RELIABLE DATA TRANSPORT IN HIGH BANDWIDTH NETWORKS USING FOUNTAIN CODE

BY

BHUMIKA PARIKH, B.Tech

A thesis submitted to the Graduate School

in partial fulfillment of the requirements

for the degree

Master of Sciences, Arts & Science

Major Subject: Computer Science

New Mexico State University

Las Cruces, New Mexico

May 2017

"RELIABLE DATA TRANSPORT IN HIGH BANDWIDTH NETWORKS USING FOUNTAIN CODE," a dissertation prepared by BHUMIKA PARIKH in partial fulfillment of the requirements for the degree, Master of Sciences has been approved and accepted by the following:

Dr. Louis Reyes
Dean of the Graduate School

Chair of the Examining Committee

Date

Committee in charge:

Dr. Satyajayant Misra, Associate Professor, Chair.

Dr. David Mitchell, Assistant Professor.

Dr. Jonathan Cook, Professor & Interim Department Head.

DEDICATION

Dedicated to my *Parents*.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my most sincere gratitude to my advisor Dr. Satyajayant (Jay) Misra, without his immense cooperation, vast knowledge and continuous support this research work would not have been complete. He has been like a father figure to me and has encouraged me continuously. He has been a role model for me and has taught me almost everything during my masters study.

I would like to thank Dr. David Mitchell, my committee member for his constant motivation and technical support during my research work. I am highly indebted to Dr. Jonathan Cook for being part of my defense committee in spite of his busy schedule. I would like to express my deep gratitude to Reza Tourani and Travis Mick for their great help and support during the whole period of research work.

I would also like to convey my immense indebtedness to my parents, Mukund Parikh and Geeta Parikh who have encouraged me and inspired me to be a better human being throughout my life. Their constant encouragement towards pursuing a higher education degree has urged me to successfully complete my masters education and research work.

A life without friends is like a vehicle with no wheels. I would like to take this opportunity to convey my immense gratitude to all my friends who have helped me during this important course of my life.

VITA

Education

2010 - 2014	B.Tech, Computer Science Engineering, Jawaharlal Nehru Technological Institute, India
2014 - 2016	MSCS. in Computer Science, New Mexico State University, USA

Professional Experience

Graduate Research and Teaching Assistant, Computer Science Department, NMSU, from Fall 2015 to Present

· Assisted students with classroom and lab instructions, record keeping, and explained programming concepts.

· Evaluated homework, tests, and maintained office hours to ensure students understood the course concepts.

ABSTRACT

RELIABLE DATA TRANSPORT IN HIGH BANDWIDTH NETWORKS USING FOUNTAIN CODE

BY

BHUMIKA PARIKH, B.Tech

Master of Sciences, Arts & Science

Specialization in Computer Science

New Mexico State University

Las Cruces, New Mexico, 2017

Dr. Satyajayant Misra, Chair

High bandwidth networks have transformed the Internet today and the Transmission Control Protocol (TCP), which is widely used transport protocol, face design issues in respect of performance tuning mainly due to its congestion control mechanism. Research history, modifying this congestion control mechanism shows that it is difficult to find an optimal solution that would meet all the requirements of high bandwidth networks. As an alternative approach, recent studies suggests on replacing this congestion control with erasure coding techniques for reliability. In order to investigate this new approach, in this research we purpose a new Application Layer Forward Error Correction code based

User Datagram Protocol (ALFEC–UDP) where an efficient and one of the most advanced fountain based erasure codes is applied on the top of UDP transport protocol to recover the lost packets and thus improve the reliability and meet all the requirements of high speed communication. To evaluate its performance, we compare our protocol with some high speed TCP variants and present analytical and simulation results using Network Simulator (ns-3).

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Background and Motivation

The nature and quality of the service requirements of the Internet have changed drastically in the last two decades. According to the statistics of Cisco's Visual Networking Index Forecast (2015-2020) [1]: high bandwidth traffic makes 70% of IP traffic today and would rise to 82% by 2020. Applications such as IPTV (PPStream, PPLive), global transfers of analytical datasets, transport of ultra-high 3D resolution data used in scientific and medical research need high bandwidth, fast and reliable communication. In the first International Workshop on Protocols for high speed and long distance networks [2], several presentations were put forward on requiring the transfer of multi-GB and multi-TB datasets over the gigabit networks per second.

Transmission Control Protocol (TCP), which is a most widely used transport protocol, is used in these scenarios; and therefore its import aspect – Flow and Congestion control mechanism should perform well in high bandwidth networks. But, as bandwidth or the delay goes on increasing, TCP shows low performance and reacts adversely mainly because of its conservative congestion avoidance algorithm. The slow-start phase of TCP increases the transfer speed exponentially and causes a large number of packet drops once the channel capacity is filled. TCP wastes a considerable amount of bandwidth once congestion avoidance phase is

reached due to detection of packet loss taking it a long time to fill the capacity (refer figure 1.1).

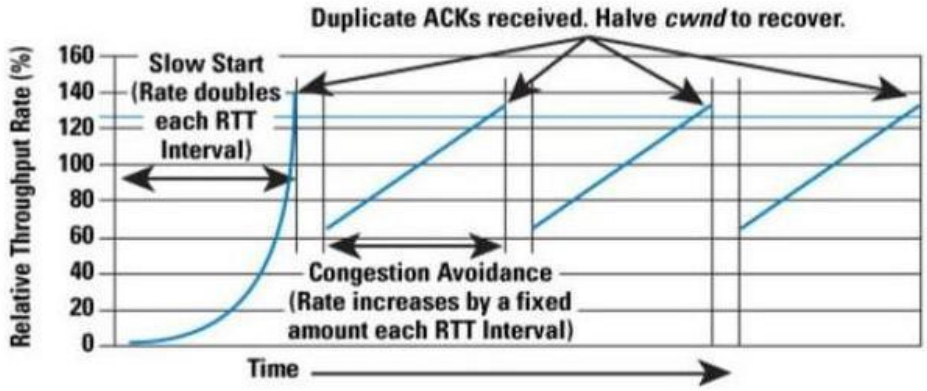

Figure 1.1: TCP behavior in high bandwidth networks [3]

To address this performance issue of TCP in high speed networks, researchers have come up with many proposals for modifying the congestion avoidance algorithm in past two to three decades. Some of them are High Speed TCP [11], H-TCP [19], BIC TCP [40], CUBIC TCP [15], YeaH TCP [5], Compound TCP [38]. But none of these variants act as a universal mechanism for improving challenges such as efficiency and bandwidth utilization in ever changing high bandwidth Internet [20] [4].

Motivation: Concerning the limitations of TCP, there is a significant justification on replacing the concept of this protocol, specifically its congestion control mechanism. Some ideas have been proposed by Global Environment for Network Innovations (GENI), suggesting a future Internet without congestion control mechanism, in which they motivate to use efficient erasure coding schemes to recover packet loss or error. Researchers have also implemented the erasure codes in the network communication. Sundararajan et al.,[37] use TCP/NC that incorporates network coding into TCP with only minor changes in the protocol stack. It

helped to increase efficiency by correcting packets that got lost due to congestion, but the congestion avoidance mechanism was not replaced totally. López et al., [22] investigated a fountain code based protocol using game theory. But, its performance in the network was similar to the performance of TCP. Botos et al., [8] presented new transport protocol for high loss environments using rateless codes making modifications to slow start and congestion avoidance phase. These modifications resulted to avoid the dramatic decrease of sending rate in high packet loss environment. Bonald et al.,[7] analyzed the network behavior in the absence of congestion control and their results confuted the common belief that network communication without congestion control would collapse. Thus, omitting congestion control and applying erasure codes show one of a realistic approach to network communication.

This research is motivated by the above observations. We take a practical step in implementing a new transport protocol, by applying fountain codes on the application layer of Internet Protocol(IP) stack, called as Application Layer Forward Error Correction based User Datagram Protocol (ALFEC–UDP).

We use User Datagram Protocol (UDP) as a transport protocol, since, it increases the speed of data transport by providing least transmission delay and best effort service. UDP reduces delay in transmission by omitting the connection process, flow control, error recovery and retransmission mechanisms. As a secondary benefit, UDP does not care about the congestion avoidance mechanism while transferring data on high bandwidth networks.

As applications using high bandwidth capacity links require reliability, packet loss recovery and error correction mechanism is implemented by applying erasure codes at the application layer of UDP. Rateless erasure codes unlike traditional erasure codes, do not have fixed coding rate and thus can a produce

potentially infinite stream of the codeword as long as it is necessary for the actual transmission to complete. The decoder can recover the sent information from any subset of the encoded stream, which is slightly larger than the original message. When the loss is experienced in the network, receiver hosts would only have to collect a subset of the encoded stream until they obtain a require amount of it to process decoding.

The first practical rateless codes introduced were Luby Transform (LT) codes. But, they failed to provide low complexity in encoding and decoding operations. We propose the use of Raptor erasure codes, [33] which has been recently declared to be one of the most powerful rateless erasure codes in Forward Error Correction (FEC) axis, in this research. Being an extension to LT codes, they offer linear time encoding and decoding complexity thus decreasing the processing time.

1.2 Contributions:

The contributions of this research include:

- Design of ALFEC–UDP protocol for reliable data transport in high speed networks.
- Implement the protocol to analyze and demonstrate its efficiency through comparisons with present high speed TCP variants.

1.3 Thesis Organization

In chapter 2, the related work on the development of UDP as a transport protocol and applications of raptor codes in network communication is presented. Chapter 3 discusses the terminology of fountain codes, erasure channel, and systematic raptor codes. In Chapter 4, we discuss the design and methodology of our proposed protocol. Chapter 5 presents the simulation topology, experiment

results, and analysis of our protocol. Chapter 6 concludes and presents the scope for further work.

Chapter 2

LITERATURE REVIEW

In this chapter, a literature review of UDP and rapidly evolving Raptor Codes is being discussed with a view of revealing what have been achieved in recent years by the researchers.

2.1 User Datagram Protocol

The UDP, a transport layer protocol, relies on connectionless communication mechanism which does not establish any connection before sending the packets. Its packets typically include a header, containing source and destination address information, the length of the payload, checksum, as well as a payload (the actual application data). The purpose of the checksum is to detect errors in the transmitted packets. UDP simply sends the packets with high speed into the network. It is faster than TCP because there is no flow control, error checking, error correction, or acknowledgment. The only cause of erasures in the packets is because of the collision in the network.

The UDP protocol only employs a cyclic redundancy check (CRC) to verify the integrity of packets, in a known manner. This mechanism can detect erasures either in header or payload and discard the packet if an error is detected. When a packet fails to arrive before its processing time, the UDP protocol declares the packet as lost. Therefore, at the receiving host, packets are either perfect or completely lost.

UDP does not perform well since there is no form of error correction mechanism other than CRC. Large packet losses and underutilization of link bandwidth are still present in the network packet communication. Without any good error correction mechanism, high throughput and moderate latency that is required for real-time applications would not be achieved. Within last decade some intense research is going on into deploying error correction in UDP by various researchers.

Chen et al.,[9] in their work developed a novel hybrid protocol. The two transmission protocols used were UDP and TCP along with FEC for approaching to the problem of packet loss. In their study, they found that delay introduced at the point of establishing the connection was too high. Furthermore, the introduction of UDP completely removed congestion control, giving rise to huge packet drops and with the introduction of TCP the bandwidth usage was increased. The authors also found that TCP behaved poorly even when congestion control was disabled and FEC required huge overhead for correcting errors in the communication.

UDP being an unreliable transport protocol, attempt to introduce retransmission of lost packets take a long time when retransmissions are ignored. FEC is proposed due to this resilience of the packets in which redundancy reduces the packet loss. Despite this, uncontrolled use of redundancy leads to high bandwidth utilization. Thus, Johanson [16] proposed an adaptive FEC scheme, using Reed-Solomon (RS) erasure coding and receiver feedback. This scheme limits the bandwidth utilization by adapting to the packet video size. But, for a larger encoded frame size, the utilization of the bandwidth is increased when the feedback receiver is introduced.

While most of the research focused their attention on implementing the features of TCP like retransmissions of lost packets, introducing feedback; in this

study, raptor codes were implemented to an application layer FEC in order to improve the performance of UDP.

2.2 Raptor codes

Raptor FEC is an erasure correction technology, capable of correcting the missing or lost data without the need of retransmission from the sender. The versatility of raptor codes is evident in that as it constitutes an efficient rateless code which provides a near-Shannon performance. These codes have been proposed to provide reliability for a variety of data delivery applications.

Shokrollahi [34] described the evaluation of raptor codes compared to other fixed rate fountain codes. In his work, he explains the discovery of Luby Transform (LT) codes. But compared to raptor code, LT codes did not provide the linear time encoding and decoding. After the invention of raptor codes on BEC, research was carried out by him on the application of raptor codes on packet based transport, Binary Symmetric Channel (BSC) and Additive White Gaussian Noise Channel (AWGC) [36].

Digital fountain codes were used for file distribution over the wireless broadcast networks [24]. The ideal ability to recover the loss, requiring low computational complexity and fountain codes flexibility made additional means of a reliability of using raptor codes for delivering large file sizes simultaneously without feedback by many users over unreliable and bandwidth-limited networks.

Kim et al.,[18] in their research compared the Reed-Solomon (RS) code at the link layer and raptor code at the application layer in the burst mode transmission to support reliable transport for Internet Protocol television (IPTV). The Maximum Likelihood (ML) decoding based on Gaussian Elimination (GE) algorithm ensured that multimedia broadcasting is possible with more speed and

reliability. Packet loss during the packet based transport is well modeled by the Binary Erasure Channel (BEC) was proved by their research.

Luby et al., [25] address reliable file delivery over mobile broadcast networks, concentrating on the raptor codes, as specified in MBMS and 3GPP. When compare to LT and raptor codes, their research concluded raptor provided low encoding and decoding complexity. The guidelines for the efficient functioning of encoding and decoding along with results is shown in their research. Luby et al., [26] in his another research confirm that Application Layer FEC (ALFEC), based on raptor code on the top of IP provides reliability to packets sent during the network transmission in the Internet communication.

Thomas et al., [35] incorporated ALFEC into the Content Delivery Protocol (CDP) and proved that mobile communication system ensured reliability in delivery of content together supporting channel efficiency, low complexity, and flexibility.

The idea of replacing congestion control with fountain code, raptor, was also proposed by Molnàr, Sàndor, et al., [31]. They implemented a novel transport protocol which uses TCP for data transfer without congestion control and showed its goodput performance is better than current TCP variants in a wide range of packet loss rates and round-trip delays environment.

In conclusion, the available research work suggests that raptor codes have wide range of applications in network communication. The use of raptor codes has been implemented to all channels, offering platform for both streaming and file downloading services. Also, most of the research has augmented raptor codes in TCP and UDP. In this work, sn emphasis is placed on implementing raptor codes on UDP as an ALFEC-UDP.

Chapter 3

RAPTOR CODES

The goal of this chapter is to briefly review some major concepts which have been used in this thesis. The following section introduces the FEC over time varying and an unknown erasure channel which is considered as the model of the channel under study. Next, we discuss the properties, encoding and decoding scheme of raptor FEC codes.

3.1 Forward Error Correction and Erasure Channel

A FEC code as described in [13] is a scheme that controls error/losses in the data transmission. The sender encodes the message in a redundant way by using an error-correcting code [28] [32], this redundancy allows the receiver to detect a limited number of errors that may occur anywhere in the message, and correct these errors without feedback with the sender.

An erasure channel is a communication channel model where a transmitter sends a bit (a zero or a one), and the receiver either receives the bit or it receives a message that the bit was not received ("erased").Though this channel was more of a theoretical concept when it was introduced, it is a perfect model in many existing and emerging communication systems.

Probably the most practical erasure channel is a link in a packet network such as the Internet. The packets transmitted on these links are either received without error or not received. More generally, any noisy channel protected with

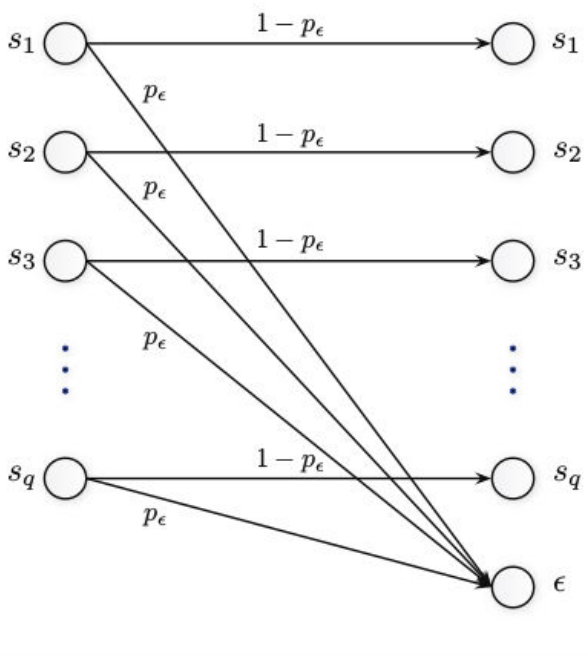

Figure 3.1: A q-ary erasure channel with an input alphabet of size q, and a output alphabet of size $q+$ 1.

an error-correcting code behaves like an erasure channel since the received symbol is either decoded or completely useless.

The model of a q-ary erasure channel could be described as in figure 3.1. The set of input symbols of such channel is $\{s_1, s_2, \ldots, s_q\}$, while output set consists of all possible inputs and an erasure ϵ. Each input symbol is assumed to have a $1-p_\epsilon$ probability of successful transmission, and will be erased by channel with probability p_ϵ.

Raptor codes can be defined as a symbols belonging to an alphabet q, and define multiplication and addition over the set of this alphabets. As an obvious choice for the symbol's alphabet is then a Galois field of size q which is referred as *GF(q)*. The coefficients for the random linear combinations will also be selected from the same alphabet.

3.2 Fountain Codes

Fountain codes, also known as "*rateless*" codes or "*Digital Fountain*" are class of erasure codes with the property to generate a theoretically endless stream of output symbols [6] [21]. Where each symbol provides partial information about the original symbols, since each output symbol is a random linear combination of the original message symbols. Once the receiver receives required amount of symbols, its starts to recover the original information. Figure 3.2 shows encoding of fountain codes briefly and following is the terminology:

Symbol

A symbol is defined as a smallest unit of data. These units are measured in bytes, also known as the symbol size. Each gray color box represents a symbol in fig 3.2

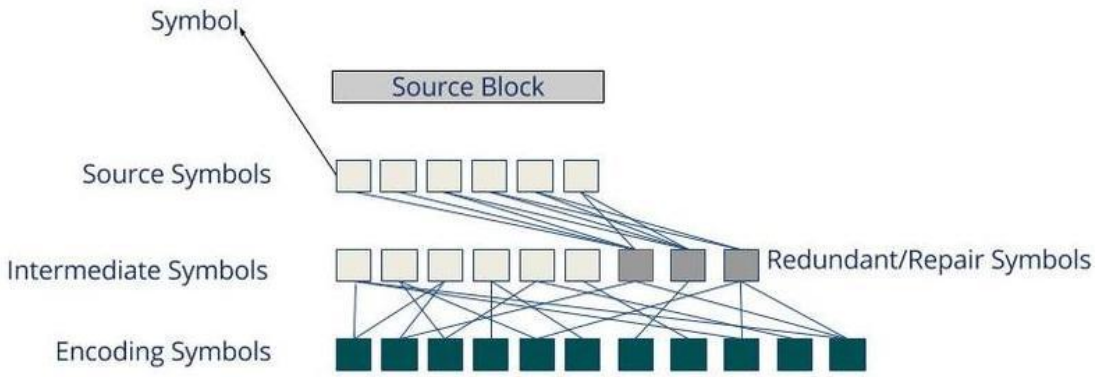

Figure 3.2: Fountain codes terminology

Source symbols

The smallest unit of data used during the encoding process. All source symbols within a source block have the same size.

Source Block

A block of source symbols that are considered together for Raptor encoding purposes.

Redundant/Repair symbols

Redundant symbols are generated from a source block and have the same size as the source symbols of that source block.

Intermediate symbols

Symbols generated from the source symbols using an inverse encoding process. The repair symbols are then generated directly from the intermediate symbols. The intermediate symbols are not included in data packets.

Encoding symbols

A symbol that is included in a data packet. The encoding symbols consist of the source symbols and the redundant symbols.

3.3 Raptor Forward Error Correction Code

Raptor Code is called either the Raptor 10 code or the R10 code. This is a class of fountain code that has the capability of Forward Error Correction (FEC). R10 is a also fully-specified FEC scheme, that is, independent implementors can implement both the encoder and the decoder from a specification that is an Internet Engineering Task Force (IETF) Request For Comments (RFC).

Raptor codes can be realized in two ways: the non-systematic and the systematic raptor codes. The method used in encoding process is notable difference between the two kinds. The properties of raptor codes include linear time encoding [25] (independent of the quantity of repair symbols generated) and linear time decoding (independent to amount of loss), ideal performance under any channel loss conditions, and ability to support large file size.

Raptor Codes used in this research are systematic codes, that are, redundant data can simply be appended to the original symbols, and receivers do not need to recover the original symbols if received correctly. Raptor code is designed to support a number of original symbols K between 4 to 8192. This section gives a detail construction of R10 code. We make references to the specification of this code from RFC 5053 [23] and use the same terminology.

3.3.1 Overview

Raptor codes as shown in fig 3.3 are a concatenation of a pre-code and LT code [17], with pre-code usually being Low Density Parity Check (LDPC) [12] code, defined over Galois field with 2 elements *GF(2)*. This pre-code maps K original symbols (or for a group of original symbols) onto L intermediate symbols (L > K). In the next step, each encoded symbol is formed by performing an exclusive-or (XOR) operation on randomly selected subsets of symbols, chosen from L intermediate symbols, based on degree distribution as specified in section 5.4.4.2 in [23]. The operation of generating encoded symbols constitutes the LT code which is responsible for the rateless property of Raptor codes – that is the ability to generate as many symbols as needed on-the-fly.

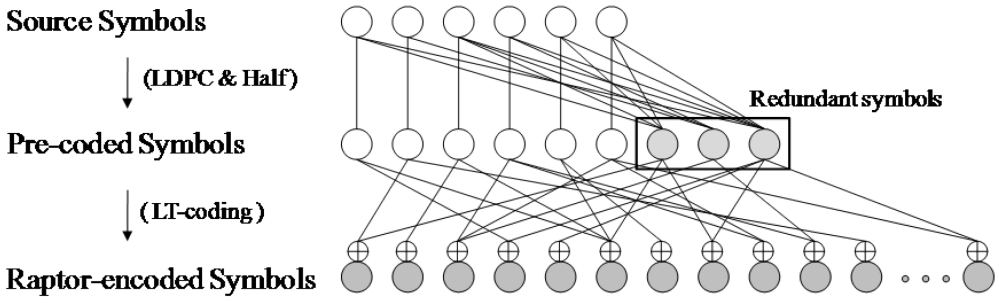

Figure 3.3: Two encoding steps of Raptor codes, LT and LDPC [14]

Section 3.3.2 describes the systematic Raptor code encoder. A number of possible decoding algorithms are possible. The efficient decoding algorithm used for this thesis simulations is described in section 3.3.3.

3.3.2 Encoder

The encoding process is summarized in two main blocks, namely *Code Constraint Processor* and *LT Encoder*. They both are represented by their equivalent generator matrices. A block diagram of Systematic Raptor Encoder/Decoder is shown in figure 3.4.

Code Constraint Processor

Let $\mathbf{t}$ denote K $(4 < K < 8192)$ original symbols that are to be encoded. The $\mathbf{d}$ that is given as input to Raptor encoder, contains $L = K + H + S$ symbols. Where S is the number of Low Density Parity Check (LDPC) redundant symbols and H is the High Density Parity Check (or) half symbols. These symbols depend on large number of original symbols, defined over GF(2) as:

$$\mathbf{d}_{[0:L-1]} = [\mathbf{z}^T \ \mathbf{t}^T]^T \tag{3.1}$$

where $\mathbf{z}_{[0:\mathrm{H}+\mathrm{S}-1]}$ is a vector of zeros. The parameter S is the smallest prime number such that

$$S \geq \lceil 0.01 \times K \rceil + X \tag{3.2}$$

where X is smallest positive integer such that

$$X(X-1) \geq 2K \tag{3.3}$$

Similarly, H is the smallest integer such that

$$\binom{H}{\lceil H/2 \rceil} = \frac{H!}{2(H/2)!} \geq K + S \tag{3.4}$$

H is chosen like this so that the columns of the matrix G_{Half} in matrix of equation 3.6 are all distinct.

Intermediate symbols, $\mathbf{c}$ are generated when $\mathbf{d}$ multiplies with the inverse of the pre-coding matrix $\mathbf{A}$ as:

$$c_{[0:L-1]} = A^{-1}_{L \times L} \cdot \mathbf{d}_{[0:L-1]} \tag{3.5}$$

with

$A_{L \times L}$ =

$(G_{LDPC})_{S \times K}$	$(I)_{S \times S}$	$(Z)_{S \times H}$
(G_{Half})		$(I)_{H \times H}$
(G_{LT})		

(3.6)

where submatrices $\mathbf{I}_S$ and $\mathbf{I}_H$ are identity matrices, and $\mathbf{Z}$ is a zero sub-matrix with dimension $S \times H$; $\mathbf{G}_{LDPC}$ is a $S \times K$ low density parity check (LDPC) generator sub-matrix and is defined as follows:

$$G_{LDPC} \cdot [c[0], ..., c[K-1]]^T = (c[K], ..., c[K+S-1])^T \tag{3.7}$$

G_{Half} is a $H \times (K + S)$ generator sub-matrix of the Half/HDPC symbols, defined as:

$$G_{Half} \cdot [c[K+S-1], ..., c[K-1]]^T = [c[K+S], ..., c[K+S+H-1]]^T \quad (3.8)$$

G_{LT} is a $K \times L$ generator sub-matrix in $\mathbf{A}$ for the first K symbols that contributes a Raptor codes systematic:

$$G_{LT} \cdot [c[0], ..., c[L-1]]^T = [t[0], ..., t[K-1]]^T \quad (3.9)$$

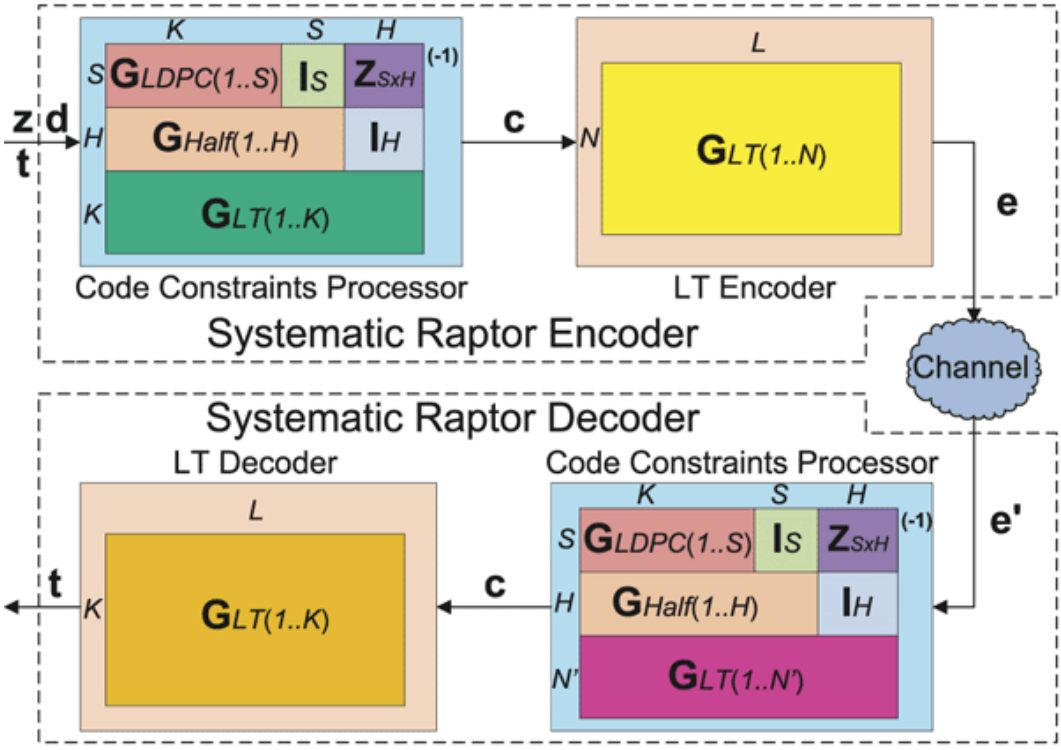

Figure 3.4: Block diagram of Raptor Encoder/Decoder [30]

LT Encoder

The source vector **t** is pre-processed by the intermediate symbols generator, then the LT encoder generate any number of encoded symbols **e** according to:

$$G_{LT} \cdot c = \mathbf{e}_{[0:N-1]} \tag{3.10}$$

where G_{LT} is an $N \times L$ LT generator matrix, with $N \geq K$. The value N is selected to be sufficiently larger than K to compensate for the possible loss of encoded symbols during transmission and thus to make matrix **A** invertible at the decoder side.

$$e[i] = d[H + S + i], \forall_{i=0,\ldots,K-1} \tag{3.11}$$

This is because G_{LT} for symbols 0,..., $K - 1$ is included in the pre-processing matrix **A**, making Raptor code systematic.

For a fixed value of K, submatrices $\mathbf{G}_{LDPC}$, $\mathbf{G}_{Half}$, $\mathbf{G}_{LT}(0, \ldots, K-1)$ are pre-generated once and stored in the memory of the sender. The structure of G_{LT} matrix shows how the encoded output symbols **e** are generated from the intermediate symbols **c**. The "1" values on the g^{th} row of matrix $\mathbf{G}_{LT}$ identify the intermediate symbols that are XORed to generate encoded symbol e[g].

Before the encoded symbols are transmitted, they are grouped and augmented with an Encoding Symbol Identifier (ESI). Encoded symbols with ESIs 0, . . . , $K - 1$ corresponds to source symbols and the ESIs K, $K + 1$. . . corresponds to redundant (or) repair symbols. This is advantageous, as these ESIs help the decoder to distinguish the original symbols and the redundant symbols while decoding; the details which are omitted from the diagram for simplicity.

3.3.3 Decoder

The Raptor decoder exchanges the position of *Code Constraint Processor* and *LT Encoder*, more precisely *LT Decoder*, as illustrated in the figure 3.4. The input vector $\mathbf{e'}$ containing N' $(K \leq N' \leq N)$ encoded symbols (may not be in order) is padded with $S + H$ zeroes to size it to $(M = N' + S + H)$. Starting with $N' = K$ the value of N' is iteratively incremented to make the matrix $\mathbf{A}$ invertible. The difference $(N' - K)$ is equal to or greater than the number of received encoded symbols lost in the channel.

The decoding is performed according to:

$$c_{[0:L-1]} = [\mathbf{z}^T \, \mathbf{e'}^T] \cdot A^{-1}_{M \times L} \tag{3.12}$$

$$t_{0:K-1} = \mathbf{G}_{LT} \cdot \mathbf{c}_{[0:L-1]} \tag{3.13}$$

where $\mathbf{G}_{LT}$ is a $K \times L$ LT generator matrix.

At the decoder side the sub-matrix $\mathbf{G}_{LT}(1..N')$ is first built from the input data. The ESI of the n^{th} received encoded symbol is used to generate the n^{th} row of the sub-matrix $\mathbf{G}_{LT}(1..N')$ through the LT encoding process.

Decoding a source block is equivalent to decode $c_{[0:L-1]}$ from known matrix $\mathbf{A}$ and vector $\mathbf{e'}$. It is clear from equation 3.12 that $c_{[0:L-1]}$ can be decoded if and only if the rank of matrix $\mathbf{A^{-1}}$ over *GF(2)* is L. Once $c_{[0:L-1]}$ is decoded, the source symbols can be recovered. ‘3.13.

The decoding algorithm of Raptor code uses Gaussian Elimination (GE) (using row operations and row and column reordering) to transform $\mathbf{A}$ to $L \times L$ identity matrix after discarding $M - L$ rows.

The transformation process is summarized below into four phases and is shown in figure 3.5

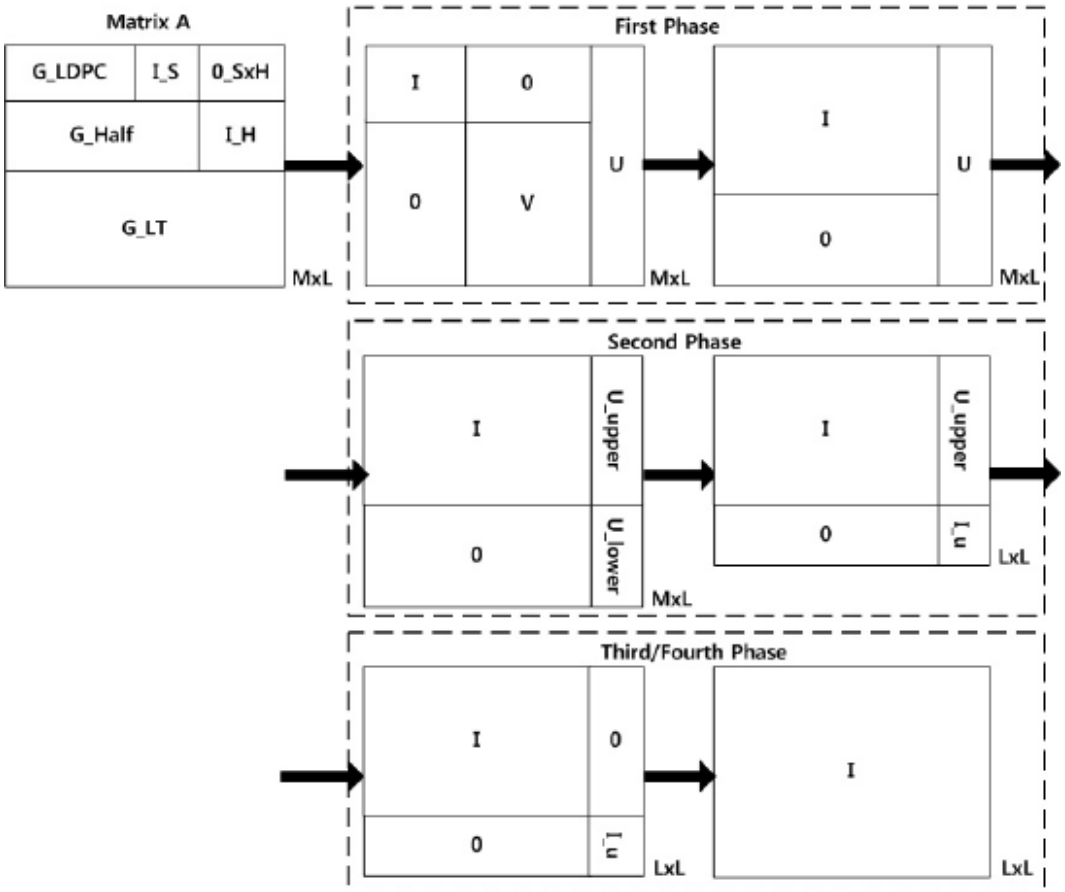

Figure 3.5: Illustration of four phases to transform matrix A into an identity matrix [39]

In the first phase, the matrix **A** is converted to three sub-matrices: I, Zero sub-matrix, and U as shown in Figure. In this figure, V is an intermediate matrix. In the beginning of phase one, V = A and I and U are Null matrices.

For each iteration, a row of V, with the minimum non-zero degree r, is chosen, where the degree of a row is defined as the number of elements "1" it has. Then, the chosen row is exchanged with the first row in V. All the columns of V are reordered so that the first and the last r 1 elements of the first row are "1".

Then, the other rows which have ones in their first column are XORed with the first row so that their columns become "0". The rank of sub-matrix **I** is increased by 1 and the number of columns in U is increased by r1. This process is repeated until the sum of columns of I and U is equal to L. This decoding phase fails when no non-zero row remains in V before V disappears, that is, the sum of columns of I and U is less than L and elements in all rows of V are zero.

In the second phase, the sub-matrix U is partitioned into U_{upper} and U_{lower}, where U_{upper} consists of the first i rows and U_{lower} contains the remaining M − i rows. If the rank of U_{lower} is less than u, the decoding of second phase fails. Otherwise, GE is performed on U_{lower} to transform it into an identity matrix with rank u. Then, the last M − L rows of $A_{M \times L}$ are discarded.

In the third phase, a pre-computation matrix U' with ceil(u/8) × 255 rows and u columns, is generated based on I_u.

In the fourth phase, the matrix U' is used to zero out U_{upper} and thus converts A into the L × L identity matrix.

Successful decoding depends on the transformation process discussed above. The intermediate symbols $c_{[0:L-1]}$ can be decoded in the meantime, based on the row operations and row/column exchanges occurring during the transformation process.

3.4 Raptor Code applications and advancements

Fountain Codes have been adopted by the Third Generation Partnership Project (3GPP), Digital Video Broadcasting (DVB), Internet Protocol Television (IPTV) and Internet Engineering task Force (IETF). According to the Digital Fountain, raptor and its most advanced code raptorQ [29] code is claimed as one of the most advanced forward error correction code for data networks.

Chapter 4

DESIGN METHODOLOGY

The performance evaluation of ALFEC–UDP was carried out through simulation using network simulation (ns-3). In this chapter, ns-3 is described briefly in section 4.1. In sections 4.2 and 4.3 system design and the methodology that has been employed for building ALFEC–UDP is explained.

4.1 Network Simulator-3

Network Simulator-3 is a discrete event network simulator used in research and education. The simulator is written from scratch using C++ and Python programming language by a team lead by Tom Henderson and funded from the U.S. National Science Foundation (NSF).

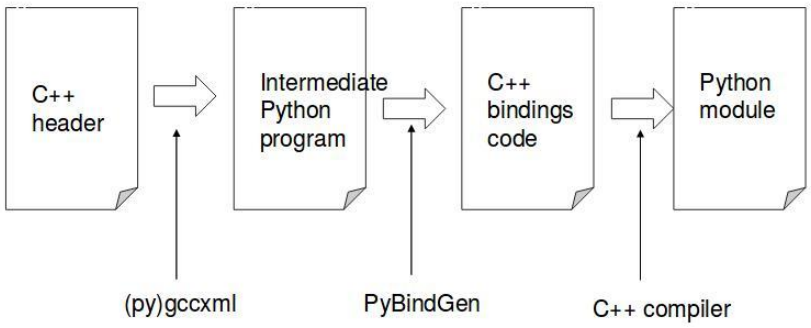

Figure 4.1: PyBindGen used to generate python bindings for all libraries

The ns-3 library is wrapped by Python and the library PyBindGen is used for parsing the headers to automatically generate the corresponding C++ files.

Fig 4.1 shows how library PyBindGen helps to generate Python bindings for all libraries.

Ns-3 provides models of how packet data networks work and perform, and provides a simulation engine for users to conduct simulation experiments. Most of the ns-3 API is available in Python, but the models are written in C++. User programs can be written in both C++ and Python.

Ns-3 is organized into two-level of software and libraries. The one is ns-3 modules that exist within the ns-3 directory as shown in fig 4.2. This simulator supports a series of modules such as routing protocols, ad-hoc networking, and access technologies while providing simulation results for wired and wireless networks alike. The other level that ns-3 supports is libraries and software that are not bound within ns-3; affording a high level of independence to researchers, allowing them the opportunity to extend the simulator by modifying the source code and to contribute to the existing module as a result of adding new functionality, thus advantageous for abstraction, emulation, scenario generation, visualization and extensibility.

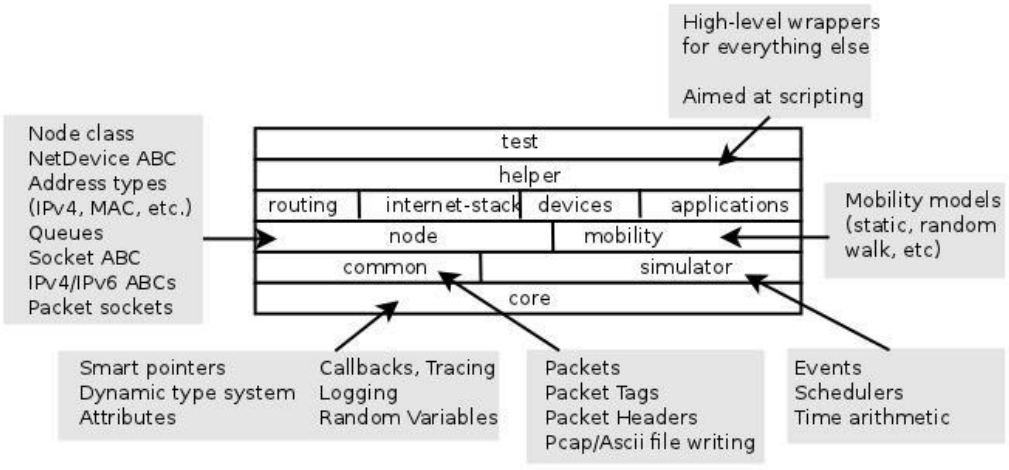

Figure 4.2: Organization of ns-3 modules directory

4.2 System Design

In this design, UDP was selected as the transmission protocol for the purpose of data transmission at the transport layer. UDP is the best fit for this design as it avoids retransmissions and delays. As UDP does not provide reliability on the datagram delivery, measures should be taken to incorporate such reliability above the transport layer. In order to achieve reliability in UDP, techniques such as Automatic repeat reQuest (ARQ) or the FEC is to be considered [10]. As UDP is aimed to transmit data to a large number of users, ARQ will be less efficient; therefore, fountain codes are used above the UDP to achieve reliability.

Raptor code is implemented at the application layer (AL). The AL sends the packets through the raptor encoders, that process the data by adding redundancy before sending to the UDP transport layer. Once this has been processed, the encoded symbols are sent to lower layers for further transport to the receiver as shown in figure 4.3. At the receiver side, the stream of encoded symbols is

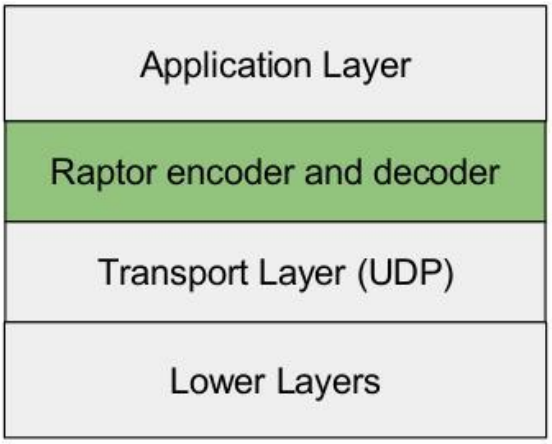

Figure 4.3: System Design of ALFEC–UDP

received from all lower layers to the AL. At this point, the encoded symbols are fed to the decoder and the data is being recovered successfully.

4.3 Methodology

The network simulator ns-3 does not contain an implementation of the raptor FEC. Thus, the design and building a functionality module of ALFEC is necessary for ns-3. The development of the ALFEC–UDP was carried out into two stages. In the first stage, the standalone raptor FEC as described in section 4.3.1 module was developed to evaluate the encoding and decoding capabilities in terms of various input parameters, overhead and file sizes. The second stage refers to the integration of raptor codes into ns-3 as explained in section 4.3.2.

As ns-3 is an integrated layering system, it allows to tightly couple the application, raptor, and transport layers. This makes raptor codes fit in between the application and the transport layer together supporting the capability of providing required information using the headers. The advantage of integrating raptor codes with ns-3 is that it would offer required visualization of network architecture and speed to run the simulation.

4.3.1 Standalone raptor FEC module

The design of the standalone raptor FEC module is developed in C++ with two entities: namely encoder and decoder. The communication between the encoder and decoder did not exist directly as shown in figure 4.4 but was established through Array_Data_Type class. During the standalone simulation, the binary reference of this data type of this class is passed from the encoder to the decoder. The implementation of the raptor encoder and decoder was carried out using the following classes:

R10_Encoder

This class was invoked as soon as the data is made available to encode. It calls the class Array_Data_Type. The parameters such as *K, S, H, L* representing the length of source symbols, the number of LDPC symbols, the number of

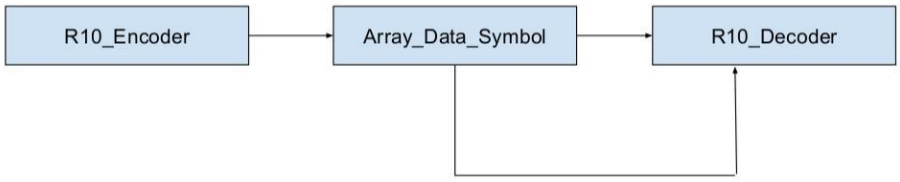

Figure 4.4: Overview of raptor encoding and decoding as standalone module.

Half symbols and the number of intermediate symbols for a single source block respectively are computed in Array_Data_Type class.

Inter_Symbol_Generator

The Inter_Symbol_Genrator class was used to generate the pre-code matrix, that is the intermediate symbols (or) Code Constraint Processors symbol matrix, as described in section 3.3.2. The matrix generated in this phase, and defined in equation 3.6 is stored in the data type Array_Data_Symbol. This matrix represents a single block of code K and is invertible, further used for encoding the message. This matrix is a must at the receiver end to start decoding process.

LT_Encoding

The LT_Encoding class implements the LT encoding, as defined in section 3.3.2. After the intermediate symbols are generated, the LT encoding is done by this class by XORing the intermediate symbols with degree distribution d, chosen randomly defined at section 5.4.4.2 in [23]. Parameters, such as degree distribution, symbol length, the length of data is stored in the header of Array_Data_Symbol.

Array_Data_Type

The Array_Data_Type class serves as an intermediate link between the encoding side and the decoding side of the raptor code. The matrix, encoded symbols

and other information used during encoding and decoding process are stored in Array_Data_type class having Array_Data_Symbols data type. A reference of this data type stored in a binary file and this is read by the receiver which allows the receiver to retrieve the encoded symbols generated by the encoder.

R10_Decoder

The receiver after reading the binary file converts the reference to type Array_Data_Symbol and then invokes the class R10_decoder as soon as required encoded symbols are received. A subset of the matrix is used during the decoding operation to help the recovery of intermediate symbols. The retrieval of intermediate symbols operation is done by class Get_Inter_Symbols of type Array_Data_Symbols. The aim of this class is to convert the sparse matrix into identity matrix using Gaussian Elimination (GE) process. Thus, intermediate symbols would be retrieved.

The class Inter_Symbols_Decoding receives the intermediate symbols and performs the process of decoding to recover the original source symbols.

4.3.2 Integration of raptor FEC module into ns-3

The first step in integration and development of ALFEC–UDP protocol was to extend the raptor standalone module as explained in section 4.3.1 into the application layer of ns-3. This was done by extending the raptor module directory using the python binding script at the application level and building ns-3 again.

The second step was to define the three new packet headers at the application level. RPSendHeader, RPRecvHeader, and CDPHeader as shown in figure 4.5 were defined for the implementation of the raptor codes and were integrated into the application module of ns-3 with the application being udp-echo-client and udp-echo-server. The CDPHeader at the sender side was used to begin the communication by communicating Content Delivery Protocol (CDP) specifications

fields such as total data length, number of source blocks and symbol length to the receiver. The RPSendHeader also defined at the sender side contained fields such as source block number and ESI of the packets being transferred and RPRecv-Header at the receiver was defined to acknowledge that it was ready to process the next block.

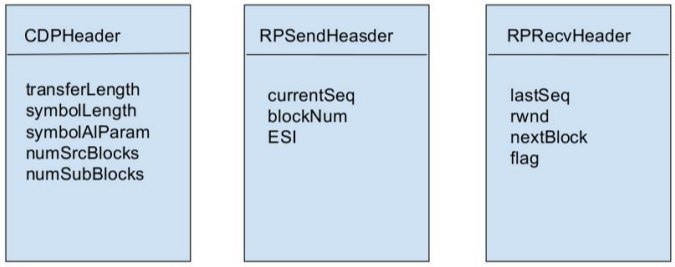

Figure 4.5: Packet header structure

After defining the headers, the following figure 4.6 represents the structure of ALFEC–UDP protocol's packet structure used for communication in ns-3.

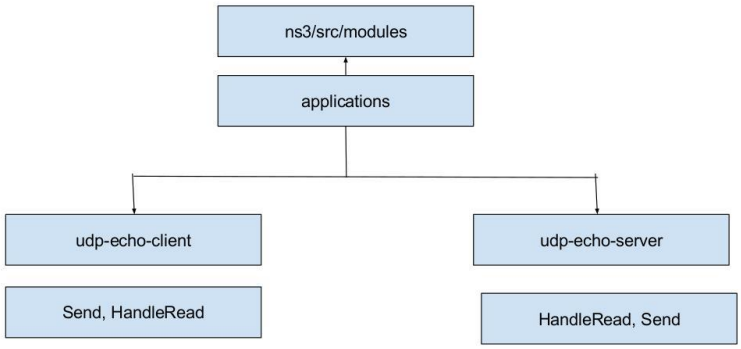

Figure 4.6: Linkage of packet structure in ns-3

The functionalities of sender and receiver are separated in the course of the protocol using udp-echo-client and udp-echo-server that acts as the sender and the receiver respectively.

The transmitter

Data transmitter have three basic functionalities: namely Schedule send (Tx), wait (W), HandleRead (Rx) as shown in figure 4.7. As soon as the socket is

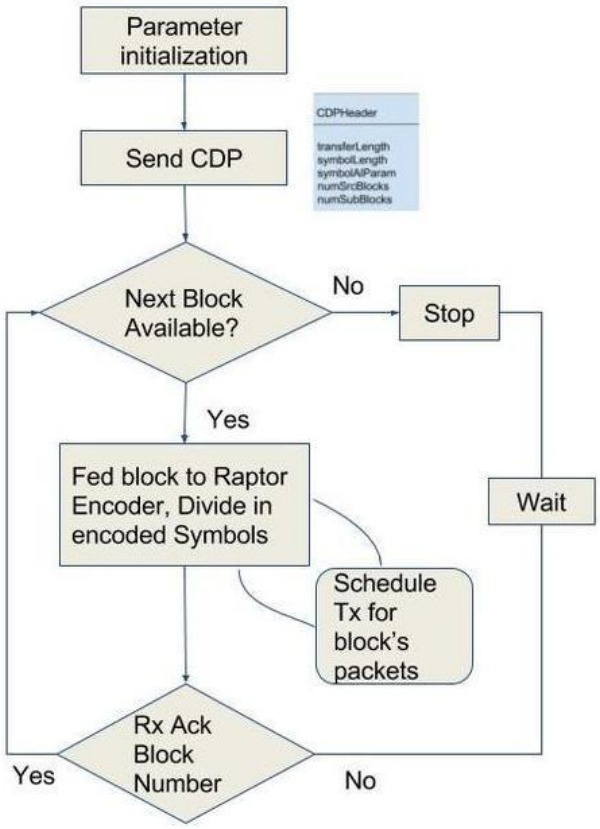

Figure 4.7: Transmitter side data flow

connected by the lower layers, the application layer at the sender interacts with the class Array_Data_Symbol to calculate the values of *K, S, H, L, Z* when given the input of total data length that is to be transported. Once these parameters are calculated, the sender initiates to send CDPHeader as the first constraint to start communication. After receiving the acknowledgment by the receiver of receiving CDP specifications, the data is chunked into source blocks each of length *K=1024* and fed to the R10_Encoder and is encoded. The encoding symbols of

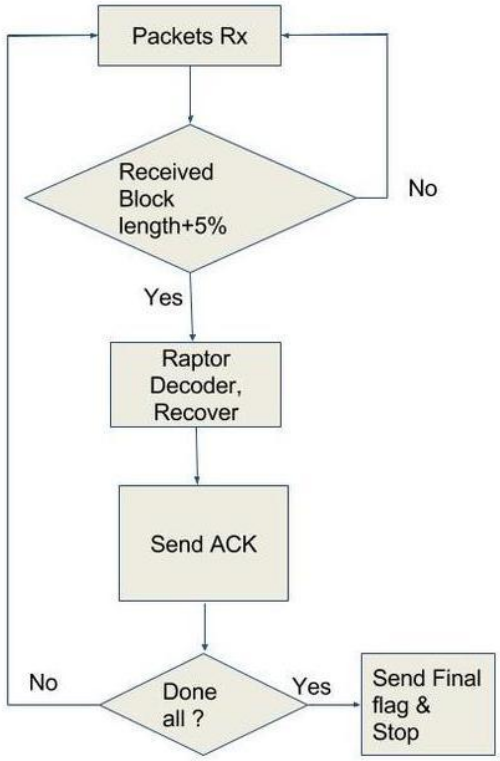

Figure 4.8: Receiver side data flow

data type Array_Data_Symbol are converted into packet API data type of ns-3 and are sent into the network by providing source block number and ESI for each packet using RPSendHeader. These packets are then fed to the UDP transport layer and further to the lower layers.

The transmitter after transmitting one chunk waits for an acknowledgment in order to process the next block. As soon as an acknowledgment is received, it then starts processing the new block. If an acknowledgment is not received it just sits in wait state until it receives an acknowledgment.

The connection is terminated as soon as the sender receives a final acknowledgment flag from the receiver after it has successfully recovered all the original source symbols.

The receiver side

Figure 4.8 depicts the representation of the receiver. Once the packet is received using from the sender using HandleRead (Rx) function, the encoded message is extracted from the packet. As soon as the required bytes (1.05*K) for a particular block is received, the decoding process is activated by invoking R10_Decoder class. If the decoding process is successful, it sends an acknowledgment using the RPRecvHeader. In order for this acknowledgment to reach successfully, we make sure that it is transmitted 3 times.

As long as receiver receives the data it either will be in the receiving state or sending acknowledgment state. A final acknowledgment to terminate the connection message is sent to the sender using the *flag* field in RPRecvHeader header.

Chapter 5

SIMULATION AND RESULTS

Raptor codes implemented in C++ and integrated with ns-3 were compared to the existing model of TCP protocol considering two different TCP congestion control variants NewReno–TCP (NR-TCP) and HighSpeed–TCP (HS-TCP) in packet network environment. As TCP provides connection-oriented and reliable delivery of data packets, we consider it as best standard for comparison. We consider only these high speed congestion control TCP variants, since other variants were not implemented in ns-3 version (ns-3.25) that was required to implement the protocol in and also the comparative study of Alrshah et al., [4] shows that other high seed variants like TCP CUBIC and TCP YeaH had almost equal performance with HS–TCP when considered in high bandwidth networks.

5.1 Simulation Configuration

To investigate the performance of ALFEC–UDP, we implemented the simple dumbbell topology with 3 sender-receiver pairs as shown in figure 5.1, with each source generating traffic with a Constant Bit rate (CBR) using Internet Protocol 4 (IPv4). In the designed protocol ALFEC–UDP, as there is no concept of congestion control, the host sends data at the maximal rates, which may result in a high extent of loss during the data transmission. However, as raptor codes are used, the extent of the loss is pointless even in a dynamically changing network since any subset of data is informative for decoding.

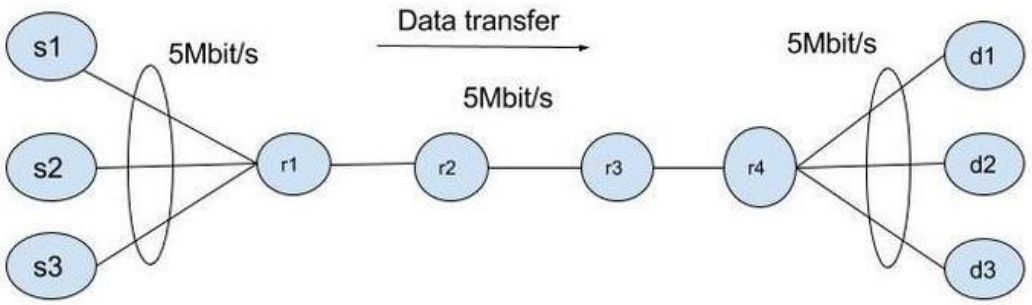

Figure 5.1: 3 Sender-Receiver pair Dumbbell Topology

5.2 Parameters

The size of payload both in ALFEC–UDP and TCP variants was segmented into 1000 bytes (B). We consider 3 senders transmitting data of length 1MB, 2MB, 3MB to its respective receiver each separately over bandwidth link with a capacity of 5Mbit/s. The rate error model of ns3, that is used to corrupt packets, was applied to all the protocols used for comparison on top of the wired point-to-point link in order to create a loss in the channel. The loss rate considered was in the unit of a packet and it varied from 0, 2, 5 and 10% of the data length respectively.

For each data length, the K value, that is a number of source symbols considered was $K = 1024$ and number of source blocks were 1, 2 and 3 for data of length 1MB, 2MB, and 3MB respectively. The time that each sender started to transmit the data was randomized between the range of 0 to 5. As ALFEC–UDP required the redundancy parameter (overhead) to encode the data, we also randomize the overhead between the range of 25% to 35% in terms of the total data length for each sender. For each receiver to start the process of decoding, it is required to receive any subset of original length plus 5% overhead for each block, since [27] suggests raptor codes can easily be implemented with an overhead of

0.05 in practice. Finally, all our results were averaged over to at least 50 simulation runs.

5.3 Metrics

Average completion time of data transport using ALFEC–UDP, TCP - New Reno and TCP - HighSpeed protocols; for all 3 sender receiver pairs was calculated at loss rates 0, 2, 5, 10% respectively. For ALFEC–UDP, we also record the average completion time to complete data transfer for each flow individually.

5.4 Results and Discussion

Figure 5.2 displays the result for the 3 sender-receiver pairs in dumbbell topology with each sender transferring 1MB data to the receiver over bandwidth capacity of 5Mbits/s. The X-axis represents the loss percentage applied during the data transport and the Y-axis represents the average delay taken to complete

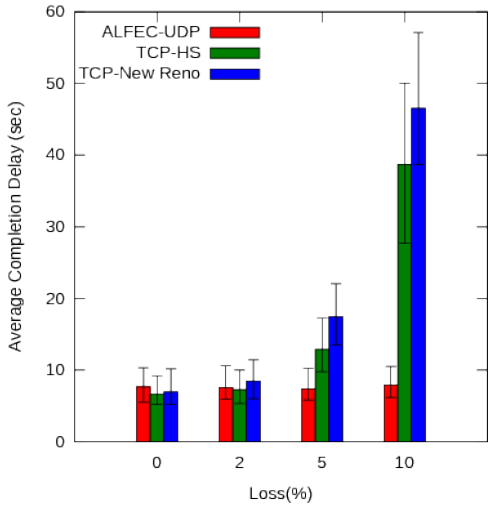

Figure 5.2: 3 sender-receiver pairs each sender transmitting 1MB data over 5Mbits/s capacity link, 0, 2, 5 and 10% packet loss rate vs Average delay

the simulation. The simulation stopped only after each user received 1MB data and recovered the original data successfully by decoding. During these simulations the packet loss rate was only varied. The bottleneck capacity and size of the data

remained constant. In all cases, we used average delay (time taken to complete all 3 users data transfer and recovery) as the performance metric. To evaluate the behavior of ALFEC–UDP, it was compared with two TCP versions, namely TCP New Reno and Highspeed. The figure 5.2 clearly indicates that ALFEC–UDP significantly outperforms both TCP versions in terms of average delay even in 10% lossy environment, showing ALFEC–UDP is less sensitive to the loss. Error bars

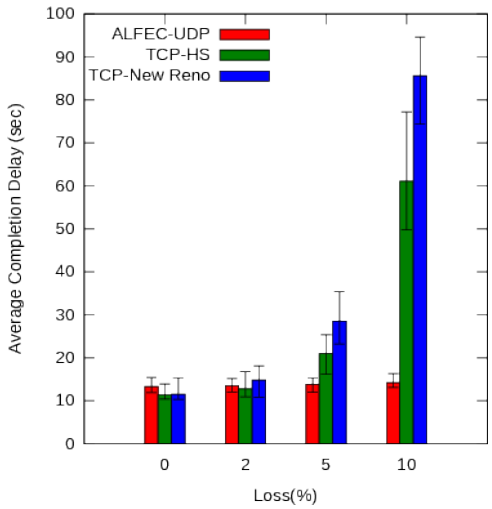

Figure 5.3: 3 sender-receiver pairs each sender transmitting 2MB data over 5Mbits/s capacity link, 0, 2, 5 and 10% packet loss rate vs Average delay

represents the minimum and maximum delay that the receivers experienced while completing the whole recovery of data. The minimum and maximum values are ranked based on the 25th and 75th percentile of all the 50 experiments performed.

The second and third experiment was set with the same setup, but the data length to transfer was increased from 1MB to 2MB and 3MB. Similar metrics as mentioned above were taken into consideration. From figures 5.3 and 5.4, it is clearly that ALFEC–UDP does not perform well then both TCP variants for the loss rate of 0 and 2% due to the fact that it required to receiving the overhead data packets in order to start the process of decoding. But as the loss percent rate

is increased, ALFEC–UDP outperforms the TCP variants again showing it to be less sensitive in a lossy environment. ALFEC–UDP on average is approximately 35% and 25% better than TCP variants in terms of average delay in receiving data of length 2MB and 3MB respectively.

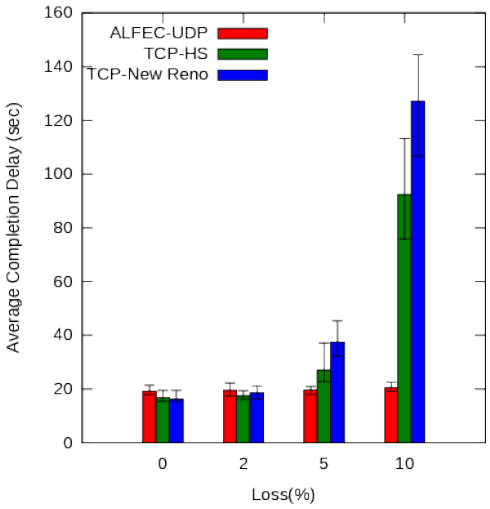

Figure 5.4: 3 sender-receiver pairs each sender transmitting 3MB data over 5Mbits/s capacity link, 0, 2, 5 and 10% packet loss rate vs Average delay

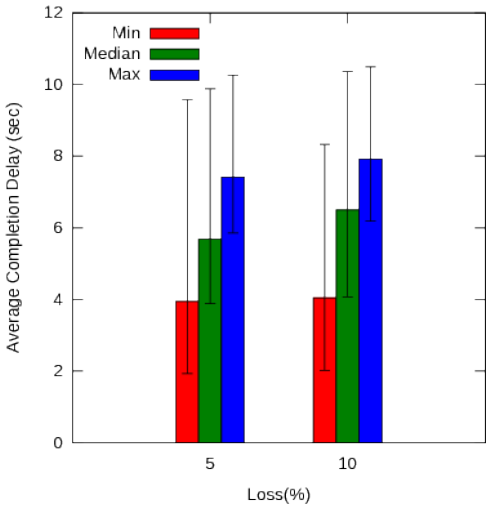

Figure 5.5: 3 sender-receiver pairs each transmitting 1MB data over 5Mbits/s, 5% and 10% packet loss rate vs each receiver's Average delay

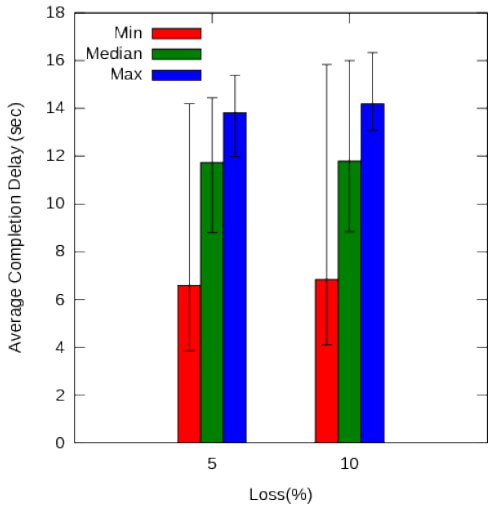

Figure 5.6: 3 sender-receiver pairs each transmitting 2MB data over 5Mbits/s, 5% and 10% packet loss rate vs each receiver's Average delay

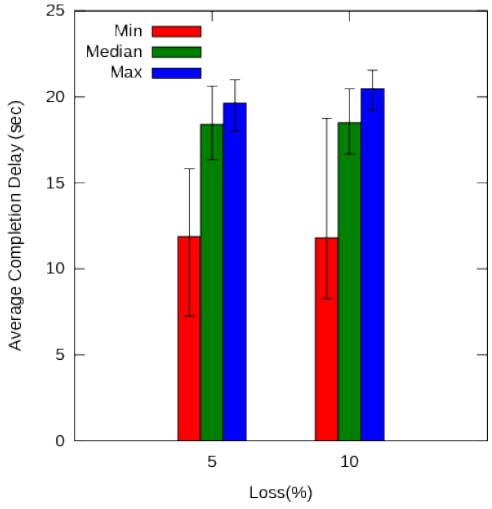

Figure 5.7: 3 sender-receiver pairs each transmitting 3MB data over 5Mbits/s, 5% and 10% packet loss rate vs each receiver's Average delay

As ALFEC–UDP protocol is developed and implemented using UDP, there is an absence of sequencing and connection oriented data delivery property. There is no guarantee on sequence data delivery and on all the users receiving the total data as there might also be a delay in different routers between the path from the

sender to the receiver in packet communication by ALFEC–UDP. Therefore, we record the average completion time of each user in the topology.

Figures 5.5, 5.6 and 5.7 shows the average delay each user experience on completely receiving the total data, processing it and decoding successfully. It is clear that no user is starved during the communication. As protocol does not require reordering, the processing time of each packet is also minimized. Any subset of the packet is useful for decoding since each packet has information about the original data.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this research, design, and implementation of proposed Application Layer Forward Error Correction based User Datagram Protocol (ALFEC–UDP) has been done. Raptor code, a fountain forward error correction paradigm was integrated into application layer of UDP to provide reliability for data transport in high bandwidth networks. Analysis of ALFEC–UDP was performed by comparing it with high bandwidth TCP variants. In order to practically evaluate the performance of designed ALFEC–UDP, the bounds of overhead packets required to send at a rateless scheme with large data sizes to achieve reliability using different loss rate models was performed. After investigating the scenarios, we compared its realistic results with TCP versions and found the performance of ALFEC–UDP to be better than that of current high bandwidth TCP variants based on average data transport completion time. Moreover, we were able to find out that ALFEC–UDP even worked better in lossy environments. The results demonstrated that performance of ALFEC–UDP was independent to the loss and delay in high bandwidth networks.

6.2 Future Work

The design and analysis of ALFEC–UDP shows the scope:

- To implement and evaluate ALFEC–UDP into more complex network topology with large data sizes
- To extend ALFEC–UDP into real Internet infrastructure and evaluate its performance
- To implement the protocol in multipath applications, where many transmitters transmits data to a receiver in order to exploit the network resources to its maximum extent.

REFERENCES

[1] Cisco. cisco visual network index, 2015. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html. Accessed: 2010-04-01.

[2] First international workshop on protocols for fast long-distance networks, 2004. https://datatag.web.cern.ch/datatag/pfldnet2003/. Accessed: 2010-04-01.

[3] Gigabit tcp, the internet protocol journal, volume-9. http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-16/gigabit-tcp.html. Accessed: 2010-04-01.

[4] Mohamed A Alrshah, Mohamed Othman, Borhanuddin Ali, and Zurina Mohd Hanapi. Comparative study of high-speed linux tcp variants over high-bdp networks. *Journal of Network and Computer Applications*, 43:66–75, 2014.

[5] Andrea Baiocchi, Angelo P Castellani, and Francesco Vacirca. Yeah-tcp: yet another highspeed tcp. In *Proc. PFLDnet*, volume 7, pages 37–42, 2007.

[6] Richard E Blahut. *Theory and practice of error control codes*, volume 126. Addison-Wesley Reading (Ma) etc., 1983.

[7] Thomas Bonald, Mathieu Feuillet, and Alexandre Proutiere. Is the"law of the jungle"sustainable for the internet? In *INFOCOM 2009, IEEE*, pages 28–36. IEEE, 2009.

[8] Anghel Botoş, Zsolt A Polgar, and Vasile Bota. Analysis of a transport protocol based on rateless erasure correcting codes. In *Intelligent Computer Communication and Processing (ICCP), 2010 IEEE International Conference on*, pages 465–471. IEEE, 2010.

[9] Zhihua Chen, J Fritz Barnes, and Bobby Bodenheimer. Hybrid and forward error correction transmission techniques for unreliable transport of 3d geometry. *Multimedia Systems*, 10(3):230–244, 2005.

[10] Patrick Guy Farrell et al. *Essentials of error-control coding*. John Wiley & Sons, 2006.

[11] Sally Floyd. Highspeed tcp for large congestion windows. 2003.

[12] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[13] Helmut Griesser and Joerg-Peter Elbers. Forward error correction coding, January 27 2009. US Patent 7,484,165.

[14] Hojin Ha and Eun-Seok Ryu. Block recovery rate-based unequal error protection for three-screen tv. *Applied Sciences*, 7(2):186, 2017.

[15] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[16] Mathias Johanson. Adaptive forward error correction for real-time internet video. In *Proceedings of the 13th Packet Video Workshop, Nantes, France*, 2003.

[17] Ashish Khisti. Tornado codes and luby transform codes, 2003.

[18] Sung Won Kim, Sun Young Kim, Saejoon Kim, and Jun Heo. Performance analysis of forward error correcting codes in iptv. *IEEE Transactions on Consumer Electronics*, 54(2), 2008.

[19] Douglas Leith and Robert Shorten. H-tcp: Tcp for high-speed and long-distance networks. In *Proceedings of PFLDnet*, volume 2004, 2004.

[20] Yee-Ting Li, Douglas Leith, and Robert N Shorten. Experimental evaluation of tcp protocols for high-speed networks. *IEEE/ACM Transactions on networking*, 15(5):1109–1122, 2007.

[21] Shu Lin and Daniel J Costello. *Error control coding*. Pearson Education India, 2004.

[22] Luis López, Antonio Fernández, and Vicent Cholvi. A game theoretic analysis of protocols based on fountain codes. In *10th IEEE Symposium on Computers and Communications (ISCC'05)*, pages 625–630. IEEE, 2005.

[23] M Luby, A Shokrollahi, M Watson, and T Stockhammer. Rfc 5053: Raptor forward error correction scheme: Scheme for object delivery. *Tech. rep.). IETF, Tech. Rep.*, 2007.

[24] Michael Luby, Tiago Gasiba, Thomas Stockhammer, and Mark Watson. Reliable multimedia download delivery in cellular broadcast networks. *IEEE Transactions on Broadcasting*, 53(1):235–246, 2007.

[25] Michael Luby, Mark Watson, Tiago Gasiba, Thomas Stockhammer, and Wen Xu. Raptor codes for reliable download delivery in wireless broadcast systems. In *Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE*, volume 1, pages 192–197. IEEE, 2006.

[26] Mike Luby, Mark Watson, Tiago Gasiba, and Thomas Stockhammer. High-quality video distribution using power line communication and aplication layer forward error correction. In *Power Line Communications and Its Applications, 2007. ISPLC'07. IEEE International Symposium on*, pages 431–436. IEEE, 2007.

[27] David JC MacKay. Fountain codes. *IEE Proceedings-Communications*, 152(6):1062–1068, 2005.

[28] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*. Elsevier, 1977.

[29] Lorenz Minder, Amin Shokrollahi, Mark Watson, Michael Luby, and Thomas Stockhammer. RaptorQ Forward Error Correction Scheme for Object Delivery. RFC 6330, August 2011.

[30] Todor Mladenov, Saeid Nooshabadi, and Keseon Kim. Implementation and evaluation of raptor codes on embedded systems. *IEEE Transactions on Computers*, 60(12):1678–1691, 2011.

[31] Sándor Molnár, Zoltán Móczár, András Temesváry, Balázs Sonkoly, Szilárd Solymos, and Tamás Csicsics. Data transfer paradigms for future networks: Fountain coding or congestion control? In *IFIP Networking Conference, 2013*, pages 1–9. IEEE, 2013.

[32] William Wesley Peterson and Edward J Weldon. *Error-correcting codes*. MIT press, 1972.

[33] Amin Shokrollahi. Raptor codes. *IEEE transactions on information theory*, 52(6):2551–2567, 2006.

[34] Amin Shokrollahi. Theory and applications of raptor codes. In *Mathknow*, pages 59–89. Springer, 2009.

[35] Thomas Stockhammer, Amin Shokrollahi, Mark Watson, Michael Luby, and Tiago Gasiba. Application layer forward error correction for mobile multimedia broadcasting. *Handbook of mobile broadcasting: DVB-H, DMB, ISDB-T and media flo*, pages 239–280, 2008.

[36] Kenneth Reginald Sturley. *Radio receiver design*. Chapman & Hall, 1953.

[37] Jay Kumar Sundararajan, Devavrat Shah, Muriel Médard, Szymon Jakubczak, Michael Mitzenmacher, and Joao Barros. Network coding meets tcp: Theory and implementation. *Proceedings of the IEEE*, 99(3):490–512, 2011.

[38] Kun Tan, Jingmin Song, Qian Zhang, and Murad Sridharan. A compound tcp approach for high-speed and long distance networks. In *Proceedings-IEEE INFOCOM*, 2006.

[39] Yeqing Wu, Fei Hu, Qingquan Sun, Ke Bao, and Mengcheng Guo. A fast raptor codes decoding strategy for real-time communication systems. *Network and Communication Technologies*, 2(2):29, 2013.

[40] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *INFOCOM 2004. Twenty-*

third AnnualJoint Conference of the IEEE Computer and Communications Societies, volume 4, pages 2514–2524. IEEE, 2004.