Toward an Understanding of the Processing Delay of Peer-to-Peer Relay Nodes

Kuan-Ta Chen and Jing-Kai Lou
Institute of Information Science, Academia Sinica
ktchen@iis.sinica.edu.tw, kaeaura@iis.sinica.edu.tw

Abstract—**Peer-to-peer relaying is commonly used in real-time applications to cope with NAT and firewall restrictions and provide better quality network paths. As relaying is not natively supported by the Internet, it is usually implemented at the application layer. Also, in a modern operating system, the processor is shared, so the receive-process-forward process for each relay packet may take a considerable amount of time if the host is busy handling some other tasks. Thus, if we happen to select a loaded relay node, the relaying may introduce significant delays to the packet transmission time and even degrade the application performance.**

In this work, based on an extensive set of Internet traces, we pursue an understanding of the processing delays incurred at relay nodes and their impact on the application performance. Our contribution is three-fold: 1) we propose a methodology for measuring the processing delays at any relay node on the Internet; 2) we characterize the workload patterns of a variety of Internet relay nodes; and 3) we show that, serious VoIP quality degradation may occur due to relay processing, thus we have to monitor the processing delays of a relay node continuously to prevent the application performance from being degraded.

Index Terms—**E-Model, Internet Measurement, Peer-to-Peer Systems, QoS, VoIP**

I. INTRODUCTION

Voice communication over IP is becoming one of the most profitable Internet businesses. It has been shown that VoIP users are willing to pay for value-added services, such as intercommunication with a PSTN phone (dialing to a PSTN phone and vice versa), voice mails, and call forwarding. Although one of the major players, Skype [25], was not the first company to provide VoIP service, but it did pioneer the delivery of such services to an unprecedented wide range of end-users. From a technical point of view, we believe that three factors are responsible for Skype's popularity: the user-friendly interface, the high quality audio codecs, and the sophisticated peer-to-peer network infrastructure.

Skype is well-known, or perhaps notorious, for its capability to "steal" computation and communication resources from a computer with a Skype instance running. This is because Skype employs a technique called *peer-to-peer relaying*, where the network communications between two call parties can be made through a third intermediate node, commonly called a relay node. Peer-to-peer relaying brings the following advantages to VoIP applications: 1) the voice quality can often be

improved by detouring traffic via a better quality network path, which can be achieved by using a relay node [21]; 2) a relay node can help establish connections if both call parties are behind NATs or firewalls [1, 6, 22]; and 3) relaying enables data aggregation, which reduces network bandwidth usage when more than two parties are involved in a conference call. For these reasons, peer-to-peer relaying is widely employed by VoIP services, such as Skype and Google Talk [7], as well as by video streaming services, such as AnySee [13] and PPLive [10].

Even though peer-to-peer relaying is beneficial to real-time applications in many aspects, we argue that its *dark side* might be easily overlooked. As relaying is not natively supported by the Internet, it is currently implemented by deploying an overlay network at *the application layer*. In this way, a packet "forwarded" by a relay node is actually a brand new IP packet that the node "clones" from the packet to be relayed. Also, since relay applications are usually run at a normal priority, their execution could be deferred due to high-priority tasks, such as system threads, device I/O request handlers, or foreground jobs. Thus, the time needed for a relay node to receive, process, and regenerate a relay packet could be unpredictably long because it depends on the workload of the node. For these reasons, *the extra delays incurred at a relay node could be considerable and even detrimental to the application's performance*.

In this paper, based on an extensive Internet measurement, we consider whether peer-to-peer relaying leads to substantial or even detrimental extra delays. Our analysis is divided into three parts. First, we describe how we *collect* the processing delays of relayed packets from a large set of relay nodes on the Internet. Second, we analyze and characterize the processing delays of the relay nodes, and show that the delays are closely related to the host's activity. Third, we investigate whether the relay process *degrades* the quality of VoIP calls and whether such degradation is a general or occasional phenomenon. To the best of our knowledge, this paper is the first large-scale study to measure the processing delays introduced by peer-to-peer relaying and their impact on an application's performance. Our contribution is three-fold:

1) We propose a methodology that can measure the processing delays at *any* relay node on the Internet, without modifying the existing network and system infrastructure.
2) We collect the relay processing delays from a large set of Internet nodes distributed all over the world. In addition,

we analyze the sampled processing delays to gain an understanding of the workload pattern of Internet nodes, the effect of packet size on relay processing, and the instability of processing delays.

3) We show that the processing delays incurred at relay nodes may significantly degrade VoIP quality based on ITU-T E-model [12] and the delays at relay nodes are generally unstable.

We consider this paper as a first step in focusing on the extra processing delays introduced by peer-to-peer relaying. From our results, *we suggest that all real-time peer-to-peer systems keep track of the workload on relay nodes to prevent unexpected performance degradation.*

The remainder of this paper is organized as follows. Section II describes related works. We then propose our processing delay inference methodology in Section III. In Section IV, we describe the trace collection methodology and discuss the quality of our collected traces. In Section V, we analyze a number of characteristics of the collected relay processing delays, such as typical workload patterns and stability. In Section VI, we evaluate the impact of the processing delays of relay nodes on VoIP quality. Finally, in Section VII, we present our conclusions.

II. RELATED WORK

To the best of our knowledge, this paper is the first to study the processing delays of peer-to-peer relay nodes and their impact on the application performance. Thus, there are no earlier studies directly related to our work. In [14], Liu and Zimmermann mentioned that the average processing delay at each overlay node of AudioPeer [29], a commercial voice chat system, was approximately 30 ms. However, they did not report how the measurement was conducted and how large and representative the data set was. One closely related research is probably that on peer-to-peer relay node selection. A number of studies [3–5, 11, 15, 21, 27] have been devoted to selecting relay nodes from a set of candidates to obtain a good quality network path. However, the selection criteria are mainly based on the network latency and loss rate, and do not consider the processing delays introduced by relay nodes. Our work complements these network-quality-based relay node selection studies by emphasizing that relay processing delays should also be considered when selecting the best relay node.

III. PROCESSING DELAY INFERENCE

In this section, we propose a methodology for measuring the processing delays induced by relaying packets at an intermediate node. The processing delay is defined as the time an intermediate node takes to send a data packet to its destination on behalf of a source host. Delays can be due to a variety of operations, such as receiving a packet from the source host, passing it to the relay application (mostly in user mode) for processing, and forwarding the packet to the target host.

The processing delay at a relay node is generally *unmeasurable* unless we place a sniffer to monitor the incoming and outgoing traffic of the node. Note that even the relay application *cannot* measure the processing delays of relayed packets directly because it has no information about when a packet arrives at or departs from the system. To measure processing delays exactly, an application must exploit a kernel-mode packet filter, such as BPF [17], or raise its thread priority to a high value so that it will always be served immediately. However, either approach is inadequate for a large-scale deployment, as they incur additional resource overhead and may disturb the execution of the user's own tasks.

Our methodology is designed to measure the processing delays experienced by relayed packets at a relay node *without any modification to the existing network infrastructure and peer-to-peer software*. In the following, we first define the terms used throughout this paper. We then explain the basic rationale behind our inference methodology, and elucidate the IPID filtering scheme for improving the estimation accuracy. Finally we evaluate both the basic and improved methods through an experiment.

A. Term Definition

In this paper, we assume a two-hop relaying scenario in which every packet from a source host transits an intermediate node before reaching the destination host.

- **Source/destination**: A pair of hosts that communicate with each other through an intermediate host.
- **Relay node (relay host)**: The intermediate host receives packets from the source and forwards them to the destination. The relay node might change the content of the relayed packet slightly depending on the application's context.
- **Source packet**: A data packet sent from the source host to the relay node.
- **Relay packet**: The data packet sent from the relay node to the destination, which is a copy (verbatim or with slight changes) of the source packet.
- **Ack packet**: When a source packet is delivered to the relay node by TCP, the relay node will acknowledge the packet by sending a TCP acknowledgement packet back to the source host. We call this an ack packet.
- **Processing delay (PD)**: The time lapse from the instant a source packet arrives at a relay node and the instant its corresponding relay packet leaves the node.
- **Data delivery time (DDT)**: The time lapse from the instant a source packet leaves the source host to the instant its corresponding relay packet reaches the destination.
- **Ack response time (ART)**: The time lapse from the instant a source packet leaves the source host to the instant its corresponding ack packet is received by the source host.

B. The Basic Method

Our method is based on the following assumptions, which are generally observable across current peer-to-peer implementations:

1) The relay node forwards a relay packet to the destination as soon as it receives a source packet, i.e., no *intentional* delays are introduced in relay packet forwarding.
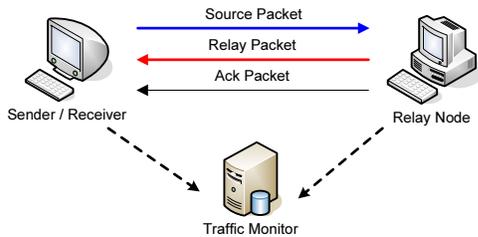
Fig. 1. Experiment setup for measuring ack packet generation delays and evaluating the accuracy of processing delay estimation.

Fig. 2. The scatter plot of data delivery time and ack response time.

2) The source transmits packets to the relay node by TCP; therefore, the relay node will respond with TCP ack packets upon the receipt of every packet (or every two packets if the delayed ack option is enabled [18]).

3) The TCP implementation (which generates ack packets) is running at a *high* priority, while the relay application is running at a relatively *low* priority. We require that ack packets are elicited with an approximate constant delay so that they can be taken as a reference for "relaying without processing delay."

Our basic concept is that, on the arrival of a TCP source packet, the relay node will respond with two packets: 1) an TCP ack packet sent back to the source, and 2) a relay packet forwarded to the destination. These packets are generated by different parts of the relay node system. An ack packet is generated by the TCP implementation, which is part of modern operating systems and normally runs at a high scheduling priority. On the other hand, a relay packet is generated by the relay application, such as Skype and PPLive, which is developed by a third-party vendor and runs at a normal scheduling priority. As a result, an ack packet can always be elicited promptly, while it usually takes some time to generate a relay packet because the relay application can only get its quantum when high-priority threads have completed their tasks[1]. In this way, a relay packet's additional processing delay can be computed as *the time difference between the time instant the relay packet and the instant its corresponding ack packet left the relay node*.

However, this technique requires us to monitor the incoming and outgoing traffic of the relay node, which is not practical for large-scale measurements. To overcome this restriction, we tactically place the source and the destination hosts at the same location, which ensures that the ack packet and relay packet traverse the same network path to the target. By so doing, we can estimate the processing delay as *the difference between the instant an ack packet arrives at the sender and the instant the corresponding relay packet reaches the destination host*. This strategy renders large-scale measurements feasible because we only need to monitor the traffic of the source and destination hosts.

1) Constancy of ACK Generation Delay: Our method only works if the TCP implementation generates ack packets with a constant delay. To verify this assumption, we conduct experiments on the network topology depicted in Fig. 1. In the

experiment, all the computers are commodity PCs equipped with Pentium III, 1 GB RAM, and Windows XP, but the traffic monitor is a SuSE Linux running `tcpdump`. During the experiments, the source keeps sending 200-byte data packets at a 30 pkt/sec frequency to the relay node, which then forwards the packets to their destination. To emulate a heavy workload on the relay node, 10 different movie clips are played simultaneously, which generates a considerable CPU workload (for video decoding and playout) and I/O workload (for reading audio/video data from the disk).

Because the source/destination nodes and the relay node are at the same Ethernet LAN, the network delay between them is negligible. Fig. 2 plots the relationship between the data delivery time (DDT) and the ack response time (ART) of each source packet (note that the x-axis is with a log-scale). The leftmost dense area indicates that a linear relationship between DDT and ART exists when no other threads are competing with the relay application for computation cycles. However, when the relay node is busy handling other tasks, the DDT increases by orders of magnitude (spread between 0 and 500 ms), while the ART is always less than 0.3 ms. The reason for such a small ART is that the TCP/IP stack gives a high priority to serving incoming TCP packets and generates ack packets at the first opportunity [26]. In contrast, the execution of a relay application can be deferred for a long time because the processor serves high-priority jobs, such as kernel-related threads, device I/O requests, and the foreground tasks, first.

C. Sample Filtering based on IPID

Although we can calculate the processing delay of a relayed packet by simply subtracting the DDT from the ART in a LAN-scenario like Fig. 1, the result will be less accurate if the relay node is on the Internet because of *network delays*. The problem is that, in the Internet scenario, both DDT and ART are subject to network dynamics and network delays *independently*. Thus, the result of subtracting DDT from ART will be affected by the network delays of the relay packet and ack packet, and deviate from the true processing delays at the relay node.

We cope with network variabilities by filtering out packets that lead to inaccurate processing delay estimates based on the IPID field. We call this the *IPID filtering* method. Since the IPID field is strictly increasing for IP packets generated by a computer, we use the information to determine the release order of packets from the relay node. The rationale of our IPID filtering method is as follows: "If a set of packets sent by a node are *reordered in the network*, then at least one of them

---

[1]This statement is somewhat simplified. Even if high-priority tasks have not been completed, the scheduler will regularly give lower-priority tasks a chance to be executed in order to avoid starvation.
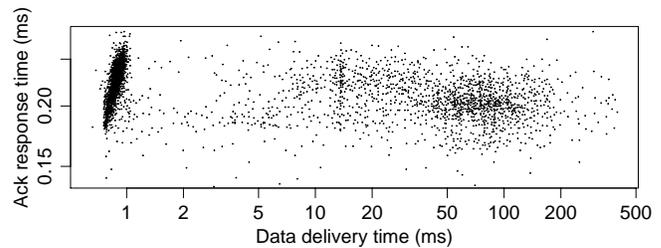
must have experienced unusual network delays and should be filtered out."

As we do not have direct access to every relay node on the Internet, the following rules are employed to determine the packet ordering at the node. 1) For packets from the source host, we detect whether they arrive at the relay node sequentially by analyzing the IPIDs of their corresponding ack packets. A smaller IPID in an ack packet indicates that its corresponding source packet arrived at the relay node earlier, and vice versa. 2) For packets from the relay node, we detect the sequence they departed the relay node by their IPIDs.

Fig. 3 illustrates three possible packet-reordering scenarios. Note that this is not an exhaustive listing of possible reordering cases. For example, a relay packet for a source packet x may exchange its order with an ack packet for a source packet y, which forms another category of reordering. Any occurrence of packet sequence rearrangement indicates that at least one packet is experiencing unusual network delays. Suppose a source packet i that departed from the source host at time $t_{s,i}$ elicits an ack packet with IPID $id_{ack,i}$ and a relay packet with IPID $id_{r,i}$, which arrive at their destinations at time $t_{ack,i}$ and $t_{r,i}$ respectively. We detect packets with unusual network delays as follows:

1) For all source packets, we obtain a sequence $\{t_s, id_{ack}\}$ sorted by t_s. We then apply the *longest increasing subsequence* (LIS) algorithm [24] to the $\{id_{ack}\}$ sequence, ID_{ack}, and obtain its longest increasing subsequence L. The set $\{L\}$ denotes the IPIDs of the packets that keep order with each other in the network. Thus, we remove the packets with IPIDs in the set $\{ID_{ack} - L\}$, because they are the packets with unusual delays that lead to inaccurate processing delay estimates.

2) We combine $\{id_{ack}, t_{ack}\}$ and $\{id_r, t_r\}$ as a sequence, and sort them by the first element. Then, we apply the LIS algorithm to the sequence formed by the second element, $ID_{ack,r}$, and obtain the longest increasing subsequence L. Once again, the packets with IPIDs in the set $\{IPID_{ack,r} - L\}$ are those that experience unusual network delays and should be removed.

After removing packets with inconsistent network delays (i.e., compared to the remaining packets), the estimated processing delays will be all greater than zero, as the DDT must be longer than the ART for a source packet. Also, the error of a processing delay estimate will be bounded by the interarrival times of the packets surrounding pairs comprised of an ack packet and a relay packet.

D. Accuracy Evaluation

We verify the effectiveness of the proposed processing delay inference methods with a series of experiments. The network setup of the experiments is identical to that described in Section III-B1 and shown in Fig. 1. However, this experiment described here is much more realistic because it takes account of *network delay variability* and *packet reordering*. We use a trace-driven approach to emulate the network dynamics. The network delay of each packet is sequentially assigned a measured ART from the set of Internet traces described
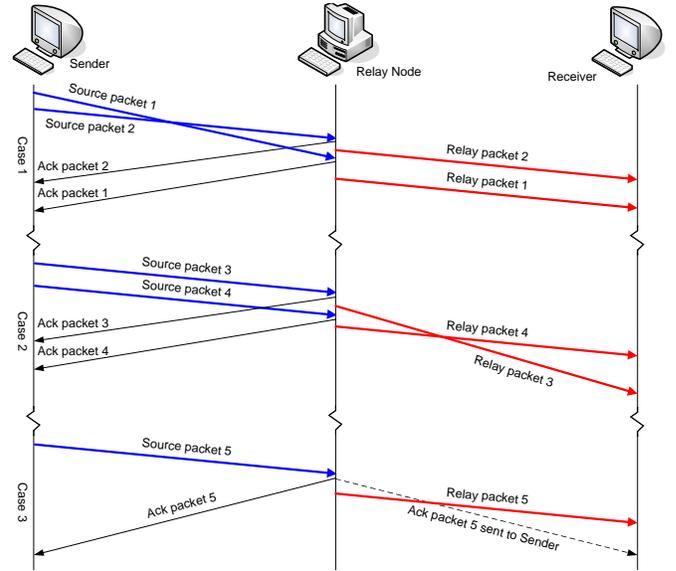

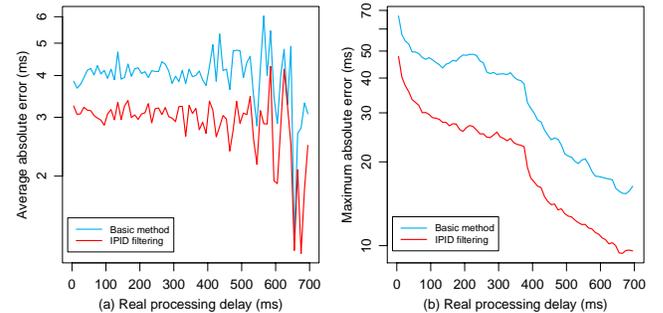Fig. 3.   Some common network reordering scenarios

Fig. 4.   The average and maximum absolute errors of processing delay estimates vs. the real processing delays.

in Section IV. This introduces a great deal of randomness into the network delay of each packet, thus simply subtracting ART from DDT might even derive a negative processing delay estimate.

We run the experiment for 500 flows, each of which lasts for 10 minutes. We first evaluate the performance of both methods by the average and maximum of the absolute estimation errors, which are computed by |Real PD − Estimated PD|. Fig. 4 shows the absolute errors of the estimated processing delays versus the real delays. We observe that the average absolute errors are only 4 ms and 3 ms for the basic method and the IPID filtering method respectively, so the difference between the two methods is not significant. However, the benefit of the IPID filtering is significant in terms of the maximum absolute errors, e.g., the error reduces by 20 ms with IPID filtering, while the real processing delays are around 200 ms.

IV. LARGE-SCALE MEASUREMENT

In this section, we describe how we measure the processing delays of relayed packets for a large set of Internet relay hosts, after which we begin with a description of our trace collection procedures and a summary of the traces. We then discuss the anomalous behavior patterns identified and their causes. The section concludes with an accuracy assessment

of the estimated processing delays derived from the Internet measurements.

A. Trace Collection Methodology

Our trace collection platform is based on Skype for several reasons.

- Skype is enormously popular and it has a very large installation base. There can be 50 million users online and 200 thousand super nodes in use at any one time [8]. As these super nodes are capable of *relaying* voice calls for regular nodes, we take them our measurement subjects. The large number of super nodes form a virtually unlimited candidate list of relay nodes for our measurement.
- Skype is *robust* in terms of establishing network connections because, if a caller host cannot reach the callee host or vice versa, Skype will find a super node to bridge the voice transmission for the two parties.
- When a relay node currently in use becomes unavailable to the caller or callee, Skype can always find an alternate relay node to replace the original one. Because of these capabilities, we can force Skype to pick an alternate super node for relaying a VoIP call whenever necessary, without worrying about running out of candidate relay nodes.

The trace collection mechanism comprises three commodity PCs deployed as shown in Fig. 5. One serves as the VoIP caller, one serves as the callee, and the third is a monitor that collects information about the traffic of both call parties. The collection procedures are as follows:

1) When the measurement program is initialized, we block the caller from directly reaching the callee with a firewall program `ipfw`.
2) The caller initiates a VoIP call to the callee (via the callee's Skype name). Because of the firewall setting, the caller will be connected to the callee host via one of its super nodes.
3) If a VoIP call is established, we know that Skype has found a super node to relay voice packets between the caller and the callee.
   Occasionally, after a few retries, a VoIP call cannot be established because the caller fails to find a usable relay node from the candidate list. In this case, our daemon will restart the Skype program and re-attempt to establish a VoIP call. The Skype program will retrieve a new list of super nodes from the central server at the startup, so further VoIP calls should be successful.
   Also, we sometimes find that voice packets from the caller to callee and vice versa are relayed through different super nodes. In this case, we simply drop the call, block both relay nodes, and re-dial.
4) To simulate a conversation, a WAV file comprised of all the English recordings in the Open Speech Repository[2] is played continuously for both parties during a call.
5) After a call has lasted 10 minutes, we block the current relay node at the caller by `ipfw` and terminate the call. We then wait for 30 seconds before re-iterating the loop from Step 2.
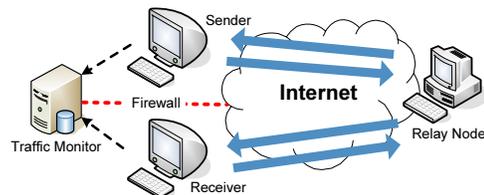

Fig. 5. Network setup for measuring processing delays at any relay node on the Internet.
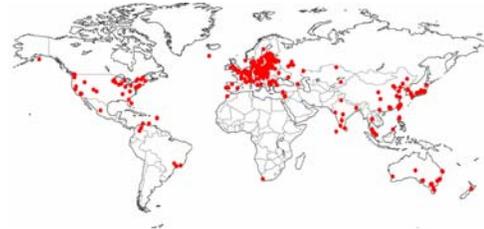

Fig. 6. Geographical locations of observed relay nodes in our trace.

We ran the trace collection procedures over a two-month period during April 9th and April 20th in 2007 and collected 1,115 calls, which are summarized in Table I. The observed relay nodes are spread over the world, as shown in Fig. 6. On average, each call lasted 10 minutes and had an average packet rate of 32 pkt/sec. The overall average ack response time was 203 ms, while the average data delivery time was 212 ms. The difference between the ART and the DDT shows that, on average, forwarding a relay packet takes 9 ms longer than generating a TCP ack packet at a relay node. This observation strongly supports our conjecture that the processing delays at relay nodes are not negligible and are therefore worthy of our attention.

B. PD Estimation Results

We apply the IPID filtering method to the collected traces and compute the processing delays at each relay node. A summary of the processing delay estimates is given in Table II. On average, each call contains 15,739 processing delay estimates, which corresponds to 26 samples per second. To gain a general idea of the processing delay estimates, the average processing delay is 5 ms, and the maximum processing delay is 239 ms, which leads to a large maximum-to-mean ratio of 48.

C. PD Estimation Accuracy Assessment

Even though we evaluated the accuracy of our processing delay inference scheme in Section III-D, we now provide further accuracy checks of the estimated processing delays of Internet relay nodes. We focus on large processing delay estimates, as they have more impact on the application performance.

- Out of 21,418,790 samples that we collected, 88% led to positive PD estimates. This statistic manifests that a DDT longer than an ART is not just a coincidence, but a consequence of the delay incurred at the relay node.
- The standard deviation of PD estimates is 7 ms on average, which is not large compared to the high PDs, say 20 ms or even 50 ms, we focus on.

[2]http://www.voiptroubleshooter.com/open_speech/

TABLE I
SUMMARY OF COLLECTED TRACES

# Calls	Time	# Pkts/call	ART	DDT
1,115	10 min	19,210 pkts	203 ms	212 ms

TABLE II
SUMMARY OF ESTIMATED PROCESSING DELAYS

# Samp.	Samp. dens.	PD (Avg / Max / SD)
15,739 per call	26 samp. / sec	5 ms / 239 ms / 7 ms

- If the PD estimates are accurate, they should be highly correlated with the IPID differences between each pair comprised of an ack packet and a relay packet, assuming that the IPIDs increase at a steady rate on a host. We find that the correlation between PDs and IPID differences has a mean of 0.62 and a standard deviation of 0.16 on average. In addition, the large PD samples should be associated with large IPID differences for accurate estimation. We find that among the top 5% of PD estimates, on average, 48% are associated with the top 5% of IPID differences.

- If the PD estimates are accurate, they should be uncorrelated with the ack response times. We find that the correlation between PDs and ack response times is merely -0.01 on average, which indicates the large PD estimates are not simply due to shorter ack response times.

- The IPID filtering method may fail to detect unusual network delays if there is a low density of packets surrounding a PD sample. To verify this possibility, we define the maximum allowed delay time of a PD estimate, where packet reordering cannot be detected by IPID filtering, as a "PD error range." We find that the top 5% of PD estimates associate with the top 65% of PD error ranges, and the top 20% of PD estimates associate with the top 56% of PD error ranges. Also, the correlation coefficient between the PD estimates and PD error ranges is only 0.08 on average. These observations strongly suggest that PD estimates, especially large ones, are not a consequence of low-density surrounding packets.

We use the above tests to confirm the estimation accuracy of the processing delays in our large-scale measurement. Having verified the validity of the estimated processing delays, we further analyze the characteristics of processing delays at relay nodes and their impact on VoIP quality in subsequent sections.

V. PROCESSING DELAY CHARACTERIZATION

In this section, we analyze the relay node processing delays obtained from the Internet measurement (Section IV). We begin with a classification of the observed processing delays, and the possible causes of each type of delay, and then analyze the effect of packet size on processing delays. Finally, we characterize the stability of the processing delays and investigate the relationship between delay stability and host activity.

A. Classification of Processing Delays

Since processing delays at a relay node are closely related to the workload on the node, we can consider a pattern of
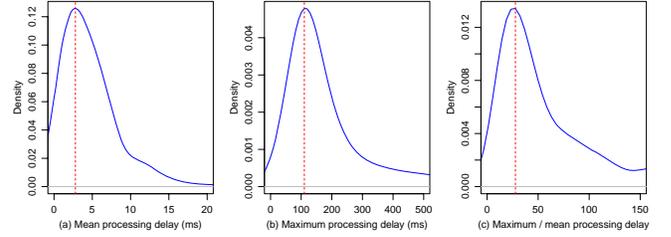

Fig. 7. The distributions of (a) average processing delays (b) maximum processing delays (c) the ratio of the maximum processing delay versus the average processing delay in a call.

processing delays as an indication of the workload structure. We find that the processing delay patterns can be classified into five categories, as shown in Fig. 14. The categories and the possible causes of each type of delay are as follows:

- **Typical**: This type of processing delay is the most common. The variation of PDs is small and their magnitude is rarely higher than 20 ms, which implies that the relay node is lightly-loaded and the computer is not in use.

- **Variable**: PDs are stable most of time, but they occasionally exhibit very different behavior. For example, the PDs of Call 561 are relatively high during the 250–450 ms period, which is likely due to a person using the computer, or an application with a time-varying workload is running.

- **Level-Shifted**: The levels of PDs are increased or decreased by a significant magnitude, say, larger than 10 ms. This indicates that a heavily loaded task starts or stops running on the relay node during the call.

- **Periodic**: Bursts of high PDs occur at regular intervals, possibly because of the behavior of an application. For example, the one-minute interval in the processing delays of Call 214 is likely caused by an email notification program with a one-minute check interval.

- **Loaded**: The level of PDs remains large, say 100 ms or higher. This implies that the relay node is under a heavy workload generated by computation- or I/O-intensive applications.

Fig. 7(a) and Fig. 7(b) plot the distributions of the average and maximum PDs of each call. It can be seen that the average PDs are generally shorter than 20 ms. The maximum PDs, however, have a much broader range, which spreads over 0 to 500 ms with a mean around 100 ms. We use the ratio between the maximum and average PDs to quantify the degree of maximum PD variation in a call, and plot its distribution in Fig. 7(c). The median of the ratios is 30, while its 90% percentile is around 200. The high variability of the ratios indicates that the variation of PDs can be extremely large, even within a 10-minute period.

B. Stability Analysis

We now turn to the stability of processing delays, which is closely related to the workload structure of a relay node. We begin our analysis with a definition of the metric *busy level*, which is designed to capture *the level of the workload* on the relay node. The busy level (BL) is defined as the 95% quantile of processing delays within a window of 10 seconds.

TABLE III
SUMMARY OF BUSY LEVELS OF STABLE AND UNSTABLE RELAY NODES

	Stable RN	Unstable RN	Ratio
BL Min	5 ms	6 ms	1.26
BL Mean	6 ms	17 ms	2.75
BL Max	8 ms	120 ms	14.17
BL SD	1 ms	16 ms	24.72

We choose the 95% quantile, rather than a more intuitive 50% quantile, because *even on a heavily loaded machine, the processing delays incurred by relay packets are not always large due to the thread scheduling mechanism.* More specifically, when a relay load is lightly loaded, a source packet is *always* serviced by the relay application as soon as it arrives at the node. In contrast, a source packet *sometimes has to wait a long time* before it is serviced on a heavily loaded node. The processing delay of a packet depends on the exact time it arrives at the relay node. *If a source packet arrives at a node just before the relay application's execution time, it will receive almost no processing delay no matter how heavy the workload is.* Otherwise, the packet will be postponed by a load-dependent period before being processed. For these reasons, the busy levels are defined bias toward high processing delays in order to obtain the true workload.

After computing the busy levels for each second in a call, we detect change points where BLs differ significantly. Rather than employ a mathematical definition of change points [9, 28], we identify a BL change from an operational point of view as follows. The time t is deemed a change point if the BLs of two consecutive seconds, t and $t + 1$, have a difference larger than 10 ms. Based on the detected change points, we classify each relay node into two categories: *stable* and *unstable*. A relay node is considered stable if its PD series contains no change points, and unstable otherwise. In our traces, 75% of relay nodes are classified as stable and 25% as unstable.

The stability of relay nodes can also seen an indicator of the *activity* of the relay host. In other words, if the busy levels of a relay node change significantly during a call, it is likely that the computer is in use for that period. We verify the correctness of the inferred stability of relay nodes based on the intuition that *a host is more likely to be active during working hours.* To do so, we compute the local time of each call by mapping a relay node's IP address to its geographical address and time zone. The ratios of all hosts and unstable hosts for each hour of a day are plotted in Fig. 8. We observe that, while the numbers of observed relay nodes are roughly constant, the numbers of unstable nodes are significantly higher from 8AM to 4PM in each node's local time. These results strongly support the contention that the measured processing delays reflect the true workload on relay nodes and can indicate whether the computer is currently being used by a person or running a workload-varying application.

We summarize the busy levels of stable and unstable relay nodes in Table III. The table also lists the ratios of the BL statistics of unstable and stable relay nodes. The results show that the busy levels of unstable relay nodes are not only significantly higher than those of stable nodes (the ratio of BL maximums is 14), but are also much more variable (the
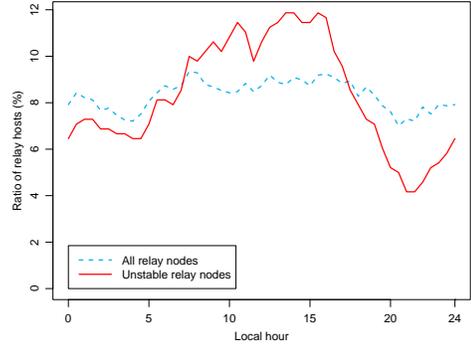

Fig. 8.   The average ratios of all relay nodes and unstable relay nodes for each hour of a day, where the ratios are relative to the number of all relay nodes and unstable relay nodes respectively.

ratio of BL standard deviations is 24).

VI. IMPACT OF PROCESSING DELAY ON VoIP QUALITY

Currently, peer-to-peer relaying technology is adopted by VoIP applications for the following reasons: 1) relaying can help bypass the connection restriction of NATs and firewalls because communication availability is an important consideration for widely used VoIP services; 2) VoIP requires a high degree of real-timeliness and interactivity, and relaying can help reduce network latency; 3) the bandwidth requirement of VoIP communication is not high, e.g., 32 Kbps, which is not a high bandwidth overhead to a relay node. In many cases, if we choose a relay node carefully, we are likely to find a relayed path that yields better network quality than the native Internet routing path. Thus, a number of studies have been devoted to finding a good relay node that yields a better quality path in terms of network latency or the packet loss rate [2–5, 11, 15, 21, 27]. However, without considering the workload on the relay nodes, the processing of relay packets may introduce significant delays to the relayed data stream and therefore degrade the application performance.

Having shown that the processing delays at a relay node are not always negligible, we now consider the impact of relay processing delays on VoIP quality. Via trace-driven simulations, we show that such delays may have a negative impact on VoIP quality. We also perform a characterization of busy periods on relay nodes and show that the busy status of relay nodes is generally quite unstable.

A. Methodology

We use trace-driven simulations to assess the impact of processing delays on VoIP quality. To measure the effect of processing delays under various network conditions, in each run, we combine a network delay trace and a processing delay trace to simulate a VoIP call with and without packet relaying. We use ack response times extracted from the collected calls as the input of network delays. As each of our 1,115 traces provides a series of network delays and a series of processing delays separately, our simulation comprises a total of 1,243,225 runs. The simulation time of each run is set to 250 sec and the packet rate is set to 33 pkt/sec.

During each simulation run, we compute the end-to-end delay and packet loss rate based on a given pair of network delay and processing delay traces. The end-to-end delay is decided by the playout buffer size, which also determines the end-to-end loss rate because a packet is considered lost if it cannot meet the playout schedule. There are two common schemes for adjusting the playout buffer size, namely *static* and *adaptive*, both of which are included in our simulations. For the static buffer strategy, we use a fixed 60-ms buffer after Skype's setting [23]. Our adaptive buffer is computed by the following equation:

$$p_i = d_i + 4 \times v_i, \tag{1}$$
$$d_i = \alpha \times d_i + (1 - \alpha) \times n_i, \tag{2}$$
$$v_i = \beta \times v_{i-1} + (1 - \alpha)\left|d_i - n_i\right|, \tag{3}$$

where p_i is the buffer size for packet $i + 1$, n_i is the network delay of packet i, and $\alpha = \beta = 1 - 0.998002$ according to [20].

We use the ITU-T E-model [12] to quantify the voice quality of each simulated call. Essentially, the E-model transforms a set of transmission impairments into a psychological satisfaction level. The main computation equation of the E-model is

$$R = R_o - I_s - I_d - I_e + A,$$

where R_o represents the fundamental signal-to-noise ratio, I_s represents the impairments occurring simultaneously with the voice signal, I_d represents the impairments related to delays, and I_e represents the impairments related to information loss. The advantage factor A is used for compensation when there are other advantages of access available to the user. Since the computation of R_o and I_e is codec-dependent, for simplicity, we assume the voice codec is the widely deployed G.711. The output of the E-model is the R-score, which represents the overall voice quality via a 100-pt scale. In our settings, the R-score reaches a maximum of 93.2 without any end-to-end delay or loss. Generally an R-score higher than 80 implies a satisfactory VoIP quality, while an R-score lower than 70 implies a quality that many users find unacceptable [19]. Following Table 1 in [19], we use a reduction of 10-points in the R-score to denote a significant degradation in VoIP conversation quality.

B. Performance Degradation

1) Transmission Delay and Loss: For each pair of network delay and relay processing delay traces, we assess the quality degradation by evaluating the network performance separately, i.e., end-to-end delay and loss, as well as voice quality, with and without relay processing. First, we consider the network performance degradation shown in Fig. 9. From the figure, we observe that the average delay increase caused by relay processing is not large because the playout buffer absorbs the variability introduced by processing delays. However, a fraction of calls still experience significant end-to-end delay increases, say, greater than 50 ms. The reason for the larger delay increase in the adaptive buffer scheme is that the buffer size is proportional to the mean deviation of network delays (Eqn. 3); thus, it becomes larger in the face of highly spiky processing delays. Moreover, the adaptive buffer does not seem
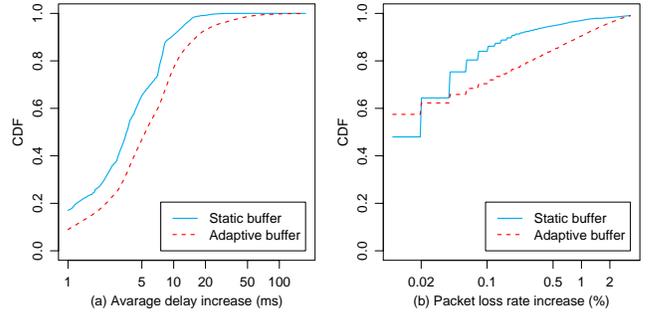

Fig. 9. The distributions of (a) average increase of end-to-end delays, (b) increase of packet loss rate in each call.
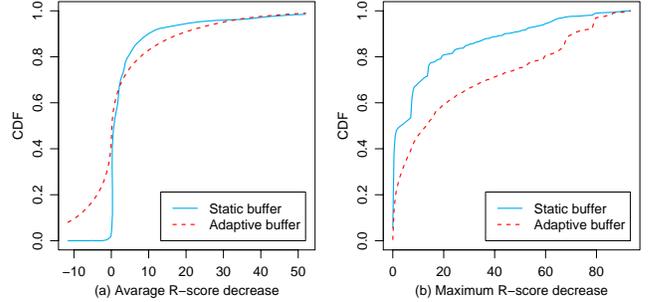

Fig. 10. The distributions of (a) average R-score increase, (b) maximum R-score increase in each call.

resilient enough to absorb highly variable processing delays, as it leads to both longer delays and higher packet loss rates than the static buffer scheme. In addition, the static buffer always absorbs delay deviations shorter than 60 ms such that only 1% of calls have an end-to-end loss rate higher than 1%, compared to 10% of calls under the adaptive buffer strategy.

2) VoIP Quality: With respect to VoIP quality degradation, we can see from Fig. 10(a) that the average R-score decrease caused by relay processing is generally small; most calls have an average R-score decrease smaller than 10. A few calls with adaptive buffering do have better voice quality, i.e., an increased R-score. Our analysis shows that this counter-intuitive behavior is due to a lower packet loss rate. The lower rate is a consequence of a larger playout buffer, which in turn is caused by a higher degree of delay variation due to relay processing. At the same time, the adaptive buffer scheme also leads to poor conversation quality, as about 20% of calls have an average R-score decrease of more than 10, which we consider a significant degradation in quality.

The performance degradation is much more obvious if we check the distribution of maximum R-score decreases in Fig. 10(b). The adaptive buffer scheme still performs much worse than the static buffer scheme. However, both schemes are subject to significant quality degradation due to relay processing; 40% and 58% of calls experienced considerable quality degradation for the static and adaptive buffer strategies respectively.

3) Original Quality vs Degradation: We note that the impact of processing delay is not equivalent for all calls. According to the E-model, a better quality call is more prone to performance degradation [12, 16]. We verify this effect by
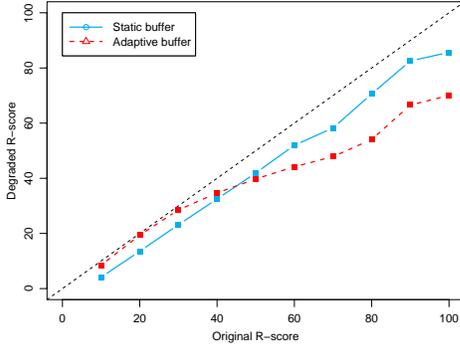
Fig. 11. The relationship between the original R-scores and the averaged degraded R-scores.
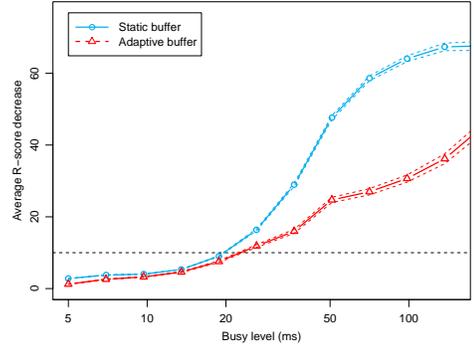

Fig. 12. The relationship between busy levels and the corresponding average R-score decreases. The dashed lines are the 95% confidence bands of the averages.

mapping the original R-score of each call to its degraded R-score, as shown in Fig. 11. Generally speaking, calls with a high R-score incur more quality degradation than those with a low R-score. We observe that the adaptive buffer scheme causes more degradation than the static buffer scheme for calls with an original R-score higher than 40. The reason is that a call with an initial high R-score is associated with short network delays; in this case, the adaptive buffer size tends to be decided by the spiky relay processing delays. On the other hand, the static buffer scheme is not susceptive to delay variations, so the quality degradation remains relatively constant.

4) Summary: Here, we formally define that a call is "degraded" if it experiences an R-score decrease greater than 10 points. Among the simulated calls, we find that the static buffer scheme causes 31% of the calls to be degraded, while the adaptive buffer scheme causes 54% to be degraded. In addition, for the two schemes, the average degradation time ratio within a call is 10% and 18% respectively.

C. Impact of Busy Levels

Having evaluated the overall VoIP quality degradation caused by relay processing delays, we now examine the relationship between workload levels and the degree of voice quality degradation. Recall that we defined the busy level to quantify the workload at a relay node in Section V-B. For each call, we first divide the busy levels and R-score decreases into 10 groups by their ranks. Then, we plot the average R-score decreases for a variety of busy levels, as shown in Fig. 12.

The figure clearly shows that higher busy levels lead to more serious quality degradation. On average, the R-score decreases are higher than 10 points when the busy level of the relay node is higher than 20 ms. This result implies that *we should avoid a relay node with a busy level higher than 20 ms, as transmitting VoIP packets through this node would lead to significant quality degradation.*

D. Busy Period Characterization

Based on the analysis results, we define that a relay node is *busy* when its busy level is higher than 20 ms. In our traces, 23% of relay nodes were even in a busy state during a 10-minute call. We also define a *busy period* of a relay node as a continuous time span during which the node is busy.
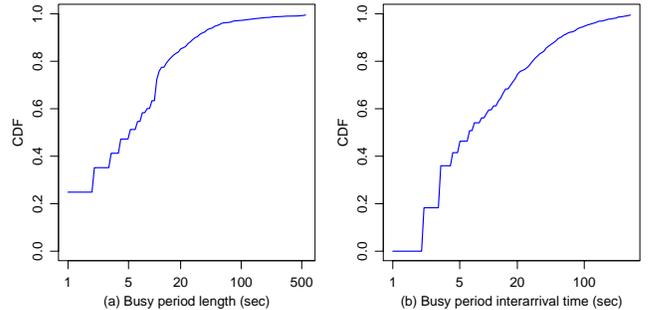

Fig. 13. The distributions of (a) busy period length, (b) busy period interarrival time in a call.

To understand the patterns of busy periods on relay nodes, we extract all the busy periods in each call and plot the distributions of their length and interarrival times in Fig. 13. We find that a busy period lasts for 18 seconds on average, where 65% of busy periods are shorter than 10 seconds. Also, the interarrival time of busy periods is 25 seconds on average. These statistics suggest that the busy status of relay nodes is quite unstable because the nodes tend to switch between busy and non-busy states frequently.

VII. Conclusion

In this paper, we consider a hidden aspect of peer-to-peer relaying—the processing delays at relay nodes. Existing related works mostly focus on how to improve current peer-to-peer systems by data relaying, but seldom discuss its adverse effects. Through the trace collection in Section IV, the statistical analysis in Section V, and the network simulations in Section VI, we show that relaying is a *double-edged sword* in that it could be also detrimental to VoIP quality if an inappropriate relay node is used. The degradation cannot be avoided completely as a lightly-loaded relay node may start running a load-intensive application at anytime. Thus, we have to monitor the processing delays of a relay node continuously, as we usually do for network conditions, to prevent the application performance from being degraded. We hope this study will motivate future peer-to-peer systems to focus on this negative aspect of the application-layer relaying technique.
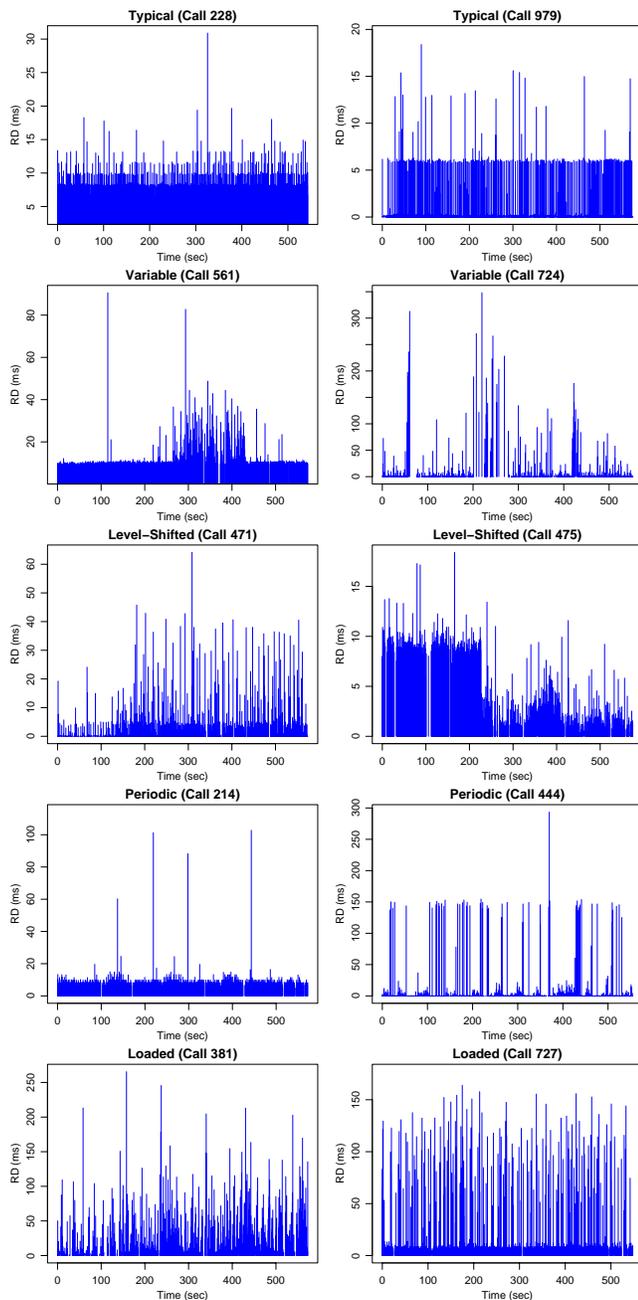
Fig. 14. A classification of common processing delay patterns. We list five categories: typical, variable, level-shifted, periodic, and loaded.

REFERENCES

[1] S. Baset and H. Schulzrinne, "An analysis of the Skype peer-to-peer internet telephony protocol," in *INFOCOM*. IEEE, 2006.

[2] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei, "Quantifying Skype user satisfaction," in *Proceedings of ACM SIGCOMM 2006*, Pisa Itlay, Sep 2006.

[3] C.-M. Cheng, Y.-S. Huan, H. T. Kung, and C.-H. Wu, "Path probing relay routing for achieving high end-to-end performance," in *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, vol. 3, 2004, pp. 1359–1365.

[4] T. Fei, S. Tao, L. Gao, and R. Guérin, "How to select a good alternate path in large peer-to-peer systems?" in *INFOCOM*. IEEE, 2006.

[5] T. Fei, S. Tao, L. Gao, R. Guérin, and Z.-L. Zhang, "Lightweight overlay path selection in a peer-to-peer environment," in *INFOCOM*. IEEE, 2006.

[6] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators," in *USENIX Annual Technical Conference*, 2005, pp. 179–192.

[7] Google, Inc., http://www.google.com/talk/.

[8] S. Guha and N. Daswani, "An experimental study of the Skype peer-to-peer VoIP system," Cornell University, Tech. Rep., Dec. 16 2005.

[9] F. Gustafsson, *Adaptive Filtering and Change Detection*. John Wiley & Sons, September 2000.

[10] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," in *IPTV Workshop, International World Wide Web Conference*, 2006.

[11] X. Hei and H. Song, "Stochastic relay routing in peer-to-peer networks," in *Proceedings 41st IEEE International Conference on Communications*, 2006.

[12] ITU-T Recommandation, "G. 107. The E-Model, a Computational Model for Use in Transmission Planning," International Telecommunication Union, CHGenf, 2002.

[13] X. Liao, H. Jin, Y. Liu, L. M.Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *INFOCOM*. IEEE, 2006.

[14] L. Liu and R. Zimmermann, "Adaptive low-latency peer-to-peer streaming and its application," *Multimedia Systems*, vol. 11, no. 6, pp. 497–512, 2006.

[15] Y. Liu, Y. Gu, H. Zhang, W. Gong, and D. Towsley, "Application level relay for high-bandwidth data transport," in *The First Workshop on Networks for Grid Applications (GridNets)*, San Jose, October 2004.

[16] A. Markopoulou, F. A. Tobagi, and M. J.Karam, "Assessment of voIP quality over internet backbones," in *Proceedings of INFOCOM*, 2002.

[17] S. McCanne and V. Jacobson, "The BSD packet filter: A new architecture for user-level packet capture," in *Proceedings of USENIX'93*, 1993, pp. 259–270.

[18] J. Nagle, "Congestion control in IP/TCP internetworks," *Computer Communication Review*, vol. 14, no. 4, pp. 11–17, Oct. 1984.

[19] M. Narbutt, A. Kelly, L. Murphy, and P. Perry, "Adaptive voIP playout scheduling: Assessing user satisfaction," *IEEE Internet Computing*, vol. 9, no. 4, pp. 28–34, 2005.

[20] R. Ramjee, J. F. Kurose, D. F.Towsley, and H. Schulzrinne, "Adaptive playout mechanisms for packetized audio applications in wide-area networks," in *INFOCOM*, 1994, pp. 680–688.

[21] S. Ren, L. Guo, and X. Zhang, "ASAP: an AS-aware peer-relay protocol for high quality voIP," in *Proceedings of ICDCS*, 2006, pp. 70–79.

[22] J. Rosenberg, R. Mahy, and C. Huitema, "Traversal Using Relay NAT (TURN)," draft-rosenberg-midcom-turn-05 (work in progress), July, 2004.

[23] B. Sat and B. W. Wah, "Playout scheduling and loss-concealments in voip for optimizing conversational voice communication quality," in *Proceedings of Multimedia'07*. New York, NY, USA: ACM, 2007, pp. 137–146.

[24] C. Schensted, "Longest increasing and decreasing subsequences," *Canad. J. Math.*, vol. 13, pp. 179–191, 1961.

[25] Skype Limited, http://www.skype.com.

[26] D. A. Solomon and M. Russinovich, *Inside Microsoft Windows 2000*. Microsoft Press Redmond, WA, USA, 2000.

[27] H. Zhang, L. Tang., and J. Li, "Impact of Overlay Routing on End-to-End Delay," in *Proceedings of ICCCN*, 2006, pp. 435–440.

[28] Y. Zhang and N. G. Duffield, "On the constancy of internet path properties," in *Proceedings of Internet Measurement Workshop*, V. Paxson, Ed. San Francisco, California, USA: ACM, Nov 2001, pp. 197–211.

[29] R. Zimmermann, B. Seo, L. Liu, R. Hampole, and B. Nash, "Audiopeer: A collaborative distributed audio chat system," *Distributed Multimedia Systems, San Jose, CA*, 2004.