SRP – A Multimedia Network Protocol

A Major Qualifying Report

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial Fulfillment of the requirements for the

Degree of Bachelor of Science

by

Kenneth French

George Oprica

Michael Piecuch

May 15,2000

Approved:

Assistant Professor Mark Claypool,

Major Advisor

Abstract

SRP – Selective Retransmission Protocol is a protocol designed to increase the performance of multimedia applications. SRP tries to balance the high loss found in UDP, and the high latency found in TCP. SRP uses a decision algorithm to determine whether or not to ask for a retransmission for a lost packet, balancing the loss and latency to an acceptable level for the application's needs. We experimentally found SRP outperformed both TCP and UDP in our quality comparison of the three protocols.

1. Introduction......5
2. Related Work......7
2.1. Multimedia Factors......7
2.2. End-to-end Solutions......10
2.2.1. Streaming Control Protocol......10
2.2.2. Real Time Protocol......11
2.2.3. Selective Acknowledgement......13
2.2.4. Multimedia Enhanced Transport System......14
2.2.5. UDP with Retransmissions......15
2.3. Routing Solutions......16
2.3.1. Class Based Queuing......17
2.3.2. Random Early Detection......17
2.3.3. Reservation Protocol......17
2.4. Multimedia Algorithms......18
2.4.1. Error Correction......18
2.4.2. Error Concealment......19
2.4.3. Encoding......19
3. Approach......20
3.1. Why UDP?......21
3.2. Implementation Details......21
3.2.1. Assumptions......21
3.2.2. Server......22
3.2.3. Client......23
3.2.4. Data id and Expected......24
3.2.5. Timing......24
3.2.5.1. Expected interval......24
3.2.5.2. Retransmission timeout......24
3.2.5.3. Time probes......26
3.2.5.4. Last Normal Reception Time......27
3.2.6. Message Events......28
3.2.6.1. Client Buffer......28
3.2.6.2. Too Early......29
3.2.6.3. Too Late......29
3.2.6.4. Expected Message......29
3.2.6.5. Timeout......30
3.2.6.6. Retransmit......30
3.2.6.7. Give up......30
3.2.7. Decision Algorithms......30
3.2.7.1. Equal Loss Latency (ELL)......31
3.2.7.2. Optimum Quality (OQ)......32
3.2.8. Sending Application......33
3.2.9. Receiving Application......33
4. Tests......34
4.1. Test Network Topology......34
4.2. Test Scenarios......35
4.3. Tools......36

4.3.1. Token Bucket Filter Router 36
4.3.2. Traffic generator 36
4.3.3. Test Server Application 36
4.3.4. Test Client Application 37
4.4. Human Perception 37
4.5. Experimental Control 37
4.6. Test Parameters 38
5. Analysis 38
5.1. Data Analysis Procedure 38
5.2. Data and Graphs 39
5.2.1. Low loss/Low latency 41
5.2.2. Low loss/High latency 45
5.2.3. High loss/High latency 49
6. Conclusion 52
7. Future Work 52
7.1. Tuning SRP 52
7.2. Added Functionality 53
8. Appendix 57
8.1. Client Interface Functions 57
8.1.1. srp_client_start 57
8.1.2. srp_client_stop 58
8.1.3. srp_recvfrom 58
8.2. Server Interface Functions 59
8.2.1. srp_server_start 59
8.2.2. srp_server_stop 60
8.2.3. srp_sendto 61
8.3. SRP Packet Definitions 62
9. Bibliography 63

1. Introduction

Since the beginning, the Internet has been dominated with traffic from text based applications such 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. Examples of multimedia applications include video on demand and teleconferencing.

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 different requirements from text-based applications. An audio stream, for example, requires that data is received in a timely fashion and is more forgiving of lost data. If a data packet arrives at the player too late, it misses the time it is 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. TCP is ineffective due to the overhead of retransmission and ignorance of timing factors. 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 delay of TCP and the 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. Several tests were performed with network conditions of low loss/low latency, low loss/high latency, and high loss/high latency.

2. Related Work

2.1. Multimedia Factors

When designing a protocol for multimedia, many factors need to be considered. The quality of multimedia streamed over the Internet is affected by several factors. Loss, round trip time, latency, jitter, consistency, data rate, and adaptation to congestion all change the perceived quality that a user sees or hears from their multimedia application and the resulting impact on the underlying network.

- Loss can severely affect the quality of a multimedia stream. If data is lost during a network transfer, it obviously cannot be played to the user. Depending on the application, loss can have a greater or lesser affect on the quality of playback perceived by the user. Video, for instance, is more forgiving to loss than audio. There are also many types of error correction and concealment that can lessen the effect of a small amount of network loss. SRP attempts to reduce the amount of loss by retransmitting for loss messages if it predicts that latency will not increase too much.

- Round Trip Time (RTT) refers to the amount of time that it takes for a message to travel from the sender to the receiver or from the receiver to the sender, and then back again. In a negative acknowledgement system, the amount of time to retransmit a lost multimedia message is heavily dependent on round trip time. A high round trip time can cause high latency when retransmitting for lost messages. The SRP protocol can decide not to retransmit a loss due to the high latency penalty caused by a high round trip time.

- Latency is defined as the time interval from when multimedia data is sent from the server to when it is seen or heard by the user. Latency becomes important for interactive applications such as conferencing over the Internet. During a conference, people need to have a tool that responds naturally, in the same way as two people would meet in real life. Long delays between words and sentences caused by large latency make communication unnatural. Multimedia broadcasting applications, such as video or audio streamed from a server, are not impacted heavily by latency since there is no real time interaction from the receiver to the sender. Latency only causes the playback to start later. Changes in latency, however, cause another phenomenon called jitter, which can greatly affect any multimedia stream. Latency is controlled under SRP by giving up on messages that are too late and continuing to the next message.

- Jitter is flickering seen in video or gaps heard in audio, and is caused by variance in the latency of a system. For example, assume ten packets of multimedia data are sent from the sender to the receiver. The first five arrive quickly, then the next five take much longer to arrive. Jitter will be seen between the fifth and the sixth packets since the first five data packets have been played back to the user completely before the sixth packet arrives. With no data to play, there is a gap in multimedia stream until the sixth packet arrives. Without correction or concealment, the user will notice these gaps in the playback.

- There have been several studies on network multimedia quality and perceived quality. Consistency was found to be a major factor in perceived quality [BS99]. Audio or video should stay at a consistent quality, and not fluctuate between high and low quality. If the multimedia stream is started at a high quality, and then the network becomes saturated with congestion, a lower quality stream may be sent, and then the high quality stream may be resumed when conditions return to normal. The changes between these different quality levels often degrade the overall quality for end users.

- The data rate is the amount of data that can be sent between two hosts in a given time. This greatly affects multimedia quality and performance because with a higher data rate more multimedia data can be sent in a smaller time. Higher data rates can allow for a higher quality stream, or added error correction support since each message has more room for additional multimedia data or redundancy information.

- Congestion is a factor in today's networks. If a network is congested, then data can be lost, greatly affecting multimedia performance issues such as increased jitter and decreased data rate. Detecting and adapting to congestion is important since congestion will not likely correct itself unless all participants reduce their usage.

2.2. End-to-end Solutions

There has been a great deal of research and development in the field of multimedia networking. Many have considered different combinations of the above stated factors. Each has their own characteristics, advantages, and disadvantages with varying similarities and differences to SRP.

2.2.1. Streaming Control Protocol

SCP (Streaming Control Protocol) was developed as a new flow and congestion control scheme to aid real time media delivery over the Internet [CPW98]. Its main design focuses were to share the network bandwidth fairly with other protocols such as TCP and to improve smoothness in streaming of media over a network. It accomplishes these goals by using sender-based discovery of network bandwidth usage and congestion. This is a similar approach to TCP in which the sender finds the state of the network when acknowledgments (ACKs) are not received or are received out of order.

For each packet that SCP sends, it marks the time of its transmission and starts a timer. Based on the timer events and when ACKs are received, the sender adjusts the size of the congestion window. In this manner, it is able to estimate network buffer size, bandwidth usage, and round trip time. It uses this information to determine in what state it should be operating in, slowStart, steady, congested, or paused. In the slowStart state, SCP increases the congestion window until it detects network congestion or full utilization of bandwidth. Once it is fully utilizing the bandwidth, it switches to steady state in which it sends an appropriate number of extra packets in case the bandwidth increases. SCP switches to congestion mode when it detects that a packet was lost or delayed. It saves the current steady state in case ACKs are received out of order and restores to the steady state if that is the case. Otherwise, it reduces the congestion window size in half and increases the timeout for ACKs. This leads to an exponential back off if packets are consecutively lost. SCP also has a fourth state, the paused state, which happens when the sender has no data to send.

Although SCP is able to effectively smooth out multimedia playback, it provides no hooks for multimedia QoS requirements; all data sent using the protocol is treated the same. Our protocol, SRP, handles different media streams in different manners. Latency and loss thresholds are different for varying media streams and these streams cannot be handled the same way. SRP deals with this concept by allowing the user to set parameters specifying the desired loss and latency. SRP then requests retransmissions for enough messages to maintain those desired parameters whereas SCP never retransmits. SRP, using negative acknowledgements, retransmits for more data if total loss is becoming high, or gives up if latency is becoming too great. The quality settings for this decision can be defined differently for any type of multimedia stream.

2.2.2. Real Time Protocol

RTP (Real-Time) Protocol is another multimedia streaming protocol that is currently used by multiple applications and is an RFC in the Internet community as RFC 1889/1890 [SCFJ96]. It is an application layer protocol that relies heavily on UDP and IP as its subcomponents, but can potentially be used over any transport protocol [KR99]. RTP may be used in conjunction with other protocols such as Real Network’s RTSP (Real-Time Streaming Protocol) which controls the connection, although RTSP is not obligated to use RTP [SRL98] . RTP is unidirectional, from sender to receiver, but supports both unicast and multicast streams by using a helper protocol, RTCP (Real-Time Control Protocol), also defined in RFC 1889. RTCP is bi-directional and provides a control channel between the sender and the receivers. It is used to identify the participants in a multicast, describe the content, quality of service information, and request for retransmissions [SCFJ96].

RTP attempts to provide a standard for multimedia applications to communicate with each other better than they would if using a proprietary protocol. To maintain this scalability, the RTP header has a 7-bit long field that describes the payload type. This means that RTP can be used for up to 128 different media types. If two different applications are using RTP and transferring the same data type there is a good possibility that they will be able to communicate properly. The RTP header also contains a 16-bit sequence number used to detect lost or out of order packets. The application can try to conceal or retransmit missing packets once it has detected loss. This field is not used for timing of the multimedia playback synchronization. The timestamp, which is a NTP (Network Time Protocol) standard timestamp, is used for synchronizing media [SCFJ96]. Because video and audio are transmitted via two different RTP sessions, synchronizing of the data is left up to the application. It uses the timestamp to piece the data together. The timestamp is not based on the time that the packet was sent, but on when in time the media was generated. Even if there is no data from the input source, the timestamp is still incremented.

Although RTP provides the mechanisms for detection and retransmission of lost data, it does not do so on its own. It places the task of retransmission on the application. RTP also does not provide any resource reservations or quality of service control and has no congestion control policy, which could result in network congestion. Because the congestion control policy is left up to the application, RTP itself cannot share the network bandwidth fairly between itself and other protocols [Sc98]. RTP simply adds sequence numbers and a means for multimedia synchronization on top of UDP. It is efficient as an addition to UDP because it has a small header of 4 bytes and is completely encapsulated in a UDP packet, but does not automatically provide retransmission or any congestion avoidance.

2.2.3. Selective Acknowledgement

Current implementations of TCP do not do well with multiple lost packets from one window of data. Because of the way the window size and acknowledgements work in TCP, some acknowledgements may get delayed and the sender may retransmit unnecessarily if a timeout occurs [MMR96]. SACK, Selective Acknowledgements has been proposed and implemented to recover from packet loss in a more efficient manner. It does not affect TCP's congestion control algorithms, but only allows the sender to better know which packets were received or lost [FF96].

TCP SACK is an extension of TCP that allows an acknowledgement to have a short list of blocks of packets that have been received. Instead of keeping track of individual packets, the ACK can effectively show which packets were lost and which were received late. The sender using this SACK info can send lost packets before sending new ones. SACK also allows the sender to recover from multiple lost packets in one round-trip time instead of multiple [G97].

Although SACK increases performance for TCP traffic, it is still not suitable for multimedia. If there is serious network congestion, TCP SACK will still have unbounded delay because the protocol times out and waits for retransmissions. Even when used in conjunction with FACK (Forward Acknowledgement), which keeps track of the total number of bytes outstanding in the network, it is not capable of delivering media effectively over the Internet [MMR96]. The concept of multiple messages being acknowledged using a single message could be added to SRP to provide better bandwidth usage and better support for blocks of several related multimedia messages.

2.2.4. Multimedia Enhanced Transport System

The METS (Multimedia Enhanced Transport System) protocol, developed by Andrew Campbell and Geoff Coulson, incorporates a QoS oriented API and a range of mechanisms to assist applications in exploiting QoS and adapting to fluctuations in QoS [CC97].

METS has two methods of flow control, one is the flow scheduler that schedules application level frames on a coarse grained frames per second basis. The flow shaper method is invoked every 1 ms to maintain a token bucket that controls the output level. Each flow accumulates credits, which represent the number of ATM cells to transmit to the network over the next interval. When the shaper runs it visits all the flow queues in a round robin fashion and if the queue contains cells and has available credits, then the shaper transmits one cell from the queue and decreases the token value.

METS is different from SRP, because it uses a different protocol structure than SRP. METS makes a layered approach into the Transport and Network layer to affect the QoS of the transmitted information. SRP is implemented on the Application layer, using the lower layers such as UDP to accomplish QoS requirements.

METS is also different from SRP because of its use of its use of acknowledgement packets. SRP sends negative acknowledgement packets to request retransmissions for lost packets only if needed.

2.2.5. UDP with Retransmissions

The well-known and simple TFTP file transfer protocol uses a request-reply system on top of UDP. It is a fairly reliable method of transmitting data over a network because the client requests individual packets from the server. If the client does not receive a response from the server in a certain amount of time, it retransmits the request. It repeats this a number of times and if then it gives up.

When dealing with multimedia streams, this technique is undesirable for several reasons: the client can fall behind; if a packet is lost, it takes linear time to decide to retransmit; round trip time is not constant; retransmissions are ambiguous. The server has multimedia data available at a constant rate. The client can easily lag behind if there is high loss on the line and the requests it sends to the server, or replies from the server, are lost. Deciding to retransmit a lost packet takes a fixed amount of time. This timeout does not take into consideration RTT (Round Trip Time) which is not a constant value. A solution to this problem uses a smoothed estimate of RTT similar to that of TCP. Calculating a valid RTT creates another problem. When a retransmission is sent and a response is received, the client cannot tell if the response is from the original request, or from the retransmitted request. This dilemma is called the retransmission ambiguity [S98]. A simple solution is for the client to attach a timestamp to each request it sends. The server would then return the requested data as well as the timestamp that the client sent. The client is able to calculate RTT on every reply it receives. Because the client is sending and receiving the timestamps, there is no need to have synchronized clocks on the client and server.

The request-reply system works well for simple and small applications such as TFTP, but when the idea is applied to a multimedia stream, it is not a good alternative. Many modifications must be made to this system to allow it to efficiently receive data from a streaming multimedia server.

UDP with retransmissions is very similar to SRP due to the way it adds retransmissions to UDP. It is not, however, designed use with multimedia streams. UDP with retransmissions does not allow loss and has no timing considerations. The SRP decision algorithm also remains unique to SRP.

2.3. Routing Solutions

There are several methods available for solving multimedia issues related on routers, the intermediate nodes between end systems. Some of these solutions involve router queuing algorithms used to decide when a message should be sent or deleted when there is congestion. Also, there are Quality of Service (QoS) algorithms that allow for the reservation of network resources. Since SRP is an end-to-end protocol, router algorithms do not directly apply to the implementation of SRP. They do, however, influence how effectively SRP can transmit data. A routing algorithm can assist an SRP type flow by allocating resources or sending SRP type messages before other types of messages or, on the other hand, the algorithm can harm or even cripple a SRP flow by potentially deleting all SRP messages. Unless very strict bandwidth allocation is maintained by a routing solution, there still needs to be an end to end control management system for a multimedia stream.

2.3.1. Class Based Queuing

Class based queuing (CBQ) is a router queuing algorithm that can identify what type of messages are in its routing queue [PJS99]. Based on this information, the algorithm can determine a send order or which packets to drop. Depending on its configuration, this algorithm has the potential to assist UDP flows, such as SRP, by acting more favorably to SRP messages than other types of messages. Conversely, the algorithm can also be set to act unfavorably to an SRP flow.

2.3.2. Random Early Detection

Another router queuing algorithm is random early detection (RED) and a flow-aware version, flow random early detection (FRED) [PJS99]. RED controls router queue congestion by randomly removing messages from its queue after a certain congestion threshold is exceeded. Probabilistically, this algorithm drops more messages from high bandwidth, non-responsive flows. FRED increases the fairness of the RED algorithm by being aware of the specific flows in the queue and removing a percentage from each. These algorithms can assist SRP by removing high bandwidth, non-responsive flows from the network. There is an addendum to RED, RED-DPM, that acts favorably to multimedia flows and removes "stale" messages (data that is too old to be played to the user in time) [PJSB98].

2.3.3. Reservation Protocol

A different type of algorithm that can be used to control the factors related to multimedia is the reservation protocol (RSVP). Instead of trying to identify certain types of flows as they enter the queue, RSVP explicitly allocates bandwidth to specific flows.

Applications can request the amount of bandwidth expected to be used. The router can then manage the amount of bandwidth that is available and the amount that each application flow requires.

2.4. Multimedia Algorithms

In addition to routing algorithms, there are multimedia algorithms that can allow transmission of higher quality multimedia. Some techniques include error correction, the repair of audio or video data due to loss or error, error concealment, methods of hiding errors or loss in audio or video so that the problem is not as noticeable to the user, and encoding techniques such as compression and separating and prioritizing video and audio data [PHH98]. Since all of these methods deal with the processing of multimedia data, they are out of the scope of SRP. These methods affect how effectively SRP performs, as well as how well the multimedia application tolerates loss from SRP transmission. SRP, for instance, can be combined with an error correction or error concealment algorithm to repair gaps caused by the SRP retransmission scheme.

2.4.1. Error Correction

Error correction techniques can reconstruct some lost data from a multimedia stream. Forward error correction (FEC) is an error correction technique that reconstructs lost multimedia data by transmitting redundant information in each message. In a video application, for example, each frame may be sent in a single message. Each message would also contain a portion of the previous frame as well. If a frame was lost, it could be partially reconstructed from the redundant information in the next frame. The disadvantage of this technique is the increased bandwidth usage. Interleaving is another error correction technique. It functions by mixing smaller chunks of multimedia data non-consecutively across several messages. If a message is lost, there are only short gaps in the playback spread out over a larger time, rather than a large gap all at once.

2.4.2. Error Concealment

Another class of multimedia algorithms is error concealment. Loss or error is made less apparent to the user during playback. One technique for error concealment is data insertion. When data from a multimedia stream is lost it is replaced by some type of data so that the gap is less noticeable. The replacement can vary from pure silence, white noise, the repetition of the previous data, to interpolation of a new data from surrounding data or fitting data from a pattern. The multimedia stream can also be spliced temporally to completely ignore lost data. These techniques vary in quality and computation time.

2.4.3. Encoding

Another application technique related to perceived multimedia quality is encoding. Multimedia can be encoded in a multitude of ways, each with differing quality, compression, data rate, and computation time. This can greatly affect the impact of a loss or an error and determines the quality of the playback stream. Multimedia data can also be formatted in priority layers. A low bandwidth, low quality layer provides the minimum amount of data for a correct playback. If congestion is low, a higher quality layer can be added to the minimum layer to provide the full playback potential of the multimedia stream.

3. Approach

SRP is 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 sender 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 retransmit for the message. This decision is performed by an algorithm that takes into account how much loss and latency quality is requested from 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 it is needed.

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

3.1. **Why UDP?**

SRP operates above UDP and offers added functionality to the already established protocol. The Internet protocols, TCP and UDP, have been established for several decades. Both have been implemented countless times on different computing platforms with different modifications and have been relentlessly studied and tested. The result is a set of stable protocols. Taking advantage of this, SRP has been designed to leverage off of UDP. Acceptance of SRP into the Internet community is easier since a SRP message is essentially the same as a UDP message.

The most important reason UDP was chosen over TCP is that UDP does not implement retransmission. Since SRP has its own specialized retransmission system, TCP would need to be replaced, while with UDP, with no retransmission system, this functionality could be added with no modification. Not only does this make it easier to implement since UDP does not need to be rewritten, but also it maintains the stability and acceptance of the UDP protocol. The installation of the SRP protocol into future applications is easier, as well, because a layer can be added between UDP and the application leaving nothing further to be modified.

3.2. **Implementation Details**

3.2.1. Assumptions

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, latency cannot be exactly 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 go 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 frame 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 above 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 testing, the network conditions during each test is constant. Even though there may be variations in network traffic for a particular test, the loss and latency of the network should remain fairly is constant for each protocol. Thus, each protocol has to deal with the same type of traffic.

3.2.2. Server

The SRP server sends a multimedia stream to the client. The 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 including 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 its sender.

3.2.3. Client

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, it is ignored and the socket is checked for more messages. If the correct data is not received within before retransmission timeout (RTO) then a decision algorithm is consulted on whether or not to retransmit for 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.

3.2.4. Data id and Expected

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 which 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.

3.2.5. Timing

3.2.5.1. Expected interval

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 doesn't arrive at the expected time then a decision can be made whether or not to retransmit for it.

3.2.5.2. Retransmission timeout

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.

[S98] Every time an interval is measured it is averaged into the current smoothed average as follows:

```
Sexp = α * Sexp + (1 - α) * Exp

Sexp:  Smoothed expected interval average
Exp:  Current measured expected interval
α:  Smoothing factor.  (SRP uses 0.85 found
experimentally [S98] )
```

The amount of deviation in the expected interval is then calculated as follows:

```
Sexpdev = α * Sexpdev + (1 - α) * abs(Exp - Sexp)

Sexpdev:  Smoothed expected interval deviation
Sexp:  Smoothed expected interval average
Exp:  Current measured expected interval
α:  Smoothing factor.  (SRP uses 0.85 found
experimentally [S98])
```

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 the following formula based from the RTO calculation from TCP [S98].

```
RTO = Sexp + 4 * Sexpdev

RTO:  Retransmission timeout
Sexp:  Smoothed expected interval average
Sexpdev:  Smoothed expected interval deviation
```

For a retransmission, the expected time is RTT since the server will immediately return a retransmission request if the data is in the server buffer. RTO is the following for a retransmission:

```
RTO = SRTT + 4 * SRTTdev

RTO:  Retransmission timeout (for a retransmission)
SRTT:  Smoothed round trip time
SRTTdev:  Smoothed round trip time deviation
```

3.2.5.3. Time probes

Maintaining an accurate round trip time is 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 as follows:

```
SRTT = α * SRTT + (1 - α) * RTT

SRTT:  Smoothed round trip time average
RTT:  Current measured round trip time
α:  Smoothing factor.  (SRP uses 0.85)

SRTTdev = α * SRTTdev + (1 - α) * abs(RTT - SRTT)

SRTTdev:  Smoothed round trip time deviation
SRTT:  Smoothed round trip time average
RTT:  Current measured round trip time
α:  Smoothing factor.  (SRP uses 0.85)
```

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).

3.2.5.4. Last Normal Reception Time

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.

Only 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. Determining the exact time to timeout for a retransmission is then 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. Last last normal reception time (last_LNRT) then 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:

```
ETA = (Last_LNRT + 1) * Sexp + LNRT

ETA: Estimated time of arrival
Last_LNRT: Number of normal messages lost since last
           LNRT update
LNRT: Last normal reception time
Sexp:  Smoothed expected interval average
```

The time interval to wait before deciding to retransmit can then be calculated by taking current time and network deviation into account as follows:

```
Timeout = (ETA - Curtime) + 4 * Sexpdev

Timeout:  Time to wait until deciding to retransmit
Curtime:  Current time
ETA:  Estimated time of arrival (computed above)
Sexpdev:  Smoothed expected interval deviation
```

3.2.6. Message Events

3.2.6.1. Client Buffer

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 last_LNRT 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.

3.2.6.2. Too Early

If a message is not found in the client buffer, a socket read is performed. If a message is received with 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.

3.2.6.3. Too Late

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.

3.2.6.4. Expected Message

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. If the message was a retransmission then the last_LNRT 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.

3.2.6.5. Timeout

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.

3.2.6.6. Retransmit

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.

3.2.6.7. Give up

When it is determined that the expected message has been lost and the decision function chooses not to retransmit for the message, there is a give up event. During a give up event, last_LNRT 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.

3.2.7. Decision Algorithms

A decision algorithm governs 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.

3.2.7.1. Equal Loss Latency (ELL)

The equal loss latency (ELL) decision algorithm decides whether or not to retransmit by estimating the resulting loss and latency if the expected message is retransmitted for and if a loss is taken by not retransmitting for it. 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 desired loss and latency to the client at initialization. The current loss ratio is then calculated as follows:

```
LossRatio = CurLoss/DesiredLoss

LossRatio: Loss ratio
CurLoss: Current loss
DesiredLoss: Desired loss given by application
```

Clearly, smaller loss ratios are desirable, a loss ratio of zero is perfect and a loss ratio of one is exactly desired. The latency ratio is calculated similarly as follows:

```
LatRatio = CurLat/DesiredLat

LatRatio: Latency ratio
CurLat: Current latency
DesiredLat: Desired latency given by application
```

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)

Diff: Difference between loss and latency
LossRatio: Loss ratio with estimated loss averaged if
           needed
LatRatio: Latency ratio with estimated latency
          averaged if needed
```

The lower of the two differences is then the choice of whether or not to retransmit.

3.2.7.2. Optimum Quality (OQ)

The optimum quality (oq) decision algorithm decides whether or not to retransmit by estimating the resulting loss and latency if the expected message is retransmitted for or given up on. Whichever result has a loss and latency closest to zero is chosen. The loss and latency ratios estimated for both retransmitting and not retransmitting are calculated exactly the same way as the equal loss latency algorithm. To calculate which is better, the Euclidean distance is calculated of the loss and latency ratio coordinates from the origin as follows:

$$SqrDist = LossRatio^2 + LatRatio^2$$

```
SqrDist: Square distance from the origin
LossRatio: Loss ratio with estimated loss averaged if
needed
LatRatio: 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.

3.2.8. Sending Application

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 it to 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. The protocol has the ability to dynamically estimate the data rate only if it changes slowly.

3.2.9. Receiving Application

There are several design considerations necessary when making a receiving application to interface with the SRP client. The receiving application is expected to create a valid UDP socket and provide and 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.

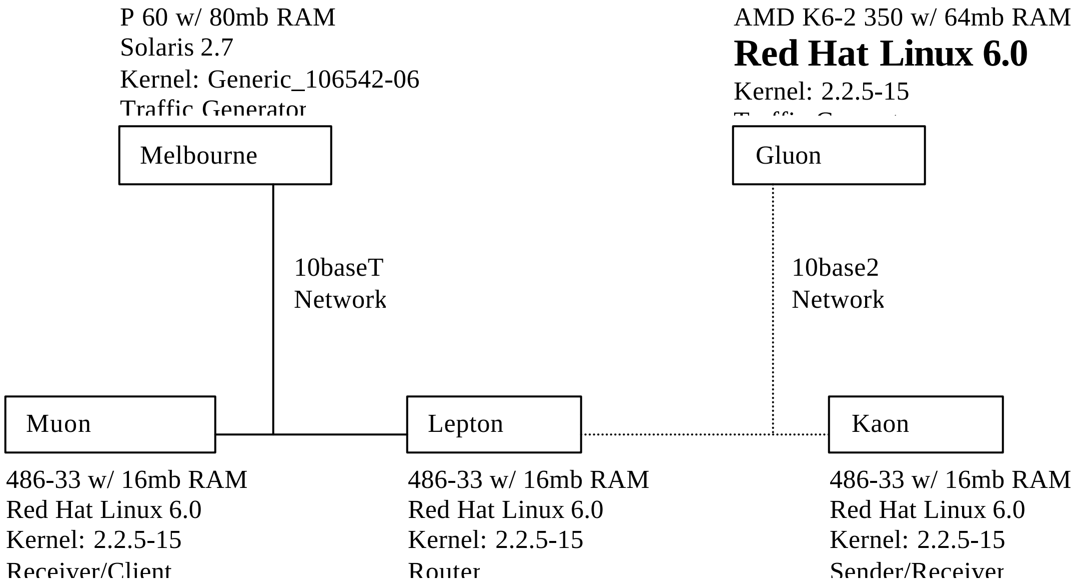
The receiving application has certain timing constraints. The application must continuously poll the SRP client for more data. The turnaround time between polls should be minimal so that arrival times of messages remain accurate.

4. Tests

After developing the SRP protocol, its performance needed to be tested against the performance of the standard 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.

4.1. Test Network Topology

The following topology diagram 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 outside users.



4.2. Test Scenarios

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.

Four main tests were performed: a low loss/low latency test, a low loss/high latency test, and a high loss/high latency test. A high loss/low latency test was not performed since situation this is uncommon on the Internet [WS97]. Traffic generating tools and a token bucket filter routing algorithm were used to create identical traffic conditions for each protocol. 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 srp_client_start function calls.

4.3. Tools

4.3.1. Token Bucket Filter Router

The router between the sender application and receiving application uses a token bucket filter algorithm to handle queue management. Under this scheme, a constant transmission rate is maintained. If messages are received at a faster rate than they can be transmitted, leftover messages are stored in a queue. If the queue fills to maximum capacity, extra packets are 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.

4.3.2. Traffic generator

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 become full, then a small amount of loss occured 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.

4.3.3. Test Server Application

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 is sent the application sleeps, so that the data rate of the multimedia stream is the same as a standard audio stream.

4.3.4. Test Client Application

An application was created to receive the test multimedia stream. After initializing the protocol with any parameters and addresses needed, the client application receives from the network socket in a tight loop. Every time the socket receiving call returns, the time is recorded as test data and the sequence number of the incoming multimedia message is checked. If the sequence number is not correct, a loss is recorded as test data. When complete, the client application again waits for a reception on the network socket.

4.4. Human Perception

SRP requires parameters from the application describing the desired loss and latency for the multimedia stream. Certain studies have found that humans can perceive a 10% loss and a 250 ms latency in a multimedia stream [WS97]. For both tests and analysis of each protocol, a loss of 10% and latency of 250 ms is considered desirable.

4.5. Experimental Control

In order to determine the average loss and latency of the test network due only to the traffic generators, a method was devised to test the state of the network before the tests were performed. 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. Many samples of the ping were averaged together to find the round trip loss and round trip time of the network during traffic contention.

4.6. Test Parameters

The following table 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	Router Queue	Router rate
Low loss/low lat.	3%	50ms	30000 bytes	295Kbps
Low loss/high lat.	3%	400ms	130000 bytes	295Kbps
High loss/high lat.	15%	275ms	70000 bytes	255Kbps

5. Analysis

During the medium and heavy traffic tests on each of the protocols, loss and arrival time statistics were gathered for analysis. The data was complied and graphed so that the performance of the protocols could be compared. Each protocol's performance is analyzed separately and then all the results are compiled together and compared.

5.1. Data Analysis Procedure

During data analysis, special care was taken to correctly gather the desired statistics needed to compare the performance of each protocol. Loss, latency and average frame rate were of particular interest.

- Loss statistics were gathered by the test applications by checking the sequence numbers of incoming multimedia messages returned from the protocol’s network socket. Gaps in the sequence numbers of messages were considered losses.

- Latency cannot be calculated exactly without synchronized clocks between the client and server. In order to approximate latency using round trip time and arrival time, latency is estimated as the time for a message to travel from the server to the client plus the time the protocol adds to the latency due to waiting for messages that were lost or for retransmissions. The time for a message to travel from server to client can be estimated as round trip time / 2 and is identical for every protocol. The time added by the protocol can be estimated as the variance in the arrivals, the average distance of all arrivals from the mean.
- Average frame rate is the average rate that messages are received. The frame rate can describe how much the server’s transmission rate was slowed due to blocking in the protocol’s send command. This is of particular interest in TCP because exponential backoff severely reduces the average frame rate to the point where the protocol cannot be used for a multimedia stream.

5.2. Data and Graphs

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.

The following quality graphs plot how well each protocol performs with respect to the desired loss of less than 10% and desired latency of less than 250ms. This is calculated by finding the ratios of average loss to desired loss and average latency to desired latency. These ratios are 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 desired 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 Pythagorean theorem (distance = $\sqrt{loss^2 + latency^2}$). The arc shown on the graph encapsulates the points within the desired quality.

As an example, refer to figure 1a 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.

5.2.1. Low loss/Low latency

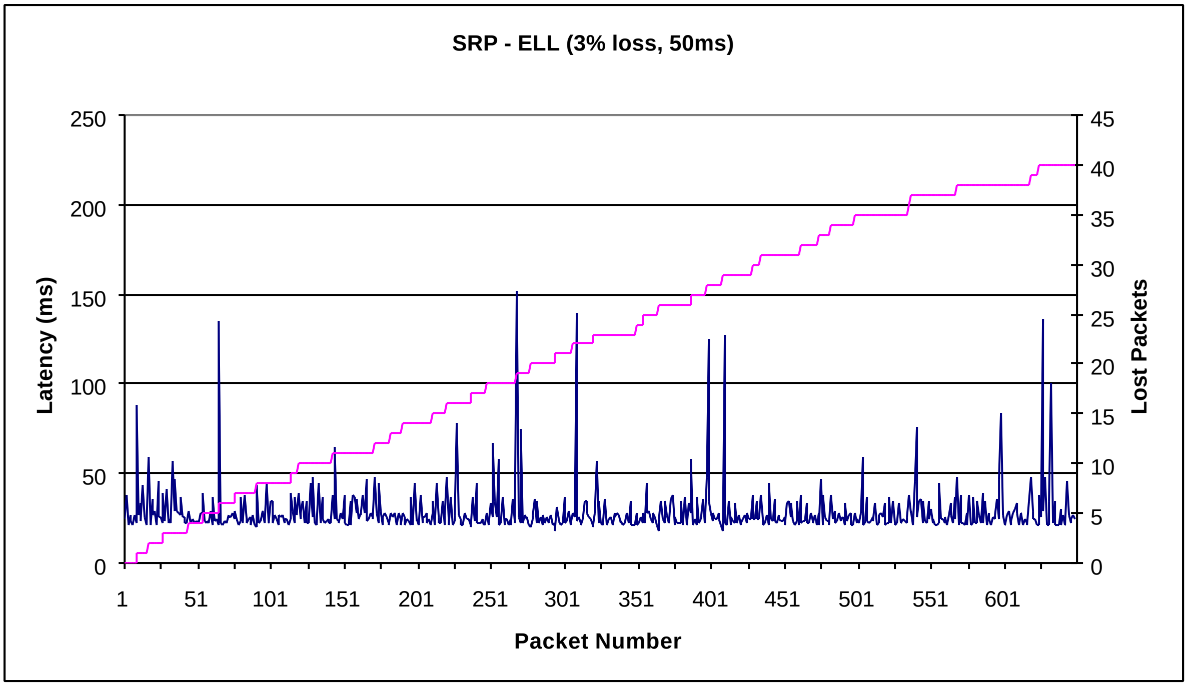


Figure 1a: Loss/Latency SRP ELL algorithm under 3% network loss and 50ms round trip time

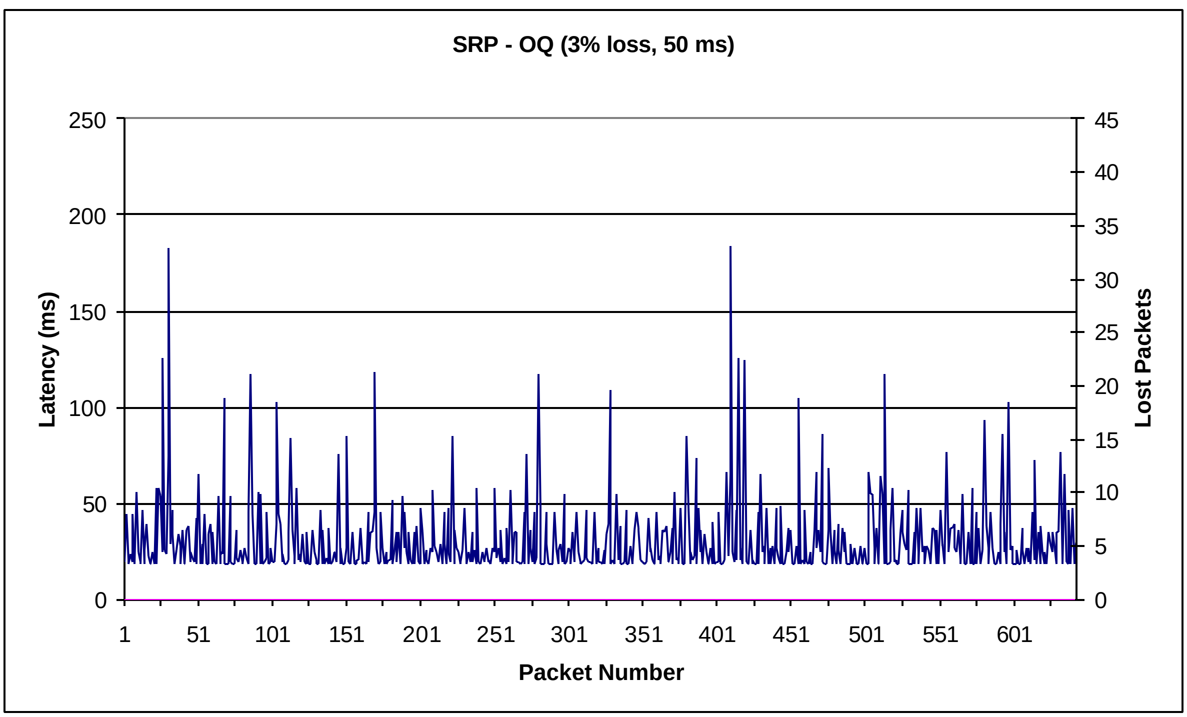


Figure 1b: Loss/Latency SRP OQ algorithm under 3% network loss and 50ms round trip time

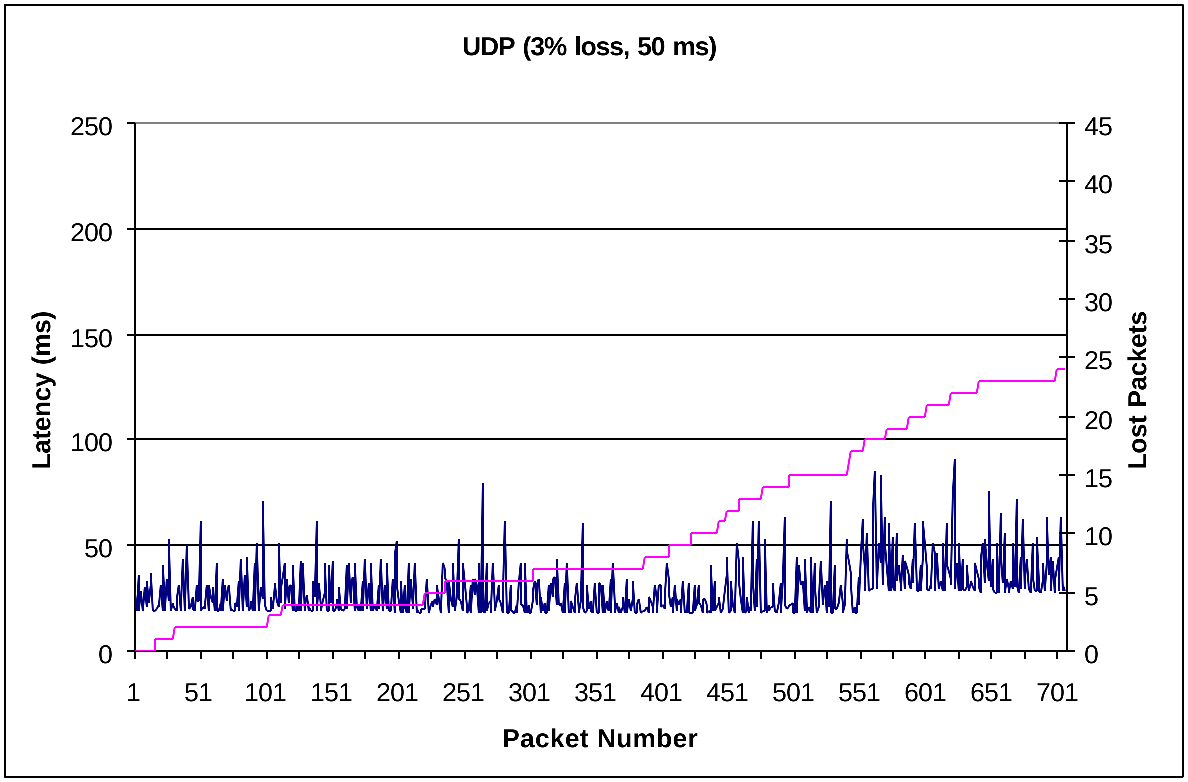


Figure 1c: Loss/Latency UDP under 3% network loss and 50ms round trip time

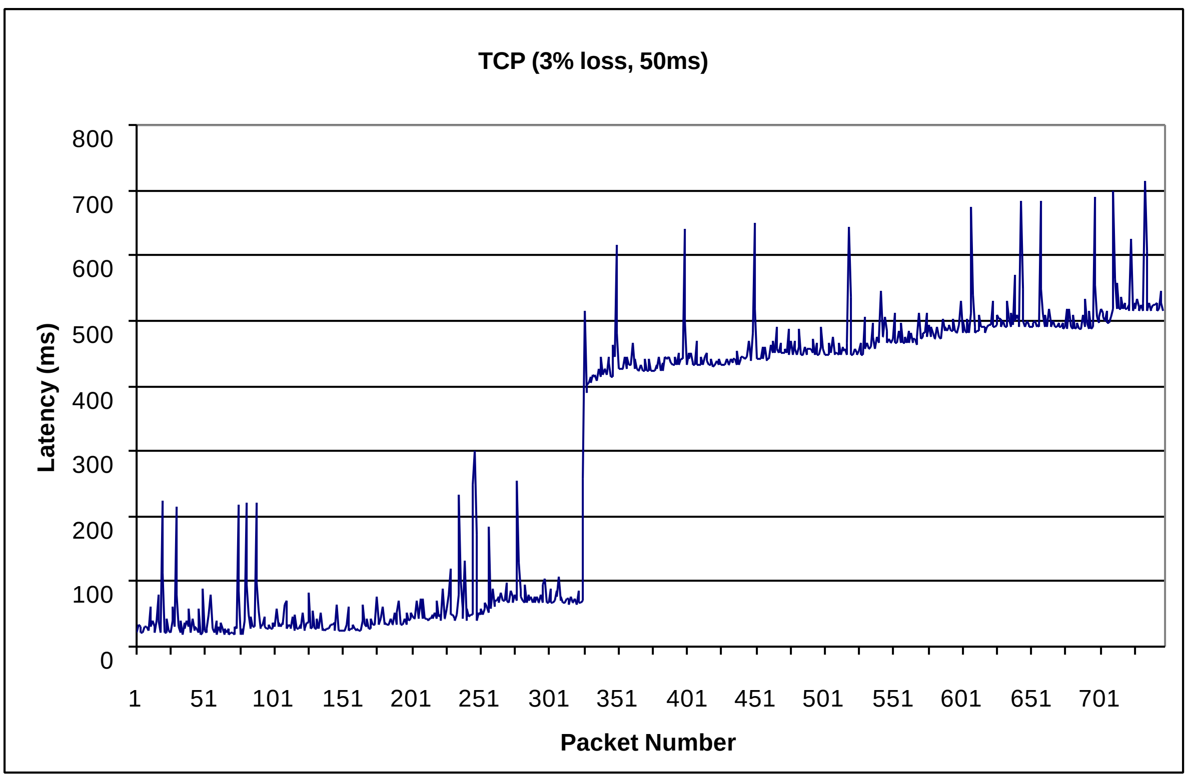


Figure 1d: Loss/Latency TCP under 3% network loss and 50ms round trip time

Protocol	Latency	Loss	Distance
ELL	0.114217	0.618238	0.6287
OQ	0.121493	0	0.121493
UDP	0.11523	0.340426	0.359399
TCP	1.171937	0	1.171937

Table 1e: Average latency and loss ratios. Euclidean distance from origin.

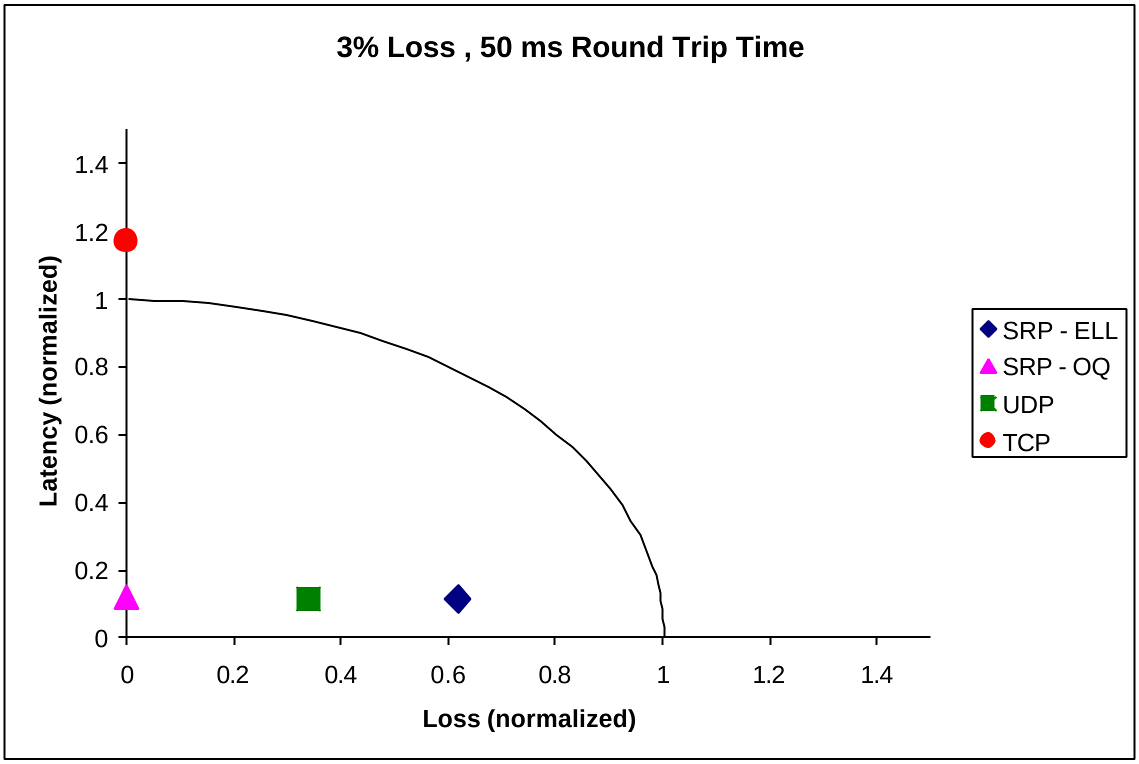


Figure 1e: Quality of all protocols under 3% network loss and 50ms round trip time

This test simulates relatively good network conditions. There is loss, but it is small, and the latency is low as well. From figure 1e, all protocols except TCP lie within the desired quality level. TCP is an outlier because exponential backoff throws off the latency calculation. Ignoring backoff, TCP performs very similarly to SRP – OQ. The exponential backoff makes TCP slow down and have slightly 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. This causes UDP loss to be higher.

SRP – ELL does not perform well in this test. The algorithm is designed to equalize loss and latency. In this case, an equal loss and latency is not optimal quality with these network conditions. In fact, the loss reported by SRP - ELL is higher than the actual loss of the network. This can occur because a very late packet can be detected as a loss. To equalize the loss and latency, the algorithm decides to give up on the message even though it was just late and hasn't been lost. It may seem that loss is high in comparison to latency and that the algorithm is not making them equal. This is not the case because loss and latency are weighted averages not totals. Only the last set (about 15 packets with the experimental smoothing factors) is effective in the average, allowing the average to adapt to network changes. The average loss soon becomes close to zero since most messages are received correctly. A loss of zero and a non-zero latency are not equal and the algorithm causes a loss to correct it.

5.2.2. Low loss/High latency

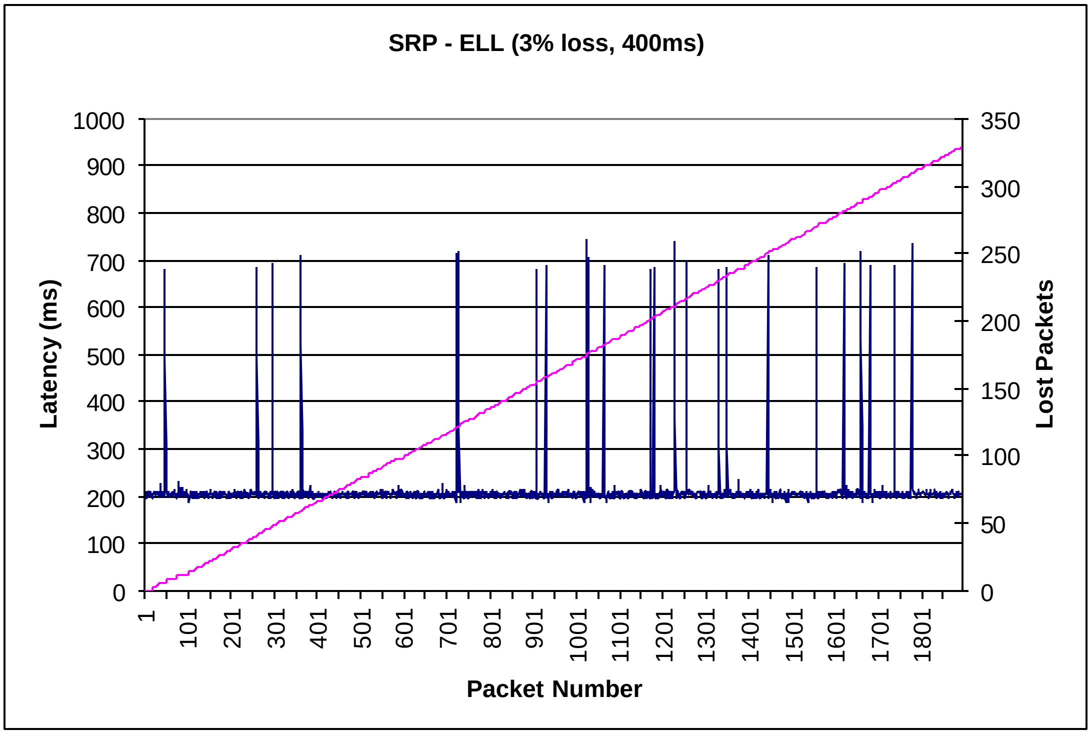


Figure 2a: Loss/Latency SRP ELL algorithm under 3% network loss and 400ms round trip time

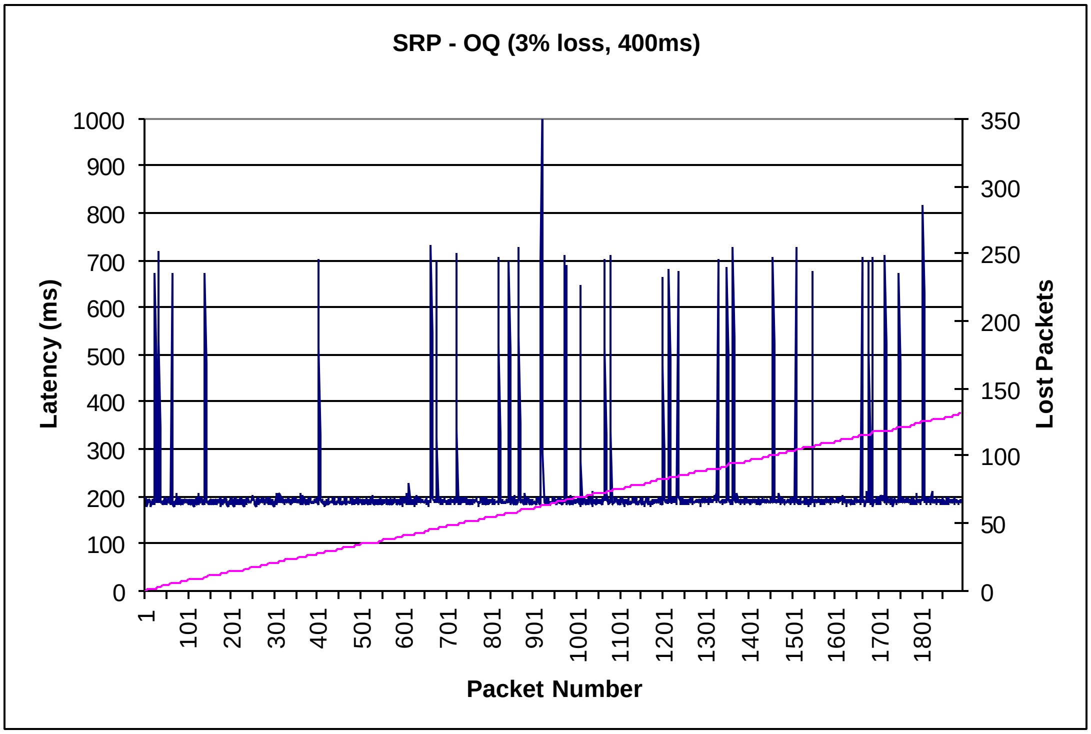


Figure 2b: Loss/Latency SRP OQ algorithm under 3% network loss and 400ms round trip time

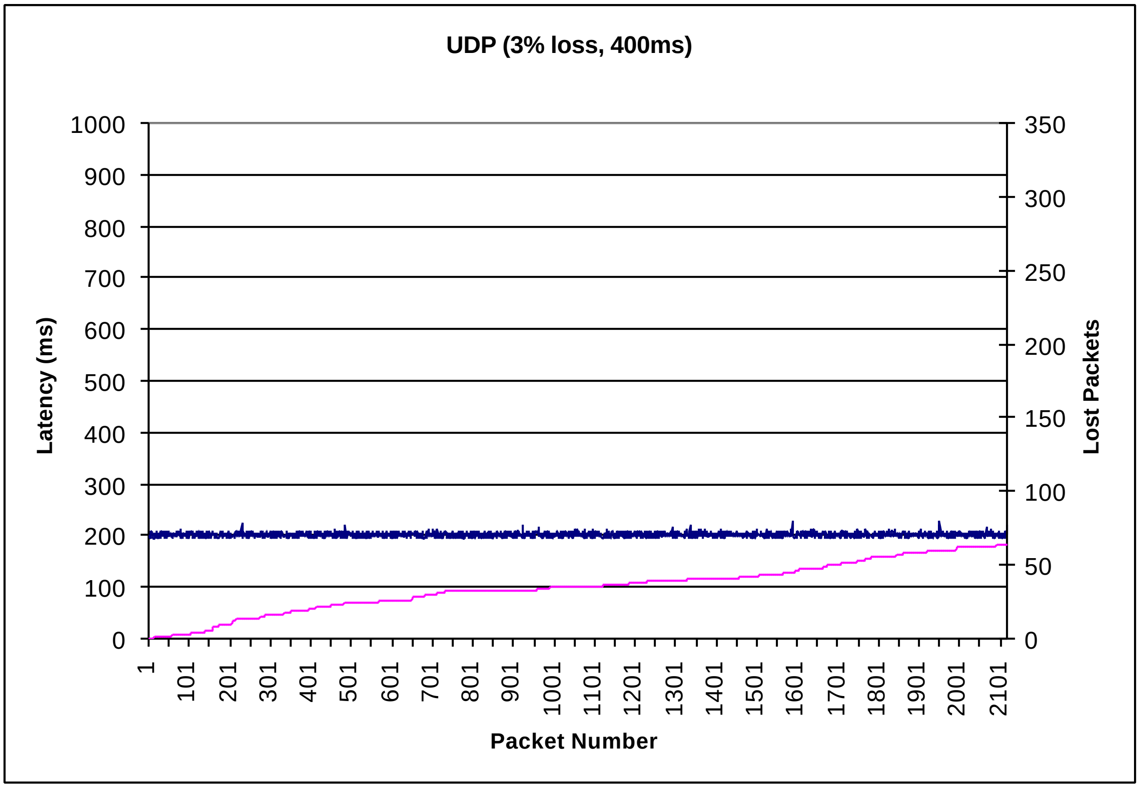


Figure 2c: Loss/Latency UDP under 3% network loss and 400ms round trip time

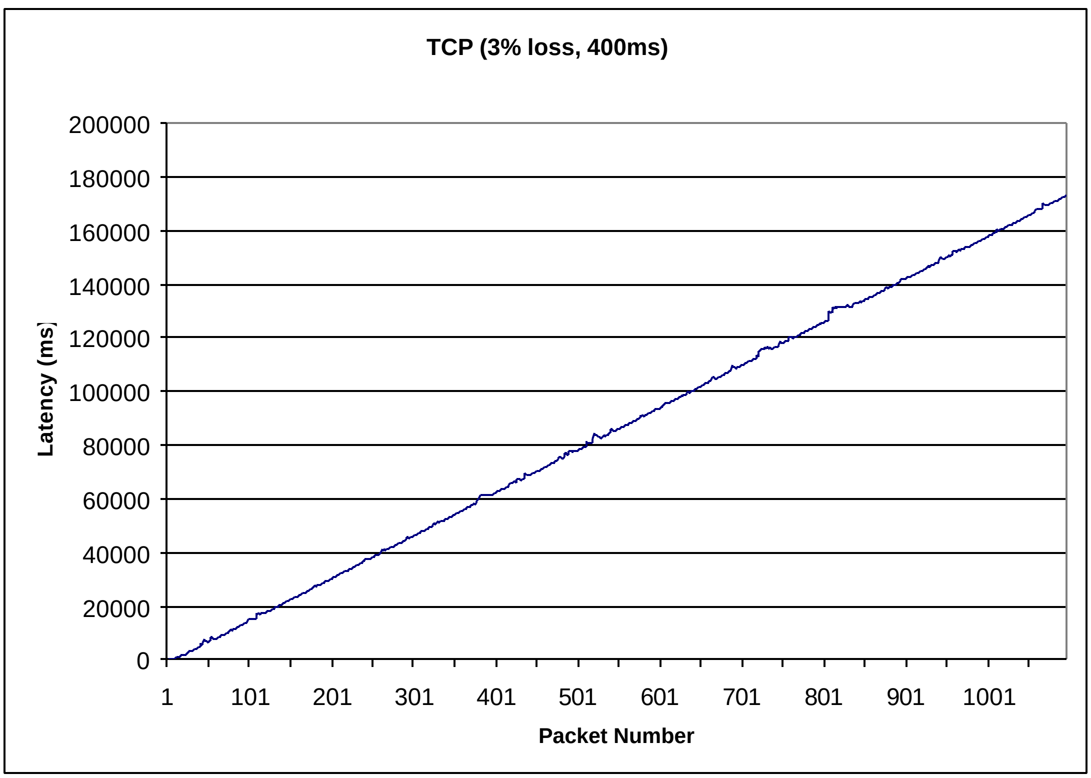


Figure 2d: Loss/Latency TCP under 3% network loss and 400ms round trip time

Protocol	Latency	Loss	Distance
ELL	0.863121	1.738901	1.941327
OQ	0.829688	0.692755	1.080876
UDP	0.805832	0.302315	0.860674
TCP	344.6209	0	344.6209

Table 2e: Average latency and loss ratios. Euclidean distance from origin.

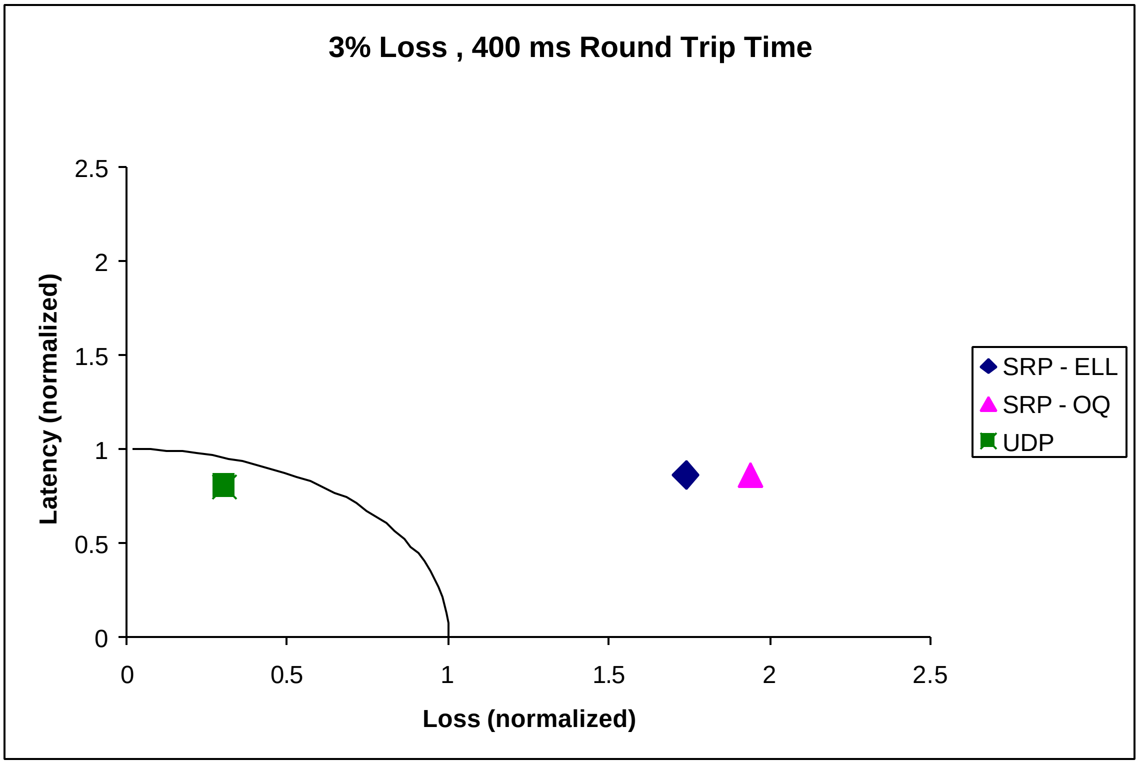


Figure 2e: Quality of all protocols under 3% network loss and 400ms round trip time

This test presents a network that has a high latency but little loss. Only UDP performs within the desired quality. TCP performs poorly in this test since there is a large penalty for retransmission and TCP must retransmit for every message lost. Exponetial backoff again plays a major role.

SRP – ELL and SRP – OQ both perform poorly because of the high penalty for retransmission. The decision algorithm will almost always give up on a lost message. Loss can then increase higher than network traffic since late messages can be detected as losses.

Since UDP never retransmits, it is not penalized by the high round trip time. Due to the lack of latency penalty and a low network loss, UDP performs relatively well.

5.2.3. High loss/High latency

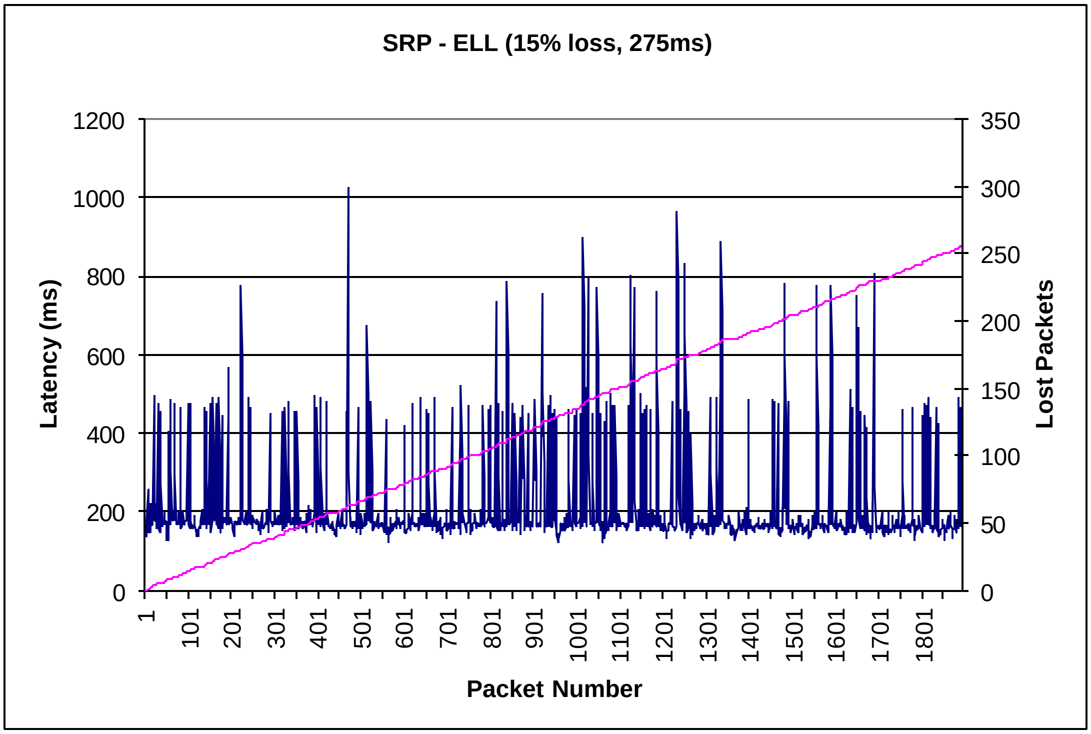


Figure 3a: Loss/Latency SRP ELL algorithm under 15% network loss and 275ms round trip time

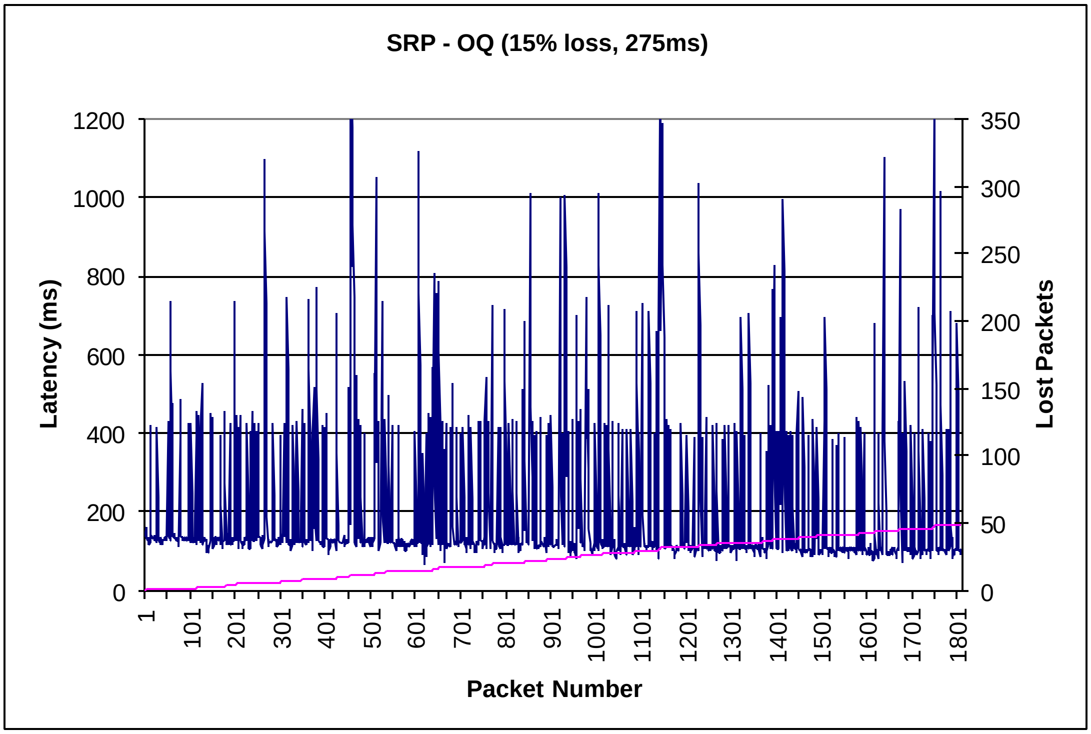


Figure 3b: Loss/Latency SRP OQ algorithm under 15% network loss and 275ms round trip time

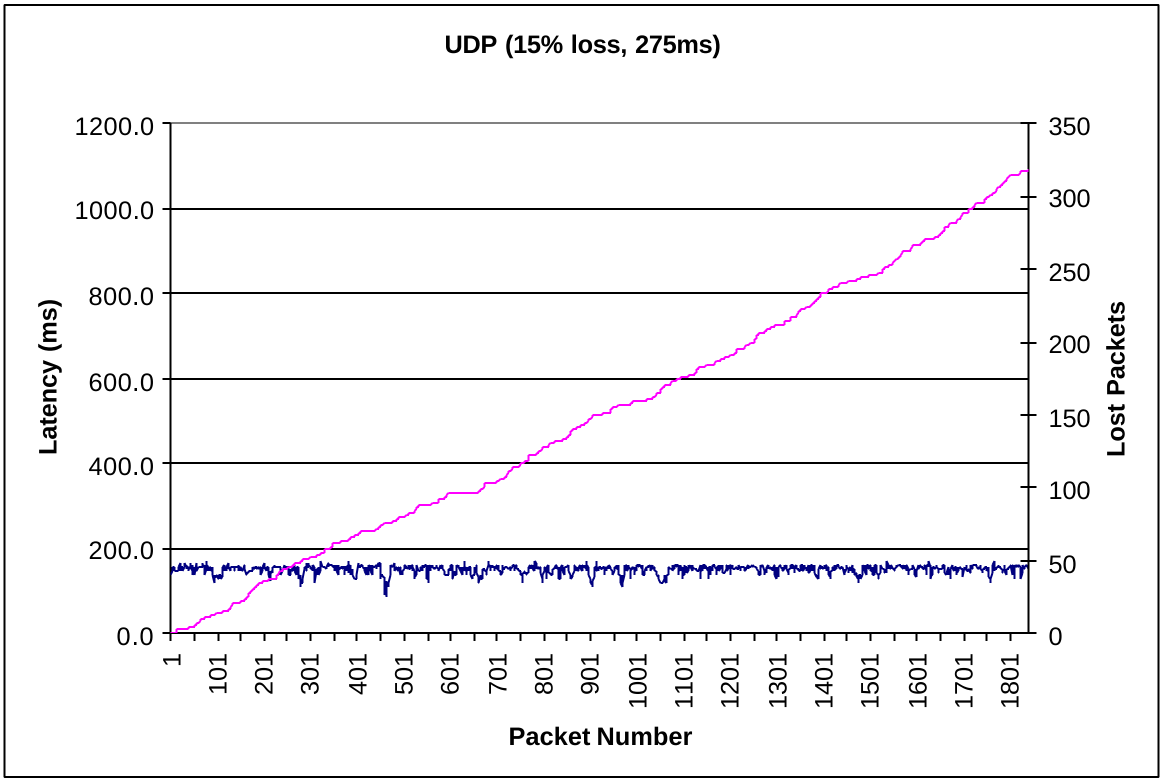


Figure 3c: Loss/Latency UDP under 15% network loss and 275ms round trip time

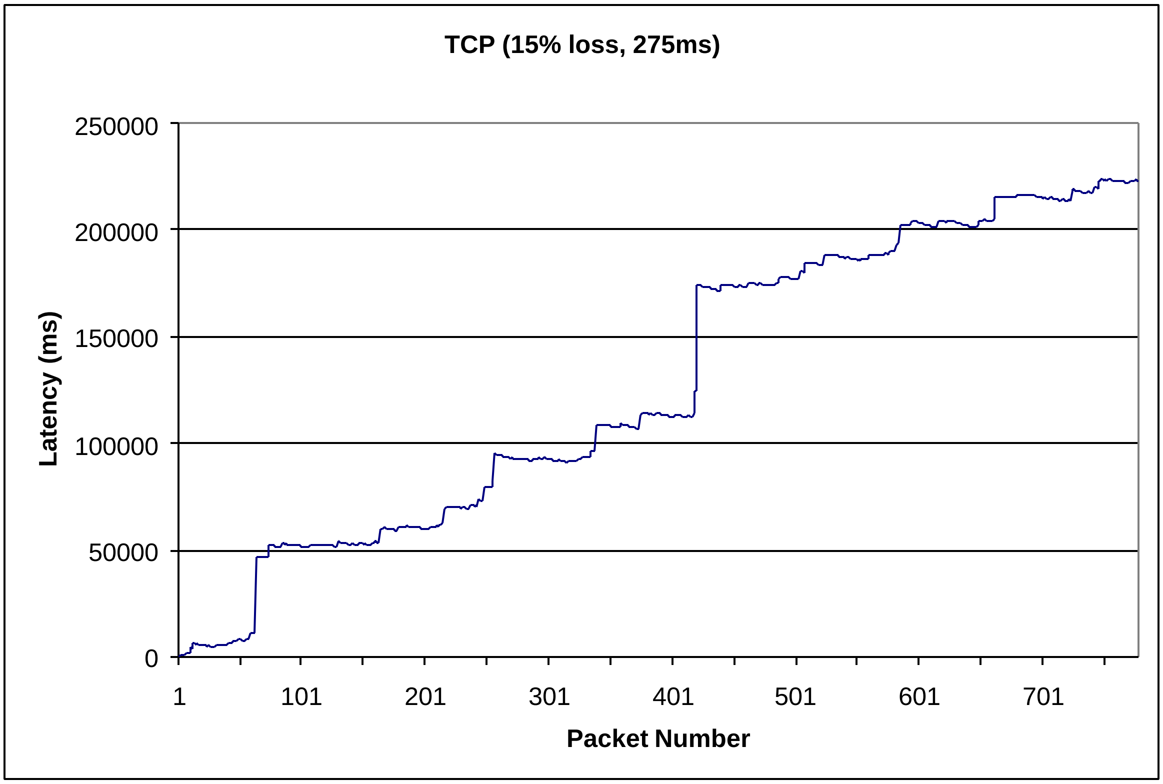


Figure 3d: Loss/Latency TCP under 15% network loss and 275ms round trip time

Protocol	Latency	Loss	Distance
ELL	0.848745	1.353781	1.59784
OQ	0.846382	0.270718	0.888623
UDP	0.605343	1.733696	1.836339
TCP	511.3582	0	511.3582

Table 3e: Average latency and loss ratios. Euclidean distance from origin.

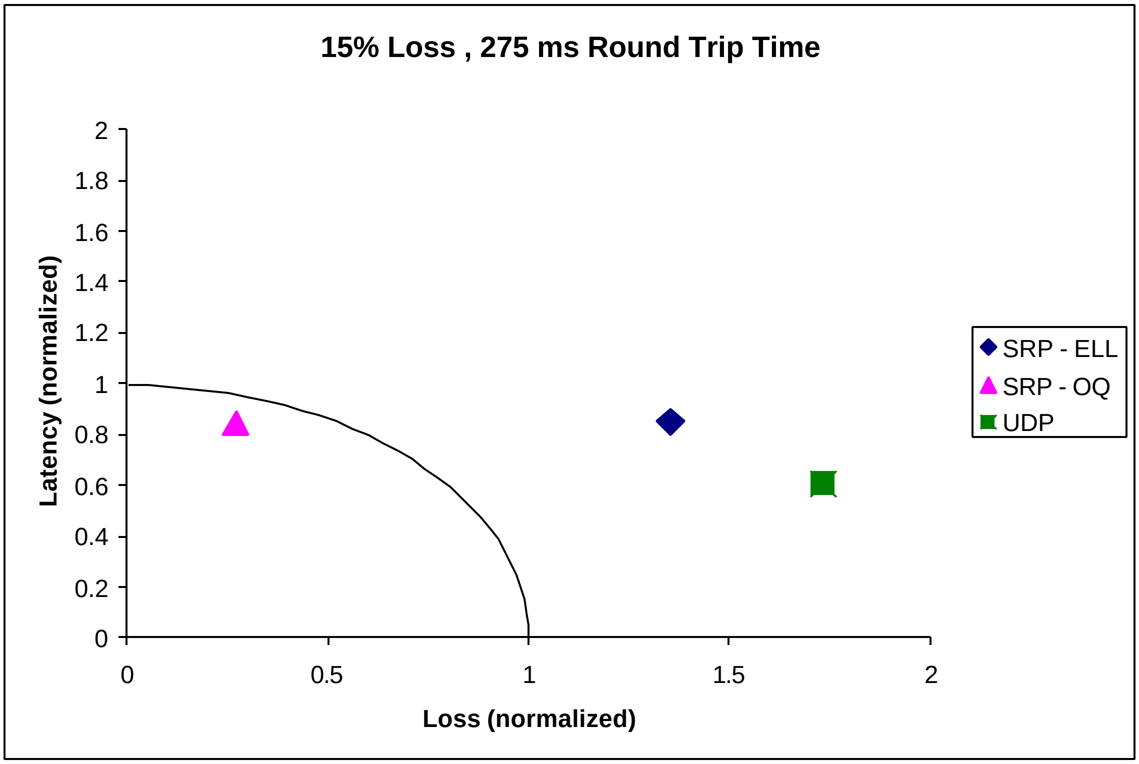


Figure 3e: Quality of all protocols under 15% network loss and 275ms round trip time

This test represents a high traffic network with high loss and high round trip time. SRP – OQ is the only protocol that performed within the desired quality. TCP, mostly due to exponential backoff, 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 exceed its quality.

6. Conclusion

Two networking protocols, TCP and UDP dominate the Internet. Both protocols have limitations when transmitting multimedia streams. TCP has high latency and no loss while UDP has high loss and low latency, neither of which is good for multimedia. We created a protocol that is better suited for these streams by combining the properties of both TCP and UDP. SRP provides lower latency than TCP and lower loss than UDP by retransmitting some, but not all lost packets.

SRP performed very well in our experiments and outperformed both TCP and UDP in respect to our quality comparison. It exceeded the quality of TCP and UDP in all network conditions except over a low loss, high latency network. This phenomenon is caused by the way SRP drops packets that don't arrive on time, which results in a higher loss rate than a non-retransmitting protocol such as UDP. Although SRP didn't perform as well as UDP in this instance, it did better than TCP in all network conditions.

Although SRP currently lacks some functionality such as back-off and flow control, it has been shown experimentally that it performs better than UDP and TCP in most circumstances. Even though SRP performed poorly in the low loss, high latency network, other retransmission decision algorithms can be written to deal with such conditions. SRP's flexibility and performance make it a desirable protocol for multimedia applications.

7. Future Work

7.1. Tuning SRP

While performing each test, several characteristics were observed about the SRP protocol. 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 a unnecessary retransmission (causing higher network traffic) or a loss even though the message will arrive.

It is possible for SRP to get behind if a very large chuck of messages is lost or is very late. Typically, this only occurs if the route from the server to the client is broken. 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 can fix this issue.

It is possible that the decision algorithm can always give up or always retransmit. Usually, this happens only if one of the above conditions occur. The reason this can occur is because there is a natural upper bound on loss (100%), however latency can increase indefinitely. 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 or caps on the amount of loss and latency reported to the decision algorithm can prevent this from occurring.

7.2. 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 a type of flow control. When TCP recognizes that traffic is high on the network, it automatically reduces its bandwidth usage in a method called exponential backoff. Exponential backoff 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.

One other advancement to the client would be a reset function that could be called when the expected data id is far greater or far smaller than the data ids being received. The function would resynchronize the client with the server by setting the expected data id to the id of the latest normal received message. This is important because messages that are too far ahead or behind the current stream are not useful.

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.

There are currently two main decision algorithms that SRP uses to determine whether to request a retransmission or take a loss based on the current network performance. SRP is adaptable in that other types of decision functions could easily be added into the system for testing and debugging. Other specific algorithms could be created depending on the network traffic, quality desired, or specific characteristics of a network such as large amounts of loss, delay or round trip time because of factors such as distance or technology utilized in the network.

8. Appendix

8.1. Client Interface Functions

8.1.1. srp_client_start

NAME

srp_client_start – Initializes the SRP client using a UDP socket.

SYNOPSIS

struct srp_client_socket *srp_client_start(int sock, unsigned int flags, const struct sockaddr *to, int tolen, float loss, int lat)

DESCRIPTION

The srp_client_start() function will initialize a SRP client to prepare for a multimedia stream. A standard UDP socket must be created and its socket descriptor must be passed as the sock parameter. The parameters flags, to, and tolen are identical to the parameters given by the standard function sendto and describe the network address of the SRP server. The parameter loss must be greater than zero and less than one and describes the desired loss percentage constraint of the network stream. The parameter lat given in milliseconds must be greater than zero and describes the desired latency constraint of the network stream.

RETURN VALUES

Upon successful completion, a pointer to a valid srp_client_sock structure will be returned which can be used to subsequent calls to other SRP functions. Otherwise a null pointer will be returned.

8.1.2. srp_client_stop

NAME

srp_client_stop – Stops the SRP client and cleans up memory used in internal buffers.

SYNOPSIS

void srp_client_stop(struct srp_client_socket *sock)

DESCRIPTION

The srp_client_stop() function will clean up the structures used by SRP. It will also remove the socket descriptor from use. The parameter sock is the pointer returned from srp_client_start().

RETURN VALUES

This function does not return any values.

8.1.3. srp_recvfrom

NAME

srp_recvfrom – Receives data from the SRP multimedia stream

SYNOPSIS

int srp_recvfrom(struct srp_client_socket *socketStr, void *buffer, int max_data_size)

DESCRIPTION

The srp_recvfrom() function permits an application program to get data from a SRP stream. The socketStr paramater must be created from srp_client_start(). The parameter buffer will be filled with the multimedia data from the stream. The max_data_size paramater describes the size of the buffer and the maximum amount of data to be read. If there is more data available, it will be truncated to max_data_size.

RETURN VALUES

Upon successful completion the function will return the actual number of bytes read. If the data was lost then it will return zero. if there was a socket error, EPIPE will be returned.

8.2. Server Interface Functions

8.2.1. srp_server_start

NAME

srp_server_start – Initializes the SRP server using a UDP socket.

SYNOPSIS

struct srp_server_socket *srp_server_start(int sock, unsigned int flags, const struct sockaddr *to, int tolen)

DESCRIPTION

The srp_server_start() function will initialize a SRP server to prepare to send a multimedia stream. A standard UDP socket must be created and its socket descriptor must be passed as the sock parameter. The parameters flags, to, and tolen are identical to the parameters given by the standard function sendto and describe the network address of the SRP client.

RETURN VALUES

Upon successful completion, a pointer to a valid srp_server_sock structure will be returned which can be used to subsequent calls to other SRP functions. Otherwise a null pointer will be returned.

8.2.2. srp_server_stop

NAME

srp_server_stop – Stops the SRP server and cleans up memory used in internal buffers.

SYNOPSIS

void srp_server_stop(struct srp_server_socket *sock)

DESCRIPTION

The srp_server_stop() function will clean up the structures used by SRP and stop any threads created by srp_server_start(). It will also remove the socket descriptor from use. The parameter sock is the pointer returned from srp_client_start().

RETURN VALUES

This function does not return any values.

8.2.3. srp_sendto

NAME

srp_sendto – Sends data to the SRP multimedia stream

SYNOPSIS

int srp_sendto(struct srp_server_socket *srp_sock, const void *msg, int len)

DESCRIPTION

The srp_sendto() function sends multimedia data to the SRP multimedia stream. The parameter srp_sock is a pointer to the srp_server_socket structure returned by srp_server_start(). The pointer msg must point to the data that will be sent across the multimedia stream and len is the size of the data pointed to by msg.

RETURN VALUES

Upon successful completion the function will return the actual number of bytes sent. Otherwise an error will be returned.

8.3. SRP Packet Definitions

SRP uses the data packet, nack packet, and probe packet to transmit data on the network. The data packet has a maximum size of 1496. The standard header information is 16 bytes long and the rest is reserved for the data. The nack packet has a size of 16 bytes, and the probe packet has a size of 8.

SRP data packet

Type	time info	data id	data size	Data…
data…				

SRP NACK packet

Type	time info	status	data id	NOT USED IN SRP NACK PACKET

SRP time probe packet

Type	time info	NOT USED IN SRP PROBE PACKET

9. Bibliography

[BS99] A. Bouch and A. Sasse. Network Quality of Service: What do users need? University College London. September 1999.

[CC97] A.T. Campbell and G. Coulson. A QOS Adaptive Multimedia Transport System: Design, Implementation and Experiences. Distributed Systems Engineering Journal, Special Issue on Quality of Service, Vol. 4, pg. 48-58, April 1997.

[CPW98] S. Cen, C. Pu, and J. Walpole. Flow and Congestion Control for Internet Media Streaming Applications. Oregon Graduate Institute of Science and Technology 1998.

[C98] K. Cho. A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers. Sony Computer Science Laboratory, Inc. June 1998.

[CR96] M. Claypool and J. Riedl. A Quality Planning Model for Distributed Multimedia in the Virtual Cockpit. University of Minnesota, 1996.

[FF96] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. Lawrence Berkeley National Laboratory. 1996.

[G97] R. Goyal. Selective Acknowledgements and UBR+ Drop Policies to Improve TCP/UBR Performance over Terrestrial and Satellite Networks. Ohio State University, April 1997.

[KR99] J. Kurose and K. Ross. Computer Networking: A Top-Down Approach Featuring the Internet. Addison-Wesley, 1999.

[KVC99] G. Kannan, B. Venkarachari, and M. Claypool. SRP – A Selective Retransmission Protocol for Quality of Service for Multimedia on the Internet. Worcester Polytechnic Institute 1999.

[MMM96] M. Mathis, J. Mahdavi and J. Mahdavi. Forward Acknowledgement: Refining TCP Congestion Control. Pittsburgh Supercomputing Center. August 1996.

[MMR96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC 2018, October 1996.

[PJSB98] M. Parris, K. Jeffay, F. Smith, and J. Borgersen. A Better-Than-Best-Effort Service for Continuous Media UDP Flows. University of North Carolina at Chapel Hill, July 1998.

[PJS99] M. Parris, K. Jeffay, and F. Smith. Lightweight Active Router-Queue Management for Multimedia Networking. University of North Caroline at Chapel Hill, January 1999.

[PHH98] C. Perkins, O. Hodson, and V. Hardman. A Survey of Packet-Loss Recovery Techniques for Streaming Audio. University College London, August 1998.
[Sc98] H. Schulzrinne. RTP: About RTP and the Audio-Video Transport Working Group. http://www.cs.columbia.edu/~hgs/rtp, 1998.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, January 1996.

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326, April 1998.

[S98] R. Stevens. UNIX Network Programming. Prentice-Hall Inc. Upper Saddle River, NJ, 1998.

[WS97] A. Watson and M. Sasse. Multimedia Conferencing via Multicast: Determining the Quality of Service Required by the End User. University College London, September 1997.