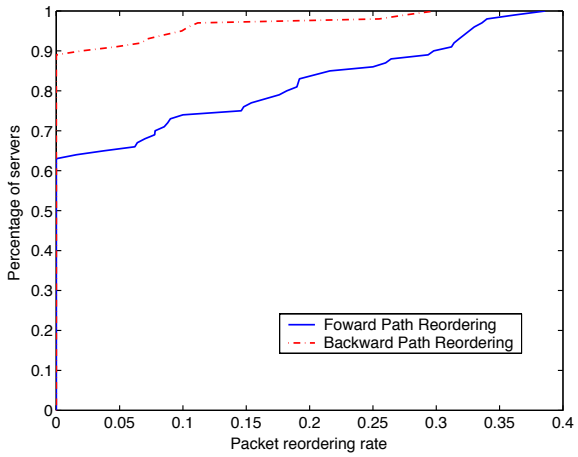


Figure 6: The empirical cumulative distribution functions of the packet reordering rates.

Unlike the previous results for the 100 websites, the results here indicate that there are more backward-path reordering events than the forward-path ones. Fig. 7(a) shows the mean backward-path reordering rates to both servers obtained from the 59 sets of measurement. Each point is an average of the 100 samples within a set of measurement. The rates for both servers are very similar in many sets of samples. Moreover, the overall means rates for both servers are approximately equal to 16.7%. To probe into the issue further, we compute the pair-difference test statistic for the 59 sets of data [33], and find that with a 99% confidence interval the backward-path reordering rates are similar for the two servers.

Next, we study and compare the time series of the measurements obtained for the two servers. We use $TR_{i,j}^{S_k}$, $i = 1, \ldots, 59$, $j = 1, \ldots, 100$, $k = 1, 2$, to denote the presence/absence of backward-path reordering in the corresponding sample. S_1 refers to `www.applecomputer.com` and S_2 refers to `kidsafe.apple.com`. $TR_{i,j}^{S_k} = 1$ in the presence of backward-path reordering; otherwise, $TR_{i,j}^{S_k} = 0$. We compute the autocorrelation value for the ith set of data according to [34]

$$\sum_{j=1}^{99-m} TR_{i,j}^{S_k} TR_{i,j+m}^{S_k},\ m = 0, \ldots, 99.$$

We then compute an average of the 59 autocorrelation values for each m and the results are depicted in Fig. 7(b). They first show that the correlation between the reordering events occurring at different times is not strong. Moreover, the two autocorrelation plots are so similar that it is very difficult to distinguish from each other.

Besides the autocorrelation function, we define a *pair-*

reorder index (PI) by

$$PI(i) = \frac{\sum_{j=1}^{100}(TR_{i,j}^{S_1} \,\&\, TR_{i,j}^{S_2})}{(\sum_{j=1}^{100}(TR_{i,j}^{S_1} \mid TR_{i,j}^{S_2}))/2},\ i = 1, \ldots, 59,$$

to measure the likelihood that two packet reordering events would occur within a measurement. The notations $\&$ and $|$ refer to the logical AND and logical OR operators, respectively. Therefore, the higher the index is, the higher the likelihood that backward-path reordering would occur in the paths from both servers to the POINTER client. Fig. 7(c) depicts the $PI(i)$ values computed from the 59 sets of experiments. The mean value of $PI(i)$ over the 59 sets is 15.5%, and 52 sets have nonzero $PI(i)$ values. This result, together with the results presented in Fig. 7(a) and Fig. 7(b), strongly suggest that the two servers shared the same part of the path to the POINTER client, where packet reordering occurred. Moreover, these three sets of statistical analysis can be applied to study the correlation of other Internet path statistics.

5 Conclusions and current work

In this paper we have presented three novel methods for end-to-end packet reordering measurement—ACM, $SAM1$ and $SAM2$. Unlike the previous approaches, we have designed the probing messages based on the TCP data channel, thus solving the practical problems of going through routers and other intermediaries on the Internet paths. Moreover, the probing message pair is carefully crafted, so that the client can predetermine the returned responses which can be used to confirm whether there is packet reordering on the forward path. Moreover, the order of the arrival of the two response packets is used to confirm whether there is packet reordering on the backward path. Thus, the methods can detect all four packet reordering scenarios. The amount of data used is also kept to a minimum.

We have implemented the three methods in POINTER and validated the methods in 20 most common systems in both a test-bed and the Internet. We are now in the process of making the tool available in different platforms. With POINTER, one can detect packet reordering on any path in the Internet. Moreover, the tool can be used to study other important issues, such as the correlation of packet reordering events presented in this paper. Furthermore, we are in the process of improving these three methods and conducting a much larger-scale measurement study on the prevalence of packet reordering in the Internet today.

Acknowledgments

The work described in this paper was partially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. PolyU 5080/02E) and a grant from the Areas of Excellence Scheme established under the University Grants Committee of the Hong Kong Special Administrative Region, China (Project No. AoE/E-01/99). We thank Mr. Kent Leung and his colleagues in the ITS office for their help in conducting the Internet measurement. We also thank the anonymous reviewers for their useful comments.

References

[1] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3), June 1999.

[2] C. Partridge, J. Bennett, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking*, 7(6), December 1999.

[3] L. Gharai, C. Perkins, and T. Lehman. Packet Reordering, High Speed Networks and Transport Protocol Performance. In *Proc. IEEE ICCCN*, October 2004.

[4] M. Laor and L. Gendel. The effect of packet reordering in a backbone link on application throughput. *IEEE Network*, 7(6), September 2002.

[5] D. Loguinov and H. Radha. Measurement study of low-bitrate internet video streaming. In *Proc. ACM Internet Measurement Workshop*, November 2001.

[6] X. Zhou and P. Mieghem. Reordering of IP packets in Internet. In *Proc. Passive and Active Measurement*, April 2004.

[7] M. Allman and E. Blanton. On making TCP more robust to packet reordering. *ACM Computer Communication Review*, 32(1), January 2002.

[8] S. Floyd, M. Zhang, B. Karp, and L. Peterson. RR-TCP: A reordering-robust tcp with DSACK. In *Proc. IEEE ICNP*, November 2003.

[9] J. Lee, C. Lim, S. Bohacek, J. Hespanha, and K. Obraczka. TCP-PR: TCP for persistent packet reordering. In *Proc. IEEE Conf. Distributed Computing Systems*, May 2003.

[10] C. Ma and K. Leung Improving TCP robustness under reordering network environment. In *Proc. IEEE GLOBECOM*, November 2004.

[11] A. Sathiaseelan and T. Radzik Improving the Performance of TCP in the Case of Packet Reordering. In *Proc. IEEE HSNMC*, June 2004

[12] I. Aad, J. Hubaux, and E. Knightly. Denial of Service Resilience in Ad Hoc Networks. In *Proc. ACM MobiCom*, September 2004.

[13] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, and J. Perser. Packet reordering metric for IPPM. draft-ietf-ippm-reordering-05.txt, Internet Draft, IETF, 2004.

[14] A. Bare, T. Banka, and A. Jayasumana. Metrics for degree of reordering in packet sequences. In *Proc. IEEE Conf. Local Computer Networks*, November 2002.

[15] C. Diot, J. Kurose, J. Jaiswal, G. Iannaccone, and D. Towsley. Measurement and classification of out-of-sequence packets in a tier-1 ip backbone. In *Proc. IEEE Infocom*, April 2003.

[16] V. Paxson. Automated packet trace analysis of TCP implementations. In *Proc. ACM SIGCOMM*, November 1997.

[17] J. Bellardo and S. Savage. Measuring packet reordering. In *Proc. ACM Internet Measurement Workshop*, November 2002.

[18] D. Wetherall, R. Mahajan, N. Spring, and T. Anderson. User-level Internet path diagnosis. In *Proc. ACM Symp. Operating Systems Principles*, October 2003.

[19] R. Spangler. Analysis of remote active operating system fingerprinting tools. http://www.packetwatch.net, May 2003.

[20] J. Postel. Transmission control protocol. RFC 793, IETF, September 1981.

[21] J. Padhye and S. Floyd. On inferring TCP behavior. In *Proc. ACM SIGCOMM*, August 2001.

[22] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communication Review*, 26(3), July 1996.

[23] V. Paxson and M. Allman. Computing TCPs Retransmission Timer. RFC 2988, IETF, November 2000.

[24] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the Internet http://www.icir.org/tbit, December 2004.

[25] TCP Tunable Parameters. Solaris Reference Manual http://docs.sun.com/app/docs/doc/816-0607/6m735r5g6.

[26] Cross-Referencing Linux. http://lxr.linux.no.

[27] Packet filtering reference, platform sdk. http://msdn.microsoft.com.

[28] Winpcap, the free packet capture architecture for Windows. http://winpcap.polito.it.

[29] Netcraft Ltd. http://uptime.netcraft.com/up/accuracy.html.

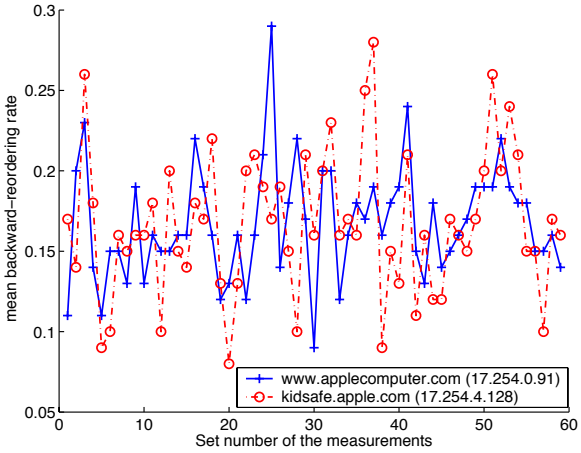
[30] VM/ESA based WWW servers. http://vm.cfsan.fda.gov/vmcms.html.

[31] L. Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, January 1997.

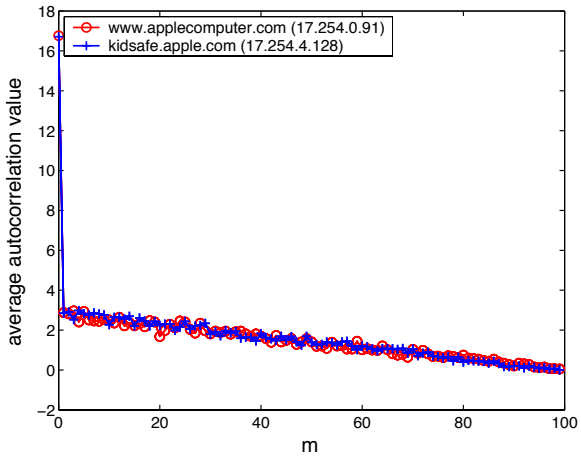
[32] Yahoo! Inc. Random Yahoo! Link. http://random.yahoo.com/bin/ryl.

[33] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

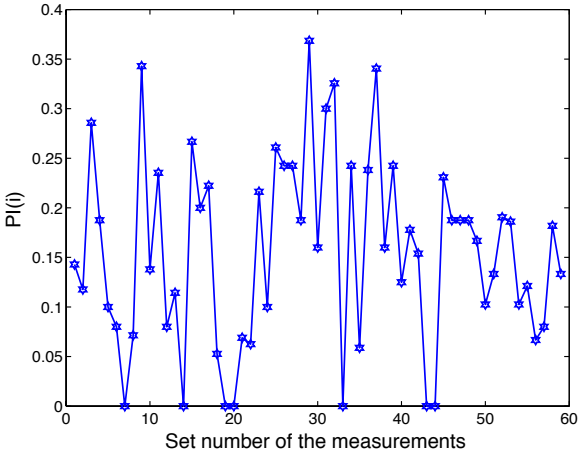
[34] S. Orfanidis. *Optimum Signal Processing. An Introduction. 2nd Edition*. Prentice-Hall, Englewood Cliffs, NJ, 1996.



(a) The mean backward-path reordering rates.



(b) The average autocorrelation of backward-path reordering.



(c) The pair-reorder index for backward-path reordering.

Figure 7: Measurement results for the paths from `www.applecomputer.com` and `kidsafe.apple.com` to the POINTER client.