4-22-2015 12:00 AM

Advanced Compression and Latency Reduction Techniques Over Data Networks

Fuad Shamieh, *The University of Western Ontario*

Supervisor: Dr. Xianbin Wang, *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Master of Engineering Science degree in Electrical and Computer Engineering
© Fuad Shamieh 2015

ADVANCED COMPRESSION AND LATENCY REDUCTION TECHNIQUES OVER DATA NETWORKS

(Thesis format: Monograph)

by

Fuad Shamieh

Graduate Program in Electrical and Computer Engineering

A thesis submitted in partial fulfillment
of the requirements for the degree of
Masters of Engineering Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

Applications and services operating over Internet protocol (IP) networks often suffer from high latency and packet loss rates. These problems are attributed to data congestion resulting from the lack of network resources available to support the demand. The usage of IP networks is not only increasing, but very dynamic as well.

In order to alleviate the above-mentioned problems and to maintain a reasonable Quality of Service (QoS) for the end users, two novel adaptive compression techniques are proposed to reduce packets' payload size. The proposed schemes exploit lossless compression algorithms to perform the compression process on the packets' payloads and thus decrease the overall network congestion. The first adaptive compression scheme utilizes two key network performance indicators as design metrics. These metrics include the varying round-trip time (RTT) and the number of dropped packets. The second compression scheme uses other network information such as the incoming packet rate, intermediate nodes processing rate, average packet waiting time within a queue of an intermediate node, and time required to perform the compression process.

The performances of the proposed algorithms are evaluated through Network Simulator 3 (NS3). The simulation results show an improvement in network conditions, such as the number of dropped packets, network latency, and throughput.

Keywords: Compression, Congestion, TCP, UDP, Latency

Acknowledgments

First and foremost, I would like thank my supervisor, Dr. Xianbin Wang, for his continuous support, guidance, patience, and motivation. He encouraged me to pursue a deeper understanding of my areas of interest.

In addition to my supervisor, I would like to thank Dr. Ahmed Refaey, Dr. Auon Akhtar, and Dr. Hao Li for their continuous guidance and support.

My sincere appreciation and gratitude also goes to Steven Andrews, Kyle Doerr, and Mohamad Khalil for helping me throughout my thesis.

I would like to thank my friends Ali Al-Saady, Alfred Kenny, Antoan Antun, Rakan Duqum, and Scott Van Heesch for keeping my spirits up.

Last but not least, I would like to thank my parents, Jamal Shamieh and Naila Farraj, and my sister, Ilein Shamieh, for their endless support and encouragement.

Finally, I would also like to acknowledge the Faculty of Engineering, especially the Department of Electrical and Computer Engineering at Western University for the excellent facilities, libraries, and resources available to graduate students.

Table of Contents

List of Figures

List of Tables

Abbreviations

ACK	Acknowledgment
API	Application Programming Interface
BLRTT	Baseline Round-Trip Time
CPI	Coding Position Indicator
CSMA/CD	Carrier Sense Multiple Access / Collision Detection
FCS	Frame Check Sequence
FPGA	Field Programmable Gate Array
HTTP	HyperText Transfer Protocol
ICT	Information and Communication Infrastructure
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IRTT	Instantaneous Round-Trip Time
LZ77	Lempel-Ziv
LZO	Lempel-Ziv-Oberhumer
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NBLRTT	New Baseline Round-Trip Time
NS3	Network Simulator 3
P-K	Pollaczek-Khinchin
P2P	Peer-to-Peer
pps	Packets Per Second
QoS	Quality of Service
RTT	Round-Trip Time
SDN	Software Defined Networking
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol
WWW	World Wide Web

Chapter 1

Introduction

The use of traditional, real-time, and delay sensitive applications and services over information and communications technology (**ICT**) infrastructure is becoming more popular. Emerging real-time applications, such as data retrieval from data centers, are delay sensitive. It follows that certain network performance metrics must be realized for an acceptable Quality of Service (**QoS**). Maintaining these expectations is becoming more difficult because of the continuous increase in size and frequency of the generated and transmitted data.

The primary challenge to overcome is the instigated end-to-end delay as the behavior and usage of a given network changes. The rapid growth of the number of clients and their dynamic behavior within an Internet protocol (**IP**) network induces largely varying data flows, yielding high data congestion. In order to improve the usage of real-time applications, end-to-end delays must be reduced [1][2].

The particular focus of this thesis is the design and validation of different compression schemes that are adaptive in nature and easily deployed. The compression schemes attempt to compress the payload of data packets being transmitted over a given network set-up. The reduction in packet size after successfully applying compression results in an increase in bandwidth utilization and a reduction in network congestion, thereby decreasing the number of dropped packets.

The advantages of the compression schemes are enhanced bandwidth utilization and increased number of active simultaneous network applications while maintaining a smooth communication session. An adaptive compression scheme is preferable over a constant compression scheme for two reasons: the slight possibility of having a compression process that may increase the overall transmission time, and increased number of CPU cycles [3][4].

In the following sections, a brief description of networks and the congestion problem is given.

1.1 Defining Networks

Traditional communication networks operate over circuit-switched network configuration. In circuit-switched networks, the route resources between a source-destination pair are reserved over the transmission medium to guarantee a certain level of performance. The concept of reserving network resources limits the number of simultaneous active source nodes as well as the allocated bandwidth per node. Additionally, a certain amount of bandwidth must be reserved at all times for a node whether it is active or not. To increase the number of nodes that can transmit data over a network, as well as the bandwidth allocated per node, packet-switched networks are deployed to replace circuit-switched networks. An example of a circuit-switched network can be seen in Figure 1.1 [1].

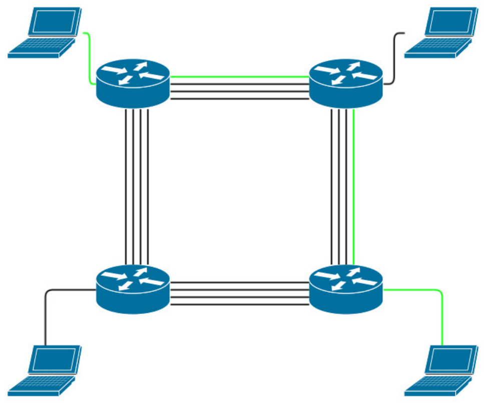


Figure 1.1: A Simple Circuit-Switched Network with Four Links and Four Routers

[1]The defined number of links is for illustrative purposes.

Packet-switched networks do not provide any service guarantees. These types of networks are designed to allocate route resources for a source-destination pair on demand. Consequently, packet switched networks are known as best-effort networks.

In such networks, messages generated by a source node are encapsulated into packets where they experience the full link speed when transmitted. Nevertheless, a packet will experience different types of delays throughout its journey, including transmission, propagation, queuing, and processing delay. An example of a packet-switched network can be seen in Figure 1.2.

The Internet is a complex example of a heavily used packet-switched network. The devices connected to this network, such as end-hosts and routers, are physically and logically connected together, as a result of which information can be exchanged. The exchanged information includes data generated by real-time and traditional applications.

To successfully exchange data over packet switched networks, communication protocols that exchange messages with certain commands and formats are used. Indeed, there is a multitude of different protocols available for use to accomplish various tasks. A few examples of these protocols include congestion-control, routing data, and electronic mail protocol [5].

Today, societies rely on best-effort networks for different services although they are known for not providing any guarantees on packet delivery or QoS. Within such networks, it is common to have dropped packets, packets utilizing different routes between a source-destination pair, and congestion [9]. Due to the importance of these networks, it is necessary to improve the condition of a communication session.

In this thesis, the focus will be on network congestion as described in the following section.

1.2 Motivation - Network Congestion

Congestion occurs when the incoming rate of packets exceeds the nodal processing rate. As a result, packet loss as well as service degradation will occur. The reason behind the packet loss is the finite memory size of the intermediate nodes of a network. Additionally, the lost packets may require retransmission and consequently the network latency may increase while the QoS decreases. An example of network congestion can be seen in Figure 1.2.

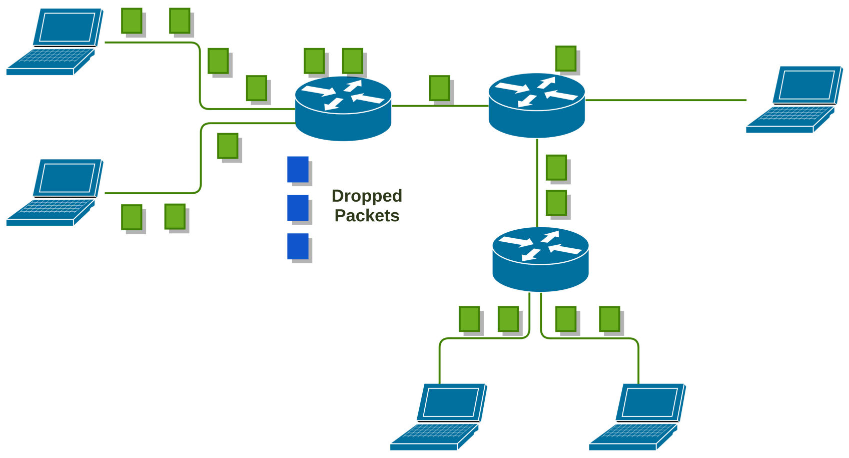


Figure 1.2: A Simple Packet-Switched Network Suffering from Congestion

Precisely, the buffer of an intermediate node will be able to hold a finite number of packets, resulting in excess packets being dropped. Additionally, increasing the size of the memory of an intermediate node will not solve the problem but rather will induce further congestion. The increase in memory size will allow a larger queue of packets to form. The larger queue will result in packets being queued for a longer period of time. If the packets are queued for long enough to time out, a retransmission will be required [8]. Therefore the congestion problem must be mitigated in a different manner.

The following section contains a brief introduction to the proposed solutions.

1.3 Proposed Solutions

To mitigate the congestion problem, two compression techniques are proposed. Both techniques are designed to be adaptive to network conditions to ensure a higher performance. The following subsections contain information about the proposed compression schemes.

1.3.1 Adaptive Compression and Decompression Scheme Based on Real-Time Network Feedback

An adaptive compression scheme is proposed where the compression and decompression processes occur within the end-hosts. In this scheme, the end-hosts determine whether to enable or disable the compression process based on a certain network value known as round-trip time (**RTT**). In simple terms, the RTT value is continuously observed and compared to another value. Once the RTT value exceeds the other value by a certain threshold, the source end-host will activate the compression scheme and commence the process of encoding data.

1.3.2 Adaptive Distributed Compression and Decompression Scheme Utilizing Intermediate Network Nodes

An adaptive compression scheme is proposed where the compression and decompression processes take place within the intermediate nodes[2] of the network. The intermediate nodes, routing data from one node to another, are responsible for compressing packets' payloads according to the network condition. The network condition is determined by calculating certain values, such as the average waiting time of a packet in the queue of an intermediate node, based on certain queuing theory models. In this compression and decompression scheme, the source-destination pairs are alleviated from the processes of encoding and decoding data.

1.4 Thesis Organization and Contribution

The main contribution of this thesis is the design and validation of the proposed compression schemes over networks utilizing the IEEE 802.3 standard for packet transmission in local and wide area networks. The testing and validation process of the proposed schemes is completed using a popular open source simulation platform known as Network Simulator 3 (**NS3**).

[2]A router is a type of an intermediate node within a network. Any node within the network responsible for moving packets from one link to the other is an intermediate node.

There are many advantages of using NS3, such as allowing the user to rapidly implement and prototype a network design as well as allowing the user to apply any necessary changes to the platform.

The following points briefly outline the main contributions contained within:

- A novel compression and decompression scheme is proposed where it operates within the end-hosts and employs lossless compression algorithms to maintain data integrity. Also, the proposed scheme contains three different subschemes that are designed for different network types. Additionally, this scheme utilizes a multi-layer feedback value known as the RTT to determine when to activate compression.

- A novel distributed compression and decompression scheme is proposed. The proposed scheme operates by compressing and decompressing packets within the intermediate nodes of a given network. Using queuing theory models and techniques, the intermediate nodes within the network are capable of activating the compression and decompression processes when necessary.

- The proposed schemes reduce network congestion. The reduced network congestion yields lower end-to-end delays and number of packet drops as well as a smoother communication session.

- The proposed schemes are seamless and transparent to the source-destination pairs as they are designed to enhance communication sessions.

The remainder of the thesis is organized as follows:

- Chapter 2 provides a brief description of the Internet protocol stack, i.e., necessary protocols to initiate and complete transmission sessions on IP networks using the IEEE 802.3 standard. Additionally, the different types of delays as well as the compression algorithms are briefly discussed.

- Chapter 3 is the literature review of the other compression techniques available. A brief summary of each existing and relative compression technique is given.

- Chapter 4 describes the design and validation of the RTT based compression technique. Additionally, the algorithm for each of the subschemes is described and presented. Also, the results and discussion of the conducted simulations for all of the sub-schemes are included.

- Chapter 5 describes the design and validation of the distributed compression technique as well as the queuing theory principles used. The results and discussion of the conducted simulations under different network loads are included.

- Chapter 6 is the conclusion of the thesis and primarily consisting of the summary of the accomplished work as well as a discussion of possible future work.

Chapter 2

Background Study on the Internet Protocol Stack, Network Delays, Compression Algorithms, and Modes of Compression

An understanding of the Internet protocol stack and the different types of network delays is essential to mitigating congestion in data networks. Every packet traversing within a data network experiences different types of delays as a result of using the Internet protocol stack as a model to achieve end-to-end communication. Although these delays can be minimized, they are inevitable.

The types of delays are easily categorized as nodal and transmission delays. The network delays are used to calculate important values, including the RTT and several buffer sizes of network devices. It is therefore important to utilize and understand the collected information from the network delays.

In this chapter, an overview of the Internet protocol stack and the delays experienced by a packet is given. The lossless compression algorithms used in this thesis are reviewed. Finally, a brief discussion regarding the concept of the minimum acceptable size of packets is included.

2.1 Internet Protocol Stack Model Overview

The Internet protocol stack is a multi-layered network architecture consisting of at least 5 layers, used to provide end-to-end connectivity between source-destination pairs. Each of the 5 layers performs a set of well-defined services and functions to ensure the connectivity between end-hosts. These services and functions include data formatting, addressing, routing, and transmission, among others.

This communication model is hierarchical; the lower layers provide services for the higher layers. When two end-devices are communicating with each other, the transferred data passes through all of the layers. Depending on the application performing the transmission and the required service type, the relevant protocols are used to initiate and complete the communication session.

For the source, data is generated by an application at the highest layer. The generated data is passed down to the lower layers, where each layer adds a header. The headers contain sensitive information, such as addresses, ports, and sequence numbers of the data being transmitted. The lowest layer, i.e., the physical layer, is responsible for the transmission of the data. At the destination, the received data moves upwards through the layers in order, where the headers are removed as the data progresses [10]. An abstract representation of a functioning Internet protocol stack can be seen in Figure 2.1.

2.2 Internet Protocol Stack Layer Usage

In any given network, communication devices must follow certain protocols to successfully exchange messages. Different network entities and applications demand different types of protocols for various reasons, including: QoS provisioning, congestion control, delay sensitivity, etc. Combining all of the available protocols creates the protocol stack [5].

The Internet protocol stack, starting from top to bottom, consists of at least 5 layers: application, transport, network, data-link, and physical. Each layer provides different services and

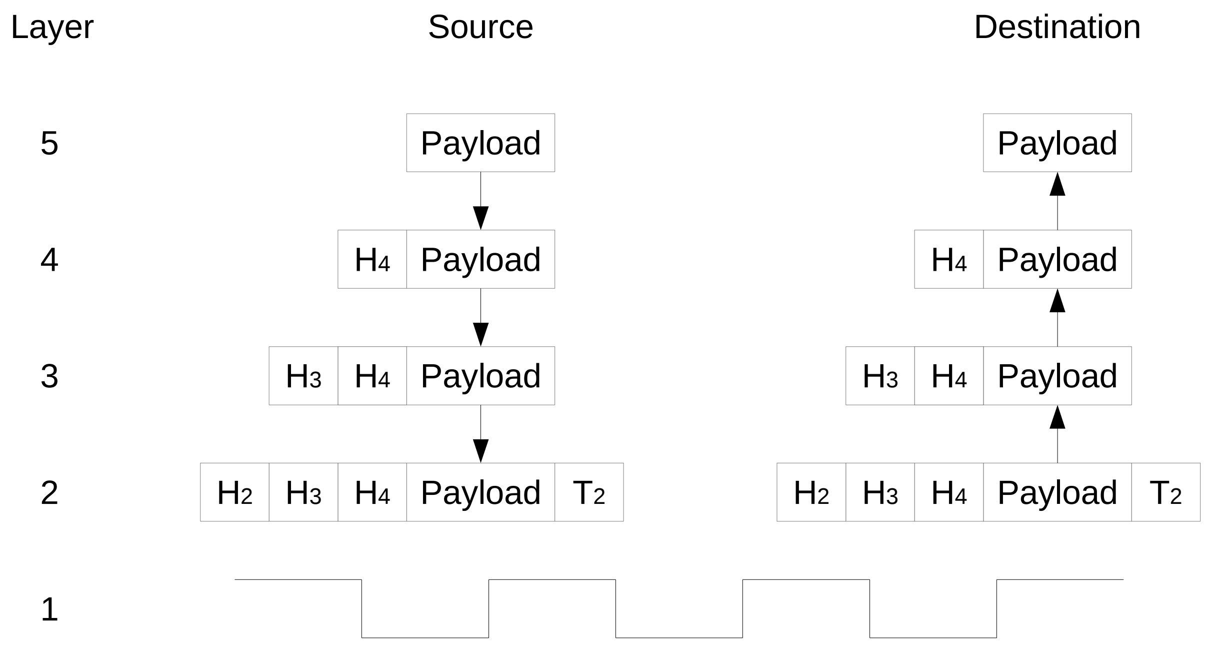


Figure 2.1: Vertical Representation on the Usage of the Internet Protocol Stack

protocols for network entities to use. The Internet protocol stack can be seen in Figure 2.2.

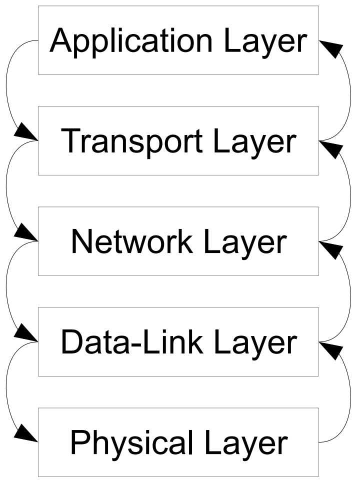


Figure 2.2: The Internet Protocol Stack Layers Reference Model

When a source sends a message, the message is encapsulated with different headers from different layers in order for the message to reach its destination. At various points throughout the network, such as routers and switches, some headers are stripped and new ones are attached prior to the message reaching its destination. Each header of the different layers contains information about the message including: application, source address, destination address, source port number, destination port number, payload size, etc. [5].

The following sections discuss the different layers of the Internet protocol stack in detail, starting with the uppermost layer.

2.3 Application Layer

Internet applications such as voice-over-Internet-protocol (**VoIP**), peer-to-peer (**P2P**) file sharing, emails, World Wide Web (**WWW**), and multi-player online gaming are the motivating force behind the expansion of the Internet. Internet applications are built to run on end-hosts; the user therefore does not have to write any software for the routers and switches within the network in order for the application data to reach its destination [11].

One of the main kinds of application architectures is known as client-server architecture. In this architecture, a fixed, dedicated, and always available server with a static, known IP address is responsible to service requests from clients. Additionally, clients do not directly communicate with each other; the server is responsible for transferring data between the clients. Moreover, such that they can respond to a large number of requests efficiently, servers are usually housed in large data centers [11].

The client and server exchange requests and messages over the network using sockets. A socket is the application programming interference (**API**) between the application layer, transport layer, and network. A socket allows hosts and processes to exchange messages over a network by identifying each other using IP addresses and port numbers, respectively [11].

The structure of the exchanged messages between the client-server applications is defined by an application layer protocol. In some cases, the application layer protocol will add its own header to the generated data. An example of an application layer protocol is HyperText Transfer Protocol (**HTTP**), which is widely used by the WWW [11][12].

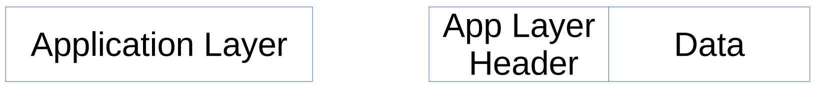


Figure 2.3: Application Layer Message Formulation by Header Encapsulation of Data

2.4 Transport Layer

Underneath the application layer lies the transport layer. The primary purpose of this layer is to provide logical, efficient, end-to-end communication services between the application layer processes of different end-hosts [13]. To achieve end-to-end communication, the transport layer provides two main protocols.

Two distinct protocols with different properties are present in the transport layer and they are used for transferring segments of data. The first protocol is known as User Datagram Protocol (**UDP**) and the second one is Transmission Control Protocol (**TCP**). It is known that UDP is unreliable and connectionless whereas TCP is known to be connection oriented and full duplex. Furthermore, UDP and TCP provide data integrity check by including a pre-transmission error detection header field known as the checksum [14].

2.4.1 TCP Operation

The services provided by the TCP protocol include flow control, reliable data transfer, transmission flow multiplexing, retransmission of packets, and security measures. The TCP protocol pushes and forwards data when necessary while assuring the sending and receiving parties that the transmitted data is not damaged, lost, duplicated, or delivered out of order. To provide the aforementioned services, TCP uses a segment sequence numbering system as well as an acknowledgement (**ACK**) mechanism where the receiving end ACKs the received segments [15].

To initiate a TCP connection, a three-way handshake process takes place, as shown in Figure 2.4. This is done by the client setting the SYN field in the TCP header to 1 and sending the SYN segment to the server. Once the server receives the SYN segment, the server allocates the receiving buffer and replies to the client with a SYNACK segment. Finally, the client allocates a receiving buffer and sends an ACK segment to the server. A similar process is used to terminate the connection, however the FIN field is used rather than the SYN field [15]. The TCP header structure is shown in Figure 2.5.

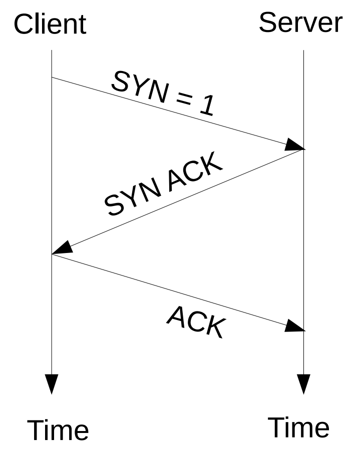


Figure 2.4: TCP Three-Way Handshake Process

While using TCP, the receiver organizes segments, ACKs data, and discards duplicates according to the segments' sequence numbers. Furthermore, the receiver uses a checksum mechanism to identify damaged segments. Precisely, the receiver verifies the checksum field in the TCP header by computing the checksum value [15].

The ACK messages are also used for flow control by informing the sending party of the receiver's buffer size. Along with the ACK message, the receiver's buffer window size is returned to the sender whilst indicating the next acceptable packet sequence number [15]. This is the receiver's method of informing the sender which packet is expected.

The sequence number assigned to segments and their ACKs work together to provide reliable data transfer. There are header fields specifically dedicated to sequence numbering and

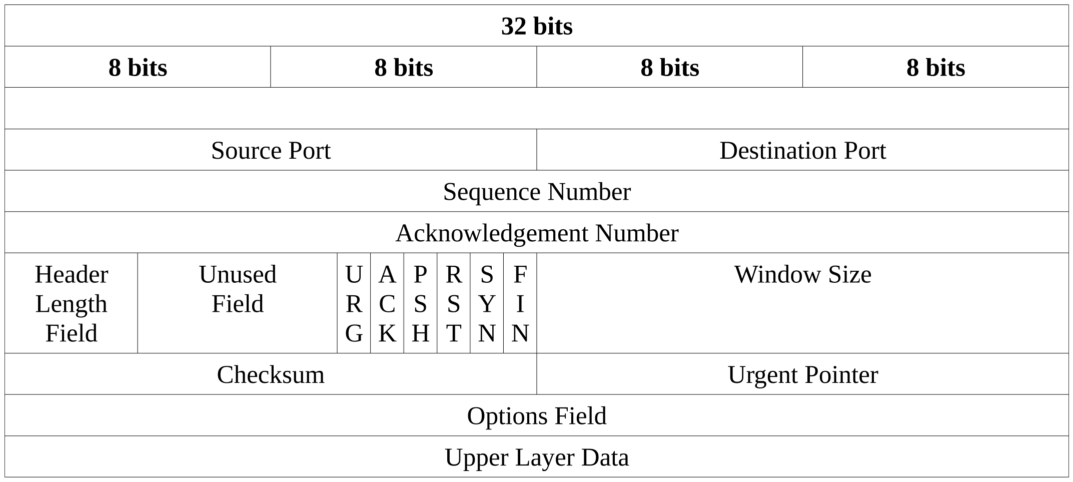


Figure 2.5: TCP Header Structure

ACKing of segments. The sequence numbering field assigns numbers to the first byte in each transmitted segment. For example, if the data size is 10,000 bytes whereas the maximum segment size (**MSS**) is 1000 bytes, the first segment is assigned a sequence number of 0, the second segment is assigned a sequence number of 1000, and so on. The ACK field is defined by the receiver and its value is set to be the sequence number of the next expected byte. For example, if the receiving end-host received all bytes from 0 to 999, the receiver will set the ACK field to 1000 and send it back to the sending end-host. Using ACKs and sequence numbers, the sender can identify which packet is lost or damaged and perform a retransmission if necessary [16].

When data is transmitted over a network using TCP, the receiver has a buffer window where the application will read the data from. The application does not necessarily read the data in the buffer instantaneously. However, to prevent the possibility of overflowing the buffer, TCP provides a flow-control service where the speed of the application reading the buffer is matched to the speed of the data being received. This is accomplished by having the sender maintain a *receive window* value, which is used to indicate the free buffer space available at the receiving end-host. The receiving window is defined by equation (2.1).

$$rwnd = RcvdBS - [LBRead - LBRcvd], \tag{2.1}$$

where *rwnd* is the receive window size, RcvdBS is the allocated receiving buffer size, LBRead is the number of the last byte read by the application in the buffer, and LBRcvd is the number of the last byte received by the buffer.

On the other side of the flow, the receiver keeps track of the last byte sent and the last byte ACKed. By keeping the difference of these two variables less than *rwnd*, the sender will not overflow the receiver's buffer [16]. The relationship of the variables is defined by equation (2.2):

$$LBSent - LBACKed \leq rwnd. \tag{2.2}$$

Along with keeping track of the *rwnd* size, the sender keeps track of another variable that defines the congestion widow size, also known as *cwnd*. The *cwnd* value is defined by the sender as the perceived network congestion where network congestion refers to the buffers of the intermediate nodes, such as routers and switches. The value of the *cwnd* affects the protocol's transmission rate, that is whether the sender should increase or decrease the rate [16]. Equation (2.3) defines the allowed amount of unacknowledged data with respect to the size of *cwnd* and *rwnd*.

$$LBSent - LBACKed \leq min\{cwnd, rwnd\} \tag{2.3}$$

In order for the TCP protocol to accomplish the aforementioned services, a header is attached to the upper layer data and the packet will be referred to as a segment, as shown in Figure 2.6.

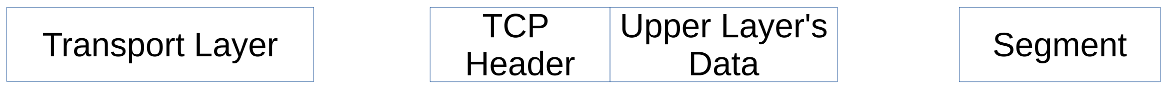


Figure 2.6: Transport Layer Segment Formulation by Header Encapsulation of Upper Layer's Data

2.4.2 UDP Operation

The UDP protocol is a much simpler protocol when compared to TCP. It is a connectionless protocol with a very small overhead. It does not provide any congestion control, ACKing methods, sequence numbering, or retransmissions and it assumes that the IP provided by the networking layer is used.

32 bits	
16 bits	**16 bits**
Source Port	Destination Port
Length	Checksum
Upper Layer Data	

Figure 2.7: UDP Layer Header Structure

2.4.3 TCP and UDP Checksum

Both TCP and UDP have an error detection mechanism known as verifying the checksum. The checksum field holds a 16-bit value and is used to detect errors such as any changes to the header segment as the segment moves from the source to the destination. The value of the checksum is calculated at the sender by adding all 16-bit words in the UDP or TCP header, such as the source and destination port numbers, followed by calculating the sum's 1s complement. Therefore, the 1s complement will be the value in the checksum field. On the receiving end, the receiver will verify the checksum by adding all of the 16-bit words in the header as well as the received checksum value. If there are no errors, the sum at the receiver's side should be

1111111111111111.

2.5 Network Layer

The application and transport layers rely on the forwarding and routing services of the network layer to provide communication means between hosts. The forwarding function is used by routers to direct incoming packets on to the appropriate output path using forwarding tables. A routing function is used to determine the path taken by packets while traversing between a source-destination pair [17].

The network layer provides the IP protocol as service for the upper layers. When an upper layer segment is encapsulated with IP version 4 (**IPv4**) header, it is known as an IPv4 datagram [18]. Figure 2.8 shows the network layer encapsulation of data passed from the transport layer.

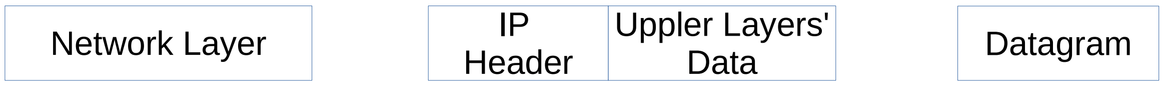


Figure 2.8: Network Layer Datagram Formulation by Header Encapsulation of Upper Layers' Data

The contents of the IPv4 header are shown in Figure 2.9. It can be seen in the IP header that there are some important fields directly related to the size of the data. These fields are: total length, datagram fragmentation related fields, and header checksum.

The total length field is a 16-bit field that includes the entire datagram length. The header length field is specific to the size of the IP header [17]. According to equation (2.4):

$$TotalLength - HeaderLength = Data, \tag{2.4}$$

where subtracting the header length field from the total length returns the data size. Since the total length field is 16 bits, theoretically the maximum datagram size is 65,535 bytes [17].

32 Bits							
8 Bits		**8 bits**		**8 Bits**		**8 Bis**	
Version	Header Length	Type of Service		Total Length			
Indetifier					DF	MF	Fragment Offset
Time-To-Live		Protocol		Header Checksum			
Source Address							
Destination Adress							
Options							
Upper Layers' Data							

Figure 2.9: IPv4 - Network Layer Header Structure

The fragmentation related fields are: identifier, flag fields, and fragment offset. The datagram is fragmented because the data-link layer frames have a maximum transmission unit (**MTU**) of 1500 bytes. Therefore, large payloads must be fragmented and identified using the 16-bit identifier field. The sender increments the value of the field with every transmitted packet. Furthermore the receiver uses the identifier, flag, and fragment offset fields to reassemble datagrams [18].

There are two 3-bit flag fields available for use. From Figure 2.9, the DF and MF fields are known as $do\ not\ fragment$ and $more\ fragments$, respectively. If the DF field bits are set to 0, it indicates that the datagram is not fragmented. If the packet cannot be fragmented, the intermediate routers will discard it where a reply message is sent to the sender stating that fragmentation is needed. If the MF field bits are set to 1, it indicates that more fragments are expected. The receiver expects to receive packet fragments until the MF field bits are set to 0 [18].

Finally, fragment offset is a 13-bit field used to identify where the fragment belongs when considering the entire datagram. The offset is considered from the header of the original datagram in chunks of 8 bytes [18].

2.6 Data-Link Layer

The primary purpose of the data-link layer is to frame datagrams received from the network layer by adding a header and a trailer. The data-link layer is also used to establish a virtual link for the network layer of the source end-host to the network layer of the destination end-host.

When a datagram reaches the data-link layer from the network layer, the datagram is framed by concatenating a header to the beginning of the datagram as well as a trailer to the end of the datagram, thus forming a frame. The newly formed frame is now transmitted over a physical medium via the physical layer [19] [20].

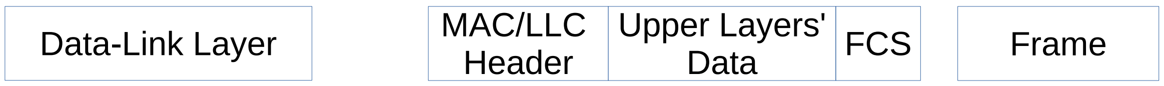


Figure 2.10: Data-Link Layer Frame Formulation by Header and Trailer Encapsulation of Upper Layers' Data

8 Bytes	**6 Bytes**	**6 Bytes**	**2 Bytes**	**46 to 1500 Bytes**	**4 Bytes**
Preamble	Dest. Address	Src. Address	EtherType	Data	FCS

Figure 2.11: Data-Link Layer Frame Structure

The data field in Figure 2.11 contains the datagram from the layer above where the MTU is 1500 bytes. Therefore, if the data size is greater than 1500 bytes, the data is fragmented. Furthermore if the data is less than 64 bytes, the data is padded with zeros [19]. The encapsulated data in the data field is checked for errors by a receiver using the frame check sequence (**FCS**) value in the FCS field. The FCS is a calculated value based on the bits in the data field [21].

2.7 Physical Layer

The final layer in the Internet protocol stack is the physical layer. It is responsible for physically moving transmitted data, bit-by-bit, across a link, from a network device to another. This is achieved by varying the voltage or current across the transmission medium.

There are network devices that only use the physical layer, such as repeaters and hubs. The job of such low-layer devices is to accept incoming bits and forward them to the right output [22] [23].

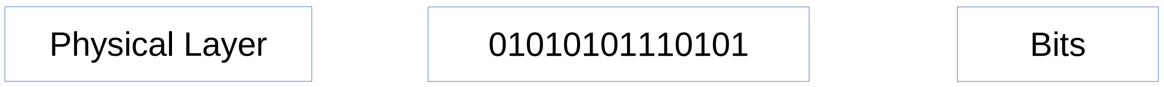


Figure 2.12: Physical Layer Encoding and Transmitting Data in the Form of Bits

2.8 Network Delays

In any packet-switched network, all of the transmitted packets suffer from primarily four types of delays including: nodal processing delay, queuing delay, transmission delay, and propagation delay. These delays are inevitable, and each packet experiences all of these delays at every node along its journey from a source to a destination. Such delays are summed together and referred to as the total nodal delay.

- Nodal Processing Delay:

  As previously mentioned, every packet has a header and a payload that are attached together and transmitted as a whole. For an intermediate node, such as a router or a switch, to determine the path the packet will take, the node must process and examine the packet's header. In certain situations, other processing takes place, such as error checks. The time needed to examine and process the packet is known as the processing delay [5].

- Queuing Delay:

  Intermediate nodes have the capability of buffering packets in case the network is congested. If a packet is queued, the time the packet waits until it is placed on the link is known as the queuing delay. The time that the packet waits in the queue depends on the number of packets that are located within the queue. If the queue is empty, the queuing delay will be minimal, however, if the queue has a substantial number of packets, the queuing delay will be large [5].

- Transmission Delay:

  Transmission delay is time needed to transmit or push a packet into a link. This type of delay is directly proportional to the size of the packet as well as the speed of the link. The delay is defined according to equation (2.5).

  $$T_{trans} = \frac{L}{R}, \tag{2.5}$$

  where L is the size of the packet in bits and R is the transmission speed of the link from one node to another [5].

- Propagation Delay:

  Propagation delay is defined by the physical distance a bit must travel from one node to another. Depending on the medium used to transmit the data, such as fiber optics cable or twisted pair copper wire, the propagation speed will differ. The propagation speed will range from approximately $2 * 10^8$ to $3 * 10^8$ meters per second. The propagation delay is defined according to equation (2.6).

  $$T_{prop} = d/s, \tag{2.6}$$

  where d is the distance between two nodes and s is the propagation speed of the bit [5].

- Nodal Delay:

  Thus, the total delay for a packet traveling from one node to another is defined according to the following equation:

$$T_{nodal} = T_{proc} + T_{queue} + T_{trans} + T_{prop}, \tag{2.7}$$

  where T_{proc} and T_{queue} are usually neglected due to their minute effect on the total delay [5].

- Round-Trip Time (RTT):

  RTT is the total time needed for a packet to travel from a source node to a destination node and back to the source node [6]. However in this thesis, the definition of RTT will be slightly changed to the total time needed for a packet to travel from a source node to a destination node in addition to the length of time needed for an ACK to travel from the destination node to the source node.

- Latency:

  Latency is the length of time it takes a packet to travel from the source to the destination node [7]. Table 2.1 contains sample latency values for a 1500 bytes packet. The queuing and processing delays are ignored due to their minute effect on the total time. The propagation speed considered was $2/3 * 3 * 10^8 = 2 * 10^8\ m/s$. To calculate the latency for the different scenarios, the following equation was used:

$$Latency = T_{trans} + T_{prop}. \tag{2.8}$$

Distance	Propagation Delay	Link Speed	Transmission Delay	Latency
1 km	5 μs	1 Mbps	12 ms	12.005 ms
		10 Mbps	1.2 ms	1.205 ms
		100 Mbps	0.12 ms	0.125 ms
10 km	50 μs	1 Mbps	12 ms	12.05 ms
		10 Mbps	1.2 ms	1.25 ms
		100 Mbps	0.12 ms	0.17 ms
100 km	500 μs	1 Mbps	12 ms	12.5 ms
		10 Mbps	1.2 ms	1.7 ms
		100 Mbps	0.12 ms	0.62 ms

Table 2.1: Sample Latency Values for Different Networks of Various Speeds and Distances. The Size of Each Packet is 1500 bytes.

2.9 Compression Algorithms

There are two primary types of data compression, particularly, lossless and lossy algorithms. In lossy compression, it is acceptable to have an encoded output file that is slightly different than the input. Furthermore, lossy compression algorithms are primarily used with audio and video files where the quality of the image or video is directly affected by the compression level. Usually, higher quality images have a lower compression level.

In lossless compression, the purpose of the algorithm is to reduce the number of bits required to represent information through different means of encoding and decoding. Furthermore, lossless compression guarantees the integrity of the original information, thus when the encoded data is decoded, the decoded data is an exact replica of the original data [24].

In this work, all of the algorithms used are lossless compression algorithms. The compression algorithms used are Lempel-Ziv-Oberhumer (**LZO**) and ZLIB. LZO is a lossless compression algorithm based upon the original Lempel-Ziv (**LZ77**) compression algorithm. LZO is also known to be an algorithm that favors speed over compression ratio whereas ZLIB prefers compression ratio over speed [25]. Moreover, compression ratio is defined according to equation (2.9).

$$C = \frac{CDS}{UCDS}, \tag{2.9}$$

where CDS is the compressed data size and $UCDS$ is the uncompressed data size [26].

The following subsections provide a brief overview of each algorithm.

2.9.1 Lempel-Ziv Compression Algorithm

Lempel-Ziv is a universal lossless compression algorithm developed in 1977 by Abraham Lempel and Jacob Ziv. This algorithm does not rely on any prior knowledge of the source statistics and depends on a learning process for discrete source characteristics [27].

LZ77

The LZ77 is a lossless dictionary-based compression algorithm that utilizes the longest matching string concept and a sliding window approach to encode data [25]. The algorithm treats an input string of bytes as two main parts where the large part consists of decoded data and the smaller part consists of a lookahead buffer.

The decoded data is used as a dictionary for the lookahead buffer, where the algorithm looks for matches between the two parts. Once matches are found, output tokens are created pointing to the location, length, and the first character seen in the lookahead buffer [24]. In other words, the algorithm reads and analyses a string of data where redundant information is replaced by tokens containing information on how to retrieve the original information from the encoded data.

There are important terms one must know to understand how compression works. These terms include: input string, coding position indicator (**CPI**), lookahead buffer, window, and output. The following is the list of the terms with their respective definitions.

- Input string:

  The input string is the string of bytes that is going to be encoded.

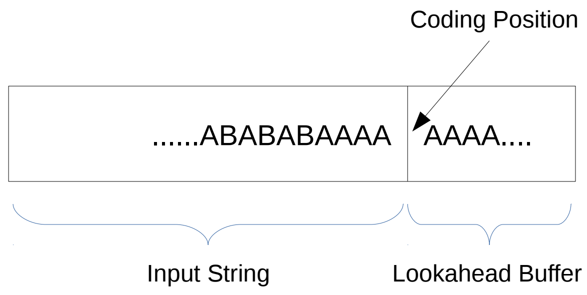


Figure 2.13: LZ77 Sample of a Window and Lookahead Buffer Separated by a Coding Position Indicator

- CPI:

  CPI is the point where the lookahead buffer begins. The coding position indicator can be seen in Figure 2.13.

- Lookahead buffer:

  The lookahead buffer contains bytes from the input data located after the CPI.

- Window:

  The size of the window is used to indicate the number of bytes considered prior to the coding position indicator. The window is empty at the beginning of the compression process. As the input string is processed, the window will grow in size until reaching a size of w. When the window is of size w, the sliding window approach occurs. The window will slide according to number of bits encoded from the lookahead buffer, placing the coding position indicator in a new location.

- Output:

  The output of the encoding process is mainly a pointer in the format of (L_B, L) with a terminating character $C(c)$. The output is represented by $(L_B, L)C(c)$, however, it is not necessary to always include $C(c)$.

  The L_B term is a value indicating the number of bytes the decoder should go back in the window. The L value is the number of bytes the decoder should copy after reaching the position indicated by L_B. The terminating character $C(c)$ indicates the first byte in the lookahead buffer located after the match indicated by the pointer.

In the event of no matches being found between the lookahead buffer and the sliding window, the output is a null pointer concatenated with $C(c)$.

The following is an example of encoding a string of data using LZ77 approach. The string of data is shown in Table 2.2.

Position	1	2	3	4	5	6	7	8	9
Byte	A	A	B	C	B	B	A	B	C

Table 2.2: LZ77 Example - Sample Input String

The encoding process begins by reading the input string of bytes from Table 2.2. The coding position indicator is placed at the first location, position 1, and is continuously moved. If there are no matches, the output will be a null pointer with the terminating character; however, if a match is found, the output will be a pointer with the terminating character. Table 2.3 shows the output of the encoding process.

Step	Position	Match	Terminating Character	Output
1	1	–	A	(0,0)C(A)
2	2	A	–	(1,1)
3	3	–	B	(0,0)C(B)
4	4	–	C	(0,0)C(C)
5	5	B	–	(2,1)
6	6	B	–	(1,1)
7	7	ABC	–	(5,3)

Table 2.3: Encoding Process of the Input String Resulting in An Output of Tokens Indicating Matches Within the Original String

The decoder processes the output of the encoder and appends the bytes together in order. More precisely, the output pointer and terminating character of the encoding process are now used as an input for the decoder to retrieve the original data from the encoded data. Table

2.4 shows the process of decoding data and how the output of Table 2.3 is used as an input to retrieve the original string.

Step	Input	Append Bytes	Output String
1	(0,0)C(A)	A	A
2	(1,1)	A	A A
3	(0,0)C(B)	B	A A B
4	(0,0)C(C)	C	A A B C
5	(2,1)	B	A A B C B
6	(1,1)	B	A A B C B B
7	(5,3)	ABC	A A B C B B A B C

Table 2.4: Decoding Process where the Output of the Encoder is Treated as an Input

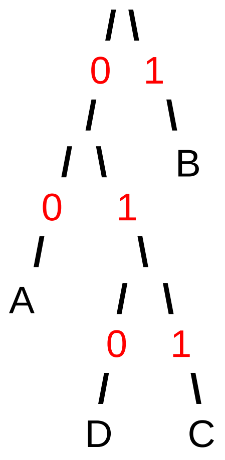
As seen from Tables 2.3 and 2.4, an input string was encoded using the sliding window and longest matching string approach. The output of the encoder was a series of pointers indicating the matches, if any, and the terminating character. The decoder processed the output of the encoder to retrieve the original string [28]. It can be seen that the retrieved string from Table 2.4 was an exact replica of the input string presented in Table 2.2.

2.9.2 ZLIB

The ZLIB compression algorithm is a lossless algorithm that uses the DEFLATE compressed data format for compressing an input string of data. More precisely, the DEFLATE algorithm utilizes a combination of the LZ77 algorithm and Huffman encoding [29], thusly achieving a high compression ratio, when compared to LZO [31]. Furthermore, the algorithm functions by dividing the input string into blocks and processing each block separately. Having the blocks processed separately results in each one having its own independent series of Huffman trees. However, the LZ77 algorithm will search for matches between the current block and previously encoded blocks.

The output of the compression algorithm consists of encoded blocks where each block has two main parts. The main parts are two Huffman code trees, which are used to provide details about the encoded data as well as the actual encoded data. The encoded data has a similar structure as that of LZ77, where pointers are used to indicate matching and non-matching strings. To be precise, the pointers are in the format of (L, L_B), where L is the length of the match and L_B is the backward distance the decoder has to move to locate the match. A Huffman code tree is allocated to provide details about the unmatched strings as well as L whereas the other tree is used to provide information about L_B [29].

Prefix coding is used to assign each symbol from a given alphabet a sequence of bits to use as means of representation. The sequence of bits, also known as codes, are unique per symbol where the decoder may parse the the encoded symbols by analyzing the bits from the encoded input. One may think of this procedure as shown in Figure 2.14 .



Symbols	Code
A	00
B	1
C	011
D	010

Figure 2.14: Simple Prefix Coding of Four Symbols

Huffman coding allows the construction of the optimal prefix code tree if the symbol frequency of a given alphabet is known. The optimal tree is constructed by allocating the shortest codes for the symbols with the highest frequency of occurrence. Additionally, bit sequences of the same length are in lexicographical order. Furthermore, shorter bit sequences lexicographically precede longer codes. Thus, the example shown in Figure 2.14 can be rearranged, as shown in Table 2.5. From Table 2.5, one can see that 110 and 111 are lexicographically consecutive.

Symbol	Code
A	10
B	0
C	110
D	111

Table 2.5: An Example of Rearranging the Codes of Symbols

Based on the lexicographical ordering rules, the Huffman codes for an alphabet are defined by using the lengths of the codes for each symbol, thus the lengths of the codes in Table 2.5 are (2,1,3,3). Once the lengths of the codes are given to the Huffman algorithm, the algorithm will proceed by counting the number of codes of the same length. Subsequently, the algorithm will compute the smallest numerical value for each code length. Finally, the algorithm will assign consecutive numerical values for all the codes of the same length starting with the smallest determined value from the previous step.

The following is an example of assigning codes to symbols.

- Assume an alphabet ABCDEFGH with bit lengths (3,3,3,3,3,2,4,4). The algorithm will start by counting how many symbols are represented by a certain number of bits.

Number of Bits	Count
2	1
3	5
4	2

- The algorithm will now calculate the lowest values of the given code lengths.

Number of Bits	Lowest Value
2	0
3	2
4	14

Symbol	Length	Code
A	3	010
B	3	011
C	3	100
D	3	101
E	3	110
F	2	00
G	4	1110

- Finally, the algorithm will assign values to all the symbols.

After the construction of the Huffman trees, the encoded blocks containing the Huffman trees as well as the encoded data are formed. Each block has a 3-bit header specifying whether the current block is the last block or not, as well as whether the block is compressed or not.

2.10 Modes of Compression

The ability to compress an input string using different approaches enables the user to achieve different levels of compression. There are primarily four different approaches to compressing a string of data. Each approach has certain advantages and disadvantages, however, the proper approach must be used to achieve the desired results. The four approaches of compression are: stateless, streaming, offline, and block compression.

2.10.1 Stateless Compression

The stateless compression approach is when packets are compressed and decompressed independently, with absolutely no relationship to any other packet. Stateless compression is also known as packet-by-packet compression where the receiver may perform a decompression process on each packet independently. This approach of compression and decompression is beneficial when packets arrive out of order at the receiving end.

2.10.2 Streaming Compression

The streaming compression approach, also known as continuous compression, is an attractive approach when reliable means of communication are available. Whilst using streaming compression, packets are compressed and decompressed with a degree of interdependence. Each packet is compressed using the history of the previously compressed packets as well as the data within the current packet. Consequently, the receiver must decompress packets in successive order [3].

Since the receiver must decompress packets in successive order, dropped packets will induce a high latency [1] as the receiver waits for their arrival. Furthermore, the receiver must have a rather large buffer, as it may need to store a hefty number of packets until all necessary packets arrive to perform the decompression process [3].

2.10.3 Offline Compression

The offline compression approach is used when compression is performed at the end-hosts rather than the intermediate nodes of the network. This approach takes place at the higher layers of the Internet protocol stack and is capable of achieving higher compression ratios when compared to the other approaches.

In this approach, data is compressed as a complete unit, in blocks, rather than in small chunks. Furthermore, the receiver is not required to store all of the packets in a buffer to perform the decompression process; instead, the receiver will forward the packet to the higher layers for further processing. Finally, it would be redundant for intermediate nodes to perform any sort of compression on the packets' payloads since the payloads are already compressed [3].

[1]Latency in this scenario refers to the total time required to receive and process the data.

2.10.4 Block Compression

The term block compression refers to a conglomerate of data being compressed at once. A lossless compression algorithm with an excellent compression ratio is considered, such as ZLIB, to better explain the difference between block and packet-by-packet compression. When using ZLIB as a compression algorithm, payload compression in blocks will provide a better compression ratio due to the availability of a longer stream of data, which in turn provides better matches of duplicate strings [30]. However, when transmitting the compressed data that is originally compressed as a block, the receiver must wait for the entire stream of data to arrive before processing it.

If the receiver holds on to data before processing it, two major problems arise. The first problem is that the receiver must have a rather large buffer to store the received packets. The second problem is the possibility of dropped packets, which will result in a significant delay [3]. This delay is broken down to processing, decompression, and transmission delays. The dropped packets must be retransmitted or the received data will be rendered useless. Also holding the packets in the buffer while waiting for a retransmitted packet may result in other packets being dropped due to a full buffer.

2.11 Minimum Size for Performing Compression

From [3], packets with a larger payload produce a better compression ratio since the compression algorithm has a longer input string to find matches. To determine the minimum and maximum size of a packet, one must consider two important aspects. The first aspect is the collision detection mechanism where a minimum frame length is needed to determine whether a collision occurred or not. The minimum frame length needed for collision detection depends on the speed of the link as well as the length of the link. The second aspect is the desired compression ratio where if a higher compression ratio is needed, the required size of packet must increase before going through the compression process [32].

Chapter 3

Literature Review on Compression Based Network Congestion Mitigation Solutions

3.1 Introduction

In this chapter, wide and local area network congestion control techniques via compression are discussed. There is a variety of literature available on techniques that perform compression within and outside the network. The following is a review of selected literature.

3.2 Compressing Packets Adaptively Inside the Network

In [26], the authors discussed mitigating induced traffic congestion over IP networks using an adaptive lossless compression technique. This technique was deployed within a given network rather than at the end-nodes to achieve greater compression efficiency and enhance the performance of the network as a whole. Moreover, the authors state that the intermediate relay nodes used to achieve compression within the network were capable of performing advanced computational functions such as packet storage and processing as well as the traditional forwarding function. Furthermore, the authors state that in a non-congested network, data compression may be redundant and there may be a slight risk of performance degradation rather than enhancement.

To avoid the creation of a bottleneck situation within the network, the authors in [26] propose three conditions designed to achieve an effective compression technique.

- All packets from all source nodes are compressed within the intermediate nodes unless the packet has been compressed prior to arriving to the next intermediate node.
- If a packet is eligible for compression, it will only be compressed after meeting a certain criteria.
- Packets are compressed individually, rather than in a block.

The criteria under which a packet is compressed is if an intermediate node receives a packet with a regular sized payload, the intermediate node will calculate the waiting time of the packet in the queue. If the waiting time is greater than a certain instantaneously calculated threshold, the node will compress the packet's payload.

$$W \geq C - (1 - R) * S/B, \tag{3.1}$$

where W is the waiting time of a packet within the relay queue of an intermediate node, C is the time required to perform the compression process of a packet, S is the packet size, B is the bandwidth of the intermediate node's output link, and R is the compression ratio.

If equation 3.1 and the aforementioned conditions are satisfied, the intermediate node will compress the packet, otherwise it will not and it will forward the packet without additional processing.

The proof of concept was conducted on the Network Simulator 2 (NS2) platform with certain necessary and valid assumptions. Additionally, the network topology used was the parking lot topology, as shown in Figure 3.1.

The simulation assumptions were:

- All packets were of equal size, 500 bytes.
- All packets were transferred using UDP.

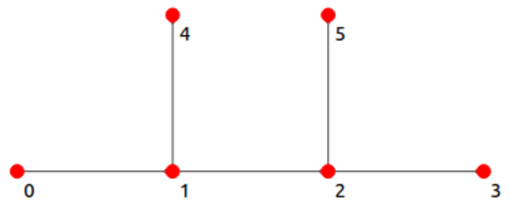


Figure 3.1: Parking Lot Topology

- The compression ratio was constant for all packets.
- The decompression process was ignored due to how insignificant it was in regards to time since LZO was used.

The results of the simulation show that this adaptive compression technique where packets are selectively compressed did indeed improve the network performance by reducing end-to-end packet delay as well as the packet loss rate. However, the authors do not show the optimal number of compressing nodes within the network.

3.3 Adaptive On-the-Fly Compression

In [31], the authors present a novel compression system to improve network performance. The proposed compression system, known as the Adaptive Compression Environment (ACE), harvests and utilizes network information and statistics, such as network bandwidth and CPU load, to predict if compression is worthwhile and to determine the appropriate compression technique to be applied. Once the proper compression technique is chosen based on the gathered network information, the communication between the end-hosts is intercepted where data is seamlessly and adaptively compressed.

To harvest network information, ACE utilizes a network forecasting toolkit to predict whether applying compression is worthwhile or not. The network forecasting toolkit is known as Network Weather Service (NWS) where the output of this toolkit is coupled with ACE to help determine if compression will be used and, if so, which lossless compression algorithm will be used.

To intercept communication between end-hosts and apply the proper compression technique, ACE extended the Open Runtime Platform (ORP) developed by Intel Microprocessor Research Lab with a module capable of performing the aforementioned tasks. The ORP module is capable of seamlessly intercepting TCP/IP communication at the socket level.

The procedure that ACE follows begins with intercepting the transmission socket of an end-host. Once a large enough block of data is being sent (i.e., 32 KB block) ACE determines if the compression is profitable in terms of improvement in transfer performance. To identify whether a packet is compressed or not and which lossless compression algorithm was used, ACE appends a 4-byte header to each 32 KB block.

To determine whether the compression of data will be profitable or not, ACE uses a prediction system to forecast whether to apply compression or not. The prediction system used by ACE is capable of predicting the compression time, compression ratio, and decompression time. Based on the attained values, the transfer time is computed and compared to the transfer time when the data is left uncompressed.

The process of predicting the compression ratio depends on the previous history of the intercepted socket. ACE will use the compression ratio of the previous block as an indicator of the compression ratio of the new block of data. In the case where ACE does not have any history on the compression ratio of the previous block, ACE will consider the CPU loads and overall network performance to determine if compression will occur. Furthermore, the authors generated a linear regression model comparing compression and decompression ratios with compression and decompression time. The linear regression model will be used by ACE to predict the necessary time to perform a compression or decompression process on a given block of data.

To evaluate the effectiveness of the proposed system, ACE was implemented on two distinct networks, fast and slow networks, where the available resources, such as CPU load and bandwidth, in each network varied. The CPU load of the end-hosts was affected by the number of processes running. As the number of processes increases, the CPU load increased. Additionally, the bandwidth of each network was affected by having dummy packet generators send data across the given networks. As the number of packet generators increased, the available bandwidth for data transmission decreased.

The results of the experiments show that ACE is capable of improving network performance in different networks as well as under various network conditions. The authors claim that ACE improves network performance by 8-93% when compared to other compression algorithms. Furthermore, in worst-case scenarios, ACE is capable of improving network performance by 50-90%.

Despite all of the improvements and network enhancement shown by ACE, the proposed technique suffers from one problem: the data is compressed in a block-like fashion. This may induce congestion at the receiver's end when packets are lost within the network. The receiver must wait until the lost packets are retransmitted and received to perform the decompression process.

3.4 IPzip: A Stream-Aware IP Compression Algorithm

In [33], the authors target not only congestion in IP networks but also the process of storing data. The authors propose a novel compression algorithm known as IPzip that is capable of performing compression for improving network transmission as well as increasing the available storage space on end-hosts. Other features of IPzip include capability of providing different compression plans to meet the changes of the network and maintain a certain level of performance.

The process of improving network transmission is referred to by the authors as online compression; furthermore, the process of performing compression to increase storage space is referred to as offline compression. The online compression process is designed to reduce network congestion when sending data from one point to another whereas the offline compression process is designed to reduce the required storage space, if need be.

The two processes have different requirements to achieve what they are designed for. To be precise, the online compression process is designed to be seamlessly applied to a data stream by having a rapid compression time and requiring a small memory space. Due to these stringent requirements, the proposed online compression process will not achieve good compression ratios.

The offline compression process has the complete opposite requirements than that of the online compression process. The offline compression process must be able to achieve good compression ratios, thus short compression time and available memory must not be an issue.

The authors of [33] view network data as a 2-tuple ordered pair of structured and unstructured data where the header of the packet is the structured component of the data while the payload is the unstructured component. The compression of the structured component will be done separately from the unstructured component. Finally, IPzip will identify header and payload correlations as well as reorganize the given data to fully exploit the identified correlations. The compression plan exploiting the correlations is then passed to a compressor known as Gzip to compress the stream of data.

The compression ratio achieved by IPzip is much better than the compression ratio achieved by Gzip alone. However, the problem with this technique is that it compresses data in blocks. Compressing data in blocks may result in performance degradation due to the possibility of packets being dropped within the network. If the receiving end-host does not receive all the necessary components of a compressed block, then the data cannot be decompressed. Thus, the data within the buffer will be wasting resources such as memory space. Furthermore, if packets that contain a component of a compressed block are dropped within the network, the latency of the network will increase due to the receiver waiting for the dropped packets to be retransmitted and received successfully.

3.5 Delayed-Dictionary Compression for Packet Networks

The authors of [3] propose a delayed-dictionary compression (DDC) algorithm for packet networks suffering from congestion. The algorithm compresses packets' payload on a packet-per-packet basis to avoid any decompression-induced latency due to dropped packets. Furthermore, the algorithm creates a dictionary for decompressing the encoded string with a delay of δ units, thus, the decompression process does not depend on the previous δ packets. This delay in the construction of the dictionary diminishes any decompression induced latency while maintaining a relatively acceptable compression ratio.

The proposed algorithm is capable of being tunable to achieve different compression ratios and decoding latencies. This is done by selecting the appropriate parameters for the dictionary delay, δ. Therefore, DDC is capable of achieving a compression ratio that is relatively close to that of a streaming compression and a decoding latency that is relatively close to zero.

Additionally, the process of compression occurs on intermediate network nodes, known as network processors, capable of performing the compression process. The authors set-up the compression scheme in a manner where the packets are compressed at the intermediate node, after the source end-host, whereas the decompression process occurs at the intermediate node prior to the destination end-host, thus achieving end-to-end compression.

The dictionary of the proposed algorithm, as mentioned before, is updated in a delay of δ units where δ is either a unit of characters or packets. In the case where DDC was combined with LZ77, the LZ77 dictionary is shifted in such a way that the compressed string will depend on the characters that precede the last δ characters. In other words, a packet's payload is compressed using older strings that are δ units old. The DDC combined with LZ77 can be seen in Figure 3.2.

Despite a significant improvement in network conditions, the authors do not mention enough details about the network processors. There is a bottleneck situation that may occur at the intermediate node performing the compression process. The bottleneck may occur if the packet generation rate of the source end-host is much higher than the processing rate of the encoder.

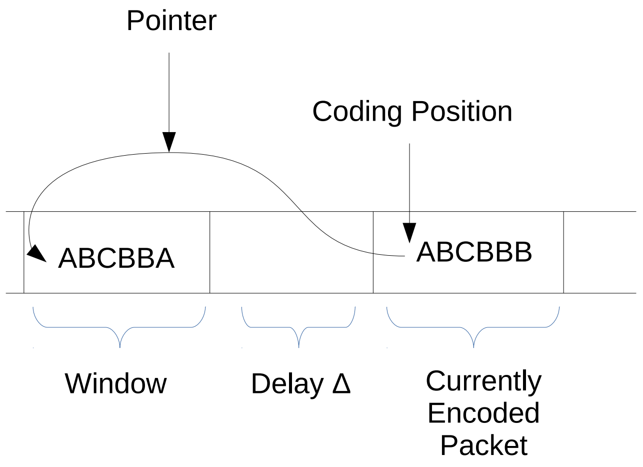


Figure 3.2: DDC Combined with LZ77 with a Delay of δ Units

3.6 Adaptive Online Data Compression

In [4], the authors present an application layer compression technique where both the communication and compression processes communicate indirectly with each other using two threads with independent queues. The size of the queues in the threads are used to determine certain operational methods and compression parameters. The most important component of the proposed technique is the output queue where data is stored prior to sending it.

This technique adaptively selects the level of compression based on the length of the output queue. As the compression level increases, the size of the transmitted data decreases and vice versa. Furthermore, as the queue size increases, the compression level selected by the compression algorithm increases. On the contrary, if the size of the queue decreases, the compression level selected by the compression algorithm decreases. There is a special case when high compression levels are used to compress blocks of data prior to sending them that occurs when the output of the compression process is adequately faster than the sending process.

The algorithm operates by reading files and data prior to the sending process and inserting the data into a compression process, if need be. The compression process forwards the compressed data to a FIFO queue where the compressed packets are stored until it is time to send them. Finally, a communication process is responsible for reading the FIFO queue and sending the data.

It is the length of the FIFO queue that determines the compression level for the next chunk of data to be sent rather than the allocated memory size of the queue. The authors developed the system to continuously monitor the FIFO queue length for changes. The length of the queue is used to directly measure the speed of the network. On one hand, if the queue length is continuously shrinking, it means that the communication process reading the FIFO queue is much faster than the compression process writing to the queue, thus the compression level is continuously reduced. It is possible that the communication process is much faster than the compression process where data compression is no longer necessary. On the other hand, if the queue length is rapidly growing, the compression level is increased.

The proposed algorithm does indeed reduce the time needed to transfer large amounts of data, however the algorithm works by compressing chunks of data rather than individual packets. When chunks of data are being compressed at once and then transmitted, there is the possibility of creating a delay induced by dropped packets. Thus to avoid the possibility of inducing such a delay, one must perform compression on a packet-per-packet basis.

3.7 IP Payload Compression Protocol - IPComp

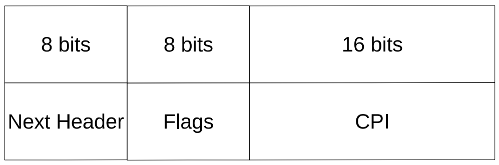
In [34], the authors propose a compression technique for IP payloads. The purpose of the proposed technique is to increase the overall communication performance between nodes. Furthermore, the authors categorize the compression process into two distinct phases. The first phase is the compression of outgoing packets' payload whereas the second phase is the decompression of incoming packets' payload. Additionally, the authors suggest only using lossless compression algorithms such as DEFLATE or LZ to ensure the integrity of the transmitted data. Furthermore, in the case where the size of the compressed packet is larger than the original packet, the original non-compressed packet must be sent.

When using the IPComp protocol, compression is performed on a packet-per-packet basis rather than in chunks to avoid the possibility of inducing delays due to lost packets and their retransmissions. Moreover, the nodes receiving inbound packets with compressed payloads must be capable of processing non-compressed packets as well.

Besides sending the non-compressed packet in the event of an unsuccessful compression, packets must be of a certain size in order to be considered for compression. According to [34], small packets are likely to increase in size rather than decrease after passing through a compression process, thus a threshold of packet size must be chosen. The threshold used in determining the packet size is application dependent.

There are certain features in IPComp that, if implemented, will save time and prevent any delays related to compression. The features suggested by the authors include that if the payload has been compressed by the application layer, the algorithm should skip the compressed payload and forward the data as is. In the case where multiple packets are skipped rather than compressed, the algorithm will decide to skip a certain number of packets without attempting any compression. After a number of packets are skipped, the algorithm will try to compress the payload, if successful; IPComp will no longer skip any packets and will resume compression.

The proposed algorithm inserts a custom IPComp header prior to the compressed payload and modifies the original IPv4 header. The custom IPcomp header is shown in Figure 3.3.



8 bits	8 bits	16 bits
Next Header	Flags	CPI

Figure 3.3: IPComp Header Structure

From Figure 3.3, the IPComp header has 3 major fields, namely Next Header field, Flags field, and Compression Parameter Index (CPI).

The purpose of the fields are:

- Next Header:

  Next Header field is an 8-bit field used to store the next protocol header. Since IPcomp does not compress the header of the payload, the original IPv4 header is stored.

- Flags:

  The Flags field is an 8-bit field set to zero. Currently it is not in use and is reserved for the future. The receiver will ignore this field.

- CPI:

  CPI is a 16-bit field where each range of bits has a well-defined meaning, as shown in Table 3.1.

Bits	Definition
0-63	are used to identify well-known compression algorithms.
64-255	are reserved for future use.
256 - 61439	are used between communicating nodes to define session attributes, such as whether to use IPComp or not.
61440 - 65535	are for reserved for private use between nodes where the decompressing node chooses the CPI value of the session. The chosen CPI value from one decompressing node is independent of the CPI value chosen by the other decompressing node.

Table 3.1: CPI Values and Their Representation

The IPComp protocol may be applied when using different compression algorithms. One may refer to [35] and [36] to see IPComp combined with DEFLATE and LempelZivStac (LZS), respectively.

3.8 Summary

It can be seen from this chapter that there are many compression schemes and techniques available for use to mitigate network congestion. The aforementioned techniques do improve network performance as they all have one common feature, which is reducing the size of packets' payloads adaptively. However there are different problems in the mentioned techniques, such as performing compression on blocks of data rather than on a packet-per-packet basis and possible bottleneck situations. In this thesis, the proposed compression techniques avoid the problems found in literature while achieving comparable results as seen in the following chapters.

Chapter 4

An Adaptive Compression Technique Based on Real-Time Network Feedback

4.1 Introduction

In this chapter, the focus will be on the design and validation of adaptive payload compression techniques where the compression and decompression processes take place at the end-hosts, as shown in Figure 4.1. The purpose of these techniques is to reduce network congestion as well as delays, thus improving communication conditions for real-time and traditional applications. Furthermore, the proposed techniques compress packets' payloads in a packet-by-packet fashion using lossless compression algorithms to ensure the integrity of the data.

It is important to compress packets' payloads in an adaptive manner rather than constantly to avoid the slight possibility of increasing the overall transmission time. Such an event may occur if the compression algorithm is unsuccessful in finding an adequate amount of matches in the input data stream. Furthermore, if compression and decompression processes are enabled, extra CPU cycles are required at the end-hosts. Hence, the proposed techniques are adaptive and active when needed.

The proposed techniques are characterized as three different modes, which are passive, intermediate and active modes. These modes use real-time RTT feedback to decide whether to be active or inactive. Each mode utilizes the RTT values to trigger a sequence of events that involve payload compression. The RTT values are observed during a transmission session where they are used to trigger payload compression, thus improving the condition of a given network. In the event where payload compression is not triggered, the transmission session will remain as is. To identify whether a payload is compressed or not, a flag in the header of the MAC sub-layer[1] is set.

The payload is compressed according to the selected mode where all of the different modes have an initial phase that precedes the transmission phase. In the passive mode, compression is applied based on the history of the previous transmissions of the network. While in the intermediate mode, compression is applied according to a set of conditions. These conditions are related to crossing two different thresholds. The first threshold is the instantaneous RTT (**IRTT**) crossing the baseline RTT (**BLRTT**), which is the RTT of the network when it is congestion-free, and the second threshold is the number of dropped packets exceeding a pre-defined value. Finally, in active mode, compression is applied when the RTT value crosses the BLRTT value.

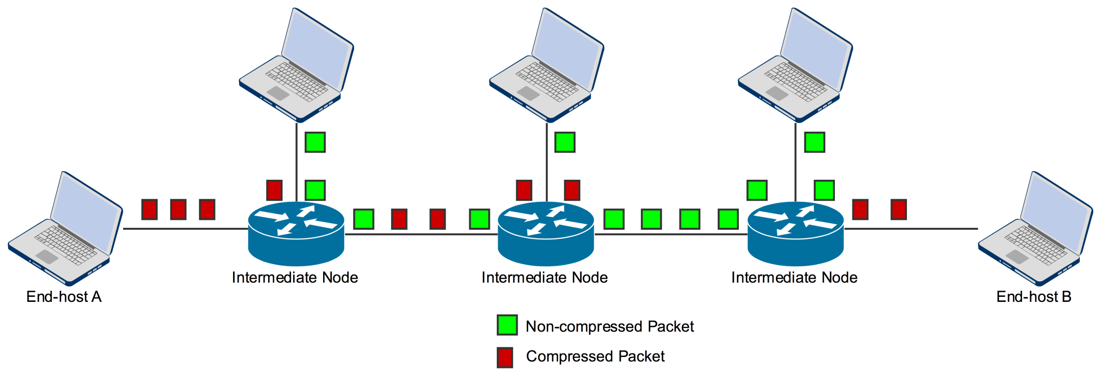


Figure 4.1: Network where the Compression and Decompression of Payloads Occurs Within the End-hosts.

[1]Medium access control (MAC) sub-layer is located within the data-link layer.

4.2 Compressed and Non-compressed Packet Identification

The receiver will be able to identify whether the packet's payload is compressed or not by identifying and using the value of the *EtherType* header field, which is located in the MAC sub-layer of the data-link layer header. The value of the header field indicates the condition of the payload, as shown in Figure 4.2.

The decision tree shown in Figure 4.2 helps the receiver identify the type of compression, if any, that is present. A header will be added to the beginning of a payload, which will include two identification bits. More precisely, the added header will contain the value 00 if the payload is not compressed, and if it is compressed using LZO or ZLIB, it will contain 10 or 11 bits.

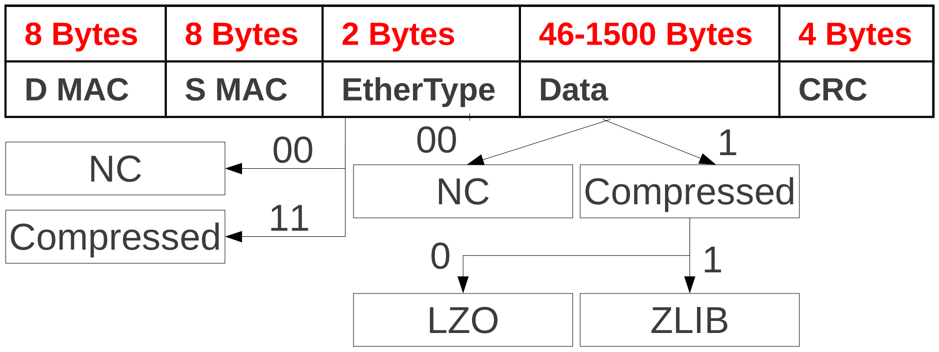


Figure 4.2: Decision Tree to Identify a Packet's Compression Status

The *EtherType* field is an Ethernet protocol parameter that is used by communicating parties as a means of indicating the type of protocol being used in a communication session. For example, the *EtherType* field may contain values to indicate that IPv4 or Internet protocol version 6 (**IPv6**) are being used. The *EtherType* field is 2 bytes large and located after the destination and source MAC addresses [39]. Table 4.1 contains sample EtherType values.

Vlaues	**Referenced Protocol**
0x0800	Internet Protocol Version 4 (IPv4)
0x0806	Address Resolution Protocol (ARP)
0x0808	Frame Relay ARP
0x86DD	Internet Protocol Version 6 (IPv6)
0x880B	Point-to-Point Protocol (PPP)
0x8847	Multiprotocol Label Switching

Table 4.1: Sample EtherType Values

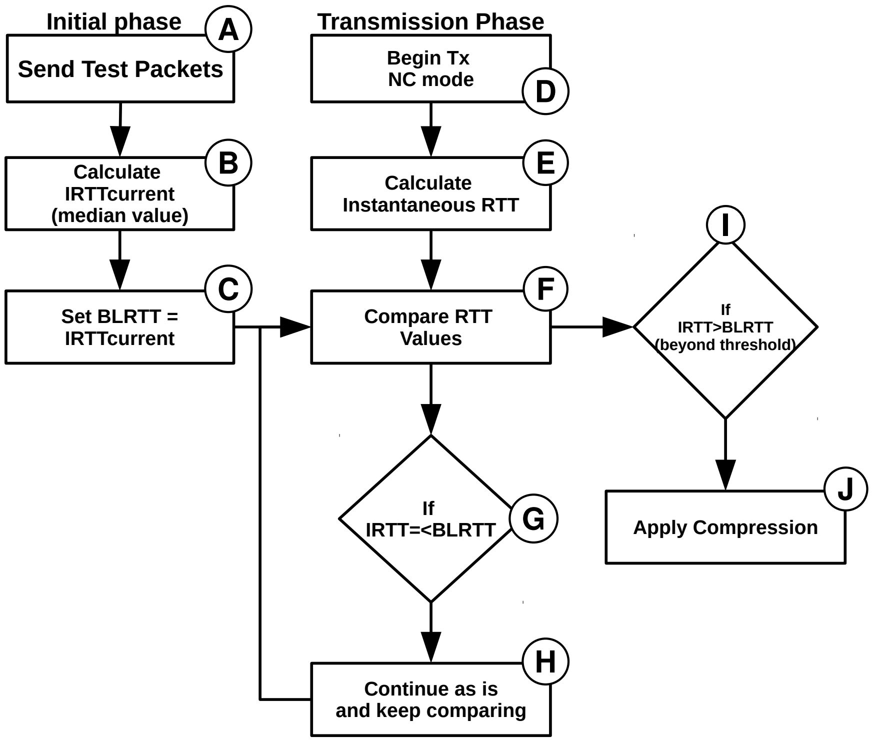
Table 4.1 contains sample EtherType values that are currently in use [39]. The suggested values of 11 or 00 are simply used for illustrative purposes.

4.3 Flow of Operations

The proposed techniques follow different rules and paths for making a decision about whether to compress the payloads of packets or not and which compression algorithm must be used. The following subsections are each accompanied by a flowchart referred to as Figure 4.3, Figure 4.4, and Figure 4.5, respectively. Each figure shows the the behavior of one of the three modes. The purpose of these flow charts is to show the motion of the different modes while transitioning from the initial testing phase to the data transmission phase.

4.3.1 Passive Mode

The passive mode, as shown in Figure 4.3, is a history-based technique in which certain network data is stored and utilized for future transmissions. Knowing the network history gives the future transmissions the necessary edge to improve the communication sessions. Only a fraction of the total data is necessary for this mode.



(A) Is the stage responsible for creating history to be used for calculating the BLRTT reference point. (B) The gathered history is used to calculate IRTTcurrent, which is the average of RTT values. (C) The BLRTT value is set as IRTTcurrent. This value will be used in the transmission phase in the comparison block, block F. (D) The transmission of actual data begins here. (E) The IRTT value is monitored here and compared to the BLRTT value in block F. (F) Is the place where the monitored IRTT value is compared to the BLRTT value from the initial phase. (G) and (I) are the possible outcomes of (F). If (G) is true, the transmission will continue without compression. If (I) is true, compression is applied, either LZO or ZLIB, to compress packets' payloads.

Figure 4.3: Passive Mode's Logic and Method of Operation Flow Chart.

When the passive mode is employed to optimize the conditions of a network, the transmitter is required to do the following:

- Store transmission history:

  The history of the network during a regular transmission or test session is observed and utilized in the future.

- Include early RTT values:

  The history considered is the RTT values of the network from the beginning of the transmission session.

- Demand a large amount of data:

  The history considered, which is used as a sample of the entire transmission session, must be of substantial quantity, otherwise a bigger sample is required.

In order to find out when to apply compression, a new BLRTT value must be set. Figure 4.3 clearly shows that during the initial phase, the BLRTT value is calculated based on observed network history. In the initial phase, multiple test packets are sent over the simulated network. Once the history is observed, the RTT values are stored within an $IRTTarray$ where the values are sorted in ascending order. Only 10% - 15% of the entire array values are used to calculate the required $IRTTcurrent$ value.

The new BLRTT value is equal to the calculated $IRTTcurrent$. In the centre of Figure 4.3, there is a comparator that continuously compares the newly defined BLRTT value against the IRTT value, for which there are two possible outcomes. If the IRTT value is greater than BLRTT, a compression algorithm is employed before transmission, however, if the IRTT value is less than or equal to BLRTT, the algorithm jumps back to the comparison block. In algorithm (1), the relationship between the observed history and BLRTT can be seen.

Algorithm 1 Calculate BLRTT

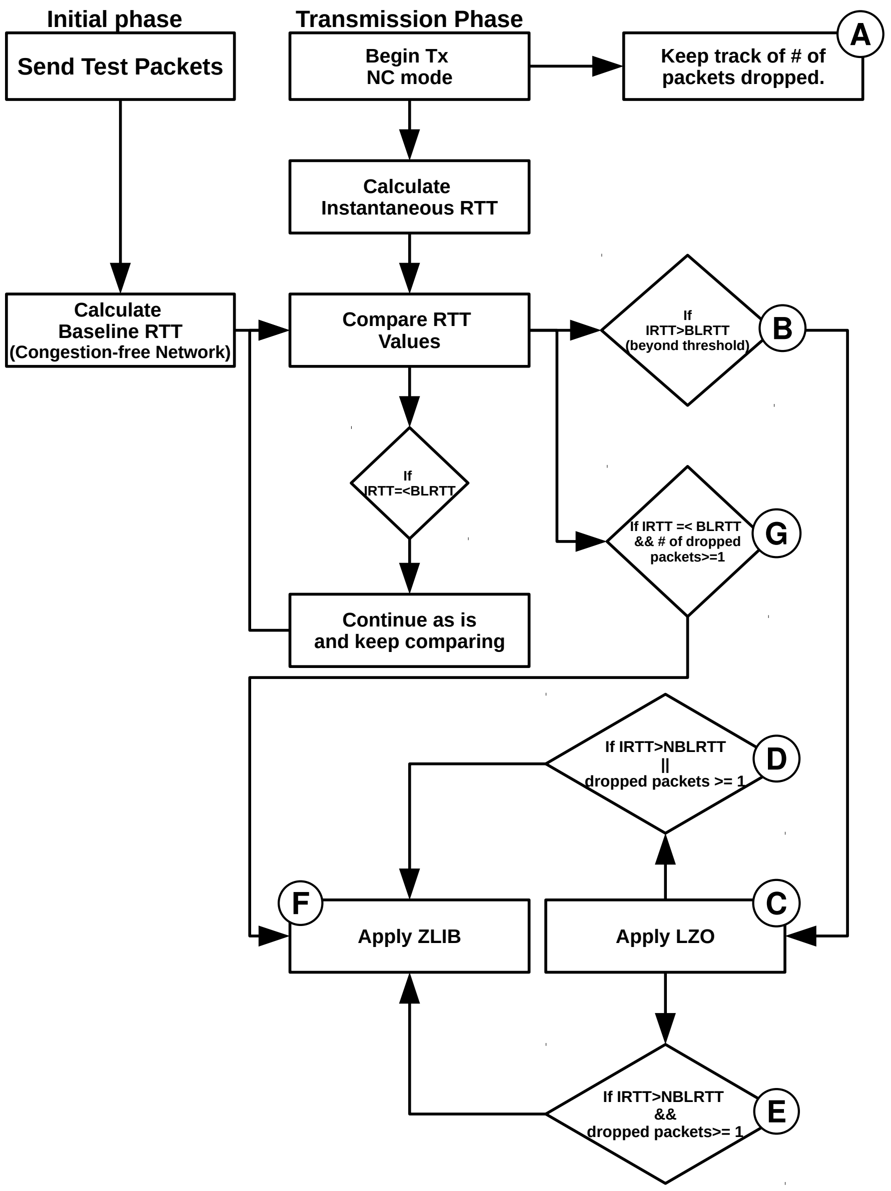
```
1: Create IRTTarray from history
2:     Sort IRTTarray from min to max
3:         Consider 15% of sorted data
4:     Find IRTTcurrent=median(sorted IRTTarray)
5:     Set BLRTT=IRTTcurrent
```

The reason why only 10% to 15% of the entire *IRTTarray* was considered was because of an observed phenomenon while testing the technique: regardless of the number of packets transmitted, the behavior of the network was consistent.

4.3.2 Intermediate Mode

The intermediate mode, as shown in Figure 4.4, is composed of combining both compression algorithms, LZO and ZLIB, in two different methods. The first method merges the mode's conditions similar to an AND logic gate, as in both conditions must be true before any advanced action is taken. The second method merges the mode's conditions similar to an OR logic gate, as in one condition must be true before any advanced action is taken. Both of these combinational methods shall be known as the AND/OR method.

The mode's conditions are watched for after an initial BLRTT is crossed by a certain threshold. As shown in Figure 4.4, the first BLRTT is calculated to be the network's congestion-free RTT value. Once again, the comparison block continuously compares the IRTT value against the initial BLRTT value. If the initial BLRTT is crossed, the LZO compression algorithm is applied to the payload of the packets before transmission. Afterwards, a new BLRTT (**NBLRTT**) is set, and based on the conditions below, the ZLIB compression algorithm is applied to the payload. In the case where the number of dropped packets is greater than a decided value at the same time that the IRTT value is below the initial BLRTT, the ZLIB compression algorithm is applied and the conditions below are ignored.


Initial phase
Transmission Phase
Send Test Packets
Begin Tx NC mode
Keep track of # of packets dropped.
A
Calculate Instantaneous RTT
Calculate Baseline RTT (Congestion-free Network)
Compare RTT Values
If IRTT>BLRTT (beyond threshold)
B
If IRTT=<BLRTT
If IRTT =< BLRTT && # of dropped packets>=1
G
Continue as is and keep comparing
If IRTT>NBLRTT || dropped packets >= 1
D
F
Apply ZLIB
Apply LZO
C
If IRTT>NBLRTT && dropped packets>= 1
E

The initial phase and transmission phase of the intermediate mode perform the same operations as that of the passive mode with additional functionalities. (A) Is the block responsible for monitoring the number of dropped packets. (B) Is the condition that determines if LZO compression should be applied. (C) At this point, all data is compressed with LZO and NBLRTT is defined. (D) and (E) are checking if the conditions, IRTT greater than NBLRTT and the number of dropped packet is greater than a threshold, are met. In (D) the conditions are treated in a manner similar to an OR gate where if one condition is met, ZLIB compression is activated. In (E) the conditions are treated in a manner similar to an AND gate where both conditions must be met for ZLIB compression to be activated. (F) ZLIB compression is applied when this block is reached.

Figure 4.4: Intermediate Mode's Logic and Method of Operation Flow Chart.

The conditions of the AND/OR combinations are respectively:

First IRTT > NBLRTT

AND / OR,

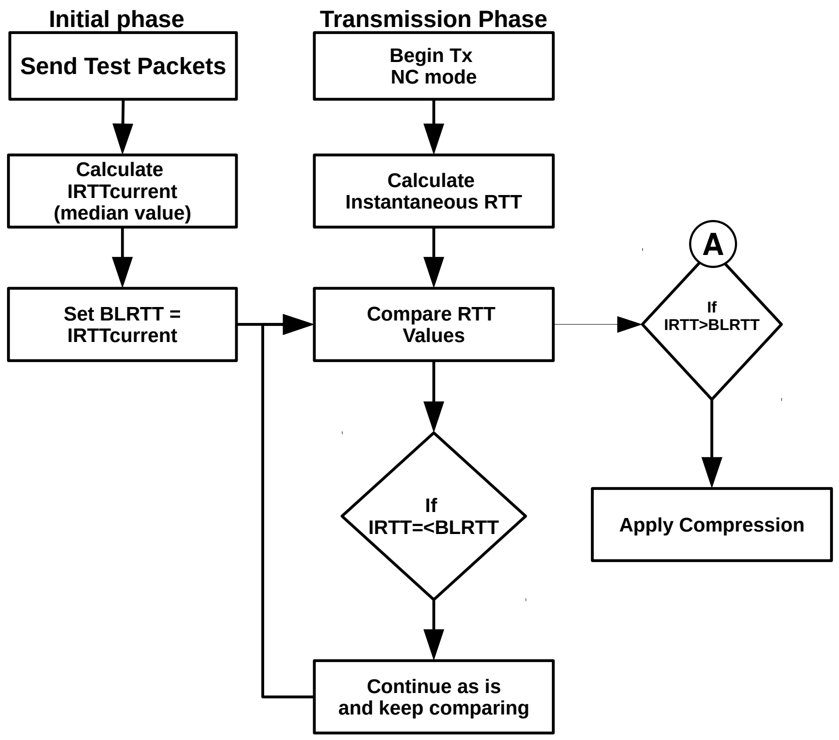
Second The number of packets dropped > certain decided value.

The method that this mode follows is based on progressive stages that are in a ladder-like fashion. The initial phase sets the BLRTT to the value of the RTT of the congestion-free network. Once this value is crossed by a certain threshold, the LZO compression algorithm is applied to the payload of the packets. As the transmission continues, a NBLRTT is configured. This value is based upon the previous history of the network or an arbitrary value.

Depending on the network load, available bandwidth, type of data being transmitted, and payload size, the combinational mode is chosen. On one hand, if there is a network with a large amount of bandwidth available, the AND mode is preferred. On the other hand, if there is a network with a very limited bandwidth, the OR mode is preferred. In fact, in small bandwidth networks, the likelihood of crossing either the BLRTT threshold or the accepted number of dropped packets is higher than a network with a large bandwidth due to the network's minimal tolerance.

4.3.3 Active Mode

The active mode, as shown in Figure 4.5, is the simplest of the proposed techniques. It does not have any tolerance or threshold before activating compression. The mode is initialized by calculating and defining the BLRTT of a congestion-free network or by defining the BLRTT in a similar manner as the passive mode. As the packet transmission begins, the IRTT value is continuously compared to the BLRTT value. If the IRTT value exceeds the BLRTT, compression is immediately activated, otherwise, the transmission session will continue as is.



(A) Unlike passive mode, there is no threshold. Compression is activated as soon as IRTT is greater than BLRTT.

Figure 4.5: Active Mode's Logic and Method of Operation Flow Chart.

4.4 Implementation

There are different approaches to implementing the aforementioned compression and decompression schemes. These approaches include software methods, hardware methods or a mixture of the two. The presented approaches are suitable for a range of network types, hardware operating within a network, and link speeds.

4.4.1 Software Implementation

Any of the aforementioned schemes can be implemented using any conventional programming language, such as C or Python. The software method of operation requires the implementation of the proposed compression and decompression scheme to intercept the communication sessions of different applications at the socket level, as shown in Figure 4.6.

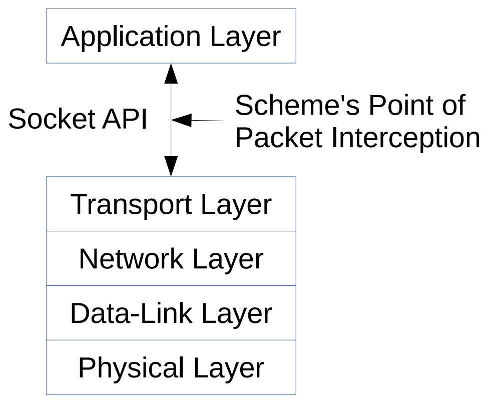


Figure 4.6: The Location where the Implemented Scheme will Intercept Communications to Perform the Compression or Decompression Process

By intercepting the communication sessions of different applications at the socket level, the employed compression and decompression scheme is capable of manipulating the generated or received packets' payloads seamlessly. The scheme process function will be able to compress or decompress the generated or received packets' payloads prior to their transmission or upon their arrival.

In the case where packets are being generated, the employed scheme will compress the packet's payload accordingly. Subsequent to the compression process, the packet headers are formed based on the newly compressed payload. Additionally, a small header is inserted at the beginning of the payload to indicate that the payload went through the compression process and the type of compression algorithm used. Finally, a header field, EtherType, is used to indicate the type of protocol being utilized and is set prior to the transmission process.

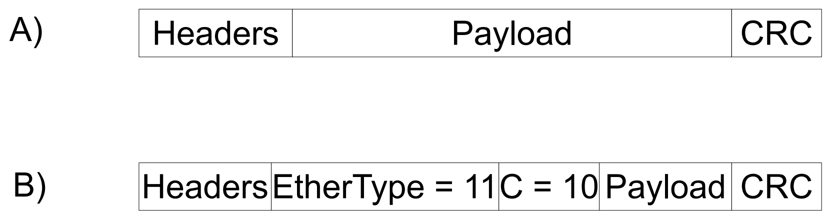


Figure 4.7: A) Is the Packet Structure of a Regular Packet In a System Without Any Compression Scheme. B) Is the Packet Structure After Performing the Compression Operation.

From Figure 4.7, the packet structure of a regular packet and a compressed packet are shown. The difference between Figure 4.7 A and B is the value of the EtherType field and the inserted payload header, C.

In the case where the packets are intercepted subsequent to entering the end-host from the network, the scheme will inspect the incoming packet to see if the decompression process is necessary. The scheme will make this determination based on the query results of the status of the EtherType field. After obtaining the query results, the end-host will further inspect the inserted header by the end-host that performed the compression process, if need be.

If the query results indicate that the payload requires the decompression process prior to reaching the target application, the scheme will proceed by decompressing the payload. Subsequently, the scheme will update the headers of the packet accordingly followed by removing the header inserted by the source end-host.

The software approach is preferably used with end-hosts that have the ability and the freedom to forfeit CPU cycles to perform the compression and decompression processes. Additionally, this type of approach is better suited in networks where the end-hosts are not generating or receiving an extraordinarily high amount of packets. Such networks may be referred to as slow or medium speed networks. If the end-hosts are expected to handle a large amount of packets, another approach, such as the hardware approach described in the following section, is preferred.

4.4.2 Hardware Implementation

The hardware implementation of the proposed schemes is similar to the software approach. However, dedicated hardware is used to perform the compression and decompression processes. In this scenario, the scheme's point of packet interception will be executed by the dedicated hardware. There are many benefits to using dedicated hardware to perform the desired processes, including higher efficiency, reducing end-host CPU load, faster compression and decompression speeds, and the added ability to process a large amount of data, among others. However, the one major disadvantage of this approach is its cost.

The hardware approach is preferably used with end-hosts that are connected to a fast network(s) where data is being exchanged in extremely high volumes. This approach can be used with slower networks, however, such computing power may seem unnecessary. In fast networks, the CPU of an end-host does not have the ability or the freedom to forfeit any CPU cycles for the compression or decompression processes, therefore, dedicated hardware is necessary to perform the desired processes.

If the software approach is used in fast networks, a bottleneck situation will occur, resulting in a performance degradation of the communication session. The bottleneck will occur due to the software approach not being fast enough to intercept packets at a high rate and perform the necessary compression and decompression processes.

4.4.3 End-hosts Mutual Agreement and Mode Selection

End-hosts establishing a communication session must determine whether to use any of the compression and decompression schemes or not. To establish a communication session, the end-hosts must perform a three-way handshake similar to that of initiating a TCP communication session. The primary difference rests in the header and payload fields of the SYN and ACK packets. The three-way handshake process can be seen in Figure 4.8.

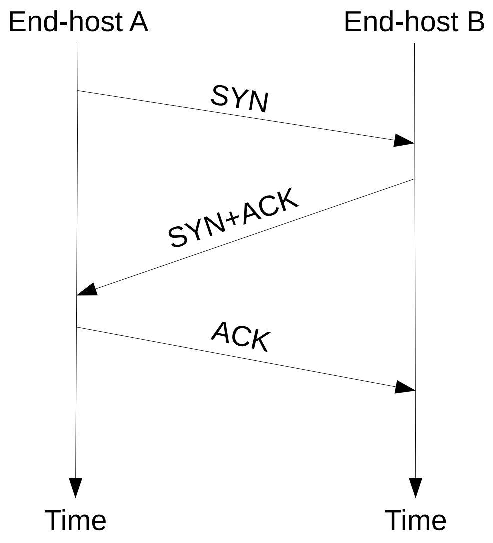


Figure 4.8: Three-way Handshake Process Between End-hosts to Agree Whether to Use Any of the Proposed Compression Schemes

The SYN and ACK packets used in the three-way handshake will contain a value in the EtherType field indicating the type of communication protocol that will be used. To be precise, the field will contain a value indicating that the end-host is capable of performing any of the compression and decompression schemes. Additionally, the payload of the ACK will contain values indicating the accessible lossless compression algorithms by that specific end-host. The ACK packet structure can be seen in Figure 4.9. The SYN packet structure is similar to that of the ACK packet, however, the payload does not necessarily need to contain any information.

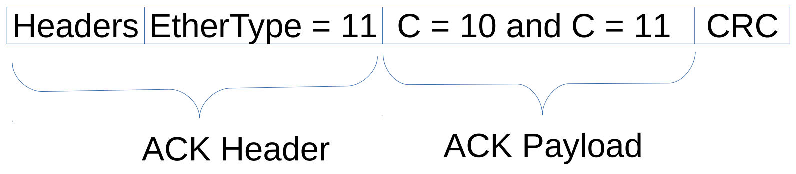


Figure 4.9: ACK Packet Structure

In Figure 4.9, the EtherType field is set to 11 and the payload field contains two consecutive values, 10 *and* 11. The definition of the values in the EtherType field and the payload are given in Figure 4.2. These values are used by the ACK packet in the three-way handshake process to indicate that the end-host sending the said ACK packet is capable of performing any of the compression schemes and utilize any of the available lossless compression algorithms, be it LZO or ZLIB. If the end-host is only capable of using one lossless compression algorithm, such as LZO, the ACK's payload will only contain one defined value, such as $C = 10$. If the end-host is not capable of utilizing any of the compression schemes, the EtherType field will contain a default value, such as the one indicating that IPv4 is being used.

The three-way handshake procedure begins by either end-host A or end-host B sending the other end-host(s) a SYN message. The SYN message EtherType field is set to 11, indicating the desire to initiate a communication session that may need to compress and decompress packets' payloads when necessary. The answering end-host replies to the initiating end-host with SYN-ACK packet. The SYN-ACK packet will have the EtherType field either set to 11, as a positive response, or a default value, as a negative response. In the event of a positive response, the end-host will include a payload in the SYN-ACK message, indicating which compression algorithms the end-host is capable of using. Once the initiating end-host receives the SYN-ACK message, it will respond with an ACK message. The ACK message will contain an EtherType value of 11 and a payload indicating the lossless compression algorithms it will be utilizing throughout the transmission session. When an end-host wants to terminate a communication session, the end-host will perform a process similar to the three-way handshake, however, with a FIN packet rather than a SYN packet.

In the event where an end-host responds with an EtherType value other than 11, the initiating end-host will terminate the handshake with a FIN message. Subsequent to the FIN message, the initiating end-host is capable of establishing a new communication session where the EtherType field of the SYN and ACK messages will contain a default value, such as a value indicating the use of IPv4.

Subsequent to the three-way handshake process, the end-hosts begin exchanging data. The method of operation followed by the end-hosts is described in Figure 4.10. It begins with the end-hosts verifying if the communication session between the involved parties is under mutual agreement to utilize lossless compression algorithms. If the end-hosts are in agreement, the compression scheme process will intercept the packets at the socket layer, otherwise it will not. Once the packets are intercepted, the payloads are subject to compression or decompression, depending on whether the packets are inbound or outbound.

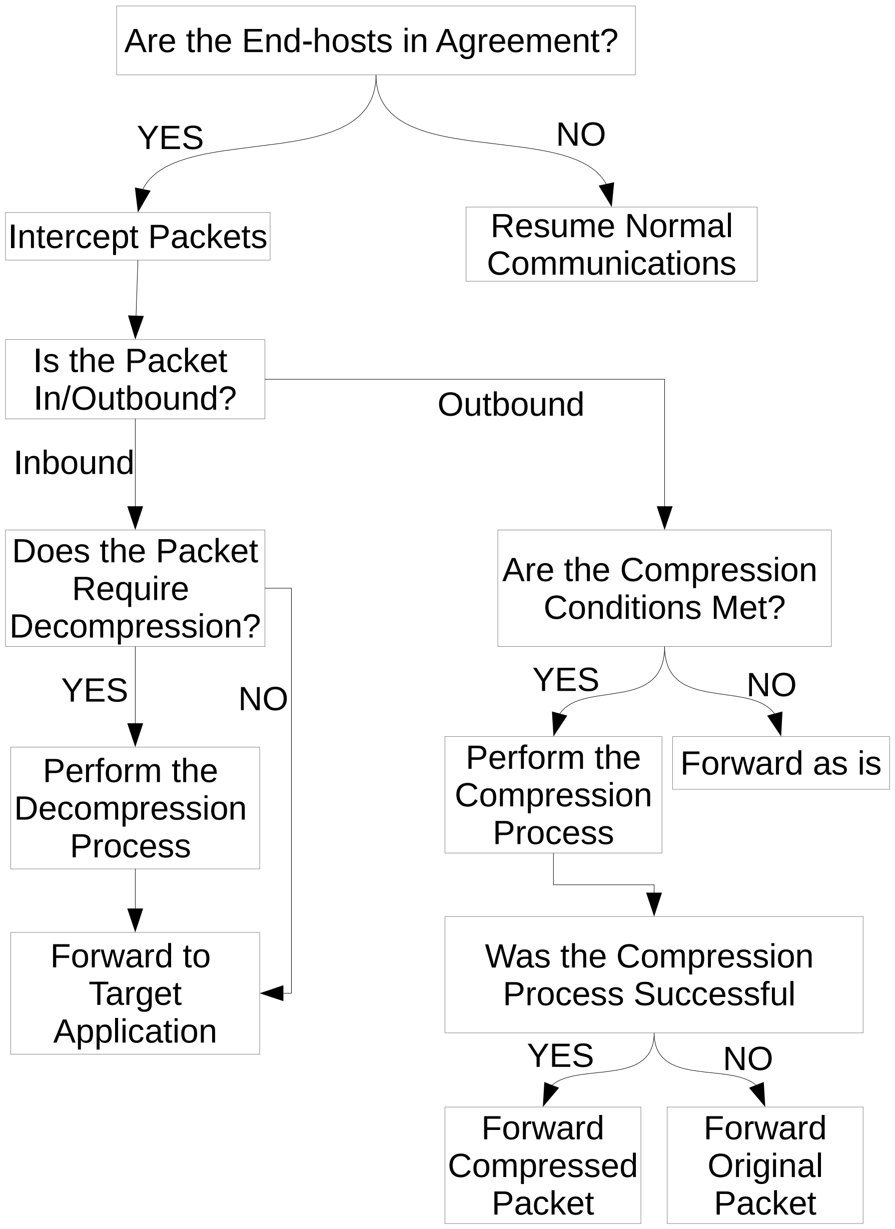

Are the End-hosts in Agreement?
YES
NO
Intercept Packets
Resume Normal Communications
Is the Packet In/Outbound?
Outbound
Inbound
Does the Packet Require Decompression?
Are the Compression Conditions Met?
YES
NO
YES
NO
Perform the Decompression Process
Perform the Compression Process
Forward as is
Forward to Target Application
Was the Compression Process Successful
YES
NO
Forward Compressed Packet
Forward Original Packet

Figure 4.10: Routine Followed By End-hosts While Using the Compression Enabled Protocol

Inbound traffic will be inspected for the necessity of decompression before being passed to the application layer. This process is done by querying the EtherType field value followed by inspecting the beginning of the payload for a header indicating the compression algorithm used by the source end-host. The payload will then be decompressed, if need be, and forwarded to the target application.

Outbound traffic will be compressed after being generated by the application layer. The headers of the outgoing traffic are then made and attached to the payload. In the process of attaching the headers, the EtherType field is triggered by the compression process to be a desired value. The desired value will indicate if the payload is compressed or otherwise.

4.4.4 Compression Scheme Selection

In this chapter, three different compression and decompression schemes were proposed. The end-hosts do not necessarily have to operate under the same schemes when in mutual agreement. Each end-host can use the preferable or optimal scheme from their perspective.

From the three available schemes, the passive mode is the most adaptive compression method that can be used with any type of network, whether slow or fast. This is purely due to the scheme being dependent on network history and previous transmissions. The lossless compression algorithm employed while using passive mode is arbitrary and can be chosen by the user.

The intermediate mode is compromised of two different mechanisms that perform the compression process once certain conditions are met. The AND method of the intermediate mode is preferably used by networks that are considered to be of medium speed. This is due to the AND method having to act like an AND logic gate before activating the ZLIB compression library. The compression method behaving like an AND gate requires the network to have a large amount of bandwidth to tolerate minimal lossless before activating the advanced stages of the scheme.

The OR method of the intermediate mode is preferably used by slow networks. This is due to the OR method acting like an OR gate before activating the ZLIB compression library. In slow networks, there is less available bandwidth and a much higher probability of meeting the necessary conditions, such as crossing the BLRTT value or the threshold of lost packets, to proceed to the advanced stages of the scheme.

The simplest of all schemes, the active mode, is preferably used by fast networks. This is because the scheme lacks any threshold to tolerate losses. Once the BLRTT is value is exceeded, the compression scheme is immediately activated.

4.5 Simulation Model

In order to test and analyze the proposed techniques on a valid network model, a simulation is conducted on an infamous platform. This platform is known as NS3, which is installed and operated on a Linux machine.

The network model used for proof of concept was known as a parking lot topology network, as shown in Figure 3.1. All of the proposed techniques were tested on the same model. Indeed, the proposed techniques were very efficient in a simple network topology, however, it proved the possibility of using these techniques efficiently in highly complex networks.

In this network, node 0 and node 4 are the packet senders and node 3 and node 5 are the packet receivers. Stream 1, which is node 4 sending packets to node 5, will be observed and analyzed. The packets are transmitted at a constant rate of 0.6 Mbps and the link speeds are 1.0 Mbps. All packet transmission delays in the network are set to 0 ms.

Each packet sent used TCP for the reliability provided by the protocol and IPv4 for addressing. The size of each non-compressed packet payload was 500 bytes. The total number of packets sent by each sending node was 1500.

4.5.1 Performance Metrics and Results

There are different ways of measuring the efficiency and the QoS of a TCP transmission session. For this proposed technique, a number of different attributes will be considered such as: number of dropped packets, average IRTT, TCP efficiency, and duration of recovery.

The number of dropped packets shows how many packets were lost during the overall transmission session, which indicates the reliability of data transfer. There are some cases in which dropped packets may be ignored if the application is loss-tolerant, such as VoIP and video transmission. However, other applications, such as banking services, cannot risk having any dropped packets. Therefore reducing the number of dropped packets is necessary and a valid step of improving network conditions [37].

The average of IRTT indicates the overall response time of the network. Lower RTT values are generally better since they indicate a lower end-to-end delay, however it does not necessarily mean the network condition is at its best. Even though the average IRTT may be low, there is still the possibility of a high number of dropped packets and the necessity of retransmitting these packets. Also, there are cases where the IRTT is high, however the number of packets dropped is lower [38]. The following equations, equations (4.1) and (4.2), are used to calculate the average IRTT.

$$AvgIRTT = \frac{TotalRTT}{Transfer\_time}, \tag{4.1}$$

$$RTT = TTP + TTA, \tag{4.2}$$

where the term known as TTP is the total time it takes to transmit the packet with a payload from the sender to the receiver and the term known as TTA is the time it takes to transmit the acknowledgement from the receiver to the sender. The TCP efficiency percentage is an important ratio to calculate. This ratio indicates the amount of bytes that are transmitted successfully. In general the higher the TCP efficiency ratio, the lower the number of packets dropped, which means the network is less congested [38].

$$TCP_{Eff}(\%) = \frac{(TxD - ReTxD)}{TxD} * 100. \tag{4.3}$$

The TCP efficiency equation has two important variables, which are TxD and $ReTxD$. The TxD variable is the total transmitted data during a session where $ReTxD$ is the retransmitted data during the same session.

Finally, the duration of recovery is the period in which the transmission of the sender is throttled, due to a lost segment or duplicate ACKs, until the sender's transmission rate is recovered.

$$DurationOfRecovery = t_{rec} - t_{tout}. \tag{4.4}$$

In equation (4.4), t_{rec} is the time variable where the sender recovers from being throttled and t_{tout} is the time variable where the sender is throttled.

The results of simulating the network whilst using the different modes are shown in Figures 4.11 to 4.13 where the quantitative analysis is done in Tables 4.2 to 4.7. These items are discussed in the following section.

Passive Mode

	NC	**LZO**	**ZLIB**
Attributes	Packets dropped: 35	Packets dropped: 24	Packets dropped: 19
	Time: 12.00 seconds	Time: 11.20 seconds	Time: 11.17 seconds

Table 4.2: Internal Network Values for Passive Mode

Passive Mode

Stream 1	**AVG RTT(ms)**	**TCP Efficiency (%)**	**Recovery (s)**
NC	280.8	97.7	3.4
LZO	263.3	98.4	2.7
ZLIB	251.5	98.7	2.5

Table 4.3: Calculated Network Values for Passive Mode

Intermediate Mode

	NC	**AND**	**OR**
Attributes	Packets dropped: 35	Packets dropped: 29	Packets dropped: 19
	Time: 12.00 seconds	Time: 11.54 seconds	Time: 11.52 seconds

Table 4.4: Internal Network Values for Intermediate Node

Intermediate Mode

Stream 1	**AVG RTT(ms)**	**TCP Efficiency (%)**	**Recovery (s)**
NC	280.8	97.7	3.4
AND	253.7	98.1	3.0
OR	250.1	98.7	2.5

Table 4.5: Calculated Network Values for Intermediate Mode

Active Mode

	NC	**LZO**	**ZLIB**
Attributes	Packets dropped: 35	Packets dropped: 30	Packets dropped: 19
	Time: 12.0 seconds	Time: 11.51 seconds	Time: 11.15 seconds

Table 4.6: Internal Network Values for Active Mode

Active Mode

Stream 1	AVG RTT(ms)	TCP Efficiency (%)	Recovery (s)
NC	280.8	97.7	3.4
LZO	263.8	98.0	2.9
ZLIB	234.1	98.7	2.5

Table 4.7: Calculated Network Values for Active Mode

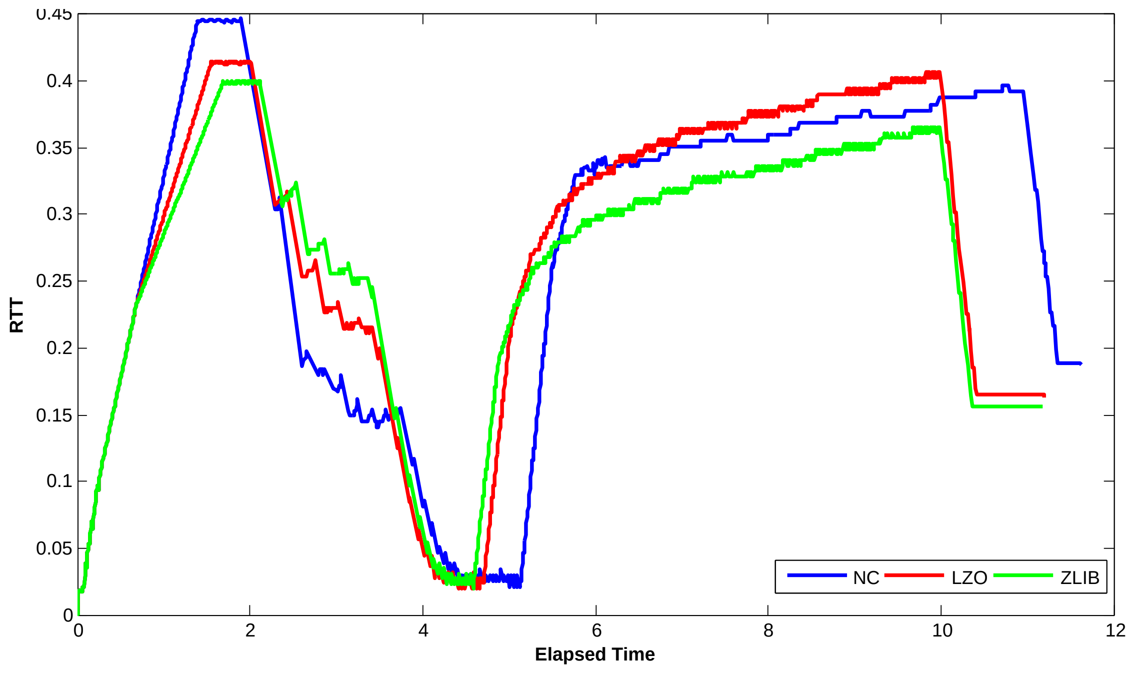


Figure 4.11: Passive Mode Simulation Results

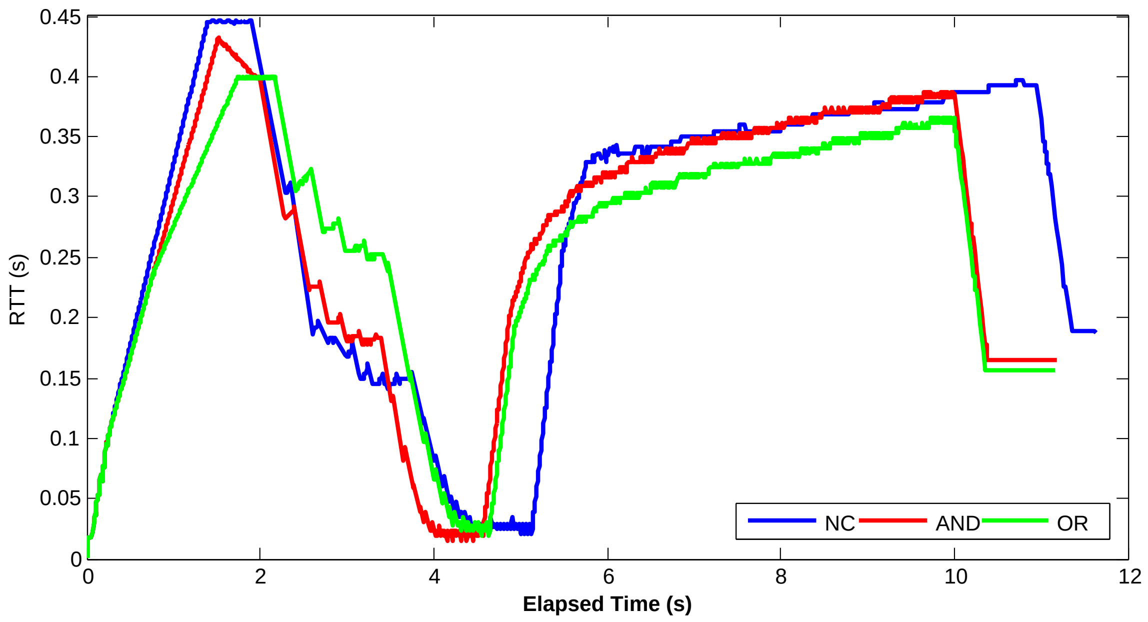


Figure 4.12: Intermediate Mode Simulation Results

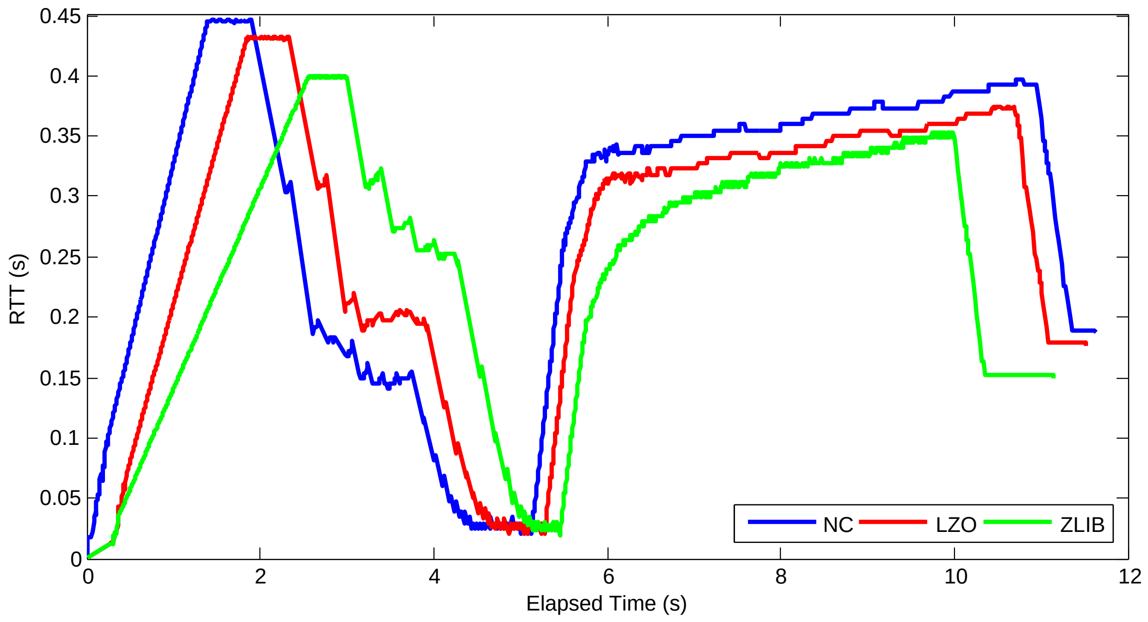


Figure 4.13: Active Mode Simulation Results

4.5.2 Analysis

To test the passive mode, the initial step was to calculate the BLRTT value. The initial BLRTT value was calculated to be 13.10 ms. Once the transmission session began, the IRTT value was continuously compared to the BLRTT value. When the IRTT value was greater than the BLRTT by a certain threshold, which was randomly determined [2], compression was applied, be it the LZO or ZLIB algorithm. In the case where the IRTT value was not greater than the BLRTT value, the transmission session continued without involving any compression algorithm.

The NC line in Figure 4.11 represented the behavior of sending non-compressed packet payloads and receiving an ACK in return. The NC session of Stream 1 and Stream 2 lost a total of 35 packets where Stream 1 was responsible for dropping 34 of the 35 packets. This was the highest number of packet drops when compared to any of the conducted simulations. Therefore, the NC line was used as a reference point to show how the proposed compression techniques improved the transmission session.

The average RTT of the NC session was approximately 280.8 ms, which was much higher than the calculated BLRTT. The high value of the RTT indicated severe network congestion. However, the average RTT value whilst using the passive mode decreased to 263.3 ms when LZO was used and further decreased to 251.5 ms when ZLIB was used. The decrease in RTT values indicated a reduction in the end-to-end delays.

Furthermore, the TCP efficiency was only 97.7 % due to being directly related to the number of packets dropped. It shows that 97.7% of the entire data stream was successfully transmitted. If the network congestion was reduced, the TCP efficiency had to increase, indicating a transmission session with a higher packet delivery success rate.

When the passive mode was activated where the LZO algorithm was employed, Stream 1 and Stream 2 lost a total of 24 packets where Stream 1 was responsible for dropping 23 of the 24 packets. The decrease in packet drops indicated the reduction of data congestion within the

[2]The ideal way of defining the threshold is by choosing it as a multiple of BLRTT.

network. The decrease in data congestion was directly reflected on the TCP efficiency percentage, which increased by 0.7%. This indicated that a higher number of successful transmissions were made while using the LZO algorithm to decrease the size of the packets.

When the passive mode was activated where the ZLIB algorithm was employed, Stream 1 lost a total of 19 packets. There was a massive decrease in the number of packet drops, which indicated a less congested network. The decrease in network congestion was directly reflected in the TCP efficiency percentage. The number did increase by 1.0%, indicating a higher rate of successful transmissions.

Considering the recovery time for the 3 different cases in passive mode, it is clear that there was a sharp decrease from the NC session to the LZO session. The recovery time for the NC session was 3.4 s whereas during the sessions where LZO and ZLIB were used, the recovery time was 2.7 s and 2.5 s, respectively. The decrease was a positive implication of transmission recovery indicating that the congestion window was no longer throttling the sender's transmission speed.

To test the intermediate mode, the initial step involved calculating the BLRTT value of the network. The BLRTT value happened to be equal to the value calculated to test the passive mode. Once the IRTT value crossed the BLRTT value by a certain threshold, the LZO compression algorithm was employed and a new BLRTT value was chosen to determine whether the compression technique would move forward and use ZLIB to perform payload compression.

From Figure 4.12, the AND method of the intermediate mode went through two distinct stages. The first stage was activating LZO compression after the IRTT value crossed the initial BLRTT value by a certain threshold. The second stage was activating the ZLIB compression algorithm after meeting both conditions where the IRTT value crossed the NBLRTT value as well as the number of dropped packets was above a certain threshold. The AND method had a total of 29 packets dropped from both Stream 1 and Stream 2.

The reason for having a high number of dropped packets after employing the AND method was due to setting the threshold for the number of dropped packets too high. This high number of dropped packets may have been avoided if the threshold of dropped packets had been reduced. In this situation, the AND conditions would have been met earlier where ZLIB compression would have been activated at earlier stages.

Nevertheless, the average RTT of the transmission session whilst using the AND method was 253.7 ms. This RTT value was much higher than the BLRTT value of the network, however it was less than the NC average RTT. Such values were a direct indication of network improvement, showing that less time was needed for packets to be transmitted and receive an ACK. This means that the end-to-end delay sharply decreased.

Furthermore, the average RTT of the AND method was less than the average RTT of LZO compression in passive mode. This was due to the AND method using both LZO and ZLIB throughout the transmission session. However, the TCP efficiency was only 0.4 % higher than the NC TCP efficiency, but 0.3 % less than the TCP efficiency of the LZO compression of the passive method.

The reason behind a lower TCP efficiency value of the AND method when compared to the passive mode's LZO session was due to setting the number of dropped packets threshold too high. The high dropped packet threshold resulted in an increase in the number of dropped packets and, as a consequence, a lower TCP efficiency. This was an example where the average RTT in one transmission session was lower when compared to another transmission session, however the number of dropped packets was higher.

The recovery time for the AND method was substantially smaller than that of the NC session. The decrease in the recovery time was 0.4 s, which indicates a network with less congestion. Furthermore, the decrease in the recovery time was directly related to the congestion window where a smaller recovery time indicates a shorter period of transmission throttling; consequently, the transmission may have occurred at higher speeds for longer periods of time.

The OR method of the intermediate mode achieved better results than the AND method. This was due to employing the ZLIB compression algorithm at an earlier stage in the transmission session. The reason behind activating the ZLIB compression algorithm at an earlier stage was due to needing to meet only one of the two conditions. Either the IRTT value was greater than the NBLRTT value or the threshold of dropped packets was exceeded.

The lowest average RTT of the intermediate mode and the passive mode was achieved by the OR method. This decrease in average RTT indicated a drastic improvement in network conditions. Furthermore, the improvement in network conditions may be seen in the reduced number of dropped packets as well as the increase in TCP efficiency. The OR method had a total of 19 dropped packets, causing the TCP efficiency to increase by 1.00 %. Additionally, the recovery time was 0.9 s smaller than that of the NC transmission session.

To test the active mode, the initial BLRTT value of the network was calculated and was equal to the previously calculated value. The technique worked in an active approach where as soon as the IRTT value crossed the BLRTT value, compression was activated. The technique was used with both LZO and ZLIB compression algorithms where ZLIB achieved better results than LZO as shown in Figure 4.13.

When the active mode was employed and LZO was used, 30 packets were dropped from Stream 1 and Stream 2. The number of dropped packets was less than that of the NC transmission session, however it was greater than the number of dropped packets when LZO was employed in passive mode. This was due to the active mode lacking the adaptive nature of the passive mode.

Furthermore, the average RTT achieved whilst using the LZO compression algorithm in the active mode was 263.8 ms, which was 0.5 ms higher than that of LZO in passive mode. Additionally, the TCP efficiency was higher than NC's TCP efficiency but lower than LZO in passive mode's efficiency. The lower efficiency was due to the increase in number of dropped packets. Hence, one can see the importance of being adaptive whilst using LZO compression algorithm as better results were achieved.

The recovery time of the LZO in active mode was 2.9 s, which was 0.5 s less than NC mode, however, it was 0.2 s larger than the recovery time of LZO in passive mode. Therefore the network conditions did improve, as the transmission was throttled for a smaller window of time. Indeed, the network conditions did improve whilst using LZO in active mode, however, LZO in passive mode achieved a greater improvement.

Finally, when the active mode was employed and ZLIB was used, 19 packets were dropped. The low number of dropped packets rendered the TCP efficiency value to be 98.7%. Additionally, the average RTT was 234.1 ms, which was by far the smallest achieved value. The reason behind such a small average RTT was due to activating the ZLIB compression algorithm at a very early stage in the transmission. Furthermore, the reason behind the great success of ZLIB in active mode was the exceptionally small post-compression packet size. Finally, the TCP efficiency value and recovery time were equal to that of ZLIB in passive mode.

The major difference between all of the previous scenarios is the packet payload size. During the original non-compressed transmission, the packet payload was 500 bytes. As LZO and ZLIB were used to compress the data, the payload size was 483 bytes and 444 bytes, respectively. This is a clear indication that the number of packet drops and network congestion are directly related to the packet size. As the payload size decreased, the network was less congested, resulting in fewer packet drops. It was not a linear relationship, however they were directly correlated.

Finally, it is important to know that the time to compress and decompress a packet's payload while using LZO was approximately 30 us and 20 us, respectively. The time to compress and decompress a packets' payload using ZLIB was approximately 100 us and 30 us, respectively. Clearly the time spent to compress and decompress a payload is worth spending due to the improved transmission results.

4.6 Summary

In this chapter, an adaptive compression technique with 3 different modes was employed to improve the condition of a given network suffering from congestion. The different compression modes were the passive, intermediate, and active modes. The passive mode depended on the history of the network to define values for certain technique parameters where the intermediate mode treated the perceived values from the network, such as number of packet drops and RTT in AND/OR gate fashion to determine whether to move to the advanced stage of the compression technique. Finally, the active mode was the simplest of all, as compression was activated in the least adaptive manner.

To test and validate the proposed compression technique, NS3 was used to simulate the network. The three different modes were tested on the same network set-up suffering from the same congestion problems. All of the results were compared to a transmission session where no compression was being used.

All of the proposed modes did improve the network conditions by reducing the number of packet drops, increasing the TCP efficiency of the transmission session, reducing the time of recovery, and finally reducing the end-to-end delay. The reduced number of dropped packets increased the TCP efficiency, as a lower number of packets required a retransmission. Furthermore, a reduction of the end-to-end delay was observed and indicated by the lower average RTT values gathered from the network.

Chapter 5

Adaptive Distributed Compression and Decompression Scheme Utilizing Intermediate Network Nodes

5.1 Introduction

In this chapter, the focus will be on the design and validation of the process of compression and decompression on the payloads of packets within the intermediate nodes of a given network as means of reducing congestion as shown in Figure 5.1. The compression and decompression processes are performed on a packet-per-packet basis to prevent and reduce high network latency. In this scenario, the end-hosts are liberated from performing the compression and decompression processes. Additionally, only one lossless compression algorithm, LZO, will be used in the proposed scheme, due to it having high compression and decompression speeds.

The advantage of compressing payload data inside the network is the packets' payloads are subject to compression, regardless of the application transmitting the data. However, to perform any advanced packet processing, such as payload compression within the network, advanced intermediate nodes[1] are required. The advanced intermediate nodes should be capable of performing traditional packet forwarding functions as well as payload processing.

[1]Advanced intermediate nodes are intermediate nodes, such as routers or switches or firewalls, capable of performing traditional and advanced packet processing functions.

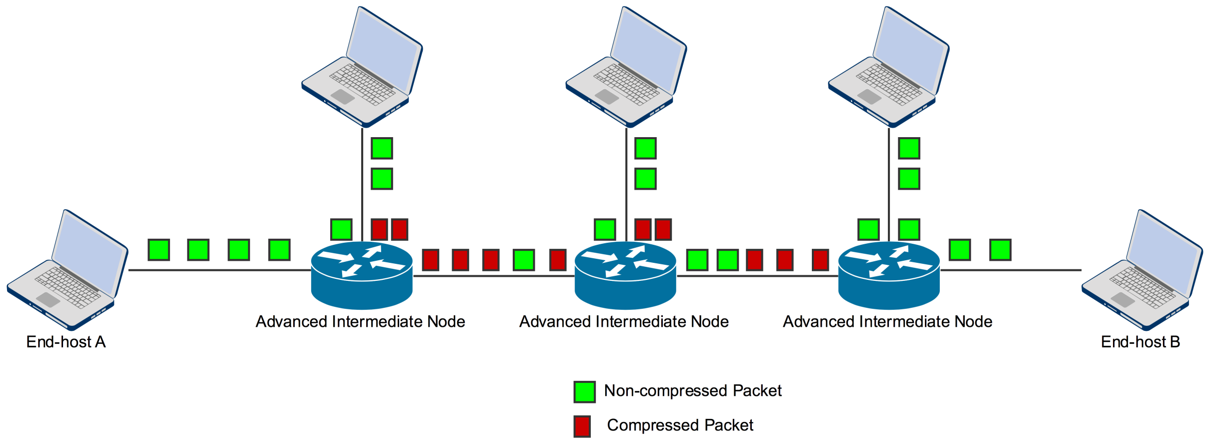

Advanced Intermediate Node
Advanced Intermediate Node
Advanced Intermediate Node
End-host A
End-host B
Non-compressed Packet
Compressed Packet

Figure 5.1: Sample Network where the Compression and Decompression Processes Occur Within the Intermediate Nodes of the Network.

The intermediate nodes have a better sense of the network condition, whether it is congested or otherwise. Therefore, these nodes have a clearer view of the necessity of compression at a given time [26]. It is known that packet queuing occurs at the intermediate nodes when the incoming rate of packets is greater than the nodal processing rate. If queuing occurs, the network is considered to be congested [43].

The proposed scheme suggests that the queues of the intermediate nodes are defined according to an M/D/1 queue. When using an M/D/1 queue to represent a node, it is understood that the incoming packet rate is defined according to a stochastic process and the nodal processing rate is fixed. Therefore, it is possible to calculate the average waiting time for each packet in the queue of an intermediate node using equation (5.3).

In this chapter, the proposed scheme compresses packets' payloads only if they are in the intermediate nodes where the average waiting time of each packet is greater than the necessary time to compress the payload of that particular packet. Additionally, each packet experiences compression once throughout its journey within the network.

As mentioned earlier, the packet generation process is defined according to a stochastic process, such as the *Poisson Process* of arrival. Each source node has a packet generation rate of λ where the rate is a fraction of the maximum forwarding rate of the intermediate nodes. Different λ rates are simulated to represent different loads within the network. As the value of

λ increases, the congestion state of the network will increase.

5.2 Queueing Model for the Network

The network topology considered is shown in Figure 5.2. In this topology, the source nodes are marked as S_i and the destination nodes marked as D_i. Most importantly, the intermediate nodes are represented as N_i. The intermediate nodes are similar in all aspects, such as queue size and average service time. Furthermore, the intermediate nodes are represented using a queuing model, as shown in Figure 5.3.

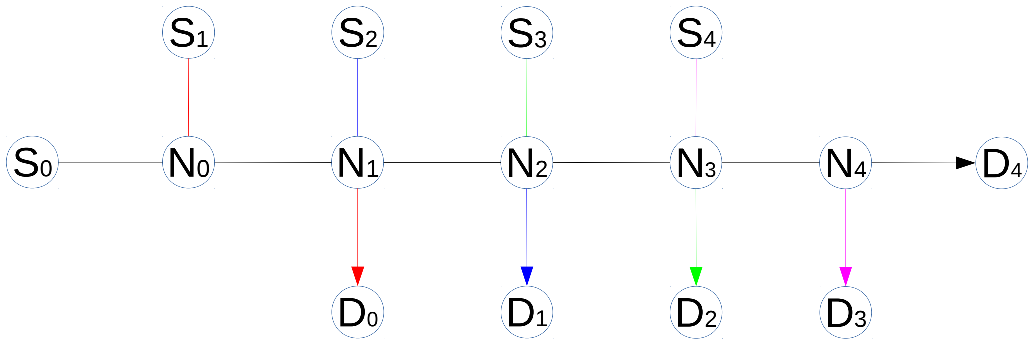


Figure 5.2: Parking Lot Topology With Intermediate Nodes Capable of Advanced Data Processing

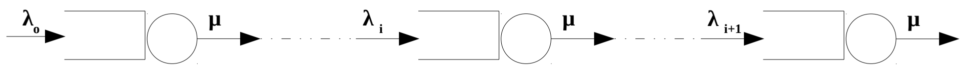


Figure 5.3: Queue View From the Intermediate Nodes' Perspective

According to the M/D/1 queuing model [2], the intermediate nodes perceive a certain stochastic external arrival rate and have a deterministic service rate. The perceived external arrival rate

[2]Queue lengths according to this model are infinitely large. However, in practice, intermediate nodes have a limited buffer space.

is directly related to the packet generation rate, λ_i, at the source nodes. The packets are generated at a rate defined by the *Poisson Process* of arrival. Furthermore, the service rate, μ_i, is a fixed and defined processing rate of the intermediate nodes.

The service rate of each packet depends on how fast the intermediate nodes service the arriving packets. In a wired network without any packet collision detection or data retransmission, the service rate will be defined as in equation (5.1) [41].

$$T_\mu = \frac{L}{R}, \tag{5.1}$$

where L is the size of the data to be transmitted and R is the bandwidth. Therefore, the service frequency of each intermediate node is defined as shown in equation (5.2)

$$\mu = \frac{1}{T_\mu}. \tag{5.2}$$

Since the service time is identical for all packets, the expected packet waiting time in a queue is defined according to the *Pollaczek − Khinchin* (**P-K**) formula, as shown in equation (5.3) [42].

$$W = \frac{\rho}{(2 * \mu * (1 - \rho))}, \tag{5.3}$$

where ρ is known as the utilization factor and defined as,

$$\rho = \frac{\lambda}{\mu}. \tag{5.4}$$

5.3 Packet Generation Rate

The source nodes have a packet generation rate defined according to a *Poisson Process* of arrival [3]. The network was designed where the intermediate nodes see a packet generation rate of λ from each source node. Furthermore, the value of λ was chosen to be a fraction of the

[3]The source nodes considered are generating compressible strings of data.

maximum number of packets a forwarding node can handle. The maximum number of packets a forwarding node can handle was defined as shown in equation (5.5).

$$N_F = \frac{R_{link}}{L_{packet\ size}}, \tag{5.5}$$

where the R_{link} variable is the bandwidth (BW) of the link and $L_{packet\ size}$ is the size of the incoming packet.

Assume a case where there are multiple S source nodes, with equal packet generation rates, connected to a forwarding node with a packet forwarding rate of N_F. In such a case, the total generation rate is $\lambda_t = \lambda_1 + \lambda_2 + ... + \lambda_S = S * \lambda$, where congestion will occur if $\lambda_t > N_F$, ultimately leading to dropped packets and a higher latency. Therefore, to avoid congestion, $\lambda < \frac{N_F}{S}$. However, in this chapter, the ratio for two scenarios was chosen to be slightly larger than $1/S$, where $\frac{\lambda}{N_F} > \frac{N_F}{S}$, thereby simulating a network suffering from congestion.

Furthermore, the average number of packets in a queue is defined as,

$$N_Q = \frac{\rho^2}{2 * (1 - \rho)}, \tag{5.6}$$

where $N_Q \to \infty$ as $\rho \to 1$ [42]. This can be shown in Figure 5.4 where, as λ increases and approaches the value of μ, the number of dropped packets will increase. This is due to the queue of the intermediate node being capable of holding a finite number of packets. The increase in the dropped number of packets is a direct reflection of the stress placed upon the network.

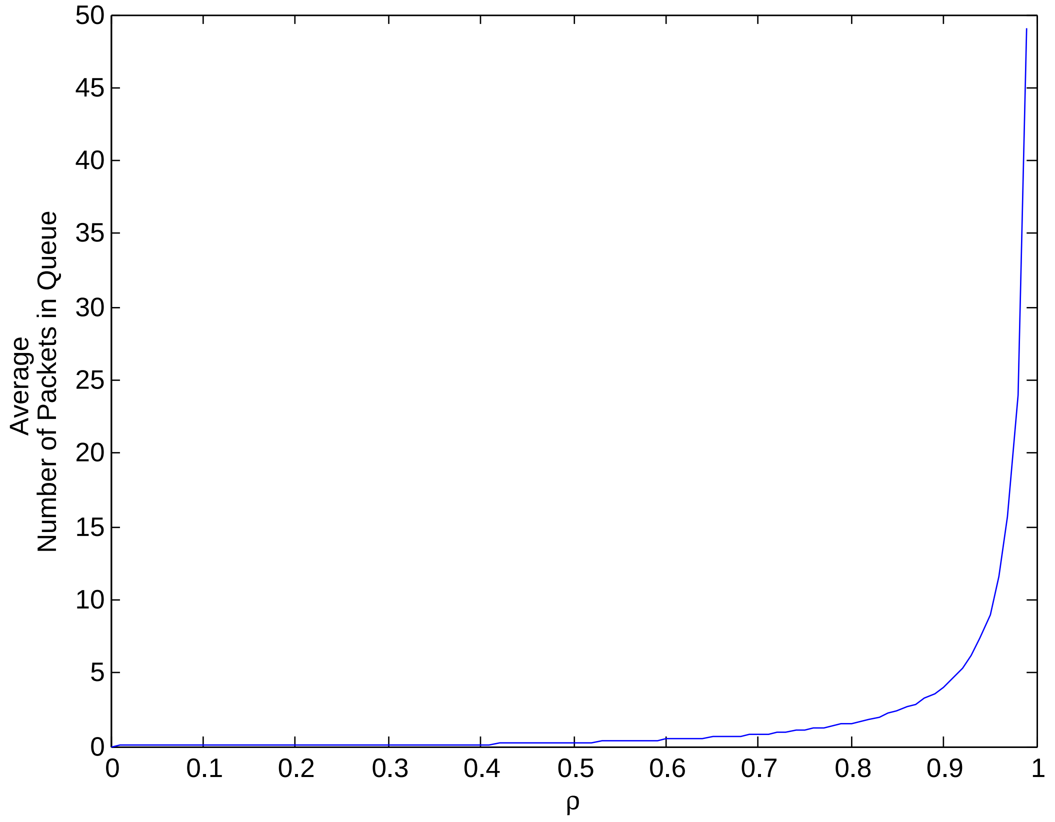


Figure 5.4: Average Number of Packets in a Queue of an Intermediate Node as a Function of ρ

5.4 Compressed and Non-compressed Packet Identification

An intermediate node will be able to identify whether a packet is compressed or not by analyzing the value in the *EtherType* header field. If a packet's payload was compressed in the N_i node, then node N_{i+1} will not perform the compression process due to unnecessary redundancy. The value of the header field indicates the condition of the payload, as shown in Figure 5.5 .

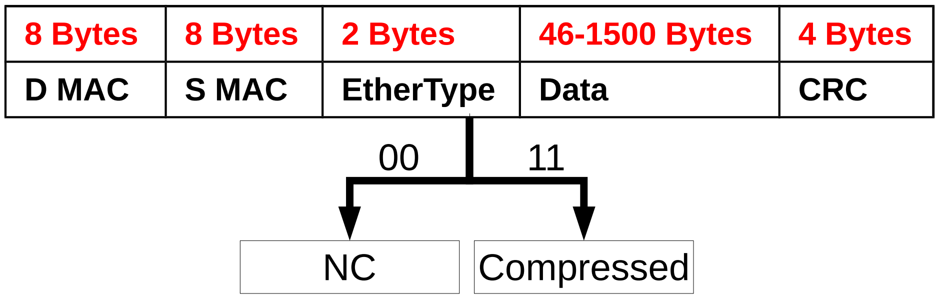


Figure 5.5: Identifying if the Payload is Compressed or Otherwise

5.5 Implementation

The implementation of the proposed adaptive distributed compression and decompression scheme requires the intermediate nodes of a network to be capable of performing traditional routing functions and advanced packet processing mechanisms. Any intermediate node capable of performing additional packet processing within its queue is known as an advanced intermediate node.

According to [26], the advanced intermediate node must have dedicated hardware to perform the compression and decompression processes. The dedicated hardware is necessary to ensure that the processes are carried out in a timely manner, without degrading the QoS while avoiding bottleneck situations.

Prior to discussing the implementation of an advanced intermediate node, the details and make-up of a generic intermediate node, such as an L3 router, are briefly reviewed in the sections to follow.

5.5.1 Generic/Traditional Routers

An abstract view of a generic L3 router is shown in Figure 5.6. From this figure, it can be seen that there are two major function planes, software and hardware, and four major components to be considered when analysing routers. The four major components include $input\ ports$, $output\ ports$, $switching\ fabric$, and $routing\ processor$. Each component has a set of well defined functions that are executed harmoniously to stimulate seamless packet handling [40].

Software and Hardware Planes

From Figure 5.6, it can be seen that the router functions on two different planes. The software plane is known as the **router control plane** where router control functions are executed by the routing processor. These functions are implemented in software and they include executing routing protocols, maintaining routing tables, and other network management operations.

The hardware plane is referred to as the **router forwarding plane** and it is responsible for executing the router forwarding functions. The collective of the router's input ports, output ports, and switching fabric constitutes the forwarding function of the router. They are exclusively implemented in hardware for timing constraints.

Input Ports

The input port is the first segment of the router an incoming packet experiences. As shown in Figure 5.7, the input port begins with a physical layer function known as *line termination*. This function terminates an incoming physical link at a router. Following the physical layer

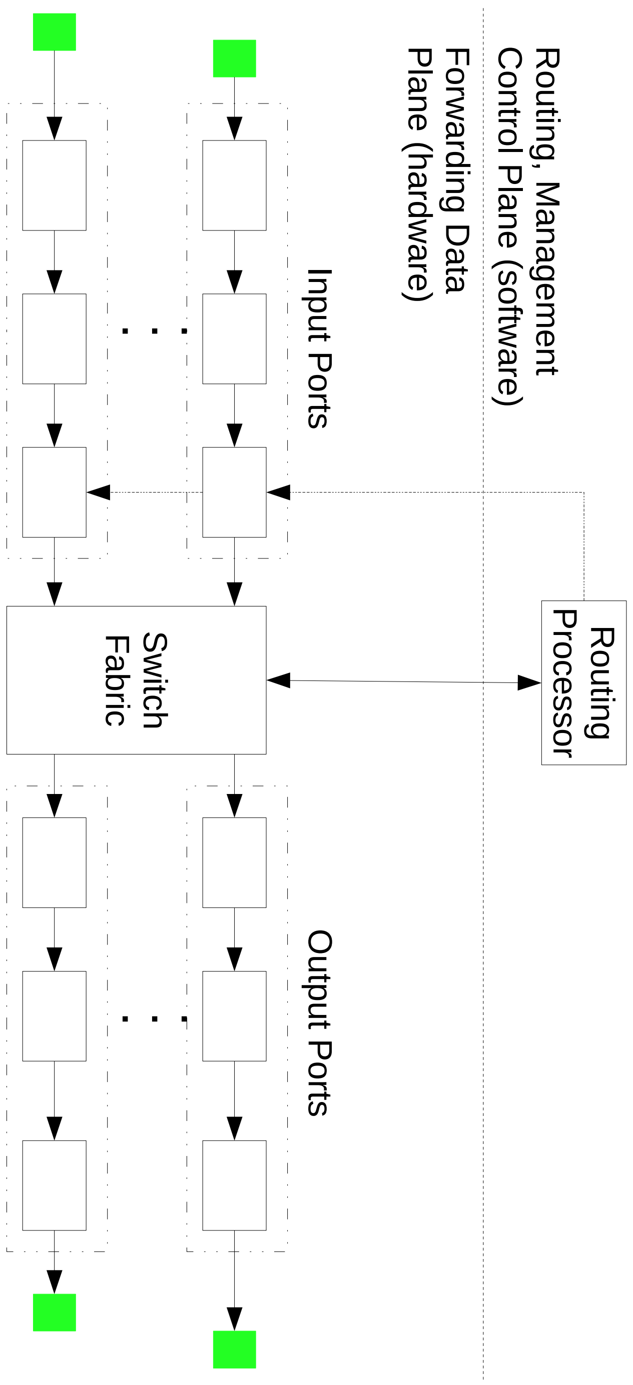


Figure 5.6: Abstract View of L3 Router Architecture

functions, the router performs data-link layer processes to interoperate with the data-link layer at the other end of the line.

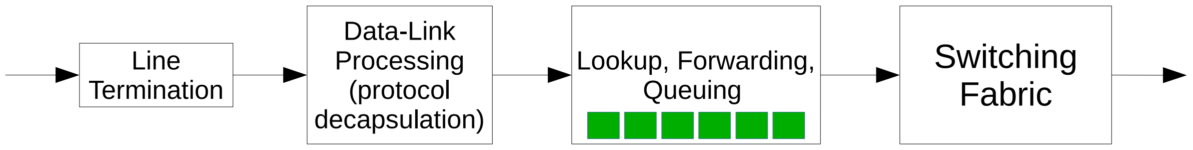

Line Termination
Data-Link Processing (protocol decapsulation)
Lookup, Forwarding, Queuing
Switching Fabric

Figure 5.7: Input Port Stages Experienced by Packets

The most important function performed by the input port is determining the correct output port using the lookup function. The router utilizes the switching fabric to move a packet from the input port to the output port, however, this occurs after consulting the routing table fabricated by the routing processor.

The routing processor continuously computes and updates the forwarding table where a copy is stored at each input port. The lookup function will perform the necessary steps, such as searching through the forwarding table, to determine the output port of a packet.

Switching Fabric

The switching fabric is responsible for moving packets from the input ports to the output ports. This process occurs after the lookup function in the input port performs a search in the forwarding table to determine the output port. There are a few methods currently in use by switching fabrics, such as switching via memory or switching via an input-output shared bus.

Output Ports

The output port of the router is responsible for handling the packets received from the switching fabric as shown in Figure 5.8. The output port will queue the incoming packets and then perform the necessary data-link and physical layer functions to successfully transmit the packet.

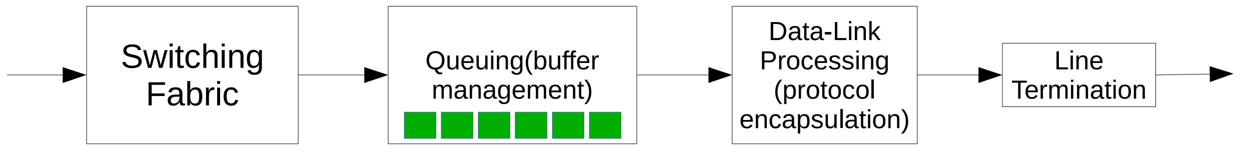

Switching Fabric
Queuing(buffer management)
Data-Link Processing (protocol encapsulation)
Line Termination

Figure 5.8: Output Port Stages Experienced by Packets

The queue of the output port is responsible for packet loss in many networks. To further understand how, assume a router with N input and N output ports connected to identical links with identical transmission speeds, R_{line} pps. Furthermore, assume that the switching fabric's transfer rate is $R_{switch} = N * R_{line}$ to minimize the queue at the input port. Additionally, assume that all packets received by the router are of identical size. In the worst case scenario, when all N input ports receive packets destined to the same output port, designing R_{switch} to be $N * R_{line}$, will clear all packets through the switching fabric before a new set of packets arrives.

Since the output port is operating at R_{line}, a queue will form since the switching fabric is operating at a higher rate. Eventually, the memory of the output port will be exhausted due to forming a large queue. Once a large queue is formed, new incoming packets are dropped due to the lack of space.

Routing Processor

The routing processor is responsible for executing different routing control functions including the execution of the correct routing protocols, maintaining and updating the routing table, maintaining the status of the attached links, maintaining and computing the forwarding table of the router, and performing network management functions.

5.5.2 Hardware Implementation

To implement the proposed scheme, an advanced router or intermediate node is required. The advanced node contains additional hardware and software that allow the node to behave in a desired way. The following sections describe the structure of an advanced intermediate node.

5.5.3 Anatomy of An Advanced Intermediate Node/Advanced L3 Router

The differences between a traditional and an advanced router primarily rest in the functionality of the router and then in the physical hardware, as shown in Figure 5.9. The advanced router can not only perform traditional routing functions, it is also capable of executing processes and services that are typically not available in a traditional router. The added functions are present in both the software and hardware planes. Furthermore, there is an increased level of cooperation between the two planes. The following subsections will describe the added functionalities and hardware to each major component of a traditional router.

Software and Hardware Planes

The software or the router control plane of an advanced router is similar to a traditional router but with added processes for the routing processor to execute. Additionally, the routing processor is now involved in triggering advanced functions in the hardware plane. The increased cooperation between both planes allows seamless operation of the proposed compression and decompression scheme. The hardware plane of an advanced router is significantly different than the traditional router due to the added hardware. The added hardware is capable of performing header inspections, decisions regarding whether a packet payload requires compression or decompression, and perform the compression or decompression processes seamlessly.

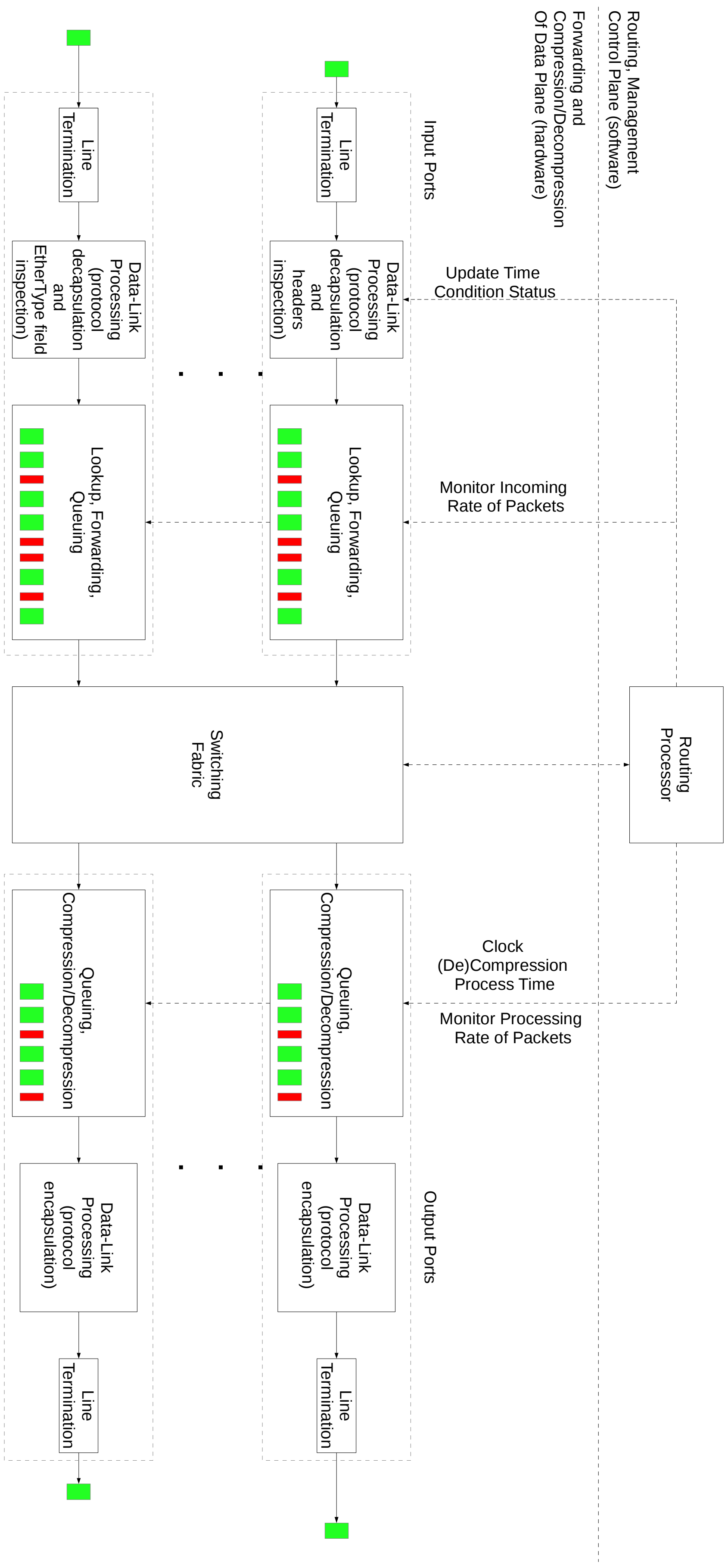


Figure 5.9: Advanced L3 Router Architecture

Input Ports

The input ports of an advanced router will perform the physical and data-link layer functions of a traditional router, however, a header inspection function is added. This function will examine the EtherType field to determine whether the packet is compressed or not. Additionally, the routing processor will inform the hardware performing the header inspection whether the timing condition, if the average expected waiting time is greater than the time required to perform the compression process, is met.

Furthermore, the input ports will have three distinct queues, as shown in Figure 5.10, including: $for-compression-queue$, $for-decompression-queue$, and $for-transmission-queue$. Incoming packets are queued in one of the three queues based on the header inspection and lookup table results. The switching fabric is responsible for moving the packets from their respective queues to the desired queue or function located in the output ports.

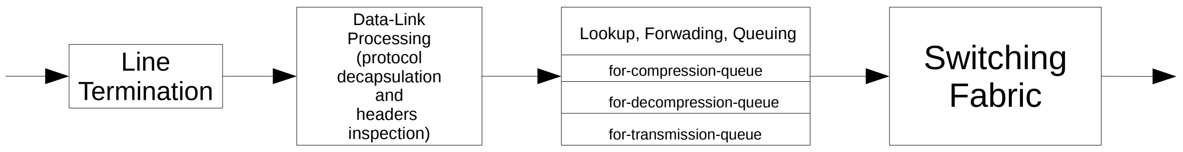


Figure 5.10: Input Ports With Advanced Functionality and Three Distinct Queues

In the case where the header inspection function returns a value from the EtherType field indicating that the payload is compressed, the packet is then forwarded to one of the three input port queues. The location of the packet will depend on the results of the lookup function. If the lookup function determines that the packet is heading towards its destination, the packet will be enqueued in the $for-decompression-queue$. However, if the packet is heading towards another advanced intermediate node, the packet will enqueued in the $for-transmission-queue$.

In cases where the header inspection function returns a value from the EtherType filed indicating that the payload is not compressed, the packet will be forwarded to one of the three input queues. Initially, the lookup function is consulted to determine whether the next hop is the packet's final destination or not. If the next hop is indeed the packet's final destination, the packet will be queued in the $for-transmission-queue$. However, if the lookup function determines otherwise, the packet will be placed in the $for-compression-queue$ if the timing condition is met.

Switching Fabric

The switching fabric will perform identical operations to that of a traditional router, that is move a packet from the input port to the correct output port based on the results of the lookup function. When the switching fabric moves a packet to the correct output port, depending on the queue origin of the packet, the switching fabric will move the packet to their respective queues. For example, if the packet is in the $for-compression-queue$ in the input port, subsequent to determining the correct output port, the switching fabric will place the packet in the $for-compression-queue$ of the compression thread in the output port.

Output Ports

The output port of the advanced router is similar to the traditional router, however, there is extra hardware responsible for performing the compression or decompression process when necessary. The hardware knows which process to execute by monitoring the compression and decompression thread queues. These queues are filled by the switching fabric as it enqueues packets in the $compression-thread-queue$ or $decompression-thread-queue$ or $for-transmission-queue$, depending on the originating queue from the input port. If packets are enqueued in the compression or decompression queues, subsequent to the processes, the resulting packets are enqueued in the $for-transmission-queue$, as shown in Figure 5.11.

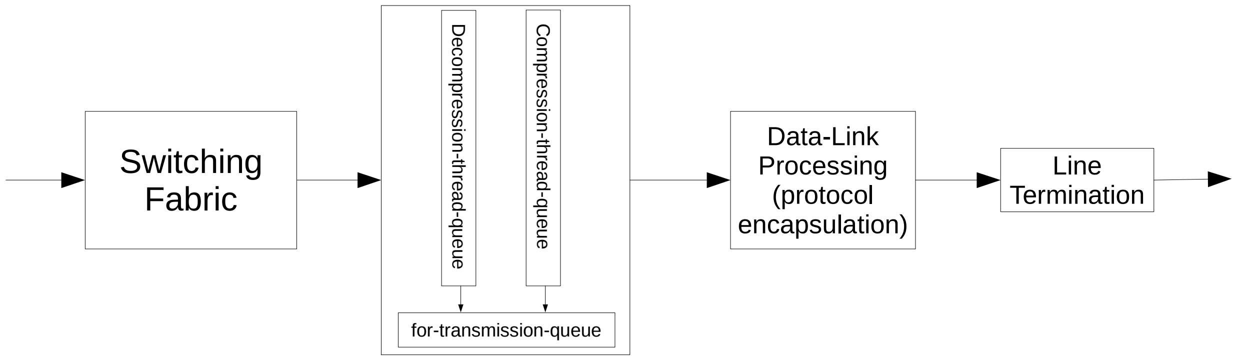


Figure 5.11: Output Ports With Advanced Functionality to Perform the Required Compression and Decompression Processes

Routing Processor

The routing processor of an advanced router performs the functions of a generic router and more. It is capable of:

- Monitoring the incoming rates of packets for all input ports.
- Monitoring the number of packets in the input ports queues.
- Monitoring outgoing rates of packets for all output ports.
- Monitoring the number of packets in the output port queues.
- Clocking the compression and decompression processes carried out by the dedicated hardware.
- Monitoring the outputs of the compression and decompression processes for successful and failed outcomes.
- Calculating the $P - K$ equation instantaneously and continuously.
- Performing the inequality/timing condition check and forwarding the values to the hardware responsible for inspecting the headers of the incoming packets.

For the routing processor to perform the traditional and advanced functions, the processor must either be much more powerful than the processor of a generic router or dedicated hardware can be used to perform the advanced functions.

5.5.4 Heuristic Algorithm

A different approach to how the advanced intermediate node operates is presented in Figure 5.12. In this figure, a flowchart representing a simplified heuristic algorithm is presented. The purpose of this flowchart is to show how the scheme operates on a high level without diving into the structure of an advanced intermediate node.

When the advanced intermediate node receives a packet, the node begins by determining if the current node is the last advanced node the packet will encounter prior to reaching its destination. Based on the nodes analysis of a packet's route, the node will decide the fate of the packet.

If the current node is the packet's final advanced node prior to reaching the destination, the node will investigate if the packet is required to pass through the decompression process. Based on the results of the investigation, the packet is either decompressed and then forwarded to the destination or simply just forwarded to the destination.

If the current node is not the last advanced node a packet will encounter prior to its destination, the node will subject the payload of the packet to the compression process if necessary and if the timing conditions are met. If the timing conditions are not met, the packet is queued and forwarded as is. However, if a packet is subjected to the compression process, i.e. the timing conditions are met, the packet is compressed prior to forwarding it to the next node.

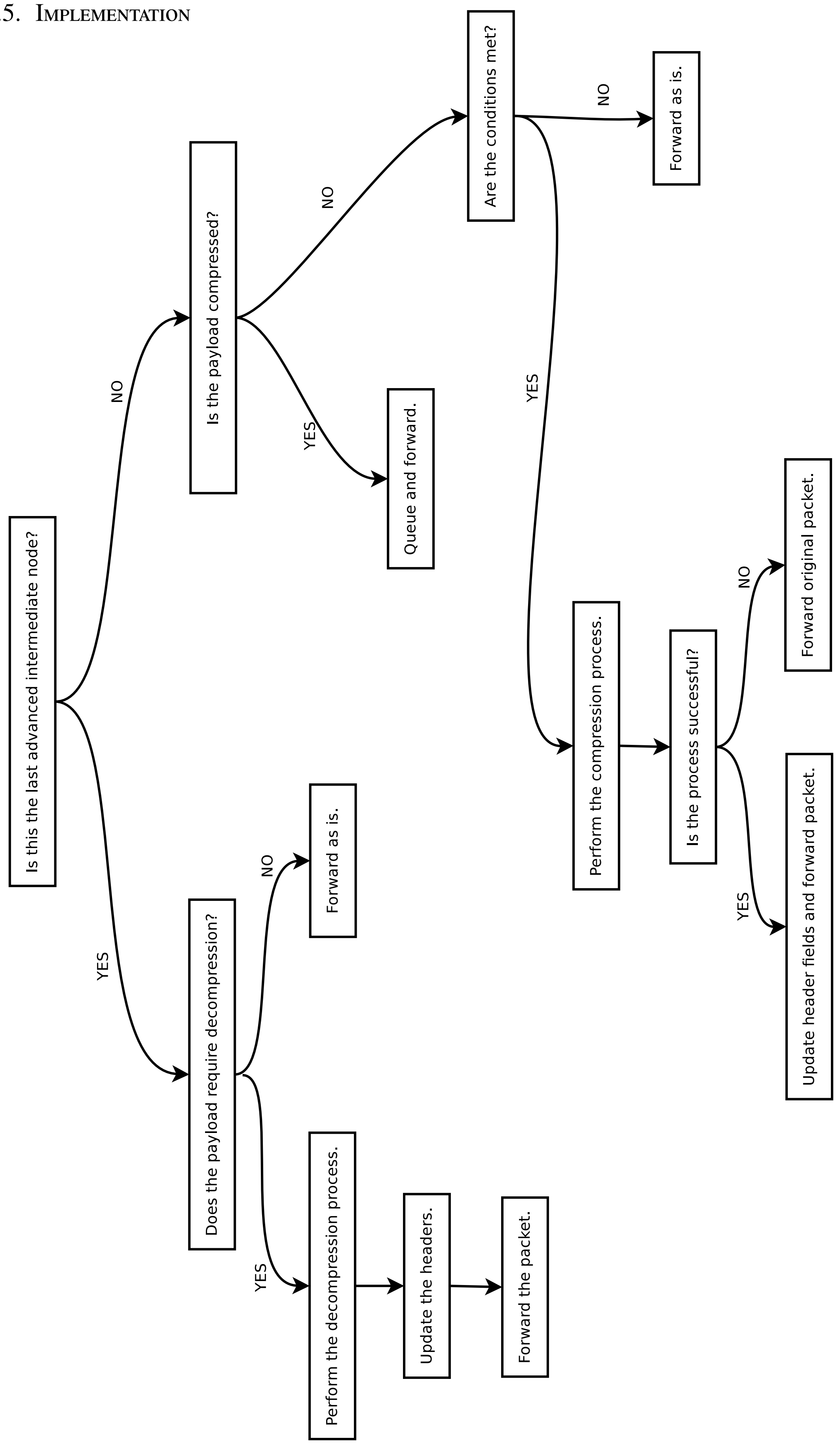

Is this the last advanced intermediate node?
NO
YES
Is the payload compressed?
NO
YES
Queue and forward.
Are the conditions met?
NO
Forward as is.
YES
Perform the compression process.
Is the process successful?
NO
Forward original packet.
YES
Update header fields and forward packet.
Does the payload require decompression?
NO
Forward as is.
YES
Perform the decompression process.
Update the headers.
Forward the packet.

Figure 5.12: Simplified Algorithm for the Proposed Scheme

5.6 Simulation Model

To test and analyze network performance with middle nodes executing compression and decompression processes, a simulation was conducted using NS3. The network topology used was a parking lot topology as shown in Figure 5.13.

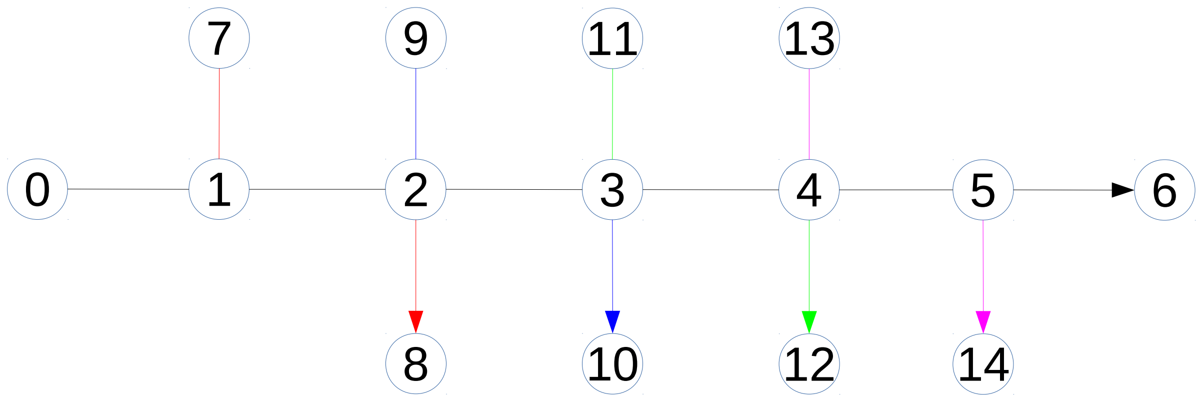


Figure 5.13: Network Topology Used for the Simulation

In this network, nodes 0,7,9,11, and 13 are packet senders while nodes 6,8,10,12, and 14 are packet sinks. The source-destination pair, node 0 to node 6, will be referred to as Stream 1. Stream 1 will be primarily observed and analyzed in the following sections.

Multiple simulations were conducted to show the performance of the compression technique under various network loads. The network loads were determined by controlling the packet generation rate. The first simulation conducted was designed to have all source nodes have a packet generation rate, λ, equivalent to 50% of the maximum forwarding rate. The second simulation was designed such that the source nodes had a packet generation rate equivalent to 60% of the maximum forwarding rate. The final simulation was designed to have each source node generate packets at a rate equivalent to 80% of the maximum packet forwarding rate.

All of the flows generated packets according to Table 5.1 and the link speeds were 1.0

Mbps. Each packet transmitted had a 500-byte payload. Furthermore, each packet used UDP as well as IPv4 protocol for addressing.

	λ *pps*	N_F *pps*	*Packet Generation Rate Mbps*
Simulation 1	$50\% * N_F = 117.93$	$N_F = 235.85$	0.5
Simulation 2	$60\% * N_F = 141.51$	$N_F = 235.85$	0.6
Simulation 3	$80\% * N_F = 188.68$	$N_F = 235.85$	0.8

Table 5.1: Packet Generation Rate of Each Source Node as a Function of the Intermediate Node's Forwarding Rate

5.7 Performance Metrics and Results

The network performance and efficiency values are indicators of the network's overall QoS. The values are quantified using the following different metrics: number of dropped packets, throughput, average number of packets in queue, average latency, and system efficiency.

The number of dropped packets is the number of packets lost during a transmission session. There are situations where intermediate nodes enqueue packets from different flows into the same buffer. This occurs when the node's rate of incoming packets is greater than the node's rate of outgoing packets. Therefore, in such an event, packets are dropped once the node's queue is full. Furthermore, this situation is a clear indicator of network congestion [43]. The ideal situation would be a transmission session with a minimal number of dropped packets.

Throughput is the total amount of data sent across the network over a given transmission period [43]. A higher throughput is highly desirable as it indicates relatively better performance of the network. Furthermore, certain applications may desire a certain throughput of $\boldsymbol{x}$ bytes per second [11]. In equation (5.7), the definition throughput is given.

$$Throughput = \frac{TS}{TT}, \tag{5.7}$$

where the term TS is known as the transfer size that defines the amount of data transferred in bytes. Furthermore, the term TT is the total time of the transmission session in seconds.

According to Little Law, the average queue length, ***L***, of a node is defined by the average service time of a packet, ***W***, as well as the average arrival rate of packets, $\boldsymbol{\lambda}$, per unit time [45]. In equation (5.8), the definition of Little Law is given.

$$\boldsymbol{L = \lambda * W,} \tag{5.8}$$

where $\boldsymbol{\lambda}$ is the average arrival rate in packets per second (**pps**) and ***W*** is the service time in seconds (s) per packet.

The ideal case involves a very small ***W*** and a very large $\boldsymbol{\lambda}$ to maximize the number of packets serviced per node. In this case, the network congestion will decrease drastically, resulting in a highly efficient network.

An extremely important network performance metric is the one-way latency of a given stream. It is the measure of time, in seconds and on a per packet basis, of how long it takes a packet to travel from the source to the destination. In this chapter, propagation delay was ignored and only transmission and average queueing delay were accounted for. The following equation describes the latency of each packet as it propagates through the network [44].

$$\boldsymbol{Latency = Td + Qd,} \tag{5.9}$$

$$\boldsymbol{Td = \frac{ps}{ls},} \tag{5.10}$$

where Td is the transmission delay of each packet defined as a function of packet size (ps) and link speed (ls). Furthermore, Qd is the queueing delay, which is a variable number that depends on how busy the node is servicing other packets.

Finally, the system improvement metric is a ratio indicating the improvement of a system in terms of dropped packets.

$$\boldsymbol{SYS_{Imp} = \frac{(PDPC - PDAC)}{PDPC} * 100.} \tag{5.11}$$

The system improvement equation has two important variables, which are PDPC and PDAC. The PDPC variable is the total number of packets dropped in a network prior to compression where the PDAC variable is the number of dropped packets in a network post compression.

The results of simulating the network under different loads are shown in Figure 5.14. Furthermore, the quantitative results are shown in Tables 5.2 to 5.7.

Stream	**1**	**2**	**3**	**4**	**5**
Packets Dropped	0	0	0	0	0
Average Latency (s)	0.344	0.184	0.096	0.057	0.037
Nodes	**1**	**2**	**3**	**4**	**5**
Throughput (Bytes/s)	124958	187437	187437	187437	124958
Average Queue Length	39.90	19.70	10.35	5.68	0.0

Table 5.2: Simulation Results with the Source Node Generation Rate of $\lambda = \mathbf{50\%}$ in the Absence of the Compression Scheme

Stream	**1**	**2**	**3**	**4**	**5**
Packets Dropped	0	0	0	0	0
Average Latency (s)	0.023	0.015	0.013	0.014	0.013
Nodes	**1**	**2**	**3**	**4**	**5**
Throughput (Bytes/s)	125380	188070	188070	188070	125380
Average Queue Length	0.50	0.0	0.35	0.13	0.0

Table 5.3: Simulation Results with the Source Node Generation Rate of $\lambda = \mathbf{50\%}$ in the Presence of the Compression Scheme

Stream	**1**	**2**	**3**	**4**	**5**
Packets Dropped	4710	4770	0	0	0
Average Latency (s)	1.242	0.428	0.425	0.261	0.163
Nodes	**1**	**2**	**3**	**4**	**5**
Throughput (Bytes/s)	124645	174731	199837	199837	124962
Average Queue Length	97.55	96.87	58.38	35.35	0.0

Table 5.4: Simulation Results with the Source Node Generation Rate of $\lambda = \mathbf{60\%}$ in the Absence of the Compression Scheme

Stream	**1**	**2**	**3**	**4**	**5**
Packets Dropped	0	0	0	0	0
Average Latency (s)	0.031	0.017	0.013	0.013	0.013
Nodes	**1**	**2**	**3**	**4**	**5**
Throughput (Bytes/s)	150996	226493	226493	226493	150996
Average Queue Length	1.25	0.38	0.64	0.65	0.0

Table 5.5: Simulation Results with the Source Node Generation Rate of $\lambda = 60\%$ in the Presence of the Compression Scheme

Stream	**1**	**2**	**3**	**4**	**5**
Packets Dropped	14223	14303	0	0	0
Average Latency (s)	1.443	0.430	0.432	0.349	0.283
Nodes	**1**	**2**	**3**	**4**	**5**
Throughput (Bytes/s)	124545	149630	224848	224848	124967
Average Queue Length	98.44	98.45	78.94	63.43	0.0

Table 5.6: Simulation Results with the Source Node Generation Rate of $\lambda = 80\%$ in the Absence of the Compression Scheme

Stream	**1**	**2**	**3**	**4**	**5**
Packets Dropped	8686	8751	0	0	0
Average Latency (s)	1.051	0.349	0.348	0.230	0.155
Nodes	**1**	**2**	**3**	**4**	**5**
Throughput (Bytes/s)	154047	208357	254469	254469	154390
Average Queue Length	98.19	98.87	63.74	41.55	0.0

Table 5.7: Simulation Results with the Source Node Generation Rate of $\lambda = 80\%$ in the Presence of the Compression Scheme

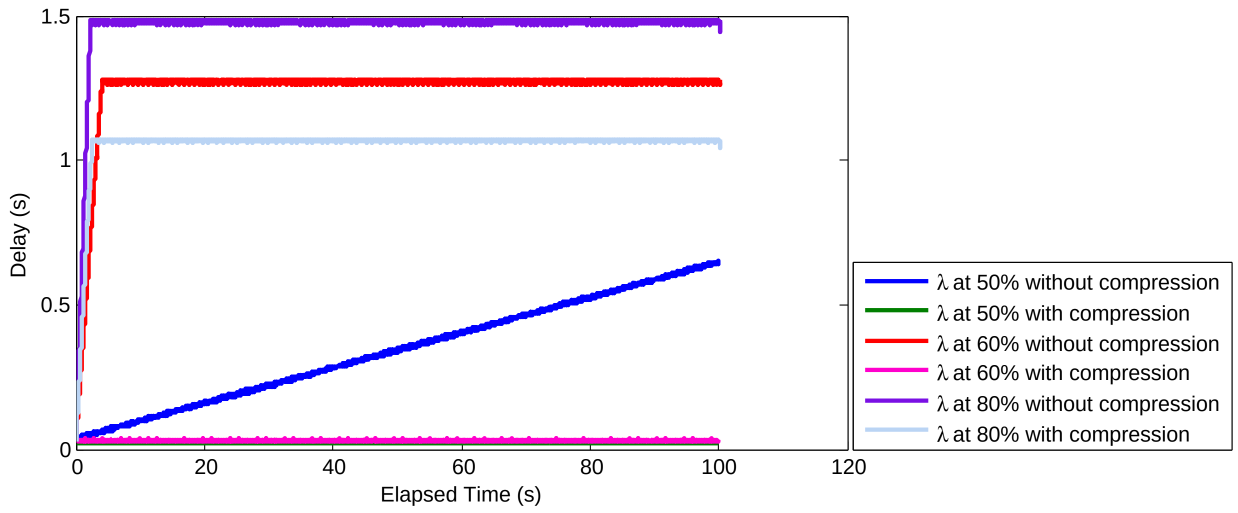


Figure 5.14: Simulation Results in the Absence and Presence of the Compression Scheme Under Different Network Loads

5.8 Analysis

In this section, Stream 1 will be primarily analyzed. Tables 5.2 to 5.7 show the results of the simulation, with and without compression, under different network loads. Each table represents the results of the system when the source nodes are generating packets at a rate that is a fraction of the maximum number of packets a forwarding node can handle.

When the source nodes were generating packets at a rate of $\boldsymbol{\lambda = 50\%}$, the network was performing relatively well due to the lack of congestion. From Table 5.2, one can see that the number of dropped packets by all streams was **0** prior to compression. Additionally, the average latency as well as the queue length size were relatively small. After enabling compression, the size of the average queue length and latency became smaller. Therefore, enabling compression in a non-congested network was beneficial, though unnecessary, due to the small gain in benefits. The reason that the network did not drop any packets was that the source nodes had a packet generation rate that was equivalent to the forwarding nodes.

Consider the case of a slightly congested network where $\boldsymbol{\lambda = 60\%}$. From Table 5.4 and 5.5, it was clearly shown that the number of dropped packets by Stream 1 decreased dramatically after enabling compression. Prior to compression, the number of dropped packets by Stream 1 was **4, 710**. Further, the total number of packets dropped by the network was **9, 480**. However, after compressing and decompressing packets in the middle nodes using LZO as a compression algorithm, the number of dropped packets by Stream 1 was **0**. Furthermore, the total number of packets dropped by the network was reduced to **0**. Using the system efficiency equation, the network was **100.0%** more efficient after enabling compression due to the sharp decrease in number of dropped packets.

The average latency for Stream 1 prior to compressing packets within the network was 1.242 s. However, after compressing packets within the network, the average latency decreased to 0.031 s, which is equivalent to a **97.50%** decrease. When comparing the average queue length of intermediate nodes prior and post compression, it is shown that there was a drastic decrease in values. The lower queue length values show that the intermediate nodes were able to forward packets rapidly due to their smaller size.

Moreover, the decrease in the number of dropped packets and the decrease in the average latency indicates a drastic improvement in the performance of the network. The improvement is a direct reflection of a successful packet payload compression rendering the network to be truly non-congested.

Looking at the intermediate nodes, Table 5.5 shows that post compression, the throughput of nodes 1,2,3,4, and 5 increased significantly. The reason behind the significant increase in throughput values was purely due to the definition of throughput using equation (5.7). Equation (5.7) defines throughput as a function of transmission time and packet size. As the packet size and transmission time decrease in value, the output of the throughput equation increased. Taking into account the other values, such as average latency or number of dropped packets, it is clear that a higher throughput is more desirable as it indicates a healthier and highly efficient network.

Finally, as mentioned before, the value of the average latency, average queue length, and number of dropped packets sharply decreased rendering the network to be in a non-congested state and as such comparable to the performance of the network when $\boldsymbol{\lambda = 50\%}$. Thus the benefits of compression in a slightly congested network are quite large.

In a heavily congested network where $\lambda = \mathbf{80\%}$, the compression technique slightly improved network performance. Prior to compression, Stream 1 dropped $\mathbf{14,223}$ packets and had an average latency of 1.443 s. However, after compression, the number of dropped packets decreased to $\mathbf{8,686}$ and the average latency decreased to 1.051 s. Using the system efficiency equation, the network was **40.0%** more efficient due to the slight decrease in the number of dropped packets. Furthermore, the average latency decreased by **27.2%**.

The average queue lengths remained relatively the same, though the throughput of the intermediate nodes slightly increased. The increase in throughput was due to the decrease in the size of the packets as well as the transmission time of each packet. Therefore, if one considers all of the attributes of the heavily congested network post compression, one will find that the network performance slightly improves and is comparable to the network when slightly congested where $\lambda = \mathbf{60\%}$.

The major difference between the packets prior to and post after enabling compression and decompression within the network was the packet's payload size. The original payload size of a packet was 500 bytes; however, after compressing the payloads with LZO, the payload of the packet decreased to 399 bytes. Using equation (2.9), the payload of the packet was reduced by approximately 20%.

Even though the size of the payload was slightly reduced, the number of dropped packets and latency decreased dramatically. The decrease in values was a reflection of lower network congestion. Therefore, performing payload compression and decompression within the network was an effective way of mitigating congestion. Finally, it is important to note that the time spent to compress and decompress a packet's payload using the LZO compression algorithm was approximately 30 us and 20 us, respectively. Based on the results of the simulation, the time spent compressing and decompressing payloads is worth investing in due to the reduced network congestion and other aforementioned benefits.

5.9 Summary

In this chapter, high latency and high packet loss rates were mitigated through enabling packets' payload compression and decompression within the intermediate nodes of the network. The proposed technique used queuing theory principles, such as modeling an intermediate node as an M/D/1 queue. The model helped calculate the average waiting time for each packet in the queue of an intermediate node. This was accomplished by knowing that the generated traffic was modelled after a stochastic process and the intermediate nodes had a deterministic service time.

Source nodes generated packets according to a ***Poisson Process*** of arrival where the packet generation rate, $\boldsymbol{\lambda}$, was designed to be a fraction of the maximum forwarding rate, $\boldsymbol{N_F}$, of the intermediate nodes. To be precise, different $\boldsymbol{\lambda}$ values represented different levels of stress on the network.

The compression of packets was performed if the average waiting time of a packet in the queue was larger than the time required to perform compression. Each packet experienced compression once throughout its journey within the network. Additionally, a lossless compression algorithm, LZO, was used in the proposed scheme to ensure the integrity of the data, as well as high compression speed to avoid negatively affecting the network condition.

Chapter 6

Conclusion

6.1 Conclusion

Data networks suffer from induced congestion due to the dynamic nature of traffic as well as the incoming rate of packets exceeding the available processing resources. Given this congestion, networks may experience high end-to-end delays, high packet loss rates, low bandwidth utilization efficiency, and low throughput values. To mitigate these congestion problems, various compression techniques have been proposed.

The proposed compression techniques are adaptive in nature and manifest in different forms. Additionally, the proposed techniques utilize compression algorithms that are lossless and so the integrity of the data is maintained. The proposed techniques operate by compressing and decompressing the packets' payloads at the source-destination pair or within the network. The compression operation is done in a packet-by-packet fashion to prevent any latency induced by decompression. When packets are compressed in this fashion, the receiver can process and decompress each packet as soon as it is received instead of holding it in the buffer until a dropped packet is retransmitted.

The first compression technique proposed is the adaptive technique based on real-time RTT feedback. In this technique, three different modes utilize the RTT values in different ways to activate the compression and decompression processes for the packets. These three modes are known as passive, intermediate, and active.

In passive mode, the network utilizes performance history to define a BLRTT. The history is gathered from the initial stage of deployment where the BLRTT is the average of the gathered RTT values. Once the BLRTT is defined, the transmission phase begins. During the transmission phase, the RTT values are continuously monitored and compared to the BLRTT value. If the monitored RTT value exceeds the BLRTT by a certain threshold, compression is activated, either LZO or ZLIB. However, if the monitored RTT value does not exceed the BLRTT, transmission will continue as is.

In the intermediate mode, the lossless compression algorithms, LZO and ZLIB, are combined together in a manner similar to using AND or OR logic gates. Firstly, an initial BLRTT value is defined. Once the transmission process commences, the IRTT value is continuously monitored. If the IRTT value crosses the BLRTT value by a certain threshold, LZO compression is activated; otherwise normal transmission will continue.

After activating the LZO compression, the intermediate mode starts to define itself. While using the AND method, two conditions are monitored and must be true before activating ZLIB compression, i.e. moving into the advanced stages of the intermediate mode. To move to the advanced stage after activating the LZO compression algorithm, a newBLRTT and a threshold for an acceptable number of dropped packets must both be defined. Once the IRTT value crosses the newBLRTT value AND the number of dropped packets exceeds the acceptable limit, ZLIB compression is activated.

However, if the OR method is selected, at the stage post activating LZO compression, a newBLRTT and the threshold of an acceptable number of dropped packets are defined. The activation of the advanced stages, i.e. ZLIB compression, depends on whether the IRTT value crosses the newBLRTT value OR the number of dropped packets exceeds the acceptable threshold. In other words, ZLIB compression is active as soon as either condition is met.

The simplest of all modes that utilize RTT as a measure of whether to activate compression or not is the active mode. The active mode lacks any tolerance in comparison to the other modes. It operates by initially defining a BLRTT value. Once the packet transmission begins, the IRTT value is continuously monitored and compared to the BLRTT value. As soon as the IRTT value exceeds the BLRTT value, compression is activated.

The performance metrics used to evaluate each mode are the number of dropped packets, average RTT values, TCP efficiency, and duration of recovery. The number of dropped packets is the total lost packets in the transmission session. As the number of dropped packets decreases, the network is considered to be less congested. The average RTT is a mild indicator of whether the network is performing better or not. It does not necessarily mean the efficiency of the network increased if the RTT value decreased. In addition, TCP efficiency is the ratio that indicates the number of successfully transmitted packets, and duration of recovery is the length of time the transmission requires to return to a normal state.

In passive mode, the number of dropped packets decreased from 35 packets, when the network did not use any compression, to 24 packets while using LZO algorithm, and 19 when utilizing ZLIB. This dramatic decrease in the number of dropped packets indicates a less congested network. Furthermore the average RTT value decreased, which indicates a decrease in the end-to-end delays. Additionally, the TCP efficiency increased, indicating a lower number of retransmission due to a lower number of dropped packets. Finally the duration of recovery decreased by 0.5 s when using LZO and 0.9 s when utilizing ZLIB.

In intermediate mode, the number of dropped packets decreased from 35 packets, when the network did not use any compression, to 29 while using the AND method, and further decreased to 19 while using the OR method. The OR method had a lower number of dropped packets because it activated ZLIB compression at an earlier stage as only one condition is required to be met to activate the ZLIB compression algorithm. The average RTT values for both AND or OR methods were much lower than the RTT values of the session without any compression, thus indicating a lower end-to-end delay. In addition, the TCP efficiency values increased and the duration of recovery time decreased.

For the active mode, the number of dropped packets while using LZO was 30, which was higher than any other scheme. This is due to the active mode not being adaptive to the network behavior. However, when ZLIB was used, the number of dropped packets was 19. This was due to using ZLIB compression algorithm at an early stage, which resulted in extremely small packets for transmission. Finally, the lowest RTT value was achieved when ZLIB was used in active mode. This low value indicates the severe decrease in the end-to-end delay within the network.

The second proposed compression technique is the distributed technique where the compression and decompression processes occur in the intermediate nodes along the path between the source-destination pair. This technique uses LZO to perform the compression and decompression processes due to the algorithm's lossless property. Furthermore, each packet experiences the compression and decompression processes once throughout its journey.

For the second compression technique, the network is modelled using queuing theory techniques as an M/D/1 queue. In such a model, the expected average waiting time of a packet in the queue of an intermediate node may be calculated using the $P - K$ equation. The time a packet waits in the system is compared to the time necessary to compress the payload. If the waiting time is greater than the time needed to perform compression then the payload of the packet is compressed. The payload of the packet will be decompressed at the intermediate node prior to the destination.

In this proposed system, packets were compressed in a packet-by-packet fashion to prevent the possible decompression induced latency, as previously mentioned. The performance metrics used to measure the efficiency of the proposed technique were the number of dropped packets, the value of the throughput, the number of packets in the queue of intermediate nodes, the average latency, and the improvement in system behavior based on the reduced number of dropped packets.

To test the proposed technique under different conditions, the arrival rate of packets at the intermediate nodes was modeled according to a *Poisson Process*. The arrival rate was set at different values where each rate corresponded to a ratio of the link speeds of the intermediate nodes. For an arrival rate of $\lambda = 50\%$, the packet arrival rate, i.e., generation rate, corresponded to 0.5 Mbps.

When λ = 50%, the number of dropped packets prior to and after compression was zero packets. Even though the number of dropped packets was zero before and after compression, the latency severely decreased. For Stream 1, the average latency decreased from 0.344 s to 0.023 s. Furthermore, the average throughput slightly increased for the intermediate nodes as the packet size decreased. Finally, the queue length of the intermediate nodes drastically decreased after compression due to the increase of throughput.

When λ = 60%, the packet generation rate occupied 120% of the link's bandwidth, i.e. the processing rate of the intermediate nodes. The lack of bandwidth resulted in packet drops due to a full buffer at the intermediate nodes. The number of dropped packets was 9480 from both Stream 1 and Stream 2. However, once the compression technique was activated, the number of dropped packets decreased dramatically to 0 packets. The average latency decreased from 1.242s to 0.031s. Furthermore, the throughput increased by a great margin, which made the length of the queues decrease.

At the point when the packet generation rate was increased to λ = 80%, the incoming packet rate exceeded the available bandwidth by 60%. The lack of available bandwidth resulted in approximately 28,500 packets being dropped, which was reduced to 17,437 after activating the compression scheme. Furthermore, the average latency for Stream 1 decreased from 1.443s to 1.051s. Finally the throughput of the intermediate nodes slightly increased and the average queue lengths remained relatively the same.

It can be seen from the previous three cases that the adaptive compression technique enhanced the performance of the network. To be precise, the number of dropped packets, the average latency, and the average queue length decreased and finally the throughput of the intermediate nodes increased, thereby proving the effectiveness of the compression technique at the intermediate nodes.

In both adaptive compression techniques, the major change within the network when they were active was the packet size. The slight decrease in packet size made the network process the data more quickly. Therefore the network congestion slightly decreased, which resulted in a decrease in the average latency.

6.2 Future Work

Several additions to the proposed work can be considered as future work. These additions are considered modifications to the current operational method of the adaptive compression technique within the network.

6.2.1 First Set of Additions

The first set of additions includes two new measures to assess which packets to compress and how to handle the compression level as the utilization factor, ρ, changes. The goal of the first new measure is to prioritize the packets for compression according to size or application or both. For example, packets of larger sizes are compressed prior to packets of smaller sizes. The goal of the second measure is to change the compression level according to the value of ρ. For example, if ρ increases, the compression level should be increased to further decrease the size of the packets.

The ideal way of implementing the said measures is by having two or more separate, yet cooperative, threads to perform the compression process. For example, one thread would be dedicated to compressing packets of a relatively large size when compared to the other packets in the queue.

If all of the above additions were activated, one must prevent a possible congestion collapse from happening. A congestion collapse occurs when the buffer of the intermediate nodes becomes too large, resulting in a large queue. The queue would be so large that some packets will time-out before leaving the queue, consequently a retransmission from the source node is required. Unfortunately, the retransmitted packet will suffer from the same fate as the original packet [46].

If a packet is compressed within the network, the packet header must be manipulated to indicate that a change occurred. To perform the header update, the intermediate node must be capable of processing the header accordingly. However, the intermediate node should compare the size of the packet prior to and after compression; if the packet increases in size after compression then the intermediate node should send the original packet.

Another important addition is an algorithm to calculate the minimum size of packets to compress and the minimum size the packet can be after going through a compression process. The minimum size of packets is important for the Carrier Sense Multiple Access/Collision Detection (**CSMA/CD**). If the packet is too small, a padding mechanism should be included to increase the overall size.

Finally, an efficient mechanism must be developed such that packet compression and encryption within the network can coexist. One proposed method is the exchange of symmetric/asymmetric encryption keys between the source-destination pairs with the intermediate nodes. Another method would be using hardware encryption and decryption schemes where only specific hardware with a certain identification marker can encrypt and decrypt the data.

6.2.2 Second Set of Additions

The second set of additions includes Software Defined Networking (**SDN**) techniques to control the process of compression. A controller would instruct the intermediate nodes whether to compress the incoming packets or not. Furthermore, the controller would know the type of data being transmitted and whether the compression would be successful or otherwise. Using this coordination, the controller would save the intermediate nodes from compressing packets where compression would be redundant.

6.2.3 Simulation and Implementation

The first action must be simulating the adaptive compression techniques with aforementioned additions. If the results show an improvement in network conditions, the said additions must be included into the final version of the adaptive compression scheme. The next item on the agenda would be implementing the compression scheme on a hardware platform such as a Field-Programmable Gate Array (**FPGA**) board.

Bibliography

[1] W. Liu, L. Parziale, and C. Mathews, “Chapter 8: Quality of Service,” in *TCP/IP Tutorial and Technical Overview*, 6th ed., Vervante, 2006, pp. 287-289.

[2] R. Braden, D. Clark, and S. Shenker, RFC 1633: Integrated Services in the Internet Architecture: an Overview, RFC EDITOR, 1994.

[3] Y. Matias and R. Refua, “Delayed-dictionary compression for packet networks,” in Annual Joint Conference of the IEEE Computer and Communications Societies, 2005, pp.1443-1454.

[4] E. Jeannot, B. Knutsson, and M. Bjorkman, “Adaptive Online Data Compression,” in IEEE High Performance Distributed Computing (HPDC11), pp. 379-388, Edinburgh, Scotland, July 2002.

[5] J. Kurose and K. Ross, “Chapter 1: Computer Networks and the Internet,” in *Computer Networking: A Top-Down Approach*, 6th ed., Addison-Wesley Publishing Company, 2012, pp. 1-60.

[6] J. Kurose and K. Ross, “Chapter 2: Application Layer,” in *Computer Networking: A Top-Down Approach*, 6th ed., Addison-Wesley Publishing Company, 2012, pp. 101-102.

[7] P. Larry and D. Bruce, “Chapter 1: Foundation,” in *Computer Networks ISE: A Systems Approach*, 4th ed., Morgan Kaufmann Publishers Inc., 2007, pp. 41-42.

[8] A. Tanenbaum and D. Wetherall, “Chapter 5: THE NETWORK LAYER ,” in *Computer Networks*, 5th ed., Prentice Hall, 2011, pp. 392-404.

[9] M. Welzl, “Chapter 2: Congestion Control Principles,” in *Network Congestion Control: Managing Internet Traffic*, John Wiley & Sons, 2005, pp. 7-10.

[10] A. Tanenbaum and D. Wetherall, “Chapter 1: Introduction ,” in *Computer Networks*, 5th ed., Prentice Hall, 2011, pp. 29-40.

[11] J. Kurose and K. Ross, “Chapter 2: Application Layer ,” in *Computer Networking: A Top-Down Approach*, 6th ed., Addison-Wesley Publishing Company, 2012, pp. 83-168.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, RFC Editor, 1999.

[13] A. Tanenbaum and D. Wetherall, “Chapter 6: THE TRANSPORT LAYER,” in *Computer Networks*, 5th ed., Prentice Hall, 2011, pp. 496-498.

[14] A. Tanenbaum and D. Wetherall,“Chapter 7: THE APPLICATION LAYER,” in *Computer Networks*, 5th ed., Prentice Hall, 2011, pp. 646-683.

[15] J. Postel, RFC 793: Transmission Control Protocol, RFC EDITOR, 1981.

[16] J. Kurose and K. Ross, “Chapter 3: Transport Layer,” in *Computer Networking: A Top-Down Approach*, 6th ed., Addison-Wesley Publishing Company, 2012, pp. 185-283.

[17] J. Kurose and K. Ross, “Chapter 4: The Network Layer,” in *Computer Networking: A Top-Down Approach*, 6th ed., Addison-Wesley Publishing Company, p. 305-412, 2012.

[18] M. Arregoces and P. Maurizio, “Chapter 7: IP, TCP, UDP,” in *Data Center Fundamentals*, Cisco Press, 2003, pp. 241-306.

[19] M. Arregoces and P. Maurizio, “Chapter 12: Layer 2 Protocol Essentials,” in *Data Center Fundamentals*, Cisco Press, 2003,pp. 479-500.

[20] A. Tanenbaum and D. Wetherall, “Chapter 3: THE DATA LINK LAYER,” in *Computer Networks*, 5th ed., Prentice Hall, 2011, pp. 194-197.

[21] J. Kurose and K. Ross, “Chapter 5: The Link Layer: Links, Access Networks, and LANs,” in *Computer Networking: A Top-Down Approach*, 6th ed., Addison-Wesley Publishing Company, 2012, pp. 433-500.

[22] C. Kozierok, “Chapter 6: OSI Reference Model Layers,” in *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*, No Starch Press, 2005, pp. 102-113.

[23] A. Tanenbaum and D. Wetherall, “Chapter 2: THE PHYSICAL LAYER,” Computer Networks, Prentice Hall, 2011, pp. 90-95.

[24] M. Nelson, The Data Compression Book, Prentice Hall, 1991, pp. 1-172.

[25] R. Gallager, Course materials for 6.450 Principles of Digital Communications I, Massachusetts Institute of Technology OpenCourseWare, 2006.

[26] M. Shimamura, T. Ikenaga, and M. Tsuru, “Compressing Packets Adaptively Inside Networks,” in Ninth Annual International Symposium on Applications and the Internet, 2009, pp.92-99.

[27] J. Ziv and A. Lempel, A Universal Algorithm for Sequential Data Compression, IEEE Trans. Information Theory, vol. 23, no. 3, pp. 337-343, May 1977.

[28] LZ77 Compression Algorithm, Microsoft Developer Network, 2015.

[29] P. Deutsch and J.-L. Gailly, RFC 1950: ZLIB Compressed Data Format Specification version 3.3, RFC Editor, 1996.

[30] Deutsch, P., RFC 1951: DEFLATE Compressed Data Format Specification version 1.3, RFC EDITOR, May 1996.

[31] C. Krintz and S. Sucu, "Adaptive on-the-fly compression," in IEEE Transactions on Parallel and Distributed Systems, vol.17, no.1, pp.15-24, 2006.

[32] A. Tanenbaum and D. Wetherall, " Chapter 4: THE MEDIUM ACCESS CONTROL SUBLAYER," in *Computer Networks*, 5th ed., Prentice Hall, 2011, pp. 290-296.

[33] S. Chen, S. Ranjan, and A. Nucci, IPzip: A stream-aware IP compression algorithm, Proceedings of IEEE Data Compression Conference, March 2008, pp. 182-191.

[34] A. Shacham, B. Monsour, R. Pereira, and M. Thomas, IP Payload Compression Protocol (IPComp), RFC 3173, 2001.

[35] R. Pereira, RFC 2394: IP Payload Compression Using DEFLATE, RFC EDITOR, Nov 1998.

[36] R. Friend and R. Monsour, IP payload compression using LZS, IETF, RFC 2395, 1998.

[37] J. Kurose and K. Ross, "Chapter 7: Multimedia Networking ," in *Computer Networking: A Top-Down Approach*, 6th ed., Addison-Wesley Publishing Company, 2012, pp. 592-593.

[38] B. Constantine, G. Forget, R. Geib, and R. Schrage, RFC 6349: Framework for TCP Throughput Testing, RFC EDITOR, 2011.

[39] D. Eastlake 3rd and J. Abley, RFC 7042: IANA Considerations and IETF Protocol and Documentation Usage for IEEE 802 Parameters, RFC EDITOR, 2013.

[40] J. Kurose and K. Ross, "Chapter 4: The Network Layer ," in *Computer Networking: A Top-Down Approach*, 6th ed., Addison-Wesley Publishing Company, 2012, pp. 320-331.

[41] D. Xi and Y. Yuanyuan, "On-line adaptive compression in delay sensitive wireless sensor networks," in Mobile Adhoc and Sensor Systems (MASS), 2010 IEEE 7th International Conference on, 2010, pp. 452-461

[42] D. Bertsekas and R. Gallager, "Chapter 3: Delay Models in Data Networks," in *Data Networks*, PrenticeHall, 1987, pp. 187-188.

[43] P. Larry and D. Bruce, "Chapter 1: Foundation," in *Computer Networks ISE: A Systems Approach*, 4th ed., Morgan Kaufmann Publishers Inc., 2007, pp. 13-17.

[44] P. Larry and D. Bruce, "Chapter 1: Foundation," in *Computer Networks ISE: A Systems Approach*, 4th ed., Morgan Kaufmann Publishers Inc., 2007, pp. 41-47.

[45] R. Serfozo, "Chapter 5: Little Law," in *Introduction to Stochastic Networks*, Springer, 1999, pp. 135-152.

[46] A. Tanenbaum and D. Wetherall, "Chapter 5: THE NETWORK LAYER," in *Computer Networks*, 5th ed., Prentice Hall, 2011, pp. 393-394.

Curriculum Vitae

Name:	Fuad Shamieh
Post-Secondary Education and Degrees:	2013-2015, Candidate, Master of Engineering Science The University of Western Ontario London, Ontario, Canada
Honours and Awards:	2013, Won IBM workshop - DAPS Trophy Award. 2013, Dean's Honours List at the University of Western Ontario. 2013, Graduated Bachelor's Degree with Distinction.
Related Work Experience:	Teaching Assistant The University of Western Ontario 2013 - 2015

Publications:

F. Shamieh, A. Refaey, and X. Wang, "An adaptive compression technique based on real-time RTT feedback," in Proc. IEEE CCECE 2014.