Prototyping the Selectively Reliable Transport Protocol

J. Mark Pullen Vincent P. Laviano
Department of Computer Science and Center for Excellence in
Command, Control, Communications and Intelligence
George Mason University

Keywords: Collision, Multicast, Reliable Service, Transport Layer (Layer 4)

ABSTRACT

Real-time multicast applications such as Distributed Interactive Simulation cannot take advantage of fully-reliable transport for all data and still achieve real-time transmission. However, DIS applications require the reliable real-time transmission of some data in a multicast environment. Since no suitable transport protocol exists today, DIS applications must compromise and use the unreliable User Datagram Protocol (UDP) in order to achieve real-time transmission. The use of an unreliable transport protocol forces these applications to perform tasks, such as error detection and recovery, that more properly belong to the network in order to ensure the reliable transmission of protocol data units (PDUs) when necessary. This increases the complexity of DIS applications, making them more costly to design, implement and maintain.

In a paper presented at the 13th DIS Workshop [PuLa95], we proposed a Selectively Reliable Transport Protocol (SRTP) which operates in three distinct modes of reliability: transmission of transient data that does not need reliable transport; transmission of reference entity-state PDUs that must be received reliably; and timely, reliable transport of data to a single, dynamically determined receiver within the multicast group. We believe that such a standardized protocol would remove an undue burden from DIS application programmers and increase interoperability among DIS applications.

We now discuss the design and implementation of a SRTP prototype, coded in ANSI C, which operates in UNIX user space and utilizes UDP and IP Multicast. SRTP is implemented via asynchronous sending and receiving processes that a DIS application communicates with through a functional API.

1. INTRODUCTION

DIS applications sometimes require the reliable, real-time transmission of data in a multicast environment. Yet, commonly available transport protocols do not meet these needs. A fully-reliable protocol such as the Transmission Control Protocol (TCP) is not sufficient due to connection overhead and acknowledgement delay. In addition, neither positive nor negative acknowledgements for all data scale well in a multicast environment. An unreliable protocol such as UDP, while it may achieve real-time transmission, clearly is not sufficient because it does not provide reliable service. Recent work on log-based receiver-reliable multicast with a variable heartbeat scheme has made the reliability of multicast higher for real-time applications such as DIS[HSC95], but it is still the case that any fully reliable protocol will not operate in real-time.

This situation forces DIS applications to compromise and use an unreliable transport protocol such as UDP in order to achieve real-time transmission. The responsibility for reliability then rests at the application layer of the protocol stack, which increases the complexity of DIS applications and hinders interoperability among different DIS applications.

2. PREVIOUS AND RELATED WORK

Previously[PuLa95], we proposed and described a Selectively Reliable Transport Protocol (SRTP) which would perform these tasks in a standard way, contributing toward the DIS Steering Committee's stated goal of standardizing DIS protocols in order to increase interoperability among DIS applications. We utilized a simulated network environment during our initial design process.

In this paper, we refine our initial design and describe our experiences building a real-world prototype of SRTP which operates in a UNIX environment, forming a sublayer between DIS applications and UDP/IP-Multicast.

Recently, a variety of multicast protocols have been developed, each with different characteristics. The Reliable Adaptive Multicast Protocol (RAMP), is perhaps the closest in nature to SRTP, because it "allows a multicast service to be provided at multiple qualities of service, with all or some of these levels transmitted reliably"[BrZa93]. However, RAMP differs from SRTP in several ways. SRTP embeds rare unicast transmissions to individual hosts in the multicast stream using a positive acknowledgement error recovery scheme, while RAMP uses a strictly negative acknowledgement scheme. In SRTP, NAKs are multcast to all group members, while in RAMP, NAKs are unicast back to the original sender. SRTP multicasts all retransmissions while RAMP dynamically determines whether to unicast or multicast retransmissions based on the number of NAKs received.

3. DESIGN OVERVIEW

SRTP operates in three distinct transmission modes in order to deliver varying levels of reliability that correspond to the needs of individual PDUs:

- Mode 0 for data that does not require reliable transmission,
- Mode 1 for data that must be received reliably by all members of a multicast group, and
- Mode 2 for data that must be received reliably by a single dynamically- determined member of a multicast group.

SRTP operates under the expectation that only a small percentage of PDUs generated by a DIS application require reliable transport. SRTP may be equally useful to other multicast applications that share this characteristic.

SRTP allows applications to label messages with a Category Code (a unique integer in the range 1 to 1023, with 0 being reserved for special use by SRTP) in order to differentiate them from other messages using the same transmission mode. This feature allows applications to register interest in specific categories of data with SRTP and receive only those categories. Category Codes are also intended to be used when there is a need to associate PDUs using different transmission modes in order to represent a compressed stream of data which consists of both reference and offset PDUs.

SRTP also allows applications to dyanmically register and unregister interest in specific multicast groups. This supports applications where membership in different multicast groups is being used as a relevance filtering mechanism.

4. SRTP PROTOCOL

A full description of the SRTP protocol is given in our previous paper[PuLa95]. However, we outline the header and the basic mechanism here.

SRTP Header

As shown in Figure 1, the SRTP header is twelve bytes long and consists of nine fields. The first six fields of the header (having a total length of four bytes) occur in all PDUs. The seventh field, Dest_Addr (which is itself four bytes long), occurs only in Mode 2 PDUs. The final two fields, FN and FSN, only occur in fragmented PDUs (which are not supported in this initial prototype). The purpose of each of the header fields is summarized in Figure 1.

SRTP Mechanism

SRTP differentiates among several types of data based upon the values of Mode and Category Code that are given to it along with a message. Mode 0, category 0 messages are independent of all mode 1 data and are automatically received by all members of the relevant multicast group. All other mode 0 messages are viewed as incremental and frequent updates to data streams with occasional reference PDUs (mode 1 data of the same category) that need to be transmitted reliably. Mode 0 data is transmitted unreliably. Mode 0 PDUs of a category other than 0 carry the same sequence number as the most recently sent Mode 1 PDU of the same category by the same sender. Mode 1 PDUs are numbered sequentially at each sender. Mode 1 PDUs are sent reliably using NAK-based error recovery. When a receiver detects that a Mode 1 PDU has been missed (by receipt of a Mode 0 PDU with an inappropriate sequence number), a NAK is multicast to all members of the multicast group provided that no other NAK for same PDU has been received by the receiver requesting retransmission within a user-specified time interval. The original sender retransmits the PDU in question by multicasting it to all members of the group.

Mode 2 data is viewed as infrequent reliable point-to-point communication between a pair of receivers. Mode 2 PDUs are numbered sequentially at each receiver. Mode 2 PDUs are transmitted reliably using ACK-based error recovery. Upon timeout without receipt of an ACK, the sender retransmits the PDU via multicast.

Unlike Mode 1 PDUs, each new Mode 2 PDU of the same category does not invalidate the previous one, so only a limited number of Mode 2 PDUs of the same category may be accepted by the SRTP sender within a single time interval. This number is based on the available memory and is user-specified. The appropriate value of this number will require further investigation.

Header Field	Bit Position	Length (in bits)	Description
Type	0-1	2	Type of PDU (DATA, ACK, or NAK)
Mode	2-3	2	SRTP Transmission Mode (0, 1, or 2)
Cat	4-13	10	SRTP Category Code (0-1023)
F_Flag	14	1	Indicates fragmented PDU
B_Flag	15	1	Indicates bundled PDUs
SN	16-31	16	Sequence Number
Dest_Addr	32-63	32	(Mode 2 Only) IP address of intended recipient
FN	64-71	8	(Fragments Only) Fragment Number
FSN	72-95	24	(Fragments Only) Sequence Number of fragmented PDU

Figure 1: SRTP Header

Although Mode 2 tranmission applies only to rare events such as collision, it may happen that there is a long period of no Mode 2 traffic, followed by several Mode 2 PDUs (i.e. combat), followed by another relatively long period of Mode 2 inactivity.

5. DESIGN CHOICES

SRTP was initially designed in a closed simulation environment on a single workstation. The design of SRTP has evolved considerably during the process of prototyping it in a real-world UNIX environment with multiple heterogeneous workstations.

Portability has been a concern. It was this concern which motivated us not to use threads to implement the asynchronous operation of the sender and receiver. However, this remains an option in the future. It was concern for portability which also lead us to choose the sockets interface to UDP which is available in both BSD and System V style environments.

We experimented using shared memory to pass information quickly between the SRTP clients and servers, but found that the shared segments were either too large or too many. Therefore, we chose to use message passing for interprocess communication.

6. SRTP INTERFACE

We designed SRTP to provide access to its services through an Application Program Interface (API) that is made available as a library archive file, *libsrtp.a*, to be linked with a DIS application at compilation time. The API consists of eight functions: *srtp_init, srtp_term, srtp_reg_cat, srtp_unreg_cat, srtp_reg_mcg, srtp_unreg_mcg, srtp_send, and srtp_recv*. Prototypes for these functions are available in an include file, *srtp.h*, that is to be included by DIS applications wishing to utilize SRTP functions. A brief description of each function follows.

Function Name: *srtp_init*
Purpose:
Register a new client program with a host's SRTP Receiver. Registration is only required by those clients wishing to receive data via SRTP. Any process may send data using SRTP without having to register. Upon initial registration, a client automatically begins to receive all PDUs with Category Code 0.
Calling Sequence:
int status = srtp_init(mc_addr)
Parameters:
*const char *mc_addr*: IP multicast address to join

Function Name: *srtp_term*
Purpose:
Unregister a client program with a host's SRTP Receiver. Upon the departure of a client, SRTP will transmit all previously sent Mode 1 and Mode 2 data reliably (up to the specified maximum number of retransmissions). However, a client will not be able to receive any more data after unregistering. Calling *srtp_term* implies consent to unregister interest in all Category Codes.
Calling Sequence:
int status = srtp_term()
Parameters: None

Function Name: *srtp_reg_cat*
Purpose:
Register interest in a certain Category Code. After calling *srtp_reg_cat*, a client will be able to receive all subsequent PDUs marked with that Category Code.
Calling Sequence:
int status = srtp_reg_cat(cat)
Parameters:

unsigned int cat: Category Code to register

Function Name: *srtp_unreg_cat*
Purpose:
Unregister interest in a certain Category Code. After calling *srtp_unreg_cat*, a client will not receive new PDUs marked with that Category Code.
Calling Sequence:
int status = srtp_unreg_cat(cat)
Parameters:
unsigned int cat: Category Code to unregister

Function Name: *srtp_reg_mcg*
Purpose:
Register interest in an additional multicast group. After calling *srtp_reg_mcg*, a client will receive all PDUs addressed to the new multicast group, provided that the client has registered interest in the Category Code that the PDU is marked with.
Calling Sequence:
int status = srtp_reg_mcg(mc_addr)
Parameters:
*const char *mc_addr*: Multicast address to register

Function Name: *srtp_unreg_mcg*
Purpose:
Unregister interest in a multicast group. After calling *srtp_unreg_mcg*, a client will no longer receive new PDUs addressed to the multicast group. A client is required to be registered for at least one multicast group at all times. Therefore a client is not permitted to unregister a multicast group if the client is only subscribed to a single group at the time.
Calling Sequence:
int status = srtp_unreg_mcg(mc_addr)
Parameters:
*const char *mc_addr*: Multicast address to unregister

Function Name: *srtp_send*
Purpose:
Send *length*-byte message stored in *msg* to SRTP sender for eventual transmission. Client indicates options by filling in fields of *opts* structure. This function does not block while SRTP sender prepares SRTP packet for delivery.
Calling Sequence:
int status = srtp_send(msg,length,&opts)
Parameters:
*const char *msg*: Data to be sent
int length: Length of data in bytes
*struct sendops *opts*: Pointer to structure containing options such as Mode, Category Code, multicast address, multicast time-to-live (TTL), and Dest_Addr (in the case of Mode 2 PDUs)

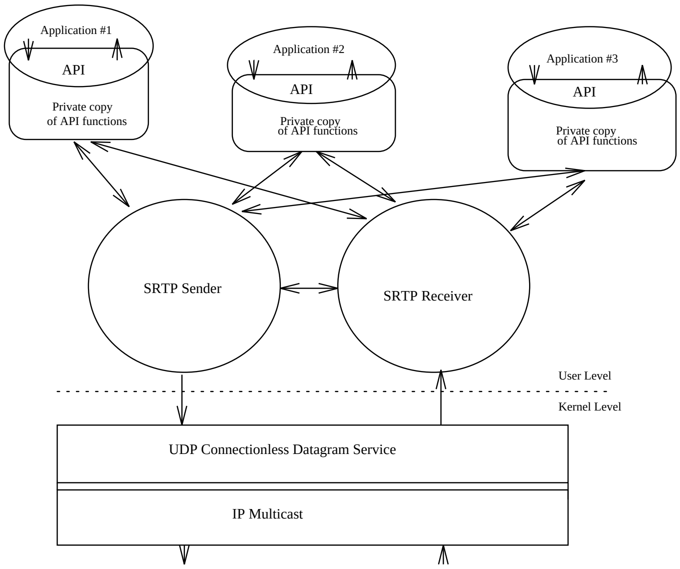

Figure 2: SRTP Software Architecture

Function Name: *srtp_recv*
Purpose:
Receive a message from SRTP receiver into *buf*. This function is non-blocking. If a message is available, *srtp_recv* returns the number of bytes received. Otherwise, -1 is returned.
Calling Sequence:
int nbytes = srtp_recv(buf)
Parameters:
*const char *buf*: Buffer for message

7. SRTP ARCHITECTURE

The SRTP software architecture (depicted in Figure 2) consists of two asynchronous servers, *srtpd* and *srtprd*, that reside at the user level on each SRTP-capable host and provide their services to a number of client programs (e.g. DIS applications). The servers, in turn, use the kernel level services of UDP and IP Multicast. It is assumed that only a single pair of servers will reside on each host, and that all of a server's clients will be local to the server's host. Based on these assumptions, *srtpd* and *srtprd* both have well-known UNIX domain addresses.

Srtpd is responsible for accepting data from clients and performing the functions of the SRTP sender described above. *Srtprd* is responsible for accepting registration requests from clients, performing the functions of the SRTP receiver as described above, and delivering messages to clients.

To initialize SRTP, it is necessary to invoke *srtpd* with certain arguments such as the SRTP port number and certain user-defined constants as described above. *Srtpd* itself invokes *srtprd*. As soon as both servers are up and running, it is possible for application programs to make use of their services.

In order to access SRTP services, applications must *#include* the *srtp.h* header file in their source code. In addition, the API functions must be made available to the application program by linking in the *libsrtp.a* library at compilation time. Each application then has a private copy of the API functions in its own address space which may be invoked as needed. Implementation details are hidden from the DIS application programmer. Access to SRTP services is through the API functions.

When DIS applications make calls to API functions, it is necessary for messages to be passed to *srtpd* or *srtprd*. Clients communicate with the SRTP servers using UNIX domain sockets. A request-reply protocol is used between clients and servers. In addition, it is sometimes necessary for the two servers on a given host to exchange data. This is also done using UNIX domain sockets. However, SRTP servers communicate with remote servers using Internet domain sockets.

8. FUTURE WORK

We intend to make our SRTP prototype available via anonymous ftp in order to allow the DIS community to use and comment on it.

We intend to test the performance of our prototype under a variety of conditions in order to confirm the scalability of our approach.

Recently products such as StarBurst MFTP have become available to take advantage of multicast for one-to-many file transfer[Star95]. This capability is required in DIS systems (e.g. distribution of terrain databases or mission plans). We intend to investigate possible additions to SRTP to facilitate such services.

References

[BrZa93] Braudes, R. and S. Zabele, "Requirements for Multicast Protocols", RFC 1458, May 1993.

[DIS94] Dis Steering Committee, The DIS Vision: A Map to the Future of Distributed Simulation, May 1994.

[HSC95] Holbrook, H. W., S. K. Singhal and D. R. Cheriton, Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation, *Proceedings of SIGCOMM '95*, August 1995.

[PuLa95] Pullen, J. M. and V. P. Laviano, A Selectively Reliable Transport Protocol for Distributed Interactive Simulation, *13th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, September 1995.

[Star95] StarBurst Communications Corporation, *StarBurst MFTP Compared to Today's File Transfer Protocols: A White Paper*, 1995.