PRIORITIES IN STREAM TRANSMISSION CONTROL PROTOCOL (SCTP) MULTISTREAMING

by

Gerard J. Heinz II

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Degree in Computer and Information Science.

Spring 2003

PRIORITIES IN STREAM TRANSMISSION CONTROL PROTOCOL (SCTP) MULTISTREAMING

by

Gerard J. Heinz II

Approved: ____
Paul D. Amer, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: ____
Sandra M. Carberry, Ph.D.
Chair of the Department of Computer and Information Science

Approved: ____
Mark W. Huddleston, Ph.D.
Dean of College of Arts and Sciences

Approved: ____
Conrado M. Gempesaw II, Ph.D.
Vice Provost for Academic Programs and Planning

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr Paul D. Amer, for careful guidance and assistance through the thesis writing process and my graduate school experience. Special thanks go to my parents for their support throughout my entire academic career. I would also like to thank Dr. Errol Llyod for his assistance with my algorithmic design and analysis. In addition, I would like to acknowledge the discussions with and advice of the members of the Protocol Engineering Laboratory, especially Armando Caro, Janardhan Iyengar, and Dr. Philip Conrad.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

This thesis introduces per-stream priorities as a method of decreasing delays of important data during periods of low bandwidth availability. We group streams into priority classes and propose a scheduling algorithm to prioritize data among these classes. Per-stream priorities are useful in applications sending different types of data, such as Instant Messaging (IM) systems. We discuss the details of present (AOL Instant Messenger) and future IM systems (using Session Initiation Protocol) and how these systems benefit from prioritized SCTP. We discuss the practical application of example applications of such a scheme are discussed. Through simulation with ns-2, we compare prioritized SCTP to non-prioritized SCTP using an application with two streams (one stream is sending high-priority sporadic data while the other stream is sending low-priority bulk data). In periods when the bulk data submission rate is greater than the available bandwidth and the transmission rate of the sporadic data is less than the link transmission rate, we demonstrate that with per-stream priorities the bulk data transfer will not affect the quality of the sporadic data transfer.

1. INTRODUCTION

With the steady improvement of computer and network systems, end users are growing more accustomed to on-line multimedia experiences. However, these experiences are often bandwidth-intensive and to be comfortable to end users require high throughput connections. While the number of broadband subscribers grows daily, the majority of Internet users still rely on dial-up modem connections, which are often insufficient for comfortable viewing of multimedia data. In addition many pocket devices, such as mobile phones and Personal Digital Assistants (PDAs), now offer web browsing and streaming video over low bandwidth, wireless connections. The maximum throughput achieved by these devices often fluctuates depending on signal strength.

In general, current network end users want to request various types of data from server applications. Therefore, the server applications must provide a way to transmit multiple data-types in parallel, and must effectively respond to periods of insufficient bandwidth.

This thesis purposes an optimization to SCTP, a new general purpose transport protocol, which allows end users to specify the relative importance of data. We introduce data priority based upon logical flows of data from sender to receiver. In Section 2, we discuss the classical approaches to logical data separation and detail the motivations for moving beyond these approaches. In Section 3, we define per-stream priorities as an optional scheduling algorithm for a SCTP sender. In Section 4, we discuss how to add per-stream priorities to SCTP implementations. We present sample applications that can use per-stream priorities in Section 5. Sections 6 and 7 detail our performance evaluation of priority-enhanced SCTP. Section 8 closes with some remarks and suggestions for future work.

2. MOTIVATION

Throughout our discussion, we refer to multimedia applications that must exchange multiple types of data between an application's host and other hosts. For instance, a media server might transmit sound data and video data to another host. In this thesis, we focus on multimedia applications in which each type of data can be ranked in order of relative importance.

Traditionally, transmitting different types of data in parallel between endpoints relied on one of three approaches. In all three situations, Host A would like to send three types of data (labeled Data 1 through Data 3) to Host B. In the first approach (part 1 of Figure 1), Host A opens three TCP connections to Host B – one connection per data type. While this approach provides logical separation of data based on type, multiple connections defeat TCP-friendly congestion control by allowing an application to gain an unfair portion of available bandwidth at the expense of other data flows in the network.

In the second approach (part 2 of Figure 1), Host A multiplexes and demultiplexes the three types of data over a single connection. Applications using this approach maintain TCP-friendly congestion control; however, this approach increases complexity for the application programmer, since the application itself must handle the complicated task of efficiently and fairly managing data transmission scheduling.

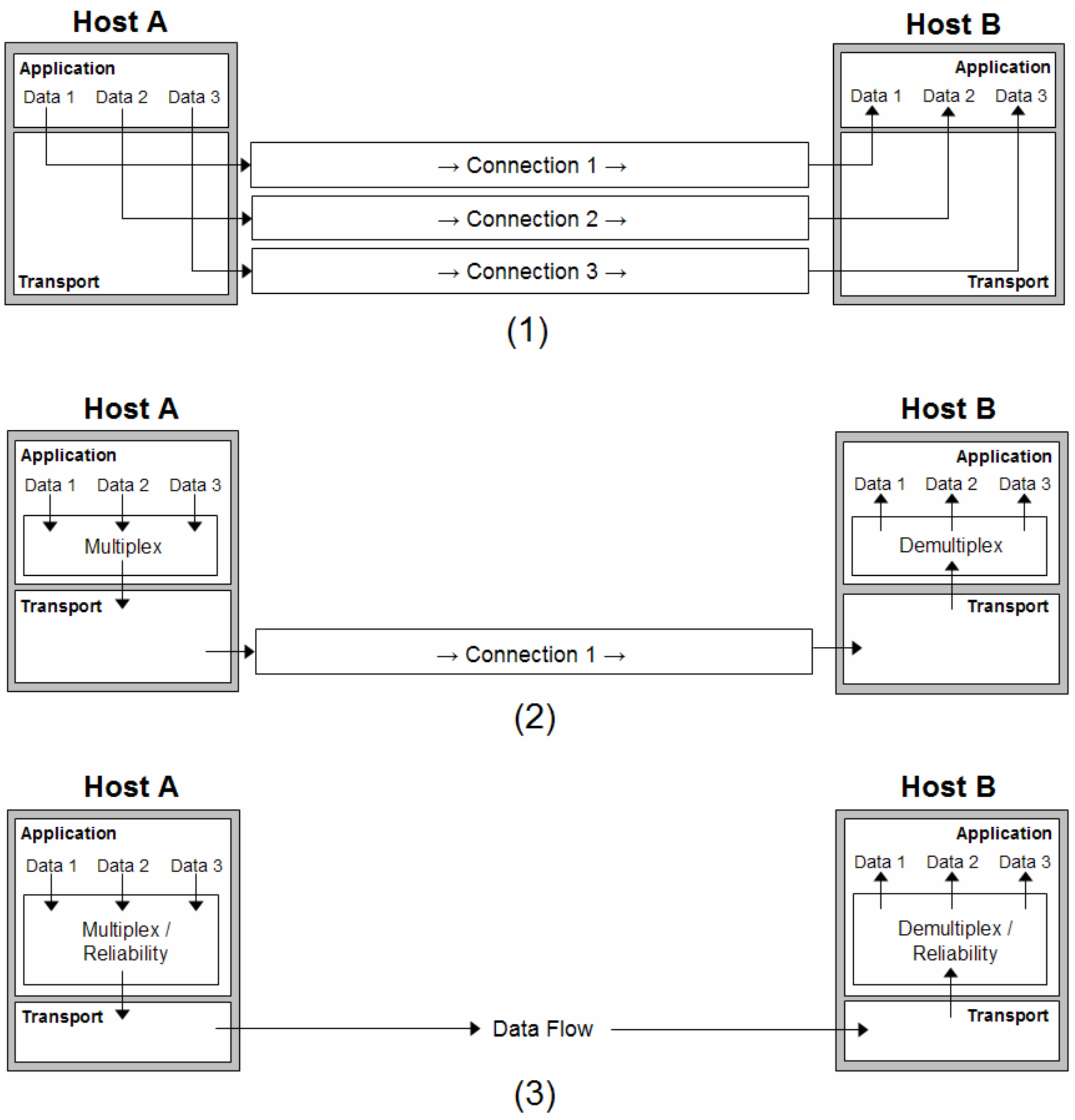


Figure 1 - Traditional Approaches to Parallel Data Transmission

The third approach (part 3 of Figure 1) has a multimedia application using UDP. This approach closely resembles the second approach; however application programmers must supply their own reliability service as well as their own multiplexing/demultiplexing due to UDP's unreliable, connectionless service.

With the introduction of the Stream Transmission Control Protocol (SCTP) [1, 2] and SCTP's concept of streams, applications are presented with a new transport layer solution to transmitting multiple types of data. This new approach combines advantages of multiple end-to-end connections and application multiplexing/demultiplexing. An SCTP stream is a logical, unidirectional communication channel that exists within an SCTP end-to-end association. An SCTP endpoint may request multiple outbound streams during association setup. Each stream is given an independent send and receive buffer (at the sender and receiver, respectively). These streams and buffers exist for the duration of the SCTP association.

Figure 2 shows an example case where Hosts A and B have established a multistreamed association. In this example, Host A would like to transmit three different types of data. Therefore, during association setup, Host A requests three streams to Host B (numbered streams 0 to 2). Host B only has one type of data to send to Host A. Therefore Host B requests and maintains one stream to Host A (numbered stream 0). Since Host A has multiple outbound streams, the SCTP implementation on Host A must provide a scheduling algorithm for data transmission across the streams. The SCTP Implementers

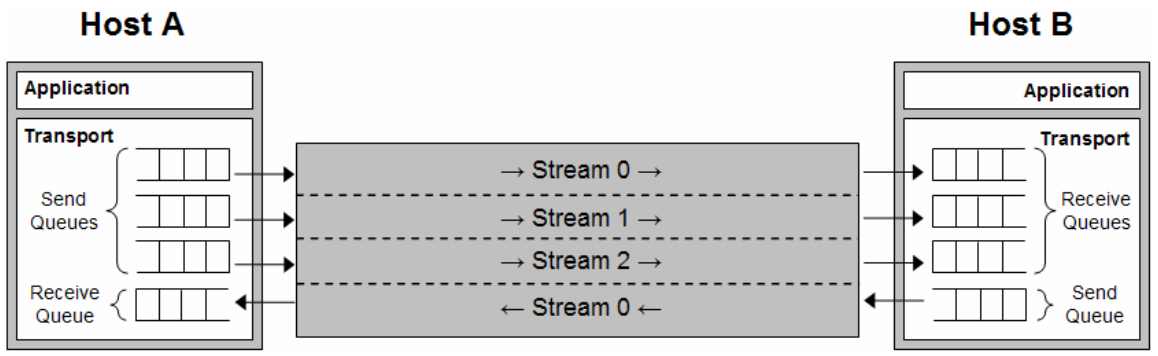


Figure 2 - A Multi-Streamed SCTP Association between Hosts A and B

Guide [3] states that an SCTP implementation must choose a scheduling algorithm that avoids stream starvation. Namely, delivery of data submitted to a certain SCTP stream may not be indefinitely postponed. The guide offers two algorithms: round-robin or first-come first-serve.

Using round-robin for scheduling, Host A would select a data chunk from stream 0's send queue. If Host A could still send data (as allowed by Host B's advertised receiver window and Host A's congestion window), Host A would send a data chunk from stream 1's send queue. A data chunk from A's stream 2's send queue would be sent next. Then, Host A would send a data chunk from stream 0. This round-robin process would continue for the life of the association between A and B.

Using first-come first-serve for scheduling, Host A would transmit each data chunk in the order the chunk was received from the application layer regardless of the chunk's send queue.

While streams were originally introduced by the SCTP authors to prevent head-of-line blocking in SS7-style signaling over IP networks, streams can also provide applications with logical data separation and a central point of connection management. Streams are managed within an association's current receiver and congestion windows. With the introduction of the PR-SCTP (Partial-Reliability) Internet Draft [4] to the IETF, SCTP streams can effectively transport loss-tolerant multimedia data in addition to loss-intolerant data. Thus, SCTP streams provide powerful features to applications without increasing application complexity.

Even with these strengths, networks whose nodes are highly mobile and volatile can prove problematic for an SCTP association using multiple streams. Namely, fluctuating available bandwidth (for example, induced by congestion or signal degradation) can introduce delays in communication between endpoints across all streams due to SCTP's data chunk scheduling algorithm. For example, multimedia network applications running on PDAs and mobile phones could experience a decrease in the quality of service and data rate as they roamed further from a signal tower. The United States Military's Future Combat Systems also experience fluctuating data rates depending on battlefield conditions and mission requirements; and thus may alternate between "high throughput mode" and "low-probability of detection/anti-jamming mode (LPD/AJ)." In either case, these networks require survivable and sustained communications during periods of "high QoS - high bandwidth" as well as during periods of "high loss - low throughput".

3. DEFINITION OF PER-STREAM PRIORITY

To grant multimedia applications an ability to address periods of poor network conditions, we investigate the theoretical and practical implications of adding priorities to SCTP streams. We describe stream priorities as an additional service available to applications. A stream priority scheme allows an application to specify relative importance of data. Given the network conditions at the moment, transmission of critical data should take precedence, thereby reducing perceived delays for critical data during periods of low quality of service. In this example situation, whenever possible, less important data should be transmitted, depending on available bandwidth.

We define an SCTP stream priority scheme as:

Data on stream j always have greater or equal priority
in relation to data on stream k, with $j < k$

The addition of stream priorities is an extension to SCTP's existing *sender-side* API and scheduling algorithm implementation only. Priorities do not change the on-the-wire SCTP protocol and thereby do not change SCTP's current packet format (i.e., no addition of a new control chunk). By avoiding such modifications, stream priorities do not require SCTP's *receiver-side* to be aware that prioritization is occurring at the sender. This transparency maintains backward compatibility with non-priority enhanced endpoints, thereby allowing any SCTP receiver to operate with both priority and non-priority enhanced SCTP senders. In addition, this transparency allows for easier Internet deployment.

Originally, we considered a strict priority scheme for SCTP. In such a scheme, items on stream j always have priority over items on stream k, with j < k. However, a strict priority scheme has an obvious weakness: in cases where an SCTP sender has bandwidth sufficient only to transmit data for streams 0-3, data on stream 4 will be indefinitely postponed. In some applications (such as SS7 signaling and stereo audio streaming applications), multiple data streams are considered of equal importance. A priority scheme must have a method of addressing this situation. By assigning the same priority to two or more streams, our priority scheme will treat those streams' data equally.

4. SPECIFICATION

To implement the priority scheme discussed in Section 3, we must first add a new field to the SCTP stream data-structure. This integer field (called `priority` below) will store a positive value corresponding to the relative priority of the stream. Upon association setup, `priority` must be initialized to the value `0` for all streams. In this manner, unless the values are changed, priority-enhanced SCTP behaves as basic SCTP.

Secondly, the priority scheme implementation requires the addition to the following interfaces to the SCTP Sockets API:

```
sctp_enablepriority ()
{
      for (j = 0; j < num_streams; j++)
      {
            streams[j].priority = j;
      }
}
```

`sctp_enablepriority` sets a default strict priority scheme among an SCTP association's streams. To adjust priorities and to rank streams of equal importance, use:

```
sctp_setequalpriority (int startStream, int endStream)
{

      int p = streams[startStream].priority;
      for (j = startStream; j <= endStream; j++)
      {
            streams[j].priority = p;
      }

      for (j = endStream + 1; j < num_streams; j++)
      {
            p++;
            streams[j].priority = p;
      }
}
```

`sctp_setequalpriority` sets all streams between (and including) streams startStream and endStream to equal priority. Then, the interface resets the priorities on all streams greater than the endStream.

To disable SCTP priorities completely, use:

```
sctp_disablepriority ()
{
      for (j = 0; j < num_streams; j++)
      {
            streams[j].priority = 0;
      }
}
```

Finally, when priorities are enabled, the following algorithm should be used to assign TSN numbers to data among the stream queues:

```
int stream_num;                    // current stream number
int current_priority               // current priority
boolean priority_class_has_data    // does the current priority
                                   //  have any streams that
                                   //  have data to be sent?
int priority_class_base_stream     // the first stream with the
                                   //  current priority

stream_num = 0;
current_priority = 0;
priority_class_has_data = false;
priority_class_base_stream = 0;

while (space exists in SCTP packet)
{
     if (streams[stream_num] has data chunks to transmit)
     {
           assign TSN to a chunk from streams[stream_num]
           bundle chunk into SCTP packet

           priority_class_has_data = true;
     }

     stream_num++;

     if ((priority_class_has_data) &&
          (streams[stream_num].priorities > current_priority))
     {
           stream_num = priority_class_base_stream;
           priority_class_has_data = false;
     }
     else if(streams[stream_num].priorities > current_priority)
     {
           current_priority++;
           priority_class_base_stream = stream_num;
           priority_class_has_data = false;
     }

}
```

The above algorithm should be added after the implementation checks for space in the congestion and receiver windows. After initialization (Lines 1-8), the algorithm performs a round-robin type operation among the streams while space exists in the SCTP packet (Line 9). First, the current stream is checked for data (Line 11). If the stream has data chunks to transmit, a TSN is assigned to the data chunk at the head of the stream's queue (Lines 13-14). This chunk is now ready for transport. Since the stream had data to send, the `priority_class_has_data` flag is set to `true` (Line 15). This flag allows the algorithm to track when the priority level should be increased.

The algorithm then considers the next stream number (Lines 17-28). If this stream is of the same priority observed before, then we loop to process the new stream's data. If this stream is of a lower priority, we check the `priority_class_has_data` flag. If the flag is set to `true`, then the algorithm resets the stream number to the first stream number with the current priority (Line 26). This ensures that all data of greater priority takes precedence over all lower priority data.

Upon initial observation of the algorithm, we might conclude that a number of factors influence the running time: namely, size of the SCTP packet, number of streams, amount of data to transmit, and number of priority classes. However, when analyzing the worst-case running time, we can limit these factors to only the number of streams and the amount of data to transmit.

We do not consider the packet size since, at any time instance, either:

1. The packet size is too small for all of the available stream data – This condition will restrict the running time, since the algorithm terminates once a packet is full, or
2. The packet size is greater than the amount of available data – In this case, the amount of available data will ultimately determine running time.

In theory, a worst-case scenario occurs when the amount of packet space remaining is infinite (however, in practice the packet space is finite!)

Priority classes do not affect running time. Classes only affect the order in which packets are transmitted during the algorithm's execution, and do not increase the total number of loop iterations.

Therefore, we consider the worst-case scenario when at most d chunks of data must be transmitted over n streams. We set the SCTP packet size to infinity to maximize the running time. Then, the running time (T) for our algorithm is:

$$T = n + d$$

The while loop will execute at least once for each stream in the association (to determine if there is data to send on that stream – Line 11) and once for each piece of data.

Therefore, we conclude that this algorithm is order $O(n + d)$.

5. APPLICABILITY

Several applications may benefit from a per-stream priority scheme. Initially, we focus on one widely used (and increasingly popular) general-purpose application – namely instant messaging.

5.1 Overview of Instant Messaging User Activities

Since the late 1990s, the number of Instant Messaging users has increased exponentially. Corporations are turning to instant messaging solutions to cut costs of telephony systems, as these solutions allow users to communicate via short text messages over the Internet.

Fundamentally, instant messaging client software allows a user connect to an instant messaging service and communicate with other users, who are also connected to the same service. Basic communication is performed through the exchange of simple, short text messages between service users. Text message conversations closely resemble person-to-person conversations in real-life. Typically, a user types a message to another user, and then awaits a reply. The reply may come quickly (if the receiver is expecting an instant message and replies immediately), may be delayed (the receiver is away or is busy), or may never come (the receiver is ignoring the sender). In addition, the sender may be impatient and may transmit multiple short messages in quick succession. Generally, we characterize an instant messaging sender's text-data traffic as sporadic and bursty.
Text messages are usually transported through the instant messaging service's network. Namely, when a user Alice wants to send a message to another user, Bob, Alice relays

the message through a server. The server then forwards the message to Bob. When Bob responds, the message first goes to the server; and then the server forwards the response to Alice.

Current instant messaging clients also support communication of types of data other than email-like text – including image, voice and video, and the exchange of files between two users. Relaying bulk data from multiple users may quickly overload servers. To avoid this overload, instant messaging services often negotiate a direct connection between two users. The following sections describe how present and future instant messaging systems implement these advanced features.

5.2 Current Instant Messaging Systems: An AOL Instant Messenger Case Study

Of the current instant messaging systems, AOL Instant Messenger (AIM) [5] maintains a commanding market lead. The AIM service boasts over 180 million registered users, giving AOL nearly 90% of the instant messaging market.

The AIM service runs over a proprietary protocol developed by AOL, known as OSCAR [9, 10]. OSCAR is an application layer protocol that primarily uses TCP for transport[1].

[1] AOL's ad service uses HTTP over UDP to transport small advertisement images that are displayed above the user's buddy list. Otherwise, OSCAR services use TCP for transport.

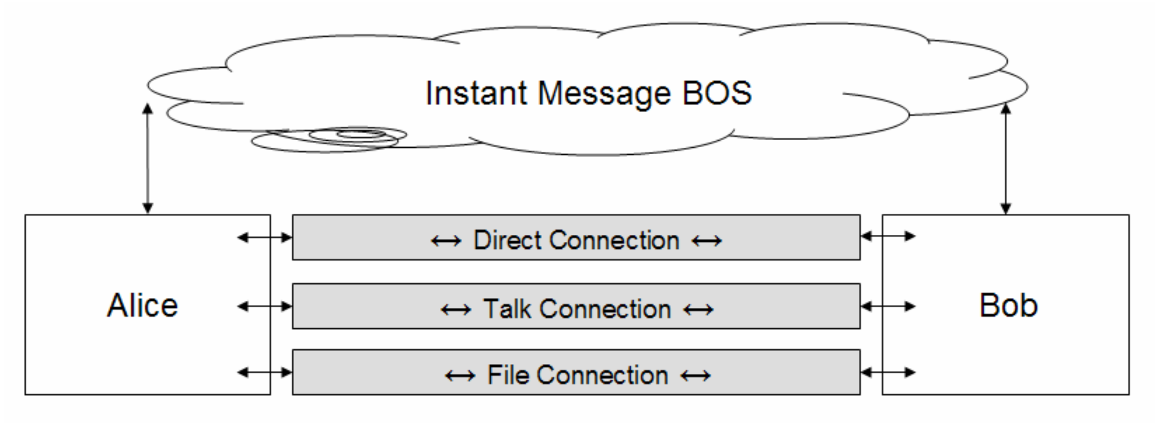


Figure 3 - Alice and Bob connected to the AIM Basic OSCAR Service

When a user, Alice, first logs on to the AIM service, she contacts an Authorizer, a server that manages authentication requests. The Authorizer then sends Alice a *cookie* that allows Alice access to any Basic OSCAR Service (BOS), namely Login/Logoff, Locate, Instant Message, and Buddy List services. Using this cookie, Alice may also log into special-purpose servers, including advertisement, stock ticker, and chat servers.

After receiving a cookie, Alice establishes a TCP connection to the BOS handling instant messages. Through this connection, Alice can send short text-only messages to any other user logged into the AIM service. In Figure 3, Alice can send text messages through the Instant Message BOS to Bob. The details of this procedure are proprietary to AOL.

Through the AIM service, Alice can negotiate TCP connections between herself and Bob to allow them to share additional types of data. These connections may be maintained simultaneously. At present, an AIM user may have at most three direct TCP connections to another user: one for images (called the *direct connection*), one for voice (called the *talk connection*), and one for file transfer (called the *file connection*). Figure 3 depicts these connections between Alice and Bob. The *direct connection* allows Alice to insert images into her conversation with Bob. These images will be displayed inside Alice's chat window on Bob's computer. While a direct connection is maintained, all text messages between Alice and Bob also go through the direct TCP connection, and not through the Instant Message BOS. Messages are multiplexed with the image data during a direct connection.

The *talk connection* allows Alice and Bob to communicate via voice using their systems' microphones and speakers. Finally, the *file connection* allows Alice to send/receive files to/from Bob. While maintaining any or all of these three connections, Alice and Bob can continue their text-messaging conversations.

Consider a scenario where Alice and Bob share a direct connection and a file connection. Before transferring a file, text messages sent by Alice will reach Bob with minimal latency. However, Alice will notice a significant increase in text-message latency if she begins sending a file to Bob in parallel (sometimes as great as 5-10 seconds).

To maintain a comfortable mode of text communication between these two hosts while Alice transmits a file to Bob, the network needs to provide low end-to-end latency for the text. Thus the text data should ideally receive priority over the file data. In our example, we assume the file transfer is secondary to the ongoing text conversation, and therefore could be sacrificed, so long as the file data is reliably delivered at some point during the conversation.

5.3 Future Instant Messaging Systems: IM over the Session Initiation Protocol Case Study

AIM's architecture is limited in that the application requires multiple TCP connections to directly connect Alice and Bob. As explained in Section 2, multiple TCP connections have various disadvantages. The next generation instant messaging protocol, SIP with SIMPLE[2] [6, 7], allows such direct connections while maintaining transport layer independence. Due to SIP's wide support by software industry giants and the 3rd Generation Partnership Project (3GPP), SIP is slated to become a future session protocol of choice for instant messaging software.

The SIMPLE extension [7] allows users to exchange text messages without any relationship (or state) between messages. Text messages traverse a SIP based network in a method similar to that of the general model discussed in Section 5.1. However, if two users want to exchange video or voice data, the full functionality of SIP is required.

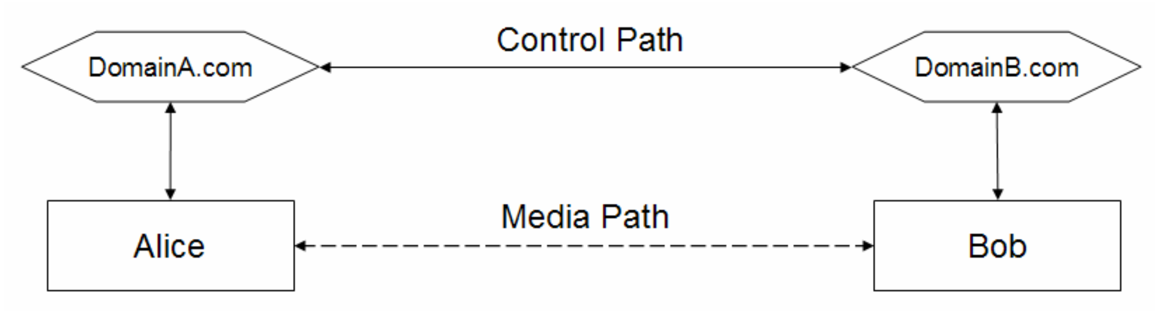


Figure 4 - A SIP call between Alice and Bob

[2] SIP – Session Initiation Protocol; SIMPLE – SIP for Instant Messaging and Presence Leveraging Extensions

SIP provides primitives used for the negotiation of multimedia ‘sessions’ between two hosts, called *calls*. Primitives support the setup and teardown of these calls between two or more users reachable on an IP network. Throughout the network, SIP Registrar Servers track user’s reach-ability for a given domain. In Figure 4, Alice belongs to DomainA.com. When she logs onto the SIP network, she exchanges information with the Registrar Server for her domain. The Registrar Server, in turn, makes note that Alice is reachable, or *present*.

If Alice then wants to call Bob, she sends Bob an INVITE message. Since Bob is not in the same domain as Alice (he is in DomainB.com), Alice forwards the INVITE to her domain’s SIP proxy server[3] (shown in Figure 4 as DomainA.com). Alice’s proxy server then performs a DNS lookup to determine the location of DomainB.com’s SIP Registrar server. When the lookup is complete, Alice’s SIP proxy forwards the request to DomainB.com’s Registrar server; which in turn forwards the INVITE to Bob. If Bob accepts the INVITE, then SIP is used to negotiate call options.

Using header information, all control data for the call will flow over the same application-layer path as the first request – that is, control information will flow from Alice to Alice’s SIP proxy to Bob’s SIP Registrar to Bob. This path is called the *control*

[3] In this scenario, the server DomainA.com acts as both a SIP proxy and a SIP registrar (analogously for server DomainB.com).

path (as denoted by the solid line). The state of the call is maintained though this control path until Alice or Bob decide to terminate the session.

During the call negotiation, a *media path* is also setup whereby Alice can transmit data directly to Bob along a direct connection (as denoted by the dotted line). In this case, media can represent any data type – including text, audio, video, or files.

SIP provides transport layer independence. That is, SIP control and media paths currently operate over either UDP or TCP. Due to SCTP's inherent benefits to signaling, SCTP has been suggested as another possible transport layer to SIP [11]. While one SCTP association could be maintained between Alice and the DomainA.com SIP Server, another association could be maintained between Alice and Bob (thereby creating the media path).

Let us return to our instant messaging scenario between Alice and Bob. Alice would like to send Bob a file while maintaining a comfortable text conversation with him. Using SIP over SCTP, the priority scheme proposed in Section 3 addresses this situation. Let the chat data go over stream 0, and the image data go over stream 1. During an SCTP association, chat data would be sent immediately while the bulk data would be delayed. If there remains room in the receiver and the congestion windows after all stream 0 data has been sent, the stream 1's bulk data would be sent. During periods when stream 0 has no data to send, stream 1 would transmit as much data as allowed by the available window space.

6. SIMULATION

To effectively investigate the theoretical and practical implications of adding stream priorities to SCTP, we add the strict priority scheme into the University of Delaware's SCTP module [12] for ns-2. This module is in wide use and provides a reliable baseline for SCTP functionality.

Through simulation we demonstrate that, based on our performance criteria, priority-enhanced SCTP outperforms basic SCTP in the instant messaging application discussed in the Section 5. In our simulation, we rate performance by measuring the end-to-end latency experienced by the text messaging data on stream 0. We demonstrate that the chat performance is not sacrificed in the presence of a file transfer. SCTP with priorities performs as well as basic SCTP in the worst conditions. Otherwise, chat data on stream 0 of priority-enhanced SCTP is delivered to the receiver sooner after submission than the chat data of basic SCTP.

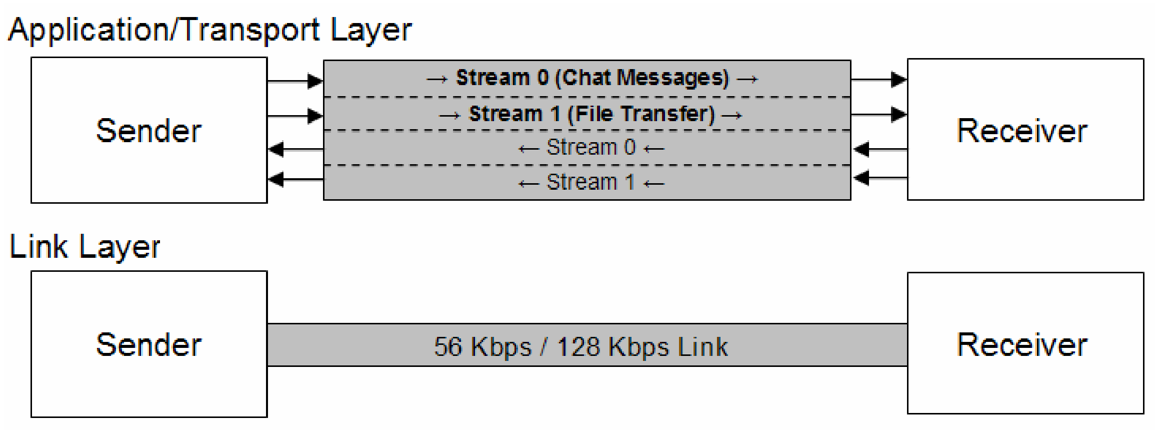


Figure 5 - Simulation Network

For our simulation experiment, we create a link between two hosts, a sender and the receiver (see Figure 5). We establish an SCTP association with two streams in either direction. The sender's stream 0 carries small packets (30 bytes of application-layer data) representative of text messages used during an instant messaging conversation. To simulate a conversation, this data is generated at 30-second intervals. The sender's stream 1 carries the bulk data for a continuous file transfer, representative of a large image file transfer.

Using this association, we compare "basic SCTP" to "priority SCTP". To illustrate the performance gain by priority SCTP, we test both versions over a simulated 56Kbps (Dial-up modem) link and a 128Kbps (ISDN) link. This link, for both speeds, has a propagation delay of 250ms.

During simulation we expect that, with priorities, the quality of the chat data on stream 0 should not be affected by the bulk data on stream 1 during certain conditions. Specifically, let:

R_0 = Rate of data submitted by the application for transmission over stream 0.

R_1 = Rate of data submitted by the application for transmission over stream 1.

$R_{Available}$ = Rate at which SCTP can send data to the receiver.

For these values, we assume application-layer data rates.

There are two conditions when priorities will have minimal, if any, effect on data transmission. Condition (1) exists where $R_0 + R_1 < R_{Available}$. During this time, there will be no transport layer queuing, and data would be transmitted immediately after receipt from the application. Condition (2) exists where $R_0 > R_{Available}$. During this period, queuing would occur on stream 0, introducing delays regardless of the amount of data on stream 1. Under this condition, a priority scheme could not prevent delays on stream 0. However, the delays experienced on stream 0 should still be less than the delays on an unprioritized stream 0.

We hypothesize that:

With per-stream priorities, the quality of sporadic data on stream 0 will not be affected by the bulk data on stream 1 when $R_0 < R_{Available} < R_0 + R_1$.

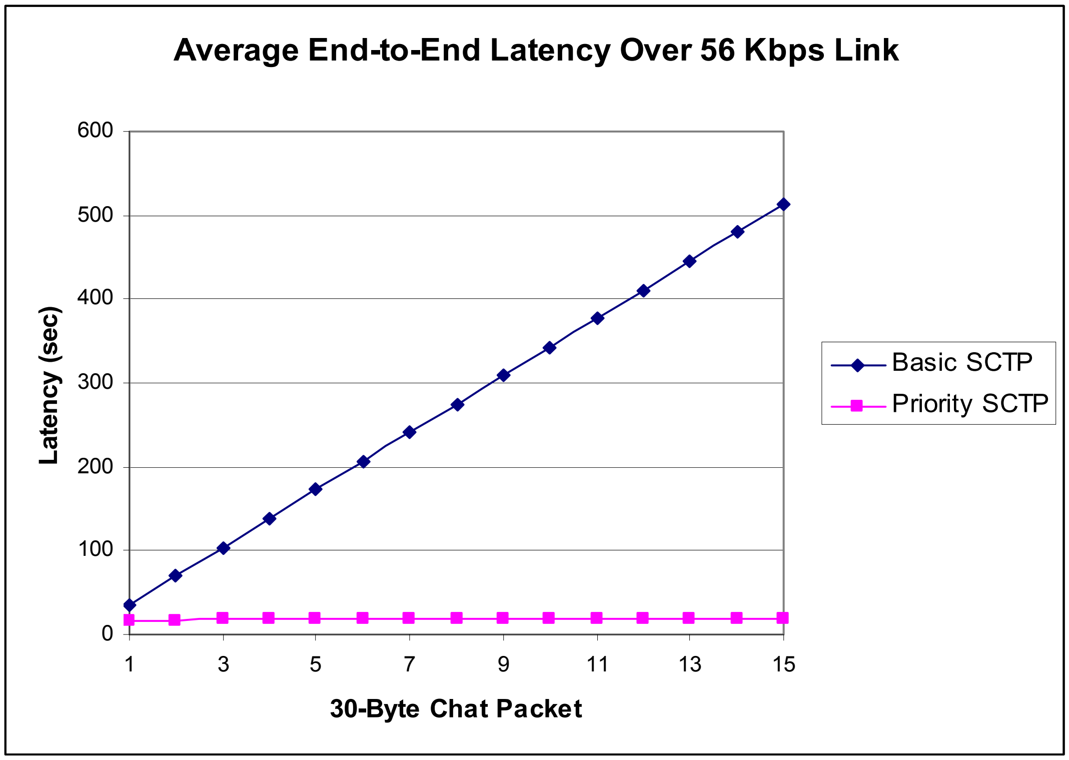
Under these conditions, the data on stream 0 would be transmitted in favor of data on stream 1.

Throughout the simulation, we analyze the 'quality of sporadic data' by measuring the end-to-end latency of data on stream 0. We define end-to-end latency as the time elapsed from when the sending application gives the text data to the SCTP sender until the SCTP receiver hands-off the text data to the receiving application.

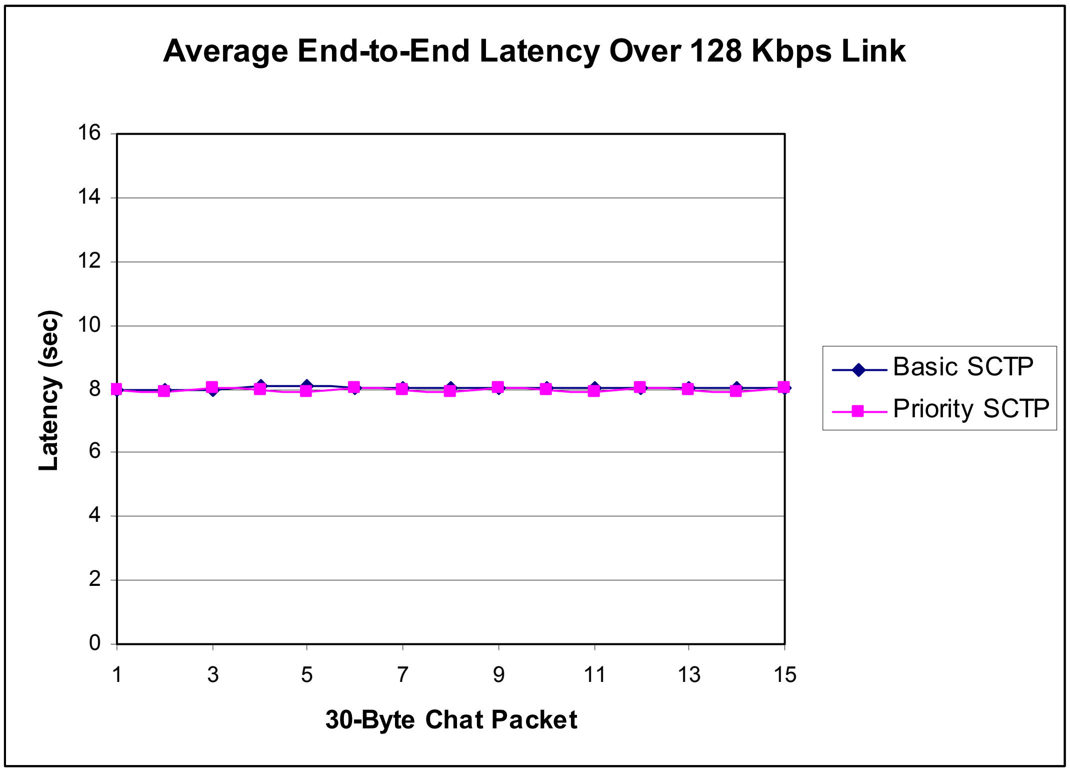
7. DISCUSSION OF RESULTS

We performed the above simulation 100 times, using a different random seed for each run. We discarded the initial 5 packets and selected the next 15 sequential chat packets from the simulation. Then, we analyzed the end-to-end latency for these packets.

In (1), we show the average end-to-end latency for the 30-byte chat packets on stream 0 across a 56Kbps line. This simulation demonstrates our hypothesis scenario, namely $R_0 < R_{Available} < R_0 + R_1$. In basic SCTP, the first-come first-serve nature of basic SCTP increases the end-to-end latency of the data on stream 0. As we hypothesized, this low bandwidth link produces an unacceptable latency for stream 0 data. Since the data on stream 1 will be queued at a constant rate, the sender's transmit queue will be filled with data from stream 1. All of this data must be transmitted before new data on stream 0 is sent. Therefore, the latency on stream 0 increases over time as more data for stream 1 is queued for transport at the sender.



(1)



(2)

Figure 6 – Latency for Chat Packets

While using priority SCTP over a 56Kbps link, the latency on stream 0 remains fairly constant, despite the lower bandwidth. The chat packets report an average delay of 8 seconds. We know that the data on stream 0 is transmitted upon receipt by the sender's transport layer from the sending application.

In (2), we demonstrate Condition 1 above: namely, $R_0 + R_1 < R_{Available}$. From the graph, we observe that at higher bandwidths (128 Kbps), priority SCTP behaves similar to basic SCTP. The link speed for both the priority and basic transmissions is faster than the rate at which the sending application produces packets. A 30-byte chat message does not experience queuing in either priority SCTP or basic SCTP. Therefore, the latency observed, derived from the propagation delay, in the graph is equivalent for both protocols. This condition demonstrates the stability of priority-SCTP.

We do not demonstrate Condition 2 above: namely, $R_0 > R_{Available}$. While maintaining a realistic chat traffic simulation, Condition 2 is unlikely in any practical scenario. To observe Condition 2 in our simulation, $R_{Available}$ must reduce to less than 1 byte per second.

8. CONCLUSIONS AND FUTURE WORK

Initially, we have summarized classical approaches to logical data separation using TCP and UDP, and detailed the motivations for moving beyond these approaches. SCTP addresses the issue of logical data separation by introducing streams. To enhance SCTP for multimedia applications (such as instant messaging), we defined a per-stream priority scheme as an optional scheduling algorithm. We have discussed how to add per-stream priorities to existing SCTP senders without changing the on-the-wire protocol. Finally, we have shown via simulation that priority SCTP reduces end-to-end latency in certain scenarios.

The strict priority scheme, presented in this paper, only addresses reliable data. Future work should investigate using priorities with PR-SCTP [4], the partially reliable data transfer option for SCTP. PR-SCTP, as mentioned before, allows streams to have varying levels of reliability. Applications using both PR-SCTP and per-stream priorities could transmit important data while expiring data of lesser importance in times of insufficient bandwidth. Such an investigation would extend per-stream priorities to effectively handle such data as left and right stereo feeds, MPEG video frames, and other time-sensitive material.

9. DISCLAIMER

Prepared through collaborative participation in the Communications and Networks Consortium sponsored by the U. S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

10. REFERENCES

[1] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson. "Stream Control Transmission Protocol," *RFC 2960*, October 2000.

[2] R. Stewart, Q. Xie. *Stream Control Transmission Protocol (SCTP): A Reference Guide*. New York: Addison-Wesley, 2002.

[3] R. Stewart, L. Ong, I. Arias-Rodriguez, K. Poon, P. Conrad, A. Caro, M. Tuexen. "Stream Control Transmission Protocol (SCTP) Implementers Guide," draft-ietf-tsvwg-sctpimpguide-08.txt, Internet-Draft, February 2003 (Work in Progress).

[4] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, P. Conrad. "SCTP Partial Reliability Extension." draft-stewart-tsvwg-prsctp-04.txt, Internet Draft, May 2003 (Work in Progress).

[5] AOL Instant Messenger, www.aim.com.

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

[7] B. Campbell, J. Rosenberg. "Session Initiation Protocol Extension for Instant Messaging," draft-ietf-simple-im-02.txt, Internet-Draft, September 2002 (Work in Progress).

[8] R. Stewart, Q. Xie, L. Yarroll, J. Wood, K. Poon, K. Fujita, M. Tuexen. "Sockets API Extensions for Stream Control Transmission Protocol." draft-ietf-tsvwg-sctpsocket-06.txt, Internet-Draft, February 2003 (Work in Progress).

[9] The UnOfficial AIM/OSCAR Protocol Specification, aimdoc.sourceforge.net/OSCARdoc

[10] Java Developer Pages: Terminology, www.aim.aol.com/javadev/terminology.html

[11] J. Rosenberg, H. Schulzrinne, G. Camarillo. "The Stream Control Transmission Protocol as a Transport for the Session Initiation Protocol." draft-ietf-sip-sctp-03.txt, Internet-Draft, June 2002 (Work in Progress).

[12] A. Caro and J. Iyengar. ns-2 SCTP module. http://pel.cis.udel.edu.