ENSC 835-3: NETWORK PROTOCOLS AND PERFORMANCE

Transportation of a Real-Time Transport Protocol Packet Stream Over an ATM Adaptation Layer 5 Backplane

April 2002

FINAL PROJECT REPORT

Kevin Ko (kkoa@sfu.ca)

Naomi Ko (nko@sfu.ca)

www.sfu.ca/~kkoa/Ensc835/

Abstract

As technology advances, the demand for new services like real-time applications has increased dramatically, creating a constant push-and-pull effect between real-time applications and higher bandwidth [12]. As transmission of large amounts of traffic increases [13], so too does the need to use available bandwidth more efficiently.

Real-time Transport Protocol (RTP) provides a mechanism for sending real-time data such as video and multimedia. Compressing the RTP data packets and coupling the result with Asynchronous Transfer Mode (ATM) technology provides a means to deliver real-time application data over a network.

This project is focused on modeling the compression of RTP/UDP/IP packets and their transmission over an ATM network, based on a paper "Encapsulation of Real-Time Data Including RTP Streams over ATM" by AT&T Labs [6]. The resulting OPNET model generates user-defined RTP/UDP/IP packets, performs header compression as described in RFC 1889 [3], and modifies the ATM encapsulation to compensate for omitted capabilities and to provide additional information.

Successful modeling of the RTP component and basic ATM functionality (modified from existing OPNET models) provide a good basis on which a complete implementation of the paper's proposed work can be achieved with a few additional enhancements.

Table Of Contents

List of Figures

List of Tables

1. Abbreviations

Abbreviation	Definition
AAL	ATM Adaptation Layer
AAL5	ATM Adaptation Layer Type 5
ATM	Asynchronous Transfer Mode
CID	(Session) Context Identification
CPCS	Common Part Convergence Sublayer
CRC	Cyclic Redundancy Check
IHL	IP Header Length
IP	Internet Protocol
IPv4	IP version 4
ITU	International Telecommunications Union
MBZ	Must Be Zero
MSB	Most Significant Bit
PDU	Protocol Data Unit
PPP	Point-to-Point Protocol
RTP	Real-Time Transport Protocol
SAP	Service Access Point
SAR	Segmentation and Reassembly
SDU	Service Data Unit
SEAL	Simple and Efficient Adaptation Layer
SIP	Session Initiation Protocol
SSRC	Synchronization Source
TOS	Type Of Service
TTL	Time To Live
UNI	User-to-Network Interface
VC	Virtual Channel
VCI	Virtual Channel Identifier
VPI	Virtual Path Identifier
VoIP	Voice-over-IP
UDP	User Datagram Protocol

2. Introduction

Prompted by an age of multimedia, considerable efforts have been expended on researching the ability to send audio and video real-time media efficiently over different types of networks. The concepts of sending real-time media over IP with RTP and streaming such media over ATM are far from new; however the implications of these technologies are important when considering the increased emergence of real-time audio and video applications. Large amounts of real-time traffic, as well as other bandwidth-consuming traffic such as file transfers demand the investigation of bandwidth conservation and efficiency. Through certain techniques, RTP packets sent over UDP/IP can be compressed to conserve a considerable amount of bandwidth. These compressed packets can be encapsulated and transported to the destination over a network such as ATM.

2.1 Background Material

Real-Time Transfer Protocol and Asynchronous Transfer Mode will be explored in detail, as they are the underlying standards and protocols in our project. This section describes the basics of each of these standards to provide the reader with a bit of technical understanding before proceeding.

2.1.1 Real-Time Transport Protocol

Real-Time Transport Protocol (RTP) is an Internet standard used for conveying real-time media streams between interactive participants, and is specified in RFC 1889. This protocol typically runs end-to-end on top of User Datagram Protocol (UDP) over Internet Protocol (IP), and has received a significant amount of industry support [14]. RTP neither addresses the reservation of resources, nor does it guarantee quality-of-service and timely delivery [1].

Applications using Real-Time Transport Protocol include Voice-over-IP (VoIP) telephony, multimedia conferencing, which includes audio, video, and data streaming, and video and audio mixers and translators (see Glossary).

RTP packets are sent within an RTP session, defined as an association among a set of participants communicating with RTP. These associations may be set-up through various protocols (see Section 4.3.1), and can consist of multiple session contexts, which are defined uniquely by the source and destination IP addresses, source and destination UDP ports, and RTP synchronization source (SSRC) [3].

2.1.1.1 RTP Packet Format

The packet format for RTP is illustrated in Figure 1, where V is the RTP version number, P is the padding flag, X is an extension bit, CC is the number of contribution sources, and M is a first and last packet marker.

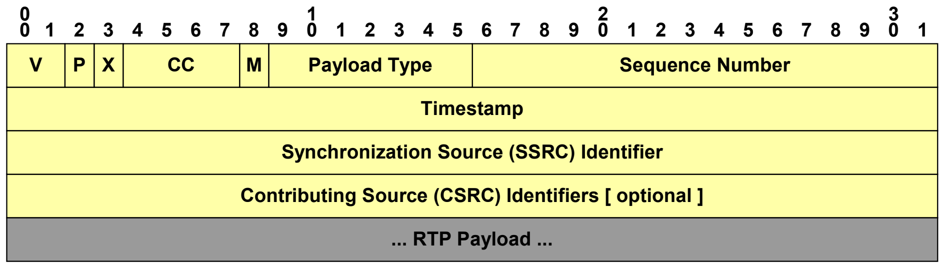

Figure 1: RTP Packet Format

For more detailed information about the RTP header fields and their use, refer to RFC 1889 [1].

A technique for compressing the RTP, UDP, and IP headers into a single header has been devised, removing the transfer of extraneous and repetitive header information. While the motivation to compress RTP packets was spawned from the desire to send audio and video over the low-speed connections of 14.4 and 28.8 kbps modems [3], it is universally applicable to conserve bandwidth over any network, particularly when RTP payloads are small. This compression technique is capable, in many instances, of compressing a 40-byte RTP/UDP/IP packet header into a 2-byte packet header. The technique is extracted from RFC 2508 [3] and will be explained in Section 3.1 and implemented in an OPNET model in Section 4.1.3.

2.1.2 Asynchronous Transfer Mode (ATM)

Asynchronous Transfer Mode (ATM) is a widely-deployed network technology, standardized by the International Telecommunications Union- Telecommunication Standardization Sector (ITU-T), described in Recommendation I.361 [9]. Its cell relay technology is used for the high-speed communication for the transmission of voice, video, data and images.

2.1.2.1 ATM Adaptation Layer Type 5

The ATM adaptation layer (AAL) enhances the service provided by ATM to support functions required by the next higher layer. Among the functions provided by the AAL are mapping between ATM and higher layers, and segmentation of data into 48-byte frames.

Type 5 AAL (AAL5), described in ITU-T Recommendation I.363.5 [8], is the most common AAL used for data and supports both connection-oriented and connectionless data [4]. AAL5 is also known as the simple and efficient adaptation layer (SEAL) since very little overhead is added to the user data. This type of AAL supports the non-assured transmission of user data frames: it assumes that higher layers will provide error recovery.

2.1.2.2 Framework of AAL5

User data is passed from higher (application) layers to the ATM adaptation layer in units of frames or AAL service data units (SDUs). Between the AAL and the ATM service access points (SAPs), the data frame passes through a number of sublayers that perform various operations.

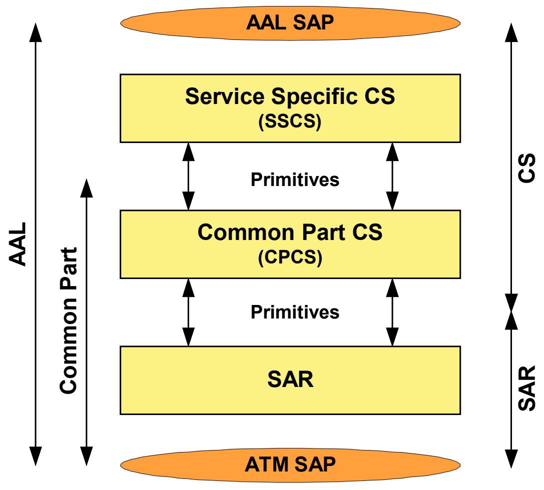

Figure 2: Structure of Type 5 AAL

As seen in Figure 2, the AAL5 framework breaks down into a convergence sublayer (CS) and segmentation and reassembly sublayer (SAR). The convergence sublayer further divides into a service-specific convergence sublayer (SSCS) and a common part convergence sublayer (CPCS). Each layer is responsible for certain functions: SSCS protocols will support specific AAL user services if desired; the CPCS appends a trailer to the user data; and the SAR sublayer separates the data and CPCS trailer into 48-byte cells, ready for ATM encapsulation.

Defining the SSCS protocol allows different AAL user services to be supported. Without any definition, the SSCS simply maps the AAL-SDU to the CPCS-SDU and vice versa.

As suggested by its name, the CPCS is common to AAL5 implementation, regardless of what SSCS protocols may be implemented. During the encapsulation process, the CPCS appends 2 fields: a variable-length padding field (PAD) and an 8-byte trailer to form the CPCS protocol data unit (PDU) shown in Figure 3. During decapsulation, the CPCS-PDU is stripped of the trailer and padding.

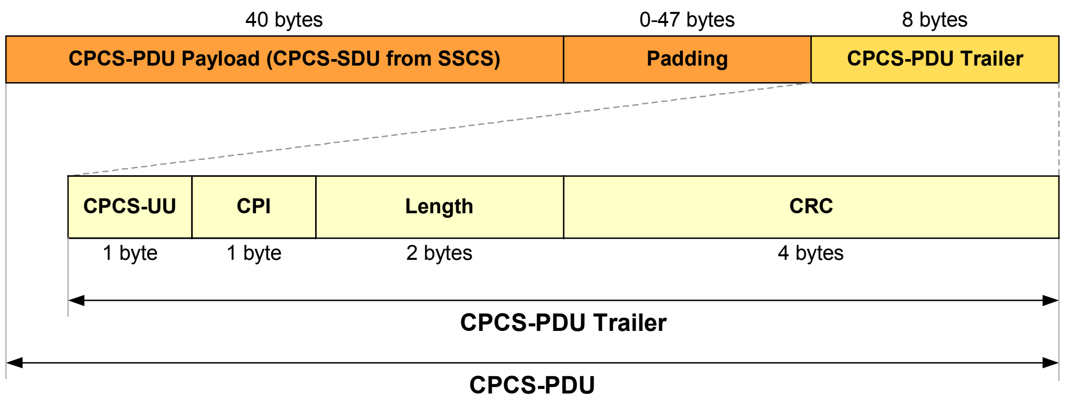

Figure 3: CPCS-PDU Format for AAL5

Like the CPCS, the SAR sublayer is also common to the AAL, regardless of the higher layer applications. This sublayer's concern is segmenting the SAR-SDU (the CPCS-PDU) into 48-byte data units, the last of which contains the CPCS-PDU trailer. The 48-byte SAR-PDUs are ready for ATM encapsulation.

When the ATM layer receives the 48-byte data cells, it prepends a 5-byte header. This ATM header designates the channel through which the packet will navigate its way through the network. Included in the header information are the virtual path identifier (VPI) and the virtual channel identifier (VCI).

2.2 *Project Objective and Scope*

In this project, we will simulate a simple network for bi-directionally transferring RTP packets between two packet generators/sinks using the OPNET modeling tools. Intermediate network elements will perform RTP/UDP/IP compression and decompression, as described by RFC 2508 [3], and encapsulate RTP/UDP/IP compressed or uncompressed packets into ATM cells to be transferred over a virtual channel, as described in the ATM Forum contribution SAA_98-0139. The end-to-end duplex system is depicted in Figure 4.

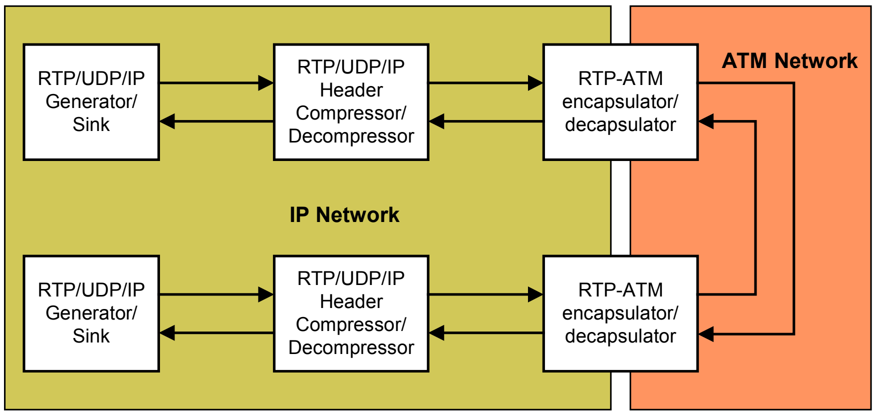
RTP/UDP/IP Generator/ Sink
RTP/UDP/IP Header Compressor/ Decompressor
RTP-ATM encapsulator/ decapsulator
ATM Network
IP Network
RTP/UDP/IP Generator/ Sink
RTP/UDP/IP Header Compressor/ Decompressor
RTP-ATM encapsulator/ decapsulator

Figure 4: RTP Stream over AAL5 Network Overview

Our main objective is to correctly model the algorithms for compression and decompression of RTP/UDP/IP headers as well as the ATM encapsulation and decapsulation of the RTP streams.

In a realistic network environment, there are many factors present that would complicate our model and require a large amount of additional consideration and effort. To help us focus on our objectives, we have made additional assumptions, which are described in more detail in Section 4.3.

3. Theory and Methodology

This section describes the technique used to compress RTP/UDP/IP headers, and the algorithm used to encapsulate the compressed packets into ATM AAL5 cells.

3.1 RTP Header Compression/Decompression

The packet header compression technique used in our project, as described in [3], is based upon a similar technique used to compress TCP/IP headers. The technique utilizes the fact that most header fields in the TCP and IP encapsulation stay constant or increment by a fixed amount [10]. In the same light, UDP and RTP header fields also exhibit zero or generally fixed first-order differences. This section will highlight the general algorithm taken to compress and decompress the RTP/UDP/IP packet headers. For a more detailed description of the compression intricacies, please refer to [3].

3.1.1 Method

A typical RTP packet over IP is composed of the IP and UDP headers, shown in Figure 5 and Figure 6 respectively, as well as the RTP packet header in Figure 1. Their relationship within the packet is illustrated in Figure 7.

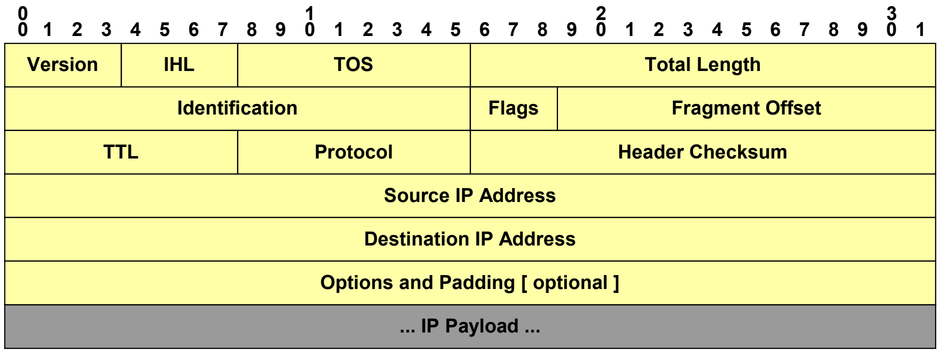

Figure 5: IP Packet Format

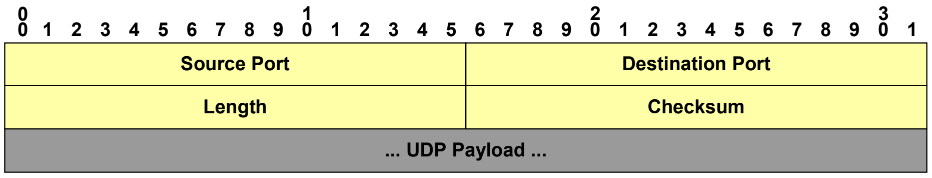

Figure 6: UDP Packet Format

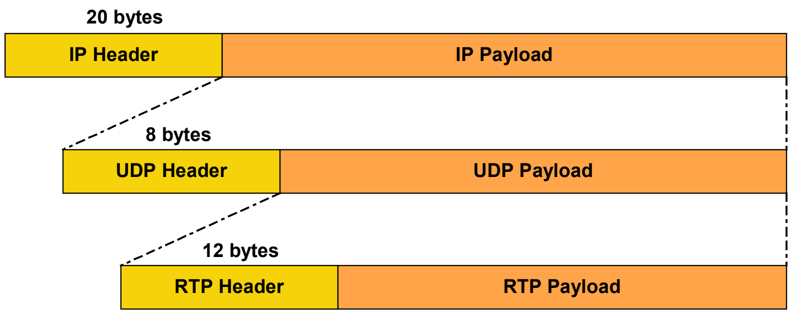

Figure 7: RTP/UDP/IP Packet Breakdown

Within these RTP/UDP/IPv4 packet headers, only a certain number of fields are constantly changing, whilst the other fields never or rarely change. Furthermore, many of changing fields increment by constants, a statistic that is taken advantage of in this compression technique.

The header fields that constantly change are summarized in Table 1.

Table 1: Changing IP, UDP, and RTP Fields

Packet Format	Field Name
IP	Packet Identification Total Length Header checksum
UDP	Length Checksum
RTP	Sequence Number Timestamp Marker CSRC Count CSRC List

In the IPv4 header, the total length of the IP packet may be derived from the link layer, and error detection may rely on the error detection of the Layer 2. A requirement of the compression technique used in this project is that the link layer provides adequate error detection towards packet transfers. The IPv4 packet ID is only used for IP fragmentation, but is transmitted for lossless compression.

As with the IP case, the UDP header can also rely on the Layer 2 protocol to handle its length field. The UDP checksum is transmitted for lossless compression, and will contain the value zero if it is not used.

The RTP header sequence number and timestamp fields usually change between packets, with the sequence number typically incrementing by one for each packet and the timestamp incrementing by a fixed duration depending on the payload carried (e.g. audio packets, video packets). When set, the RTP marker (M) bit indicates that a packet is either the first or the last of an RTP stream. Finally, if packets flow through an RTP mixer, then the CSRC list and CC count may also change.

Amongst the identified changing fields, the IPv4 packet ID, RTP sequence number, and RTP timestamp fields are the most likely to change by a constant value (i.e. constant first-order difference, and second-order difference of zero). If any of these fields should change by a new increment, the new first-order difference is sent with the compressed packet.

An RTP session, as described in Section 2.1.1 may transmit RTP packets from several session contexts. Each session context is identified through a unique 8- or 16-bit context identifier (CID) depending on the number of contexts required. Each packet, whether compressed or uncompressed, must carry the CID and 4-bit link sequence number that is used to detect packet loss.

For IP version 4, a context shares the following information:

- The full IP, UDP, and RTP headers last sent by the compressor or reconstructed by the decompressor
- The first-order difference for the IPv4 ID field (default 1), RTP sequence number (default 1), and RTP timestamp field (default 0)
- The last value of the 4-bit link sequence number

Three packet formats are used between the header compressor and header decompressor, varying in degrees of compression: COMPRESSED_RTP, COMPRESSED_UDP, and FULL_HEADER (uncompressed).

3.1.1.1 COMPRESSED_RTP Packet Format

When each of the RTP, UDP, and IP packet headers may be compressed – such that the only fields that have changed are the IP identification, RTP timestamp, RTP sequence number, the RTP marker bit, and the RTP contributing sources – the COMPRESSED_RTP format is used. This format is shown in Figure 8 with the required fields lightly shaded, and the optional fields in white.

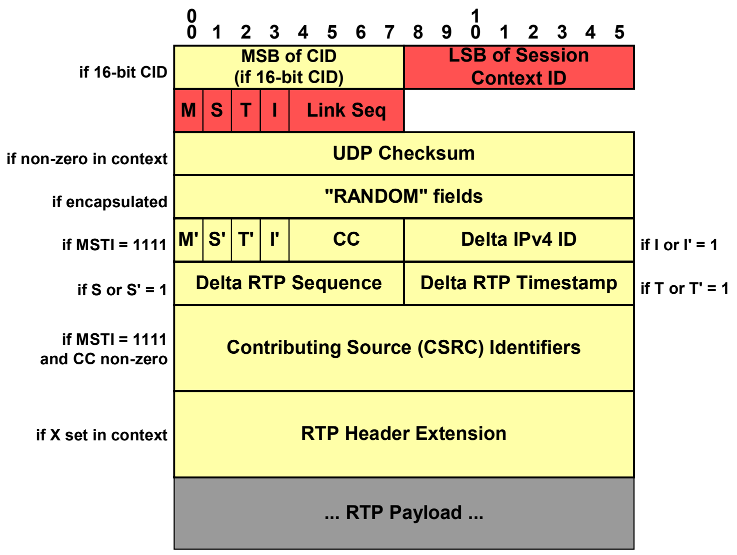

Figure 8: COMPRESSED_RTP Packet Format

The COMPRESSED_RTP format provides maximum compression, possibly reducing a 40-byte RTP/UDP/IP header to as few as 2 bytes. Additional header fields are added if the RTP session uses 16-bit context identifiers, the UDP checksum, or if either of the RTP sequence number, RTP timestamp, IP identification, or CSRCs have changed. The S, T, I, and MSTI (4-bits together) flags indicate whether or not these additional first-order differential fields are required.

When using the COMPRESSED_RTP packet format, the original packet shown in Figure 7 is reduced to the packet illustrated in Figure 9.

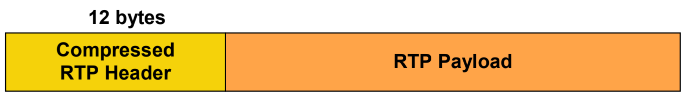

Figure 9: Compressed RTP Packet

3.1.1.2 COMPRESSED_UDP Packet Format

The COMPRESSED_UDP format is used when an RTP field that normally stays constant undergoes a change and the RTP header cannot be compressed. In this case, the full RTP header can be carried in the payload of the COMPRESSED_UDP. This packet follows the format shown in Figure 10.

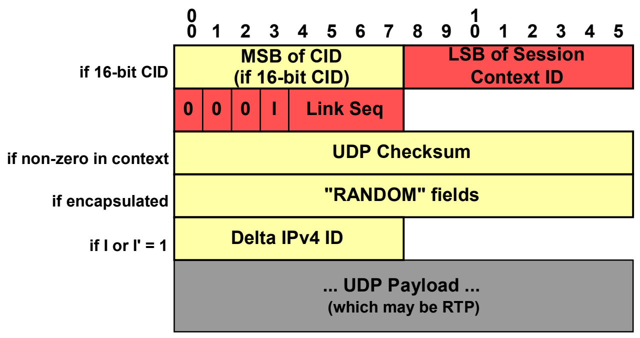

Figure 10: COMPRESSED_UDP Packet Format

When the COMPRESSED_UDP packet format is used, the original packet breakdown shown in Figure 7 is reduced to the packet illustrated in Figure 11.

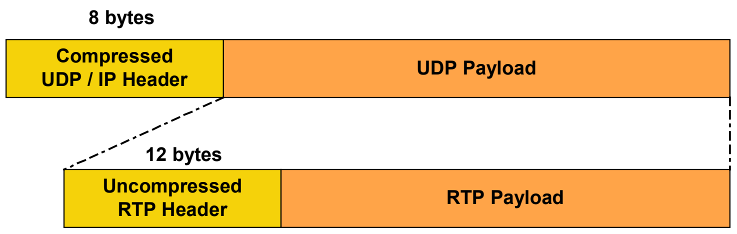

Figure 11: Compressed RTP Packet

3.1.1.3 FULL_HEADER Packet Format

This uncompressed format is the same as that of the original RTP/UDP/IPv4 packet, with the IP total length and UDP length fields used to carry the context identifier and link sequence values. Figure 12 and Figure 13 show the IP and UDP length fields for session contexts with 8-bit and 16-bit CIDs respectively.

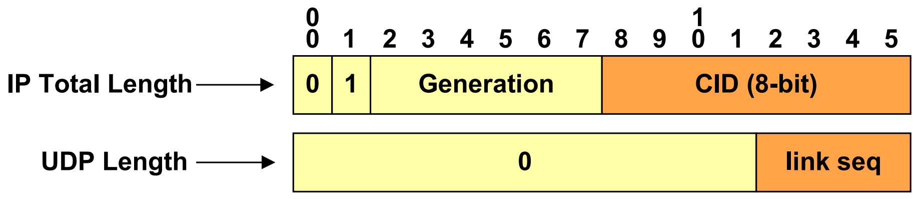

Figure 12: IP and UDP Length Fields in FULL_HEADER (for 8-bit CID)

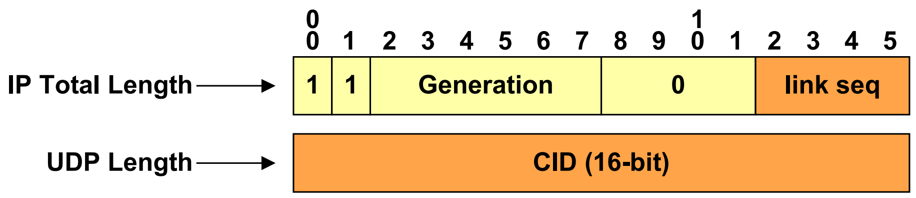

Figure 13: IP and UDP Length Fields in FULL_HEADER (for 16-bit CID)

A FULL_HEADER packet is sent in two scenarios: as the first packet of a session context, and in the infrequent case where neither the UDP or IP headers can be compressed.

As the first packet of a session context, the RTP, UDP, and IP headers are stored by the compressor and decompressor; the delta IPv4 ID, delta RTP sequence number, and delta RTP timestamp are reset to their defaults (1, 1, 0 respectively); and the link sequence for this new context is set to 0. Thereafter, when a FULL_HEADER packet is transmitted, the RTP, UDP, and IP headers stored in the compressor and decompressor are refreshed, and the delta values are reset to their defaults.

Although this packet format is uncompressed, for the remainder of this document, the FULL_HEADER packet format will be included in the "compressed packet formats".

In a physical system, both the RTP/UDP/IP header compressor and decompressor must store context information about the RTP streams it is sending or receiving, as depicted in Figure 14. These context states are used by the compressor to determine the level of compression that can be done on the headers, as well as which additional fields are required to communicate changing first-order differences. The decompressor uses the context states in conjunction with the incoming compressed packet to reconstruct the original packet.

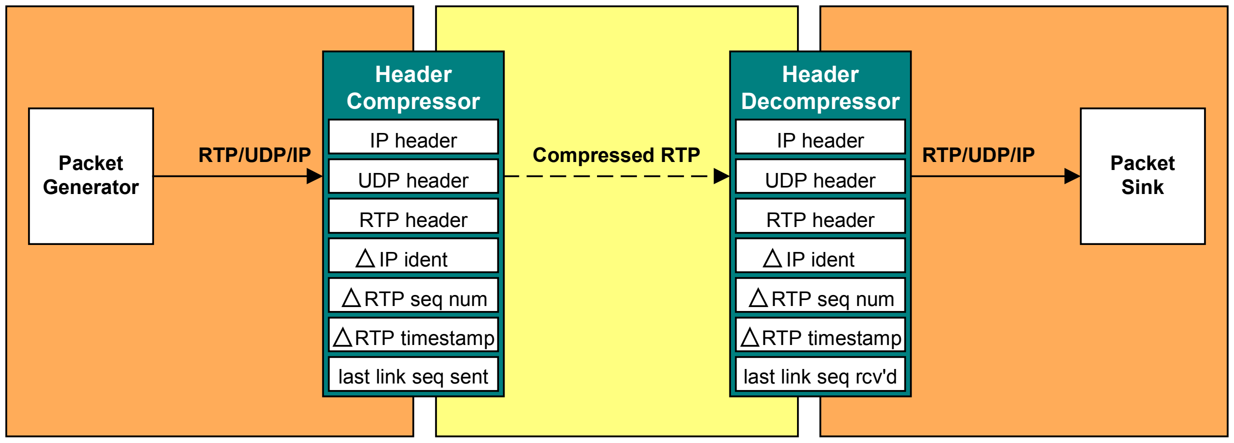

Figure 14: Context Information Storage in Compressor and Decompressor

3.1.2 Flow Chart

The header compressor is responsible for determining the amount of compression that may be done on incoming uncompressed RTP/UDP/IP packet headers. In compressing the headers, the compressor also determines which additional fields are required for carrying first-order differential (delta) values.

The header compressor follows the flowchart shown in Figure 15.

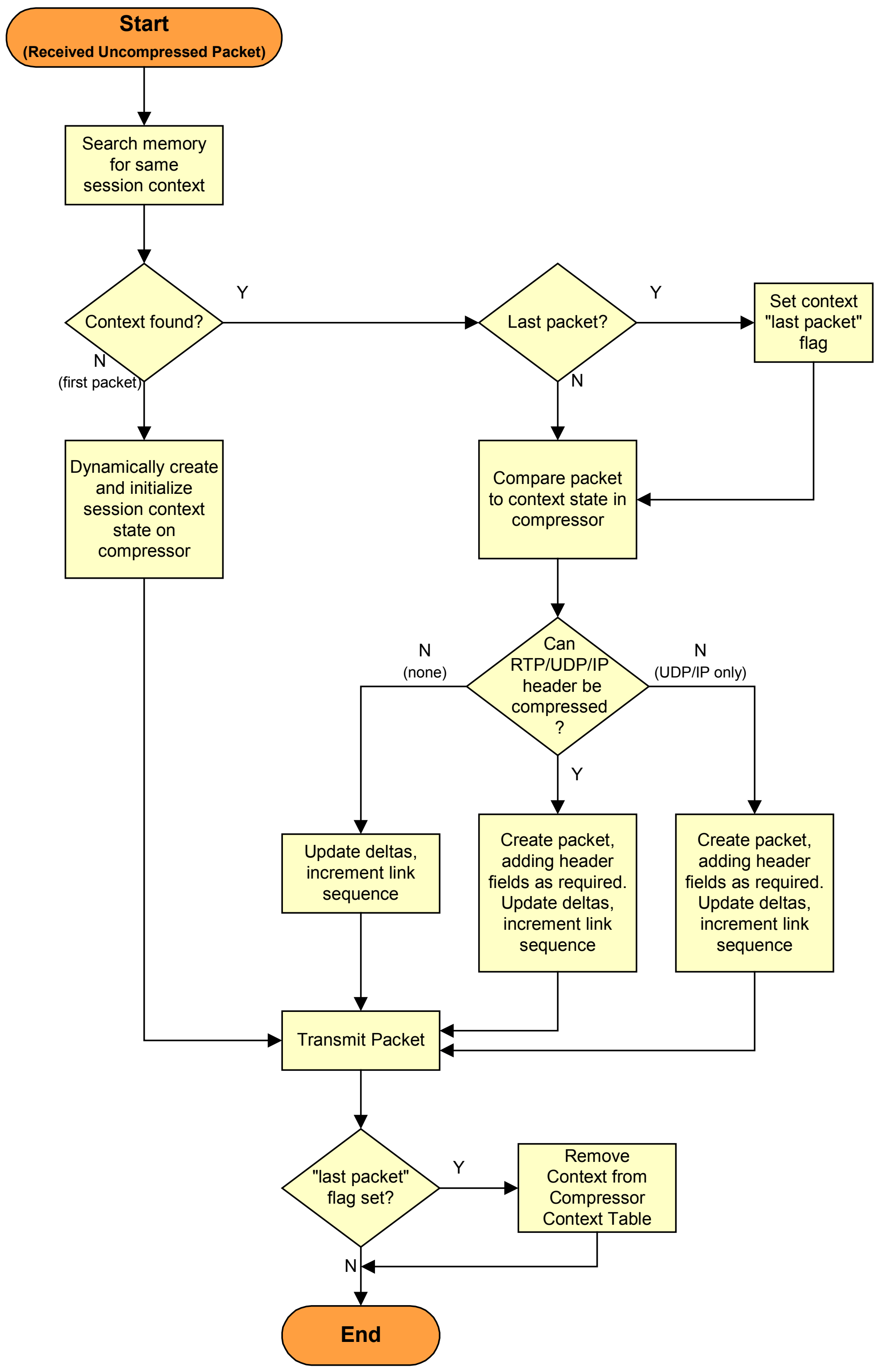
Start
(Received Uncompressed Packet)
Search memory for same session context
Context found?
Y
N
(first packet)
Dynamically create and initialize session context state on compressor
Last packet?
Y
N
Set context "last packet" flag
Compare packet to context state in compressor
Can RTP/UDP/IP header be compressed ?
N
(none)
N
(UDP/IP only)
Y
Update deltas, increment link sequence
Create packet, adding header fields as required. Update deltas, increment link sequence
Create packet, adding header fields as required. Update deltas, increment link sequence
Transmit Packet
"last packet" flag set?
Y
Remove Context from Compressor Context Table
N
End

Figure 15: Flowchart for RTP/UDP/IP Header Compressor

The header decompressor accepts the incoming compressed packets from the compressor, creates new RTP/UDP/IP packets based upon the received packets and the delta fields, updates the context state for that particular CID, and sends off the reconstructed packet to the IP destination.

The header decompressor follows the flow chart shown in Figure 16.

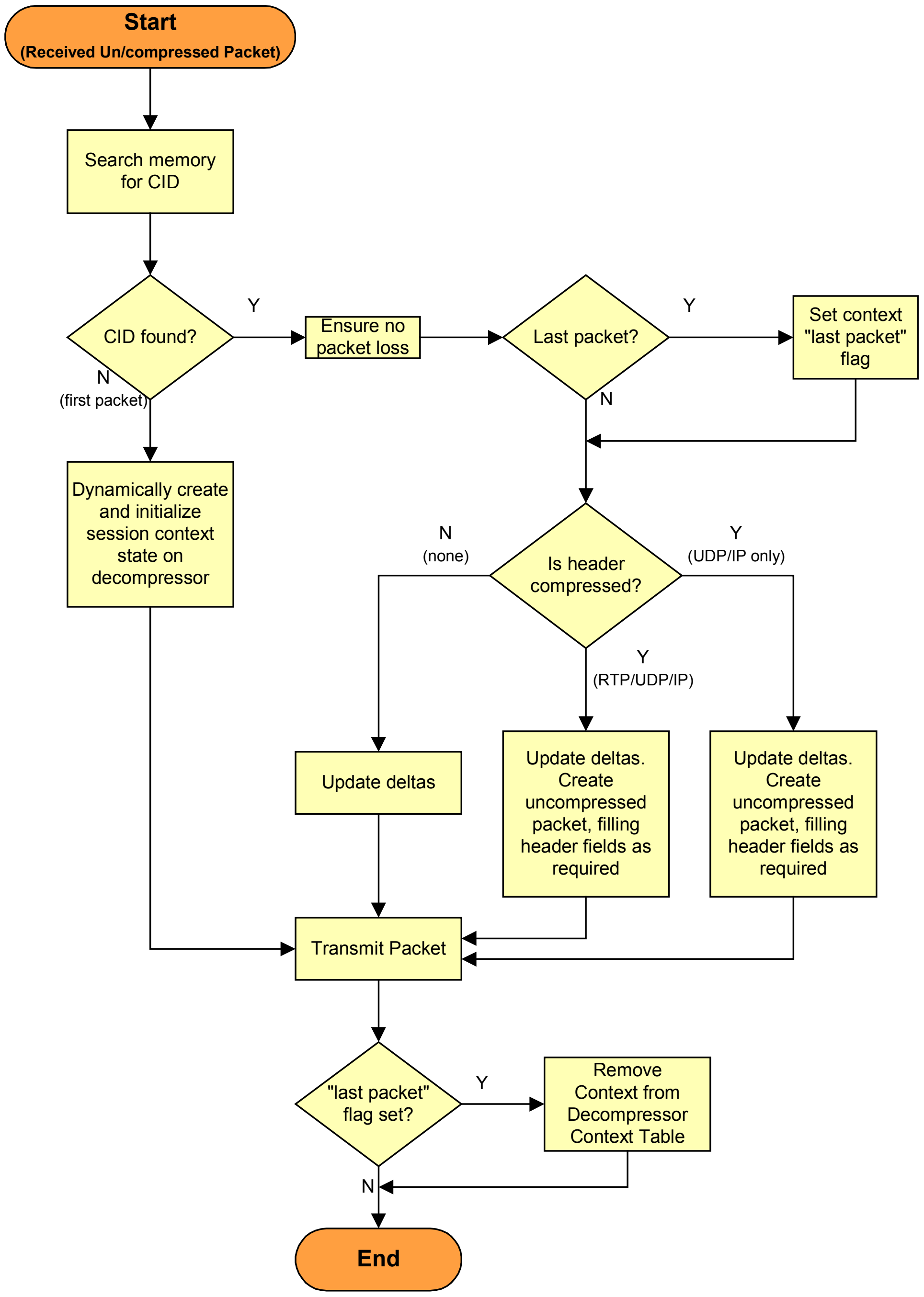
Start
(Received Un/compressed Packet)
Search memory for CID
CID found?
Y
N
(first packet)
Ensure no packet loss
Last packet?
Y
N
Set context "last packet" flag
Dynamically create and initialize session context state on decompressor
Is header compressed?
N
(none)
Y
(UDP/IP only)
Y
(RTP/UDP/IP)
Update deltas
Update deltas. Create uncompressed packet, filling header fields as required
Update deltas. Create uncompressed packet, filling header fields as required
Transmit Packet
"last packet" flag set?
Y
N
Remove Context from Decompressor Context Table
End

Figure 16: Flowchart for RTP/UDP/IP Header Decompressor

3.2 ATM Encapsulation/Decapsulation

Packets from the RTP/UDP/IP header compressor/decompressor must be transported over a network to the destination host. This transmission is performed by an ATM network. The ATM device at the User-to-Network Interface (UNI) must support RTP/UDP/IP packet formats, and recognize the other two formats associated with the RTP compression scheme used. The unit responsible for the ATM layer encapsulation and decapsulation of these RTP (compressed and uncompressed) packets will be referred to as the ATM module.

To avoid implementing an entirely new ATM adaptation layer, an existing one was chosen to best suit the needs of transporting real-time data over ATM. Although an AAL1 encapsulated cell includes a 3-bit sequence number and a 4-bit checksum, allowing 47 bytes of payload, it is more suited to PBX-PBX communication [6].

The AAL5 trailer lacks a sequence number field that would be much desired for packet loss and mis-ordering detection, but the CPCS-UU byte is available for transparently transferring information between users, so it has been customized to house an extension bit and a 7-bit sequence number field (see Figure 17). The resulting Real-Time AAL5 encapsulation format was devised in Voice over ATM to the Desktop work [6]. Additionally, the length field permits variable length payloads such as RTP packets. However, a 40-byte payload is preferred for optimization purposes: this payload can be transmitted in a single ATM cell.

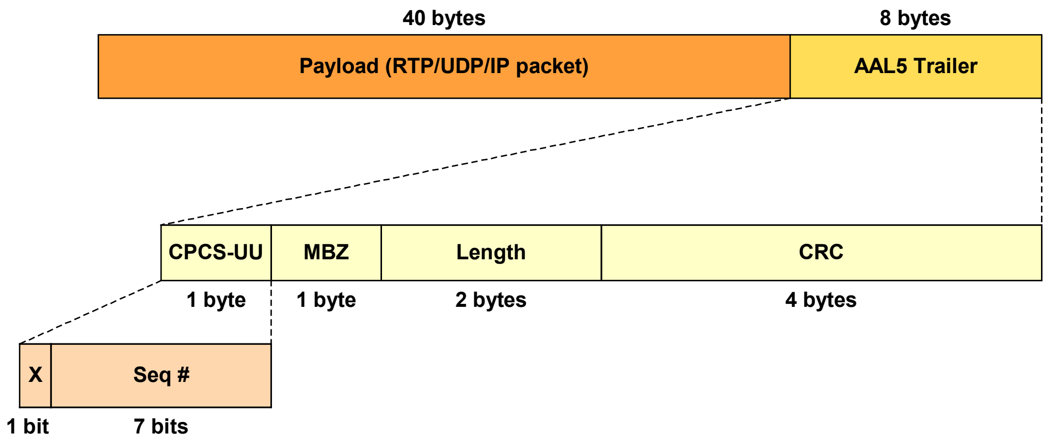

Figure 17: Real-Time AAL5 Encapsulation

3.2.1 Method

The assignment of AAL5 field values is based largely on field values extracted from the higher layer data packet. As mentioned in the previous section, the CPCS User-to-User field in the AAL5 trailer was split into an extension bit and a 7-bit sequence number. The AAL5 sequence number takes on the value of the incoming IP packet's 4-bit sequence number (right-justified and left-padded with zeros).

If the AAL5 encapsulation discerns the SDU as an uncompressed packet (FULL_HEADER format, described in Section 3.1.1.3), the extension bit is assigned the value 1. For either of the compressed packet formats – COMPRESSED_RTP (Figure 8) and COMPRESSED_UDP (Figure 10) – the extension bit takes on value 0.

During the decapsulation process, the ATM module must also detect context state packets. When the receiving-end RTP layer detects a corrupted packet, or one whose link sequence number changed by more than 1 from the previous packet received (indicating a lost packet), a context state packet is sent to inform the source of the packet loss or corruption. The ATM module on the receiving end must understand not to modify any field values from normal operation when the context state packet is sent; the packet should be passed to the AAL5 and ATM layers as is.

An RTP session may contain more than one RTP stream (each with its own unique SSRC value), but does not have any field dedicated to identifying itself uniquely. Thus, the ATM module cannot create or assign virtual circuits at the ATM UNI based on the RTP session to which a packet belongs. Moreover, the source and destination IP addresses are not readily available in the compressed packet formats.

To circumvent this issue, the ATM module keeps a list of RTP context ids in use, and all the information associated with that particular id (IP addresses, UDP ports, etc.). This list allows the ATM module to look up the destination address for a packet without knowing the decompression scheme in use and to map the appropriate outgoing VC, as shown in Figure 18. At the receiving UNI, the ATM module refers to the CID list again to deliver the compressed packet to the correct IP address.

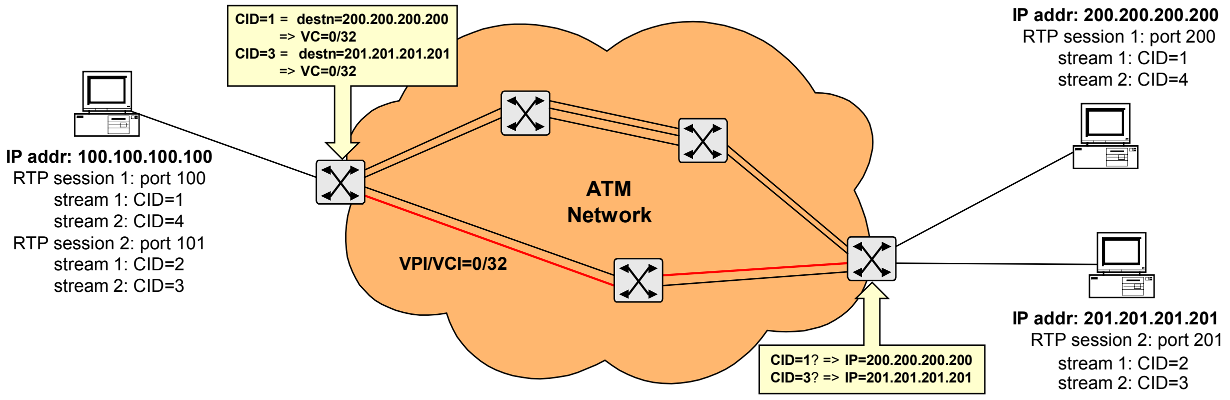

Figure 18: Virtual Circuit Assignment for Multiple RTP Sessions

3.2.2 Flow Chart

The flow charts of Figure 15 and Figure 16 show the logic followed by the ATM encapsulator/decapsulator.

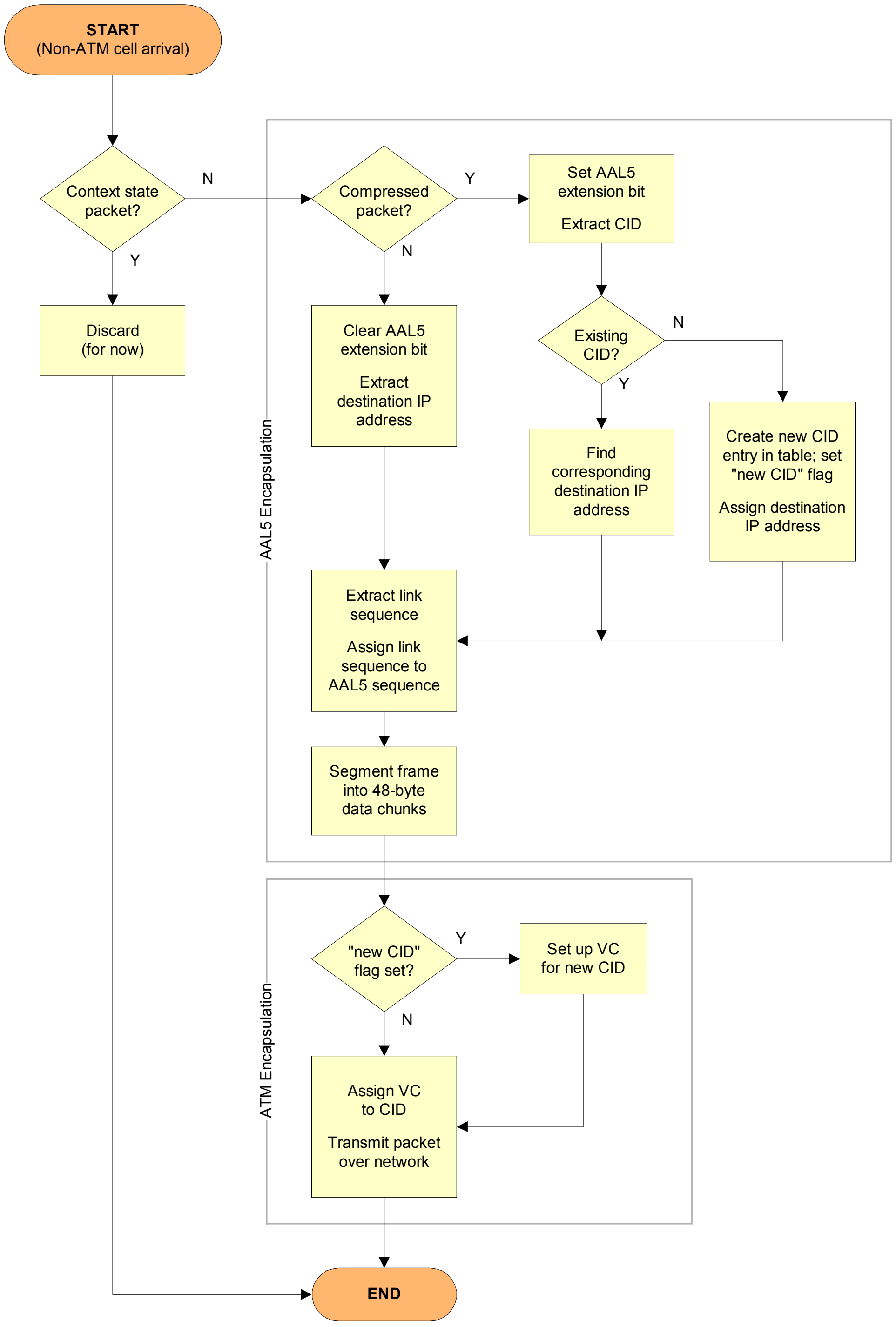

Figure 19: Flowchart for ATM Encapsulator

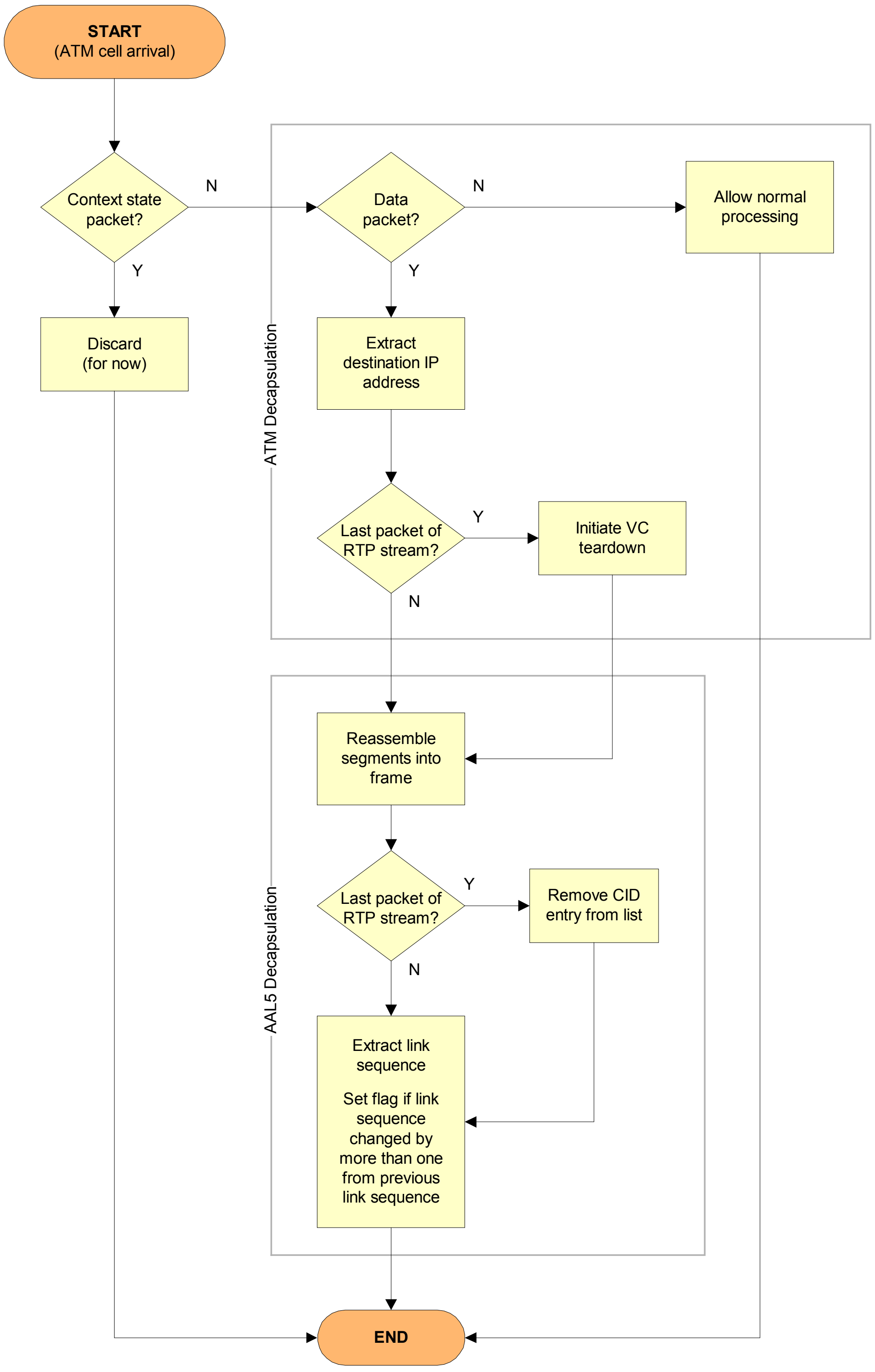
START
(ATM cell arrival)
Context state packet?
N
Y
Discard (for now)
ATM Decapsulation
Data packet?
N
Allow normal processing
Y
Extract destination IP address
Last packet of RTP stream?
Y
Initiate VC teardown
N
AAL5 Decapsulation
Reassemble segments into frame
Last packet of RTP stream?
Y
Remove CID entry from list
N
Extract link sequence
Set flag if link sequence changed by more than one from previous link sequence
END

Figure 20: Flowchart for ATM Decapsulator

4. OPNET Implementation

Three main components constitute our end-to-end system: an RTP/UDP/IP generator and sink, an RTP/UDP/IP header compressor/decompressor, and an ATM encapsulator/decapsulator.

The remaining sections of this project will describe our OPNET project, and the node and process models that construct these three components.

A complete OPNET project would consists of the following elements:

- 2 RTP/UDP/IP generators and sinks
- 2 RTP/UDP/IP header compressor/decompressors
- 2 ATM encapsulator/decapsulators
- 2 ATM switches

These network elements are logically arranged in the organization seen in Figure 21.

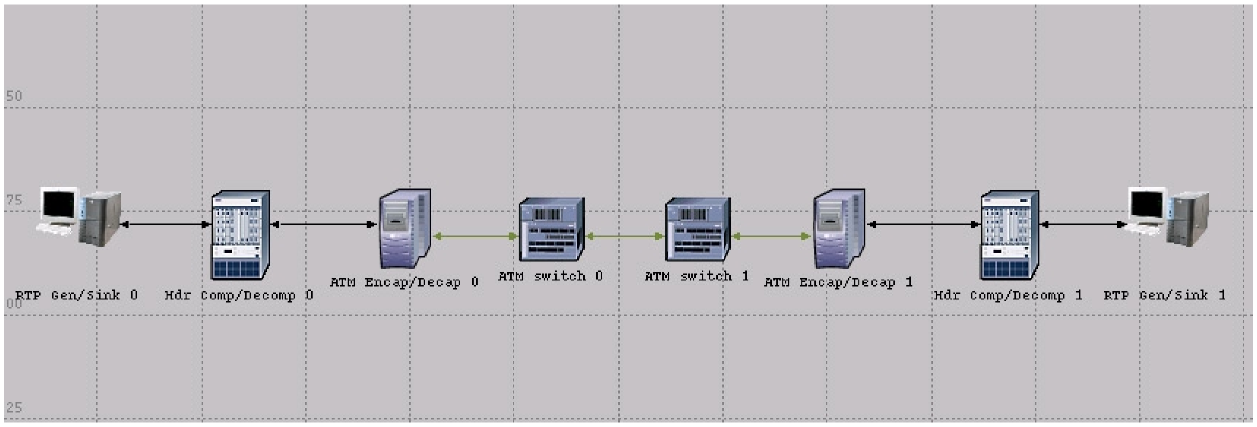

Figure 21: OPNET Project

This project can be regarded as two identical data paths, traveling in opposite directions. Each data path is seen in Figure 22.

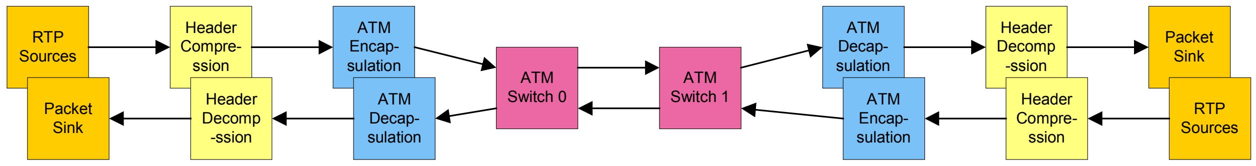

Figure 22: Single Direction Data Path

Each data path consists of three traffic sources, each sending a unique RTP/UDP/IP traffic stream to an aggregator. The aggregator combines the packet streams from the three generators into a single point-to-point link, which is connected to the RTP/UDP/IP header compressor. The

RTP/UDP/IP header compressor compresses the packet headers as necessary and sends each packet towards the ATM encapsulator. Upon receiving the compressed packet, the ATM encapsulator encapsulates the packet into an ATM cell, and transmits the cell over a virtual channel to the ATM decapsulator through two ATM switches. The decapsulator removes the compressed packet from the ATM cell and sends the packet to the RTP/UDP/IP header decompressor. The decompressor reconstructs the packets based upon its stored context state information and transmits the uncompressed packet to the packet sink.

The same data transfer process takes place in the opposite direction, using the counterparts of the same components used in the first direction of data transfer.

Due to time constraints, our project was implemented in two separate subnetworks, which would be integrated together to form the end-to-end system shown in Figure 21. The first subnetwork consists of the RTP stream generation, header compression, header decompression, and packet sink. The second subnetwork includes the ATM encapsulation, switching, and decapsulation. These two subnetworks are described in Sections 4.1 and 4.2.

4.1 RTP Subnetwork

The RTP subnetwork handles all of the RTP-related responsibilities: generating traffic, compressing packets, decompressing packets, sinking and verifying the data. The node model for this subnetwork is shown in Figure 23.

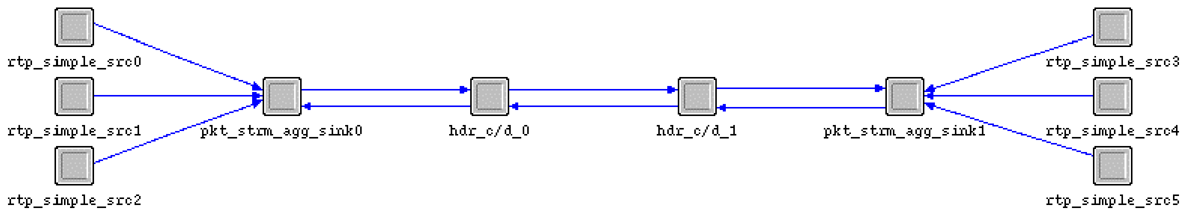

Figure 23: RTP Subnetwork

4.1.1 RTP Simple Sources

Three RTP Simple Sources are used in each direction to simulate different RTP streams starting at different times, and sending a different number of packets (this will be discussed in further detail in the Verification section in 4.4).

The RTP Simple Source process model is based on the predefined OPNET process model "simple_src". The process model of the RTP/UDP/IP Simple Source contains the same states and state transitions as the simple_src process model; however, modifications have been made to the packet generation code such that only RTP/UDP/IP packets are created, and that their header fields are filled with predefined values based upon those in a fixed array of packet headers. The header values have been selected to test the compression algorithm used in the RTP/UDP/IP header compressor (refer to Section 4.4 for more details about the header values).

4.1.2 Aggregator and Sink

The Aggregation and Sink Processor combines multiple RTP streams (from RTP Simple Sources) into a single output stream. The xmt state is responsible for this aggregation. The number of input streams is scalable, requiring only simple modifications to add new streams. This process model is also used for sinking packets that have been sent from the far end of our end-to-end system. In the rcv state, packets are disassembled, and their header contents are stored in an array for verification purposes (see Section 4.4). The process model for the Aggregator and Sink is shown in Figure 24.

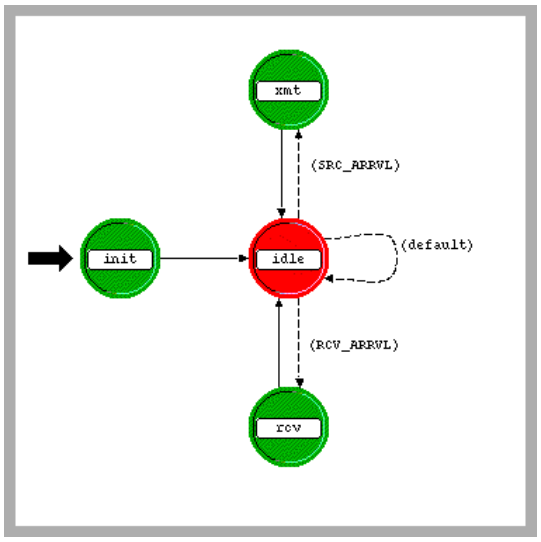

Figure 24: Aggregator and Sink Process Model

4.1.3 Header Compressor/Decompressor

The Header Compressor/Decompressor process model is shown in Figure 25. The Compressor state handles incoming RTP/UDP/IP packets, compressing them to the degree possible before sending them off. The Decompressor state receives compressed packets and reconstructs a packet with the full packet headers. This process model contains two separate dynamic tables of context states to store information about the RTP streams sent/received by the compressor/decompressor.

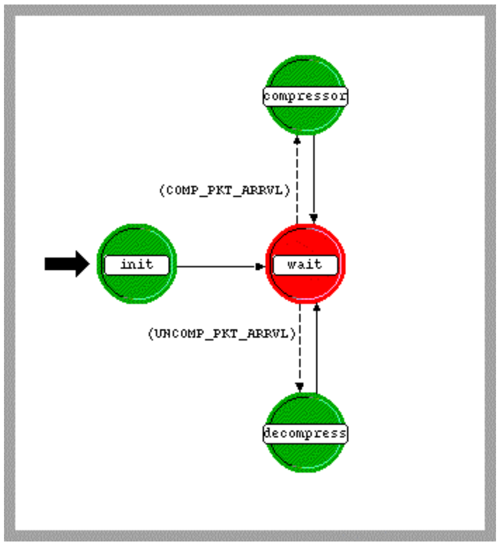

Figure 25: Compressor/Decompressor Process Model

4.2 ATM Subnetwork

ATM has been in the network industry for many years now, and OPNET provides a complete model suite; therefore the ATM implementation portion started with an investigation of what models and features were available in ATM Model Suite.

To concentrate on the focus of the project, the ATM network "cloud" was simplified to two switches connecting the user-to-network devices seen in Figure 26 below.

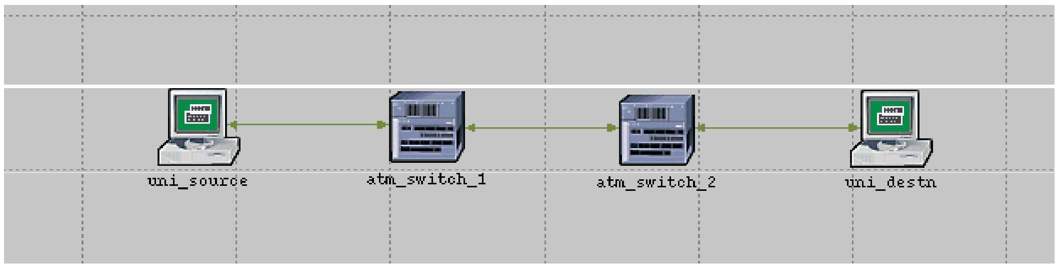

Figure 26: ATM Subnetwork Components

4.2.1 ATM Encapsulator/Decapsulator

Normally, both the encapsulator and decapsulator would be contained within the ATM encapsulator/decapsulator unit. However, to work on the functionality, they were left as separate components, to be integrated at a later time.

4.2.2 Encapsulator

The uni_source node in Figure 27 is based on OPNET's atm_uni_src_adv node, which generates raw unformatted packets, processes them through the AAL and ATM layers and transmits them over the network.

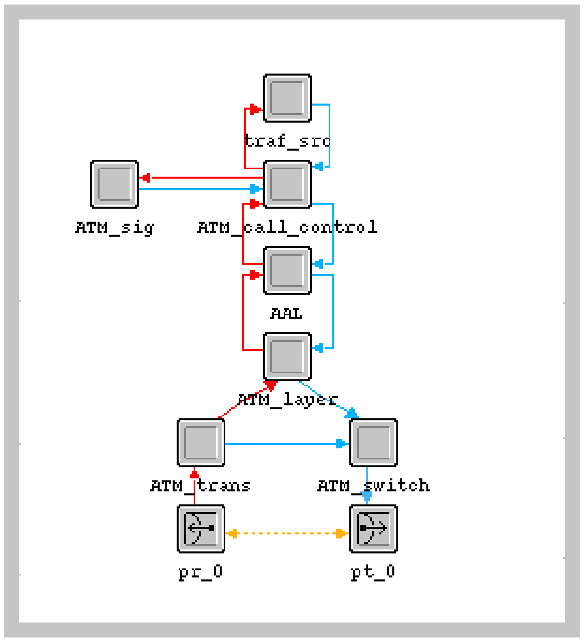

Figure 27: ATM uni_source Node

Instead of unformatted packets, the traffic generation function within uni_source node has been modified to generate traffic of the three packet formats produced by the RTP/UDP/IP header compressor: FULL_HEADER, COMPRESSED_UDP, COMPRESSED_RTP. In order to simplify verification of the changes made at the AAL, simple field values are set such that outcome is predictable.

For data packets coming into the ATM adaptation layer, the CPCS user-to-user indication is overridden, as described in Section 3.2.1. This modification is accomplished in the ams_aal5_conn_v3 process model (Figure 28), a child process of ams_aal_disp_v3 (called from within the to_atm state "enter" execs). Once created, a packet format cannot be changed, so the CPCS-UU is calculated and set to

$$CPCS\_UU = (extension\_bit) * 128 + link\_sequence$$

so that the extension bit indicating a compressed or uncompressed packet resides in the most significant bit of the one-byte field. The extension bit is determined by the incoming packet format, and the link sequence is extracted from its header fields. The RTP session context identification is also collected at this layer. For uncompressed RTP/UDP/IP packets, the IP packet's length field and that of its encapsulated payload are used to store the CID and link sequence number; these length fields are processed according to the formats for an 8-bit or 16-bit CIDs, shown in Figure 12 and Figure 13, respectively. While this portion has been implemented for both 8-bit and 16-bit CIDs, only the 8-bit CID is used in this project.

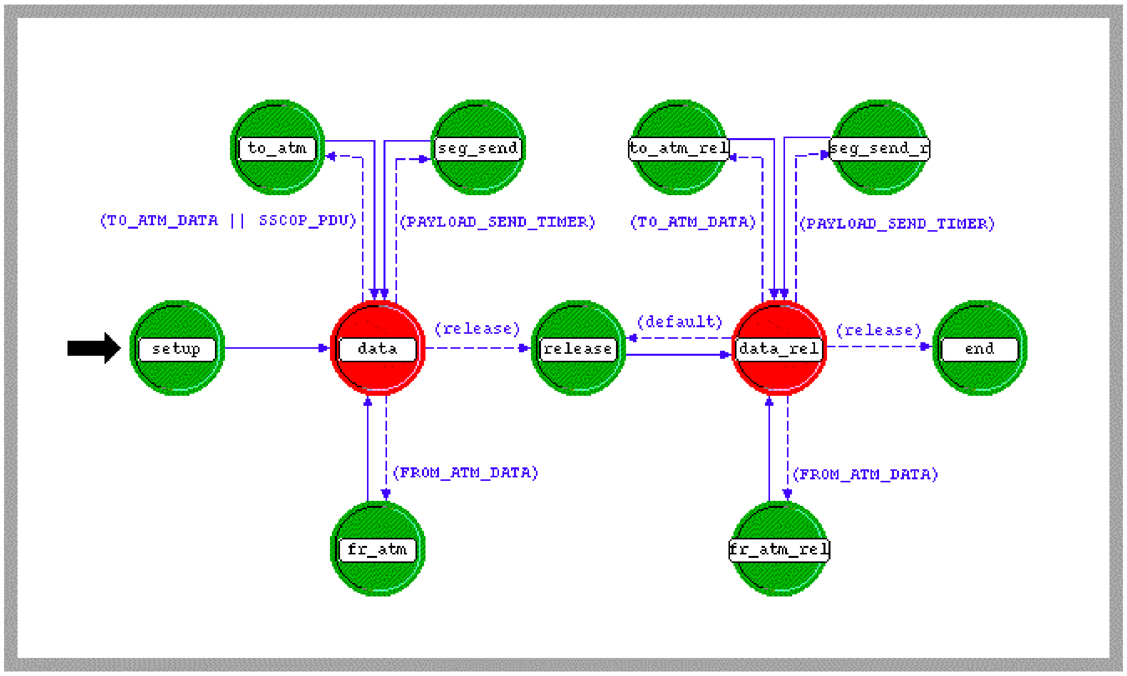

Figure 28: ams_aal5_conn_v3 Process Model

The CID was to determine the virtual channel on which to send the packet; however, due to various reasons outlined in Section 5.2 (Difficulties), the CID-to-VC mapping was not implemented.

4.2.3 Decapsulator

The uni_destn node shown in Figure 29 is based on OPNET's atm_uni_dest_adv node, which sinks incoming packets, after their ATM header and AAL trailer have been removed and the segments reassembled.

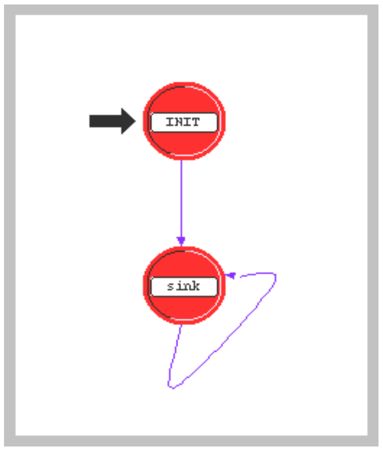

Figure 29: Process Model for uni_destn Node

Before sinking the packets (or, when fully integrated, passing the packets to the RTP/UDP/IP header decompressor), the destination node checks the link sequence number of the packet, noting if it has changed by more than one since the previous packet for that particular RTP stream (distinguished by the CID). If the link sequence number has increased by more one, the occurrence is flagged, indicating a probable lost packet.

Minor changes to this process model included verifying reception of the correct packet type and CPCS-UU field.

4.3 Assumptions

In order to keep focused on our objectives and to keep the scope of this project reasonable, we have made a number of assumptions with regards to the environment in which we intend to send our RTP/UDP/IP. The assumptions described in this section are only those that we have made for the purposes of simplifying our project's implementation in OPNET, and are not necessarily stated by the documents describing compression algorithm and ATM encapsulation (RFC 2508 and ATM Forum/SAA-98-0139 respectively).

Our assumptions include an already established RTP session between our RTP session endpoints, and ideal IP and ATM environments. We also decided to begin with the use of only one RTP session, which may contain multiple RTP streams. Lastly, we have made certain requirements on the packet characteristics and contents.

4.3.1 Establishing RTP Sessions

Before RTP stream packets are sent between users, an RTP session must be set up. These sessions may be established through a number of protocols including Session Initiation Protocol (SIP) and H.323. We also rely on the session-establishing protocol to terminate and teardown the session when it is no longer used.

SIP is an application-layer control protocol defined in RFC 2543 that can be used to establish, maintain, and terminate calls between two or more end points. H.323 is an ITU standard that incorporates multiple protocols, including H.245 for negotiation and Registration Admission and Status (RAS) for session control [5].

As there is an ATM component to our network, we also assume that the appropriate VCs have been established in the ATM network to transport data between the ATM encapsulator and decapsulator, since these must already be set up in order for the RTP session to be established.

4.3.2 Link Layer Requirements

As mentioned in Section 3.1.1, the link layer must provide adequate error detection and must also be capable of indicating lengths of packets being sent via its protocol. An example of acceptable error detection is the Point-to-Point Protocol's (PPP's) Cyclic Redundancy Check (CRC) as described in RFC 1661 [3].

These requirements are in place such that their respective fields may be excluded or replaced in the compressed packets.

Because we are not dealing with the link layer in our simulation, the total packet lengths will be transmitted along with the packet between the two Header Compressors/Decompressors via an OPNET Interface Control Information (ICI). The ICI will also contain the IP header checksum.

4.3.3 Ideal Network Environment

As a starting point, the network environment, in the IP and ATM domains, are assumed to be ideal. This assumption includes, but is not limited to, the following: no dropped packets, no transmission errors, no packet collisions, and no delay of transmission.

4.3.4 RTP Session and Streams

We have decided that in order to prove the function of our end-to-end system, that we do not need more than one RTP session set up (see assumption stated in Section 4.6.1). The RTP session will, however, have multiple RTP streams/contexts – identified by the source and destination IP addresses, UDP ports, and RTP SSRC.

Our implementation also assumes that each RTP stream contains more than one packet, as the RTP Marker bit is used to indicate the first and last packets of the stream. Having only one packet in a stream will cause confusion, as there will be no indication of the stream's end. We believe that this is a fair assumption in the practical world as well.

4.3.5 Packet and Payload

We have assumed that only the IP version 4 protocol will be used. Future work may include incorporating IPv6 header compression into our process model. While the ATM Encapsulator/Decapsulator implementation allows it, the RTP Compressor/Decompressor assumes that only one IP header is present – as opposed to having IP packets encapsulated in IP packets.

Furthermore, we have decided not to utilize the UDP checksum field for our testing. Accounting for the UDP checksum would be a small task, and would only require that additional fields be included in the compressed packets.

4.4 Verification Method

Verification of our models has been done both manually and automatically. When programming each process model, the expected behavior of the model was verified through printouts. These printouts were used to confirm proper header compression and ATM encapsulation had been done.

In order to verify the end-to-end system, after each RTP stream had finished sending its packets, the receiving end sink would compare the array of received header values with the array of header values sent from the RTP Simple Source. Any discrepancy would be reported; otherwise a "Success" message would be printed (see Section 5.1.1).

The details of each RTP Simple Source packet stream are as follows:

Table 2: RTP Simple Source Packet Streams

Source	Source IP Address	Destination IP Address	Source UDP Port	Destination UDP Port	RTP SSRC	Number of packets	Start time (sec)
Rtp_src_0	100.100.100.100	175.175.175.175	100	175	246248	2000	0.0
Rtp_src_1	200.200.200.200	175.175.175.175	200	175	1191	1000	100.4
Rtp_src_2	200.200.200.200	175.175.175.175	2000	175	75930	3000	250.3
Rtp_src_3	150.150.150.150	225.225.225.225	150	225	91122	2000	20.0
Rtp_src_4	150.150.150.150	225.225.225.225	150	2250	8286	1000	140.3
Rtp_src_5	250.250.250.250	225.225.225.225	250	225	788128	3000	75.0

Within each RTP stream, header field values of certain packets are modified such that the compression algorithm can be tested.

Verification of the ATM subsystem was less involved, since the mapping was easily verified by inspection. The packet formats generated were assigned fixed values (that is, hard-coded) to distinguish easily the different packet types and wrongly mapped field values. The following table shows field values that affect AAL data collection.

Table 3: Hard-Coded Field Values for AAL Mapping (IPv4 Packet)

	8-bit CID	16-bit CID
length 1 (IPv4)	4096+111 (CID=111)	4096+6 (link seq = 6)
length 2 (encapsulated IP)	1024+1 (link seq = 1)	2222 (CID = 2222)
length 2 (encapsulated UDP)	1024+2 (link seq = 2)	2222 (CID = 2222)
Expected UU Value (IP/IP)	1*128+1=129	1*128+6=134
Expected UU Value (UDP/IP)	1*128+2=130	1*128+6=134

Table 4: Hard-Coded Field Values for AAL Mapping (Compressed Packets)

	COMPRESSED_UDP Packet	**COMPRESSED_RTP Packet**
session_context_id	117	75
link_sequence	3	5
Expected CPCS-UU Value	0*128+3=3	0*128+5=5

Printouts show which nodes have been visited and the processing that was done at that node. A sample of the ATM output is provided in Section 5.1.2.

5. Discussion

This section analyzes the results achieved through our simulations and explains difficulties that were encountered during the implementation of our end-to-end network. Some potential future enhancements are also included

5.1 Results

Sections 5.1.1 and 5.1.2 describe the results from the RTP Subnetwork and ATM Subnetwork respectively.

5.1.1 RTP Subnetwork Results

The RTP Subnetwork was successfully implemented with simulations showing expected behaviour by the compressors and decompressors. When errors were manually inserted, they were correctly reported; likewise when no errors were expected, none were reported. After running a simulation with the RTP streams specified in Section 4.4, the following output printed:

```
Aggregator 3: Initialized.
Aggregator 4: Initialized.
Source 5: Initialized.
Header Comp/Decomp 6: Initialized.
Header Comp/Decomp 7: Initialized.
Source 8: Initialized.
Source 9: Initialized.
Source 10: Initialized.
Source 11: Initialized.
Source 12: Initialized.

Compressor 6: CID 0 assigned to source with SSRC 246248.
Decompressor 7: CID 0 (SSRC 246248) added to Context State Table.

Compressor 7: CID 0 assigned to source with SSRC 91122.
Decompressor 6: CID 0 (SSRC 91122) added to Context State Table.

Compressor 7: CID 1 assigned to source with SSRC 788128.
Decompressor 6: CID 1 (SSRC 788128) added to Context State Table.

Compressor 6: CID 1 assigned to source with SSRC 1191.
Decompressor 7: CID 1 (SSRC 1191) added to Context State Table.

Compressor 7: CID 2 assigned to source with SSRC 8286.
Decompressor 6: CID 2 (SSRC 8286) added to Context State Table.

Compressor 6: CID 2 assigned to source with SSRC 75930.
Decompressor 7: CID 2 (SSRC 75930) added to Context State Table.

STREAM 1 SUCCESS: All header field values match original values (1000 packets).
Decompressor 7: Removing CID 1 (SSRC 1191) from Context State Table... DONE.
Compressor 6: Removing CID 1 (SSRC 1191) from Context State Table... DONE.

STREAM 4 SUCCESS: All header field values match original values (1000 packets).
Decompressor 6: Removing CID 2 (SSRC 8286) from Context State Table... DONE.
Compressor 7: Removing CID 2 (SSRC 8286) from Context State Table... DONE.
```

```
STREAM 0 SUCCESS: All header field values match original values (2000 packets).
Decompressor 7: Removing CID 0 (SSRC 246248) from Context State Table... DONE.
Compressor 6: Removing CID 0 (SSRC 246248) from Context State Table... DONE.

STREAM 3 SUCCESS: All header field values match original values (2000 packets).
Decompressor 6: Removing CID 0 (SSRC 91122) from Context State Table... DONE.
Compressor 7: Removing CID 0 (SSRC 91122) from Context State Table... DONE.

STREAM 5 SUCCESS: All header field values match original values (3000 packets).
Decompressor 6: Removing CID 1 (SSRC 788128) from Context State Table... DONE.
Decompressor 6: That was the last element of contextStateRcvList.
Compressor 7: Removing CID 1 (SSRC 788128) from Context State Table... DONE.
Compressor 7: That was the last element of contextStateXmtList.

STREAM 2 SUCCESS: All header field values match original values (3000 packets).
Decompressor 7: Removing CID 2 (SSRC 75930) from Context State Table... DONE.
Decompressor 7: That was the last element of contextStateRcvList.
Compressor 6: Removing CID 2 (SSRC 75930) from Context State Table... DONE.
Compressor 6: That was the last element of contextStateXmtList.
```

Figure 30: RTP/UDP/IP Header Compressor/Decompressor Output

The output above indicates that a context state entry is inserted into the dynamic context state tables of the compressor and decompressor each time the first packet of a new RTP stream is received. When the last packet of the stream is received, the header fields of the received packets are compared to the header fields of the original/sent packets – the result of these comparisons is Success (i.e. matching header fields). Finally, the context state entry for an ended stream is removed from the context state tables of the compressor and decompressor.

Our simulation was run with a fixed RTP payload of 40 bytes. Looking at the number of received and sent bytes at the Header Compressor/Decompressor node hdr_c/d_0, we may observe the conservation of bandwidth. Figure 31, Figure 32, and Figure 33 compare the traffic received against the traffic sent for Compressor 0, Compressor 1, and Decompressor 1 respectively. It is noted that the Compressor 0 Sent traffic is the same as the Decompressor 1 Received traffic, as expected.

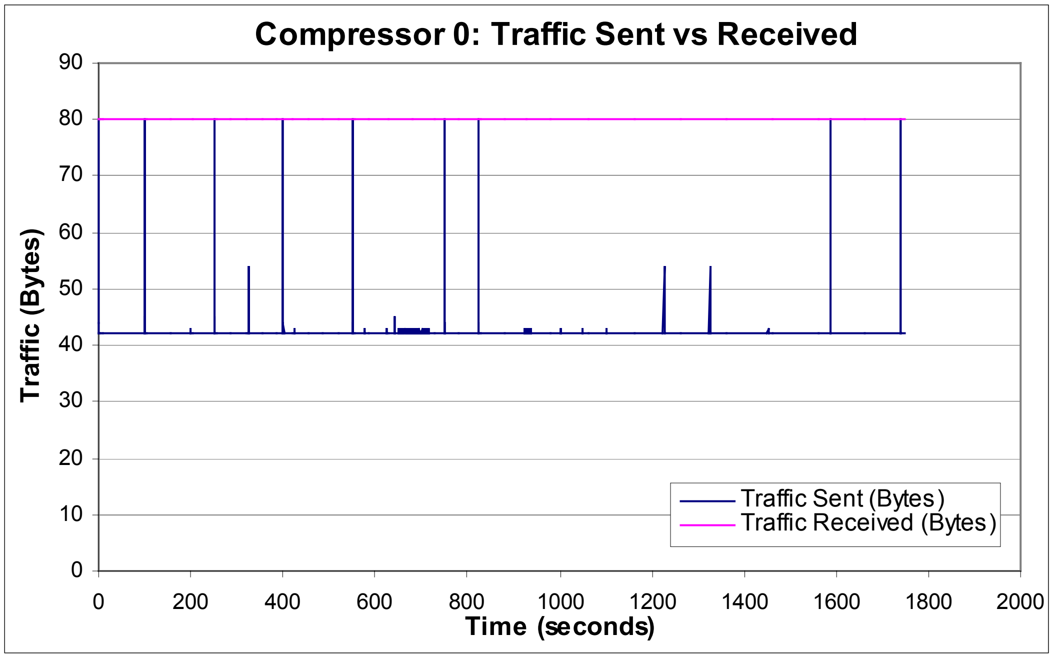

Figure 31: Bytes Sent Versus Bytes Received at Compressor 0

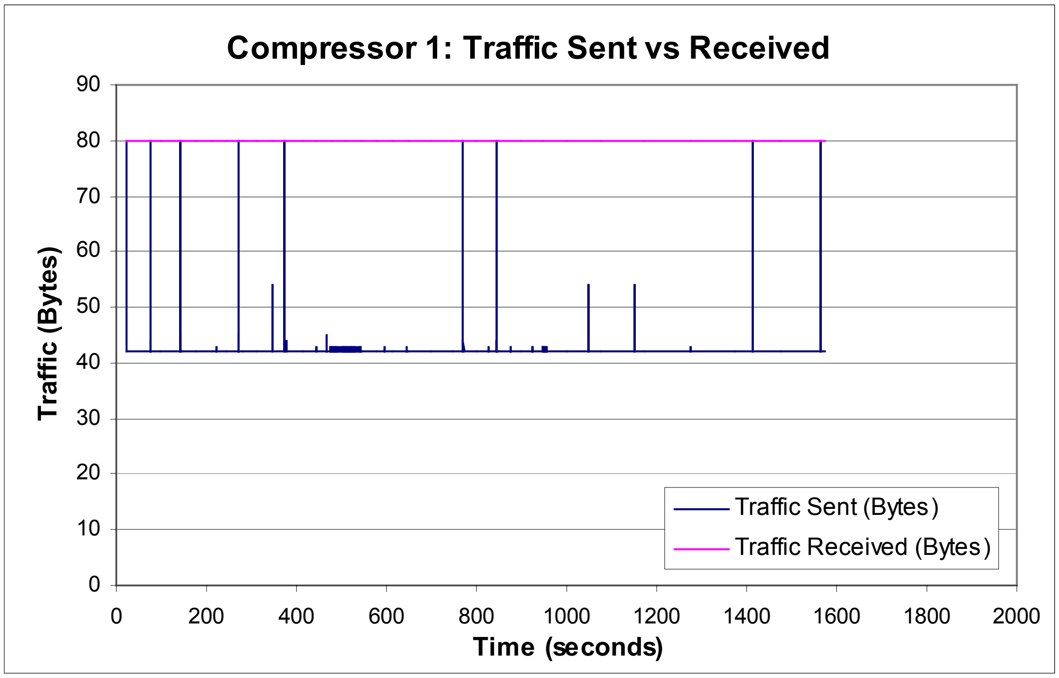

Figure 32: Bytes Sent Versus Bytes Received at Compressor 1

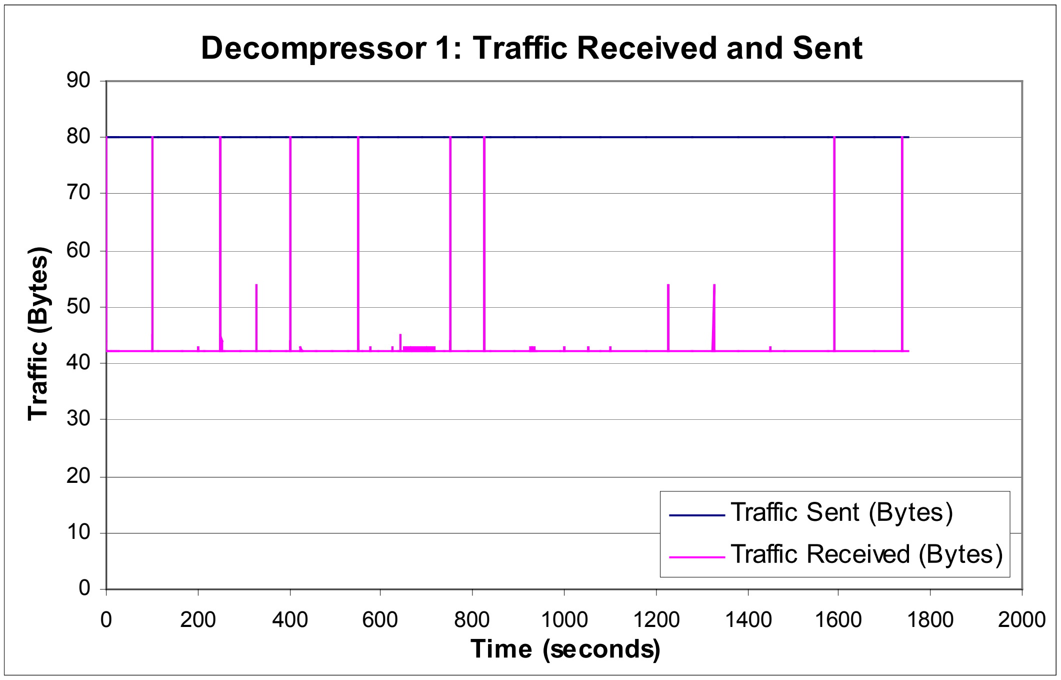

Figure 33: Bytes Sent Versus Bytes Received at Decompressor 1

From the three previous figures, we can see where the COMPRESSED_RTP format is used, with packet sizes of 42 bytes or slightly higher when a delta header field is sent; the COMPRESSED_UDP format is used, with packet sizes of 54 bytes or slightly more; and the FULL_HEADER format is used, with 80-byte packets. With different RTP payload sizes, the graphs will change only slightly. The difference between the bytes received and bytes sent will stay constant (given the same header values), however the graph may be offset by a different amount (instead of 40 bytes). As such, the percent of bandwidth savings due to header compression is dependent on the RTP payload size.

With 40-byte RTP payloads, the total number of bytes received and sent from each compressor are shown in Figure 34 and Figure 35. Again with larger payloads, the gap between the total traffic received and total traffic sent will stay constant, however both lines would rise faster.

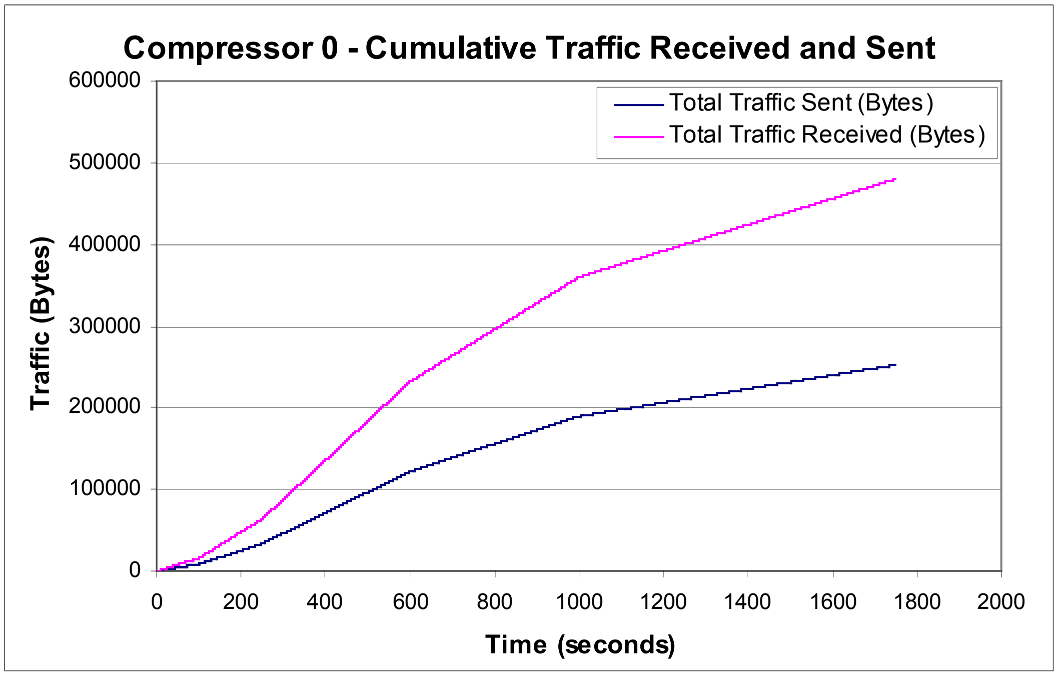

Figure 34: Total Bytes Sent Versus Total Bytes Received at Compressor 0

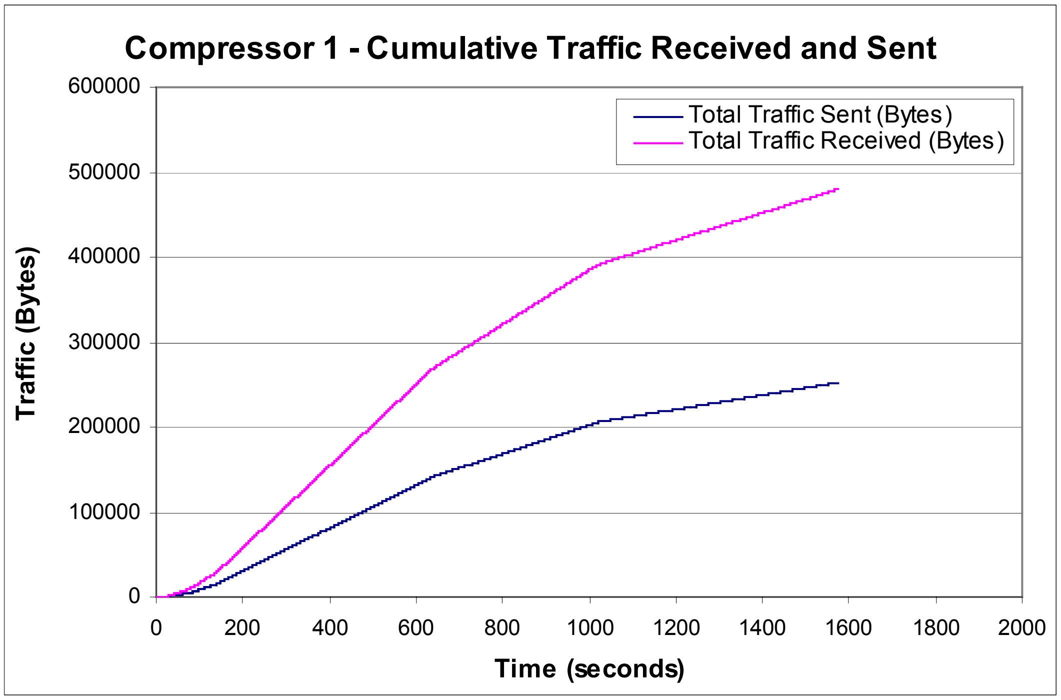

Figure 35: Total Bytes Sent Versus Total Bytes Received at Compressor 1

5.1.2 ATM Subnetwork Results

Because the ATM subnetwork is not concerned with any measures of performance or network behaviours, mapping are verified simply by comparing output values. A sample of the ATM Encapsulator/Decapsulator's output for each type of incoming packet format is listed in the following figure.

```
IPv4 packet fields set!  (IP encapped)

encapsulating CPCS PDU (original UU = 0)
   packet format  : ams_aal5_cpcs_pdu
      field  0: payload
      field  1: UU
      field  2: CPI
      field  3: Length
      field  4: CRC
   payload format : ip_v4_pkt
      field  0: version
      field  1: ihl
      field  2: tos
      field  3: length
      field  4: ident
      field  5: flags
      field  6: frag_offset
      field  7: ttl
      field  8: protocol
      field  9: header_checksum
      field 10: src_addr
      field 11: dest_addr
      field 12: data
   yay, let's go set the CPCS-UU field now!

   (Using 8-bit CID)
      length1 = 4207
      CID = 111
      length2 = 1025
      link_seq = 1
   extension_bit = 1
   new user-to-user = 129
      extension bit = 1 (ip_v4_pkt)
      link sequence = 1
atm_switch_1
atm_switch_2
uni_destn :
   cell is for this node!
   arriving user_to_user = 129
[ip_v4_pkt]
-------------------
compressed RTP fields set!

encapsulating CPCS PDU (original UU = 0)
   packet format  : ams_aal5_cpcs_pdu
      field  0: payload
      field  1: UU
      field  2: CPI
      field  3: Length
      field  4: CRC
   payload format : crtp_main_hdr
      field  0: session_context_id
      field  1: M
      field  2: S
      field  3: T
      field  4: I
      field  5: link_seq
      field  6: data
   yay, let's go set the CPCS-UU field now!

   CID = 75
   link_seq = 5
   extension_bit = 0
   new user-to-user = 5
      extension bit = 0 (crtp_main_hdr)
      link sequence = 5
atm_switch_1
atm_switch_2
uni_destn :
   cell is for this node!
atm_switch_1
atm_switch_2
uni_destn :
   cell is for this node!
   arriving user_to_user = 5
[crtp_main_hdr]
-------------------
compressed UDP fields set!

encapsulating CPCS PDU (original UU = 0)
   packet format  : ams_aal5_cpcs_pdu
      field  0: payload
      field  1: UU
      field  2: CPI
      field  3: Length
      field  4: CRC
   payload format : cudp_main_hdr
      field  0: session_context_id
      field  1: M
      field  2: S
      field  3: T
      field  4: I
      field  5: link_seq
      field  6: data
   yay, let's go set the CPCS-UU field now!

   CID = 117
   link_seq = 3
   extension_bit = 0
   new user-to-user = 3
      extension bit = 0 (cudp_main_hdr)
      link sequence = 3
atm_switch_1
atm_switch_2
uni_destn :
   cell is for this node!
atm_switch_1
atm_switch_2
uni_destn :
   cell is for this node!
   arriving user_to_user = 3
[cudp_main_hdr]
-------------------
```

Figure 36: ATM Encapsulator/Decapsulator Output

For some packets, atm_switch_1 and atm_switch_2 (and uni_destn) are traversed multiple times, indicating that the original packet was segmented into several ATM cells.

Comparison of these output values with those assigned in the traffic generation node demonstrate that the CPCS-UU mapping was done correctly.

5.2 Difficulties

Throughout the development of our project, we encountered various roadblocks, largely due to OPNET. Complications arising from the OPNET software libraries resulted in many frustrations and lost programming time. It was no surprise that a good amount of time was spent learning to use OPNET's kernel procedures as well.

Errors were encountered when trying to join nodes with generic links, causing us to put the different processors (RTP Simple Source, Aggregator/Sink, Header Compressor/Decompressor) into a single node model, instead of different node models – which is more logical.

Because the COMPRESSED_UDP and COMPRESSED_RTP packet formats require additional fields when sending new delta values is necessary, it was important to find a way to give this kind of flexibility. Our solution was to create additional packet formats to simulate these individual optional fields. If a particular optional field were required, then our packet would be encapsulated by an additional header (which would be the single field).

OPNET has an undesirable habit of changing a packet, or its contents sometimes, even when that packet is not the object of focus. For example, the command

```
op_pk_nfd_set (the_pdu, “payload”, sdu)
```

not only changes the_pdu, but also releases the contents of sdu. As a result, sdu's format was still provided but OPNET, but any fields that were previously present were no longer accessible and generated an error during simulation. An attempt to circumvent this issue involved accessing the field values via C's memory operations and bit-shifting the values to obtain the field of interest; however, C's bit operators require knowledge of variable length, and MSBs shifted off a value cannot be retrieved. It would be desirable to have an OPNET command that forces an existing packet into a format, mapping binary bits regardless of field boundaries.

The ATM models installed with OPNET are read-only, and highly dependent on each other. Any small changes had to be locally saved, often meaning changing the name a header file: Because of the interdependent models, changing the name of a header file forced the user to change the name is every other process model used (new names for all other models!). Much time was spent trying to work around the interdependencies and duplicating editing models to avoid redundant declarations and the such.

5.3 Future Enhancements

Due to time constraints, we decided on simplifying our implementation. For a more complete project, we believe that some additional future work is required.

RFC 2508 also describes a CONTEXT_STATE packet format, sent from a decompressor to a compressor when dropped packets are detected (identified using the link sequence values). This would be a necessary addition to our implementation in order to satisfy the RFC's specification.

The current RTP compressor/decompressor implementation removes a context entry from the context state tables when the last packet has been received and forwarded (from the compressor and the decompressor). Future work would include adding an acknowledgement from the RTP destination to indicate to the compressor and decompressor that the last packet has successfully arrived.

In order not to restrain the position of a UNI device on a network, two-way ATM communication would be necessary. Currently, the traffic source and destination sink are implemented using separate nodes; ideally these would be combined so that a UNI device could transmit and receive the compressed RTP packets.

The traffic coming into the ATM end devices would more realistically come from a local area network or from another computer. So an enhancement to our project would be to deliver traffic via packet streams (from another LAN device), instead of generating it internally at the UNI.

6. Conclusion

As Internet traffic increases with demand, demand for better bandwidth usage also increases. The ATM Forum Contribution by AT&T Labs researchers Alexander Fraser, Peter Onufryk, and K.K. Ramakrishnan, describes the ATM encapsulation of compressed RTP packets, saving bandwidth for real-time applications like voice and video.

A simplified version of the proposed encapsulation scheme was successfully implemented in OPNET Modeler. Certain assumptions were placed in order to accomplish the main focus of the paper – the encapsulation of compressed and uncompressed RTP/UDP/IP packets.

Results show that the compression of packet headers before ATM encapsulation saves on the transmission of much repetitive information that can be stored at the source and destination.. The new packet formats introduced into the system must be recognized and processed by the ATM encapsulation/decapsulation module, to accommodate for assumptions made by implementing the header compression scheme.

To fully implement a model as proposed in the research paper, several future enhancements have been presented. These suggestions include the initiation and setup of the RTP session and the ATM virtual channel, consideration of multiple concurrent RTP sessions, and two-way ATM communication. We feel these works are necessary to fully benefit from the advantages offered by the RTP over ATM encapsulation method in a more realistic environment.

7. References

[1] ATM Forum, “About ATM Technology”, http://www.atmforum.com/pages/aboutatmtechfs1.html, March 10, 2002.

[2] Casner, S., Frederick, R., Jacobson, V., and Schulzrinne, H., “RTP: A Transport Protocol for Real-Time Applications”, RFC 1889, GMD Fokus, Precept Software, Inc., Xerox Palo Alto Research Center, Lawrence Berkeley National Laboratory, January 1996.

[3] Casner, S. and Jacobson, V., “Compressing IP/UDP/RTP Headers for Low-Speed Serial Links”, RFC 2508, Cisco Systems, February 1999.

[4] Cisco Systems, Inc. “Asynchronous Transfer Mode (ATM) Switching”, http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/atm.pdf, March 14, 2002.

[5] Cisco Systems, Inc. “Guide to Cisco Systems’ VoIP Infrastructure Solution for SIP – Version 1.0”, pp 1-8, 2000.

[6] Fraser, A., Onufryk, P., and Ramakrishnan, K., “Encapsulation of Real-Time Data Including RTP Streams over ATM”, ATM Forum/SAA-98-0139, AT&T Labs. Research, February 1998.

[7] Heinanen, J., “Multiprotocol Encapsulation over ATM Adaptation Layer 5”, RFC 1483, Telecom Finland, July 1993.

[8] International Telecommunication Union, “B-ISDN ATM Adaptation Layer specification: Type 5 AAL”, ITU-T Recommendation I.363.5, August 1996.

[9] International Telecommunication Union, “B-ISDN ATM Layer Specification”, ITU-T Recommendation I.361, November 1995.

[10] Jacobson, V., “Compressing TCP/RTP Headers for Low-Speed Serial Links”, RFC 1144, LBL, February 1990.

[11] Marshall, Dr. Alan, “ATM Asynchronous Transfer Mode: An Overview”, http://www.pcc.qub.ac.uk/tec/courses/network/ATM/ATMV1_1.html, March 12, 2002.

[12] Philp, Ian, “Research Directions for the Next Generation Internet: Application-Specific Network Services”, http://www.cra.org/Policy/NGI/papers/philpWP, March 05, 2002.

[13] Rutkowski, Tony. “Internet Survey Reaches 109 Million Internet Host Level”, http://www.ngi.org/trends/TrendsPR0102.txt, March 06, 2002.

[14] Webopedia, “RTP”, http://www.pcwebopedia.com/TERM/R/RTP.html, March 14, 2002.

Appendix A – Glossary

Mixer [RFC 1889]: An intermediate system that receives RTP packets from one or more sources, possibly changes the data format, combines the packets in some manner and then forwards a new RTP packet. Since the timing among multiple input sources will not generally be synchronized, the mixer will make timing adjustments among the streams and generate its own timing for the combined stream. Thus, all data packets originating from a mixer will be identified as having the mixer as their synchronization source.

Translator [RFC 1889]: An intermediate system that forwards RTP packets with their synchronization source identifier intact. Examples of translators include devices that convert encodings without mixing, replicators from multicast to unicast, and application- level filters in firewalls.