Reliable Communication in Overlay Networks

Yair Amir and Claudiu Danilov
Johns Hopkins University
{yairamir, claudiu}@cs.jhu.edu

Abstract

Reliable point-to-point communication is usually achieved in overlay networks by applying TCP on the end nodes of a connection. This paper presents a hop-by-hop reliability approach that considerably reduces the latency and jitter of reliable connections. Our approach is feasible and beneficial in overlay networks that do not have the scalability and interoperability requirements of the global Internet.

The effects of the hop-by-hop reliability approach are quantified in simulation as well as in practice using a newly developed overlay network system that is fair with the external traffic on the Internet. The experimental results show that the overhead associated with overlay network processing at the application level does not play an important factor compared with the considerable gain of the approach.

1 Introduction

Reliable point-to-point communication is one of the main utilizations of the Internet, where over the last few decades TCP has served as the dominant protocol. Over the Internet, reliable communication is performed end-to-end in order to address the severe scalability and interoperability requirements of a network in which potentially every computer on the planet could participate. Thus, all the work required in a reliable connection is distributed only to the two end nodes of that connection, while intermediate nodes route packets without keeping any information about the individual packets they transfer.

Overlay networks are opening new ways to Internet usability, mainly by adding new services (e.g. built-in security) that are not available or cannot be implemented in the current Internet, and also by providing improved services such as higher availability [2]. However, the usage of overlay networks may come with a price, usually in added latency that is incurred due to longer paths created by overlay routing, and by the need to process the messages in the application level by every overlay node on the path.

Reliable communication in overlay networks is usually achieved by applying TCP on the edges of a connection. This surely works. However, this paper argues that employing hop-by-hop reliability techniques considerably reduces the average latency and jitter of reliable communication. When using such an approach one has to consider networking aspects such as congestion control, fairness, flow control and end-to-end reliability. We discuss these aspects and our design decisions in Section 2.

In Section 3, we demonstrate through simulation that our approach provides tremendous benefit for the application as well as for the network itself, even when very few packets are lost. Simulations usually do not take into account many practical issues such as processing overhead, CPU scheduling, and most important, the fact that overlay network processing is performed at the application level of general purpose computers. These may have considerable impact on real-life behavior and performance. Therefore, we test our approach in practice on an overlay network platform called Spines that we have built.

We introduce Spines in Section 4. Spines [16] is an open source research platform that allows deployment of overlay networks in the Internet. We run the same experiments that were simulated, on a Spines overlay network. The results are presented in Section 5. We show that the benefit of hop-by-hop reliability greatly overcomes the overhead of overlay routing and achieves much better performance compared to standard end-to-end TCP connections deployed on the same overlay network.

We describe existing related work and compare it with our approach in Section 6, and end the paper, concluding that hop-by-hop reliability is a viable and beneficial approach to reliable communication in overlay networks.

2 Hop-by-hop reliable communication in overlay networks

An overlay network constructs a user level graph on top of an existing networking infrastructure such as the Internet, using only a subset of the available network links and nodes. An overlay link is a virtual edge in this graph and

may consist of many actual links in the underlying network. Overlay nodes act as routers, forwarding packets to the next overlay link toward the destination. At the physical level, packets traveling along a virtual edge between two overlay nodes follow the actual physical links that form that edge.

Overlay networks have two main drawbacks. First, the overlay routers incur some overhead every time a message is processed, which requires delivering the message to the application level, processing it, and resending the message to the next overlay router. Second, the placement of overlay routers in the topology of the physical network is often far from optimal, because the creator of the overlay network rarely has control over the physical network (usually the Internet) or even the knowledge about its actual topology. Therefore, overlay networks provide longer paths that have higher latency than point to point Internet connections.

The easiest way to achieve reliability in Overlay Networks is to use a reliable protocol, usually TCP, between the end points of a connection. This mechanism has the benefit of simplicity in implementation and deployment, but pays a high price upon recovery from a loss. As overlay paths have higher delays, it takes a relatively long time to detect a loss, and data packets and acknowledgments are sent on multiple overlay hops in order to recover the missed packet.

2.1 Hop-by-hop reliability

We propose a mechanism that recovers the losses only on the overlay hop on which they occurred, localizing the congestion and enabling faster recovery. Since an overlay link has a lower delay compared to an end-to-end connection that traverses multiple hops, we can detect the loss faster and resend the missed packet locally. Moreover, the congestion control on the overlay link can increase the congestion window back faster than an end-to-end connection, as it has a smaller round-trip time.

Hop-by-hop reliability involves buffers and processing in the intermediate overlay nodes. These nodes need to deploy a reliable protocol, and keep track of packets, acknowledgments and congestion control, in addition to their regular routing functionality. Although such an approach may not be feasible to implement at the level of the Internet routers due to scalability limitations, we can easily deploy it at the level of an overlay network, thus allowing us to pinpoint the congestion, limiting the problem to the congested part of the network.

Let's consider a simple overlay network composed of five 10 millisecond links in a chain, as shown in Figure 1. Such a network may span a continent such as North America or Europe. Every time a packet is lost (say on link C-D), it will take at least 50 milliseconds from the time that packet was sent until the receiver detects the loss, and at least 50 additional milliseconds until the sender learns about

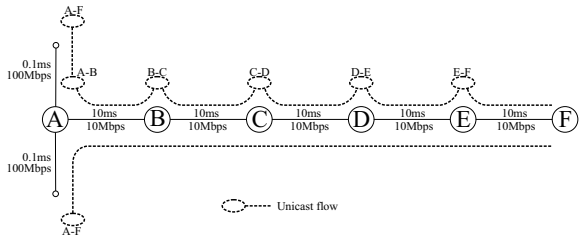

Figure 1. Chain Network Setup

it. The sender will retransmit the lost packet that will travel 50 more milliseconds until the receiver will get it. This accounts for a total of at least 150 milliseconds to recover a packet. If the sender continues to send packets during the recovery period, even if the new packets arrive at the receiver in time (assuming no loss for them), they will not be delivered at the receiver until the missing packet is recovered, as they are not in order. Our experimental results presented in Sections 3 and 5 show that the number of packets delayed is much higher than the number of packets lost.

Let us assume that we use five reliable hops of 10 milliseconds each instead of one end-to-end connection. Suppose the same message is lost on the same intermediate link, as in the above scenario. On that particular link (with 10 milliseconds delay) it will take only about 30 milliseconds for the receiver to recover the missed packet. Moreover, as the recovery period is smaller, a smaller number of out of order packets will be delayed. This effect is more visible as the throughput increases.

2.2 End-to-end reliability and congestion control

Simply having reliable overlay links does not guarantee end-to-end reliability. Intermediate nodes may crash, overlay links may get disconnected. However, such events are not likely to happen and most of the reliability problems (generated by network losses) are indeed handled locally at the level of each hop. Therefore we still need to send some end-to-end acknowledgments from the end-receiver to the initial sender, at least once per round-trip time, but not for every packet. This means that for some of the packets we will pay the price of sending two acknowledgments, one on each of the overlay hops for local reliability, and one end-to-end, that will traverse the entire path. However, acknowledgments are small and are piggy-backed on the data packets whenever possible. We believe that the penalty of sending double acknowledgments for some of the packets is drastically reduced by resending the missed data packets (which are much bigger than the acknowledgments) only locally, on the hop where the loss occurred, and not on the entire end-to-end path.

Intermediate overlay nodes handle reliability and con-

gestion control only for the links to their immediate neighbors and do not keep any state for individual flows in the system. Packets are forwarded and acknowledged per link, regardless of their originator. This is essensial for the scalability with the number of reliable sessions in the system.

Since the packets are not needed in order at the intermediate overlay nodes, but only at the final destination, in case of a loss there is no need to delay the following packets locally on each link in order to forward them FIFO on the next link. We choose to forward the packets even if out of order on intermediate hops, and reestablish the initial order at the end receiver.

Our tests show that out of order forwarding reduces the burstiness inside the network. It also contributes to the reduction of the end-to-end latency (although that contribution is not as significant as the latency reduction achieved by the hop-by-hop reliability). The latency effect of out of order forwarding is magnified when multiple flows use the same overlay link. In that case, they do not need to reorder packets with respect to each other but only according to their own packets. The same occurs when more than one overlay link is congested and looses packets.

Overlay links are seen as individual point-to-point connections by the underlying network. Since overlay flows coexist with external traffic, each overlay link needs to have a congestion control mechanism in place. Our approach uses a window-based congestion control on each overlay link, that very closely follows the slow start and congestion avoidance of TCP [11].

The available bandwidth is different on each overlay link, depending on the underlying network characteristics, and is also dynamic, as the overlay link congestion control adjusts to provide fairness with the external traffic. If, at an intermediate node, the incoming traffic is bigger than the outgoing available bandwidth of the overlay link, that node will buffer the incoming packets, but if the condition persists it will either store an infinite number of packets or will start dropping them. Since end-to-end recovery is expensive, there needs to exist a congestion control mechanism that will limit, or even better, avoid packet losses at the overlay level. As opposed to the regular mechanism in TCP that uses packet losses to signal congestion, we use an explicit congestion notification scheme [15] where congested routers stamp the header of the data packets. Upon receiving such a stamped packet, the end receiver will send an end to end acknowledgment signaling the congestion immediately, and the sender's congestion control will treat that acknowledgment as a loss, even though the sender will not resend the corresponding packet. Note that the initial sender still sends retransmissions if necessary (e.g. in case of node failures and rerouting).

Since end-to-end acknowledgments are not sent for every packet, the end-to-end window may advance in big chunks once a cummulative acknowledgment is received. If the network path is not congested, this phenomenon does not affect the burstiness of the traffic, as the sending throughput is anyway smaller than the size of the window. However, in case of congestion the receiver sends end-to-end acknowledgments for every packet (stamped by an intermediate overlay router) until the congestion is resolved.

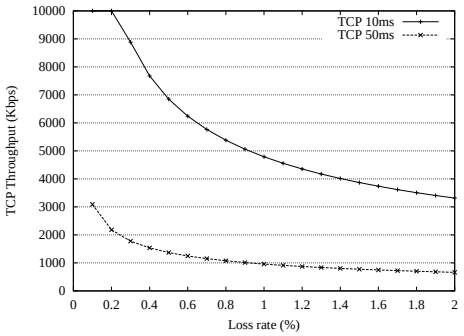

Figure 2. TCP throughput (analytical model)

2.3 Fairness

Since we intend to deploy our protocols on the Internet we need to share the global resources fairly with the external TCP traffic. A "TCP-compatible" flow is defined in [3] as one that is responsive to congestion notification, and in steady state, it uses no more bandwidth than a conformant TCP running under comparable conditions (loss rate, round-trip time, packet size, etc.).

The throughput obtained by a conformant TCP flow is evaluated analytically in [13], where the authors approximate the bandwidth B of a TCP flow as a function of packet size s, loss rate p and round-trip time RTT, where T_0 is the retransmission timeout and b is the number of packets that have to be received before sending an acknowledgment.

$$B = \frac{s}{RTT\sqrt{\frac{2bp}{3}} + T_0 3\sqrt{\frac{3bp}{8}}p(1+32p^2)}$$

Considering $b = 1$ and $T_0 = RTT$ in the ideal case, on a network topology such as in in Figure 1 the throughput obtained by an end-to-end TCP connection (50 millisecond delay) and by a short one hop TCP connection (10 millisecond delay on link CD) sending 1000 byte packets are shown in Figure 2 as a function of loss rate.

Clearly, an end-to-end reliable connection with a delay of 50 milliseconds will achieve less bandwidth than a hop-by-hop flow that will be limited only by the short bottleneck link C-D with 10 milliseconds delay, where the losses occur. This phenomenon happens because TCP throughput is biased against long connections. Analytically, RTT

appears at the denominator of the throughput formula, and in practice it will take more time for the long connection to recover its congestion window (the congestion avoidance protocol adds one to the congestion window for each RTT).

Note that achieving more throughput by a hop-by-hop flow does not happen with respect to external TCP connections that run outside of the overlay traffic. Each of the overlay links provides fairness and congestion control with respect to the external flows. A comparison of the throughput obtained by a single flow traversing multiple hops on the overlay network with one that uses the Internet directly cannot be done because of several factors:

- Flows that run within the overlay network usually have longer paths (higher delay) than direct Internet connections (due to the overlay routing which is usually far from optimal), and therefore achieve less throughput.
- In general, multiple connections coexist within an overlay network, so there is more than one stream using a single overlay link. In that case, multiple streams will share a single overlay link using only a part of what they could get if each of them used the Internet directly by opening a separate TCP connection. One way to overcome this problem is to open multiple connections between two overlay nodes depending on the number of internal flows using that overlay link. However, we see an overlay network as a single distributed application, no matter how many internal flows it carries; therefore, it should get only one share of the available bandwidth.

Some mechanisms can be deployed in order to limit the internal hop-by-hop throughput to the one obtained by an end-to-end connection that uses the overlay network. Such mechanisms can evaluate the loss rate and round-trip time of a path and adjust the sending rate accordingly, in a way similar to [7]. We believe such mechanisms are not necessary in our case - since we provide end-to-end congestion control, obtaining more throughput is just an effect of pinpointing the congestion and resolving it locally. However, in all the experiments of this paper we choose a conservative approach and limit the sending throughput to values achievable by both end-to-end and hop-by-hop flows, and focus only on the latency of the connections.

3 Simulation Environment and Results

In this section we analyze the multihop reliability behavior using the ns2 simulator [12]. We run a simple end-to-end TCP connection from node A to node F on a network setup as shown in Figure 1, while changing the packet loss rate on link C-D. Since this paper focuses on the latency of reliable connections, we limit the sending throughput to the same value for end-to-end and hop-by-hop flows in order to keep the same network parameters for our latency measurements.

We record the delay of each packet for the different sending rates and packet loss for both end-to-end and hop-by-hop reliability approaches. We define the delay of a packet as the difference between the time the packet was received at the destination, and the time it was initiated by a constant rate sending application. Note that there is a difference between the time a packet is sent by an application and the time that packet is actually put on the network by the reliable protocol (in our case, TCP). If TCP shrinks its window or reaches a timeout, it will not accept or send new packets until it has enough room for them. During this time, the new packets generated by the application will be stored in a buffer owned either by the host operating system or by the application itself. We believe that a delay measurement that is fair to the application would count the time spent by packets in these buffers as well.

The ns2 simulator offers a variety of TCP implementations. Out of these, we used TCP-Fack - TCP with forward acknowledgments - as we believe it resembles a behavior closest to the actual TCP implementation in the Linux Redhat 7.1, that we use in Section 5. The Linux kernel allows adjustment of different TCP parameters (for example, turning off forward acknowledgments would give us a version similar to TCP-SACK), however we opted for leaving the default protocol in the kernel unaltered.

Table 1 shows the average packet delay given by different TCP variations in ns2, as well as the Linux TCP implementation and the Spines link protocol (described in Section 4) when a 500Kbps stream is sent on an end-to-end A-F connection in the network showed in Figure 1, with link C-D experiencing 1% loss. The Redhat 7.1 TCP and the Spines link protocol delays were measured on an emulated network setup described in Section 5.

We compare the performance of the standard end-to-end approach to that of our hop-by-hop approach, where we forward packets reliably on each link, A-B, B-C, ... up to link E-F. For hop-by-hop reliability we use a modified version of TCP-Fack: the initial sender (at node A) adds its original sequence number in an additional packet header, intermediate receivers deliver packets out of order, and the destination delivers packets FIFO according to the original sequence number available in the new header. We did not change the congestion control or the send and acknowledge mechanisms in any way. We verified that our modified TCP and the original TCP-Fack in ns2 behave identically with respect to each packet on a point-to-point connection under different loss rates. All the simulations in this section were run for 5000 seconds, sending 1000 byte messages.

Figure 3 shows that the average delay for a 500 Kbps data stream increases faster with an end-to-end connection while a hop-by-hop flow maintains a low average delay even

Table 1. Average latency for under loss

Protocol	Tahoe	Reno	NewReno	SACK	Fack	Vegas	Redhat 7.1	Spines
Avg. delay (ms)	407.49	217.52	155.76	144.70	84.66	74.07	90.06	117.55

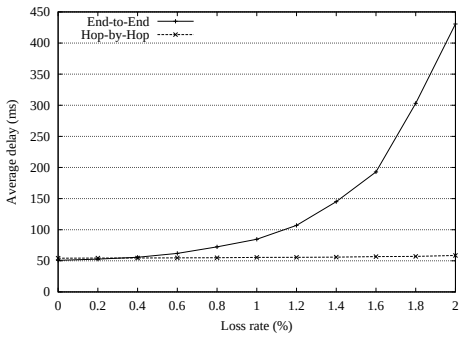

Figure 3. Average delay for a 500 Kbps stream (simulation)

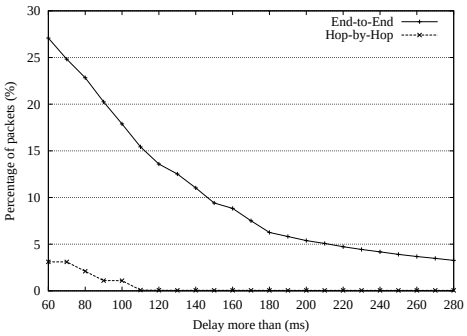

Figure 6. Packet delay distribution for a 500 Kbps stream (simulation)

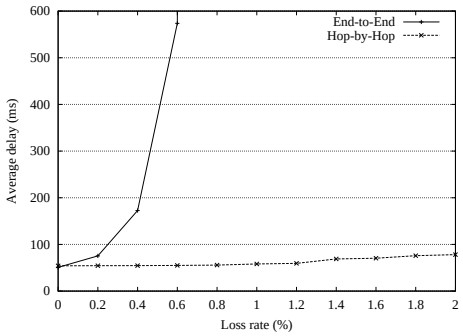

Figure 4. Average delay for a 1000 Kbps stream (simulation)

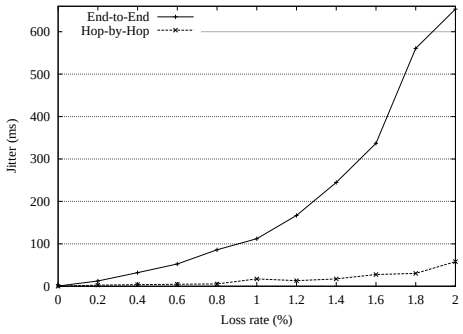

Figure 5. Average jitter for a 500 Kbps stream (simulation)

when it experiences a considerable loss rate. This phenomenon is magnified as the throughput required by the flow increases, as depicted by Figure 4 for a 1000 Kbps data stream.

Jitter is an important aspect of network protocols behavior due to its impact both on other flows at the network level and on the application served by the flow. Figure 5 shows that the jitter of an end-to-end connection is considerably higher and increases faster than the jitter of a hop-by-hop connection for a 500 Kbps stream. We computed the jitter as the standard deviation of the packet delay.

It is interesting to see the distribution of the packet delay for a certain loss rate. In Figure 6, we see that for a 500 Kbps data stream under 1% loss rate, over 27% of the packets are delayed more than 60 milliseconds (including the 50 milliseconds network delay) for an end-to-end connection, while for a hop-by-hop connection only about 3% of the packets are delayed more than 60 milliseconds. Similarly, about 18% of the packets are delayed more than 100 milliseconds by the end-to-end connection, while for a hop-by-hop connection only 1% of the packets are delayed as much. Note that the actual number of packets delayed is much higher than the number of packets lost.

We studied how the performance is affected by the number of intermediate reliable hops in an overlay network. We consider the same network of 50 milliseconds delay, and we measure the percentage of packets that are delayed as we increase the number of intermediate hops from 1 to 10, while keeping the total path latency constant. First, we use two hops of 25 milliseconds each, then three hops of 16.66 milliseconds each, and so forth. Figure 7 shows the per-

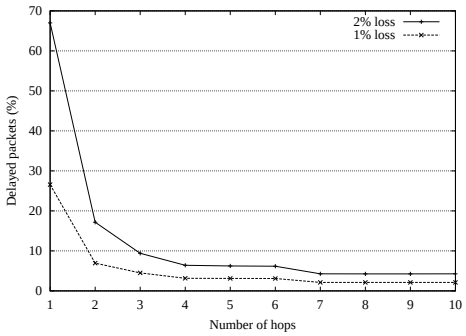

Figure 7. Increasing the number of hops (simulation)

centage of packets delayed more than 60 milliseconds (10 milliseconds more than the path latency) for a 500 Kbps data stream with 1% and 2% packet loss as the number of hops increases. It is interesting to note that two to four hops appear to be sufficient to capture almost all of the benefit associated with hop-by-hop reliability. This is encouraging as small overlay networks are relatively easy to deploy.

The important factor in obtaining better performance with hop-by-hop reliability is the latency of the lossy link rather than the number of hops in the end-to-end connection. The reason for the phenomenon depicted in Figure 7 is that increasing the number of hops from one to two reduces the latency of the lossy link by approximately 50 percent (25 milliseconds in our case), while increasing the number of hops from nine to ten reduces the latency of the lossy link only by approximately 1 percent (0.55 milliseconds).

It is important how well we can isolate a potentially lossy or congested Internet link in an overlay link that is as short as possible. This can be achieved in practice by placing a few overlay nodes such that we create close to equal latency overlay links, as we do not usually know in advance which Internet connections will be congested.

We believe that the simulation results are promising. The reminder of the paper will investigate whether the same behavior is not limited to our simulation environment but is in fact achieved in practice.

4 The Spines Overlay Network

In this section we introduce Spines, an open source research platform that allows the deployment of an overlay network in the Internet. We use Spines to evaluate the hop-by-hop reliability properties in practice.

Spines instantiates overlay nodes on participating computers and creates virtual links between these nodes. Once a message is sent on a Spines overlay network it will be forwarded on the overlay links until it reaches the destination. Many Spines overlays can coexist in the Internet, and even overlap on some of the nodes or links. Both the source and the destination of a connection should be part of the same Spines overlay network.

Spines runs a software daemon on each of the overlay nodes. The daemon acts both as a router, forwarding packets toward other nodes, and as a server, providing network services to client applications.

Clients use a library to connect to a daemon through an API very similar to the Unix Socket interface. A *spines_socket()* call will return a socket, which is actually a TCP/IP connection to the daemon. The application can use that socket to bind, listen, connect, send and receive, using Spines library calls (e.g. a *spines_bind()* call is the equivalent to the regular *bind()*, etc.). The interface is almost transparent, and virtually any socket-based application can be easily adapted to work with Spines. In addition to the TCP-like interface, the Spines API also provides UDP-like functions for unreliable, best effort communication.

The Spines daemon communicates with clients through a Session layer as seen in Figure 8. There is one session for each client connection, and if the client requests a reliable connection, the daemon will instantiate an end-to-end Reliable Session module that will take care of end-to-end reliability, FIFO ordering, and end-to-end congestion control.

An overlay link consists of three logical components.

- An *Unreliable Data Link* sends and receives data packets with no regard to ordering and reliability. It is used for unreliable, best effort, fast communication as it has no buffering other than the ones provided by the operating system.

- A *Reliable Data Link* provides link reliability through a selective repeat protocol and congestion control, but does not provide FIFO ordering. Packets are buffered before being sent on a *Reliable Data Link* only in case the congestion control or available link capacity limit the outgoing bandwidth to a lower value than the incoming throughput. The explicit congestion notification mentioned in Section 2 is based on the size of these buffers. The link congestion control allows the deployment of Spines in the Internet, providing fairness with external TCP traffic. Figure 9 shows the throughput obtained by an end-to-end TCP stream and by the Spines link protocol for a 10 and a 50 millisecond delay link of 10 Mbps capacity under different levels of losses, and compares it to the analytical TCP model from [13]. The throughput achieved by Spines is very close to that of a TCP connection under similar conditions. Note that for a 10 millisecond link, as the throughput of both TCP and Spines approaches the maximum capacity of 10Mbps, they start developing

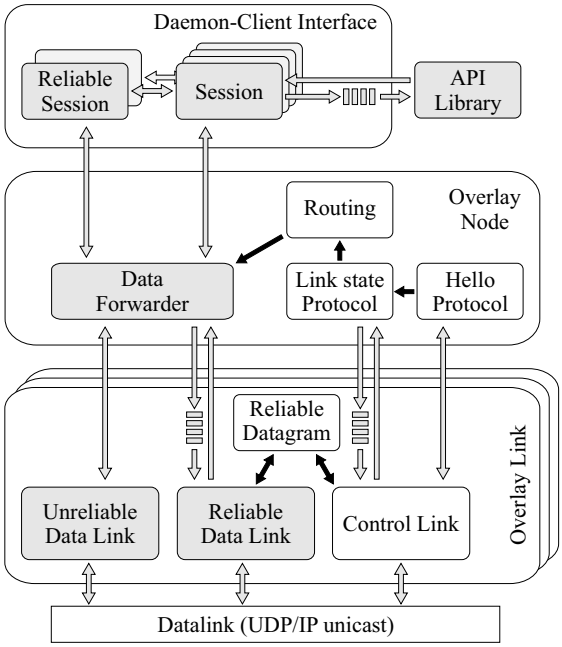

Figure 8. Spines daemon architecture

their own additional losses in order to probe the available bandwidth. This is why they appear to achieve less than the analytical model that takes into account only the original losses we enforced on the link.

- A *Control Link* is used for sending and receiving control information between two neighbor daemons. It provides both reliable and unreliable communication. In case of buffering for the reliable data, the unreliable packets will bypass the buffer and go directly on the network.

The overlay node is responsible for maintaining connections to its neighbors and forwarding data packets either on the overlay links or to its own clients. A *Data Forwarder* parses the header of each message and sends it on the next link or to the daemon-client interface. The *Data Forwarder* allows any combination of reliable and unreliable session and reliable and unreliable link in order to experiment with different forwarding mechanisms. The type of Session and Data Link requested are stamped in the header of each message. For example, one can create a reliable end-to-end session using either unreliable links or reliable links.

Neighboring overlay nodes ping each other periodically using unreliable hello packets. The Spines *Hello Protocol* is responsible for creating, destroying and monitoring overlay links between neighbor daemons. Each Spines daemon sends information about the links to its neighbors through a reliable link state protocol, only when the state of its links change, or periodically at large intervals for garbage collection. The link state protocol provides a complete information about the existing overlay links, out of which a *Routing* module chooses the neighbor providing the shortest path to each destination. The choice of link state routing is purely arbitrary, any other routing protocol could have been used without afecting the hop by hop reliablility mechanisms.

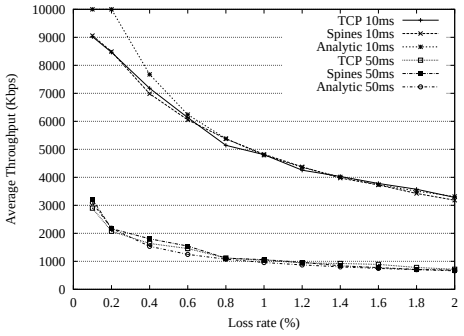

Figure 9. Spines congestion control (Emulab)

In addition to the IP and UDP headers, Spines adds its own headers for routing and reliability. Also, for reliable connections Spines sends acknowledgments for every packet at the level of each link for hop reliability and at least four acknowledgments per end-to-end window for end-to-end reliability and congestion control. When possible acknowledgments are piggybacked with data packets. The control traffic is relatively small, being composed only

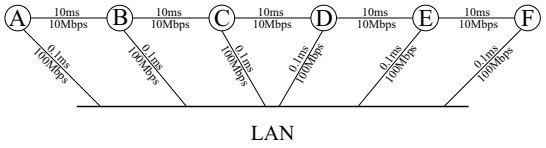

Figure 10. Emulab Network Setup

by hello packets (currently, two 28 byte packets per second on each link) and link state packets that are sent only when the network conditions change. A single link state packet can contain information about up to 90 links, depending on the dispersion of the network. Due to this overhead, our experiments show that when compared with a standard TCP connection running alone on a network link with capacity ranging from 500 Kbps to 100 Mbps, the Spines link protocol achieves about 3.5% less data throughput, and the end-to-end connection that uses both levels of reliability and congestion control (on the hop and end-to-end) shows an overhead of at most 5.7%. The best effort, unreliable protocol in Spines has an overhead of about 2.3%.

5 Experimental results

In this section we evaluate the hop-by-hop reliability behavior using the Spines overlay network deployed on the Emulab testbed. Emulab [1] [5] is a network facility that allows real instantiation in a hardware network (composed of actual computers and network switches) of a given topology, simply by using an ns script in the configuration setup. Link latencies, loss rates and bandwidths are emulated with additional nodes that delay packets or drop them according to specified link characteristics.

We instantiated on Emulab the network setup presented in Figure 10 that follows the topology used in our Section 3 simulations. In addition to the five links A-B, B-C,... E-F we also connected the nodes through a fast, local area network that was used to obtain accurate clock measurements between the overlay nodes.

The routing was set up such that all the experiment traffic went on the 10 millisecond links, while on the local area network we continuously measured (every 100 milliseconds) the clock difference between the computers making the end nodes of a connection. The one-way delay of the data packets was calculated as the difference between the timestamp at the sender and the current time at the receiver, adjusted with the clock difference between the end nodes.

On the overlay network, the round-trip delay between nodes A and F measured with ping under no traffic was 99.96 milliseconds, and the throughput achieved by a TCP connection on each of the 10 millisecond links was about 9.59 Mbps. On the local area network the round-trip delay between any two nodes was about 0.135 milliseconds, which gave us a very good accuracy in measuring the clock difference and one-way delay of the packets. For each experiment in this section we sent 200000 messages of 1000 bytes each.

[1]The Utah Network Emulation Testbed (www.emulab.net) is primarily supported by NSF grant ANI-00-82493 and Cisco Systems

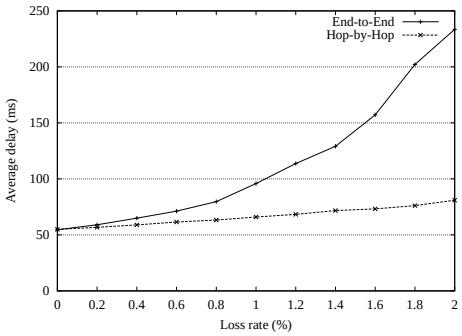

Figure 11. Average delay for a 500 Kbps stream (Emulab)

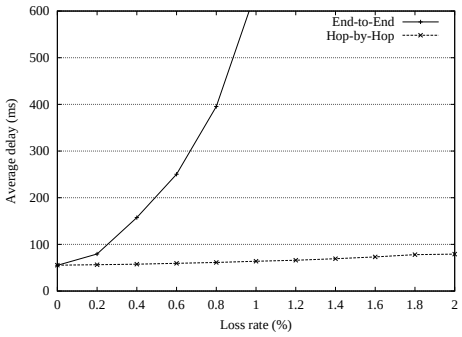

Figure 12. Average delay for a 1000 Kbps stream (Emulab)

We compared the packet delay of a data stream using an end-to-end TCP connection between nodes A and F, with that of a hop-by-hop connection using Spines on the overlay nodes, while varying the sending rate (at node A) and the loss rate on the intermediate link C-D. Note that the end-to-end TCP connection does not go through the Spines application-level routers, but only through the overlay nodes A, B, ... F - so it is not affected in any way by the Spines overhead in user-level processing and added headers.

Figure 11 and Figure 12 show that the low latency effect of hop-by-hop reliability is very signifficant also in the experimental setting, overcoming by far the overhead of user-level processing at the level of the intermediate overlay network nodes. The latency of a real TCP connection is lower than the simulation result (presented in Figure 3

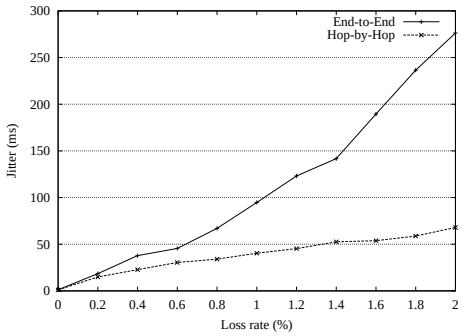

Figure 13. Average jitter for a 500 Kbps stream (Emulab)

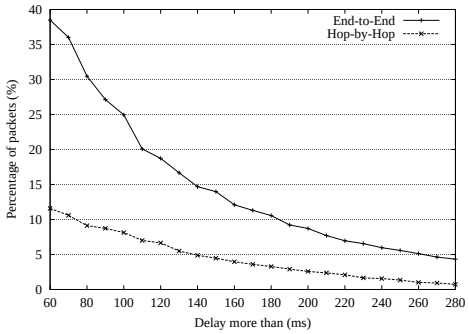

Figure 14. Packet delay distributuin for a 500 Kbps stream (Emulab)

and Figure 4), especially at high loss rates, which shows us that the TCP model we used in the simulation (TCP-Fack), even though the closest, does not resemble exactly the Linux kernel implementation. The latency achieved by Spines hop-by-hop reliability is slightly higher than the latency obtained in the simulator, mainly due to simplifying assumptions of the simulation. However, the hop-by-hop latency remains very low, and increases much slower compared to the latency of the end-to-end TCP connection.

Jitter follows a similar pattern, as seen in Figure 13 (and compared with Figure 5). Packets sent through the Spines overlay network arrive at the destination with a jittter up to three to four times smaller than the jitter of an end-to-end connection. In Figure 14, although the delay distribution for the end-to-end TCP connection is almost identical to the result of the simulation (Figure 6), the overhead of the application-level routing is clearly visible in the hop-by-hop delay distribution. However, even with this overhead, the number of packets delayed by Spines is significantly (more than three times) lower than the number of packets delayed by the end-to-end connection.

6 Related Work

The idea of using reliable intermediate links is not new. In 1976 the International Committee for Telegraph and Telephony (CCITT) recommended X.25 as a store-and-forward connection oriented protocol between end-nodes (DTE) and routers (DCE). In [14], the authors give a detailed description of the X.25 protocol. However, since the Internet was developed as a conectionless, best-effort network (which allows better scalability and interoperability), it did not incorporate the X.25 specifications, but relied on end-to-end protocols such as TCP to provide reliable connections.

One of the early uses of overlay networks in the Internet was in a proposed overlay network called EON (Experimental OSI-based Network) [10] on top the IP network, that would allow experimentation with the OSI network layer. The scheme was only experimental and did not specify hop-by-hop reliability. More recently, overlay networks emerged mainly by providing new services to the application. The Mbone [6] is a routing mechanism that creates an overlay infrastructure over the global Internet and extends the use of IP multicast by creating virtual tunnels between the networks that support native IP multicast. The Mbone facilitates the use of multicast services on the global Internet but does not provide reliability by itself.

TRAM [4] is a tree-based reliable multicast protocol that uses repair trees to localize recoveries, and aggregates end-to-end acknowlegenents at intermediate nodes. TRAM was designed specifically for single-source multicast. If applied to multiple flows (unicast or multicast), TRAM requires intermediate nodes to keep packet-based state for each end-to-end session in order to provide end-to-end reliability and congestion control. Since we use two completely separated levels of reliability (hop-by-hop and end-to-end) our approach allows an unlimited number of reliable sessions, as per flow information is only handled at the end nodes. SRM [8] provides a form of localized recovery for reliable multicast by using randomized timeouts for sending retransmission requests and the retransmissions themselves. SRM does not guarantee recovery from the nearest node, as the closest one may set its timeout to be higher than that of an upstream node. Its probabilistic algorithm allows for double retransmission requests and recovery messages to be sent. The Spread system [1] uses a network of daemons to provide wide area group communication, where missed messages are recovered from the nearest daemon on the path, localizing message recovery in a way similar to ours. The system is confined to group communication and does not provide a generic service such as ours.

Yoid [9] is a set of protocols that allows host-based content distribution using unicast tunnels and, where available, IP multicast. Yoid has the option of using TCP as the link

protocol on the overlay network, but does not guarantee either end-to-end congestion control or end-to-end reliability. In addition to these guarantees, our approach uses an out of order forwarding mechanism that provides less burstiness at the network level, and lower packet latency and jitter.

The X-Bone [17] is a system that uses a graphical user interface for automatic configuration of IP-based overlay networks. RON [2] creates a fully connected graph between several nodes, monitors the connectivity between them, and, in case of Internet route failures, re-directs packets through alternate overlay nodes. Both X-Bone, and RON are implemented at the IP level, do not provide reliability other than the regular end-to-end offered by TCP, and are complementary to our work.

7 Conclusion

This paper presented a hop-by-hop reliability approach that considerably reduces the latency and jitter of reliable connections in overlay networks. We first quantified these effects in simulation.

Overlay networks pay a performance price due to the need to process each message at the application level, and to maintain the overlay. The paper presented experimental results with a new overlay network software we have built. These results resemble the simulation results and show that the overhead associated with overlay network processing does not play an important factor compared with the considerable gain of the approach. We also learned that having a small number of approximately equal hops (two to four) is sufficient to capture most of the performance benefit.

While network bandwidth increases exponentially over time, latency is very slow to improve. This work shows how coupling cheap processing and memory with the programmable platform provided by overlay networks and paying a small price in throughput overhead, can considerably improve the latency characteristics of reliable connections.

Acknowledgment

The authors would like to thank Mike Dahlin for insightful comments and discussions.

This work was partially funded by DARPA grant F30602-00-2-0550 to Johns Hopkins University.

References

[1] Y. Amir, C. Danilov, and J. Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. In *Proceeding of International Conference on Dependable Systems and Networks*, pages 327–336. IEEE Computer Society Press, Los Alamitos, CA, June 2000.

[2] D. G. Andersen, H. Balakrishnan, and M. F. K. R. Morris. Resilient overlay networks. In *Operating Systems Review*, pages 131–145, December 2001.

[3] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partrige, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroklawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the internet. RFC 2309, April 1998.

[4] D. M. Chiu, M. Kadanski, J. Provino, J. Wesley, H.-P. Bischof, and H. Zhu. A congestion control algorithm for tree-based reliable multicast protocols. In *Proceeding of IEEE Infocom*, pages 1209–1217, June 2002.

[5] The Utah network emulation facility. http://www.emulab.net/.

[6] H. Eriksson. Mbone: the multicast backbone. In *Communications of the ACM*, volume 37, pages 54–60, August 1994.

[7] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *ACM Computer Communications Review: Proceedings of SIGCOMM 2000*, volume 30, pages 43–56, Stockholm, Sweden, August 2000.

[8] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, Dec. 1997.

[9] P. Francis. Yoid: Extending the internet multicast architecture. http://www.icir.org/yoid/docs/yoidArch.ps, April 2000.

[10] R. Hagens, N. Hall, and M. Rose. Use of the internet as a subnetwork for experimentation with the osi network layer. RFC 1070, February 1989.

[11] V. Jacobson. Congestion avoidance and control. *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988*, 18, 4:314–329, 1988.

[12] ns2 network simulator. Available at http://www.isi.edu/nsnam/ns/.

[13] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe. Modeling TCP throughput: A simple model and its empirical validation. In *ACM Computer Communications Review: Proceedings of SIGCOMM 1998*, pages 303–314, Vancouver, CA, 1998.

[14] R. Perlman. *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*. Addison- Wesley Professional Computing Series, second edition, 1999.

[15] K. K. Ramakrishnan and S. Floyd. A proposal to add explicit congestion notification (ECN) to IP. RFC 2481, January 1999.

[16] The Spines overlay network. http://www.spines.org/.

[17] J. Touch and S. Hotz. X-bone: a system for automatic network overlay deployment. *Third Global Internet Mini Conference in conjunction with Globecom98*, November 1998.