1-800-OVERLAYS: Using Overlay Networks to Improve VoIP Quality

Yair Amir, Claudiu Danilov, Stuart Goose, David Hedqvist, Andreas Terzis

Technical Report CNDS-2004-2 - Aug 23, 2004
http://www.cnds.jhu.edu

Abstract— **Although telephony subscribers are accustomed to the consistent voice quality and high reliability of the traditional PSTN, the promise of a single converged IP network to carry voice and data – and the cost savings therein – motivates the interest to adopt voice-over-IP (VoIP) technologies. However, the Internet provides best effort delivery, without any inherent quality of service guarantees. Low latency is a key factor in supporting high quality interactive conversations, and as such contemporary VoIP solutions use UDP to transfer data over the IP layer, despite being subject to network loss and failures.**

This paper describes the use of an overlay network through which streams of voice packets are transmitted. Flexible application level overlay routers can understand the stringent requirements of VoIP and implement new algorithms that mask the limitations of the underlying Internet. We describe two protocols that facilitate localized recovery for lost packets and rapid rerouting in the event of network failures. Experimental results indicate that these two approaches can be combined to yield a quantitative improvement to voice communication quality.

Index Terms—**System design, Simulations, Experimentation with real networks/Testbeds**

I. Introduction

It is non-trivial to engineer a solution that meets the stringent constraints expected by humans of a high quality, reliable, real-time voice communication service. Delays of 100-150 msec and above are detectable by humans and can impair the interactivity of conversations. By comparison, humans are far less tolerant of audio degradation than of video degradation. Hence, to meet these requirements it is crucial to minimize primarily the network latency and secondarily packet loss as much as possible. To minimize latency, contemporary VoIP solutions rely upon UDP as the transport protocol. However this has the potential to expose VoIP packets to network loss and failures. Although the Internet can offer reasonable quality (relatively low loss and good stability) for the majority of VoIP streams, it has been shown [1] [2] [3] that it remains vulnerable to occasional bursts of high loss and link failures that preclude it from delivering a constant, high quality service demanded for telephony.

This paper describes an overlay architecture that can easily be deployed to address these intervals of network loss and failures. It maintains a high packet delivery ratio even under high loss, and adds minimal overhead under low, or no loss conditions. Our first observation is that it is often possible to recover packets even given the tight delay budget of VoIP. While many VoIP streams exhibit large latencies that prohibit timely end-to-end recovery, it is possible to perform recovery for many short links that are in the order of up to 30 msec. Our second observation is that by breaking long links into several smaller links, an overlay network architecture can help localize the packet recovery within overlay hops. Thus, even for VoIP streams with end-to-end latencies considerably larger than 30 msec, the vast majority of the packet losses can be rapidly recovered on the shorter overlay hop on which they were dropped. Overlay networks facilitate the deployment of flexible routing protocols that can address the needs of a specific application. Our third observation is that the overlay approach allows the deployment of a routing algorithm that optimizes the probability of packets being delivered within the delay requirements of VoIP, hence having a significant impact on the resulting voice quality.

The contribution of this paper is an overlay network system that is tailored to support VoIP by judiciously combining two complementary mechanisms: First, a real-time[1] packet recovery protocol that immediately delivers

Y. Amir, C. Danilov and A. Terzis are with The Johns Hopkins University, Department of Computer Science, Baltimore, Maryland. Email: {yairamir, claudiu, terzis}@cs.jhu.edu
S. Goose and D. Hedqvist are with Siemens Corporate Research, Inc. Princeton, New Jersey. Email: {stuart.goose, david.hedqvist}@scr.siemens.com

[1]Our definition of real-time refers to timely recovery of packets on short overlay links. Protocols such as RTP and RTCP, that do not recover packets, work independently of our protocols.

newly received packets, similarly to UDP, but this protocol attempts to recover missing packets. Recovery is attempted only once, and only if a packet is likely to arrive at the destination within the VoIP delay constraint. This protocol is deployed on every overlay link. Second, an adaptive overlay routing protocol tailored to VoIP, that optimizes path selection based on an approximation metric that combines the measured latency and loss of a link.

The system was implemented as part of an open source overlay network platform, Spines [4]. The behavior of our protocols was evaluated under controlled network conditions on the Emulab [5] testbed and directly in the Internet on the Planetlab [6] network. The performance of the proposed routing metric was evaluated through extensive simulations, comparing it to other metrics, on thousands of random topologies with various loss and delay link characteristics. We show that by leveraging our overlays for disseminating VoIP streams, the loss rate of the communication can be drastically reduced. For example, for a network loss rate of 5%, our system can usually recover within the latency constraints all but 0.5% of the packets. This leads to a commensurate increase in the voice quality of the calls. Our results reveal that using a standard voice codec, we could achieve PSTN voice quality despite loss rates of up to 7%. Our routing metric achieves good performance, selecting paths that optimize packet delivery ratio. It achieves better performance than individual latency, loss or hop-based routing schemes – especially in high latency networks where the voice delay constraint becomes more stringent.

The rest of the paper is organized as follows: In Section II we present the motivation and background of our work. In Section III we introduce our overlay architecture. We present and evaluate our protocols in Section IV. The routing limitations of overlay networks and how they can be addressed in real systems are described in Section V. Section VI discusses how we can integrate our approach in the current VoIP infrastructure. Section VII presents related work, and Section VIII concludes our paper.

II. Background

A. *Voice over IP*

As opposed to media streaming, VoIP communication is *interactive*, i.e. participants are both speakers and listeners at the same time. In this respect, delays higher than 100-150 msec can greatly impair the interactivity of conversations, and therefore delayed packets are usually dropped by the receiver codec.

Voice quality can be adversely affected by a number of factors including latency, jitter, node or link failures, and by the variability of these parameters. The combined impact, as perceived by the end-users, is that voice quality is reduced at random. Contemporary VoIP codecs use a buffer at the receiver side to compensate for shortly delayed packets, and use forward error correction (FEC) or packet loss concealment (PLC) mechanisms to ameliorate the effect of packet loss or excessive delay. The error correction mechanisms usually add redundancy overhead to the network traffic and have limited ability to recover from bursty or sudden loss increase in the network.

In the experiments of this paper we used a well-understood, widely deployed and good quality codec, the standard ITU-T G.711 [7], combined with its PLC [8] mechanism. The G.711 codec we used samples the audio signal at a rate of 8kHz and partitions the data stream into 20 msec frames, thus sending 160 byte packets at a rate of 50 packets/sec.

The VoIP quality is evaluated using an objective method described in ITU-T recommendation P.862 [9], known as Perceptual Evaluation of Speech Quality (PESQ). The PESQ score is estimated by processing both the input reference and the degraded output speech signal, similarly to the human auditory system. The PESQ score ranks speech signals on a scale from -0.5 (worst) to 4.5 (best), where 4.0 is the desired quality of regular PSTN.

B. *Internet loss characteristics*

Data packets are lost in the Internet due to congestion, routing anomalies and physical errors, although the percentage of physical errors is very small at the core of the network. Paxson in [2] studied the loss rate for a number of Internet paths and found that it ranged from 0.6% to 5.2%. Furthermore in that study and a follow-up [10], Paxson discovered that loss processes can be modeled as spikes where loss occurs according to a two-state process, where the states are either "packets not lost" or "packets lost". According to the same studies, most loss spikes are very short-lived (95% are 220 msec or shorter) but outage duration spans several orders of magnitude and in some cases the duration can be modeled by a Pareto distribution. In a recent study, Andersen et al in [3] confirmed Paxson's earlier results but showed that the average loss rate for their measurements in 2003 was a low 0.42%. Most of the time, the 20-minute average loss rates were close to zero; over 95% of the samples had a 0% loss rate. On the other hand, during the worst one-hour period monitored, the average loss rate was over 13%. An important finding in [3] is that the conditional probability that a second packet is lost given that the first packet was lost was 72% for packets sent back-to-back and 66% for packets sent with a 10-msec delay, confirming the results in [10].

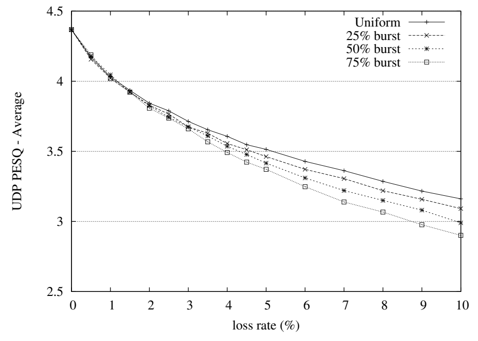

Fig. 1. Network loss - Average PESQ

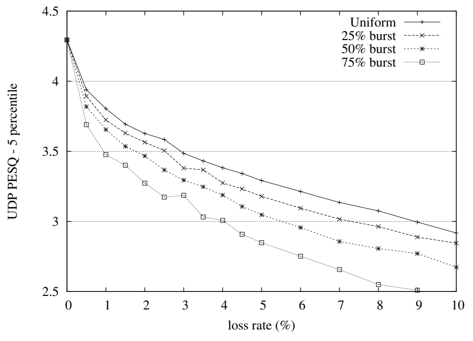

Fig. 2. Network loss - 5 percentile PESQ

In addition to link errors and equipment failures, the other major factor contributing to packet losses in Internet is *delayed convergence* of the existing routing protocols. Labovitz et al [11] use a combination of measurements and analysis to show that inter-domain routes in the Internet may take tens of minutes to reach a consistent view of the network topology after a fault. They found that during this period of delayed convergence, end-to-end communication is adversely affected. In [12], Labovitz et al. find that 10% of all considered routes were available less than 95% of the time and that less than 35% of all routes were available more than 99.99% of the time. In a followup study [13], Chandra et al showed that 5% of all failures last more than 2 hours and that failure durations are heavy-tailed and can last as long as 20 hours before being repaired. In a related study performed in 2003, Andersen et al [3] showed that while some paths are responsible for a large number of failures, the majority of the observed Internet paths had some level of instability. All these statistics indicate the Internet today is not ready to support high quality voice service as we are going to show in the following section.

C. *Voice quality degradation with loss*

We evaluated the effect of a loss pattern such as the one reported on the Internet on the VoIP quality, using the the standardized PESQ measure. To do so, we instantiated a network with various levels of loss and burstiness (we define burstiness as the conditional probability of loosing a packet when the previous packet was lost) in the Emulab [5] testbed, and measured the quality degradation when sending a VoIP stream on that network.

Emulab is a testing environment that allows deployment of networks with given characteristics composed of real computers running Linux, connected through real routers and switches. Link latency, capacity and loss[2] are emulated using additional computers that delay packets or drop them with certain probability or if their rate exceeds the requested link capacity. All the Emulab machines are also directly connected through a local area network through which they are managed and can be accessed from the Internet. On this local area network we constantly monitored the clock synchronization between the computers involved in our experiments and accurately adjusted our one-way latency measurements.

[2]Emulab cannot set conditional loss probability on the links. For burstiness experiments we dropped packets with conditional probability at the application level, before processing them.

We used the G.711 codec with PLC to transfer a 5 minute audio file using UDP over the lossy network, repeating each experiment for 20 times. The network had a 50 msec delay and 10 Mbps capacity, enough to emulate a trans-continental long-distance call over a wide area network. We finally decoded the audio file at the destination, divided it into 12 second intervals corresponding to normal conversation sentences, and compared each sentence interval with the original to generate its PESQ score.

Figure 1 shows the average PESQ score of all the sentence intervals as a function of loss rate and burstiness of the link. We can see that on average, the G.711 codec can handle up to 1% loss rate, while keeping a PESQ score higher than 4.0 (the expected PSTN quality level). Burstiness does not play a major role until the loss rate is relatively high, when the voice quality is anyway low. However, given the regular expectancy of high quality phone calls, we also need to analyze the most affected voice streams. Figure 2 presents the 5 percentile of the above measurements. We can see that for the most affected streams burstiness does have a significant impact, and even at 0.5% loss rate the G.711 codec cannot provide PSTN standard voice quality. At 0.5% loss and 75% burstiness the PESQ score dropped to 3.69.

Considering the fact that current loss rate measurements in the Internet average at about 0.42% with an average burstiness of 72%, and that occasionally loss can be even much higher, these experiments show that new solutions are required to improve the quality of VoIP traffic if it is to compete with the existing PSTN.

III. An Overlay Architecture

Overlay networks allow easy deployment of new services, as they allow full control over the protocols running between participating nodes. While the Internet provides generic communication solutions that are not tailored to a specific application, an overlay network usually has a limited scope and therefore can deploy application aware protocols.

The use of overlay networks usually comes with a price, partially due to the management overhead of the overlay, but mostly due to sub-optimal placement of the overlay routers in the physical network topology. However, overlays are small compared to the global underlying network, and therefore protocols that exploit the relatively limited size and scope of overlays not only can overcome their drawbacks, but can actually offer better performance to end-user applications.

A. Spines

Spines [4] is an open source overlay network that allows easy deployment and testing of overlay protocols. It runs in user space, does not need root access or kernel modifications, and encapsulates packets on top of UDP. Spines offers a two-level hierarchy in which applications (clients) connect to the closest overlay node, and then the node is responsible for forwarding and delivering data to the final destination through the overlay network. The benefit of this hierarchy is that it limits the size of the overlay network, thus reducing the amount of control traffic exchanged.

Overlay nodes act both as servers (accepting connections from various applications) and as routers (forwarding packets towards clients connected to other overlay nodes). Applications may reside either locally with the Spines nodes or on machines different than the overlay node they connect to.

In order to connect to a Spines overlay node, applications use a library that enables UDP and TCP communication between the application and the selected Spines node. The API offered by the Spines library closely resembles the Unix socket interface, and therefore it is easy to port any application to use Spines. We describe in Section VI the necessary steps to adapt current VoIP applications to use Spines. Each application is uniquely identified by the IP address of the overlay node it connects to, and by an ID given at that node, which we call *Virtual Port*. Spines provides both reliable and best-effort communication between end applications, using the applications' node IP address and the *Virtual Port* in a way similar to TCP and UDP. Similar to the *socket()* call, a *spines_socket()* function returns a descriptor that can be used for sending and receiving data. A *spines_sendto()* call resembles the regular *sendto()*, and a *spines_recvfrom()* resembles the regular *recvfrom()*, with similar parameters. *Virtual Ports* are only defined in the context of an overlay node, and have no relation to the actual operating system ports.

Spines nodes connect to each other using *virtual links* forming the overlay network. Spines offers a number of protocols on each virtual link, including a best effort service, a TCP-fair reliable protocol [14] and a real time recovery protocol that we describe below in section IV-A.

Each overlay node pings its direct neighbors periodically to check the link status and latency. Round trip time measurements are smoothed by computing a 5%-95% decaying average. Spines nodes add a link specific sequence number on every data packet sent between two neighboring overlay nodes. The receiving overlay node uses this sequence number to estimate link loss rate. The loss rate is computed by averaging the number of packets received between two subsequent loss events over the last L loss events (in our implementation $L = 50$). This way, the loss estimate converges relatively fast when loss rate increases (less number of packets will be received between two loss events), but is conservative in switching to opportunistic low-loss overlay links. Based on link loss and latency, a cost for each link is computed as described in Section IV-B and propagated through the network by an incremental link-state mechanism that uses reliable control links created among neighboring Spines nodes.

The control traffic required for maintaining the overlay network is small compared to the overall data traffic, consisting in our implementation of periodical *hello* messages and small *link updates*. One 32 byte hello message is sent every second by each of the two end-nodes of a direct link. A single link update is propagated to all the nodes in the overlay through a reliable flooding algorithm only in case of a network change, such as a variation in the estimation of delay or loss rate of the link, or when a link or node goes down. On the initial state transfer, when a new node is brought up, as well as in the case of multiple network events that happen simultaneously, multiple link updates are aggregated, so that a regular Ethernet packet can carry between 60 and 90 distinct updates, depending on the sparsity of the network. In the current implementation, Spines scales to up to several hundred overlay nodes, and up to one thousand clients per node.

IV. Protocols for increasing VoIP performance

Traditional VoIP systems use the UDP best effort delivery service to transfer data, exposing the audio channels to packet losses and path failures. One of the main reasons

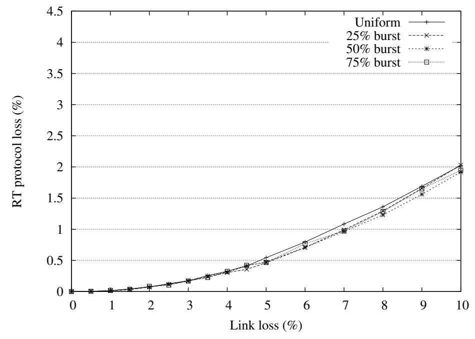

Fig. 3. Real-time recovery loss - 1 link

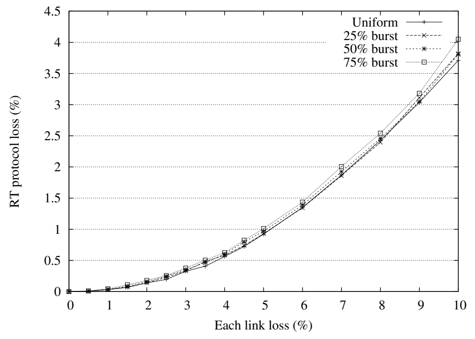

Fig. 4. Real-time loss recovery - 2 concatenated links

for not using packet retransmissions is that lost packets, even when recovered end-to-end from the source, are not likely to arrive in time for the receiver to play them. Overlay networks break end-to-end streams into several hops, and even though an overlay path may be longer than the direct Internet path between the two end-nodes, each individual overlay hop usually has smaller latency, thus allowing localized recovery on lossy overlay links.

A. *Real-time recovery protocol*

Our overlay links run a real-time protocol that recovers packets only if there is a chance to deliver them in time, and forward packets even out of order to the next hop. We describe our real time recovery protocol as follows:

- Each node in the overlay keeps a circular packet buffer per outgoing link, maintaining packets sent within a time equal to the maximum delay supported by the audio codec. Old packets are dropped out of the buffer if they are expired, or when the circular buffer is full.
- Intermediate nodes forward packets as they are received, even out of order.
- Upon detecting a loss on one of its overlay links, a node asks the upstream node for the missed packet. A retransmission request for a packet is only sent once. We only use negative acknowledgments, thus limiting the amount of traffic when no packets are lost.
- When an overlay node receives a retransmission request it checks in its circular buffer, and if it has the packet it resends it, otherwise it does nothing. A token bucket mechanism regulates the maximum ratio between the number of retransmissions and the number of data packets sent. This way we limit the number of retransmissions on lossy links.
- If a node receives the same packet twice (say because it was requested as a loss, but then both the original and the retransmission arrive), only the first instance of the packet will be forwarded towards the destination.

The protocol does not involve timeouts and never blocks for recovering of a packet. The downside is that this is not a fully reliable protocol and some of the packets will be lost in case the first retransmission attempt fails. Such events can appear when a packet is lost, the next packet arrives and triggers a retransmission request, but the retransmission request is also lost. For a symmetric link with independent loss rate p in both directions, this happens with probability: $p\cdot(1-p)\cdot p = p^2-p^3$. Another significant case is when the retransmission request does arrive, but the retransmission itself is lost, which can happen with probability: $p\cdot(1-p)\cdot(1-p)\cdot p = p^2-2p^3+p^4$. Other types of events, that involve multiple data packets lost can happen, but their probability of occurrence is negligible. We approximate the loss rate of our real-time protocol by $2p^2 - 3p^3$, assuming a uniform[3] loss probability on the link.

The delay distribution of packets follows a step curve, such that for a link with delay T and loss rate p, $(1-p)$ fraction of packets arrive in time T, $(p - 2p^2 + 3p^3)$ are retransmitted and arrive in time $3T + \Delta$, where Δ is the time it takes the receiver to detect a loss, and $(2p^2 - 3p^3)$ of the packets will be lost by the real time recovery protocol. For a path that includes multiple links, the delay of the packets will have a a composed distribution given by the combination of delay distributions of each link of the path. The time Δ it takes the receiver to trigger a retransmission request depends on the inter-arrival time of the packets (the receiver needs to receive a packet to know that it lost the previous one) and on the number of out of order packets that the protocol can tolerate. For a single VoIP stream, packets usually carry 20 msec of audio, so they arrive at relatively large intervals. In our overlay approach, we aggregate multiple voice streams sent

[3]In many cases, the loss rate probability may not be uniform. Later in the paper, we investigate the impact of burstiness on our protocols.

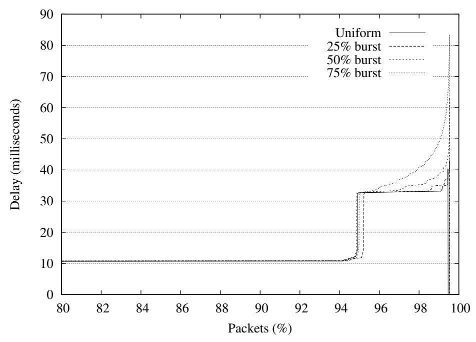

Fig. 5. Delay distribution - 1 link, 5% loss

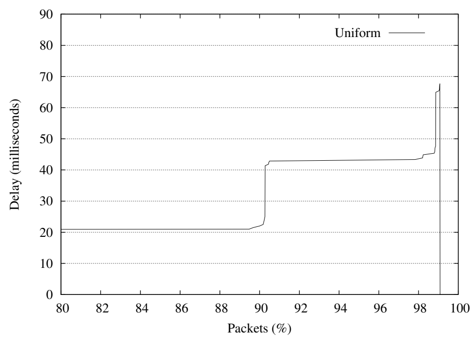

Fig. 6. Delay distribution - 2 concatenated links, 5% loss each

by different applications within a single real-time recovery protocol on each link. The overlay link protocol handles packets much more often than a single VoIP stream, and therefore the inter-arrival time of the packets is much smaller. Standard TCP protocol needs three packets out of order before triggering a loss. Since latency is crucial for VoIP applications, and as packet reordering happens relatively rarely [15], in our experiments we trigger a retransmission request after receiving the first out of order packet.

We implemented the real time protocol in the Spines overlay network platform and evaluated its behavior by running Spines on Emulab. Figure 3 shows the loss rate of the real time recovery protocol on a symmetric 10 msec link with various levels of loss and burstiness, and Figure 4 shows the combined loss for two concatenated 10 msec links that experience the same amount of loss and burstiness, in both directions, running Spines with the real-time protocol on each link. For each experiment, an application sent traffic representing the aggregate of 10 VoIP streams for a total of two million packets of 160 bytes each, and then average loss rate was computed. We can see that the level of burstiness on the link does not affect the loss rate of the real-time protocol. The real-time loss rate follows a quadratic curve that matches our $2p^2 - 3p^3$ estimate. For example, for a single link with 5% loss rate, applying the real-time protocol reduces the loss rate by a factor of 10, to about 0.5% regardless of burstiness, which yields an acceptable PESQ score (see Figure 1).

For the single 10 msec link experiment with 5% loss rate, the packet delay distribution is presented in Figure 5. As expected, 95% of the packets arrive at the destination in about 10 milliseconds. Most of the losses are recovered, showing a total latency of 30 msec plus an additional delay due to the inter-arrival time of the packets required for the receiver to detect the loss, and about 0.5% of the packets are not recovered. In the case of uniform loss probability the delay of the recovered packets is almost constant.

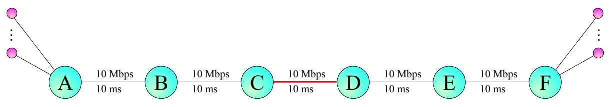

Fig. 7. Spines network - 5 links

However, when the link experiences loss bursts, multiple packets are likely to be lost in a row, and therefore it takes longer for the receiver to detect the loss. The increase of the interval Δ results in a higher delay for the recovered packets. Obviously, the higher the burstiness, the higher the chance for consecutive losses, and we can see that the packet delay is mostly affected at 75% burstiness.

Figure 6 shows the delay distribution for the two-link network, where both links experience 5% uniform distribution loss rate. As in the single link experiment, most of the losses are recovered, with the exception of 1% of the packets. We notice, however, a small fraction of packets (slightly less than 0.25%) that are lost and recovered on both links, and that arrive with a latency of about 66 msec. This was expected to happen with the compound probability of loss on each link, $p_c = 0.05 \cdot 0.05$. Burstiness results for the two-link network, not plotted in the figure, follow the same pattern as shown in Figure 5.

In order to evaluate the effect of local recovery on voice quality we ran the same experiment depicted in Figure 1 and Figure 2 on top of a Spines overlay network. We divided the 50 msec network into 5 concatenated 10 msec links as shown in Figure 7, ran Spines with the real-time protocol on each link, and sent 10 VoIP streams in parallel from node A to node F. We generated losses with different levels of burstiness on the middle link $C - D$ and set the threshold network latency for the G.711 codec to be 100 msec.

Figure 8 presents the average PESQ score of the G.711 streams using Spines, and compares it with the results obtained when sending over UDP directly. Since most of the packets are received in time to be decoded at the re-

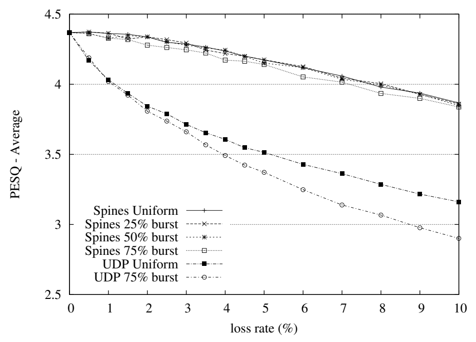

Fig. 8. Real-Time protocol - Average PESQ

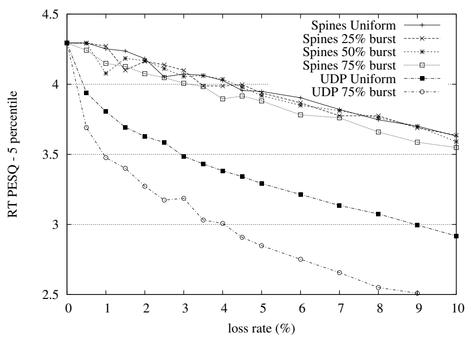

Fig. 9. Real-Time protocol - 5 percentile PESQ

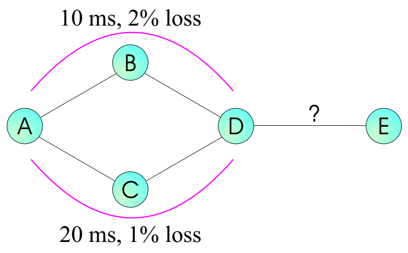

Fig. 10. Two-metric routing decision

ceiver, we can see that when using Spines, regardless of burstiness, the G.711 codec can sustain on average even network losses of 7% with PSTN voice quality.

As discussed earlier, users of phone services expect high quality service. Therefore, in addition to average performance characteristic, it is important to look at the performance of the worst cases. Figure 9 shows the quality of the worst 5 percentile calls. We can see that the codec can handle up to 3.5% losses with PSTN quality and even for the worst 5% calls, the burstiness did not play a major role in the voice quality.

B. *Real time routing for audio*

The real time protocol recovers most of the missed packets in case of occasional, or even sustained periods of high loss, but if the problem persists, we would like to adjust the overlay routing to avoid problematic network paths.

Given the packet delay distribution and the loss rate of the soft real-time protocol on each overlay link, the problem is how to find the overlay path between a pair of source and destination, for which the packet delay distribution maximizes the number of packets that arrive within a certain delay, so that the audio codec can play them. The problem is not trivial, and deals with a two metric routing optimizer. For example, in Figure 10, assuming a maximum delay threshold for the audio codec to be 100 msec, if we try to find the best path from node A to node E, even in the simple case where we do not recover packets, we cannot determine which partial path from node A to node D is better (maximizes the number of packets arriving at E within 100 msec) without knowing the latency of the link D-E. On the other hand, computing all the possible paths with their delay distribution and choosing the best one is prohibitively expensive.

However, if we can approximate the cost of each link by a metric dependent on the link's latency and loss rate, taking into account the characteristics of our real-time protocol and the requirements of VoIP, we can use this metric in a regular shortest path algorithm with reasonable performance results. Our approach is to consider that packets lost on a link actually arrive, but with a delay T_{max} bigger than the threshold of the audio codec, so that they will be discarded at the receiver. Then, the packet delay distribution of a link will be a three step curve defined by the percentage of packets that are not lost (arriving in time T), the percentage of packets that are lost and recovered (arriving in $3T + \Delta$), and the percentage of packets missed by the real-time protocol (considered to arrive after T_{max}). The area below the distribution curve represents the expected delay of the packets on that link, given by the formula: $T_{exp} = (1-p) \cdot T + (p - 2p^2 + 3p^3) \cdot (3T + \Delta) + (2p^2 - 3p^3) \cdot T_{max}$. Since latency is additive, for a path consisting of several links, our approximation for the total expected delay will then be the sum of the expected delay of each individual link. We call this metric *expected latency* cost function.

We evaluated the performance of the *expected latency* based routing and compared it with other cost metrics. We used the BRITE [16] topology generator to create random topologies using the Waxman model, where the probability to create a link depends on the distance between the nodes. We chose this model because it generates mostly short links that that fit our goal for localized recovery. We assigned random loss from 0% to 5% on half of the links of each topology, selected randomly. We considered every node generated by BRITE to be an overlay node, and every link to be an overlay edge. For each topology we

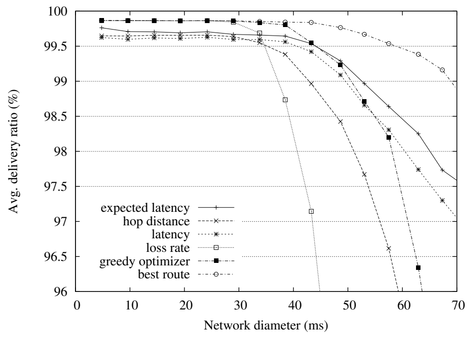

Fig. 11. Comparing routing metrics - 15 node networks

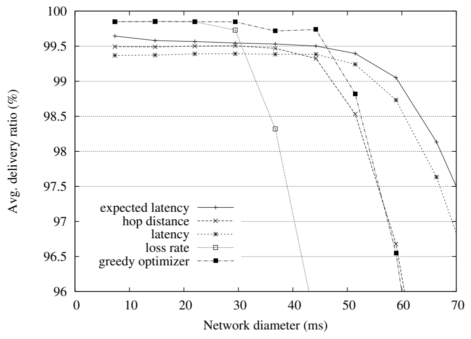

Fig. 12. Comparing routing metrics - 100 node networks

determined the nodes defining the diameter of the network (the two nodes for which the shortest latency path is longest), and determined the routing path between them given by different cost metrics.

By adjusting the size of the plane in which BRITE generates topologies, networks with different diameters are created. For each network diameter we generated 1000 different topologies and evaluated the packet delivery ratio between the network diameter nodes when running the real-time protocol on the links of the network, using different routing metrics. Figure 11 shows the average delivery ratio for network topologies with 15 nodes and 30 links, and Figure 12 shows the delivery ratio for network topologies with 100 nodes and 200 links. For a link with direct latency T and loss rate p, considering an audio codec threshold T_{max} = 100 msec and the packet interarrival time Δ = 2 msec, the cost metrics used are computed as follows:

- Expected latency: $Cost = (1-p) \cdot T + (p - 2p^2 + 3p^3) \cdot (3T + \Delta) + (2p^2 - 3p^3) \cdot T_{max}$
- Hop distance: $Cost = 1$
- Link latency: $Cost = T$
- Loss rate: $Cost = -log(1-p)$
- Greedy optimizer: We used a modified Dijkstra algorithm that, at each iteration, computes the delay distribution of the selected partial paths and chooses the one with the maximum delivery ratio.
- Best route: All the possible paths and their delay distributions were computed, and out of these the best one was selected. Obviously, this operation is very expensive, mainly because of the memory limitation of storing all combinations of delay distributions . Using a computer with 2GB memory we could not compute the best route for networks with more than 16 nodes.

As expected, for small diameter networks the loss-based routing achieves very good results, as the delay of the links is less relevant. With the increase in the network diameter, the latency-based routing achieves better results. At high latencies, the packet recovery becomes less important than the risk of choosing a highly delayed path, with latency more than the codec threshold. The expected latency routing achieves lower delivery ratio than the loss-based routing for small diameter networks, but behaves consistently better than the latency-based routing. The slight drop in delivery ratio for low diameter networks is causing just a small change in VoIP quality (see Figures 1 and 2), while the robustness at high network delays is very important. Interestingly, the greedy optimizer fails at high latency networks, mainly due to wrong routing decisions taken early in the incremental algorithm, without considering the full topology.

Our conclusion is that the *expected latency* metric, while being slightly worse than other routing metrics for small diameter networks achieves better routing in high latency networks, exactly where we need it the most.

V. Routing limitations of overlay networks

Running overlay nodes in user level space gives us great flexibility and usability, but comes at the expense of packet processing through the entire networking stack, and process scheduling on the machines running the overlay nodes.

Executing overlay network functionality on loaded computers naturally degrades the performance of the overlay system. This degradation is critical especially for latency sensitive VoIP streams. For example, if the overlay daemon runs as a user level process on a computer that has other CPU intensive processes, it is common for the overlay network system not to be scheduled for several hundred milliseconds, and even seconds, which of course, is not useful for VoIP. In fact, our experience with another messaging system, the Spread toolkit [17] that is commonly deployed on large websites, shows that on heavily loaded web servers a process may be scheduled only after eight seconds. It is common practice on such systems to assign the messaging system higher priority (real time priority in Linux). For a VoIP service it is reasonable to

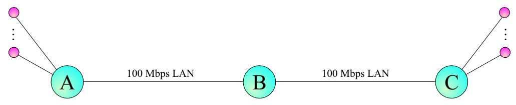

Fig. 13. Spines network - 2 links

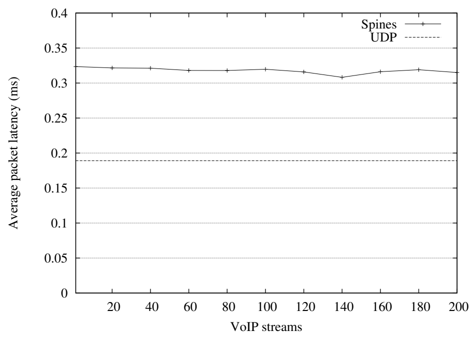

Fig. 14. Spines forwarding delay

expect that the overlay nodes will be well provisioned in terms of CPU and networking capabilities.

A. *Routing performance in Spines*

It is interesting to evaluate the overhead of running the overlay nodes as regular applications in the user space, and how the routing performance is affected by the amount of traffic or load on the computers. We deployed Spines in Emulab on a three node network as shown in Figure 13, where the middle node B had two network interfaces and was directly connected to nodes A and C through local area links. All the computers used were Intel Pentium III 850MHz machines. The one-way UDP latency between node A and node C measured sending 160 byte packets and adjusted with the clock difference between the nodes was 0.189 msec.

We ran a Spines node on each node, using the real-time protocol on the links $A-B$ and $B-C$. Then we sent a varying number of voice streams in parallel (from 1 to 200 streams), consisting of 20000 packets of 160 bytes each, from node A to node C using Spines. We measured the one way latency of each packet, adjusted by the clock difference between machines A and C. When forwarding 200 streams, the middle node B running Spines showed an average CPU load of about 40%. However, the sending node A, on which both Spines and our sending application were running, reached a maximum 100% CPU utilization.

Figure 14 shows the average latency of packets forwarded through Spines as the number of parallel streams increases from 1 to 200, and compares it to the base network latency measured with UDP probes. The standard deviation of all the measurements was a very low 0.012, and the highest single packet latency measured, which happened when we sent 200 streams in parallel, was 0.963 msec. What we see is that regardless of the number of streams, the three Spines nodes add a very small delay totaling about 0.15 msec due to user-level processing and overlay routing.

We evaluated the routing performance of Spines on a CPU loaded computer by running a simple *while(1)* infinite loop program on the middle node B, and repeated the above experiment. Running Spines with the same priority as the loop program, when forwarding a single voice stream we achieved a very high packet delay average of 74.815 msec, and the maximum packet delay was 154.468 msec. When competing with 4 loop programs in parallel, with the same priority as Spines, the average packet delay for a single stream went up even more to 298.354 msec (about 900 times more than without CPU competing applications), and the maximum packet delay was 604.917 msec. Obviously, such delays are not suitable for VoIP. However, when we set real-time priority for the Spines process, the high CPU load did not influence our performance. Even when competing with 10 loop programs and a load of 200 streams, the average packet delay was a low 0.315 msec and the maximum packet delay measured was 0.469 msec.

TABLE I
PLANETLAB SITES

1	CMU	9	UC San Diego
2	Columbia	10	U. of Georgia
3	Dartmouth College	11	U. of Maryland
4	Duke	12	U. of Oregon
5	Intel Resrc. Berkeley	13	U. of Virginia
6	Intel Resrc. Seattle	14	U. of Washington
7	MIT	15	U. of Utah
8	Princeton		

B. *Case study: Planetlab*

Planetlab [6] is a large overlay testbed that can be seen as a collection of computers distributed around the world, each of them directly accessing the Internet. Currently Planetlab has 403 nodes at 169 sites. As opposed to Emulab, which has a reservation mechanism that completely allocates computers to a particular experiment, Planetlab uses a shared environment. Users create *slices* on each computer they need to run their programs on, and each slice acts like a virtual machine, sharing the computer resources with other users, running within their own slices. In the current deployment of Planetlab it is not possible to set process priorities higher than other processes in other

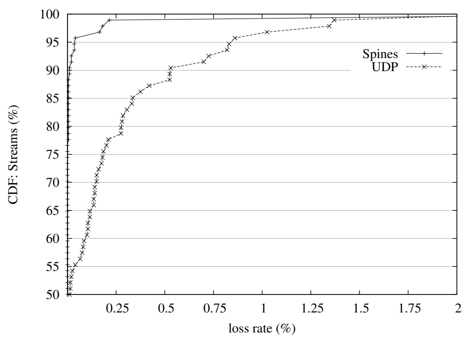

Fig. 15. Planetlab - lost packets

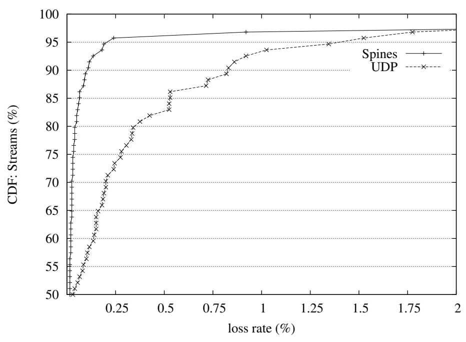

Fig. 16. Planetlab - missed packets

slices, and the CPU load and availability on the machines is dependent on the particular applications that various researchers run at the same time.

Out of the sites in the US we found 15 to have computers synchronized to under 30 msec. The 15 sites are presented in Table I. Note that our overlay protocols do not require clock synchronization, but we do need synchronized clocks in our experiments to evaluate the number of packets arriving within a certain delay constraint. The average load measured on the machines at these sites varied from 0.56 to 5.95, with an average of 2.91 (i.e. at any point in time there are on average 2.91 processes ready and competing for CPU), with only 2 nodes having a load below 1. Such a high load is expected and normal for a shared testbed, and can be easily handled by Spines if we run it with real-time priority. However, as we cannot set higher process priorities against users running in different slices, we expect that any overlay path that uses at least an intermediate high CPU loaded node on Planetlab, would delay much more packets than it can recover using the real-time protocol, and therefore behave worse than the direct Internet connection between the end nodes.

It is interesting though to see how the real time recovery protocol behaves, even at high CPU load, on the direct connections between the Planetlab nodes. We deployed a Spines overlay network consisting of a fully connected graph, such that each of the 15 overlay nodes had a direct overlay link to each of the other 14 nodes. We then sent streams of 20000 packets, 160 byte each at a rate of 50 packets/sec from each node to all the other nodes, both directly using UDP and using Spines. We combined nodes in alternating groups of 7 pairs, such that each node can only be a sender or a receiver at a time, but not both. We also alternated UDP and Spines streams, so that we minimized as much as possible the effect of loss or delay variations between the same nodes when running different streams.

We considered only 94 streams for which the minimum packet delay was less than 30 msec (so that the real-time protocol had a chance to recover any packets) and set the delay threshold for the codec to be 115 msec (we add 15 milliseconds to compensate for the 0 to 30 msec clock difference between the nodes). For each UDP and Spines stream we counted the lost packets, the late packets (arriving at destination after more than 115 msec) and the packets missed by the codec as the sum of both lost and late packets.

Figure 15 presents the CDF of the streams as a function of lost packets. First, we see that many of the UDP streams in Planetlab are affected by loss. About 22% of the UDP streams lost more than 0.25% of the packets, and about 4% of the UDP streams lost more than 1% of their packets. Spines all the packets for most of the streams, such that there was only one stream that lost more than0.25% of the packets. That stream lost 609 packets total, most of them in three bursts of 102, 359 and 87 packets respectively. We believe this happened either because of short outages between Planetlab nodes, but most likely because Spines was not scheduled to run for several seconds.

Figure 16 presents the CDF of the streams as a function of missed packets. Even though Spines did recover almost all of the packets, some of the recoveries did not arrive in time at the destination, also due to process scheduling delays when sending retransmissions. Nevertheless, while almost 27% of the UDP streams missed more than 0.25% of the packets, only about 4% of the Spines streams had more than 0.25% of the packets missed.

VI. Integration with the current infrastructure

Given the large installed base of VoIP end clients and the even larger planned future deployments it is imperative that our system integrates seamlessly with the existing infrastructure. We explain what are the necessary steps to achieve this in the rest of this section.

The first component of the integration has to do with how VoIP clients are able to find their closest Spines server. We assume that each domain that wants to take advantage of the benefits provided by our system will deploy a Spines node as part of their infrastructure. In this case VoIP clients can use DNS SRV [18] records to locate the Spines node that is serving their administrative domain. This DNS query will return the IP address of (at least one) Spines node that can serve as their proxy in the Spines overlay. Once this node is found the VoIP clients can communicate with it using the interface we described in Section III-A. Then, the VoIP clients have to direct media traffic to flow through the Spines network rather than directly with over UDP.

We have two proposed solutions for this issue: one that requires changes to the end-clients and one that does not. We begin with the solution that requires "Spines-enabled" clients. In the current architecture, the Session Initiation Protocol (SIP) [19] is used as a signaling protocol so that the two communication endpoints can negotiate the session parameters, including the IP address and ports that each client is waiting to receive media traffic on. A Spines-enabled VoIP client announces its capability using the *parameter negotiation* feature that is part of SIP, within the initial INVITE SIP message. The VoIP client includes in the same INVITE message the IP address and the *Virtual Port* that it's Spines server is waiting to receive media traffic for the client. If the peering VoIP client is also Spines-enabled it will reply positively and include in its reply the address and *Virtual Port* at its own server. On the other hand, if the peer is not Spines-enabled it will return an error code indicating to the session initiator that it will have to revert to a "legacy" session. After the SIP negotiation has successfully finished, each source will send media traffic through its local Spines server towards the Spines server indicated by the peer client. As the traffic is forwarded through the overlay network, the egress Spines node will finally deliver it to the destination VoIP client.

RTP [20] and RTCP data is sent seamlessly through the Spines network, offering the end clients information about the network conditions along the overlay path they use.

While this first solution is *architecturally pure*, it requires changes to the end clients which may not be initially possible. In this case, we propose to use a solution similar to the *NAT-traversal* in SIP [21]. Specifically, Spines nodes will be required to intercept SIP INVITE messages and change the IP address and ports to point to themselves rather than to the VoIP clients. This way all the media traffic will flow through the Spines network which will eventually deliver it to the end-hosts.

VII. Related work

Our goal in this work is to reduce the effect of Internet losses on the quality of VoIP traffic. We do so by using an overlay network that attempts to quickly recover lost packets by using limited hop-by-hop retransmissions and an adaptive routing algorithm to avoid persistently lossy links. In this respect our work is related with techniques that try to reduce the loss rate of underlying Internet paths and with other work in overlay networks.

Multi Protocol Label Switching (MPLS) [22] has been recently proposed as a way to improve the performance of underlying network. This is done by pre-allocating resources across Internet paths (LSPs in MPLS parlance) and forwarding packets across these paths. Our system is network agnostic and therefore does not depend on MPLS, but it can leverage any reduction in loss rate offered by MPLS. At the same time, MPLS will not eliminate route and link failures or packet loss. Since it runs at a higher level, our overlay network can continue to forward packets avoiding failed network paths. Forward Error Correction (FEC) schemes [23] have also been proposed as a method of reducing the effective loss rate of lossy links. These schemes work by adding redundant information and sending it along with the original data, based on the feedback estimate of loss rate given by RTCP, such that in case of a loss, the original information (or part of it) can be recreated. Most of the VoIP solutions today (including the G.711 codec we use in this paper) use some form of FEC to ameliorate the effect of loss. Given the occasional bursty loss pattern of the Internet, many times the FEC mechanisms are slow in estimating the current loss rate, and therefore we believe that localized retransmissions are required for maintaining voice quality. Moreover, since our approach increases the packet delivery ratio, FEC mechanisms will notice a considerable reduction in loss, and therefore reduce their redundancy overhead.

Overlay networks have emerged as an increasingly growing field over the last few years, motivated mainly by the need to implement new services not supported by the current Internet infrastructure. Some of the pioneers of overlay network systems are X-Bone [24] and RON [25], which provides robust routing around Internet path failures. Other overlay networks focus on multicast and multimedia conferencing [26],[27]. Our work uses the same basic architecture of an overlay network but it is targeted towards the specific problems of VoIP traffic. Finally, OverQoS [28] is probably closest to our work, as it proposes an overlay link protocol that uses both retransmissions and FEC to provide loss and throughput guarantees. OverQoS depends on the existence of an external overlay system (the authors suggest RON as an option) to provide path selection and overlay forwarding. In this respect, our system can use OverQos as a plug-in module as an alternative to our real-time recovery protocol presented in Section IV-A, probably with the necessary modifications that take into account the special requirements of voice traffic.

VIII. Conclusion

In this paper we have shown that current conditions inhibit the deployment of PSTN quality Voice over IP, and we proposed a deployable solution that can overcome the bursty loss patern of the Internet. Our solution uses the open source Spines overlay network to segment end-to-end paths into shorter overlay hops and attempts to recover lost packets using limited hop-by-hop retransmissions. Spines includes an adaptive routing algorithm that avoids chronically lossy paths in favor of paths that will deliver the maximum number of voice packets within the predefined time budget. We evaluated our algorithms and our system implementation in controlled networking environments in Emulab, on the Internet using the Planetlab testbed, and through extensive simulations on various random topologies. Our results show that Spines can be very effective in masking the effects of packet loss, thus offering high quality VoIP even at loss rates higher than those measured in the Internet today.

References

[1] Athina Markopoulou, Fouad A. Tobagi, and Mansour J. Karam, “Assessing the quality of voice communication over internet backbones,” *IEEE/ACM Transactions On Networking*, vol. 11, no. 5, pp. 747–760, October 2003.

[2] Vern Paxson, “End-to-End Packet Dynamics,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 3, pp. 277–292, 1999.

[3] David G. Andersen, Alex C. Snoeren, and Hari Balakrishnan, “Best-Path vs. Multi-Path Overlay Routing,” in *Proceedings of IMC 2003*, Oct. 2003.

[4] “The Spines Overlay Network,” http://www.spines.org.

[5] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar, “An integrated experimental environment for distributed systems and networks,” in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, Dec. 2002, USENIX Association, pp. 255–270.

[6] L. Peterson, D. Culler, T. Anderson, and T. Roscoe, “A Blueprint for Introducing Disruptive Technology into the Internet,” in *Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I)*, Oct. 2002.

[7] “ITU-T Recommendation G.711: Pulse code modulation (PCM) of voice frequencies,” http://www.itu.int/rec/recommendation.asp?type=items&lang=E&parent=T-REC-G.711-198811-I.

[8] “ITU-T Recommendation G.711 appendix I: A high quality low-complexity algorithm for packet loss concealment with G.711,” http://www.itu.int/rec/recommendation.asp?type=items&lang=E&parent=T-REC-G.711-199909-I!AppI.

[9] “ITU-T Recommendation P.862: Perceptual evaluation of speech quality (PESQ),” http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-P.862-200102-I.

[10] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, “On the Constancy of Internet Path Properties,” in *Proceedings ACM SIGCOMM Internet Measurement Workshop*, Nov. 2001.

[11] C. Labovitz, A. Ahuja, and F. Jahanian, “Delayed Internet Convergence,” in *Proceedings of SIGCOMM 2000*, Aug. 2000.

[12] C. Labovitz, C. Malan, and F. Jahanian, “Internet Routing Instability,” *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 515–526, 1998.

[13] B. Chandra, M. Dahlin, L. Gao, and A. Nayate, “End-to-End WAN Service Availability,” in *Proceedings of 3rd USISTS*, Mar. 2001.

[14] Yair Amir and Claudiu Danilov, “Reliable communication in overlay networks,” in *Proceedings of the IEEE DSN 2003*, June 2003, pp. 511–520.

[15] Gianluca Iannaccone, Sharad Jaiswal, and Christophe Diot, “Packet reordering inside the Sprint backbone,” Tech. Rep. TR01-ATL-062917, Sprintlab, June 2001.

[16] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers, “BRITE: An approach to universal topology generation,” in *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems - MASCOTS ’01*, August 2001.

[17] “The Spread Toolkit,” http://www.spread.org.

[18] A. Gulbrandsen, P. Vixie, and L. Esibov, “A DNS RR for specifying the location of services (DNS SRV),” *RFC 2782*, Feb. 2000.

[19] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, “SIP: Session Initiation Protocol,” *RFC 3261*, June 2002.

[20] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A transport protocol for real-time applications,” RFC 1889, IETF, Jan. 1996.

[21] Fredrik Thernelius, “SIP, NAT and Firewalls,” Master’s thesis, Department of Teleinformatics, Kungl Tekniska Hgskolan, May 2000.

[22] E. Rosen, A. Viswanathan, and R. Callon, “Multiprotocol Label Switching Architecture,” *RFC 3031*, Jan 2001.

[23] Jean-Chrysostome Bolot, Sacha Fosse-Parisis, and Donald F. Towsley, “Adaptive FEC-based error control for internet telephony,” in *INFOCOM (3)*, 1999, pp. 1453–1460.

[24] J. Touch and S. Hotz, “The x-bone,” in *Third Global Internet Mini-Conference at Globecom ’98*, Nov. 1998.

[25] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, “Resilient overlay networks,” in *Proc. of the 18th Symposium on Operating Systems Principles*, Oct. 2001, pp. 131–145.

[26] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, “Scalable application layer multicast,” in *Proc. of ACM SIGCOMM*, 2002.

[27] Yang hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang, “Enabling conferencing applications on the internet using an overlay multicast architecture,” in *ACM SIGCOMM 2001*. ACM, Aug. 2001.

[28] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy Katz, “OverQoS: An Overlay Based Architecture for Enhancing Internet QoS,” in *USENIX NSDI ’04*, Mar. 2004.