A Selective Retransmission Protocol for Multimedia on the Internet

Mike Piecuch, Ken French, George Oprica, Mark Claypool

Worcester Polytechnic Institute Computer Science Department
100 Institute Rd
Worcester, MA 01609

ABSTRACT

Internet multimedia applications have different requirements than do traditional text-based applications, placing new demands on TCP and UDP, the de-facto Internet transport protocols. We propose a Selective Retransmission Protocol (SRP) to balance the potentially high loss found in UDP with the potentially high latency found in TCP. SRP uses an application-specific decision algorithm to determine whether or not to ask for a retransmission for a lost packet, adjusting the loss and latency to the optimum level for the application. We develop and experimentally evaluate an audioconference using SRP on a wide-area network testbed. We find SRP outperforms both TCP and UDP in terms of multimedia application quality.

Keywords: Multimedia Streaming, Network Protocol, Quality of Service

1. INTRODUCTION

The Internet has been dominated with traffic from text-based applications such as telnet, ftp, and recently by http. With the emergence of new technologies and an increased diversity of Internet use, there has been a new demand for non text-based applications, particularly multimedia, such as video on demand and teleconferencing[1].

Text-based applications have some characteristics and requirements in common. Most, such as telnet, ftp, and http, require guaranteed delivery. Every unit of data must be delivered without loss or error. Otherwise, the result could be corrupted files and invalid commands. As a result, these applications use the Transport Control Protocol (TCP), which provides guaranteed delivery by automatically retransmitting lost or corrupted data packets.

Certain applications, such as TFTP or DNS, may not need a strictly reliable protocol, but rather a simple protocol with minimal delay and overhead. These applications commonly use the User Datagram Protocol (UDP). UDP does not provide any protection against loss; however, it does not have the overhead of retransmission allowing it to provide a fast, "best-effort" delivery.

Multimedia applications have requirements different than text-based applications[6]. An audio stream, for example, requires that data is received in a timely fashion and is more forgiving of lost data[2]. If a data packet arrives at the player too late, it misses the time it needed to be played. This phenomenon, called jitter, causes gaps in the sound heard by the user. In many cases, a late data packet in a multimedia application contributes nothing to the playback and is equivalent to a loss. Small losses in the playback stream can be replaced with substitute data or concealed so that the listener does not notice.

When designing a multimedia application, a protocol must be chosen that provides a solution for timing issues such as jitter as well as loss[7]. TCP is ineffective due to the overhead of retransmission and ignorance of timing factors[3]. While it provides a service with no loss of data, it does not support any time constraints. Data can arrive at a receiver with unbounded delay. UDP, conversely, provides a "best-effort" service that is timely. It does not, however, offer any guarantees on data loss. With UDP, potentially all data sent can be lost.

To provide a balance between the potentially high delay of TCP and the potentially high loss of UDP, we propose the Selective Retransmission Protocol (SRP). SRP retransmits only a percentage of the data that was lost, providing a compromise between TCP, which retransmits all lost data, and UDP, which retransmits no data. The amount that is retransmitted depends on several Quality of Service (QoS) factors including current loss and latency, round-trip time, network congestion and the desired quality requested by the user.

The performance of SRP was evaluated using an isolated test network and a simulated audio stream. A quality comparison of the protocol was performed against TCP and UDP where high quality was defined as low average loss and low average latency. Tests were performed with network conditions of low loss/low latency and high loss/high latency.

The rest of the paper is as follows: Section 2 describes our approach in detail, Section 3 is the experiments that were used to test the performance of SRP over various network conditions, Section 4 shows the results from the experiments, Section 5 concludes the paper and Section 6 describes future enhancements that could be done to the SRP protocol.

2. APPROACH

We designed SRP as an application level protocol that uses UDP to transmit messages. It consists of a server that sends a multimedia stream in a series of datagrams at a constant rate and a client that receives them. A sending application and receiving application drive the SRP server and client via interface functions. The sender application gives data to the SRP server at even intervals for transmission and the receiver application continuously calls the SRP client for more data.

Anytime that the client detects that a frame was lost (a packet does not arrive at the expected time), it makes a decision whether or not to request retransmission of the message. This decision is performed by an algorithm that takes into account how much loss and latency is tolerated by the user and the current measured loss and latency. If the decision is to retransmit, a retransmission request is sent to the server and the client waits for the response. If the response does not arrive when expected, the decision to retransmit is evaluated again. If the decision is to not retransmit, the client gives up on that particular data message and returns a failure to the receiving application. Any messages that are received while waiting for a retransmission are placed in a buffer until needed by the receiver application.

Expected reception times are generated dynamically as messages are received. The time between arriving messages is measured and averaged. Retransmissions are expected to arrive in one round trip time, measured though time probes that are sent to the server and returned periodically.

During the design of SRP, the following assumptions were made:

- The time to send a message from server to client is the same as from client to server. Without synchronized clocks between the server and client, the exact latency cannot be known. Only the server knows when a message is sent and only the client knows when it is received. To estimate latency, one must know how long it takes for a message to get from the server to the client. This is approximately half of the round trip time if the time for a message to travel from client to server is the same as server to client.
- The client application continuously reads and has a small turn around time. Note, if the client spent a long time processing data, the SRP client will fall behind.
- The server application sends messages at a constant rate. The server does not have any flow control or knowledge of the client. This allows the client to estimate an expected arrival time.
- The application controlling SRP creates the UDP socket and connects with the server application before initiating SRP. The application is also responsible for tearing down the socket once the transfer is complete.
- During our comparison testing, the network conditions during each test are constant. Even though there may be slight variations in network traffic for a particular test, each protocol has to deal with the same type of induced traffic.

The SRP server sends a multimedia stream to the client. A thin server/thick client design leaves the client responsible for most of the computation. Upon initialization, a thread is spawned for receiving. The main thread returns to the sending application while the new thread waits for socket activity. At even intervals, the sending application requests data to be sent. The SRP server constructs a SRP data message controlling a data sequence number and sends it to the client via UDP. This data is also stored in a buffer where it can be retrieved later if needed for a retransmission. Simultaneously, the other thread waits for a reception on a UDP socket. If a request for a retransmission is received, the data is retrieved from the buffer and resent. If the data is not present, the retransmission request is ignored. If the thread receives a time probe, it is immediately sent back to the sender.

The SRP client is responsible for receiving a multimedia stream from the server and requesting retransmissions if necessary. After initialization, the receiving application requests data from the SRP client. First, the client's buffer is checked for the data with the appropriate sequence number. If present, it is returned to the calling application. Otherwise, the UDP socket is

checked for incoming messages. If a message is received, the sequence number is checked against what is expected. If the sequence number is higher than needed (too early), then the message is stored in the client's buffer and the socket is checked for more messages. If the incoming message has the expected sequence number then it is returned to the sender application. If the incoming message has a sequence number that is less than needed (too late), it is ignored and the socket is checked for more messages. If the correct data is not received before retransmission timeout (RTO) then a decision algorithm is consulted on whether or not to retransmit the needed message. There are currently two decision algorithms implemented: equal loss latency (ell) and optimum quality (oq). If the algorithm decides to retransmit, then a retransmission is sent and the receiving process is repeated, otherwise an error message is returned to the application.

2.1 Messages

There are three types of messages that can be received by the client from the UDP socket: normal data messages, retransmission replies, and time probes. Normal data messages are packets that arrive at the client without being retransmitted. Retransmission replies are packets that are received by the client as a result of a retransmission request. Time probes are messages that have no multimedia data but contain timing information.

Every message sent by the server is assigned a data id. This id is a sequence number that begins at one and is incremented for each message that is sent. Retransmissions do not increment the data id but rather use the same data id that was included when it was sent originally. The client maintains an expected data id that is incremented each time a message is returned to the receiving application or when the client gives up on that message (discontinues retransmissions). Maintaining the expected data id allows the client to know when a incoming message is too early or too late and also guarantees that the receiving application will get messages in the correct order. When the data id reaches a constant maximum, it is reset to zero. To prevent incorrect ordering the constant maximum needs to be larger than the amount of frames that the client can be ahead or behind and it must be clear whether an incoming message is greater or less than the expected data id even if it has been reset.

Under optimum network conditions, the client will always receive messages at the same interval, since the server sends them at even intervals. The client can measure the interval and deduce the time that the next message should arrive. If the message does not arrive at the expected time then a decision can be made whether or not to request a retransmission.

Since network conditions are rarely optimum, this interval can vary greatly. To take this into account, SRP uses a smoothing algorithm based on TCP's smoothed RTT[4]. Every time an interval is measured it is averaged into the current smoothed average. Deciding when to request a retransmission for a lost message is important. The time interval to wait before retransmitting is called the retransmission timeout (RTO). If the RTO is to short, then an unnecessary retransmission may be sent for a message that was just a little late. If the RTO is too long then a large amount of latency will be generated waiting for a message that is lost and will never arrive. To calculate when a message has likely been lost, the expected time and the network variance must be taken into account. SRP uses a formula based fromthe RTO calculation from TCP[4].

For a retransmission, the expected time is the Round Trip Time (RTT) since the server will immediately return a retransmission request if the data is in the server buffer. Maintaining an accurate round trip time is therefore important for the client to be able to compute latency and the expected arrival time of retransmissions. In order to measure the current RTT of the network, small time probes are sent to the server. Each time probe contains the current time. When a time probe is received at the server, the server sends the time probe back. When the reply time probe is received, RTT is calculated by subtracting the time stored in the time probe with the current time. The current RTT is used in a smoothed average and deviation calculation. To reduce traffic from time probes, timing data is added to retransmission requests and replies so that RTT can be calculated from the retransmission system as well as time probes. Time probes are then only sent if a new RTT has not been calculated for a long period of time (one second in our implementation).

Normal data messages arrive at even intervals since retransmission replies and time probes can be sent by the server anytime. In order to maintain an accurate expected interval, only normal data messages can be used in the average. The exact time to timeout for a retransmission is the time that the last normal data message was received plus the RTO (which is essentially the smoothed expected interval). Since messages can be lost, some of the normal data messages might not arrive. To maintain the expected arrival time even through loss of normal data messages, last normal reception time (LNRT) stores the time the last normal data message arrived. An LNRT multiplier stores how many normal data messages have been lost since LNRT was updated. With this information, the estimated time of arrival of the next normal data message can be calculated even under losses as follows: Expected_Arrival_Time = LNRT_multiplier * expected_time_interval + LNRT

2.2 Message Events

If a message arrives at the client that has a higher data id than what it is expecting (too new), it is placed into the client buffer for retrieval later. The arrival time and round trip data are stored in addition to the data. When the receiving application requests data from the SRP client, the client buffer is first checked to see if the data has already arrived. If the message is in the buffer, LRNT and latency are updated using the arrival time, round trip time information stored in the buffer and LNRT multiplier is reset. In order to prevent the buffer from overflowing, messages that arrive that are too early (based on buffer size) are not buffered. When a message is retrieved, it is removed from the buffer ensuring that the buffer will not contain old messages. The expected data id is incremented and the data is returned to the receiving application.

If a message is not found in the client buffer, a socket read is performed. If the message received has a data id that is greater than expected, it is considered too early. Messages that are received too early are placed in the client buffer for use when needed. The arrival time and RTT are recorded in the buffer for latency and LNRT calculations. Since receiving out of order messages is very rare in practice, receiving a message that is too early while waiting for a normal reception is a good sign that the a message was lost and should trigger a retransmission decision.

If a message received on the socket has a data id that is less than the expected data id then the message is considered too late. Late messages are most often highly delayed retransmissions that were wrongly considered lost. Since the message was considered lost and the receiving application was notified that it was lost, the old message cannot be used and is dropped.

If a message received on the socket has the same data id that is expected it is considered a good message. If it was received normally (not a retransmission) then LNRT is updated with its arrival time. If the previous message was received normally as well, then the expected interval is updated by finding the difference between the arrival times. Otherwise, if the message was a retransmission then the LNRT multiplier must be incremented since a message was received and LNRT was not updated. Finally, the expected data id is incremented and the data is returned to the receiving application.

If a message is not received within retransmission timeout (RTO) then a timeout occurs. When a timeout occurs, the decision function is called to determine whether or not to retransmit for the expected data id. RTO is changed depending on whether the client is waiting for a retransmission or a normal reception.

When it is determined that the expected message has been lost, the decision function can request a retransmission. The retransmission request message contains the data id of the lost message to be retransmitted. Upon sending the retransmission request, the current time is recorded for easy calculation of when the reply should arrive.

When it is determined that the expected message has been lost and the decision function chooses not to request a retransmission, there is a give up event. During a give up event, LNRT multiplier is incremented since LNRT has not been updated and the expected data id is incremented for the next message. An error code is returned to the receiving application denoting that the message was lost. The receiving application then has the opportunity to use error concealment or correction to the multimedia data.

2.3 Decision Algorithms

A decision algorithm decides whether or not to request a retransmission for a message that was detected as lost. This algorithm, in effect, controls the balance between the overall loss and overall latency of the multimedia stream as seen by the application. Our implementation explored two ideas for this algorithm, equal loss latency and optimum quality.

2.3.1 Equal Loss Latency (ELL)

The equal loss latency (ELL) decision algorithm decides whether or not to retransmit by estimating both the resulting loss and latency if the expected message is retransmitted and if a loss is accepted. Whichever result better equalizes loss and latency is chosen. In order to determine the relative value between loss and latency, the receiving application supplies a maximum tolerable loss and latency to the client at initialization. The current loss ratio is the ratio of the current loss to the maximum loss, the latency ratio is calculated in the same way. Clearly, smaller loss ratios are desirable, a loss ratio of zero is perfect and a loss ratio of one is the maximum tolerable.

Loss and latency ratios are calculated for estimates of when to retransmit and when not to retransmit. To estimate the time for a retransmission, round trip time is averaged into the current latency before calculating the ratio. If not retransmitting, one loss is averaged into the current loss before calculating the ratio. The difference is then calculated for each case as follows: Diff = abs(LossRatio – LatRatio) where Diff is the difference between loss and latency, LossRatio is the loss ratio with estimated loss averaged if needed and LatRatio is the latency ratio with estimated latency averaged if needed The lower of the two differences is then the choice of whether or not to retransmit.

2.3.2 Optimum Quality (OQ)

The optimum quality (oq) decision algorithm decides whether or not to retransmit by estimating both the resulting loss and latency if the expected message is retransmitted and if a loss is accepted. Whichever result has a loss and latency closest to zero is chosen. The loss and latency ratios are calculated exactly the same way as the equal loss latency algorithm. The decision to retransmit is determined by the Euclidean distance calculated by using the loss and latency ratio coordinates from the origin
$SqrDist = LossRatio^2 + LatRatio^2$, where SqrDist is the square distance from the origin, LossRatio is the loss ratio with estimated loss averaged if needed and LatRatio is the latency ratio with estimated latency averaged if needed

Whichever distance is closer to zero is chosen as the decision to retransmit or not. Note that square distance is computed rather than actual distance. Either could be used as a comparison, but the square root needed to find the real distance is computationally expensive and not necessary.

2.4 Applications using SRP

When designing a server application to interface with the SRP server, there are several issues to consider. The server is expected to create a UDP socket and bind to a port. The UDP socket number and any UDP parameters, including the SRP client's address, must be specified to SRP upon initialization. The sending application is also expected to do any handshaking necessary to begin a multimedia session. The SRP server should be called to send multimedia data at a constant interval. The SRP protocol is dependent on a constant data stream, but SRP has the ability to dynamically estimate the data rate if it changes slowly.

The receiving application is expected to create a valid UDP socket and provide any handshaking necessary to confirm communications with the sending application. The SRP client, upon initialization, must be given the active UDP socket number, the address of the SRP server and any UDP parameters that are desired. In addition, the receiving application must give a non-zero desired loss and latency. The receiving application has certain timing constraints. The application must continuously poll the SRP client for more data. The turnaround time between polls should be minimal so that arrival times of messages remain accurate.

3. EXPERIMENTS

After developing the SRP protocol, we compare its performance against that of Internet protocols TCP and UDP. Testing comprised of transferring a simulated audio stream over a test network with competing traffic. Average loss and latency was gathered for each protocol under several different network conditions and compared. Quality was determined as the least amount of average loss and average latency, normalized based on user-defined maximum acceptable limits.

3.1 Test Network Topology

Figure 1 describes the network used to test the SRP, TCP, and UDP protocols. The network consists of a sender whose purpose is to simulate a multimedia server, a receiver that simulates an end user client, and a router to simulate an Internet router. In addition, two traffic generators are used to flood the test network and simulate congestion from other network flows

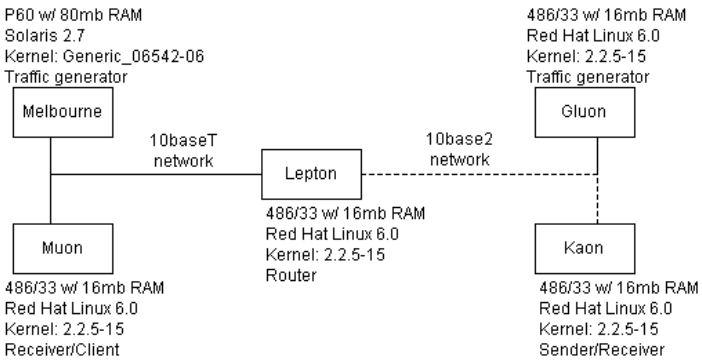

Figure 1. Network Topology

The SRP protocol was tested in comparison to TCP and UDP on the above test network. The network is designed to provide a sample client and server audio session across a router with competing traffic. In order to create a stream that approximates an audio stream, multimedia data must be sent at the correct rate. A typical audio stream over a network has a data rate of 8000 bytes/sec with a sample rate of 160 ms. Each message then must be 8000 bytes/sec * 0.160 sec = 1280 bytes. For each protocol, 1280 bytes of random data were sent every 160 ms.

The router between the sender application and receiving application used a token bucket filter algorithm to handle queue management. Under this scheme, a constant transmission rate is maintained. If messages were received at a faster rate than they can be transmitted, leftover messages were stored in a queue. If the queue filled to maximum capacity, extra packets were discarded. Functionality was added to the normal token bucket filter algorithm to randomly discard packets by a chosen probability. This enabled us to create the desired loss for the network.

One tool used to create competing traffic for the network tests was a UDP packet blaster. UDP packets of size 1024 were sent at a very fast rate across the network. These packets filled the router's queue and created latency across the network If the router queue became full, then a small amount of loss occurred as the router discarded overflowing packets. Varying the sending rate of the traffic generator and the size of the routing queue created the desired latency for each test.

An application was created to send a multimedia stream across the network to the client. The application first initialized the protocol under test with any addresses and parameters necessary to facilitate the multimedia stream. Random numbers were used for the simulated multimedia data along with a sequence number to identify each packet. After each packet the application sleeps, so that the data rate of the multimedia stream is the same as a standard audio stream.

An application was created to receive the test multimedia stream. After initializing the protocol with any parameters and addresses needed, the client application received from the network socket in a tight loop. Every time the socket receiving call returned, the time was recorded as test data and the sequence number of the incoming multimedia message was checked. If the sequence number was not correct, a loss was recorded. When complete, the client application again waited for a reception on the network socket. SRP requires parameters from the application describing the desired loss and latency for the multimedia stream. Certain studies have found that a 10% loss and a 250 ms latency in a multimedia stream is the maximum acceptable[5]. For both tests and analysis of each protocol, a minimum loss of 10% and latency of 250 ms was considered tolerable.

In order to determine the average loss and latency of the test network due only to the traffic generators, before starting each experiment, the traffic generators for the test were started and a single ping was started at a slow transmission rate to measure the state of the network. The pings were averaged together to find the round trip loss and round trip time of the network during traffic contention.

Two main tests were performed: a low loss/low latency test and a high loss/high latency test. Each protocol was controlled by identical applications with the exception of the different actions needed to initialize each protocol such as the TCP's connect and SRP's start function calls. Table 1 lists the one-way loss and round trip time of the network measured by ping flows for each test as well as the routing parameters used to obtain those conditions.

Name	Loss	Round Trip Time	Queue Size	OutgoingLink Rate
Low loss/low lat.	3%	50 ms	30000 bytes	295 Kbps
High loss/high lat.	15%	275 ms	70000 bytes	255 Kbps

Table 1. Table of loss and round trip times

4. RESULTS AND ANALYSIS

For each of the following latency/loss graphs, approximate latency and the total number of lost messages are plotted for each incoming multimedia message. The scale on the left side is for the latency while the scale on the right side shows the number of total messages that have been lost.

As an example, refer to Figure 2 below. Along the X-axis, packet number refers to the sequence number of the message. For each packet number, the data plot indicates the amount of time in milliseconds that it took for the message to travel from the sender to the receiver (latency). Note at approximately packet number 270, there is a spike in the latency data. This is likely a retransmission attempt where the algorithm had to wait for a message to be retransmitted. The scale for this data is given on the left hand side. The data series increasing as a diagonal line shows the losses in the stream. Any time the line steps up, there has been a loss. The scale on the right hand side shows the total number of losses in the stream. Note that some tests do not have losses. TCP never has any loss.

The following quality graphs plot how well each protocol performed with respect to the maximum acceptable loss of less than 10% and maximum acceptable latency of less than 250ms. This was calculated by finding the ratios of average loss to maximum acceptable loss and average latency to maximum acceptable latency. These ratios were plotted for each protocol. The origin of the graph represents the best possible performance, where there is no loss and latency. A value of one for loss or latency represents exactly the maximum acceptable loss or latency. The distance to the origin of each point is related to quality such that the closer the point is to the origin, the better the performance. Distance is calculated using the Euclidean distance and the arc shown on the graph encapsulates the points within the maximum acceptable quality

(distance = $\sqrt{loss^2 + latency^2}$).

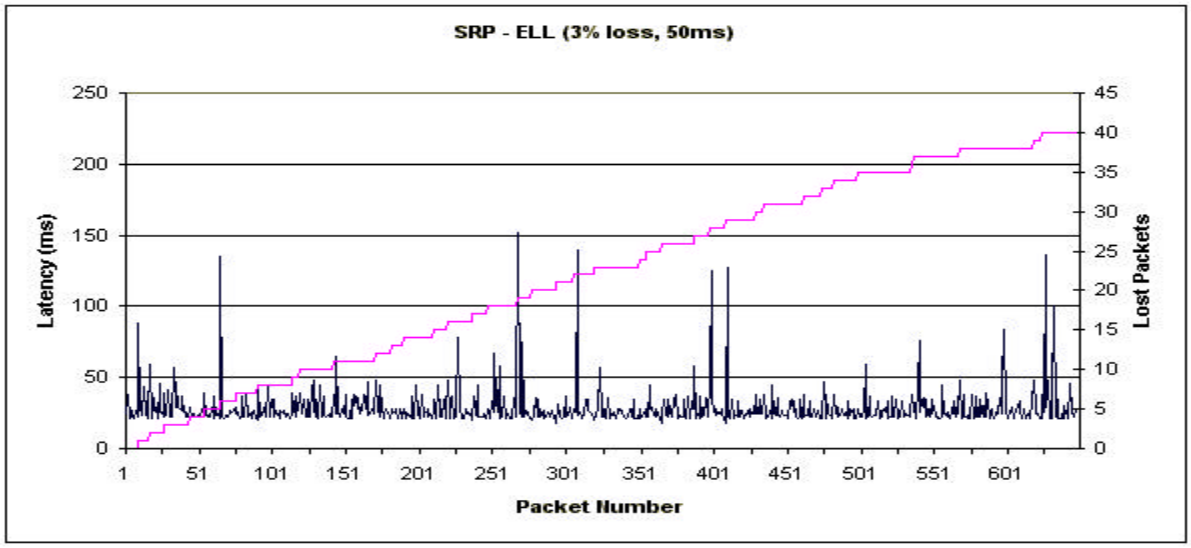

Figure 2. Example graph of loss and latency

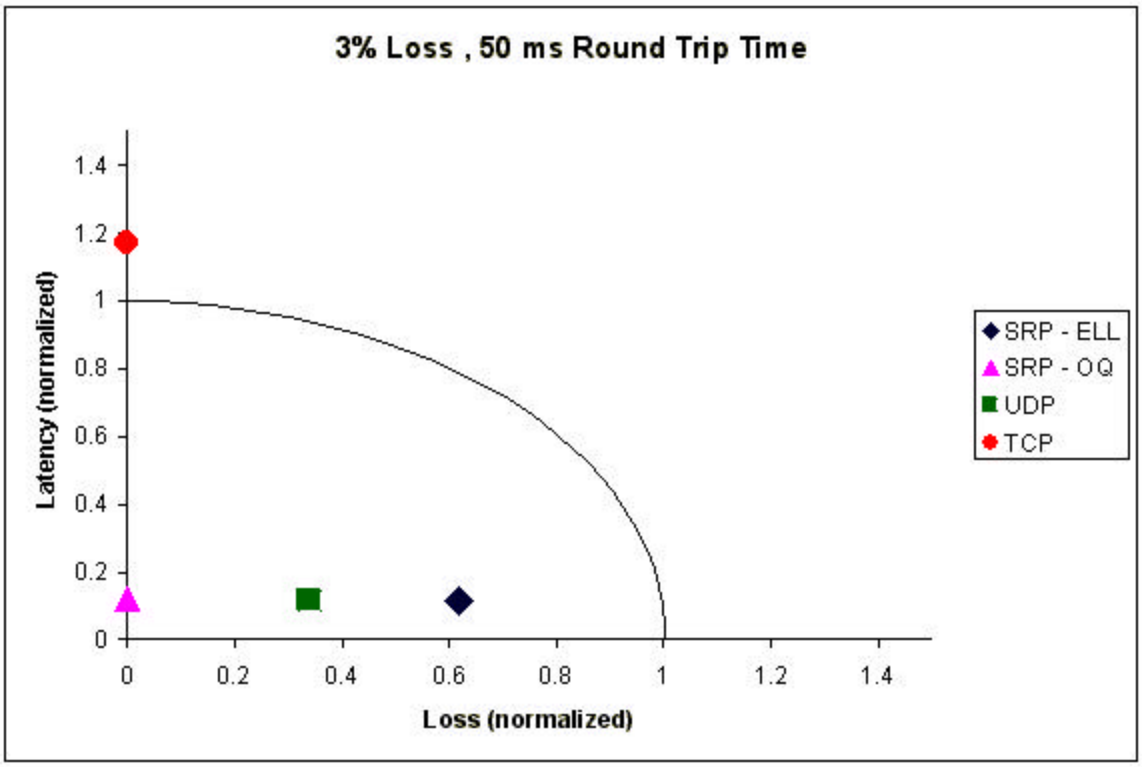

Figure 3. Quality of all protocols under 3% network loss and 50ms round trip time

The first test evaluated the performance of the network protocols under 3% network loss and 50ms round trip time. This test simulates relatively good network conditions. There is loss, but it is small, and the latency is low as well. Figure 3 shows that all protocols except TCP lie within the desired quality level. TCP's flow control mechanism makes the latency calculation difficult. Ignoring congestion control, TCP performs very similarly to SRP – OQ. The congestion control makes TCP slow down and have worse latency than SRP – OQ.

SRP – OQ performs well since the OQ decision algorithm is designed to maintain the least distance from the origin as possible. In this case, since latency is very low, the optimum decision is to retransmit for every lost message, keeping loss low. Latency is slightly higher than UDP because SRP – OQ needs to wait for the retransmissions of lost messages while UDP does not wait. Without retransmission, the UDP's loss is higher.

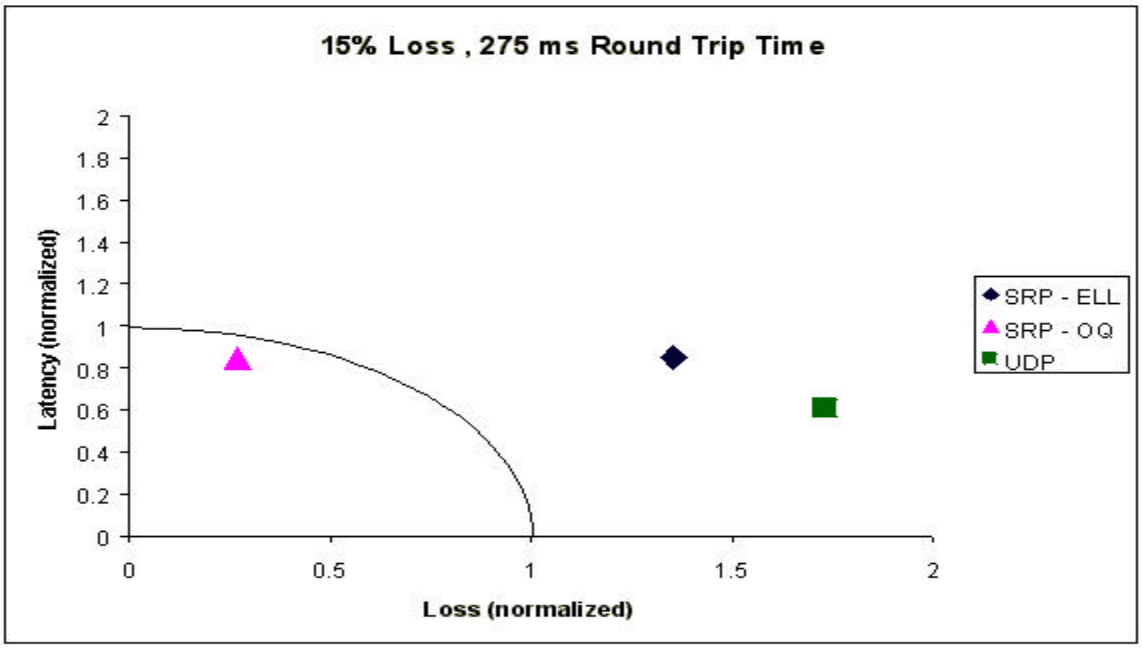

Figure 4. Quality of all protocols under 15% network loss and 275ms round trip time (High Loss High Latency)

The second test evaluates the performance of the protocols under 15% network loss and 275ms round trip latency as depicted by figure 4. SRP – OQ is the only protocol that performed within the desired quality. TCP, mostly due to congestion control, does not have a reasonable quality and could not even be considered for a multimedia stream. The TCP client only received approximately 800 messages during six minutes, while the other protocols received nearly 2000. UDP has a high loss rate causing SRP – OQ and SRP – ELL to have better quality.

5. CONCLUSION

The increase in multimedia applications over the Intemet has placed new requirements on current Internet protocols. The de-facto transport protocols, TCP and UDP, are both designed to support text-based applications with few timing constraints and strict loss requirements. Multimedia applications, on the other hand, can tolerate small amounts of data loss but are very sensitive to delay and variance in delay. Neither TCP, with potentially high latency and no loss, nor UDP, with potentially high loss and low latency, are suitable for multimedia applications.

We propose a Selective Retransmission Protocol (SRP) that requests retransmissions when loss rates get too high (as may happen in UDP), while avoiding high latencies (as may happen in TCP). Thus, SRP can provide lower latency than TCP and lower loss than UDP by selectively retransmitting some, but not all lost packets. Moreover, SRP decides when to request a retransmission based on the application's requirements, allowing SRP to be tuned to achieve the best balance of latency and loss for the application.

To evaluate our proposal, we designed and implemented SRP as an application layer protocol on top of UDP. We simulated an interactive audio-conference running on top of SRP in a Wide-Area network testbed that consisted of end-hosts communicating over a router. By carefully controlling the latency and loss that the router introduced into the stream, we were able to evaluate the performance of SRP compared with TCP and UDP, over a variety of network conditions.

We found SRP generally outperformed both TCP and UDP in respect to the quality of the application. SRP exceeded the quality of TCP and UDP in all network conditions except over a low loss, high latency network, in which UDP did slightly better. Moreover, the algorithm for deciding whether or not request a retransmission can be customized not only to each application, but also to adjust for alternate network environments, such as the low loss, high latency network. Thus, SRP's flexibility and performance provide the potential framework for Internet multimedia applications to have better quality.

6. FUTURE WORK

6.1 Tuning SRP

While performing each test, several characteristics were observed about the SRP protocol and some improvements to these characteristics have been suggested. SRP performs poorly when network conditions change radically and quickly. This characteristic is most likely caused by the need for the SRP protocol to gather statistics about the network such as round trip time and expected arrival time. SRP needs time to adjust these averages to sharp changes in the network. The retransmission timeout needs to be very carefully adjusted. Waiting too long for a message causes high latency. Not waiting long enough causes either an unnecessary retransmission (causing higher network traffic) or a loss even though the message will arrive.

It is possible for SRP to get behind if several consecutive messages are lost or are very late. If the expected data id on the client is much smaller than the data id of the message being sent by the server then the client will drop all messages since they are all too new. Adding a reset function that corrects the expected data id may correct this problem.

It is possible that the decision algorithm can always give up or always retransmit. Usually, this happens only if one of the above described conditions occur. The cause of this is that there is a natural upper bound on loss (100%), however latency can increase indefinitely. The unbounded latency can then overpower even a 100% loss and cause either a decision to always retransmit or always give up. Careful design of the decision algorithm and caps on the amount of loss and latency reported to the decision algorithm can prevent this issue.

6.2 Added Functionality

In the interest of further developing the performance and functionality of the SRP protocol, several improvements can be made. The SRP server can be adapted to handle multiple clients at once, flow control can be added to slow down the transmission rate under heavy traffic conditions, modifications can be made to better handle other media types such as video, and video conferencing with audio and the SRP client may be modified to allow for a reset function if the client gets too far behind in receiving messages. In addition, the decision algorithms used with loss and latency can easily be modified or added to allow for flexibility in different network conditions and desired functionality.

The SRP server can be made to allow multiple clients to attach to one server and request different streams of information. This is important because, in practice, one server could potentially be serving many clients simultaneously. Some changes that would need to be made to the server would involve creating server_socket structures for each connection so that each session can have its own buffer and state information. The server can then share a common receiving thread for all clients. Whenever a retransmission request is made, it would have to be able to identify which session's buffer to retransmit from.

In order for SRP to be friendly to other streams on the network, it is important to implement congestion control. When TCP recognizes that traffic is high on the network, it automatically reduces its bandwidth. A similar congestion control method can be implemented on the SRP server. The client maintains the current loss and latency. This information can be communicated to the server by piggy-backing the information onto time probes and retransmission requests. Since it is more likely for messages to be lost when traffic is high, losses of time probes and retransmission requests can also be used as an indicator to the server that backoff is needed.

Due to the standard maximum transmittable unit (MTU) size of a UDP message over Ethernet, SRP was designed to handle audio streams because an entire multimedia sample will fit in one message. In order to use SRP for video, frames would need to be split into multiple messages in order to fit within the MTU. Care would need to be taken so that a loss of a message does not render subsequent dependent messages useless. A priority scheme may need to be investigated so that important key frames are given more chances to be retransmitted over less important frames.

SRP is currently written as an application layer protocol, this allows for easy porting into many different platforms, but some performance is lost because of the translations through the lower layers. Advancement in performance could come from moving SRP from the application layer to the transport layer. This will increase the speed of the protocol by an application using SRP function calls to transfer data from a client to a server.

IP Multicast[8] provides an efficient mechanism of disseminating data from a sender to a group of receivers. Instead of sending a separate copy of data to each individual receiver, the sender sends a single copy to all receivers. IP multicast, like UDP, provides a best-effort delivery mechanism that results in high packet loss in the presence of network congestion. To obtain TCP-like reliability, the general principle is to have receivers manage their own reliability requirements[9]. Future work involves extending existing reliable multicast approaches to support "semi-reliable" multicast using the principles of SRP.

REFERENCES

1. A. Bouch and A. Sasse. Network Quality of Service: What do users need? University College London. September 1999.

2. M. Claypool and J. Riedl, End-to-End Quality in Multimedia Applications, Chapter 40 in Handbook on Multimedia Comp uting, CRC Press, Boca Raton, Florida, 1999.

3. S. Floyd, M. Handley, J. Padhye, and J. Widmer, Equation-Based Congestion Control for Unicast Applications, In Proceedings of ACM SIGCOMM, Stockholm, Sweden, May 2000.

4. R. Stevens. UNIX Network Programming. Prentice-Hall Inc. Upper Saddle River, NJ, 1998.

5. A. Watson and M. Sasse. Multimedia Conferencing via Multicast: Determining the Quality of Service Required by the End User. University College London, September 1997.

6. X. Guo and C. Pattinson, Quality of Service Requirements for Multimedia Communications, In Proceedings of Time and the Web Workshop, Staffordshire, UK, June 1997.

7. R. Steinmetz, Human Perception of Jitter and Media Synchronization, IEEE Journal on Selected Areas in Communications, Vol. 14, No. 1, January 1996.

8. S. Deering, Host Extensions to IP Multicasting, RFC 1112 January 1989.

9. S. Floyd, V. Jacobson, C. Liu, S. McCanne, L. Zhang, A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing, ACM Transactions on Networking, November 1996.