Measuring Packet Reordering

John Bellardo and Stefan Savage
Department of Computer Science and Engineering
University of California at San Diego

Abstract— **The Internet architecture provides an unsequenced datagram delivery service. Nevertheless, many higher-layer protocols, such as TCP, assume that packets are usually delivered in sequence, and consequently suffer significant degradation when packets are reordered in flight. While there have been several recent proposals to create protocols that adapt to reordering, evaluating their effectiveness requires understanding the dynamics of the reordering processes prevalent in the Internet. Unfortunately, Internet packet sequencing is a poorly characterized and understudied behavior. This failing can be largely attributed to the lack of accurate and universally applicable methods for measuring packet reordering. In this paper, we describe a new set of active measurement techniques that can reliably estimate one-way end-to-end reordering rates to and from arbitrary TCP-based servers. We validate these tools in a controlled setting and show how they can be used to measure the time-domain distribution of the reordering process along a given path.**

I. Introduction

This paper considers a deceptively simple, yet practically challenging problem: how to accurately measure packet reordering in the Internet environment and how to report such measurements in a useful way.

The motivation for this problem arises from the interaction between the abstract service guarantees provided by the Internet architecture and the concrete service assumptions made by higher-layer protocols. The Internet architecture guarantees only a noisy, unsequenced and unreliable datagram service – any packet may be corrupted, dropped, or reordered in transit. In principal, higher-level protocols must be designed to tolerate and recover from all such errors transparently. In practice, real protocols and applications are optimized around common case assumptions about how real Internet infrastructures behave under normal conditions. Consequently, most protocols assume that corruption, packet loss and reordering are infrequent events or occur primarily under deterministic conditions. When these assumptions are not met, existing protocols can suffer in performance, correctness, or both. For example, it has been shown that the Transmission Control Protocol (TCP) fares poorly under some wireless channels due to the increased levels of random packet loss [18]. Similarly, recent empirical analyses of packet corruption have suggested that many errors may be undetected in long-lived TCP streams due to the design assumptions of the Internet checksum algorithm [15].

While less appreciated, packet sequencing errors can present similar problems to current protocols. For example, TCP's fast retransmit optimization assumes that packet reordering is sufficiently rare than any reordering event spanning more than a few packets implies a loss [14]. Since TCP also assumes that losses are primarily caused by buffer overflows, these reordering events can be misinterpreted as congestion signals and cause the sender to dramatically reduce its throughput [10], [3]. Similarly, interactive streaming media protocols, such as Voice-Over-IP, assume that sequencing errors are sufficiently rare that media playout buffers are only necessary to absorb inter-packet jitter [7]. Finally, some protocols incorporate compression or non-idempotent state transitions that can produce inconsistent results in the presence of sequencing errors [4], [17], [13].

However, compared to network characteristics such as loss or delay, the dynamics of packet reordering are far less well understood. A significant part of the problem is the lack of standard experimental techniques for measuring the phenomena. Previous reordering studies have used measurement tools that are inherently biased, such as *ping*, or methodologies that scale poorly, such as analyzing multi-site packet traces. As well, the lack of a standard measurement methodology has hampered the creation of a standard reordering metric. Consequently, the results between different studies can superficially vary by an order of magnitude – creating significant confusion and controversy about the prevalence of reordering.

In this paper, we describe a collection of techniques that provide one-way reordering measurements in both directions between a probe host and most TCP-based servers on the Internet. We also propose a primitive metric – the number of exchanges between pairs of test packets – that can be used to summarize reordering activity for a known load. While simplistic, this metric captures the essence of any reordering process and can be further parameterized to capture more sophisticated phenomena. As an example, we show how parameterizing packet exchanges by intervening delay can be used to describe the reordering process as a function of time.

The remainder of this paper is structured as follows: In Section II we review the previous work in this area and describe its strengths and limitations. Section III describes how our measurement algorithms work, followed by section IV which describes our controlled validation results and our experiences using the resulting tools in the Internet environment. This section concludes with our early experience extracting time-domain distributions of reordering behavior, leading to our conclusion in Section V.

II. Related Work

The two best known measurement studies that discuss reordering behavior are Paxson's 1997 general measurement study [10] and Bennett et al's study of exchange point reordering [2]. Each is worth discussing briefly.

Paxson's study is based on a series of measurements taken between 35 computers on the Internet as follows: 100KB files were transferred between all pairs of machines, packet traces were collected passively, and the TCP sequence numbers in each trace were analyzed to determine if any TCP segments were delivered out-of-order. One important strength of this approach is that it allows unidirectional measurements to be made – and in fact Paxson reports that reordering activity was asymmetric on the paths measured (although, as we will explain, it is difficult to distinguish if this is a property of the path or of the test). Overall, Paxson reports reordering in two ways: the fraction of sessions with at least one reordering event (12 and 36 percent across two measurement periods) and the fraction of packets that were reordered (2 and 0.3 percent in the direction of data transfer and 0.6 and 0.1 percent in the opposite direction).

However, there are several shortcomings to this approach. First, it is difficult to scale this methodology operationally. Performing such tests requires permission to execute code on both endpoints of any measured path – a requirement that seems difficult to scale to significant fractions of the Internet. For example, it seems unlikely that such an approach could be used to estimate reordering on paths to commercial information services such as cnn.com or yahoo.com. A second problem is that this technique does not produce an unbiased estimate of the reordering behavior on a path. Since reordering is known to be a time-varying process, the delay between sample packets could have a strong influence on how much reordering will occur. TCP's natural dynamics will produce highly variable samples as packets are sometimes compressed and sometimes spread apart. Sources of this variation include the delayed acknowledgment mechanism (also mentioned by Paxson), differences in serialization delay due to different packet sizes, and the usual flow and congestion control mechanisms that can suddenly increase or decrease the number of packets sent during an interval. The end result is that this style of passive measurement is effective at estimating the reordering seen by a one-way 100KB TCP data transfer in situ, but it is unclear if it can be generalized to infer what might happen to a different application with different dynamics.

Bennett et al. describe a very different approach to measuring reordering based on active probes. The authors generate estimates by sending repeated ICMP echo request packets to a remote host and then evaluating the order of the ICMP echo reply packets that are generated in response. The benefit of this approach is that it allows paths to arbitrary hosts to be measured, so long as the end host will respond to ICMP requests. As with Paxson, Bennett et al., also report reordering in two different ways. For bursts of five 56-byte packets they report that over 90 percent saw at least one reordering event. Then, isolating a host with significant reordering, they report that 100 packet bursts of 512 bytes each produce similar results using a synthetic metric based on how many Selective Acknowledgment (SACK) option records would be needed to cover the out-of-order replies [6].

The most obvious limitations of this approach arise from the use of the ICMP echo request/reply protocol for measurement. Using this method it is not possible to distinguish if a packet was reordered before it arrived at the remote host or after it left the remote host. Consequently, the measurements produced can both underestimate the total reordering and overestimate the re-ordering in either direction. Since most protocols are more sensitive to reordering in one direction than another this ambiguity can be quite important [2], [3]. Also, the use of ICMP as a fine-grained measurement tool is problematic since system and network operators alike increasingly filter and rate-limit such traffic to address security concerns. As well, neither metric used in this study seems easy to generalize: The number of bursts that have one reordering event is highly sensitive to the size of the burst, while the number of SACK blocks covering a reordered sequence is highly TCP-dependent.

Finally, there have been efforts within the IETF to define active measurement protocols designed expressly for measuring packet reordering [8]. These approaches eliminate the biases caused by overloading existing TCP sessions or the ICMP protocol, but still require deployment at each endpoint measured.

Despite the limitations of existing studies, several researchers have used them to justify modifications to TCP designed to better tolerate packet reordering [16], [3], [20], [1]. Most of these approaches dynamically change the fast retransmit threshold in response to estimated changes in the reordering rate. All of these projects would benefit from access to contemporary empirical data, since they cannot validate the impact of these changes without understanding the prevalence of reordering phenomena in today's networks.

III. Reordering measurement techniques

In developing tools that actively measure reordering we had two principle goals: accuracy and ubiquity. An ideal tool should produce correct and consistent results, providing estimates of the reordering activity on the *forward path* from the user's host to an arbitrary endpoint and on the *reverse path* from the endpoint back to the user's host. Achieving these goals practically is a tricky matter, however, since it is generally infeasible to deploy new services on the remote endpoints. Consequently, our techniques leverage the behavior of *existing* TCP and IP protocols and their implementations – turning any host exporting a TCP/IP service into a de facto measurement server. While this approach is very powerful, it can also be sensitive to minor protocol violations and variations in implementation. In particular, we encountered considerable challenges posed by network aliasing devices, such as load balancers, that required us to develop multiple techniques.

In the remainder of this section, we first review the TCP and IP features that we leveraged to produced our algorithms, and then describe a series of measurement techniques that use standard TCP behavior and common IP implementation characteristics to arrive at estimates of one-way reordering behavior under varying practical limitations.

A. TCP/IP review

IP is an unreliable, unsequenced datagram protocol. It provides three important features: network-layer addressing, per-protocol demultiplexing, and fragmentation. For the purposes of this paper only the last quality is worth reviewing. Since different network links may support different maximum frame sizes, it is possible that a router may need to fragment an IP datagram into several smaller datagrams. To allow a remote host to reassemble these fragments, the IP packet header includes an identification field (IPID), that is specified by the sender. When a datagram is fragmented, the same identification field is copied into each fragment and the receiver uses this value as a key to associate fragments from the same datagram. For this procedure to work the value of IPID must be unique for all outstanding packets between a sender and receiver pair. In practice, most implementations guarantee this property by minimizing the probability that *any* IPID will be repeated in a given time period. The traditional implementation of IPID uses a single global counter that is incremented for every packet that a host sends. This common implementation artifact implies that when looking at two packets from the same host, with IPID x and IPID y, if $y > x$, the packet with value x was sent before the packet with value y (modulo wraparound, which is easily detected). However, some newer implementations use alternative schemes for security reasons, such as small random increments and per host-pair counters, so any assumptions about this field must be validated before they can be trusted.

TCP is a connection oriented protocol that provides full-duplex reliable in-order data communication over unreliable network protocols, particularly IP. To provide the ordering guarantee, TCP includes a sequencing mechanism that associates a sequence number with each byte of data. Each TCP header contains the sequence number of the first byte of data being transmitted, and an acknowledgment (ACK) number indicating the next byte of data that the host is expecting to receive. The receiving host uses the sequence numbers written by the sender to reassemble packets in the correct order, while the sending host uses the ACK numbers written by the receiver to advance its sliding flow control and congestion control windows.

A receiver can generate an ACK for each data packet received, but in practice most TCP implementations use a *delayed acknowledgment* algorithm in the hopes of piggybacking an ACK on an outgoing data packet. Implementation guidelines indicate that ACKs should not be delayed by more than 500ms or two received data packets, but these guidelines are not uniformly followed, especially during slow start [9].

Finally, when a TCP receiver receives an out-of-order data packet it responds by acknowledging the last in-sequence packet it received. This signal allows the sender to quickly infer that a packet has been lost if multiple duplicate acknowledgments are received. To make this optimization work, the delayed acknowledgment algorithm is suspended for out-of-order data and acknowledgments are sent immediately.

B. Single Connection Test

The *Single Connection Test*, as its name implies, uses a single connection to the remote host, established using the

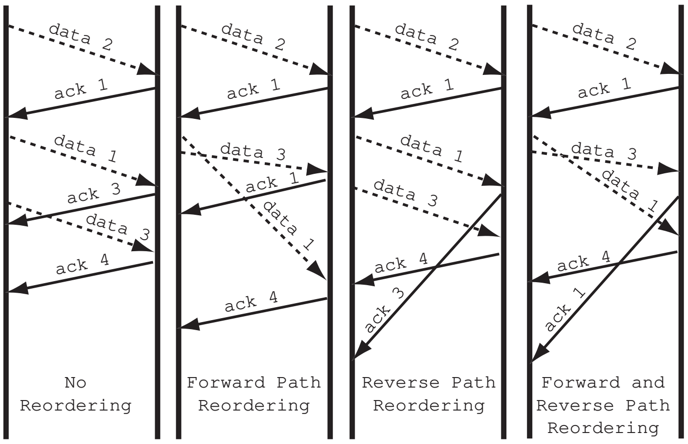

Fig. 1. Packet sequences generated by the single connection test. All forward path packets are denoted with a dashed line and all reverse path packets a solid line. Sequence numbers and acknowledgment numbers are labeled explicitly above each packet.

standard TCP three-way handshake. Once this connection is established, measurement samples are obtained one at a time, each composed of two phases: a preparation phase and a measurement phase. During the preparation phase, a sequence "hole" is created at the receiver by sending a slightly out-of-order packet repeatedly until the sender receives an acknowledgment indicating that an earlier packet is expected. For example, in Figure 1, it is assumed, without loss of generality, that the initial sequence number expected is 1. The local host then transmits a packet with sequence number 2 (abbreviated here as **data 2**) to the remote host. The receiver responds with an acknowledgment indicating that byte 1 is still expected (abbreviated **ack 1**) and permits the sender to infer that byte 2 has been queued at the receiver. Once the connection has been prepared a reordering measurement is made by sending two sample packets to the remote endpoint. The sequence numbers of these packets are selected to straddle the previous out-of-order data packet queued at the remote host. In the illustrated example, these sample packets are 1 byte TCP data packets labeled **data 1** and **data 3** respectively.

This particular relationship between the sequence numbers used is chosen because it forces the receiver to respond differently depending on the order in which the sample packets are received. This quality allows the sender to eventually infer whether the packets, or their acknowledgments, were reordered in flight. If both packets arrive in order, then the receiving host will acknowledge them in order – **ack 2** and **ack 4** in the figure. If the packets are delivered out-of-order, then the receiving host will first generate an acknowledgment for the "hole" followed by an acknowledgment for the whole series – **ack 1** followed by **ack 4**. Since these are the only two combinations possible, the sender can determine if the acknowledgments were reordered themselves by checking if the acknowledgment for the whole sequence, **ack 4**, arrives first.

A limitation of this approach is that it requires two samples to be generated and delivered from the remote endpoint. There are two practical reasons why this may not occur. First, if one of the sample packets or one of the acknowledgments is lost then the resulting single acknowledgment may be insufficient to determine whether the sample packets on the forward path were reordered and can never determine if there was reordering on the reverse path. For reasonable loss rates this problem can be addressed simply by discarding such samples (although this may bias the measurement if loss is correlated with reordering).

However, the second and more serious problem is posed by the delayed acknowledgment algorithm. In the common case in which packets are delivered in-order, the receiver may choose to only send a single acknowledgment for the whole sequence, **ack 4**, which does not allow the sender to infer anything about ordering in either direction. To mitigate, but not solve, this problem we can reverse the order in which sample packets are sent (**data 3** is sent followed by **data 1**) since the receiver will not delay acknowledgments for out-of-order packets (as mentioned before, this is critical to support the fast retransmit error detection function). To infer whether these packets are reordered, we simply invert the forward path algorithm (e.g. the arrival of **ack 1** followed by **ack 4** indicates that the packets arrived in order, while **ack 3** followed by **ack 4** indicates reordering). However, the sending host still cannot differentiate between a lost packet on the reverse path and a forward path reordering event (e.g. a lone **ack 4** is ambiguous).

C. Dual Connection Test

To address the previous limitations, we developed an alternative approach leveraging the traditional IPID generation function across pairs of TCP connections between local and remote hosts. In this *Dual Connection Test*, a measurement sample is a pair of packets, one sent on each connection, that are labeled with sequence numbers one greater than that expected by their respective receivers. Since these packets are interpreted as being out-of-order they are acknowledged immediately by the receiver, thereby avoiding the delayed acknowledgment issue. Under the assumption that IPID increases monotonically across TCP connections to the same destination, we can infer whether the acknowledgments were reordered by validating that packets are received in the same order of their respective IPIDs. Once we have determined the or-

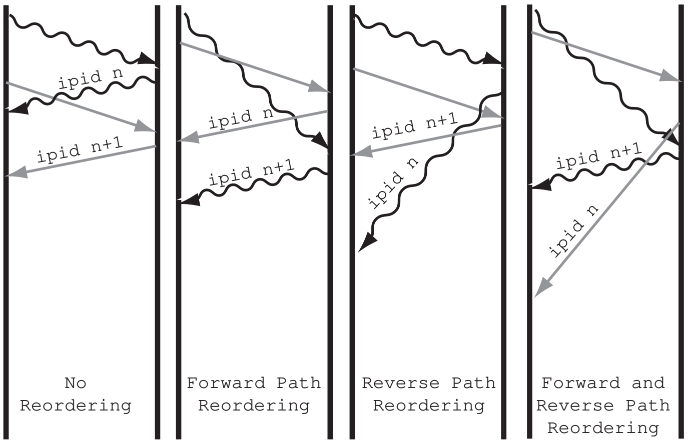

Fig. 2. Packet sequences generated by the dual connection test. It is assumed that *two* TCP connections are already established. Packets belonging to each connection are distinguished by wavy and straight lines respectively.

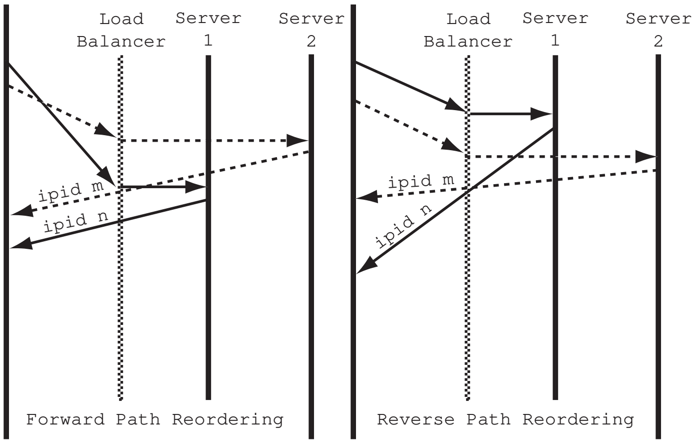

Fig. 3. An example of what happens to the *Dual Connection* test when there is a transparent load balancer in the middle. There is no way to differentiate between the two situations in this example.

der the packets were sent by the remote host it is possible to determine the order in which the sample packets were received. This is based on the observation that packets are acknowledged in the order they are received by the remote host (this is true in almost all systems since transport-layer processing is handed in the kernel, frequently driven directly by an interrupt). Consequently, the sender may infer whether its sample packets were reordered by comparing whether the difference between the IPIDs of the acknowledgments is consistent with the order in which the sample data packets were sent. It is for this reason that two connections are used, since each data packet can be easily associated with its companion acknowledgment using the source and destination port numbers as a key.

Unfortunately, while the dual connection test does not fall prey to delayed acknowledgments, it does have problems of its own. The first problem is the reliance on a strictly increasing function used to generate the IPID values. For example, recent versions of Linux (2.4 and on) use MTU Path Discovery by default and, since fragmentation cannot happen, transmit packets with IPID equal to 0. Moreover, modern versions of OpenBSD generate pseudorandom IPIDs and FreeBSD has a similar option that can be enabled . [1] Consequently, we must validate any assumptions about IPID before using it to disambiguate packet order. Another, more serious, related issue is that our reliance on two connections creates problems when connections are aliased through a transparent load balancing device. While in our experience these devices do not manipulate the IPID field, by design they may assign each connection to a separate host – thereby invalidating our assumption that both connections use the same IPID address space and producing spurious results. Figure 3 provides a graphical depiction of this problem. Unfortunately, while most hosts on the Internet do not sit behind load balancers, many sites of interest do – particularly those providing consumer content.

To address both issues, we compare the difference of the IPID values between each pair of adjacent packets *between* connections and each pair *within* a connection. If the underlying IPID space is shared between connections and is strictly increasing, then the latter differences will continually dominate the former (since a new data packet is only sent after both connections receive their ACK). In contrast, if IPID is set randomly or if a load balancer is present then there will be poor correlation in this measure. This analysis allows us to validate whether a particular host is amenable to the dual connection test before collecting spurious measurements.

D. SYN Test

For the specific problems presented by load balancers, we present a third approach, the *SYN Test*, based on related pairs of TCP SYN packets. We observe that load balancers cannot operate on a per-packet basis, but instead must balance requests per-flow or at larger granularities. Consequently, for any given TCP connection a load balancer will always forward packets to the same host. The most common implementation strategy to ensure per-flow granularity is to hash on the four-tuple: (sources address, source port, destination address, destination port), but others will use SYN packets to establish explicit per-flow state.

We can exploit the per-flow granularity of load bal-

[1] As well, modern versions of Solaris maintain a per-destination IPID counter, but since our techniques do not depend on IPID being unique across destinations this is not a complication.

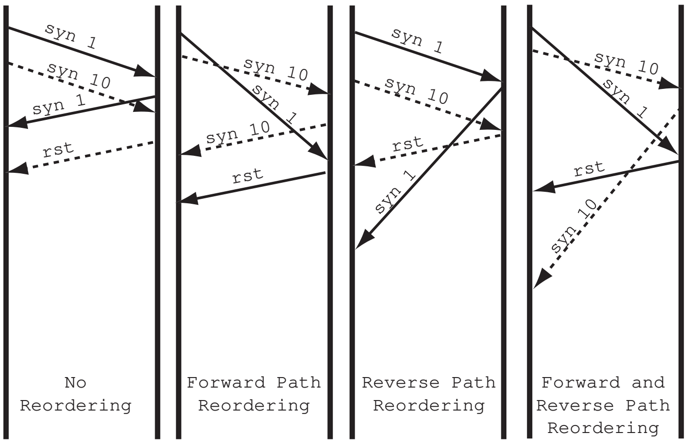

Fig. 4. All possible packet sequences for the *SYN Reorder* test.

ancers by transforming our measurement packets to *appear* to belong to the same flow. In particular, we send pairs of sample packets that are identical TCP SYNs that differ only in their starting sequence number being slightly offset. Since all fields that are used to key a flow identity are the same, these packets will be routed to the same host.

To determine reordering, this test exploits details of TCP's three way handshake. The first SYN packet will cause the remote host to enter the SYN_RECV state and generate a SYN/ACK in response. However, when the second SYN arrives, a number of things may happen. Strictly following the TCP specification, if the second SYN's sequence number is inside the allowable window (which it will be if the SYNs arrive in order) then a RST should be generated, otherwise the remote host should respond with a pure ACK (indicating the SYNs arrived out of order). However, in practice, this portion of the TCP specification is poorly understood and the most common implementations *always* respond to a second SYN with a RST, allowing reordering to be inferred simply as shown in Figure 4. Finally, we can always determine if the original SYNs were reordered in flight by examining the sequence number in the first transmitted SYN/ACK. Since the SYN/ACK will acknowledge the sequence number of the first packet received, it is easy to determine if it was sent in response to the first or second packet transmitted by the sender.

While this test is singularly useful for tolerating the effects of load balancers, it has two obvious drawbacks. First, it depends on parts of TCP semantics that are not consistently implemented. For example, a small number of implementations generate dual RST packets or only respond to the first SYN. Second, this test has a significant non-technical challenge since a large number of trials may be misconstrued as a small "SYN Flood" Denial-of-Service attack. In practice, we have not experienced this problem, but we have also been very careful to limit the rate at which SYNs are generated and make sure to fully establish a connection and then close it down.

E. TCP Data Transfer test

Finally, an obvious point of comparison is to simply initiate a TCP data transfer (for example, an HTTP GET request to a Web server) and evaluate whether packets are delivered in order. We can mitigate the problems of TCP congestion control dynamics by generating acknowledgments for the largest sequence number received, even if intermediate data is lost, and by artificially restricting the flow of information from the remote server by restricting the advertised receiver window and the advertised Maximum Segment Size (MSS). However, this test is only useful for measuring reverse path reordering, from the remote host to the local host. It cannot be used to determine if packets sent to the Web server are likely to be reordered. As well, it requires that the remote host be running a Web server, or similar public data transfer server, and requires that there be an object of sufficient size to fill two packets (this is a problem in practice for sites that use HTTP redirects, which fit in a single packet).

IV. Validation and Experience

All of the tests previously described were implemented as an extension to the *sting* tool described in [12]. Programmable packet filters and firewall filters were used to allow a user-level test program to generate and receive arbitrary IP packets without conflicting with the kernel's network stack.

A. Controlled Validation

We tested the resulting tool in a controlled environment in which all traffic was routed through a FreeBSD-based router under our administration. We modified the popular *dummynet* traffic shaping package on the router to swap adjacent packets according to a specified probability distribution [11]. During the testing period a machine in close proximity to the measurement machine was chosen as the remote host to keep the amount of real reordering at a minimum. We used two separate uniform random distributions for the forward and reverse path reordering rates, and the mean of each distribution was varied to include all combinations of 1%, 3%, 5%, 10%, 15%, and 40% (in the TCP data transfer test only the reverse path distribution was manipulated). We collected 100 samples for each measurement technique for each combination of the distribution means. A network trace was captured for every test run and this trace was analyzed to find the actual number of sample packets that were reordered during the trace. This

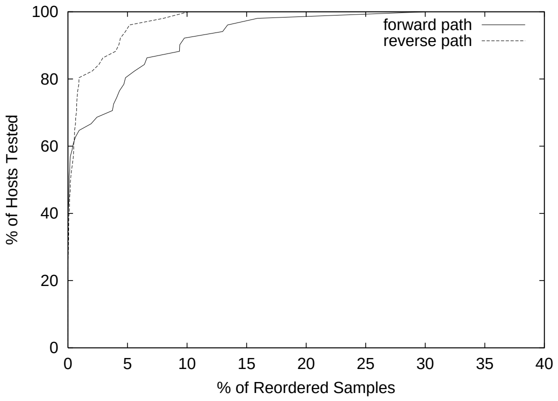

Fig. 5. CDF of reorderering rates across all paths measured.

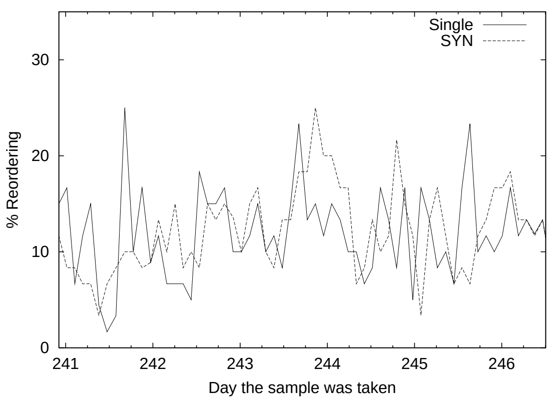

Fig. 6. Forward path reordering as measured by the *Single Connection* test and the *SYN Test*. The *Dual Connection* test could not be used because www.apple.com uses a load balancer.

number was compared to the number reported by the various reordering tests. Out of the 114 tests there were 8 discrepancies in the forward direction and 2 in the reverse direction. Of these, 7 of these were off by one reorder event (out of 100 samples) and the remainder were off by 2 reorder events. We believe both discrepancies reflect minor implementation corner cases that were incorrectly handled (although we have not completely confirmed that hypothesis). Overall, of the 114,000 samples, 99.99% of the samples were confirmed as correct.

B. Experience

We also ran the same tests against a set of live hosts on the Internet from a single probe machine located at UCSD. We selected 15 hosts to include a representative from all major operating systems and to include several highly popular hosts (such as yahoo.com and hotmail.com). We also included 35 hosts chosen randomly from approximately 18,000 hosts generated from the Yahoo random URL database [19]. Over the duration of 20 days, we conducted continuous measurements of these hosts, cycling through all four tests on each host and then cycling round-robin to the next host. Over the three week period, this produced approximately 850 measurements per host per test, where each individual measurement consisted of 15 samples . [2] Not all tests were able to work with all hosts. In particular, the dual connection test was ruled out due to non-monotonic IPID behavior from 8 hosts (likely due to transparent load balancers) and a constant IPID value of 0 from another 9 hosts (likely running Linux 2.4).

These measurements are not intended to be representative of the Internet as a whole, but simply to demonstrate the variety of results that can be obtained using our methods. Overall, as illustrated in the CDF in Figure 5, we observed that over 40% of the paths tested experience some reordering during the 20 day period. It is also clear that for the sites we chose there is more forward path reordering that reverse path reordering. It is possible that this is an artifact of using a single vantage point, but it is impossible to determine without using additional probe hosts. Furthermore, we found that the amount of reordering is directly related to the time between consecutive packets sent from the probing host; we study this behavior in more detail in Section IV-C. Finally, although not shown here, our experiments showed more than 15% of measurements had at least one reordered sample.

It is difficult to perfectly validate results in the Internet setting since we do not know the true order in which packets were sent or received by the remote host. Instead, the best alternative is to compare the results of the different tests to one another, under the assumption that since they all measure the same underlying process their results should be similar. For example, Figure 6 illustrates the mean reordering rate measured on the path to www.apple.com using the single connection test and the SYN test. As demonstrated by this example, the comparison cannot be perfect since the samples of each test are taken at different times. Consequently, these measurements can only be considered "paired" under the assumption that the reordering process is stationary over the time-period between measurements.

Given these limitations, we compute a standard pair-difference test statistic [5] for each host, comparing the results of each pair of tests. The null hypothesis is that the difference between tests can be explained purely in terms of intra-test variability. With a 99.9% confidence interval we find that the single connection test and the SYN test provide similar results (78% of the forward path tests and

[2] The TCP Data Transfer test is an exception. It consists of variable number of samples depending on the number of packets required to transfer the root Web object from the remote host

93% of the reverse path tests support the null hypothesis). The dual connection test however shows lower similarity to the other forward path tests, and to the SYN reverse path test, but has high similarity to the other two reverse path tests. We do not currently have an explanation for this discrepancy, other than the limitations of this analysis (visual inspection of the raw measurement data does not reveal obvious differences). Finally, the results from the TCP data transfer test closely matched the SYN and dual tests (90%) but was significantly different from the single connection test (73% of hosts rejected the null hypothesis). Examining these cases by hand, we find that during periods of significant reordering, the TCP data transfer tests can produce significantly lower estimates of reordering than the other approaches – sometimes less than half as many reordering events.

C. Characterizing Reordering Dynamics

The discrepancy between our tests and the baseline TCP data transfer experiment led us to reexamine the metric we use to describe reordering. In all of our approaches we chose to report reordering as the probability that a pair of back-to-back packets is reordered over a given time interval. This is a fairly primitive notion of reordering that can easily be extrapolated to calculate the reordering probability of a sequence of packets under the assumption that reordering is an IID process. However, for several reasons, this assumption is almost certainly invalid.

In thinking about this problem, we reflected on the physical source of reordering in existing Internet routers. From previous work [2] and discussions with router designers we identified two likely culprits. The first is inverse multiplexing and load balancing within a switch, which can allow a newer packet to be placed in a long queue while an older packet is placed in a short queue. Generally, most vendors using these techniques spend considerable effort to avoid this situation, particularly within a flow. Another source of packet reordering is packet striping across multiple L2 links. Many vendors continue to implement such striping on a per-packet basis and consequently, if a newer packet is placed on a link with a longer queue than an older packet, then reordering may occur. Since queues drain at a constant rate, the likelihood that this occurs is related to the inter-arrival time between the two packets. Packets that are more spread apart can tolerate a greater queue imbalance than those that are more compressed in time.

In the case of the TCP data transfer test, full-sized data packets of 1500 bytes were used, whereas the other tests consist of minimum sized packets of roughly 40 bytes. Consequently, the added serialization delay in the former case increases the delay between the leading edge of each

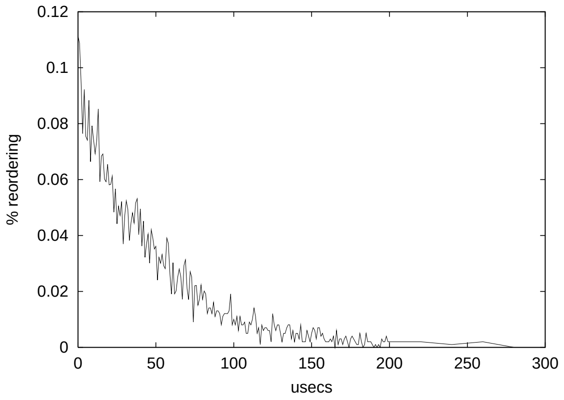

Fig. 7. Reordering probability along a single path as a function of inter-packet spacing between two minimum sized TCP packets using the dual connection test. The spacing, indicated on the y-axis, is measured in microseconds. 1000 samples were taken at each point using 1 usec increments between points for all spacings below 200 usecs, and 20 usec increments thereafter.

packet – possibly reducing the probability that the two packets will be reordered if they are assigned to different queues.

To explore this hypothesis, we modified our tests to accept an inter-packet gap as a parameter. By varying the spacing between two sample packets we can evaluate the dynamics of the reordering process as a function of time (although this is currently limited to forward path reordering where we can easily control packet spacing in a fine-grained manner). Figure 7 illustrates the results of this approach applied to one path that demonstrated significant reordering. Minimum-sized back-to-back packets are reordered more than 10 percent of the time, which quickly drops off to less than 2 percent after 50 microseconds of delay is added and approaches zero after 250 microseconds. Distribution measurements such as these are strictly more powerful than a traditional summary statistic, such as the average reordering rate. Using the distribution it is possible to predict how different protocols and applications would be impacted by the reordering process, without needing to construct a unique test (e.g., SACK blocks) for each protocol. For example, we can infer that, during bulk data transfer, full-sized data packets are less likely to be reordered than streams of compressed acknowledgment packets.

V. Conclusion

The Internet architecture makes few guarantees about the underlying service provided. Nevertheless, higher-layer protocols are engineered to take advantage of the common behavior, and as a consequence they are penalized when these assumptions are not met. At the same time the Internet is highly heterogeneous and is constantly changing. For any set of assumptions, there is a time or place on the network where they are invariably incorrect and will cause problems. To debug such problems and understand how network characteristics are evolving, it is critical to have the tools and metrics for measuring and describing these conditions in a complete manner.

In this paper we focused on the problem of packet reordering. Packets may be reordered for many reasons, including DiffServ scheduling and buffer management, multi-path routing, layer 2 retransmission (particularly across wireless links), or simply due to highly variable load when routers make use of fine grained data parallelism. Whatever the reason, this violation of the in-sequence assumption can significantly degrade the performance of protocols like TCP. However, there has not previously been an accurate technique for easily measuring reordering, nor sufficient understanding of how reordering works to define a meaningful metric.

We have developed three techniques for accurately measuring one-way packet reordering rates in the existing Internet infrastructure. Each of these has its strengths and flaws, but as a whole they provide a significant advancement on previous methods. We have validated each approach in a controlled environment and shown that the tests are roughly consistent with one another in a live Internet setting. Moreover, we demonstrate that our tests can be modified to measure the distribution of the reordering process over time and argue that this representation is more meaningful than a single scalar metric. Finally, looking forward, we believe that the toolset we have developed provides a reasonable framework for producing more conclusive results about reordering in different parts of the Internet and evaluating whether it occurs frequently enough to justify changes to existing protocol assumptions.

VI. Acknowledgments

We would like to thank our shepherd, Vern Paxson, and the anonymous reviewers for their comments and suggestions. This research was supported in part by DARPA Grant N66001-01-1-8933.

References

[1] M. Allman, H. Balakrishnan, and S. Floyd. RFC 3042: Enhancing TCP's Loss Recovery Using Limited Transmit, January 2001.

[2] J. Bennett, C. Partridge, and N. Shectman. Packet Reordering is not Pathological Network Behavior. *IEEE/ACM Transactions on Networking*, 7(6):789–798, 1999.

[3] E. Blanton and M. Allman. On Making TCP More Robust to Packet Reordering. *ACM Computer Communication Review*, 32(1), 2002.

[4] V. Jacobson. RFC 1144: Compressing TCP/IP Headers for Low-Speed Serial Links, February 1990.

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

[6] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. RFC 2018: TCP Selective Acknowledgement Options. RFC2018, October 1996.

[7] D. Minoli and E. Minoli. *Delivering Voice over IP Networks*. John Wiley & Sons, 1988.

[8] A. Morton, L. Ciavattone, and G. Ramachandran. Internet Draft: Reordering Metrics for IPPM using Non-Reversing Sequence. draft-morton-ippm-nonrev-reordering-00.txt, February 2002.

[9] V. Paxson. Automated Packet Trace Analysis of TCP Implementations. In *Proceedings of the 1997 SIGCOMM Conference*, pages 167–179, Cannes, France, September 1997.

[10] V. Paxson. End-to-end Internet Packet Dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[11] Luigi Rizzo. Dummynet: a Simple Approach to the Evaluation of Network Protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[12] S. Savage. Sting: A TCP-based Network Measurement Tool. In *USENIX Symposium on Internet Technologies and Systems*, pages 71–79, Boulder, CO, October 1999.

[13] W. Simpson. RFC 1661: The Point-to-Point Protocol (PPP), July 1994.

[14] W. Stevens. RFC 2001: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, January 1997.

[15] J. Stone and C. Partridge. When the CRC and TCP Checksum Disagree. In *Proceedings of the 2000 SIGCOMM Conference*, pages 309–319, Stockholm, Sweden, August 2000.

[16] Y. Tamura, Y. Tobe, and H. Tokuda. EFR: A Retransmit Scheme for TCP in Wireless LANs. In *IEEE Conference on Local Area Networks*, volume 23, pages 2–11, 1998.

[17] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, and B. Palter. RFC 2661: Layer Two Tunneling Protocol, August 1994.

[18] G. Xylomenos and C. Polyzos. TCP Performance Issues over Wireless Links. *IEEE Communications Magazine*, 39(4):52–58, April 2001.

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

[20] M. Zhang, B. Karp, S. Floyd, and L. Peterson. Improving TCP's Performance under Reordering with DSACK. Technical Report TR-02-006, International Computer Science Institute, July 2002.