MODIFIED QUIC PROTOCOL WITH CONGESTION CONTROL FOR IMPROVED NETWORK PERFORMANCE

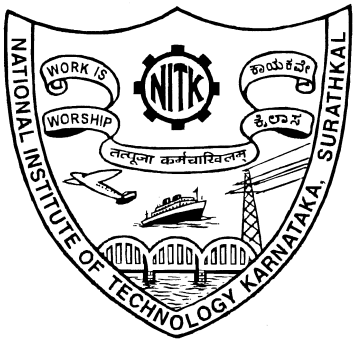
Thesis

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

KHARAT PRASHANT KRISHNARAO



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL, MANGALORE - 575025

OCTOBER, 2019

DECLARATION

I hereby ***declare*** that the research Thesis entitled **MODIFIED QUIC PROTOCOL WITH CONGESTION CONTROL FOR IMPROVED NETWORK PERFORMANCE** which is being submitted to the ***National Institute of Technology Karnataka, Surathkal*** in partial fulfillment of the requirement for the award of the Degree of ***Doctor of Philosophy*** in **Department of Electronics and Communication Engineering** is a ***bonafide report of the research work carried out by me***. The material contained in this research Thesis has not been submitted to any University or Institution for the award of any degree.

Kharat Prashant Krishnarao
Reg. No.: 165007 EC16F04
Department of E & C Engg.

Place: NITK, Surathkal
Date: October, 2019

CERTIFICATE

This is to certify that the Research Thesis entitled **MODIFIED QUIC PROTOCOL WITH CONGESTION CONTROL FOR IMPROVED NETWORK PERFORMANCE** submitted by **KHARAT PRASHANT KRISHNARAO** (Reg. No.: 165007 EC16F04) as the record of the research work carried out by him, is accepted as the *Research Thesis submission* in partial fulfillment of the requirements for the award of degree of **Doctor of Philosophy**.

Dr. Muralidhar Kulkarni
Research Guide

Chairman-DRPC

Acknowledgements

I would like to take this opportunity to express my sincere thanks to people who helped me directly or indirectly to made this work possible.

First and foremost, I would like to express my gratitude and heartiest thanks to my research guide and advisor Dr. Muralidhar Kulkarni, for his continuous support, patient guidance, encouragement, precious advice, motivation and moral support throughout the research work. It was a great opportunity for me to work with him and no words to express regards towards him.

Besides my advisor, I would like to thank the rest of my RPAC members, Dr. N. S. V. Shet and Dr. Annappa for their critical comments, insights and encouragement throughout my research work. I express profound gratitude to Dr. T. Laxminidhi, Professor and Head of the Electronics and Communication Engineering department and former Heads, Dr. U. Shripathi Acharya and Dr. M. S. Bhat and faculty members and supporting staff having provided me with all the facilities to carry out this research work.

I take this opportunity express my deep gratitude towards my parent institute Walchand College of Engineering, Sangli, Maharastra, for giving me an opportunity to undergo this Ph.D. with wholehearted support. I sincerely thank to honorable Director Dr. G. V. Parishwad, Deputy Director Dr. P. G. Sonavane and Head of the Department Dr. S. P. Sonavane for moral support throughout the deputation period. Special thanks to Government of Maharastra, All India Council for Technical Education (AICTE), Dr. Madhukar Waware, Deputy Director and Dr. Amit Salunkhe, Assistant Director, AICTE for help in getting financial assistantship.

I am extremely thankful to a host of my friends and fellow research scholars of NITK, Surathkal for their sincere co-operation and providing a friendly atmosphere during the course of my work.

With heartful gratitude the most precious jewels of my life, wife Jayashree and daughters Aditi and Anvi for sacrificing for the last three years. I am very much thankful to my in laws Tai and Bapu for their continuous support and encouragement. I thank Aniket, my parents, brothers and brother in law for their cooperation throughout this work.

Above all, I thank, God Vitthal and Goddess Tuljabhavani, who enabled me with philosophy, perception and motivation to present this work, after so many, hurdles and obstacles.

Place: NITK, Surathkal Kharat Prashant K.
Date: October, 2019

Abstract

In a computer communication network, transport layer is responsible for reliable data delivery with guaranteed Quality of Service (QoS). Congestion control mechanism is one of the prime features of transport layer and predominantly responsible for maintaining the performance of network. The main contributions of this thesis are to modify the existing handshaking mechanism of Quick UDP Internet Connections (QUIC) protocol, called modified QUIC (ModQUIC) to reduce control overhead and Congestion Window (*cwnd*) size update delay. The suggested modification fine tunes the window update mechanism with Acknowledgment (ACK), that results in smooth variation in *cwnd* growth. Further, it regulates the network traffic which in turn helps to control congestion.

In the first phase, we have suggested modification of the existing QUIC protocol called the ModQUIC protocol and its performance has been evaluated with Chromium server-client model testbed. The results and analysis are presented in comparison to QUIC and TCP in terms of performance measures like throughput and delay. Here, the performance has been tested for limited and sufficient link bandwidth in presence of loss, whereas the validation of results have been carried out with the help of linear regression model. In the result analysis, it was evident that ModQUIC achieves a throughput improvement of 35.66% and 51.93% over QUIC and TCP respectively, whereas delay is also reduced to 3% and 5% in comparison to QUIC and TCP. In the second phase, ModQUIC performance is verified in emulated environment using the Mininet for transport layer and browser network. The experimental results of ModQUIC are compared with QUIC, TCP and TCP/HTTP2 based on throughput, delay and fairness. It has been observed that ModQUIC outperforms throughput with an increased speed by minimizing transmission delay and improvised fairness. In addition, specifically for lossy link and low bandwidth, ModQUIC performance is better compared to QUIC and TCP.

One of the QUIC features is inbuilt congestion control mechanism, which is responsible for maintaining streaming data. If there is no congestion, regular window size update is carried out as a step by step approach. In next phase of research work, ModQUIC protocol is investigated with CUBIC and Bot-

tleneck Bandwidth Round-trip-propagation (BBR) congestion control mechanisms and suggested use of a decreased value for the factor $\beta = \beta_{TCP}/n$ for n flows, which are competing to acquire bottleneck resources, where β_{TCP} is the decrease factor used in TCP. The Chromium server-client testbed experiment results show that the ModQUIC with BBR giver better performance in terms of throughput, delay and datarate. The result analysis actually gives an improvement of 6.8%, 19.06% and 27.9% for the ModQUIC/BBR as compared to ModQUIC/CUBIC, QUIC/BBR and QUIC/CUBIC respectively in terms of throughput. Furthermore, the delay is reduced by 8.02%, 6.56% and 14.38% over ModQUIC/CUBIC, QUIC/BBR and QUIC/CUBIC respectively.

The QUIC protocol versions are updating faster and new versions of the same are available for users. In the final contribution of the research work, the performance of QUIC protocol has been tested with respect to congestion control using India's rapidly growing Internet Service Provider (ISP); Reliance Jio 4G (JioFi) network. This experimental study investigated ModQUIC performance for congestion control mechanisms CUBIC and BBR in JioFi. The experiment is conducted using a testbed, developed with JioFi and Raspberry Pi-3 wireless router along with network emulator; Netem. Furthermore, the QUIC performance is verified with respect to Throughput and Retransmission Ratio (RTR) in which it is observed that overall ModQUIC/BBR performance is better than ModQUIC/CUBIC in the current Internet. We observed that Reliance Jio is an economical solution for the highly populated country like India, but not a contemporary solution to fulfill India's newly launched digitization project.

Keywords: Handshaking mechanism; QUIC protocol; ModQUIC protocol; Congestion control; Network performance; Internet protocol technology.

Contents

List of Figures

List of Tables

List of Symbols

Symbol	Meaning
λ	mean rate
λ_n	arrival rate
σ	mean size
μ_n	departure rate
α	increase factor
β	decrease factor
β_{TCP}	TCP decrease factor
β_{QUIC}	QUIC decrease factor
N	number of segments
$cwnd_{initial}$	Initial *cwnd* size
T	Time
ms	mili second
$FlightSize$	Flight size
C	network capacity
GHz	Giga hertz
@	at the rate
min	minimum value
max	maximum value
W	*cwnd* size
W_{max}	maximum *cwnd* size
W_{min}	minimum *cwnd* size
D	maximum size of data packet
P	steady state probability
R_e	expected rate
W_u	window update size
ST	step size
F	Fairness index
f_i	resource allocation for i^{th} flow
n	number of flows
Mbps	Mega bits per second
Gbps	Giga bits per second
MB	Mega bytes
KB	Kilo bytes

Acronyms and Abbreviations

Abbreviation	Expansion
5G	5^{th} Generation
ABE	Available Bandwidth Estimation
ACK	Acknowledgment
AI	Additive Increase
AIMD	Additive Increase Multiplicative Decrease
ARPA	Advanced Research Project Agency
BBR	Bottleneck Bandwidth Round-trip-propagation
BDP	Bandwidth Delay Product
BIC	Binary Increase Congestion-control
BtlBw	Bottleneck Bandwidth
CA	Congestion Avoidance
CDMA	Code Division Multiple Access
CPU	Central Processing Unit
cwnd	Congestion Window
DCCP	Datagram Congestion Control Protocol
DCN	Data Center Network
DEC	Digital Equipment Corporation
DHCP	Dynamic Host Configuration Protocol
ECN	Explicit Congestion Notification
E-mail	Electronic mail
FAN	Flow Aware Networking
GoS	Guarantee of Service
HTTP	Hyper Text Transfer Protocol
HOL	Head Of Line
HS-TCP	High Speed TCP
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
LTS	Long Term Support
MAC	Media Access Control
MD	Multiplicative Decrease
MIMD	Multiplicative Increase Multiplicative Decrease
MIMO	Multiple Input Multiple Output

Abbreviation	Expansion
ModQUIC	Modified QUIC
MLQG	Modified Linear Quadratic guass
MPQUIC	Multipath QUIC
MPTCP	Multipath TCP
MTU	Maximum Transfer Unit
NACK	Not An ACK
NAT	Network Address Translation
NCPLD	Non-Congestion Packet Loss Detection
NIC	Network Interface Card
OS	Operating System
PLT	Page Load Time
QoS	Quality of Service
QUIC	Quick UDP Internet Connections
RAM	Random Access Memory
RFC	Request For Comment
RIP	Routing Information Protocol
RMSS	Receiver Maximum Segment Size
RTprop	Round Trip propagation
RTR	Retransmission Time
RTT	Round Trip Time
rwnd	Receiver Window
SACK	Slective ACK
SDN	Software Defined Networks
SMSS	Sender Maximum Segment Size
SNA	System Network Architecture
SNMP	Simple Network Management Protocol
SS	Slow Start
$ssthresh$	Slow Start Threshold
TCP	Transmission Control Protocol
TFO	TCP Fast Open
TLS	Transport Layer Security
UDP	User Datagram Protocol
WAN	Wide Area Network
WLAN	Wireless Local Area Network
WTCP	Wireless TCP
www	World Wide Web

Chapter 1

Introduction

The world wide success of Internet leads to a rapid adaptation of Internet protocol technology, which facilitates creating various networks, such as private corporate network, military communication enabling network, personal network and the rapidly growing cellular network. In today's world, more than trillions of devices are connected to the network via IP enabled services such as Internet of Things (IoT) framework. The rapid development in the field of IP enabled services presents performance challenges to network service providers on Guarantee of Service (GoS) and Quality of Service (QoS). The central theme of this thesis is to make aware of Internet users about current development in the field of transport layer protocol technology and extend this contribution towards reducing network latency.

Transmission Control Protocol (TCP) is the predominant transport layer protocol used by IP technology to support Internet services. This chapter begins with introduction of transport layer with respect to development and deployment of current Internet transport layer solutions followed by congestion control mechanisms and Quick UDP Internet Connections (QUIC) protocol design, development and functionality rationales. The chapter ends with the metrics used for performance analysis of transport layer solutions, motivation and organization of the thesis.

1.1 Transport Layer

Transport layer is the fourth layer of TCP/IP model, which provides end-to-end connectivity; a point-to-point connection rather than hop-to-hop, between the source and destination hosts, to deliver messages. The structure of transport layer is shown in Figure 1.1. The transport layer encapsulates the application data into data units called segments. The prevailing

protocols used by transport layer to enhance it's capabilities are Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Datagram Congestion Control Protocol (DCCP)

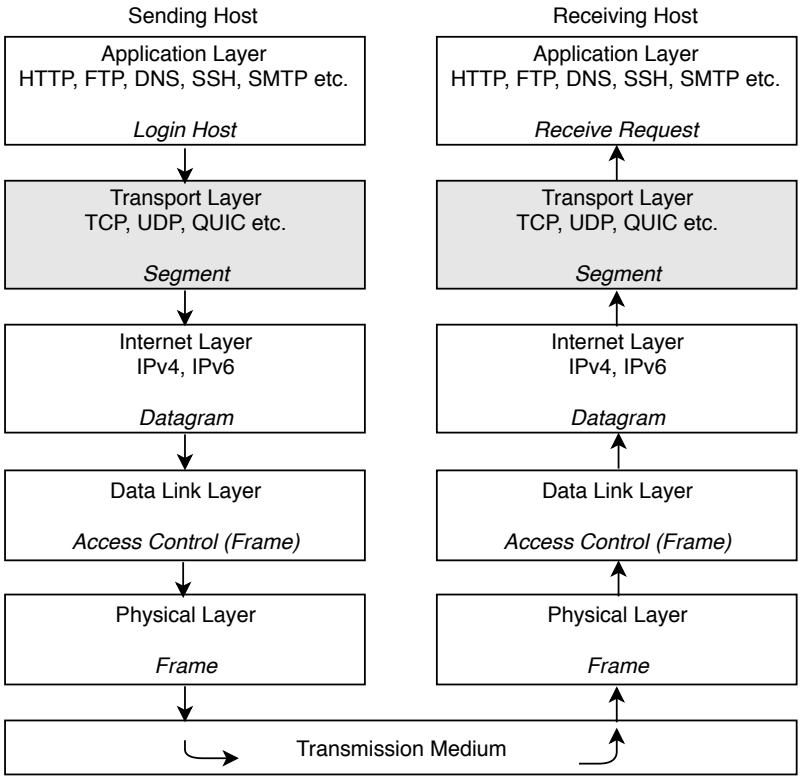


Figure 1.1: Data path through TCP/IP stack from sending to receiving host

1.1.1 Transport Layer Functionalities

- **Process to process delivery:** There are multiple processes running on host and to deliver segment to the addressed host similar to data link layer, needs the Media Access Control (MAC) address[1] of source and destination host to correctly deliver a frame. However, network layer uses Internet Protocol (IP) address to route packet correctly. In a similar way, transport layer requires a port number to correctly deliver the segments of data to the correct process. A port number is a 16 bit integer used to identify processes based on application. This number is set automatically by Operating System (OS) or set by user or default to popular applications.

[1] 48 bits address contained inside the Network Interface Card (NIC) of every host machine

- **End-to-end connection between hosts:** Transport layer creates end to end connection between client and server by using TCP or UDP. TCP is a reliable, secure, connection-orientated protocol, which uses a handshake to establish a robust connection between two end users. UDP is a stateless and unreliable protocol which ensures best-effort delivery. It is suitable for the applications which have concern about delay require to send bulk of data such as video conferencing.

- **Multiplexing and Demultiplexing:** In transport layer, multiplexing allows simultaneous use of different applications over a network which are running on a host. Multiplexing allows to send packets from diverse applications concurrently over a network. As shown in Figure 1.1, transport layer accepts packets from application layer identified by their port numbers and hands over to network layer by adding proper header. Similarly demultiplexing is used to deliver received segment to proper process running on the receiver's machine.

- **Congestion control:** When more number of packets are sent than the capacity of the network, then router buffer overflow event occurs and packets get dropped. This situation is called congestion, and it further increases due to packet retransmission. To control such situations, transport layer provides control over the congestion through various congestion control algorithms [Afanasyev *et al*. (2010)]. The standard techniques used for congestion control are Additive Increase Multiplicative Decrease (AIMD), leaky bucket etc.

- **Data integrity and Error correction:** Transport layer uses error detection technique to verify correctness of the message received from application layer by computing checksums. In this Acknowledgment (ACK) and Not An ACK (NACK) control messages are used to inform sender about data reception status. Transport layer is also responsible for the authenticity and integrity of data.

- **Flow control:** Transport layer provides a flow control mechanism between the adjacent layers of the TCP/IP model to avoid packet loss due to a fast sender and a slow receiver. A well known flow control mechanism used by TCP is sliding window, in which receiver sends window back to the sender to inform size of the data it can receive.

TCP is the predominant transport protocol used by IP technology to support popular Internet services. The first reference of TCP was in 1973 by Cerf under title of 'A Partial

Specification of an International Transmission Protocol' [Cerf (1973)]. This is the era when networking, router and layering concepts were not in existence. Most of the tasks were handled by network control protocol. In 1974, Cerf and Kahn discussed different design specifications of TCP in 'A Protocol for Packet Network Interconnection' [Cerf and Kahn (1974)]. The first official specification of TCP was declared in 1976, which was specified by Cerf *et al.*, whereas second version written in 1977 by Cerf. In January 1978, Cerf and Postel splitted TCP into two parts: TCP and IP, in which TCP is responsible for packetization, error control, retransmission and reassemble while IP is responsible for packet routing [Cerf and Postel (1978)].

In February 1980, the department of defense, U.S. opted TCP/IP as an authorized protocol and declared every site connected to Advanced Research Project Agency (ARPA) net should switch to TCP/IP by 1983. In and around 1980s, there were many other networking protocol stacks competed with TCP/IP, such as Digital Equipment Corporation (DEC) net, System Network Architecture (SNA), AppleTalk etc. Most of these protocols are vendor specific or proprietary, whereas TCP is open source deployment.

1.1.2 TCP Protocol Applications

IP is the baseline protocol used for the Internet services, in which transport and application protocols are used to build a service on top of IP. TCP and UDP are the two transport layer protocols, of which TCP provides a reliable service to loss-sensitive applications, whereas UDP supports a more lightweight transport without packet retransmission to assist delay-sensitive applications. Most of the popular Internet applications are using TCP.

- **Electronic mail (E-mail):** This application allows users to send/receive mails electronically. Many employers run their own TCP/IP network and provide e-mail accounts to their employees.

- **World Wide Web (www):** This protocol allows us to download images or other objects from another web site using TCP service.

- **File transfer:** TCP/IP networks include a file transfer application that allows users to send and receive arbitrarily large files. These files may contain text, program, image, audio and video files.

- **Remote login:** Using the remote login application, user can open an interactive session with a remote machine through the Internet.

In addition to TCP, TCP/IP stack provides another transport protocol, UDP. Unlike TCP, UDP is very simple and provides bare minimum services to application. UDP is connectionless (connection establishment and termination procedure is not implemented), unreliable (acknowledgment, retransmission, sequence number for datagram and flow control mechanisms are not present) and datagram based protocol. The application is supposed to supply segmented data to UDP for transportation as an independent datagram.

1.1.3 UDP Protocol Applications

- **Multicasting:** An application that sends same piece of data to many receivers e.g. video conferencing, web casting.
- **Network management:** Use of short request/response messages using UDP, which removes connection establishment and termination. e.g. Simple Network Management Protocol (SNMP).
- **Routing table update:** The routing applications, reply on query-response type of communications. e.g. Routing Information Protocol (RIP).
- **Real-time multimedia:** The real-time applications can tolerate occasional packet losses but cannot tolerate long delays caused by retransmissions of lost packets. By the time the retransmitted packet would arrive at the destination, it would become useless and be discarded. For such scenario UDP is more suitable compared to TCP.

Table 1.1: Difference between TCP and UDP

TCP	UDP
Connection-oriented service	Connectionless service
Stream based protocol	Datagram based protocol
Reliable service	Unreliable service
Flow control mechanism present	Flow control mechanism absent
Congestion control mechanism present	Congestion control mechanism absent

IP used by both TCP and UDP to transfer segments and datagrams via payload of IP datagram. The IP protocol provides a connectionless, unreliable service through networking. IP encapsulates the higher layer protocol units within it's datagram payload, creates the IP header and forwards the complete IP datagram to the next hop router towards the

destination. Each intermediate router processes the IP datagram header and forwards it to the next router along the path until it reaches the destination. The connectionless service provided by IP simplifies the router design as it does not need to maintain connection information which scales well for large number of hosts. Routers also get flexibility of choosing an appropriate path for each datagram based on link bandwidth available. Table 1.2 shows list of protocols with features which commonly placed at transport layer.

1.2 Congestion Control

Congestion control is an artificial feedback mechanism to control buffer overflow and packet dropping events in the networks. Congestion control is one of the key feature of TCP in which four states, namely slow start, congestion avoidance, fast retransmit, and fast recovery are convolved. These states are four different algorithms, which are devised in Jacobson (1988) and Jacobson (1990) and standardized in Braden (1989).

The growth of data network depends on QoS provided to the users, in which congestion control plays a very important role. When nodes in the network carry more data than link can handle, the QoS gets deteriorated due to congestion in the network, which adds delay and block new connections. To compensate packet loss due to congestion, normally transport layer protocols use aggressive retransmission of packets. If offered load in an uncontrolled distributed system exceeds system capacity in flow control mechanism (fluid dynamics), the effective load goes to zero (collapse) as increase in load. In this case, system experiences very low throughput almost equal to zero, which is termed as 'congestion collapse'.

1.2.1 Definitions

For better understanding and convenience of the reader, we define various terminologies that will be used throughout the document.

- **Segment:** An accepted stream of data from application layer that TCP divides among different chunks and adds a specific header is called segment.
- **Datagram:** Datagram is a basic transfer unit associated with a packet-switched (connectionless service) network. This is an independent, self contained message whose delivery, arrival time and order of arrival is not guaranteed.

Figure 1.2: Transport layer protocol comparison

Feature	UDP	UDP Lite	TCP	MTCP	SCTP	DCCP	RUDP	QUIC
Packet header size	8 Bytes	8 Bytes	20-60 Bytes	50-90 Bytes	12 Bytes	12/16 Bytes	14+ Bytes	2-19 Bytes
Data packet overhead	8 Bytes	8 Bytes	20 Bytes	20 Bytes	48+ Bytes	12/16 Bytes	14 Bytes	8 Bytes
Transport layer packet entity	Datagram	Datagram	Segment	Segment	Datagram	Datagram	Datagram	Datagram
Connection oriented	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Reliable transport	No	No	Yes	Yes	Yes	No	Yes	Yes
Delivery	Unordered	Unordered	Ordered	Ordered	Both	Unordered	Unordered	Unordered
Data checksum (size)	Optional	Yes	Yes	Yes	Yes	Yes	Optional	Optimal
Path MTU	No	No	Yes	Yes	Yes	Yes	No	No
Congestion control	No	No	Yes	Yes	Yes	Yes	No	Yes
Flow control	No	No	Yes	Yes	Yes	No	Yes	Yes
ECN	No	No	Yes	Yes	Yes	Yes	No	Yes
Multiple streams	No	No	No	No	Yes	No	No	Yes
Bundling/Nagle	No	No	Yes	Yes	Yes	No	Yes	Yes

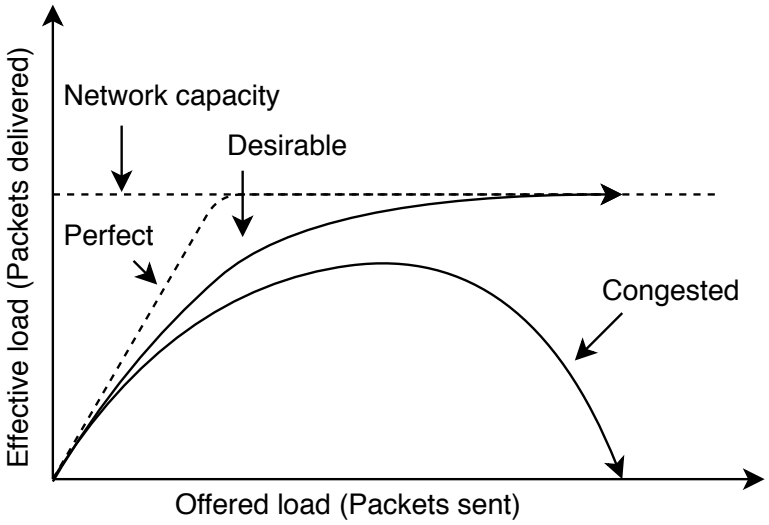


Figure 1.3: Congestion collapse

- **Maximum Transmission Unit (MTU):** MTU is the largest data size that can be transfered within single network layer transaction. Larger the MTU, lesser is the overhead, whereas smaller MTU values can reduce network delay.

- **Sender Maximum Segment Size (*SMSS*):** This is the maximum segment size without TCP/IP header that sender can send. This is single non-fragmented piece of data and its value is based on the MTU.

- **Receiver Maximum Segment Size (*RMSS*):** The largest segment size without TCP/IP header that receiver can handle or accept is called $RMSS$. The value for $SMSS$ is need to specify in the MSS options during connection setup. If the MSS option is not used, $SMSS$ is set to 536 bytes.

- **Receiver Window (*rwnd*):** This is advertised by receiver, based on recently accepted data.

- **Congestion Window (*cwnd*):** This TCP state variable represents the amount of data a TCP can send. The size of *cwnd* increases exponentially or linearly for every successful packet delivery. A *cwnd* is the sender side limit based on the amount of data the sender can have in flight before receiving an acknowledgment. The *cwnd* is a private variable, not advertised or exchanged between sender and receiver. At any point of time, a TCP can not exceed data sending limit greater than sum of highest

acknowledged sequence number and minimum of *cwnd* or *rwnd*. Maximum amount of data in flight between client and server is minimum of *rwnd* and *cwnd*.

- **Slow Start Threshold (*ssthresh*):** This variable is the boundary between slow start and congestion avoidance phase of TCP. If *cwnd* is less than or equal to $ssthresh$, then TCP is in slow start phase (exponential growth), else congestion avoidance phase (linear growth).

- **Duplicate Acknowledgement:** An acknowledgment is considered to be a 'duplicate' when (i) receiver of the ACK has outstanding data, (ii) empty acknowledgment, (iii) SYN and FIN bits are zero. SYN, synchronizes sequence numbers to initiate a TCP connection, whereas FIN indicates the end of data transmission to finish TCP connection (iv) the acknowledgment number is equal to the greatest acknowledgment received on the given connection and (v) current advertised window is equal to the previous advertised window.

1.2.2 Congestion Control Algorithms

This section describes congestion control algorithms: slow start and congestion avoidance, fast retransmit and fast recovery.

Slow Start (SS) and Congestion Avoidance (CA): The slow start and congestion avoidance algorithms are control parameters, which shows the state of the TCP sender. These algorithms are implemented by using two variables *cwnd* and *rwnd*, which decide data probing limit into network buffer. The minimum of *cwnd* and *rwnd* regulates data transmission rate. One more state variable, $ssthresh$ is used to decide whether the slow start or congestion avoidance algorithm is used to control data transmission. Every TCP connection must go through the slow start phase. At the beginning, sender cannot use the full capacity of the link immediately. Instead, it starts with a small *cwnd* size and double it for every Round Trip Time (RTT), that is 'exponential growth' given in equation (1.1). The time (T) required to reach *cwnd* size equal to N segments is given by

$$T = RTT * \left[Log_2 \left(\frac{N}{cwnd_{initial}} \right) \right] \tag{1.1}$$

Initially, data transmission starts with unknown network conditions and increases slowly to probe the network learn about accessible link capacity to avoid congestion. At the start, *cwnd* is set to one segment and gets doubled after every successful acknowledgment. This

shows exponential growth in data transmission as seen in Figure 1.4. Unless this exponential growth is checked at some level, it may quickly lead to congestion. To avoid congestion well before, a threshold, *ssthresh*, a dynamic variable is set. Once data transmission rate reaches to *ssthresh*, *cwnd* size grows linearly by incrementing *cwnd* with $1/cwnd$ per successful acknowledgment.

The initial value of *cwnd* size calculated using Algorithm-1 as an upper bound.

Algorithm 1 : Initial Congestion Window Size Algorithm

1: **Input:** $SMSS$, $cwnd_{initial}$,
2: **if** $SMSS > 2190\ bytes$:
3: $cwnd_{initial} = 2 * SMSS\ bytes\ and\ not\ more\ than\ 2\ segments$
4: **if** $SMSS > 1095\ bytes\ and\ SMSS <= 2190\ bytes$:
5: $cwnd_{initial} = 3 * SMSS\ bytes\ and\ not\ more\ than\ 3\ segments$
6: **if** $SMSS <= 1095\ bytes$:
7: $cwnd_{initial} = 4 * SMSS\ bytes\ and\ not\ more\ than\ 4\ segments$

The *cwnd* size should not get increased with SYN/ACK or acknowledgment of the SYN/ACK. To regulate traffic, at $cwnd_{initial}$ size more than one segment and MSS used is large, *cwnd* is reduced for smaller segments. This reduction is done by the factor of $[(old\ segment\ size)/(new\ segment\ size)]$.

The value for $ssthresh_{initial}$ is usually set equal to the size of largest possible advertised window and reduces in response to congestion. As shown in Figure 1.4, the slow start algorithm is used when *cwnd* is less than *ssthresh*, while the congestion avoidance algorithm is used when *cwnd* is greater than *ssthresh*. When *cwnd* is equal to *ssthresh*, the sender may use either slow start or congestion avoidance. During slow start, TCP increments size of *cwnd* by maximum equal to $SMSS$ bytes for every acknowledgment. The slow start ends when *cwnd* size exceeds *ssthresh* or when congestion is detected. While in simple practice, TCP implementations have increased *cwnd* size by precisely $SMSS$ bytes upon receipt of an ACK. The recommended increase in *cwnd* is,
$cwnd+ = min(M, SMSS)$
where, M is the number of unacknowledged bytes acknowledged in the incoming ACK.

During congestion avoidance phase, *cwnd* is incremented by roughly one full-sized segment per RTT and congestion avoidance continues until congestion is detected. The basic guidelines for incrementing *cwnd* during congestion avoidance is $cwnd+ = min(N, SMSS)$, but not more than $SMSS$ bytes. Another way to update *cwnd* size in congestion avoidance

phase is given by

$$cwnd+ = SMSS * \left(\frac{SMSS}{cwnd}\right) \tag{1.2}$$

Equation (1.2) provides an acceptable approximation to the underlying principle of increasing *cwnd* size by one full-sized segment per RTT.

When a TCP sender detects segment loss according to the retransmission timer and the given segment has not yet been resent by way of the retransmission timer, the value of $ssthresh$ set as

$$ssthresh = max\left(\frac{FlightSize}{2},\ 2 * SMSS\right) \tag{1.3}$$

where, $FlightSize$ is the amount of outstanding data in the network.

However, when TCP sender detects segment loss according to the retransmission timer and the given segment has already been retransmitted at least once by way of the retransmission timer, the value of $ssthresh$ is kept constant. Basically, slow start and congestion avoidance mechanisms are the dynamic resource management problems that can be formulated as system control problems, in which system senses its state and feedback to those users who are adjusting their controls.

Fast Retransmit/Fast Recovery: A duplicate ACK is used by TCP to inform sender about dropped segment, out of order delivery, reordering of segment etc. This mechanism is useful for sender to recover from a loss. A TCP receiver sends an immediate duplicate ACK, when an out-of-order segment arrives. The purpose of this ACK is to inform the sender that a segment was received out-of-order and which sequence number is expected. From the sender's perspective, duplicate ACKs can be caused by numerous reasons such as dropped segments, reordering of data segments and replication of ACK or data segments by the network. In addition, TCP receiver sends an immediate ACK when the incoming segment fills in all or part of gap in the sequence space. This will generate more timely information for a sender recovering from the loss through a retransmission timeout, fast retransmit or advanced loss recovery algorithm.

The 'fast retransmit' is an algorithm used by TCP sender to detect and repair loss, which depends on the number of duplicate ACKs received. In fast retransmit, the receipt of three duplicate ACKs indicates segment has been lost. On receipt of three duplicate ACKs, TCP performs retransmission of missing segment without waiting for expiry of retransmission timer. After fast retransmit, the 'fast recovery' algorithm governs the transmission of new data until a non-duplicate ACK arrives. The duplicate ACKs also indicate that most likely segments are leaving network and those are present in the receiver's buffer and not consum-

ing network resources. Furthermore, since the ACK 'clock' is preserved, the TCP sender can continue to transmit new segments. The fast retransmit and fast recovery algorithms are implemented together as per the following steps:

1. On receipt of the first and second duplicate ACK, sender sends a segment of previously unsent data provided that the receiver's advertised window allows the $[FlightSize_{total} <= cwnd + (2 * SMSS)]$ and that new data is available for transmission. Note that a sender using SACK need not send new data unless the incoming duplicate acknowledgment contains new SACK information.

2. When the third duplicate ACK is received, TCP sets *ssthresh* to less than or equal to, $[max\ (\frac{FlightSize}{2},\ \ 2 * SMSS)]$.

3. The lost segment is retransmitted and *cwnd* is set to $[ssthresh + (3 * SMSS)]$. This artificially 'inflates' the *cwnd* by the number of segments (three) that have left the network and which the receiver has buffered.

4. For each additional duplicate ACK received (after the third), the *cwnd* incremented by $SMSS$. This artificially inflates the *cwnd* in order to reflect the additional segment that has left the network.

5. When previously unsent data is available and the new value of *cwnd* and the receiver's advertised window allow, then TCP sends $[1 * SMSS]$ bytes of previously unsent data.

6. When the next ACK arrives that acknowledges previously unacknowledged data, TCP sets *cwnd* to *ssthresh*, the value set in step-2, called window 'deflating'.

Figure 1.4 shows one of the approaches, AIMD, to control congestion. In slow start phase, *cwnd* growth is exponential due to after every successful acknowledgment of packets *cwnd* size gets doubled. The exponential growth of the *cwnd* against RTT may quickly lead to congestion. To avoid congestion before it happens, the congestion avoidance algorithm is implemented after it reaches to *ssthresh*. The linear increase during congestion avoidance is achieved by incrementing the *cwnd* by $[1/cwnd]$ each time an ACK received. In this way the *cwnd* is effectively increased by one every RTT. Transition from slow start to congestion avoidance phase is controlled by the variable *ssthresh*. Multiplicative Decrease (MD) is the algorithm that controls this variable. With MD, TCP sets *ssthresh* to half of the current *cwnd* each time a timeout occurs. Therefore, if there are consecutive timeouts, MD reduces the sending rate exponentially.

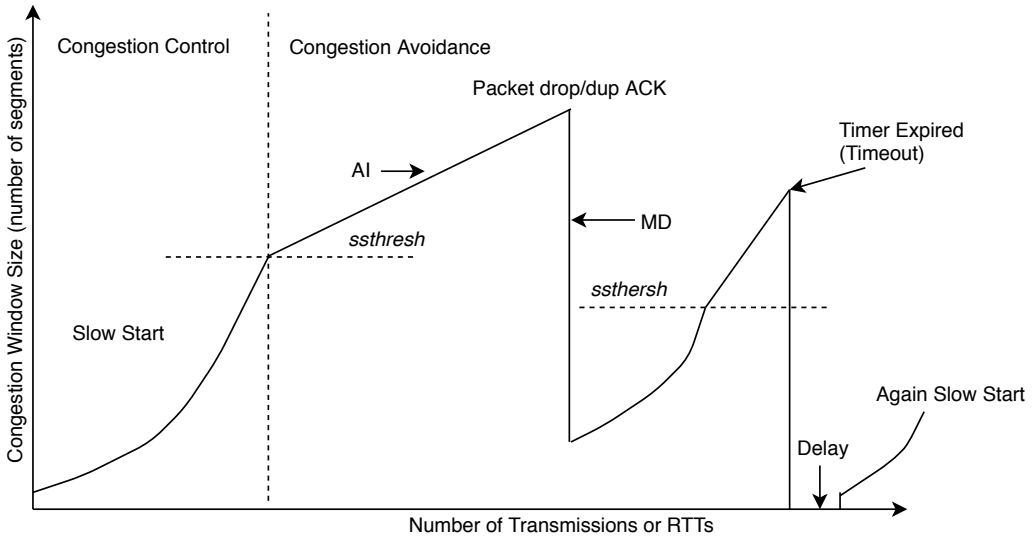


Figure 1.4: Congestion control algorithm: AIMD

1.3 QUIC Protocol

QUIC is an experimental transport layer accompanying application layer protocol, which has been designed and developed by Google group in 2012 [Willis (2013)]. The main motive for this initiative is to enhance existing estimated performance of TCP. QUIC's subsidiary goals are to reduce connection establishment latency and congestion control using bandwidth estimation. Unlike TCP, QUIC extended congestion control scope to application space instead of restricting to kernel space. This will empower congestion avoidance algorithms and faster upgradation.

To achieve prime motive, QUIC reduces overhead during connection establishment phase. In TCP, most of the HTTP connections are using Transport Layer Security (TLS) with compression, whereas QUIC uses key exchange as initial handshake and part of protocol. Once a client starts connection, the response packet inserts the useful data required for encryption to the future packets. This setup information is used in future to setup new or repeated connection [Bright (2018)].

As shown in Figure 1.5, QUIC is built on top of UDP and has not used error correction at the bottom space. However, every QUIC request is multiplexed and error corrected separately by QUIC driver, instead of underlying transmission protocol. In QUIC, control data includes packet routing information, which helps to continue serving streams independently other than error detection. This mechanism of QUIC assists to improve performance of er-

ror, prone links. In TCP, prior to identifying link error, considerable amount of data may be received and this data will be blocked or get flushed during error correction. However, in QUIC, received data after error occurs is free to be processed while the single multiplexed stream is repaired [Behr and Swett (2018)].

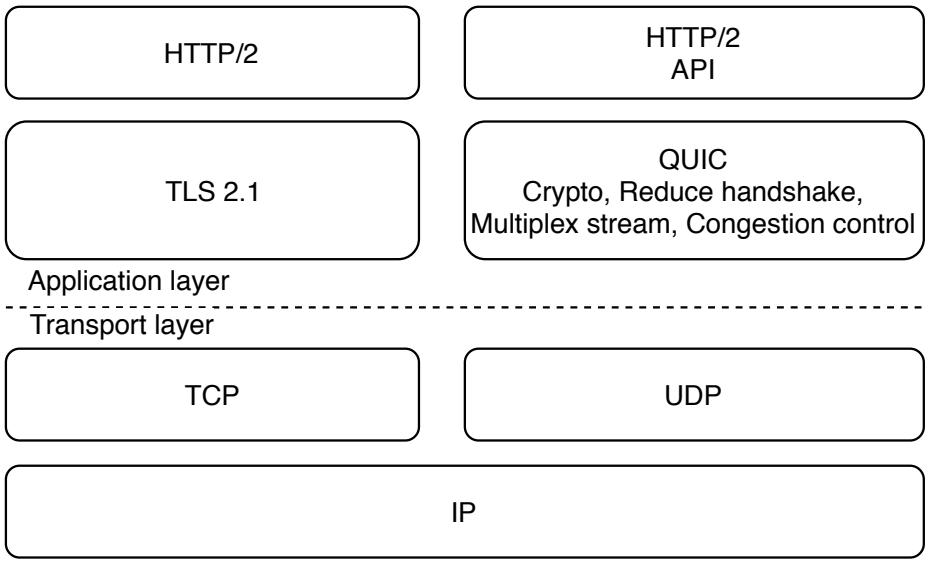


Figure 1.5: QUIC placement in protocol stack

To improve overall throughput and reduce latency of the network, QUIC protocol structure has been designed. For example, packet sizes are selected in such a way that it will sprawl at the boundaries of the encryption protocol used, which results in encrypted data, not to wait for partial packets. However, as TCP is unaware about underlying supporting protocols, 'one size fits all' approach is used. Even though packet selection procedure needs to negotiate with layers running on top, QUIC finishes off the procedure within a single handshake [Roskind (2013)].

The QUIC supports connection migration (Figure 1.6e) in which during call, user can move out from one network to another network (network switch event) without call break. However, in TCP, existing connection is completely torn and need to be re-established with new network to continue the call. To support for connection migration, QUIC has introduced 64 bit connection identifier, which holds both multiplexing data and unique network-wide ID. This mechanism re-establishes connection only by sending a packet carrying connection ID even if the user's IP address changes. As QUIC is implemented in application space, this invokes additional overhead due to the data moving between applications (context switching). This issue is resolved by hosting each application on separate connection

on top of UDP, which is similar to TCP with HTTP/2.

The structure of QUIC allows future updates on the fly, as it does not require changes at kernel level. To improve congestion control performance at initial stage of development, QUIC comes with Forward Error Correction (FEC) functionality. But due to failure, this is kept as longer-term goal. At present, QUIC uses HTTP/2 header compression and header frames suffers from Head Of Line (HOL) blocking. By adopting FEC mechanism, HOL blocking issue will get resolved as source errors can be eliminated before they reach to the application layer.

The important fret to switch to QUIC is wide deployment of TCP in the existing network infrastructure, as well as many of the 'middle-boxes' are tuned for TCP and may even block UDP to control packet flooding. Google carried out a number of experimental investigations to justify QUIC presence by proving very few connections which were blocked in this manner. In addition, QUIC made provision for a quick fallback-to-TCP structure. The chromium's network stack opens both QUIC and TCP connections concurrently, which allows QUIC to fallback with zero latency [Kuehlewind and Trammell (2017)].

1.3.1 QUIC Protocol Features

The basic features of QUIC protocol is described in [Hamilton *et al*. (2016)].

1. **Multiplexed streaming:** As shown in Figure 1.6a, QUIC multiplexes different streams over same UDP connection. QUIC is on top of UDP, hence out of order delivery is possible, which helps in solving HOL blocking as shown in Figure 1.6d.

2. **Less connection establishment latency:** The time taken to set-up connection by QUIC is at most one RTT and if in case the client has already communicated with the server, then it takes zero-RTT as shown in Figure 1.6b. This reduces connection establishment latency as compared to traditional TCP. Even in authentic and secure connection, QUIC (QUIC-Crypto) will take one-RTT, in contrast TCP+TLS need three-RTTs.

3. **Authenticated and encrypted header and payload:** To secure data delivery by avoiding third-party manipulation, QUIC packets are authenticated and payload part is encrypted. However, if the payload is partially encrypted, it still gets authenticated by the receiver.

4. **Stream and connection level flow control:** QUIC consists of connection level (like TCP) and stream level (within connection multiple streams are present) flow control mechanisms. Based on QUIC receiver capacity, it will advertise the absolute bytes of data within each stream or connection (aggregate stream data).

5. **Flexible congestion control:** QUIC has flexible/pluggable congestion control mechanism. At present, QUIC has CUBIC [Ha *et al.* (2008)] and BBR [Cardwell *et al.* (2017)] functionalities. Out of these, as a default, CUBIC congestion control with packet pacing mechanism is used to handle network traffic. The packet pacing mechanism, shown in Figure 1.6c, is useful to manage busty data. In QUIC, ACK frame supports up to 256, NACK (duplicate ACK in TCP) ranges, so that QUIC withstands more effectively in the reordering situation than TCP with Selective-ACK (SACK).

6. **Connection migration:** QUIC connections are identified by a 64 bit connection ID (instead of four-tuple in TCP) randomly generated by the client. As shown in Figure 1.6e, in case of connection migration, QUIC connection identification number remains the same throughout the communication time so that it can survive for IP address changes and Network Address Translation (NAT) re-bindings. Also, the same session key has been used for automatic authentication and cryptographic verification of migrating client.

1.3.2 Challenges with QUIC Protocol

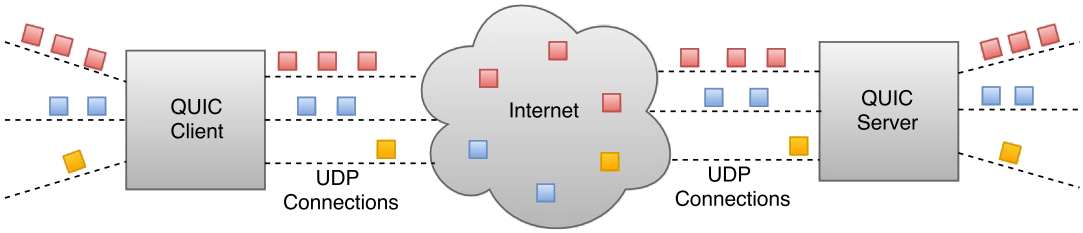
Following are the key challenges while working with QUIC protocol:

- Within past few years, due to growth rate of web technology, there is a demand in higher speed of operation and faster data rate requested by the users. Faster data rate leads to better user satisfaction, which results in user retention.

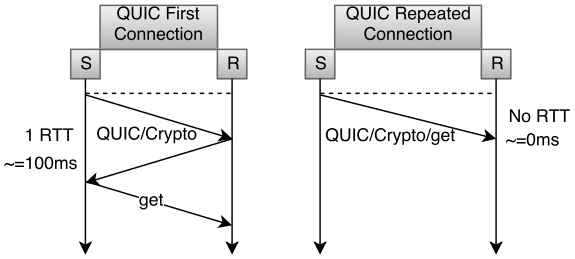
- To evaluate performance, the protocol specification and working model need to understand.

- To identify parameters that fairly compare performance with other transport layer protocols.

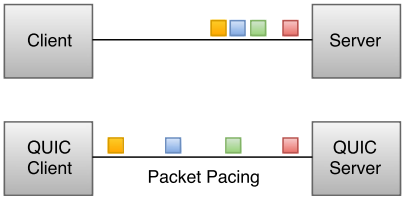
- QUIC protocol has been developed to compete with TCP, which is the most dominant and worldwide deployed transport layer protocol.



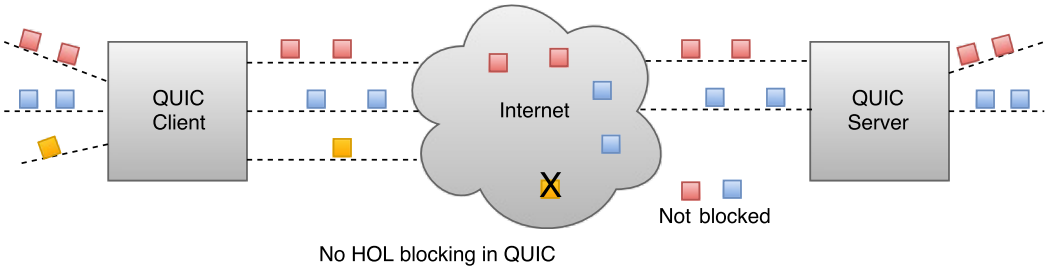
(a) Multiplexed transport



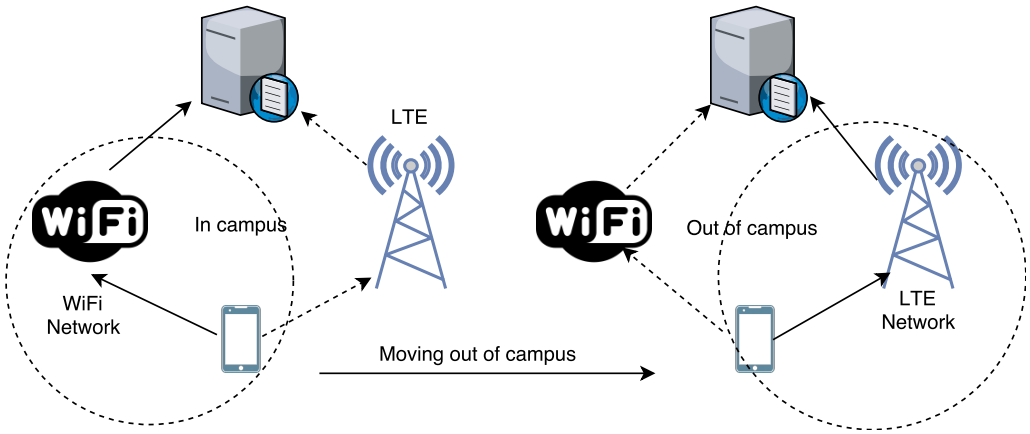
(b) Reduced handshake



(c) Packet pacing



(d) Solution to head of line blocking



(e) Connection migration

Figure 1.6: QUIC protocol features

- QUIC protocol source code is publicly available, for experimental investigations, but there maybe possibility that there is a gap between publicly available and actually deployed on Google client.
- QUIC source code is a complicated structure. To locate, modify and compile code is a tedious task.
- QUIC protocol is under rapid development since 2013, 43! stable versions are released. Limited literature and experimental studies are available which became obsolete before publication.

1.4 Transport Layer Performance Metrics

Following are the metrics used to evaluate performance of the protocol:

1.4.1 Throughput

The amount of data that can be transferred by the network from a sender to receiver during a period of time expressed in Kilo or Mega bits per second, as given in equation (1.4). In case of multiple flows, throughput is the sum of throughput of all flows.

$$Throughput = \frac{cwnd}{RTT} \tag{1.4}$$

1.4.2 Delay

A period of time required for a packet to reach receiving end. In networks total delay given in equation (1.5) is the sum of transmission delay, propagation delay, processing delay, and queuing delay, expressed in mili-seconds (ms).

$$Delay = Transmission\ delay + Propagation\ delay + Processing\ delay + Queuing\ delay \tag{1.5}$$

where
Transmission delay: Amount of time required to push all the packet bits into the link, which is a function of the packet's length and data rate of the link.
Propagation delay: Amount of time required for a message to travel from the sender to

receiver, which is a function of distance over speed with which the signal propagates. This is the function of physical distance between the two entities and speed of light.
Processing delay: Amount of time required to process the packet header, check for bit-level errors and determine the packet's destination.
Queuing delay: Amount of time the packet is waiting in the queue until it can be processed.

1.4.3 Fairness

A fairness is represented in terms of fairness index, which varies between 0 to 1. Fairness index is a measure when two or more applications compete to acquire network resources such as bandwidth, throughput, buffer space, in which, when all are getting equal share of resource, then fairness index is 1. One of the popular method used to measure fairness index is Jain's fairness index given in [Hassan and Jain (2003)] and expressed as in equation (1.6).

$$F(f_1, f_2, ..., f_n) = \frac{(\sum_{i=1}^{n} f_i)^2}{n.\sum_{i=1}^{n} f_i^2} \tag{1.6}$$

where,
n = number of flows sharing the resource,
f_i = resource allocation for i^{th} flow
Jain's fairness index is a predominant fairness measure for TCP flows. This fairness index ranges from $1/n$ to 1. The value of this index tends to 1, only if each flow has an equal share, and tends to $1/n$, if a single flow acquires all network resources.

The phase plot in Figure 1.7 shows fairness and efficiency of the network. Each axis represents a particular user or sender allocation. In this case, there are two users and their allocations are represented by x1 and x2. If the capacity of the network is C, then an optimal operating line is at (x1 + x2 = C).

In general, AIMD converges to fairness and efficiency. Suppose, if at any point 1, TCP window size of both senders Additive Increase (AI) their sending rates, AI results moving line parallel to x1 and x2 towards efficiency line, since both senders increase their rates by the same amount. AI continues until network becomes overloaded that is touching to efficiency line. At this point of time, both senders decrease their sending rates by a MD fashion with a specific factor, in this case, half of previous. The lines drawn parallel to x1 and x2 cross at new operating point 2. Then senders increase their sending rates in additive increase fashion and once again hits efficiency line towards optimal point. In this way, sender's rate converges towards optimal operating point [Molnár *et al*. (2009)]. This means,

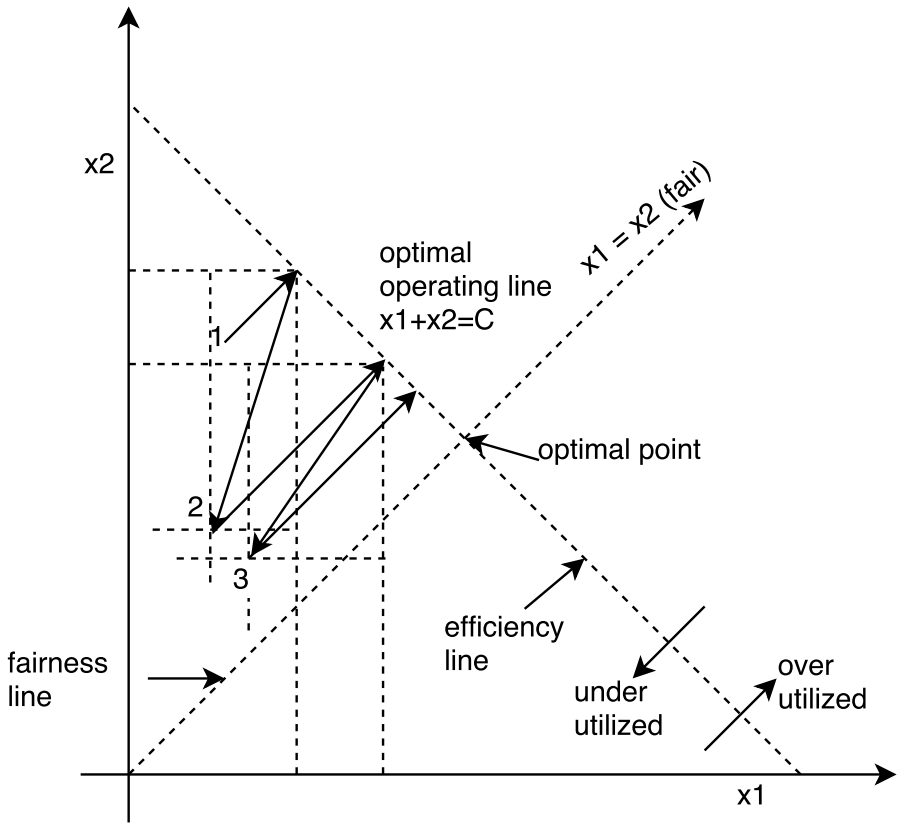


Figure 1.7: Phase plot shows relation of fairness with efficiency

AI increases efficiency, whereas MD improves fairness.

1.4.4 Transmit Time

A *cwnd* size is one of the important parameters and decides time required for sender to transfer available data to receiver. Equation (1.7) is used to calculate time (T) in second, which is required for sender to send data.

$$T = RTT * log_2 \left[\frac{N}{cwnd_{initial}} \right] \tag{1.7}$$

where,
RTT = round trip time in second,
N = number of segments,
$cwnd_{initial}$ = initial congestion window size in segments

whereas, $1 segment = 1520 - 40$ (TCP header size) = 1480 byte.

1.5 Motivation and Organization of the Thesis

In today's fast paced world, the rate and quality of information transfer has ascended to paramount importance. By 2021, Cisco Visual Networking Index estimates that annual global IP traffic will hit 3.3 ZB, of which 63% will be shared by wireless and mobile traffic. Furthermore, the report states that video traffic will make up 83% of all consumer IP traffic, of which 13% will be live streaming video. As mobile and IP enabled devices grow into an IoT driven by 5G technology, IP traffic is bound to increase even faster in the forthcoming years [Cisco (2018)].

In this scenario, it becomes ever more relevant to establish networking protocols to handle the rapid growth in IP traffic usage. The popular Hypertext Transfer Protocol (HTTP), published in RFC2616 in 1999, has become outdated with the changes in Internet usage and traffic. To overcome this, Google developed its SPDY web transfer protocol [Elkhatib *et al.* (2014)], using multiplexed TCP streams and header compressions on top of HTTP1.1 with the aim of improvement in Page Load Time (PLT). In 2015 the Internet Engineering Task Force (IETF) released the largely SPDY based HTTP/2 protocol in RFC7540 [Belshe *et al.* (2015)] as a successor to the HTTP 1.1 protocol.

To maximize bandwidth utilization and data delivery rate, UDP protocol can be one of the options, whereas on the contrary TCP protocol limits data rate by adding reliability as the key feature. Till today, numerous modifications in TCP have been proposed and presented as the TCP variants. These variants have limitations to enhance data rates, desired for video streaming which is responsible for congestion in the network. This ultimately affects QoS by reducing packet delivery ratio. The encryption technique, TLS, used in TCP causes jitter effects, which degrade the performance of TCP and modifies TCP connections. QUIC protocol developed by Google overcome limitations of TCP such as connection migration, reduced handshake, crypto security and streaming based on on HTTP/2 services. Internet draft of QUIC protocol mentioned that the congestion control being introduced and needs improvement as per application demand [Hamilton *et al.* (2016)].

Chapter 2 presents literature survey based on study of existing contributions related to dissertation topic. Mainly survey is based on three subtopics transport layer, congestion control and QUIC protocol. This includes existing theories, both generic and specific arti-

cles written on the topic, research done in the field in order of oldest to latest and challenges being faced in ongoing work.

Chapter 3 elaborates proposed modified QUIC protocol and mathematical illustration of congestion window growth. To reduce transport latency window update frame is attached to acknowledgment frame instead of being sent separately, which reduces delay to update *cwnd* size. To test the performance of proposed modification two different testing environments are created. The results are compared with existing mechanism of QUIC and TCP with respect to throughput and delay.

Chapter 4 addresses congestion control issues of ModQUIC and QUIC protocol and provides a solution to resolve those. ModQUIC and QUIC protocols by default equipped with CUBIC congestion control mechanism and analysis shows that performance limitations are due to high BDP and cubic functionality. To improve performance in this, BBR, an alternative congestion control mechanism to CUBIC, is suggested. However, to improve performance with respect to cubic function, decrease factor $\beta = 0.3/n$, is suggested instead of 0.3 as in CUBIC for n, number of flows. The experimentation section verifies the ModQUIC performance with CUBIC and BBR congestion control mechanisms.

Chapter 5 presents a case study, to verify the performance of the protocol in public domain, since in previous chapters, the performance of protocol is verified by using institute campus reliable network. For conducting the case study,a testbed setup has been developed using Reliance Jio along with Raspberry-Pi as a router.

Chapter 6 presents the conclusions driven by this research and the possible future work.

Chapter 2

Literature Survey and Research Objectives

2.1 Introduction

In recent days almost all computing, communication and control are carried out through TCP/IP services. Many of us access some form of TCP/IP networks several times a day, either from fixed locations or while moving and through wired or wireless medium. The present day network services range from traditional information gathering to critical business transactions. As an example, in online business, if PLT is high, users are irritated and this results in low profit. To maximize the benefits of online services, it is absolutely important to enhance the performance of TCP/IP network.

The emergence of high-speed network is not covered by deployed popular TCP variants like Reno, NewReno, SACK etc. The data transmission speed of the network normally depends on increase in *cwnd* size along with RTT. The growing demand for wireless network technologies highlighted the need of transport layer protocol modification. As most of the TCP variants are designed and developed for wired networks, these variants blindly decide that congestion is the only cause of packet loss. But packet loss may also occur due to non-congestion related issues such as short term radio frequency interference, bit errors or random losses. This indicates that there is no congestion as buffer overflow is not a cause of packet loss. To handle such non-congestion packet losses, there is a demand to develop a loss recovery mechanism rather than simply halving *cwnd* size.

To resolve congestion collapse, many of the proposals were submitted by researchers from all over the world. Most of them suggested solution as network aware rate control

with receiver driven flow control. A *cwnd* size is the estimation of number of data packets that network can accept without any congestion. To calculate bound on the number of data packets present in the network for a particular situation is the bound for outstanding packets in the network, e.g. *rwnd* (flow control limit), is less than *cwnd* (congestion control limit). This is the real definition for TCP rate bound, but in most of the proposals, only *cwnd* is considered as a rate controlling parameter. Usually, it is assumed that data processing capacity of the end nodes are much greater than the data transfer rate of the network.

To replace or, as an alternative to complex structure of TCP, Google has developed a QUIC protocol. Initially, developed TCP variants were designed to avoid overflow at input buffers to the receiver end. This mechanism was developed based on the *rwnd* size, where a sender transmits a prepared data packet. This must not increase receiver's capacity (*rwnd* size) and receiver may not be able to process data as fast as sender sends. This results in buffer overflow and to avoid this, receiver reports by reducing sliding window size. That is, the whole transmission will eventually synchronize with receiver's processing rate.

To avoid unnecessary packet drops, precaution must be taken by the end nodes. A Floyd (1994), suggested solution to this by sending congestion notification to the sender. When bulk data is considered, data arriving time of last packet is only important and there is no need to take care of individual packets. For delay sensitive traffic such as mice traffic which exists for a very little time in the network, for example, pressing like button in facebook or telnet traffic need to take care. If unnecessary packet drops or packet retransmissions take place, this results in very high throughput degradation for such low bandwidth network. To solve this, Sally Floyd designed Explicit Congestion Notification (ECN) mechanism, which was investigated by Kwon and Fahmy (2004) and further extended by Ramani and Karandikar (2000) for wireless and lossy links.

However, nowadays, due to digitization more digital documents are generated by numerous applications. These digital documents come under the category of elastic traffic, and the network performance depends on number of connections currently sharing links of the network, which varies based on start and end of the flows. To analyze flow level performance of the network, a flow based network state traffic model shown in Figure 2.1 was suggested by Sukhov *et al*. (2011). The flow level congestion control protocol standards sometimes remain unaware of network resources available, which creates unexpected effects on the Internet.

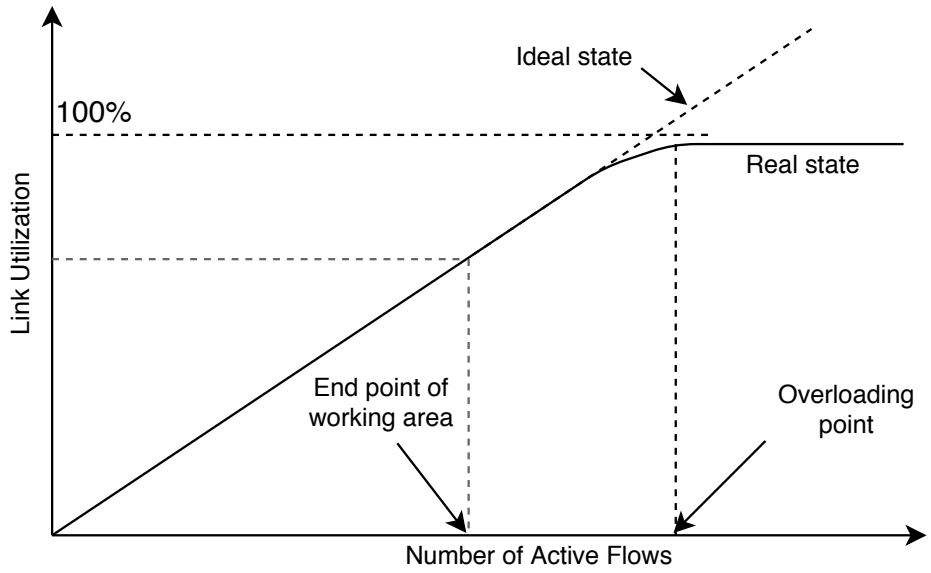


Figure 2.1: Flow based traffic model for uncongested backbone links

2.2 Literature Survey on Congestion Control

To estimate congestion state in the network, most of the congestion control mechanisms are divided into three groups: Reactive (Loss-based), Proactive (Delay-based) and Flow level. The popularity of TCP brought many approaches into existence based on fine tune congestion state impairments. The congestion control is an intelligent network resource aware mechanism with effective use of resources available in packet-switched network. Congestion control is a one of the largely studied areas in the Internet research conducted over the last 25 years. A number of proposals were submitted targeting to improve various aspects of the congestion-responsive data flows. Several surveys were prepared by researchers targeting specific requirement of interest like congestion control in adhoc networks by Hanbali *et al*. (2005), congestion control for mobile adhoc networks for single and multiple flow by Lochert *et al*. (2005), and Dong *et al.* (2015), congestion control for non-TCP protocols by Widmer *et al*. (2001), congestion control for networks with high levels of packet reordering by Leung *et al*. (2007), fairness issues in congestion control by Hasegawa and Murata (2001), Host-to-host congestion control for TCP by Afanasyev *et al*. (2010), congestion control for wireless networks by Balakrishnan *et al*. (1997). This section presents a survey on congestion control schemes for high-speed data networks and QUIC protocol develop-

ment[1]. Table 2.2 summarises information of standard TCP variants with their services and limitations.

2.2.1 Congestion Control Challenges

In literature, congestion control mechanisms are presented as an artificial feedback system and are deployed in all routers. The Internet features are continuously growing in size, diversity and applicability. These features play the most significant role to integrate many networks such as trading and banking. A firm understanding of how this fundamental resource (Internet) is controlled becomes even more decisive. Following are the key challenges in front of researchers working to reduce congestion in the networks [Hassan and Jain (2003), Peterson and Davie (2007)].

1. **High speed:** In recent years, the growth rate of web technology and performance optimization industry indicates an increasing importance and demand for speed of operation. The results show that speed is not a psychological need in accelerating and connecting world. Through online business, within a click, items are available at our door step. But the performance of this business is decided by how fast the online traders are compared to their competitors. More speed leads to faster sites and better user satisfaction, user satisfaction leads to user retention and retention leads to higher conversions.

   The high speed networking demand needs to understand many factors and analysis of fundamental limitations. Latency and bandwidth are the critical parameters, which play very important role in analyzing performance of the network along with speed.

2. **High packet delivery ratio:** This is the ratio of number of correctly received packets at the receiver to the total number of packets transmitted by the sender. The major sources of packet loss are buffer overflow at the intermediate routers and packet corruption caused by transmission errors. A high packet loss rate can severely degrade the performance of data and multimedia applications.

3. **Effective throughput:** Effective throughput is defined as the number of application bytes transferred within a second. For large file transfers, the effective throughput

[1]Prashant Kharat and Muralidhar Kulkarni (2019). Congestion controlling schemes for high-speed data networks: A survey, *Journal of High Speed Networks*, 25(2019), 41-60

Figure 2.2: Features and limitations of standard TCP congestion control mechanisms

Mechanism	Features	Solution for	Limitations
TCP-Tahoe [Jacobson (1988)]	Reactive, slow start, congestion avoidance and fast retransmit.	Congestion control, fast network resource discovery and long-term efficiency.	Already obsolete, straining the network with high-amplitude periodic phases.
TCP-Reno [Jacobson (1990)]	Reactive, fast Recovery	Congestion control with slow start.	Severe performance degradation in the presence of consecutive packet losses, random losses and packet reordering, unfair for different RTT flows, low efficiency for high-speed/long-delay networks.
NewReno [Floyd and Henderson (1999)Floyd *et al*. (2004)]	Reactive, fast recovery, offering resistance to multiple losses.	When multiple packet losses occurs as part of a single congestion event.	Prolonged recovery, waste resources if actual losses deviates from assumed pattern.
TCP-SACK [Mathis *et al*. (1996)]	Additional information in feedback messages, successfully delivered data packets report.	Limited information available in cumulative ACK.	TCP specification restricts the length of option field to 40bytes, SACK is not a universal solution to the multiple loss.
Eifel [Ludwig and Katz (2000)]	Differentiation between transmitted and retransmitted data packets.	To alleviate negative effects of packet reordering in TCP throughput, distinguish reordering and real loss.	Prolonged recovery, waste resources.
DSACK [Floyd *et al*. (2000)]	Reporting duplicate segments, receiver side modification.	Reordering or replicate data packet.	Receiver can send wrong information, either by intention or unintentionally.
TCP-RR [Claeys *et al*. (2016)]	Reactive, fast recovery, exponentially backs off the sending rate after detecting the first packet loss within a window	Packet recovery in single congestion event, link utilization,	Fails to detect and recover in multiple loss, waste of resources if actual loss deviate from assumed.

of the application is a key performance measure. An inefficient transport layer algorithm can significantly reduce the effective throughput even if the underlying network provides a very high-speed communication channel.

4. **Less throughput variation:** Throughput variation is a metric to measure the variability in the received bandwidth over a given time scale. In general, larger the time scale is, lower the throughput variability. For a given context, it is important to define a time scale over which throughput variability should be measured.

5. **Fairness:** Fairness is the property of transport layer, which plays an important role when two or more applications compete for resources in a congested router. Fairness can be defined over long term or short term. Long term fairness refers to the fair allocation of resources in the long run, whereas short-term fairness is defined in much smaller time scales. A deployed network algorithm may allocate bandwidth fairly in the long run yet exhibit unfairness in the short-term.

6. **Short-term radio frequency interference:** Gradually the demand of the wireless network is fattening and the traditional TCP variants are basically designed to serve wired network demands. This needs modifications as per the demand of wireless network. If we think of transport layer design in wired network, congestion is the primary cause of packet loss. Hence, traditional TCP variants do not respond to non-congestion packet loss. For example, if packet loss is due to interference caused by short term radio frequencies in which there is no router overflow event but still TCP reaction is to reduce transmission rate by reducing *cwnd* size. This reduction in transmission rate results in poor network throughput.

2.2.2 Random and Link Loss Based Congestion Control

The growing demand for mobile and wireless network technologies such as Cellular Communication and Wireless Local Area Network (WLAN) is shown in Figure 2.3 [Ludwig *et al.* (2002)], highlighting the need of transport layer protocol modification. It is seen through different surveys that most of the TCP variants are designed and developed for wired networks. These variants blindly decide that congestion is the only cause of packet loss [Acharya *et al.* (2010)]. If packet loss is due to short term radio frequency interference or bit errors or random losses, these are non-congestion events and buffer overflow is not a cause of packet loss. To handle this non-congestion packet loss, there is a need to develop

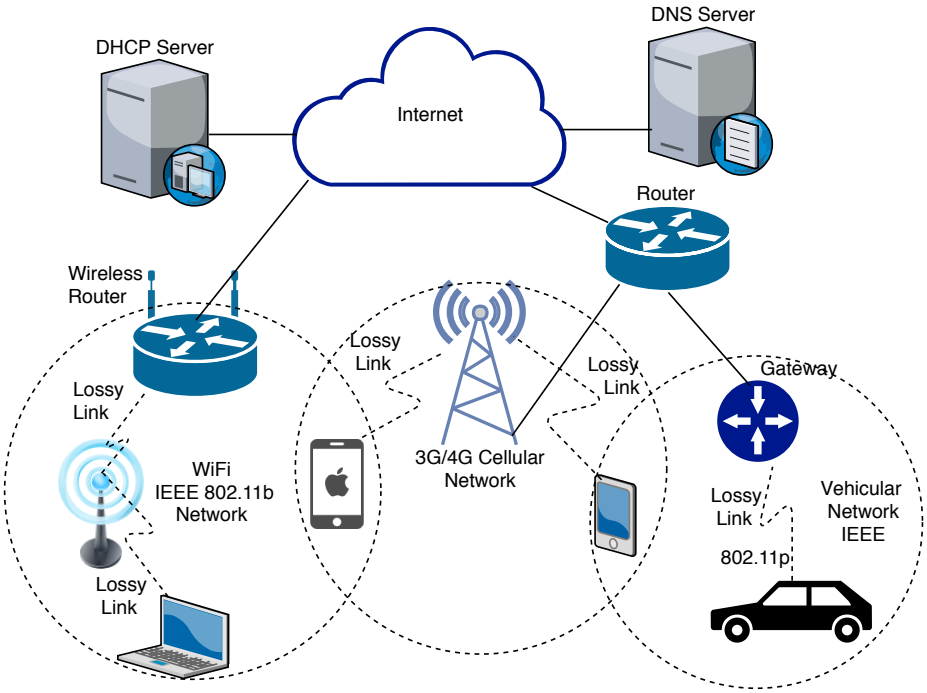


Figure 2.3: Typical network scenario with lossy links

a loss recovery mechanism rather than simply halving *cwnd* size. Figure 2.3 shows typical hybrid network with lossy links in which link losses are mainly due to interference and random bit errors. Table 2.5 summarizes the proposals which contribute towards random and link loss.

First time in 1999, congestion control policies applied to wireless multimedia Code Division Multiple Access (CDMA) network by Liu and Silvester (1999) in which they applied burst level congestion control approach. In this they studied reverse link and issues related to multimedia traffic in wireless network. Balakrishnan *et al.* (1997) compared different mechanisms for improving TCP performance over wireless links and tested in Local Area Network (LAN) and Wide Area Network (WAN) environments using throughput and goodput as the metrics. It has been observed that in traditional wired networks, congestion is the primary cause of packet loss. The networks with wireless and other lossy links also suffer from significant losses due to bit errors and handoffs. The selective acknowledgments and explicit loss notification techniques are quite effective and show significant performance improvement in lossy link especially when losses occur in bursts. Ratnam and Matta (1998) in their Wireless-TCP (WTCP) handle loss due to wireless link and mobility by using local retransmission strategy. Samaraweera (1999) suggested Non-Congestion Packet Loss

Detection (NCPLD) algorithm, which differentiates loss due to congestion from loss due to link noise. If there is packet loss due to any reason, recovery is the reaction. For this, retransmission of the packet is one of the solutions. But as soon as packet drop takes place, reduction in the rate of the transmission is also a reaction assuming that there is congestion. To identify loss is due to non-congestion type, which may be due to link noise, a RTT based calculations are used to identify network utilization and if it shows underutilized fast retransmission takes place assuming that loss is due to non-congestion. As we come to know that loss is due to non-congestion event and continues transmission with the same rate as no further reduction in *cwnd* size.

Parsa and Garcia-Luna-Aceves (1999) proposed TCP SantaCruz to improve TCP congestion control performance in heterogeneous network scenario. This protocol addressed issues such as asymmetrical paths, out of order delivery, lossy links, less bandwidth and delay variation. In this protocol, only forward path delay is measured instead of RTT. To avoid congestion, a specific threshold is assigned to the number of packets present in the bottleneck link. Here, there is no need to count acknowledgments to set next limit for *cwnd*. The simulation results show that, TCP-SantaCruz achieves higher throughput, smaller delays, and smaller delay variances than Reno and Vegas. Ma *et al*. (2000) developed TCP-Fast for IP network with a wireless link. The basic idea of the TCP-Fast algorithm is to delay the ACKs, being transferred from the TCP destination towards the TCP source. TCP-Fast can speed up TCP flow control time, reduce buffer oscillation, increase bandwidth utilization, increase throughput, and reduce packet losses in the IP networks.

Ramani and Karandikar (2000) suggested modified ECN mechanism for wireless and lossy links in this sender aware mechanism was developed, which reports some of the packet losses are not because of congestion. This was done through explicit feedbacks from the network as a congestion notification about the link status. Mascolo *et al.* (2001) in TCP-Westwood rely on end-to-end bandwidth estimation to differentiate whether the cause of packet loss is either congestion or wireless channel effect, which is a major problem in TCP-Reno. Unlike TCP-Reno, Westwood used $ssthresh$ and *cwnd* parameters to control congestion rather than three duplicate ACKs concept to control *cwnd* size. In Westwood, random losses over radio link are differentiated from congestion and cause of random loss is radio interference and avoids unnecessary *cwnd* reduction. TCP-Westwood is extremely effective in hybrid (wired and wireless) networks where they claimed throughput improvements up to 550% as per their observations [Wang *et al.* (2002), Wang *et al.* (2002)].

Fu and Liew (2003) in TCP-Veno, monitored the network congestion level and used

that information to decide whether packet losses are likely to be due to congestion or random bit errors. TCP-Veno modified multiplicative decrease to improve its applicability by using *ssthresh* adjustment based on the level of congestion rather than fix factor. Veno tried to mention connections in the operating point region for longer or full time so that whole bandwidth of the network gets utilized. One of the salient features of Veno was it's only sender side modification. Xu *et al.* (2004) developed TCP-Jersey which is capable to distinguish the packet loss due to wireless effects like radio interface, bit error, random errors from the congestion effect and decisions are made. TCP-Jersey was divided into two parts: Available Bandwidth Estimation (ABE) algorithm and routers with congestion warning mechanism. The ABE is a sender side modification which contentiously monitors and estimates bandwidth available to the application and based on ABE's estimation, sender will adjust the rate of transmission. A congestion warning is the router configuration for a network through which routers generate and send an alert message to end nodes by marking all packets as soon as incipient (beginning to develop) congestion is detected. Based on packet marking, sender comes to know whether packet loss is due to congestion or due to wireless link errors [Shi *et al.* (2010)].

Biaz and Vaidya (2005) proposed de-randomization to distinguish congestion losses and random losses. As described in the introduction part, most of the variants were designed basically by assuming packet loss is due to congestion. To mitigate this misconception de-randomization is one of the solutions in which bias is used to differentiate congestion and wireless link loss. Mainly this concept is used in heterogeneous wireless error prone links. Through simulation results, accurate boundaries were found out to differentiate congestion loss and wireless link loss. This technique produced 95% accuracy in congestion loss detection, whereas 75% accuracy in case of wireless link loss detection.

Rath *et al.* (2006), proposed a cross layer based congestion control technique called Reno-2 to the wireless networks. In this, both transport layer and physical layer were working hand in hand to control congestion. In Reno-2, as per channel conditions and interference level, physical layer controls the power of transmission signal and transport layer controls congestion by controlling flow. MATLAB simulation results showed that the cross layer congestion control technique provides performance improvement in terms of throughput and *cwnd* variations. The data transmission using wireless network suffers from insufficient bandwidth, transmit latency and high interference. The standard TCP congestion control mechanisms fail to identify available bandwidth or buffer size and blindly halves *cwnd* after receiving three duplicate ACKs. Chang and Li (2012) proposed cross

layer messages to identify bottleneck link status as a access or shared. The *cwnd* size is adjusted based on the link location and packet loss and differentiated as network congestion or random loss. The cross-layer messages based on Adaptive Modulation and Coding (AMC) were used to adjust *cwnd* size. In this, the numerical results proved that the proposed approach significantly improved the Goodput with precise packet loss identification at various loss rates. As an example, for wireless link with 4% packet loss rate, the proposed approach increases Goodput up to 77.25%, compared to NewReno. Figure 2.4 shows two cross-layer design approaches; (a) for small amount of information exchange, direct communication between each other is possible, normally defined as a non-manager method. (b) in this, indirect communication takes place between layers through vertical plane (shared information), normally defined as a manager method [Fu *et al.* (2014), Pham and Hwang (2017)].

van den Berg *et al.* (2015) proposed a complete distributed algorithm, which creates prioritized TCP flows to allocate network resources like bandwidth within flows. This enables autonomous conversions to loss of network resources due to cyber attack or failures by ensuring that users receive prioritized utility from available network resources. This approach is fully-distributed, self-adaptive to prioritization of mission-critical TCP with a weighted Nash bargaining solution to distribute network bandwidth among the flows.

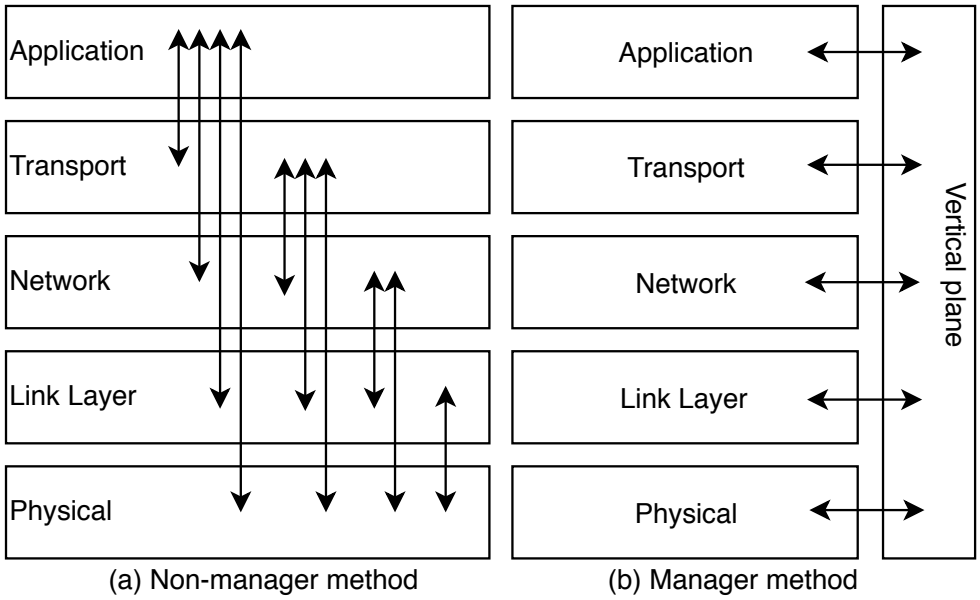


Figure 2.4: Cross layer approach

Figure 2.5: Congestion control mechanisms addressing random and link losses

Mechanism	Features	Solution for	Limitations
TCP Westwood (2001)	Available bandwidth estimation (ACK granularity), faster recovery.	Random errors due to wireless interference.	Unfair if network has limited buffering capability, Bandwidth estimate gives inaccurate results.
TCP Westwood+ (2004)	Estimate of available bandwidth (RTT granularity)	Random errors due to wireless interference	Unfair if network has limited buffering capability, bandwidth estimate gives inaccurate results.
TCPW CRB (2002)	Available bandwidth estimate (combination of ACK and long-term granularity), identifying predominant cause of packet loss.	Random errors due to wireless interference	Unfair if network has limited buffering capability, Only simulation results are available no real life experimentation
TCPW ABSE (2002)	Available bandwidth estimate (continuously varied sampling interval), varied exponential smoothing coefficient	Random errors due to wireless interference	Unfair if network has limited buffering capability, only simulation results are available no real life experimentation
TCPW BR (2003)	Loss type estimation technique (queuing delay estimation threshold, rate gap threshold), retransmission of all outstanding data packets, limiting retransmission timer back-off	Random errors due to wireless interference	Only simulation results are available no real life experimentation
TCPW BBE (2003)	Effective bottleneck buffer capacity estimation, adaptive reduction coefficient, congestion window boosting	Unfair if network has limited buffering capability, hybrid estimation technique	Only simulation results are available no real life experimentation
TCP Jersey (2004)	Available bandwidth estimation and congestion warning approach	To improve performance in heterogeneous network	Only simulation results are available no real life experimentation

2.2.3 Improving Efficiency in High-speed and Long-delay Networks

The problem of improving efficiency in high-speed and long-delay networks sometimes referred as a Bandwidth Delay Product (BDP) problem. To discover network resources, minimum time required is $[D * (W/RTT)]$, where D is maximum size of the data packet and W is the *cwnd* size. A network with Giga bit capacity takes almost hours of time to deliver all packets that depend on the time required to discover network resources. This section deals with congestion control algorithms for high speed links, use of network resources, quick response to network changes and fairness [Labovitz *et al.* (2010), Briscoe *et al.* (2016)].

The convergence time is the speed with which system reaches its stable state. It is the most important parameter. However, because of binary nature of feedback used in the network to control congestion, the system generally does not converge to a single steady state. As shown in Figure 2.6, the time to reach target load determines responsiveness and size of oscillation determine smoothness. Ideally, time and oscillations are to be small [RFC3649 (2003), Floyd (2003)].

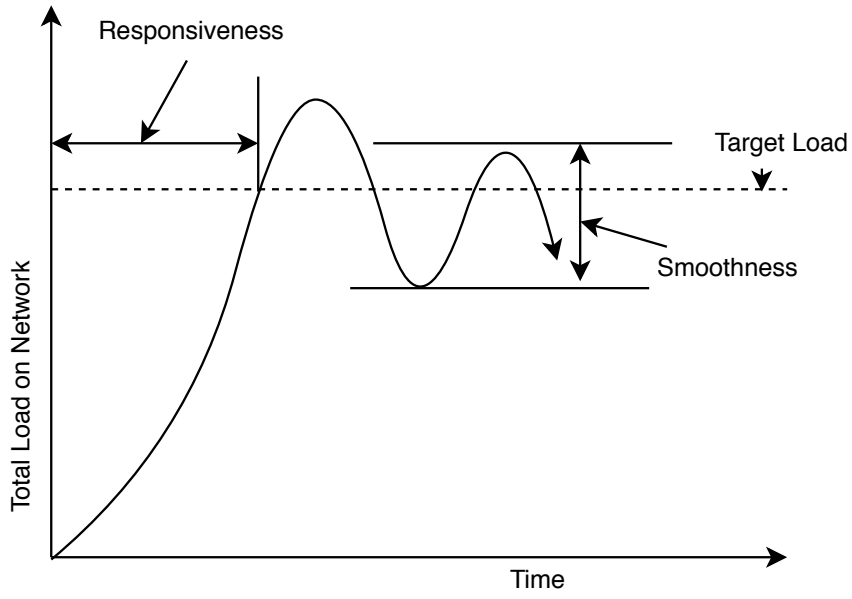


Figure 2.6: Network convergence response

Standard TCP variants are suitable to solve pure congestion problems and not efficient for high speed networks such as optical or satellite network in which reaction time is very less. To handle congestion and to improve efficiency in high-speed networks, High Speed

TCP (HS-TCP) has been proposed by [Floyd (2003)]. In this experimental approach, efficiency improvement in high BDP networks and fairness in the lossy environment are the set objectives. To achieve set objectives in HS-TCP standard, NewReno protocol is modified by replacing increase factor α as a function of W, $\alpha(W)$, in congestion avoidance phase and decrease factor β by $\beta(W)$ after loss detection. When *cwnd* size is more, that is no loss situation, HS-TCP grab network resources aggressively, whereas vice-versa after loss detection. This opportunistic behavior improves the efficiency of high-speed and long-delay networks. But this aggressive and conservative behavior causes significant packet loss and occurs frequently. Table 2.7, summarizes proposals which are contributed towards high-speed and long-delay networks.

The researchers claim that traditional AIMD congestion control approach is not suitable for high-speed and long-delay networks. Kelly (2003) says that, AIMD is not effective and proposed Scalable-TCP (STCP) as an alternative to HS-TCP. In AIMD, complicated task is to calculate increase and decrease rate coefficients in STCP. This is simplified by introducing Multiplicative Increase Multiplicative Decrease (MIMD) approach. The increase and decrease approaches used in STCP are very aggressive, which result in sharpening the characteristic graph. But because of this aggressiveness, it creates intra-fairness congestion and it leads further to the congestion collapse. Particularly for higher RTT values due to MIMD policy STCP flows are completely unfair with both STCP (intra-fairness) and other TCP flows (inter-fairness) [Radhakrishnan *et al.* (2011)].

Leith and Shorten (2004), Leith (2008) suggested H-TCP to cope up with fairness (inter, intra and RTT based) for competing flows with different RTT value and to improve the efficiency of TCP. The previous studies proved that flows with greater RTT always lose to flows with shorter RTT. In H-TCP, *cwnd* size increase function, α in congestion avoidance phase is a non-decreasing function of elapsed time, t since last congestion event, whereas H-TCP increase *cwnd* size by $\alpha(t)$ for any reference RTT instead of per RTT to resolve fairness issue.

Caini and Firrincieli (2004), claimed that standard TCP-NewReno gives lower throughput for long-delay networks. Normally in long delay networks, RTTs are differing largely causing terrible unfair resource distribution. To resolve this issue TCP-Hybla algorithm suggested modification in slow start and congestion avoidance phases, which makes it semi-independent on RTT value. In TCP-Hybla, scaling factor, ρ is calculated as, $\rho = RTT/RTT_{ref}$. That is higher RTT value and higher scaling factor result into fast *cwnd* growth and W increases with each received ACK. Increase factor in each phase is as: in

slow start phase, $W = W + 2^{\rho} - 1$ and in congestion avoidance phase, $W = W + \rho^2/W$. In addition to this TCP-Hybla supports: i) to calculate initial slow start threshold by estimating network capacity, which helps to improve convergence speed. ii) to smooth busty nature of transmission by setting the minimum delay between consecutive two packets using pacing.

In Binary Increase Congestion-control (BIC) TCP proposal, Xu *et al.* (2004) suggested a solution for RTT unfairness when the loss is detected simultaneously by two competing flows. In such situation in HS-TCP flow having RTT, x times smaller will get network share $x^{4.56}$ times more and in STCP for similar situation flow with larger RTT will get nothing. Basically, BIC is a modification in NewReno by adding rapid convergence phase. In rapid convergence phase, optimal *cwnd* size based on available network resources is discovered by binary search approach on detection of packet loss. There are two limits W_{min} and W_{max}, which are set to search *cwnd* size for successful packet delivery. On every achievement of W_{max}, W_{min} raises by setting to previous *cwnd* size. As soon as packet loss is detected, W_{max} is set to current *cwnd* size and enters into the fast recovery phase of NewReno. Also, BIC uses a different strategy for low loss network environment by reducing multiplicative decrease factor from 0.5 to 0.125 when *cwnd* size is more than 38 (value drawn from HS-TCP, $W = 1.2/\sqrt{p}$ where, p is loss rate 10^{-3} packet/s). To avoid uncontrolled growth of *cwnd* size as binary (logarithmic) search is used, precaution is taken by BIC. A BIC uses limited slow start and puts a limit on rapid convergence when search range is too high (high-BDP). There are two threshold values to control growth factor by S_{max} to limit maximum *cwnd* size growth value and S_{min} to increase *cwnd* size minimum by this factor in rapid convergence stage. In BIC *cwnd* probing decreases as window size approaches to set target value.

Ha *et al*. (2008), proposed an enhanced version of BIC known as CUBIC, in which they introduced RTT independent *cwnd* growth function. To achieve this, CUBIC picked up H-TCP approach of *cwnd* size calculation as a cubic function of elapsed time, t since last congestion event.

$$W_{CUBIC} = C\left(t - \sqrt[3]{\frac{\beta.W_{max}}{C}}\right)^3 + W_{max} \tag{2.1}$$

where,

W_{CUBIC} is *cwnd* size,

W_{max} is a *cwnd* just before last window reduction,

C is predefined constant (scaling factor), which preserves BIC properties RTT fairness,

limited slow start and rapid convergence.
β is decrease factor.
Window size reduction at the time of loss event is $W(t) = W(t^*) * (1 - \beta)$, where $W(t^*)$ is the *cwnd* size at the time t^* of packet loss i.e. W_{max}.

As an additional precaution, CUBIC has a mechanism to ensure performance must not be worse than standard TCP-Reno by checking simultaneously by calculating W_{reno}. With other experimental studies, it shows that performance and fairness property of CUBIC is very good. Also as CUBIC is available in Linux TCP suite (kernel version 2.6.16), currently this is most used congestion control algorithm.

Jin *et al.* (2005), proposed TCP-FAST, which is similar to Vegas with few modifications in *cwnd* estimation. In TCP-FAST, *cwnd* (W) is updated with fixed-rate, where $W = W_{current} * (RTT_{min}/RTT_{current}) + \alpha$, in which α need to select based on scalability and stability. When α is very high that will scale, W to any value, which results in wrong estimation of queuing delay. However, if α is small it stabilizes; but is not scalable. To avoid parameter fluctuations in the TCP-FAST exponential smoothing is used to calculate *cwnd* size. Results with different experiments show that TCP-FAST is not friendly with standard TCP variants like Reno, NewReno or SACK [Wei *et al.* (2006)].

NewVegas is an extended version of Vegas suggested by Sing and Soh (2005). NewVegas scheme defines a new phase called, rapid window convergence by retaining original benefits of Vegas and standard NewReno. In particular, delay based congestion control approach is more beneficial for high BDP networks due to sufficient time to converge network for different conditions. In rapid window convergence phase, slow-start state of the network is extended when buffering is more or it exceeds a certain threshold value. But, as a precaution, resource probing must be under control which means it is exponential like Vegas but with reduced intensity. In this phase for every RTT, *cwnd* size is increased by number of packets, x as; $x = (W_r)^{(-2^{3+n})}$ where, W_r is *cwnd* size when window convergence moves from normal to rapid state and n number of times early congestion indicator triggers in rapid window convergence phase. If n crosses a certain value (suggested is > 3) it terminates rapid window convergence phase to normal Vegas congestion avoidance phase. In NewVegas, if packet loss occurs, it switches to fast recovery mode, whereas, when time out (RTO) occurs, it will switch to slow start phase by resetting *cwnd* size. To avoid busty traffic at the initial phase of transmission because of rapid growth, packet pacing technique like Hybla is used. In pacing technique, a minimum delay is set between consecutive packets. To improve efficiency, NewVegas holds transmission till *cwnd* size increased to required

cwnd size to transmit the available packet, which resolves RTT estimation problem.

To improve TCP performance and preserve friendliness to standard TCP in high-speed networks, the Shimonishi and Murase (2005), suggested a solution as the adaptive Reno. In this proposal, a combination of constant increase and scalable increase techniques based on congestion state are used. When the network is congestion free (queuing delay = 0), a scalable window update function is used which supports a rapid growth of *cwnd* size, whereas, when network experience congestion (queuing delay = max), a scalable function approaching to zero. In this way, it gets adapted by switching from normal growth to rapid growth of *cwnd* size based on congestion state. Adaptive Reno improves network utilization up to a certain extent with a few of limitations, but it fails for high BDP.

A Modified Linear Quadratic Guass (MLQG) is proposed to control over long range dependence network and compared with standard LQG algorithm [Vernersson (2015)]. Here, congestion issue is solved with a stochastic optimal control problem. Kaneko *et al.* (2007) combine some useful characteristics of the Westwood, DUAL and Vegas to develop TCP-Fusion. Normally in Fusion technique, to achieve expected objectives, useful properties of different variants were collected and used. In Fusion algorithm, author suggested three different levels of queuing delay in second to set threshold, those are:
i) If the current queuing delay is less than the threshold, then *cwnd* size increased very fast per RTT with Westwood's scalable increase factor.
ii) If queuing delay is greater than three times of threshold value, then *cwnd* decreased by a number of packets in the buffer at that time (similar to Vegas estimation).
iii) If queuing delay lies in between one to three times of threshold value, *cwnd* size remain unchanged.

To take a precaution for optimum performance, Fusion must be equal to standard Reno. If Fusion *cwnd* size (W_f) is less than Reno *cwnd* size (W_r), then W_f is reset to W_r. As well as *cwnd* reduction factor in fast recovery phase is also modified to $\beta = max(0.5, RTT_{min}/RTT)$. But in Fusion, defining threshold is a big challenge and it makes Fusion more complex and undesirable. As our discussion is related to speed and latency issue, Radhakrishnan *et al.* (2011) and Grigorik (2018) addressed web traffic latency in their TCP Fast Open (TFO) proposal and book High-Performance Browser Networking respectively. While taking care of congestion control care must be taken for over all latency issues due to any reason. To address congestion control verses latency issue, in web application, latency and PLT are important factors that influence user satisfaction with a website. Even small improvements in latency lead to noticeable increases in site visits and user satisfaction, which result in higher

revenue generation. To achieve the 50% PLT improvement, SPDY (one of the application layer protocol) is aimed to make more efficient use of the underlying TCP connection. The SPDY has introduced a new binary framing layer to enable request and response multiplexing, prioritization and header compression. The loading of web page often requires fetching hundreds of resources from dozens of different hosts. In turn, this might require the browser to establish many new TCP connections, each of which will have to incur the overhead of the TCP handshake. Needless to say, this can be a significant source of web browsing latency, especially on slower mobile networks. TFO is a mechanism that aims to eliminate the latency penalty imposed on new TCP connections by allowing data transfer within the SYN packet. However, it does have its own set of limitations such as i) there are limits on the maximum size of the data payload within the SYN packet, ii) only certain types of HTTP requests can be sent, and iii) it works only for repeat connections due to a requirement for the cryptographic cookie [Briscoe *et al.* (2016), De Cicco *et al.* (2013)].

In similar fashion to contribute towards congestion control and reduce latency Google suggested QUIC protocol. Google released Internet drafts Chromium (2016) and Hamilton *et al*. (2016) in 2015-16 with full specifications of the protocol. QUIC protocol is a new multiplexed and secure transport with reduced handshake atop UDP. In both TFO and QUIC, once connection gets established between sender and receiver, for every next transmission, direct data transmission takes place within a time out. To improve bandwidth utilization and fast data delivery, UDP protocol is the option. For reliable service, TCP uses handshake for connection establishment, which slows down the network due to overhead. To overcome TCP disadvantage by preserving reliability property, QUIC is a new multiplexed and secure transport layer protocol[1] designed by Google group. QUIC is on top of UDP based protocol designed from the ground up and optimized for HTTP/2 semantics [Gizis (2016), Chromium (2016), Hamilton *et al*. (2016), Wei and Swaminathan (2014)]. The experimental performance analysis survey of QUIC protocol is separately discussed in the section 2.3.

2.2.4 Flow Level Congestion Control

In flow level congestion, when the number of flows with mean size σ arrive to a link with capacity C and mean transmission rate λ, where link load is $\rho > 1$. When number of flows in progress are increases then throughput tends to zero due to arrival rate, λ is greater than the

[1]there is a difference of opinions about QUIC location in the protocol stack in some of the literature it appears as an application layer protocol

Figure 2.7: Congestion control mechanisms improve efficiency in High-speed and/or Long-delay networks

Mechanism	Features	Solution for	Limitations
TCP-Santa Cruz (1999)	Path asymmetries, out-of-order packet delivery, and networks with lossy links, limited bandwidth and dynamic changes in delay.	High-speed, lossy network	Unfair for different RTT flows, low efficiency for high-speed and long-delay networks.
Fast-TCP (2000)	Flow control mechanism is located on the output of the access unit to the IP interface and controls ACK output rate according to congestion information from forward connection.	High-speed, lossy network	The flow cannot utilize the available network resources fairly.
HS-TCP (2003)	AIMD the congestion window size, limited slow start.	High-speed long-delay networks.	Fairness if flows have different RTTs.
STCP (2003)	MIMD scheme for congestion avoidance.	Data transfer effectiveness in high-speed/long-delay networks.	Due to MIMD policy flows are extremely unfair.
H-TCP (2004)	Congestion window increase steps as a function of time elapsed since the last packet loss detection, scaling increase step to a reference RTT, MD coefficient adaptation.	Fairness and effectiveness in high-speed/long-delay networks.	Prolonged recovery, waste resources.
TCP-Hybla (2004)	Scaling the increase steps in slow start and congestion avoidance to the reference RTT, data packet pacing, initial slow-start threshold estimation.	Throughput with NewReno in long-delay networks, RTT fairness in heterogeneous networks.	Like Reno unable to work effectively in high-speed networks with relatively small delay.
BIC-TCP (2004)	Binary congestion window search, limited slow start	RTT fairness of STCP and HS-TCP.	In certain environment have low RTT-fairness and inter-fairness values.

TCPW-A (2008)	Logarithmic congestion window increase.	Effectively utilize resources of high-speed or long-delay networks.	In some scenarios it loose its scalability like initial SS phase prematurely terminated will not able to rapidly discover new high-speed networks.
TCP-CUBIC (2008)	Window control based on cubic function of time elapsed from last congestion event.	Scales well in high-BDP networks and has good intra, inter and RTT fairness properties.	Network resource utilization is less than 100%, and more packet loss in the networks.
TCP-FAST (2003)	Constant-rate congestion window equation-based update.	Idea of congestion control with queuing delay for high-speed and long-delay networks.	Selection of α value is challenging task.
TCP-Libra (2005)	Adaptation of the packet pairs to estimate the bottleneck link capacity, scale the congestion window increase step by the bottleneck link capacity and queuing delay.	To resolve scalability while preserving and improving the RTT-fairness.	Not always outperform other congestion control approach, worsen further because of estimation biases.
TCP-NewVegas (2005)	Rapid window convergence, packet pacing, packet pairing.	Reduce convergence time and improve high-BDP link utilization, solution for generation of bursty traffic during initialization, bias estimation.	No scalable features are designed for congestion avoidance and fast recovery.
TCP-AR (2005)	Congestion window increase steps as a function of the achievable rate and queuing delay estimates.	To improve TCP performance and preserve friendliness in high-speed networks.	When queuing delay is wrongly estimated to be close to the max, then losing ability to scale in high-BDP network.
TCP-Fusion (2007)	Congestion window increase steps as a function of the achievable rate and queuing delay estimates.	Effectively utilize resources of high-speed or long-delay networks.	Manual fusion configuration is highly undesirable and usually impossible.
TCP-Africa (2005)	Switching between fast (HS-TCP) and slow (NewReno) mode based network state estimation.	Effectively utilize resources of high-BDP networks.	Not been implemented in real network.
QUIC (2013)	UDP based, Multiplexed, secure equivalent to TLS and connection migration, reliability and congestion control.	High-RTT/low-bandwidth networks.	Poor performance in high-speed networks.

average rate of flow completion. Figure 2.8 illustrate the behavior of flow level congestion control.

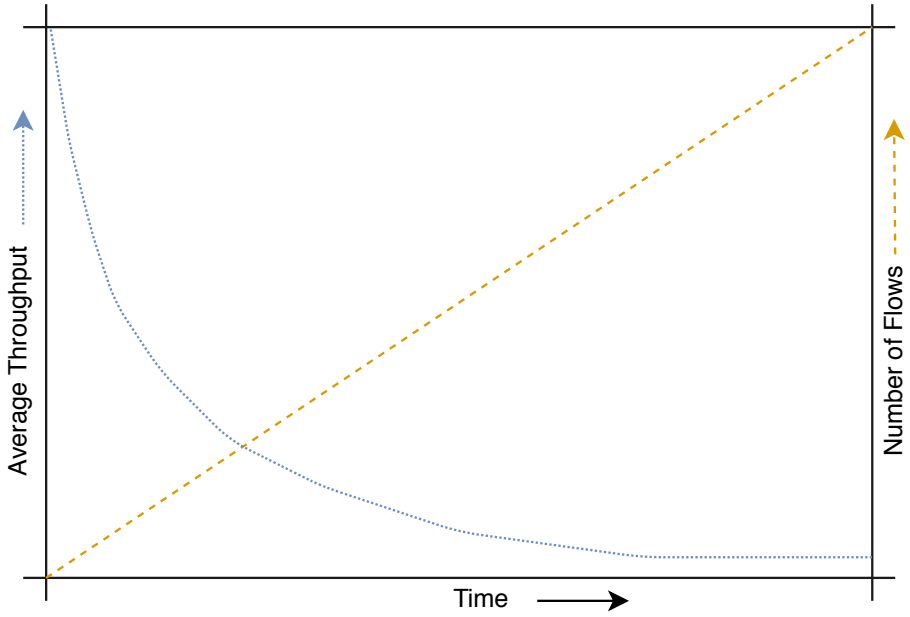


Figure 2.8: Network throughput performance for shared bandwidth during transient overload in terms of number of flows

There is limited literature available for flow level congestion control. However, it is worth to present survey on flow level congestion control as it is an important aspect in network performance and QoS. Barakat *et al.* (2003) designed traffic model for uncongested IP backbone links. They modeled traffic at flow level by using Poisson shot-noise process. This model shows very good approximation with respect to real backbone traffic. The applicability of this model is investigated with network dimensioning and provisioning, prediction of the total rate and generation of backbone traffic. Oueslati and Roberts (2005) proposed Flow Aware Networking (FAN) in this admission control [Kortebi *et al.* (2004)] and scheduling is applied on user defined flows. They claimed QoS and performance guarantee with cost effective solution as FAN. The FAN supports only for the cross-protect enabled routers, which improves performance very slow for network with flooded links.

A study of congestion at flow level using statstical bandwidth sharing is carried out by Fred *et al.* (2001), in which packet level simulations are used to extract the properties of TCP, whereas theory of stochastic network was proposed to trace the observations. They observed that if the sessions are appearing as per Poisson process then the throughput is insensitive to both the flow size distribution and the flow arrival process. In this author

has demonstrated that fluid flow statistical bandwidth sharing models can accurately predict the results of *ns* packet-level simulations. They claimed that, in fair network distribution, number of flows in progress and the expected flow have very simple relation which is valid for wide range real traffic. Sukhov *et al*. (2011) used Gaussian approximation to locate the working area of a link based on its utilization and vigilant operators indicating overload points. They used indicators to upgrade capacity, which avoids congestion at flow level. To validate hypothesis, they prepared testbed using border gateway routers and tested for wide range of link utilization.

2.3 Literature Survey on QUIC Protocol

The sufficient literature is available based on the transport layer performance analysis. But most of the attention is given to TCP [Claeys *et al.* (2016), Hassan and Jain (2003), Dong *et al.* (2015), Wang *et al.* (2014), Elkhatib *et al*. (2014)]. To improve further performance of the transport layer, Google proposed QUIC protocol. Before this, there was one attempt made by Vernersson (2015) with UDP based reliable transport.

The studies and developments on QUIC protocol are mostly available in the form of Internet drafts proposed by researchers from Google such as, secure connection with QUIC-Crypto [Langley *et al.* (2017)], loss recovery and congestion control [Iyengar and Swett (2015), Swett (2015)], QUIC contribution in Internet transport [Swett (2016)], QUIC used as test drive application [Gizis (2016)] and QUIC Internet draft for HTTP2 [Hamilton *et al*. (2016), Iyengar and Thomson (2017)].

As the proposed work needs an experimental study with comparative analysis of the ModQUIC protocol, this section presents survey on experimental investigation of QUIC protocol. In one of the experimental investigation, Google claimed 3% improvement with QUIC in mean PLT compared to TCP [Chromiumblog (2015)]. In Langley *et al.* (2017) Google claimed that QUIC reduces search latency by 8% and 3% for stable and mobile users respectively. However, it reduces buffering time by 18% for stable and 15.3% for mobile users. In addition to this, Google highlighted features like reduced HOL blocking, improved congestion control and loss recovery.

Other than Google, many researchers have explored QUIC performance for various scenarios. Carlucci *et al.* (2015), prepared two different experimental setups to investigate congestion control dynamics and web PLT. They observed goodput[2] performance of QUIC

[2]The ratio of the amount of application-level data to the total time taken until the completion of its delivery.

with FEC and noticed that for multiple objects in presence of loss, QUIC underperformed compared to TCP/HTTP. Megyesi *et al.* (2016) and Biswal and Gnawali (2016), tested QUIC performance in an emulated environment with QUIC enabled desktop client and Google server. In both studies, author observed TCP/HTTP outperform for high BDP and more number of large objects, but there are controversial comments on performance by both studies in presence of packet loss.

Das (2014), in his M.S. work evaluated QUIC performance in Mahimahi, a web performance measurement toolkit. He found QUIC performance was very good for low bandwidth and high RTT links. Cook *et al.* (2017), used local and remote testbeds and tested QUIC performance for, which scenario is most efficient. They have investigated QUIC performance with respect to the type of access network with packet loss and added delay. They concluded that QUIC outperforms over TCP/TLS with HTTP/2 in unstable networks such as wireless and mobile, but in case of a stable and reliable network there is no significant contribution.

Srivastava (2017), in his M.Sc. thesis has compared QUIC and TCP performance with respect to throughput, delay and fairness. He found that, for an added delay and loss, QUIC outperform and in case of competing flows QUIC was unfair compared to TCP. Kakhki *et al.* (2017), carried out extensive experimentation for a variety of network conditions. They found that QUIC outperforms TCP/HTTP in nearly every scenario, and QUIC is very sensitive with out of order delivery and shows very poor performance. They found that QUIC is unfair when competing with TCP flows.

Duan *et al.* (2017), proposed Intel DPDK based cQUIC: a scalable QUIC transport protocol implemented in Click. They have introduced a modular L2-L3 network stack. They claimed performance is improved over the Google QUIC server by analyzing real traffic. They observed that 18% of QUIC based connections are using 2-RTT handshake by limiting scalability. De Coninck and Bonaventure (2017), motivated by the success of Multipath-TCP (MPTCP) and proposed Multipath-QUIC (MPQUIC). They succeeded in enabling the QUIC connection to use different paths. They presented a comparative analysis of MPQUIC with MPTCP and observed that in lossy link scenarios, MPQUIC performance seen to be better than MPTCP. Hussein *et al.* (2018), integrated QUIC and Software Defined Networks (SDN) for improved resource utilization and network security. They implemented QUIC enabled SDN architecture to maximize bandwidth utilization and to secure data services.

Our work added new and extended contribution in terms of ModQUIC, a modified hand-

shaking mechanism to improve overall throughput and reduce congestion window update delay compared to others which have been summarized in Table 2.1.

Table 2.1: Comparative contribution analysis of QUIC protocol

Contributor	QUIC Version	Performance Analysis						Testing Environment	Added Contribution
		Throughput	Loss	Delay	Fairness	PLT	Link B/W		
Megyesi et al.	20	N	Y	N	Y	Y	N	Wired	N
Carlucci et al.	21	Y	Y	N	N	Y	Y	Wired	N
Das	23	N	N	N	N	Y	N	Wired	N
Biswal & Gnawali	23	Y	N	N	N	Y	Y	Wired	N
Cook et al.	25	N	Y	Y	N	Y	N	Wire/Wireless	N
Srivastva	25 to 36	Y	Y	Y	Y	N	Y	Wired	N
Kakhki et al.	25 to 37	N	Y	N	Y	Y	Y	Wire/Cellular	N
Duan et al.	Not Mentioned	Connection setup rate (c/s)						Wired	cQUIC
Coninck & Bonaventure	Not Mentioned	Y	Y	Y	N	N	N	Wired	MPQUIC
Hussein et al.	Not Mentioned	Security analysis					Y	Wire/Cellular	N
This work	**25 to 39**	**Y**	**Y**	**Y**	**Y**	**N**	**Y**	**Wired**	**ModQUIC**

2.4 Research Gaps

Research gaps were identified within literature survey and while working with existing experimental studies on TCP and QUIC protocol.

- To satisfy current customer demand of fast and reliable data delivery, there is a need to design reliable network with improved response time.
- Application based transport layer protocol, such as in HD video streaming packet loss rate is considerable.
- QUIC protocol may be one of the alternative to improve transport layer performance. QUIC protocol is built on top of UDP in which reduced handsahking mechanism is introduced.
- In the QUIC protocol window update is based on reception time analysis, which consumes more time to generate window update signal.
- The congestion control schemes used in transport layer are to be more efficient.

- Congestion control mechanism plays an important role to improve network.
- A dynamic data fragmentation based on available link bandwidth.
- There is a necessity of designing suitable network infrastructure based on available resources needs, which supports for connection migration.

2.5 Research Objectives

This is the world of ubiquitous and pervasive computing and communication, where there is a need to build congestion free network to experience high speed (low latency) data delivery.

Following are the objectives proposed to achieve prescribed goal.

1. To develop a modified handshaking mechanism to enhanced congestion control performance of the transport layer protocol considering QUIC as a base protocol.
2. To test the performance of developed protocol (ModQUIC) with respect to CUBIC and BBR congestion control mechanisms.
3. To test the developed protocol for heterogeneous network scenario, which may constitute combination of wired and wireless networks (A case study).

2.6 Summary

The data networks are scalable and accommodate plenty of users in wide spread. Internet users are generating huge data traffic, which makes network administrator responsible to furnish congestion free network services. Numerous congestion controlling proposals which were discussed in this survey are limited to standard, lossy network and high-speed & long-delay data network. The congestion control at flow level is another focused area of research, which is helpful to improve overall network performance and QoS. There is no universal solution available to address all issues, so we need to contribute based on situation. IoT is a technological development in which all devices (things) are connected via network, which provides scope to contribute towards congestion control. In this survey, large (elephant), medium (cat) and small (mice) size live traffic handling situations are not addressed separately. In most of the contributions, all were treated similarly, which may lead to HOL blocking. Fairness of the algorithm is measured with Jain's fairness index,

which is very simple control system model of n flows sharing the same link and receiving the same feedback signal. It can well describe the static properties of competing flows. However, the characteristics of new network architectures and environments were dynamic in which heterogeneity is more if new or different traffic characteristic cannot be well captured in all aspects by that model. Bufferbloat problem is not completely solved in which processing delay is more than the connection establishment time. In bufferbloat, queuing delay is more than RTT and packets get delayed shows congestion in the network. To solve this an active and passive queue management techniques are suggested. The standard buffer size suggested is equal to BDP but in long-delay networks this is not the feasible solution. QUIC is the alternate solution to the TCP, which we decided to use as a base protocol. Further, at the end we confer about few research gaps and based on that research objectives are set.

Chapter 3

Modified Handshaking Mechanism using QUIC Protocol

3.1 Introduction

Today's fast paced world demands data transfer with almost zero latency. Online business is an example wherein vendor's website suffers high PLT, which results in poor customer satisfaction. TCP/IP is the most popularly used protocol to access Internet data using wired or wireless technology. To maximize the customer satisfaction in traditional information gathering or online services, it is necessary to monitor TCP/IP network parameters to improve performance [Grigorik (2018), Hassan and Jain (2003)]. TCP/IP has contributed extensively to the networking industry by delivering remarkable results. Google has a detailed analysis of TCP/IP performance, where 30% to 35% of all Internet traffic passes through its servers. In addition to this, Google's Chrome is the most popular browser with approximately 40% of market share [Flach *et al.* (2016)]. Based on the experience in 2013, Google developed a new protocol using UDP, known as Quick UDP Internet Connections protocol. QUIC protocol is simple and is present on the top of UDP in protocol stack, thereby making it suitable for supporting application protocols for booting (Figure 1.5).

As per the estimation of the Cisco Visual Networking Index, Annual Global IP traffic will hit 3.3 ZB by 2021. The report states that up to 83% of consumers IP traffic is consumed by streaming video and 13% by live streaming [Cisco (2018)]. In this scenario, it becomes even more relevant to establish networking protocols to handle the rapid growth in IP traffic usage. The popular HTTP published with RFC2616 in 1999 has become outdated with the changes in Internet usage and data traffic. Google has also developed SPDY web transfer

protocol using multiplexed TCP streams [Elkhatib *et al*. (2014)], consisting of a header compression technique on top of HTTP/1.1 to improve PLT.

[1]This chapter deals with the performance verification of QUIC protocol and contributes to the modification of existing handshaking mechanism. The modified QUIC protocol is renamed as ModQUIC, which is a transport layer solution to improve network performance by reducing both control overhead and window update delay. In ModQUIC, window update frame is attached with ACK frame instead of being sent separately. This modification avoids the control signal required to trigger window update, which helps in reducing transport latency along with fine-tuning of *cwnd* growth with every ACK. This results in smooth variation in *cwnd* growth, which prevents consecutive packet dropping and consequently reduces timeout events. Hence, improvement in throughput and reduction in transport latency are achieved. A CUBIC congestion control mechanism is fair to manage resources in multiple flow scenario. [2]To evaluate the performance of ModQUIC, two testbed setups were prepared using OpenFlow Mininet platform and Chromium server-client model. Test cases were created by varying load in terms of number of packets, bandwidth and loss rate. Comparative analysis shows that in critical situations, ModQUIC performance is very good in terms of throughput and speedup (by a reduction in delay) over QUIC, TCP and TCP/HTTP2 with maintained fairness property.

3.2 Congestion Window Growth Analysis

This section deals with congestion window growth analysis with the aid of mathematical illustration.

3.2.1 Mathematical Illustration

To determine the flow rate of a node, the probabilities of both collision and number of packets in the system are calculated. To model the system, birth-death process is used to handle arrival and departure rates [Ross *et al.* (2016)]. However, to analyze system behavior, the Poisson Distribution Function is used. The queuing model based on pure birth-death process is shown in Figure 3.1.

[1]Prashant Kharat and Muralidhar Kulkarni (March, 2019), "Modified QUIC protocol for improved network performance and comparison with QUIC and TCP", *International Journal of Internet Protocol Technology, Vol. 12, No. 1, pp. 35-43*

[2]Prashant Kharat and Muralidhar Kulkarni (under review), "ModQUIC: A Modified QUIC Protocol to Improve Network Performance", *IEEE/ACM Transactions on Networking*.

Let n is the number of packets in the network, λ_n is the arrival rate, μ_n is the departure rate and P_n is the steady state probability of n packets, where P_n is a function of λ_n and μ_n. These variables are used to determine system performance. Under steady state condition (i.e. $n > 0$), the expected flow rate into and out of the state are equal.
If $(n-1)$ to $(n+1)$ is the change in the state then,

$$\text{Expected rate } R_e \text{ of flow into state } n = \lambda_{n-1}.P_{n-1} + \mu_{n+1}.P_{n+1} \tag{3.1}$$

$$\text{Actual rate } R_a \text{ of flow out of state } n = (\lambda_n + \mu_n).P_n \tag{3.2}$$

To achieve maximum matching for expected rates with assumption of steady state, RHSs of equations (3.1) and (3.2) are equated. Hence,

$$\lambda_{n-1}.P_{n-1} + \mu_{n+1}.P_{n+1} = (\lambda_n + \mu_n).P_n \tag{3.3}$$

Using state transition diagram (Figure 3.1) for $n = 0, 1, 2, ...$

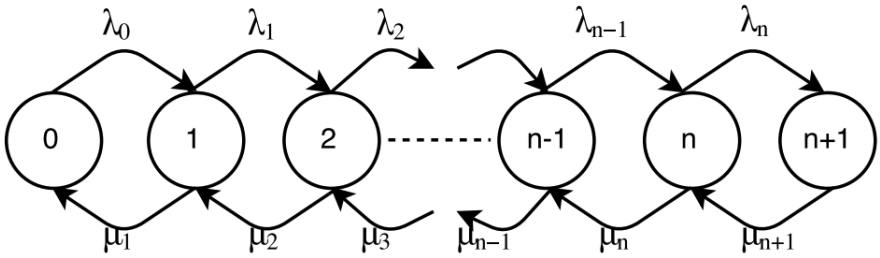


Figure 3.1: Poisson queue transition state

For $n = 0$, equation (3.3) becomes

$$\lambda_{0-1}.P_{0-1} + \mu_1.P_1 = (\lambda_0 + \mu_0).P_0 \tag{3.4}$$

As, (-1) state is absent, λ_{0-1}, P_{0-1} and μ_0 will be zero. Therefore, equation (3.4) becomes,

$$\mu_1.P_1 = \lambda_0.P_0 \tag{3.5}$$

therefore,

$$P_1 = \frac{\lambda_0}{\mu_1}.P_0 \tag{3.6}$$

for $n = 1$,

$$\lambda_0.P_0 + \mu_2.P_2 = (\lambda_1 + \mu_1).P_1 \tag{3.7}$$

using P_1 from equation (3.6) in equation (3.7),

$$P_2 = \frac{(\lambda_1.\lambda_0)}{(\mu_2.\mu_1)}.P_0 \tag{3.8}$$

hence,

$$P_n = \frac{(\lambda_{n-1}.\lambda_{n-2}.....\lambda_0)}{(\mu_n.\mu_{n-1}.....\mu_1)}.P_0 \tag{3.9}$$

for $n = 0, 1, 2.....$

$$\sum_{n=0}^{\infty} P(n) = 1 \tag{3.10}$$

To estimate congestion, packet emission probability based on the previous state is used. Packet emission probability is calculated based on window update information sent by receiver node. To control packet transmission rate, three state window update strategy is used at the source end.
Let the window update information be W_u :

$$W_u = 0, \text{ if } P_n = P_{n-1} \tag{3.11a}$$

$$W_u = 1, \text{ if } P_n > P_{n-1} \tag{3.11b}$$

$$W_u = -1, \text{ if } P_n < P_{n-1} \tag{3.11c}$$

The above conditions lead to the following observations or window update strategies:

1. By decreasing the packet size, the number of packets in the network are increased to send the same amount of data. This will result in increase in the probability of collision, and hence reduction in window size, leading to control congestion.

2. The packet size depends on the available bandwidth (allowed rate to send the data), which ultimately depends on the window size.

3. The window update information contains state to decrease the window size and to control the rate if congestion is detected.

4. If flow control is achieved, packet size simultaneously decreases which will lead to an increment in number of packets as per the state-1, to insure that the probability of packets emitted is less, the rate has to be controlled.

5. The outcome of this approach is to control the congestion, which results in an improvement of the throughput performance and reduction in the transport latency.

3.3 Proposed Handshaking Mechanism

ModQUIC is a modification in the existing handshaking mechanism of QUIC protocol. In this modification, as shown in Figure 3.4, window update frame is attached with ACK frame. Initially, ModQUIC establishes a connection between server and client by sending QUIC-Crypto request message. In the initial handshake, both server and clients negotiate in *cwnd* size. This modification helps to reduce control overhead and packet transmission delay which improves the overall throughput and reduce the latency in the network. The size of *cwnd* varies with respect to cubic function given in equation (2.1).

3.3.1 QUIC-ACK Frame Structure

The ACK frame is used by the receiver to inform the server about received and missing packets. The structure of QUIC-ACK frame is different from TCP-ACK frame as shown in Figure 3.2 [Hamilton *et al*. (2016)]. In QUIC, NACK indicates gaps in the received packets, which in turn are detected as missing packets. The server periodically sends Stop-waiting frames to inform the receiver to stop waiting for packets below a particular sequence number. The fields in ACK frame structure are as follows:

- Frame Type (8 bit): This field contains '01ntllmmB' flags. In which first two bits are set to '01' indicates that this is an ACK frame, 'n' bit shows presence of NACK ranges, 't' bit shows truncated ACK frame; truncation can happen when the complete ACK frame does not fit within a single QUIC Packet, or when the number of NACK ranges exceeds the maximum range (255). The bits 'll' and 'mm' encodes the length of the Largest Observed field and Missing Packet Sequence Number Delta field as 1, 2, 4, or 6 bytes long.

- Received Entropy (8 bit): Unsigned integer specifies the cumulative hash of entropy in all received packets.

- Largest Observed (variable): Unsigned integer represents the largest packet number the peer has observed.

- ACK Delay Time (16 bit): Unsigned float, in which 11 bits are used for mantissa and 5 bits for exponent, which specifies the time elapsed in microseconds from when largest observed was received until this ACK frame was sent.

- Timestamp Section: Num Timestamp (8 bit): Unsigned integer indicates the number of timestamps that are included in this ack frame. Delta Largest Observed (8 bit): Unsigned integer specifies the packet number delta from the first timestamp to the largest observed. First Timestamp (32 bit): Unsigned integer specified by Largest Observed minus Delta Largest Observed. Time Since Previous Timestamp (16 bit): Unsigned integer specifies delta from the previous timestamp.

- Missing Packet Section: Num Ranges (8 bit): An optional field indicates the number of missing packet ranges between largest observed and least unacked. Missing Packet Sequence Number Delta (variable): This delta value indicates the number of packets received between successive NACK ranges. Range Length (8 bit): Unsigned integer specifies one less than the number of sequential NACKs in the range.

- Revived Packet Section: Num Revived (8 bit): Unsigned integer specifies the number of recovered packets. Revived Packet Sequence Number (variable): Unsigned integer representing a packet the peer has recovered.

3.3.2 QUIC Window Update Frame Structure

The window update frame, as shown in Figure 3.3, has been used to inform the peer of an increase in end point's flow control receive window [Hamilton *et al*. (2016)]. Window update can be applied for connection level or stream level flow control. Violating flow control by sending more bytes than the prescribed limit will result in the closure of the connection by receiving endpoint. The default window size available in QUIC is 16 KB, which gradually increases during handshaking by exchanging window control parameters.

Following are the fields of window update frame:

- Frame Type (8 bit): This field must be set to 0X004 to indicate a window update frame.

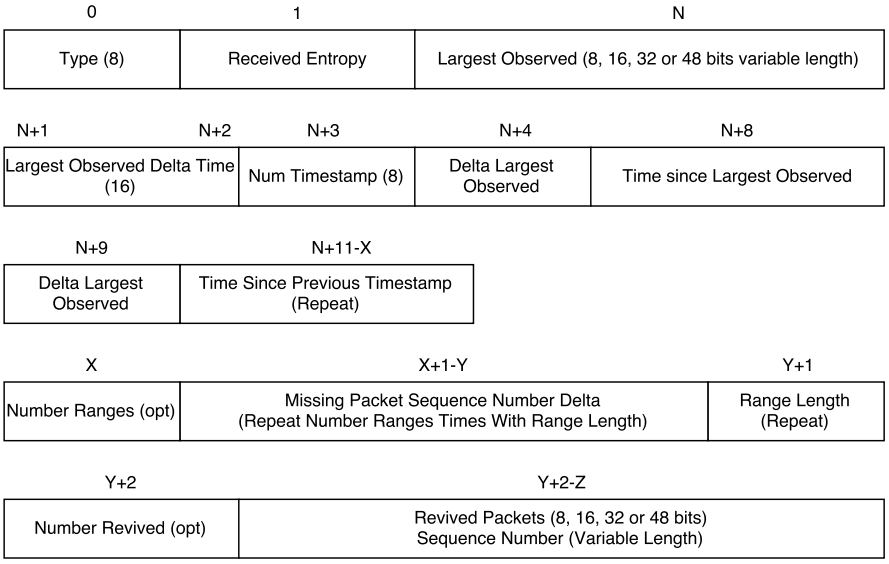


Figure 3.2: ACK frame structure

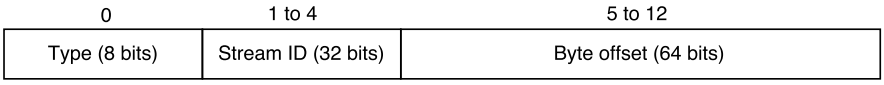


Figure 3.3: Window update frame structure

- Stream ID (32 bit): Integer value greater than 0 in this field indicates the stream ID whose flow control window is being updated, otherwise it indicates connection-level flow control.

- Byte offset (64 bit): An unsigned integer used to represent maximum amount of data can be sent on the open stream. However, in connection level flow control stream, cumulative data bytes from all open streams are to be considered.

3.3.3 Proposed ACK Frame Structure for ModQUIC

A proposed ACK frame structure is shown in Figure 3.4. As per the working mechanism of QUIC, there is a specific sequence of work-flow in which, window update state appears after packet gets acknowledged. In execution of work-flow, ACK and window update frames arrive separately one after another. The window update has been carried out based on analysis of ACK reception time. In the proposed scheme, window update frame is attached to the ACK frame instead of being sent separately. This modification reduces control overhead and window update delay, which results in improvement of packet transmission. In the pro-

posed structure, maximum size of the window update is 32 bytes, which is fixed, but can be made adaptive according to the network condition and application demand.

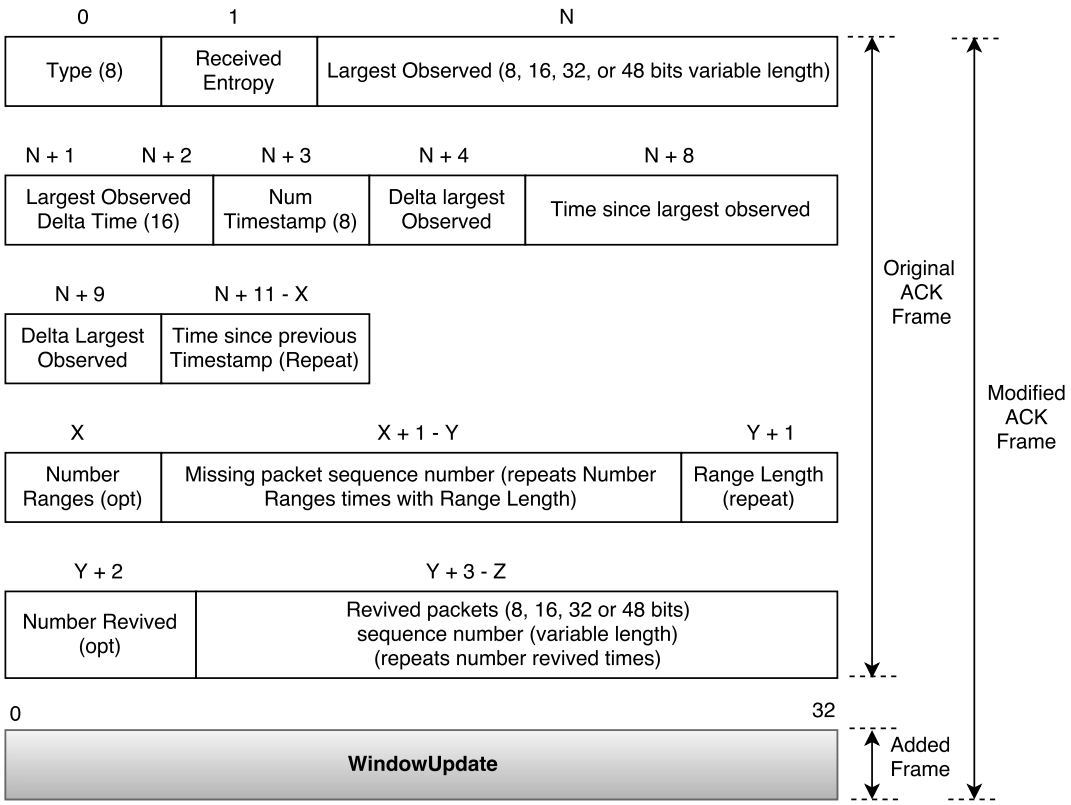


Figure 3.4: Proposed frame structure

3.3.4 Proposed Handshaking Mechanism

In line with the proposed frame structure, a modified handshaking mechanism for initial and repeated connection is shown in Figure 3.5.

Following steps are the flow of execution for the proposed handshaking mechanism.

- After creating QUIC based server-client model, first connection establishment process is carried out. It takes 0 or 1 RTT based on initial or repeated connection request.
- On receiving a data packet, the receiver sends ACK to the sender. In case the receiver is disconnected, the sender stops receiving the ACK and assumes that the receiver is temporarily disconnected. Then sender controls the rate of transmission and freezes timers by sending back-off persist packet to the receiver till it receives ACK. As soon

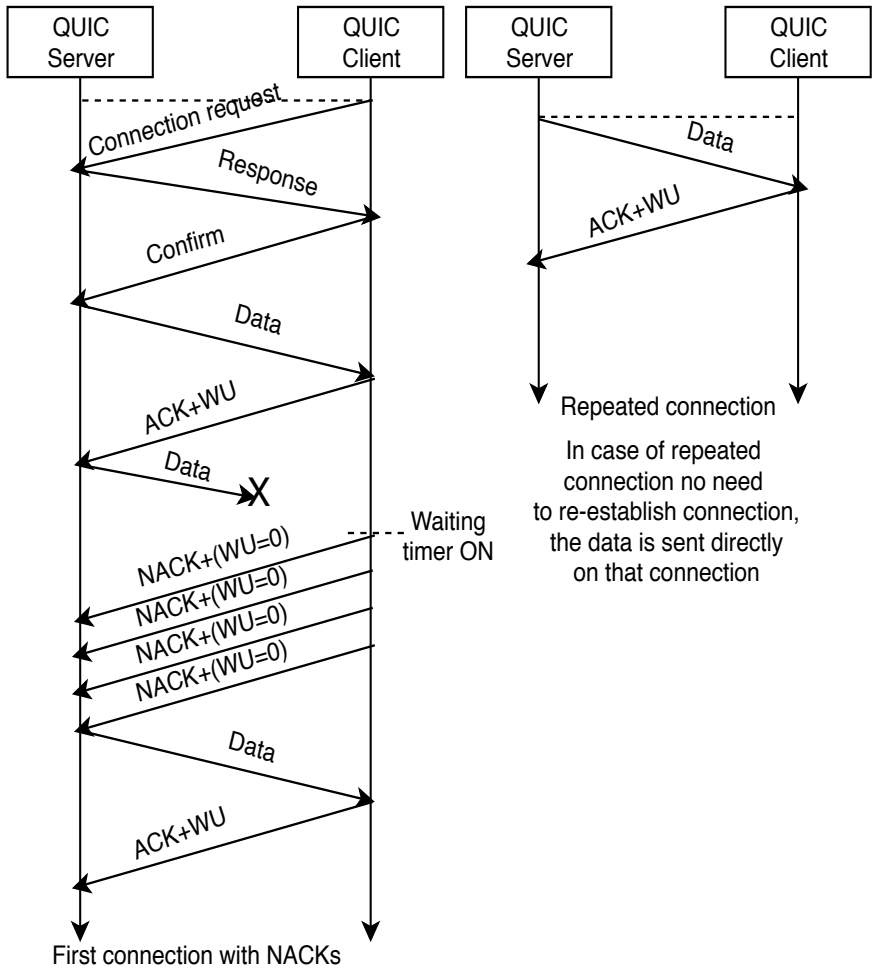


Figure 3.5: Proposed handshaking mechanism

as the congestion is under control, it restarts frozen timers. This results in transmission with full rate by the sender.

- If, sent data is lost, the sender re-sends data and updates window size by sending ACK + window update.

- If the same situation is repeated, the receiver sends ACK (with NACK) to the sender continuously with zero window update until congestion is under control. Once congestion is under control, receiver updates the window size to sender and resends the lost data to the receiver.

3.4 Performance of Modified Handshaking Mechanism

The performance investigation of the modified handshaking mechanism using QUIC as a base protocol is discussed in this section. The two different environments and testbed sets were developed to test the performance of the ModQUIC protocol.

3.4.1 Evaluation Metrics

Performance evaluation of ModQUIC, QUIC and TCP is carried out with respect to throughput, delay and fairness as given in section 1.4 for different bandwidth and loss rate.

3.4.2 Experiment Setup using QUIC Server-client Model

3.4.2.1 Server-client configuration

Performance of the suggested modification is tested by using the dummy QUIC server-client model present in the Chromium browser code-base available at https://code.google.com/p/chromium/. Figure 3.6 shows testbed environment with QUIC version 33. The data generation is performed by using Google certified www.example.org. For this investigation, TCP server application with TCP-CUBIC functionality is used. QUIC source code is modified to add ModQUIC functionality and logged relevant variables. On client side, three different configurations ModQUIC enabled, QUIC enabled and TCP enabled (QUIC disabled) of Chromium are deployed. Experimentation is carried out multiple times by using loopback technique to increase the traffic and create logs of almost ten thousand packets with payload size of 270 bytes.

3.4.2.2 System configuration

Intel® Core^{TM} i5-2400 CPU @ 3.10 GHz X 4, 8 GB RAM, Ubuntu 14.04 LTS, 64 bit operating system. Chrome version 63.0.3239.132. Linux Kernel Version 4.4.0-93 generic.

3.4.2.3 Internet connectivity

1.9 Gbps across 4 lease lines with (2:1:1) load balancing with core switch 2X VDX8770-8, 1 X VDX6740-T and router configuration of 2 X 2 MIMO, 433Mbps/client on 5 GHz,

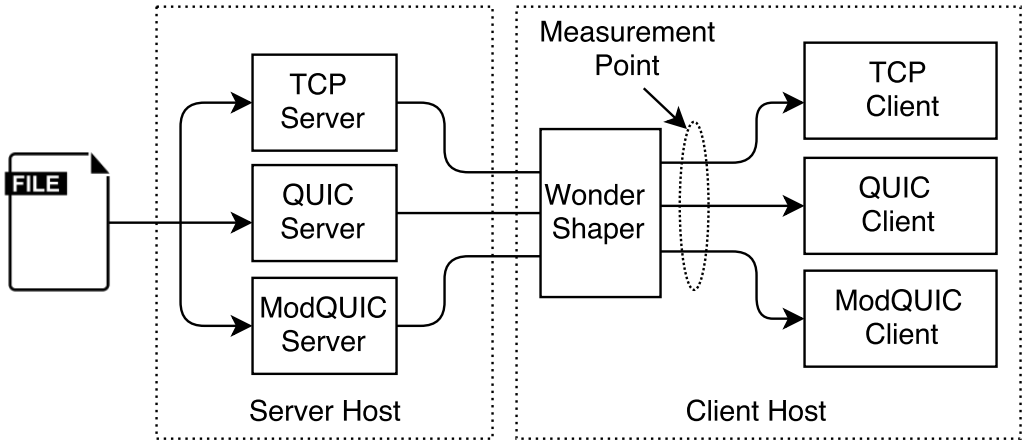


Figure 3.6: Testbed environment

72Mbps/client on 2.4 GHz.

3.4.2.4 Analysis and traffic shaping tools

iPerf, a network performance measurement tool, is run on the client machine to measure an amount of data transferred and bandwidth available for server-client. The wondershaper tool deployed on the client is used for managing traffic, to manipulate bandwidth, to fix packet loss and to set the propagation delay.

3.4.2.5 Result analysis

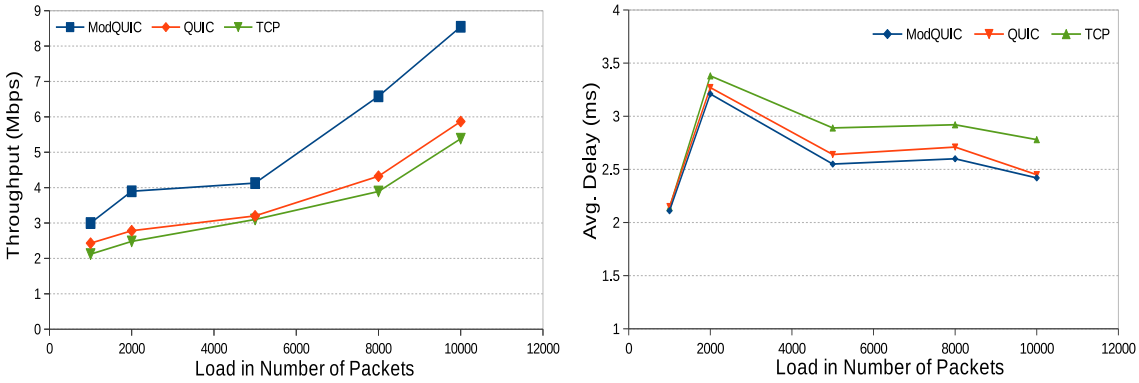
The results are presented in the form of comparative performance analysis of ModQUIC, QUIC and TCP protocols. In the experiment, base RTT value considered is 20 ms and queue size is equal to BDP with drop-tail policy.

In QUIC, congestion is detected based on analysis of ACK reception time and window update sent from the client only when there is a congestion. However, in ModQUIC, every ACK is associated with window update instead of based on ACK reception time, which provides sufficient ground to increase window size to specified level. The results observed are given in the Table 3.1/Figure 3.7a, which shows average throughput improvement of 39.10% over QUIC and 51.93% over TCP.

In ACK reception time analysis, if new ACK reception time is greater than previous ACK reception time, then there is no window update. This is the threat in updating con-

Table 3.1: Throughput and Delay performance comparison

No. of Packets	Throughput (Mbps)					Average Delay (ms)				
	ModQUIC	QUIC	TCP	%Improvement		ModQUIC	QUIC	TCP	%Improvement	
				QUIC	TCP				QUIC	TCP
1000	2.99	2.43	2.12	23.41	41.46	2.11	2.15	2.14	1.86	1.40
2000	3.89	2.78	2.48	40.09	57.09	3.21	3.27	3.38	1.83	5.09
5000	4.13	3.20	3.10	29.51	33.16	2.55	2.64	2.89	3.40	11.76
8000	6.58	4.32	3.98	52.26	69.18	2.60	2.71	2.92	4.06	10.96
10000	8.54	5.87	5.38	50.24	58.75	2.42	2.45	2.78	1.22	12.94



(a) Throughput **(b)** Delay

Figure 3.7: Throughput and Delay performance comparison

gestion window size in QUIC. Further, if QUIC is sporadic, additional time is required to analyze the ACK reception time and to send window update frame. In proposed mechanism, this threat and window update delay is abolished by attaching window update frame to ACK frame. This improves the overall performance of ModQUIC compared to QUIC and TCP resulting in maximum bandwidth utilization.

Updating window size dynamically for every ACK reception ensures delivery of data within optimum time resulting in bandwidth optimization. This reduces number of retransmissions, as minimal loss of data is observed. The data loss check reduce on redundancy in packet delivery, which results into more number of packets delivered within time compared to existing system. By this way average end to end packet transmission delay is reduced as seen in Figure 3.7b.

Loss	ACKs Generated (%)		
Rate (%)	ModQUIC	QUIC	TCP
0	91.40	89.30	98.23
1	90.32	88.67	93.57
2	85.12	82.34	85.36
5	97.76	93.82	85.47
8	99.56	98.26	81.88
10	97.98	96.71	80.19

Table 3.2: Performance with loss

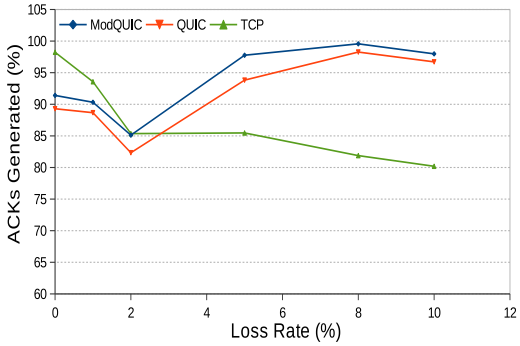


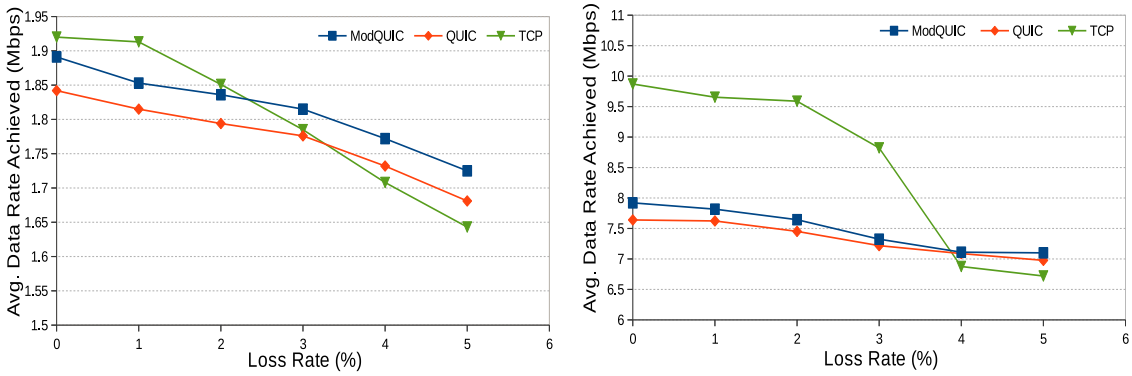
Figure 3.8: Performance with loss

In ModQUIC and QUIC, a loss based congestion control mechanism, CUBIC, is employed to serve for congestion. In which, response to loss is a cubic function, which is slowly but exponentially growing. This results in improving performance of ModQUIC and QUIC in the presence of loss compared to TCP. However, it is observed that greater bandwidth has been occupied by ModQUIC and QUIC compared to TCP, which is unfair. The percentage of ACK generated signifies that, higher the percentage of ACKs generated, more are the number of packets received. Validation of above mentioned analysis has been presented in Table 3.2/Figure 3.8. When the loss is 0%, TCP outperforms due to it's initial aggressiveness, whereas ModQUIC and QUIC performance is almost similar. The performance of TCP gradually decreases with increase in loss rate, whereas in ModQUIC and QUIC, due to multiplexed streams and out of order delivery, even for lossy links, the performance is superior. The performance of ModQUIC is better than QUIC due to bandwidth occupancy limitation, which in turn depends upon the window size used. Even though slow start is avoided in QUIC, its default window size is updated only with reference to an analysis of previously sent packet's success and their rate of transmission. This improves reception of ACKs and the graph shows that ModQUIC outperforms QUIC and TCP. In ModQUIC, maximum bandwidth utilization is observed due to successful reception of ACK which is responsible for window update.

Table 3.3/Figure 3.9a shows that for lossless bottleneck link of 2 Mbps, all three flows are closely competing with each other, whereas as loss rate increases link becomes congestive and on every ACK, as a reaction, TCP reduces data rate gradually. But for ModQUIC and QUIC, base protocol is UDP. Hence their performance is better. However, for ideal condition, such as sufficient bandwidth (10 Mbps) and lossless link, TCP is dominating, as packet pacing becomes overhead for ModQUIC and QUIC. Once loss rate increases, TCP performance suddenly drops down below ModQUIC and QUIC as observed in Table

Table 3.3: Data rate achieved for 2 Mbps and 10 Mbps link with loss

Loss Rate (%)	Average Data Rate (Mbps)					
	2 Mbps Link			10 Mbps Link		
	ModQUIC	QUIC	TCP	ModQUIC	QUIC	TCP
0	1.891	1.842	1.920	7.920	7.640	9.870
1	1.853	1.815	1.913	7.817	7.623	9.654
2	1.836	1.794	1.851	7.644	7.452	9.588
3	1.815	1.776	1.785	7.324	7.218	8.822
4	1.772	1.732	1.708	7.111	7.088	6.876
5	1.725	1.681	1.643	7.100	6.977	6.720



(a) 2 Mbps link **(b)** 10 Mbps link

Figure 3.9: Data rate achieved for 2 Mbps and 10 Mbps link with loss

3.3/Figure 3.9b.

3.4.3 ModQUIC Performance with One-hop and Browser Network

To verify the performance of ModQUIC protocol, this section presents an experimental setup in which client is one hop away from the server. In this experimental investigation, live data with higher video file sizes and bandwidth varying from limited to sufficient are used.

3.4.3.1 Window size update algorithm

A strategy suggested to update window size is given in Section 3.2.1 by using Algorithm-2. The available bandwidth estimation is carried out with the help of a MAC layer. If the present estimated bandwidth is less than the previous bandwidth, the window size needs

to be reduced with a step size of ST. If present bandwidth is greater than the previous bandwidth, window size is increased by step size of ST, where default step size ST is equal to 1. The data rate is automatically adjusted according to the updated window size. This fine-tuned mechanism along with ACK frame results in smooth variation of *cwnd* size and shows stable system performance.

Algorithm 2 : Window Update Algorithm

Input: $Packet_{rate}, Bandwidth, WindowSize$
  if $Bandwidth_{new} < Bandwidth_{old}$
  $WindowSize_{new} = WindowSize_{old} - ST$
  if $Bandwidth_{new} > Bandwidth_{old}$
  $WindowSize_{new} = WindowSize_{old} + ST$
  if $Bandwidth_{new} > Bandwidth_{old}$
  $WindowSize_{new} = WindowSize_{old}$
  if $WindowSize_{old} == WindowSize_{new}$
  $Packet_{ratenew} = Packet_{rateold}$
  if $WindowSize_{old} < WindowSize_{new}$
  $Packet_{ratenew} = Packet_{rateold} + ST$
  if $WindowSize_{old} == WindowSize_{new}$
  $Packet_{ratenew} = Packet_{rateold} - ST$
  end

3.4.3.2 Proposed handshaking mechanism for one-hop network

According to the suggested modification in the ACK frame structure, the modified handshaking flow is shown in Figure 3.10.

- Create a QUIC based server-client model. To establish a connection for the first time, it takes single RTT, whereas for repeated connection zero-RTT (refer Figure 1.6b).

- Once link has been established, a sender (SH) sends a data packet to the receiver (RH). When the RH receives a data packet, it sends ACK to the SH. In case the RH is disconnected or if timeout takes place, the SH stops receiving the ACK with an assumption that RH is temporarily disconnected. Therefore SH controls the rate of transmission and freezes its timers. For this period, SH sends back-off persist packet to the RH until it receives ACK. As soon as the congestion is under control, it restarts frozen timers and starts transmitting packets with full rate.

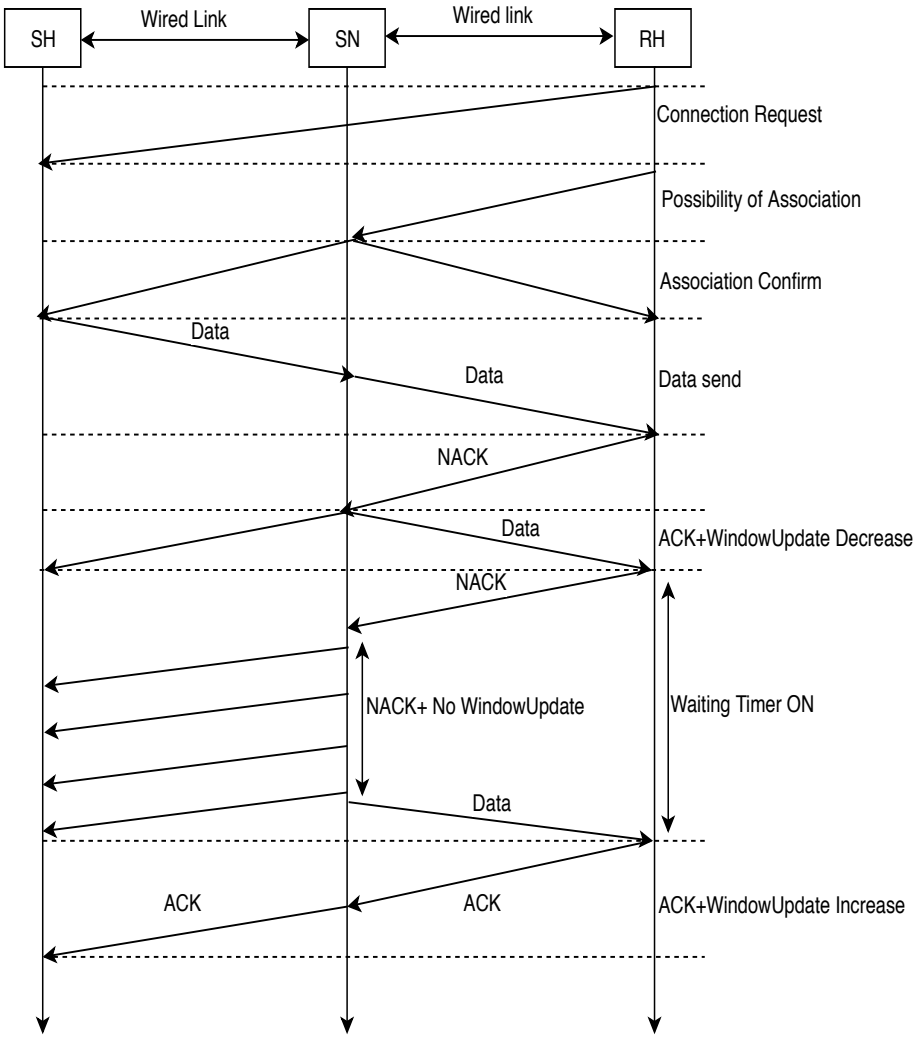


Figure 3.10: Proposed handshaking mechanism

- The rate control mechanism can be achieved by an indication of window size in the ACK frame. If data sent by the router (SN) is lost, the SN re-sends data and update window size towards SH by sending ACK+Window-update.

- If the same situation is observed repeatedly, it will result in congestion notification. The RH sends ACK (with NACK) to SH continuously with zero window size (no window update) and waits until congestion is under control. Once congestion is under control, it updates the window size to the SH and resends the lost data to the RH.

3.4.4 Performance verification with one-hop network

ModQUIC performance is verified with respect to evaluation metrics such as throughput, delay, speedup and fairness by creating testbed environment.

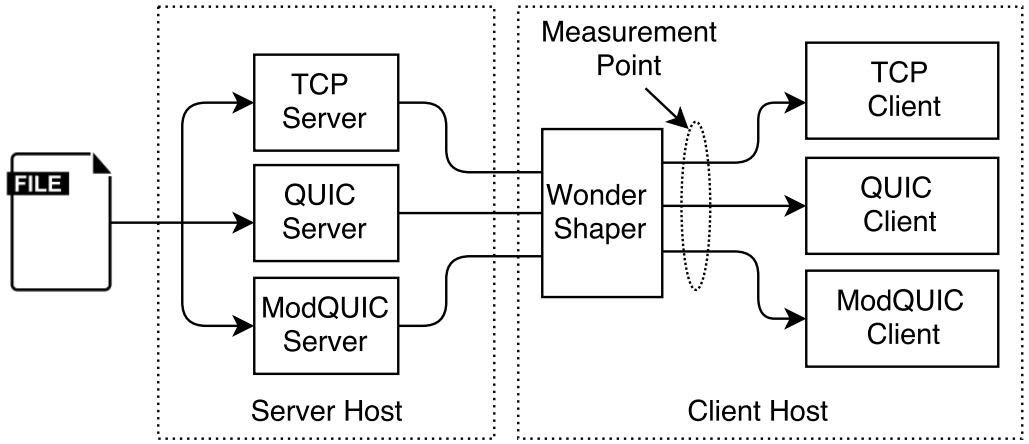
3.4.4.1 Parameter space and testbed environment

The testbed environment and assembled setup are shown in Figure 3.11a and 3.12a which are utilized to carry out the experimental evaluation of ModQUIC, QUIC, and TCP performance. However, Figure 3.11b and 3.12b are used to carry out evaluation of ModQUIC and TCP/HTTP2 for browser network.

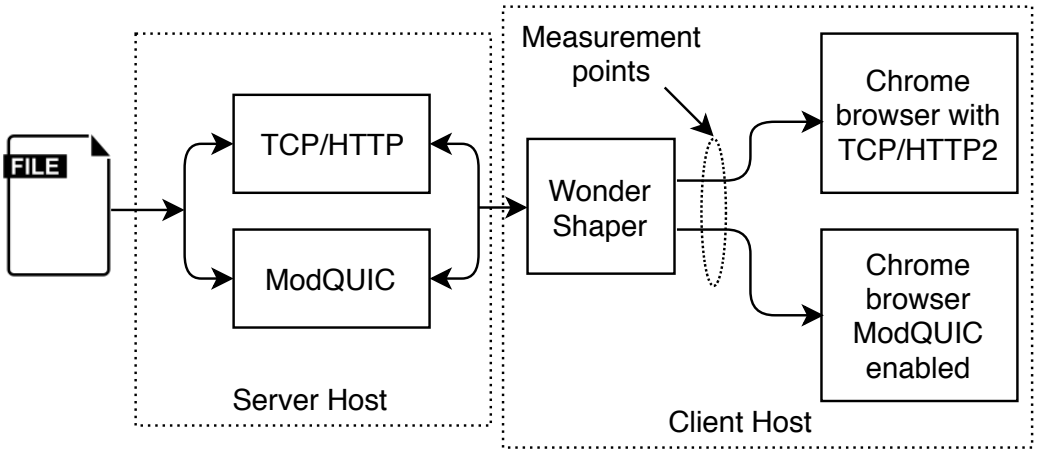
The testbed shown in Figure 3.12a has been set up using OpenFlow Mininet platform, which creates a network of virtual hosts. Mininet hosts run standard Linux network software and support OpenFlow for highly flexible custom routing and software defined networking. The libquic library with 'golang' programming language created by Google is used as platform package to analyze the performance. The results are obtained by libquic analysis script written in python. The QUIC toy-server-client program is used to analyze the performance. The client tries to establish a connection with the server for FTP via the external host machine. To perform experimentation, hosts are at one hop distance away from server. The performance is observed by changing the loss rate and link bandwidth given in Table 3.4 using wondershaper, a traffic shaping tool.

The testbed shown in Figure 3.12b has been set up with a dummy QUIC server-client model available in the Chromium browser code-base (https://code. google.com/p/chromium/). For the experimentation TCP server application with TCP-CUBIC functionality is used with a modified source code of QUIC to add ModQUIC functionality and logged relevant variables. On the client side, two different configurations of Chromium, ModQUIC and TCP are enabled (QUIC disabled) and deployed. This implementation of a ModQUIC server has been used to test results from a MacBook Pro laptop (running chromium browser built from source) which has been used to carry out measurements[1]. It has been noted that implementation is meant for integration testing and not to test performance at the scale.

[1]Intel Core i5 CPU @2.5 GHz with 4 GB RAM running MacOS Sierra 10.12.6 (64 bit). With chromium version of 63.0.3229.0 (64 bit) and Darwin kernel version 16.7.0.



(a) Transport layer environment

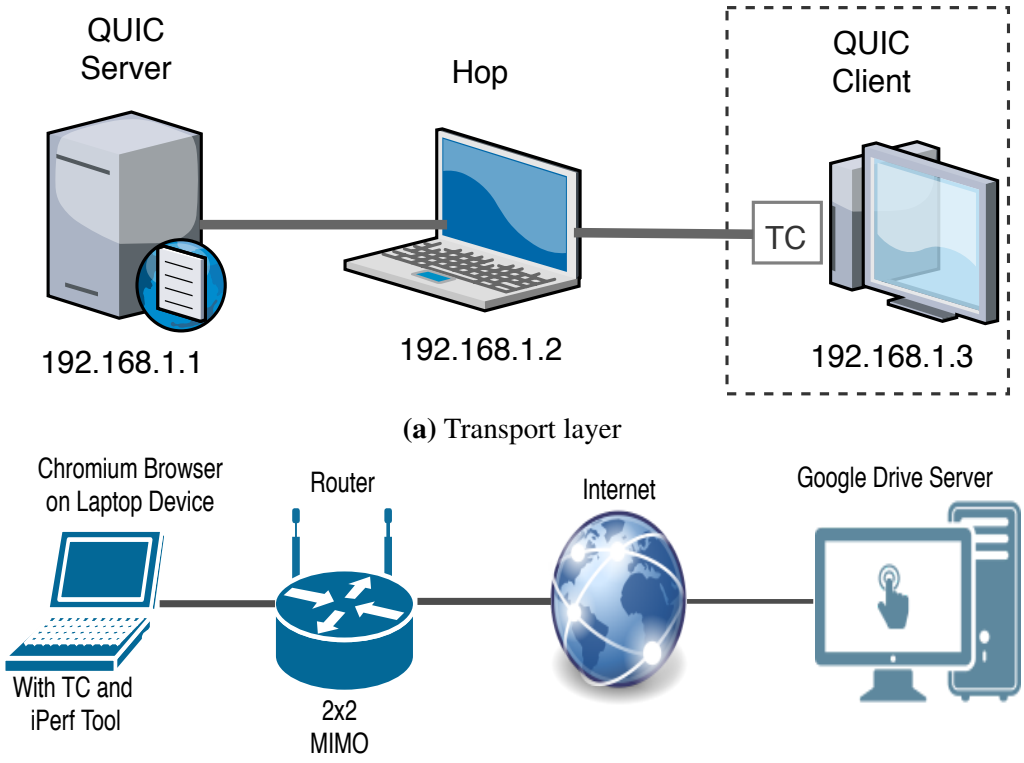


(b) Browser network environment

Figure 3.11: Testbed environment

Table 3.4: Parameter space used for experimentation

Category	Parameters	Values
Network parameters	Bandwidth	5 Mbps, 10 Mbps, 50 Mbps
	Link capacity	72Mbps/client for 2.4 GHz and 433Mbps/client for 5 GHz
	RTT	20 ms, 50 ms
	Packet loss	0%, 2%, 5%, 10%
Server side	Video file size	10 MB, 30 MB, 50 MB
	No. of Clients	3 (with virtual node), 3 (with browser)



(a) Transport layer

(b) Browser network

Figure 3.12: Assembled testbed

3.4.4.2 System configuration

- Server-client configuration: Intel Core^{TM} i5-2400 CPU @ 3.10 GHz X 4, 8 GB RAM, Ubuntu 14.04 LTS, 64 bit operating system.
- Hop configuration: Intel Core^{TM} i5 CPU M560@2.67GHz X 4, 4 GB RAM, Ubuntu 14.04 LTS, 64 bit operating system.
- Internet connectivity: 1.9 Gbps across 4 lease lines with (2:1:1) load balancing with core switch 2 X VDX8770-8, 1 X VDX6740-T and router configuration of 2 X 2 MIMO, 433Mbps/client on 5 GHz, 72Mbps/client on 2.4 GHz.

3.4.4.3 Analysis and traffic shaping tools

iPerf tool running on a client machine is used to measure link bandwidth available between server and client. A wondershaper tool is used to manage traffic, to fix packet loss and to

allow propagation delays to be set on a client machine. It can also be used for setting up packet loss rate, while downloading data in the browser in order to create loss effect.

3.4.4.4 Result analysis

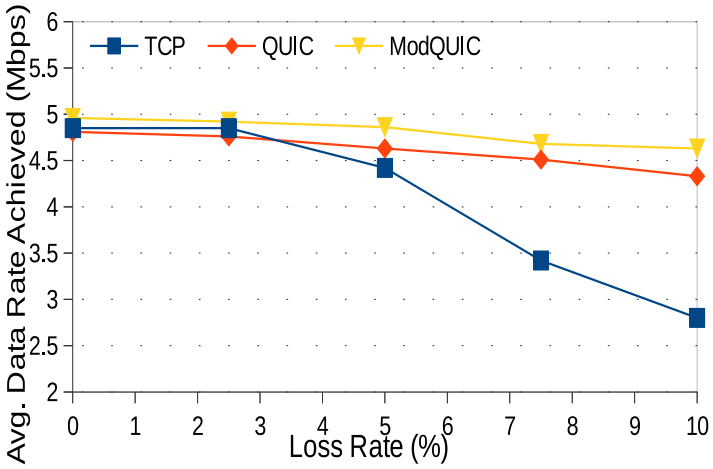
The comparative performance analysis of ModQUIC, QUIC and TCP has been carried out on the basis of throughput and delay, which is based on average datarate achieved and the packet sent per RTT respectively. However, fairness analysis is used to verify fair resource allocation.

a) Throughput and Delay analysis

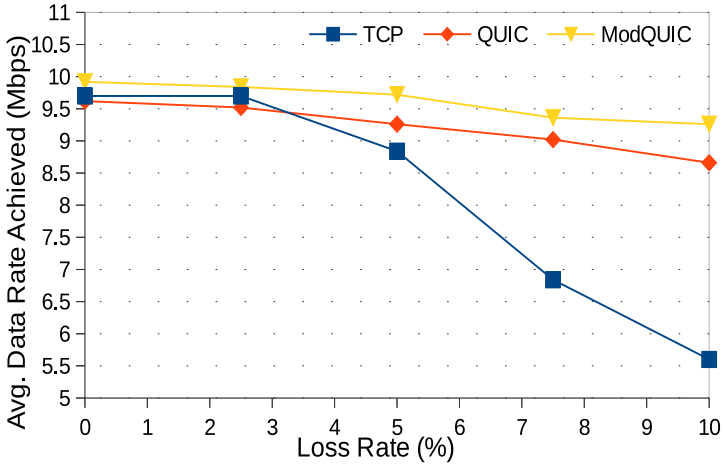
Figures 3.13a, 3.13b and 3.13c shows performance comparison of ModQUIC, QUIC and TCP with respect to loss rate for different link bandwidths. This has been accomplished based on the achieved average data rate using generated ACKs. At a loss of 0% and when sufficient link bandwidth is available, TCP outperforms due to its initial aggressiveness, whereas ModQUIC and QUIC performance are almost similar. The TCP performance gradually decreases at the rate of 0.21 Mbps per percent of loss rate. In ModQUIC and QUIC due to multiplexed streams and out of order delivery, better performance compared to TCP has been observed in the lossy link. However, ModQUIC is better than QUIC due to bandwidth occupancy limitation which in turn depends on the used window size. Even though the slow start is avoided in QUIC, its default window size is updated only with respect to an analysis of a previously sent packet's success rate and rate of transmission. In ModQUIC, maximum bandwidth utilization is observed, which in turn is responsible for window update. This fine-tuned window update mechanism per ACK reception results in reception of more number of packets within specified time and hence, as a whole ModQUIC outperforms.

For lossless bottleneck link of 5 Mbps and 10 Mbps, all three flows were closely competing with each other. As loss rate increases, the link becomes congestive and as a reaction, TCP reduces data rate gradually. However, it has been observed that performance of ModQUIC is improved by 21% over TCP and 3.43% over QUIC, due to built on top of UDP and fine-tuned window update mechanism. For sufficient bandwidth (50 Mbps) and lossless link, TCP dominates as packet pacing becomes an overhead for ModQUIC and QUIC. Once loss rate is increased, the performance of TCP suddenly drops down below ModQUIC and QUIC.

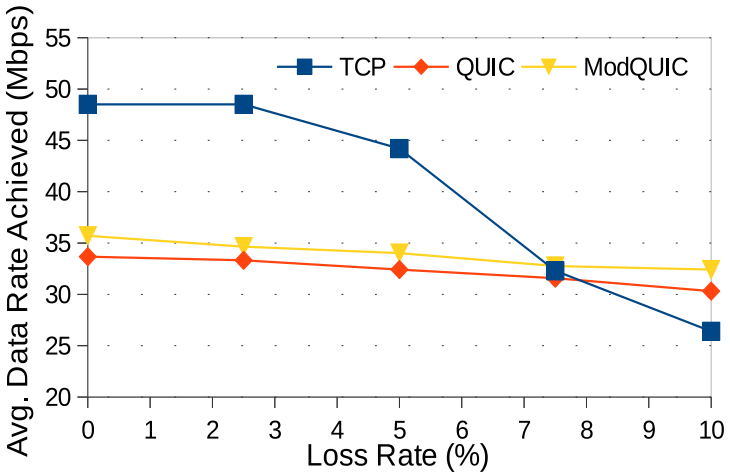
The delay is measured in terms of time required to send packets per RTT for different link bandwidth and loss rate (e.g. TCP0, QUIC0 and ModQUIC0 indicates performance of TCP, QUIC and ModQUIC for 0% loss, a similar representation is considered for 1% and



(a) 5 Mbps link

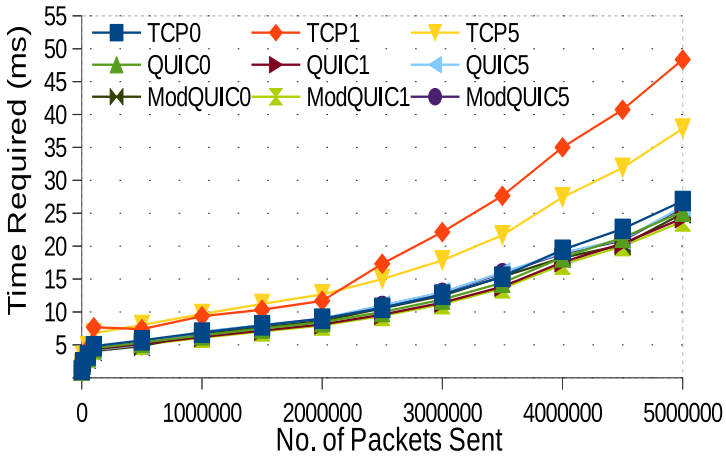


(b) 10 Mbps link

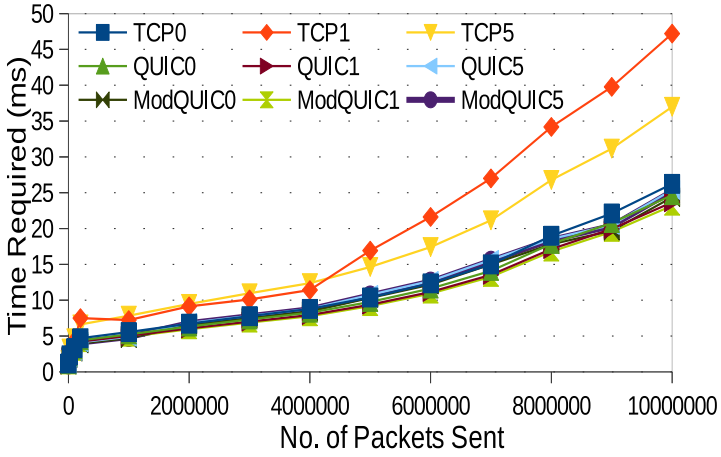


(c) 50 Mbps link

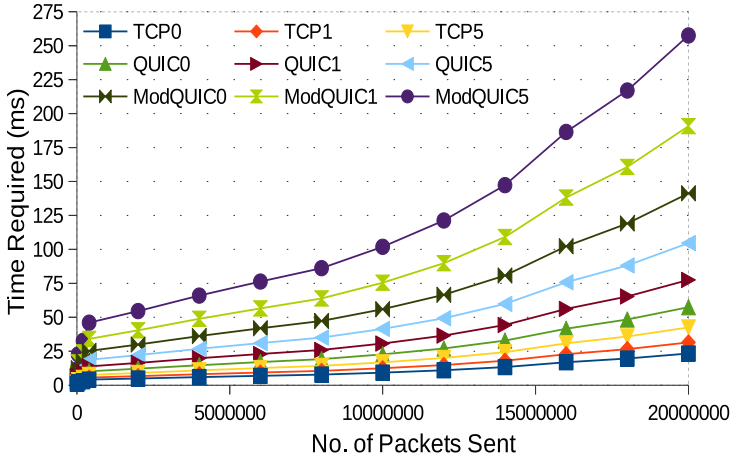
Figure 3.13: ModQUIC performance for different link bandwidth and loss rate



(a) 5 Mbps link



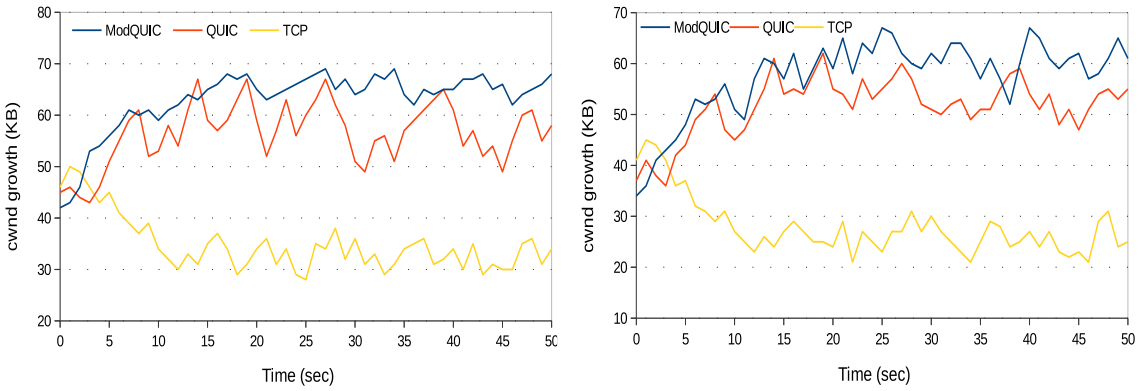
(b) 10 Mbps link



(c) 50 Mbps link

Figure 3.14: Time required based on RTT (20 ms) against number of packets sent in terms of link bandwidth and loss rate

5%) as shown in Figure 3.14a, 3.14b and 3.14c.



(a) 5 Mbps link with 2% loss

(b) 50 Mbps link with 2% loss

Figure 3.15: *cwnd* growth in KB against Time in second for ModQUIC, QUIC and TCP (RTT = 20 ms, buffer size = BDP)

Experimental results shown in Figure 3.15a and 3.15b investigate the *cwnd* growth with respect to time. To extract *cwnd* variation, ModQUIC and QUIC source codes are equipped, whereas tcpprobe [Persson *et al.* (2005)] is used for TCP. Figures 3.15a and 3.15b shows *cwnd* growth for link bandwidth of 5 Mbps and 50 Mbps with 2% loss, this shows smooth *cwnd* variation as well as maximum bandwidth utilization in ModQUIC compared to QUIC and TCP. As compared to TCP and QUIC, ModQUIC is able to achieve a greater share of the bandwidth. It is observed that even though all three protocols are using CUBIC congestion control mechanism, ModQUIC and QUIC increase their window more aggressively, observed both in terms of slope, and in terms of more frequent window size increases. As a result, ModQUIC and QUIC are able to snatch bandwidth faster than TCP, leaving TCP unable to acquire its fair share of the bandwidth. This unfairness is observed at the early stage of data transfer and fairness gets improved with time and file size, which is discussed in the Section 3.4.4.4(b).

Table 3.5 shows ModQUIC performance over TCP/HTTP2 in terms of throughput and speedup for single and multiple flows. The throughput characteristics drastically vary when ModQUIC and TCP are competing for flows as opposed to none at all. With extensive experimentation in a live network; results show that ModQUIC throughput outperforms TCP/HTTP2, especially for single dominant flow. In case of multiple streams (flows), two from each ModQUIC and TCP/HTTP2, HTTP2 creates multiple dedicated connections to serve each flow. However, ModQUIC uses multiplexed UDP streams in addition to dedi-

Table 3.5: Throughput and Speedup for ModQUIC against TCP/HTTP2 for various video file sizes

No. of Packets	Parameters	File Size		
		10 MB	30 MB	50 MB
1	ModQUIC Throughput (Mbps)	0.82	0.88	0.68
	TCP/HTTP2 Throughput (Mbps)	0.69	0.73	0.52
	Speedup (ModQUIC over TCP/HTTP2)	1.18	1.21	1.31
4	ModQUIC Throughput (Mbps)	0.61	0.78	0.91
	TCP/HTTP2 Throughput (Mbps)	0.57	0.71	0.79
	Speedup (ModQUIC over TCP/HTTP2)	1.07	1.09	1.16

cated connections which multiplexes TCP streams. This causes a fall in TCP packet transmission rate that results in throughput improvement of ModQUIC over TCP.

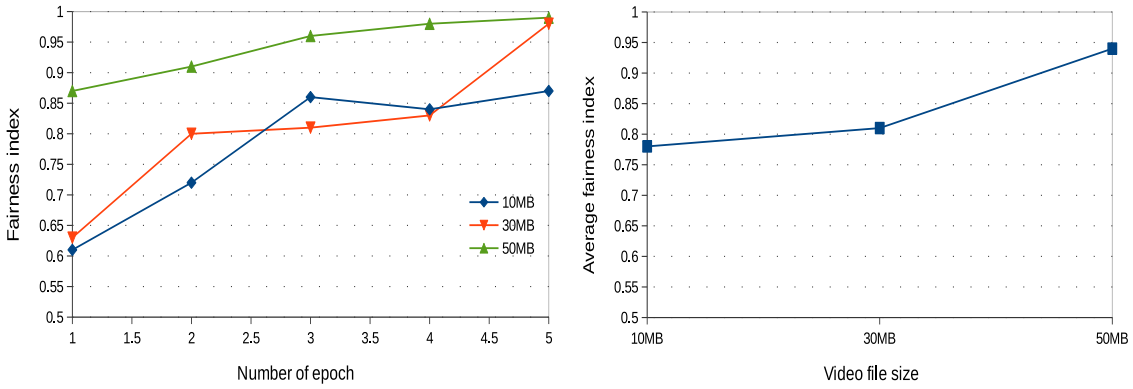
b) Fairness analysis

ModQUIC, QUIC and TCP flows are competing for bottleneck links of 5 Mbps, 10 Mbps and 50 Mbps for different RTT values and loss rates. The observations reveal that ModQUIC is a fair solution to serve multiple streams sharing bottleneck link.

The procedure has been carried out to test competing flows serviced by ModQUIC, QUIC, TCP and TCP/HTTP2. ModQUIC, QUIC and TCP flows were created with virtual nodes using OpenFlow Mininet platform, whereas separate browsers are opted to create TCP/HTTP2 flows. The TCP/HTTP2 flows are generated using Mozilla Firefox and Opera. The multiplexing nature of TCP/HTTP2 causes the use of a single TCP/IP connection which limits to test durability under multiple flows. The video files of size 10 MB, 30 MB and 50 MB were used and serviced by any or all flows. In a similar way, the same size files were used and serviced by TCP/HTTP2 using browser network.

The available bandwidth and ping (RTT) were tested during each epoch with Speedtest network monitoring tool [Speedtest (2016)], and each corresponds to the video file (for example, 18 Mbps bandwidth and 54 ms RTT for the 1 MB file download). The parameter space used for experimental analysis is given in Table 3.4.

Figure 3.16a and 3.16b shows fairness performance calculated based on Jain's fairness index. In this fairness, values considered to plot graphs are average values of 5 Mbps, 10 Mbps and 50 Mbps used links. Fairness index increases with respect to file size, and mostly for long live traffic, it is at higher side. The requests serviced via TCP/IP reserved large segment of the bandwidth, for smaller size files and were the primary contributor, the relatively more unfairness is observed.



(a) Fairness index **(b)** Average fairness index

Figure 3.16: Fairness analysis

3.4.5 Validation of Results

To validate and check consistency of the obtained results, linear regression model, R^2 (R-squared) has been employed. Regression is a statistical analysis indicating percentage of variance, given in equation (3.12). Regression analysis is a set of statistical processes for estimating the relationships among variables. To be specific, regression analysis helps us to understand how the typical value of the dependent variable changes when any one of the independent variables is varied, while the other independent variables are fixed.

$$R^2 = \frac{Variance\ explained\ by\ the\ model}{Total\ Variance} \times 100 \tag{3.12}$$

where, R^2 value ranges between 0% to 100%.

Table 3.6: R^2 values for ModQUIC performance validation

Parameters	R^2 Value (%)	
	ModQUIC v/s QUIC	ModQUIC v/s TCP
Throughput	97	92
Delay	96	70
Loss rate	96	33
Data rate (2 Mbps link with Loss)	99	96
Data rate (10 Mbps link with Loss)	98	89

A 0% R^2 value indicates that the suggested model is not consistent with respect to performance parameters, whereas 100% indicates model is perfect with respect to performance parameters. Larger the R^2 value, better is the regression model that fits for observations. Ta-

ble 3.6 shows R^2 values for performance comparison of ModQUIC protocol with QUIC and TCP. Overall observations are seen to fit the regression model. However, the values 70% and 33% are lower with respect to the desired output. It may be noted that, the lower R^2 value indicates that ModQUIC performance is improving significantly as compared to TCP. In delay and loss result analysis, ModQUIC and QUIC are almost following similar patterns, whereas ModQUIC and TCP are seen to drift from each other. In ModQUIC, there is a higher reduction in delay compared to TCP with respect to number of packets in flight. TCP performance is seen to be very poor in presence of loss compared to ModQUIC.

3.5 Summary

ModQUIC is a transport and application layer solution, which enhances the throughput, reduces latency and is easily deployable in an existing network. To determine the rate of transmission, a birth-death process based queuing model is used and window update information is set using steady state probability. The performance of the proposed modification in QUIC protocol is verified by using two different testing environments. The results are presented in the form of comparative analysis of ModQUIC, QUIC and TCP with the help of throughput, delay and behavior in presence of loss for limited and sufficient bandwidth for the prescribed parametric space. The observed improvement for ModQUIC in throughput over QUIC and TCP are 35.66% and 51.93% respectively. A marginal reduction in delay is observed with ModQUIC compared to QUIC, whereas delay reduction is significant over TCP. It has been observed that, for lossy link, TCP shows poor performance compared to ModQUIC and QUIC. However, the performance of ModQUIC and QUIC is found to be better and stable with time. It has been observed that for high-speed link, QUIC and ModQUIC act as a performance bottleneck. It has been observed that the performance of ModQUIC and QUIC is better in high BDP network and for large file sizes. The fairness analysis shows that fairness index improves with respect to file size and BDP. In this experimental analysis, it has been observed that the limitations of the ModQUIC are mainly due to cubic functionality. To overcome high BDP performance limitations, alternative congestion control mechanism to CUBIC, which is more aggressive in slow start phase (NewReno) or adaptive with respect to bottleneck bandwidth (BBR), may be useful solutions. Results were validated with the help of R^2 regression model.

Chapter 4

ModQUIC Protocol Performance with CUBIC and BBR Congestion Control Mechanisms

4.1 Introduction

A simultaneous investigation of network performance is carried out with situation based congestion by conducting experiments on customized testbed. This is an extension to ModQUIC protocol performance investigation, which has been contributed to congestion control dynamics of the protocol[1]. This work adds one more dimension to congestion control by using Bottleneck Bandwidth Round-trip-propagation-time (BBR) and suggested a CUBIC decrease factor $\beta = \beta_{TCP}/n$, for n flows, which are competing to acquire bottleneck resources, where $\beta_{TCP} = 0.3$ which is the decrease factor used in the TCP. A testbed has been prepared with Chromium server-client model and traffic monitoring tool. Results show that ModQUIC with BBR outperform ModQUIC with CUBIC and QUIC with CUBIC and BBR.

[1]Prashant Kharat and Muralidhar Kulkarni (communicated), "ModQUIC Protocol Performance Verification with CUBIC and BBR Congestion Control Mechanisms", *International Journal of Internet Protocol Technology, Inderscience*.

4.2 Congestion Control Mechanisms

There are various congestion control techniques presented in Afanasyev *et al*. (2010), Kharat and Kulkarni (2018), of which CUBIC has been deployed in QUIC protocol. At present, CUBIC is available by default in the QUIC stack, which is more stable, whereas QUIC with BBR is still experimental.

4.2.1 CUBIC Congestion Control

CUBIC is an enhanced version of BIC [Xu *et al.* (2004)] proposed by Ha *et al*. (2008), in which RTT independent *cwnd* growth function has been introduced. Basically, CUBIC is using H-TCP [Leith (2008)] approach to calculate *cwnd* size, which is a cubic function of elapsed time, t, since last congestion event.

$$W_{CUBIC} = C\left(t - \sqrt[3]{\frac{\beta.W_{max}}{C}}\right)^3 + W_{max} \tag{4.1}$$

where,
W_{CUBIC} is *cwnd* size,
W_{max} is a *cwnd* size just before last window size reduction,
C is predefined constant used as a scaling factor,
β is decrease factor.
Window size reduction at the time of loss event is $W(t) = W(t^*) * (1 - \beta)$, where $W(t^*)$ is the *cwnd* size at the time t^* of packet loss i.e. W_{max}.

Figure 4.1 shows the behavior of CUBIC with respect to the cubic function against time, in which $K = \sqrt[3]{\frac{\beta.W_{max}}{C}}$ and whenever packet dropping event occurs, the *cwnd* is reduced by a decrease factor β, otherwise increased by α for every successful ACK. For TCP with CUBIC, $\beta = 0.3$, whereas in QUIC with CUBIC, $\beta = 0.3/2 = 0.15$.
This work suggested, $\beta = \beta_{TCP}/n$ for QUIC with CUBIC and ModQUIC with CUBIC, where n are the number of flows competing to acquire bottleneck resource and β_{TCP} is the decrease factor used in TCP.
Every new epoch starts at $t = 0$ and α is set to some predefined value. In our experimental analysis we set it to 4, whereas in Cisco it has been set to 10. W_{max} is the initial *cwnd* size (*cwnd*(0)), where packet loss occurred previously. However, equation (4.1) preserve properties of BIC such as RTT fairness, limited slow start and rapid convergence. As an additional

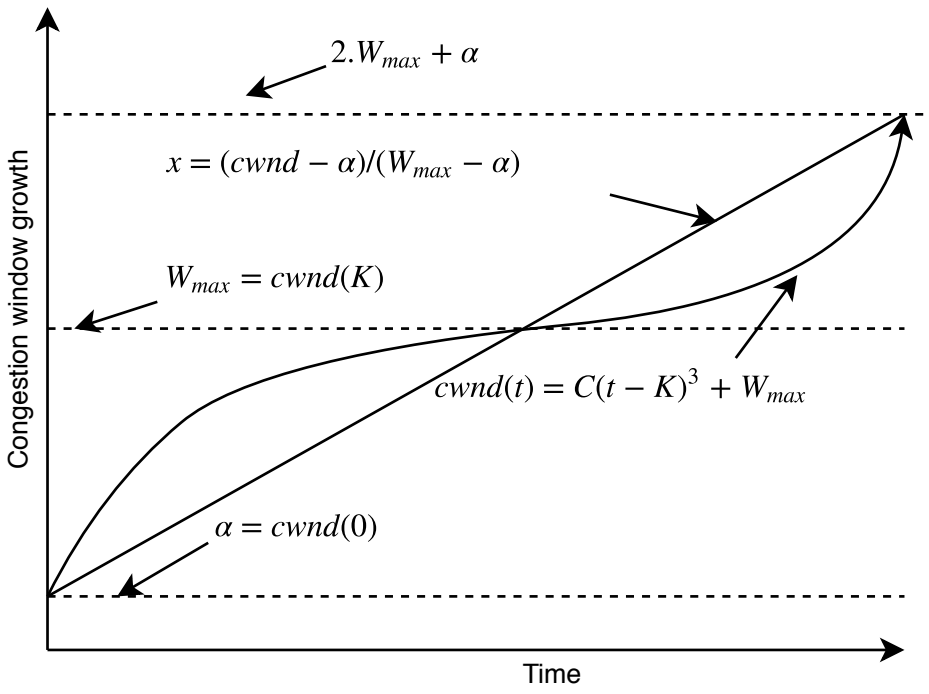


Figure 4.1: Congestion window growth with respect to cubic function against time

precaution CUBIC has a mechanism to ensure that the performance isn't worse than standard Reno with simultaneous checking and calculation of W_{reno} parameter. Based on other experimental studies, it has been observed that the performance and fairness properties of CUBIC are better than other congestion control mechanisms. Also, as this is available in the Linux TCP suite (kernel version 2.6.16), currently it is the most widely used congestion control mechanism.

4.2.2 BBR Congestion Control

In our previous studies, it has been observed that loss-based congestion control, such as CUBIC is the primary cause of major delay in modern fast networks, especially over large distances. This is due to CUBIC interpreting packet loss as congestion, an equivalence which was more or less valid at the time of the algorithm's conception. In modern days, Internet speed is transitioning from Kbps to Gbps, the equivalence of packet loss with congestion is not quite so straightforward.

The slowest link of Internet Explorer is the rate-determining factor for information transfer, which acts as a bottleneck for QUIC connection. A bottleneck often goes hand in hand

with a queue formed due to arrival rate exceeding departure rate. Thus solving the problem of the bottleneck will go a long way towards increasing throughput and minimizing delay during information transfer. In the current CUBIC implementation, when bottleneck buffers are large, the loss-based nature of CUBIC keeps them full, causing *bufferbloat*. When bottleneck buffers are small, CUBIC misinterprets loss as a signal of congestion as per its design, which leads to the low throughput.

To overcome these demerits of CUBIC, Google suggested BBR, a distributed congestion-control mechanism which reacts to actual congestion and not to the packet loss or transient queue delay. Also BBR is converges to an optimal operating point. This distributed approach to control congestion based on measuring the two parameters that characterize a path: Round-Trip-time-propagation (RTprop) and Bottleneck-Bandwidth (BtlBw), which are shown in Figure 4.2 [Cardwell *et al*. (2017)].
The key features of BBR are as follows:

- Considers RTprop and BtlBw as constraints for transport performance.
- This can be thought of as length (RTprop) and minimum diameter (BtlBw) of network pipe.
- Fewer data in a pipe: BBR measures RTprop and BtlBw parameters to accurately react to congestion, not loss or queuing effects.

A connection has maximum throughput and minimum delay when

- Rate Balance: *Bottleneck packet arrival rate = BtlBw*
- Full Pipe: $Total\ Data\ in\ Flight = BDP = BtlBw * RTprop$

In the Figure 4.2 constraint lines intersect at [$inflight = BtlBw * RTprop$], as good as the pipe's BDP. Since the pipe is full before this point, the [$inflight$ - $BDPexcess$] creates a queue at the bottleneck. Certainly, packets have been dropped once they exceed buffer capacity. As seen in Figure 4.2 congestion is carried by the right side of BDP line. However, congestion control is a mechanism, which holds a connection to operate at an average level of bandwidth limited region. A loss-based congestion control, CUBIC, operates at the right side edge of the bandwidth-limited region. This provides full BtlBw at the cost of high delay and frequent packet loss. At the time when memory chip was quite expensive, buffer size was slightly greater than the BDP, which minimized congestion control's excess

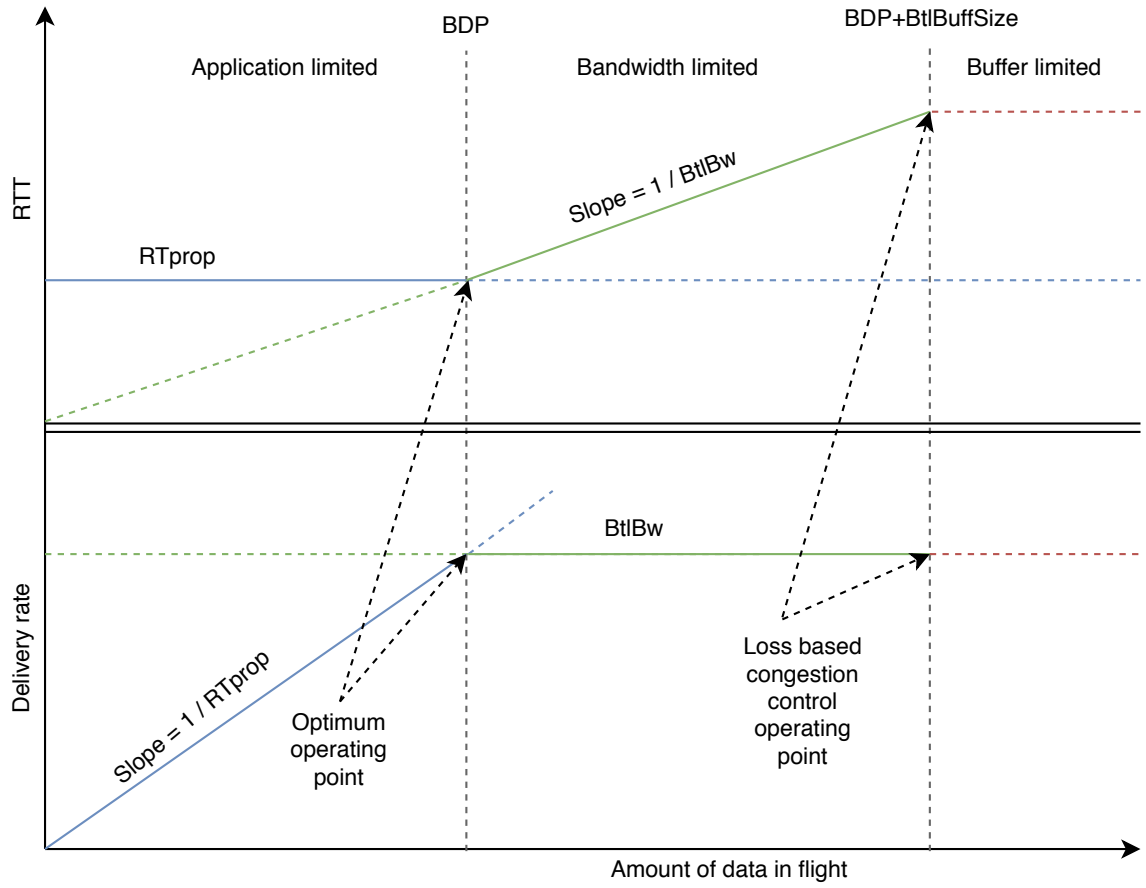


Figure 4.2: BBR response with respect to delivery rate and round trip time v/s amount of data in flight

delay. Larger buffer sizes create bufferbloat, which converts RTT value from milliseconds to seconds.

Thus, CUBIC assumes loss and buffer occupancy to capture the behavior of congestion, which is not realistic. However, BBR characterizes congestion based on how it occurs through RTT based BtlBw and RTprop estimation. BBR maintains a small queue size and optimum network utilization through distributed control loop with the help of packet pacing and control over sending rate.

4.3 Performance Verification with Congestion Control

To verify ModQUIC protocol performance with CUBIC and BBR congestion control mechanisms, an experiment setup has been developed and performance evaluated in terms of

throughput, delay and datarate achieved.

4.3.1 Experiment Setup

To carry out experimental performance evaluation, a testbed with Google certified, QUIC server-client model is developed using Chromium browser code-base [QUIC test server (2016)]. The Figure 4.3 shows a testbed environment with QUIC version 39 and the data has been generated by using www.example.org. By instrumenting QUIC source code as per the proposed mechanism presented in Section 3.3, ModQUIC functionality is integrated and logged relevant variables. There are two different configurations: ModQUIC enabled and QUIC enabled (default is TCP enabled) of Chromium, were deployed. The experiment is carried out multiple times by using loopback technique to verify the performance of both configurations with CUBIC and BBR. In ModQUIC and QUIC, by default loss based congestion control mechanism; CUBIC is present and BBR functionality was enabled by modifying a file which has a flag available at: chromium/src/net/quic/core/quic_flags_list.h is set to true. The logs of almost ten thousand packets with a payload size of 270 bytes have been recorded.

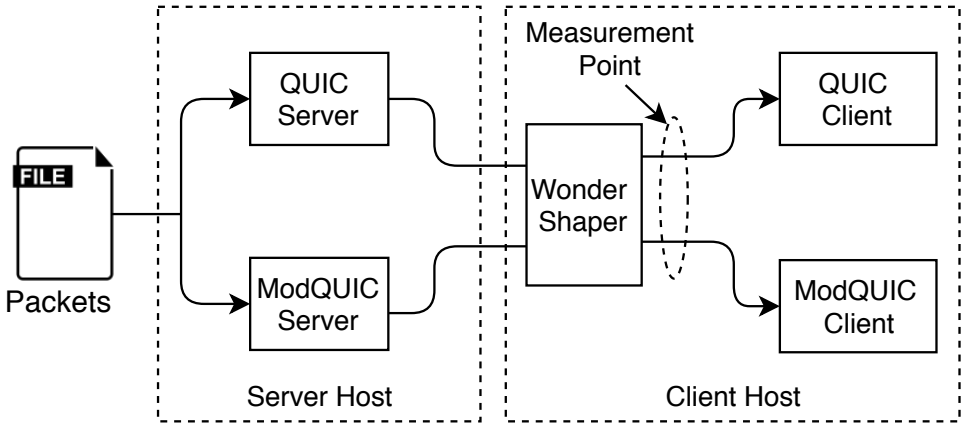


Figure 4.3: Testbed environment

System configuration used is Intel® Core^{TM} i5-2400 CPU @ 3.10 GHz X 4, 8 GB RAM, Ubuntu 14.04 LTS, 64 bit operating system. Chrome version 63.0.3239.132. Linux Kernel Version 4.4.0-93 generic. To monitor network performance, an iPerf tool is run on the client machine to measure the amount of data transferred and bandwidth available. However, wondershaper tool is deployed to manage traffic, to manipulate bandwidth, to fix packet loss and to set propagation delay.

4.3.2 Metrics and Parameter Space

The performance of ModQUIC and QUIC is measured in terms of throughput, delay and datarate achieved.
The parameter hyperspace used for the experiment is specified in the Table 4.1.

Table 4.1: Parameter hyperspace

Parameter	Values
Number of Packets	0 to 10000
Link Capacity	2 Mbps and 10 Mbps
Loss Rate	0%, 2%, 5% and 10%
Round Trip Time	20 ms

4.3.3 Performance Evaluation

This section presents comparative analysis based on experimental results in terms of throughput, delay and achieved datarate by ModQUIC with CUBIC and BBR. A dumbbell shape network topology has been created by adding 8% background traffic and with 20 ms base RTT to verify protocol performance. A buffer with size equal to BDP with a drop-tail policy is used. In this experiment, the total network utilization is around 92%, of which four flows ModQUIC/BBR, ModQUIC/CUBIC, QUIC/BBR and QUIC/CUBIC utilized 28%, 25%, 21% and 18% respectively, whereas background traffic contributed 8%. It has been observed that BBR performance is better and more stable compared to CUBIC. However, overall ModQUIC/BBR performance is better in all aspects.

Table 4.2: Throughput performance comparison of ModQUIC/BBR, ModQUIC/CUBIC, QUIC/BBR and QUIC/CUBIC by varying number of packets over 2 Mbps link

No. of Packets	Throughput (Mbps)			
	ModQUIC/BBR	ModQUIC/CUBIC	QUIC/BBR	QUIC/CUBIC
1000	3.213	2.924	2.740	2.431
2000	3.919	3.781	3.417	2.812
5000	4.723	4.223	3.928	3.301
8000	7.248	6.923	5.029	5.119
10000	9.112	8.529	7.223	6.512

In QUIC, window update is sent from the client only when if there is no congestion. The congestion situation is detected based on analysis of ACK reception time. However,

in ModQUIC, every ACK is associated with window update instead of depending on ACK reception time, which provides sufficient ground to increase the window size to a specified level. This results in ModQUIC throughput improvement over QUIC, which is observed in Table 4.2. Window size is dynamically updated for every ACK to ensure data delivery within optimum time. This reduces the number of retransmissions, which indicates minimal loss of data and results in more number of packets delivered within time, compared to the existing system. By this way, average end to end packet transmission delay is reduced. Even further improvement is observed with ModQUIC/BBR by reducing queuing delay (bufferbloat) as shown in Table 4.3.

Table 4.3: Performance comparison of ModQUIC/BBR, ModQUIC/CUBIC, QUIC/BBR and QUIC/CUBIC by varying number of packets over 2 Mbps link in terms of delay

No. of Packets	Average Delay (ms)			
	ModQUIC/BBR	ModQUIC/CUBIC	QUIC/BBR	QUIC/CUBIC
1000	2.053	2.118	2.132	2.153
2000	2.134	2.716	2.214	3.271
5000	2.482	2.554	2.583	2.648
8000	2.531	2.463	2.752	2.716
10000	2.466	2.357	2.778	2.450

Table 4.4: Performance comparison of ModQUIC/BBR, ModQUIC/CUBIC, QUIC/BBR and QUIC/CUBIC by varying number of packets over 2 Mbps link with respect to loss

Loss (%)	ACKs Generated (%)			
	ModQUIC/BBR	ModQUIC/CUBIC	QUIC/BBR	QUIC/CUBIC
0	96.20	94.40	93.00	92.34
2	94.13	92.76	92.11	89.62
5	97.25	88.45	94.25	80.21
10	83.45	81.64	71.58	74.55

In CUBIC, response to the loss is a cubic function, in which growth is slow at the beginning and exponential later, whereas BBR operates near to optimal region. So no queue builtup is observed and shows consistent and stable performance. This results in the performance improvement of ModQUIC and QUIC with respect to loss. However, aggressive nature of ModQUIC to acquire resource occupies greater share of bandwidth compared to QUIC, which is unfair. Table 4.4 shows percentage ACK generated, which indicates received packets, higher the percentage of ACKs generated, more are the packets received.

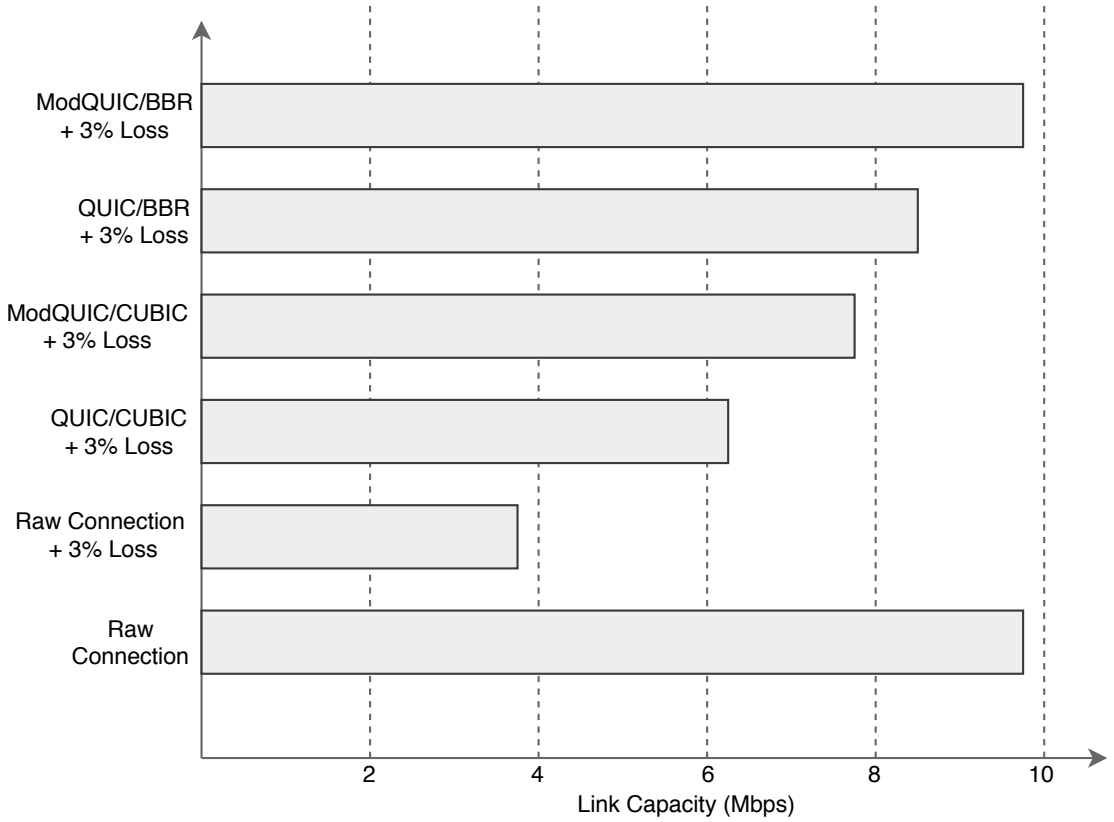


Figure 4.4: Performance comparison of ModQUIC with QUIC for a 10Mbps/20ms link with buffer size equal to BDP in presence of 3% loss

Table 4.4 shows that ModQUIC/BBR performance is better compared to others and overall ModQUIC outperforms. Figure 4.4 shows a case study in which it has been observed that ModQUIC performance in presence of 3% loss is equal to TCP raw connection. In ModQUIC and QUIC, due to multiplexed streams and out of order delivery, even in lossy links, the performance is superior. The performance of ModQUIC is better than QUIC due to bandwidth occupancy limitation which in turn depends upon the window size used. Even though the slow start is avoided in QUIC, its default window size is updated only with reference to an analysis of previously sent packet's success and their rate of transmission. This improves the reception of ACKs and the graph shows that ModQUIC outperforms QUIC. In ModQUIC, maximum bandwidth utilization is observed due to successful reception of ACK which is responsible for window update.

Table 4.5 shows that for lossless bottleneck link of 2 Mbps, all four flows are closely competing with each other to acquire bottleneck resource. An increase in loss results into congested link and in such situation ModQUIC performance is seen to be better than QUIC,

Table 4.5: Datarate achieved for 2 Mbps link in presence of loss

Loss (%)	Average Datarate Achieved (Mbps)			
	ModQUIC/BBR	ModQUIC/CUBIC	QUIC/BBR	QUIC/CUBIC
0	1.983	1.961	1.911	1.823
2	1.978	1.922	1.910	1.852
3	1.973	1.883	1.902	1.728
5	1.907	1.732	1.872	1.672
8	1.810	1.703	1.825	1.622
10	1.662	1.600	1.681	1.537

Table 4.6: Datarate achieved for 10 Mbps link in presence of loss

Loss (%)	Average Datarate Achieved (Mbps)			
	ModQUIC/BBR	ModQUIC/CUBIC	QUIC/BBR	QUIC/CUBIC
0	8.112	7.920	8.101	7.640
2	8.152	7.523	8.107	7.181
3	8.137	7.421	7.941	6.923
5	7.918	6.302	7.121	6.419
8	6.587	6.780	6.259	6.065
10	5.129	5.341	5.180	5.240

as window update is faster than QUIC. However, in the ideal condition, where sufficient bandwidth and lossless link is available, packet pacing creates overhead for ModQUIC and QUIC, which shows overall bandwidth utilization is less and further increases with loss as given in Table 4.6.

In fairness analysis, bottleneck link of 2Mbps/20ms and buffer size equal to BDP is considered as a resource for performance measurement. Table 4.7 shows average throughput for five iterations, in which it has been observed that when competing flows with similar transport layer protocol and congestion control strategy (Sl. No. 1) achieves equal throughput, due to fair share of resource allocation. However, competing flows with different transport layer protocol or congestion control strategy (Sl. Nos. 2 to 4) achieve unequal throughput, due to unfair share of resource allocation. The *cwnd* growth analysis shown in Figure 4.5 indicates that flows with different transport layer protocol or congestion control strategies show an unfairness, whereas flows with similar transport layer and congestion control strategies show friendliness and fair resource sharing.

Table 4.7: Average throughput with standard deviation of ModQUIC/BBR, ModQUIC/CUBIC, QUIC/BBR and QUIC/CUBIC competing flows for bottleneck link of 2Mbps/20ms and buffer size of BDP

Sl. No.	Competing Flows	Number of Flows	Average Throughput (Standard Deviation)
1	ModQUIC/BBR v/s ModQUIC/BBR	ModQUIC/BBR-1 ModQUIC/BBR-2 ModQUIC/BBR-3	0.62 (0.51) 0.61 (0.53) 0.64 (0.49)
2	ModQUIC/BBR v/s ModQUIC/CUBIC	ModQUIC/BBR-1 ModQUIC/CUBIC-1 ModQUIC/CUBIC-2	0.67 (1.02) 0.59 (0.43) 0.61 (0.81)
3	ModQUIC/BBR v/s QUIC/BBR	ModQUIC/BBR-1 QUIC/BBR-1 QUIC/BBR-2	0.92 (0.77) 0.54 (0.61) 0.51 (1.1)
4	ModQUIC/BBR v/s QUIC/CUBIC	ModQUIC/BBR-1 QUIC/CUBIC-1 QUIC/CUBIC-2	0.98 (0.46) 0.49 (0.23) 0.46 (0.39)

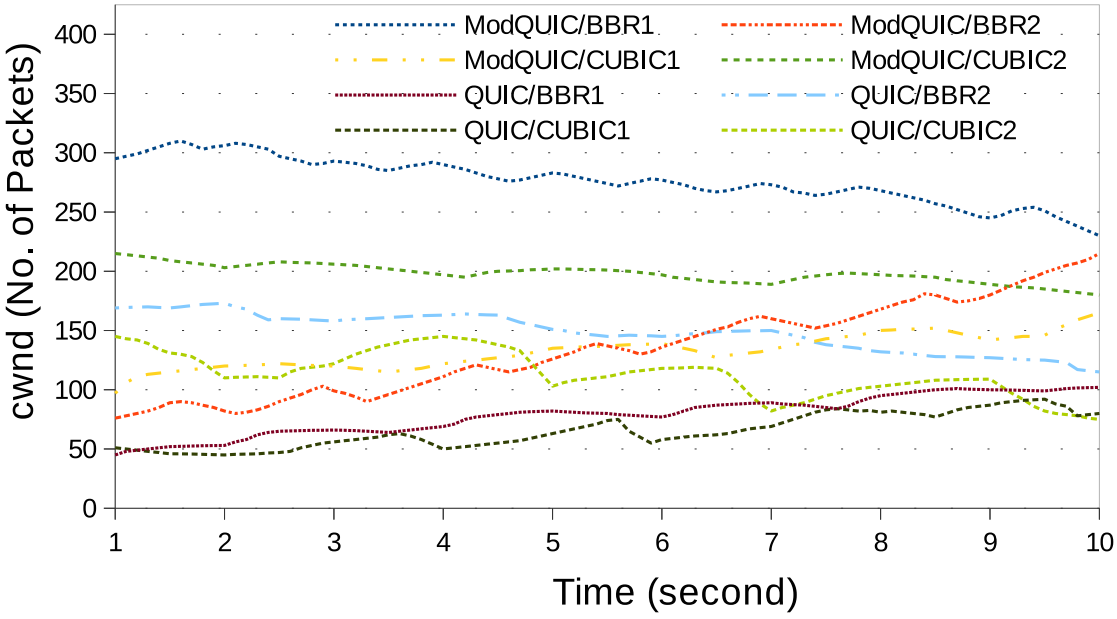


Figure 4.5: Congestion window growth of ModQUIC/BBR, ModQUIC/CUBIC, QUIC/BBR and QUIC/CUBIC competing flows in 2Mbps/20ms bottleneck link with buffer size equal to BDP

4.4 Summary

The performance of ModQUIC has been investigated with CUBIC and BBR congestion control mechanisms and the results were presented in the form of comparative analysis of ModQUIC/BBR, ModQUIC/CUBIC, QUIC/BBR and QUIC/CUBIC. The overall ModQUIC/BBR performance is better and stable for varied scenarios compare to other combinations. The improvement of ModQUIC/BBR throughput is 6.8%, 19.06% and 27.9% and delay reduction of 8.02%, 6.56% and 14.38% over ModQUIC/CUBIC, QUIC/BBR and QUIC/CUBIC respectively has been observed. It has been noticed that in lossy link, ModQUIC/BBR is more stable and outperform ModQUIC/CUBIC, QUIC/BBR and QUIC/CUBIC. For high-speed link, both ModQUIC and QUIC act as performance bottlenecks, whereas unfairness is observed when multiple flows are competing for bottleneck resource.

Chapter 5

Congestion Control Performance Investigation of ModQUIC Protocol using Jio-Fi Network: A Case Study

5.1 Introduction

Recently, Government of India promoted digitization of government services. This demands an improvement in online infrastructure and Internet connectivity to empower India digitally. Reliance Jio (JioFi) has brought digital empowerment to all Indians through connectivity with affordable data connection. The Reliance Jio as of now captured 16.02% of market share for a total of 160.09 million of subscribers [Gupta and Mukherjee (2018)]. [1]In this experimental study, the performance has been evaluated using emulated network conditions, against servers deployed with ModQUIC. The ModQUIC performance is tested with CUBIC and BBR congestion control mechanisms with respect to Throughput and packet Retransmission Ratio (RTR).

5.2 Experimental Investigation

To verify the performance of the ModQUIC protocol a testbed setup shown in the Figure 5.1 has been prepared and the performance is tested for parameter space given in Table

[1]Prashant Kharat and Muralidhar Kulkarni (under review), "Congestion Control Performance Investigation of ModQUIC Protocol using Jio-Fi Network: A Case Study", *Journal of High Speed Networks*.

5.1. The BBR functionality is enabled by modifying the flag bit in the Chromium source code available at chromium/src/net/quic/core/quic_flags_list.h [Chromiumblog (2015)]. The default status of this flag is set to false enables CUBIC as the default congestion control mechanism. The video files were uploaded on Google drive with sizes of 1 MB, 3 MB, and 5 MB. The net-internals tool of Chromium was used to monitor all network activities. This tool records network parameters like protocol used, packets sent, packets lost, packets received, throughput for all live connections serviced by ModQUIC.

5.2.1 Testbed Setup and Parameter Space

During experimental analysis, the main objectives are to examine the performance of ModQUIC in real network congestion for a live streaming. As shown in the Figure 5.1, JioFi[1] is used as an Internet Service Provider (ISP), whereas wireless router has been prepared with network emulator (Netem) enabled Raspberry Pi-3 board. A MacBook Pro laptop running chromium browser built from source is used to carry out measurements[2].

The following tools and packages were used to setup the testbed:

- Raspberry Pi-3 running Ubuntu MATE with Netem installed,
- The dnsmasq and hostapd packages for DHCP allocation of incoming WLAN,
- NAT input forwarding to WLAN for wireless broadcast.

To the server side, Google drive has been used to serve requests from the client due to the relative rarity of live ModQUIC enabled server. The Google services, such as Drive and YouTube, use ModQUIC as a native protocol, as opposed to other web servers where QUIC has limited deployment. It is also possible to build a ModQUIC server-client topology from the chromium code base as given in QUIC test server (2016). This implementation of a ModQUIC server has been used to test results from the laptop specified.

The Netem's Traffic Control (Tc) was used to introduce latencies, packet loss, and packet reordering to all outgoing packets according to the parameter space in Table 5.1. This wireless router served the ModQUIC enabled Chromium Client[3] from a Google drive storing video files of different sizes, as seen in Table 5.1.

[1]WAN: LTE (2300/1800/850MHz) IEEE 802.11b/g/n 2.4 G

[2]Intel Core i5 CPU @ 2.5 GHz with 4 GB RAM running MacOS Sierra 10.12.6 (64 bit) and Chromium version was 63.0.3229.0 (64 bit), whereas Darwin kernel version was 16.7.0

[3]Intel Core i5 CPU @ 2.5 GHz with 4 GB RAM running MacOS Sierra 10.12.6 (64 bit). A Chromium version is 63.0.3229.0 (64 bit) and Darwin kernel version is 16.7.0

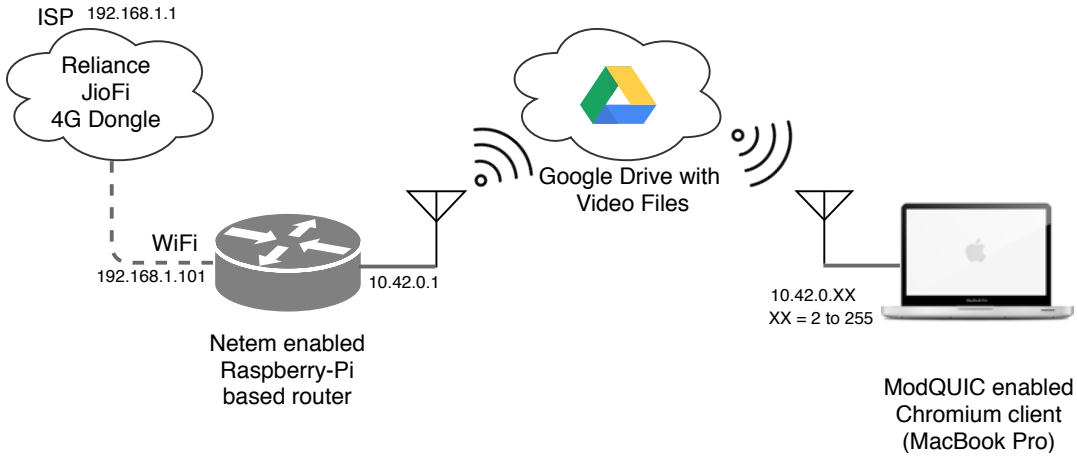


Figure 5.1: Testbed setup built with ModQUIC server provided as part of Chromium source code, Netem enabled Raspberry-Pi wireless router with Jio as a source with RTT of 20 ms

Table 5.1: Parameter hyperspace

Category	Parameter	Values
Server Side (Google Drive)	File Size	1 MB, 3 MB, 5 MB
Netem Router (Network)	Link Capacity	500 Kbit
	Loss	0%, 2%, 5%
	RTT	20 ms
	Packet Reorder	10%, 25% correlation

The variation of data speeds observed through the Jio network during the experimentation process was noted using a Python script and graphically shown in the Figure 5.2 to reflect the real behavior.

5.2.2 Result Analysis

Two parameters, RTR and Throughput has been analyzed across varying link capacity and loss for different file sizes, whereas calculated by using equations (5.1) and (5.2).

$$RTR = \frac{Number\ of\ Packets\ Received}{Number\ of\ Packets\ Transmitted} \tag{5.1}$$

$$Throughput = \frac{Flie\ Size}{Downloaded\ Time} \tag{5.2}$$

The performance of ModQUIC with CUBIC and BBR in presence of loss for various file sizes has been presented in Table 5.3, 5.4 and 5.5. The RTR and Throughput values

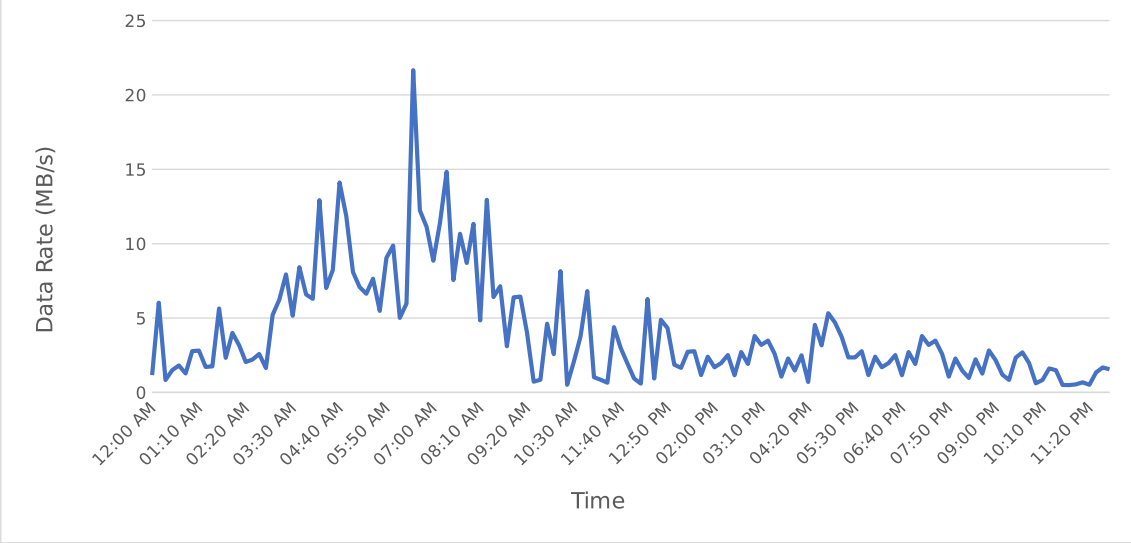


Figure 5.2: 24-hour time series analysis of Jio data rate variations

presented across 5 epochs for all file sizes. The values for packets transmitted, packet lost, packet received, packet retransmitted, retransmit ratio, download time and throughput are logged.

The net-internals tool is used to calculate RTR and Throughput over all permutations of the hyperspace detailed in Table 5.1. The Figure 5.6 shows trend of the Throughput and packet retransmission. For small video files, BBR throughput suffers with increase in loss, as seen in the Figure 5.6a. This is due to insufficient time for BBR to calculate estimates of BtlBw and RTprop and converge preemptively to react for congestion. In this case, loss based congestion control reacts to the loss, faster than BBR can build the model. However, as file sizes increases, BBR shows a clear improvement in throughput over CUBIC in all scenarios, as seen in the Figure 5.6b. Even as the loss increases, due to increase in convergence time BBR is able to accurately prevent congestion by modifying the transmission data rate. This effect is amplified for the large file size of 5 MB, where CUBIC throughput drops significantly for higher loss, whereas BBR reacts preemptively to prevent bufferbloat, as seen in the Figure 5.6c.

The RTR analysis was relatively straightforward as the trends were consistent across all scenarios. For a given file size, as shown in the Figure 5.6, it is observed that there is a direct correlation between loss and RTR in all cases with increase in loss the RTR decreases for both BBR and CUBIC. For smaller file size in the Figure 5.6d, BBR has less transmissions than it does for higher file sizes as seen in the Figure 5.6e and 5.6f, as it has

Figure 5.3: RTR and Throughput for 0% loss, where [1]packet, [2]transmit, [3]receive, [4]retransmit

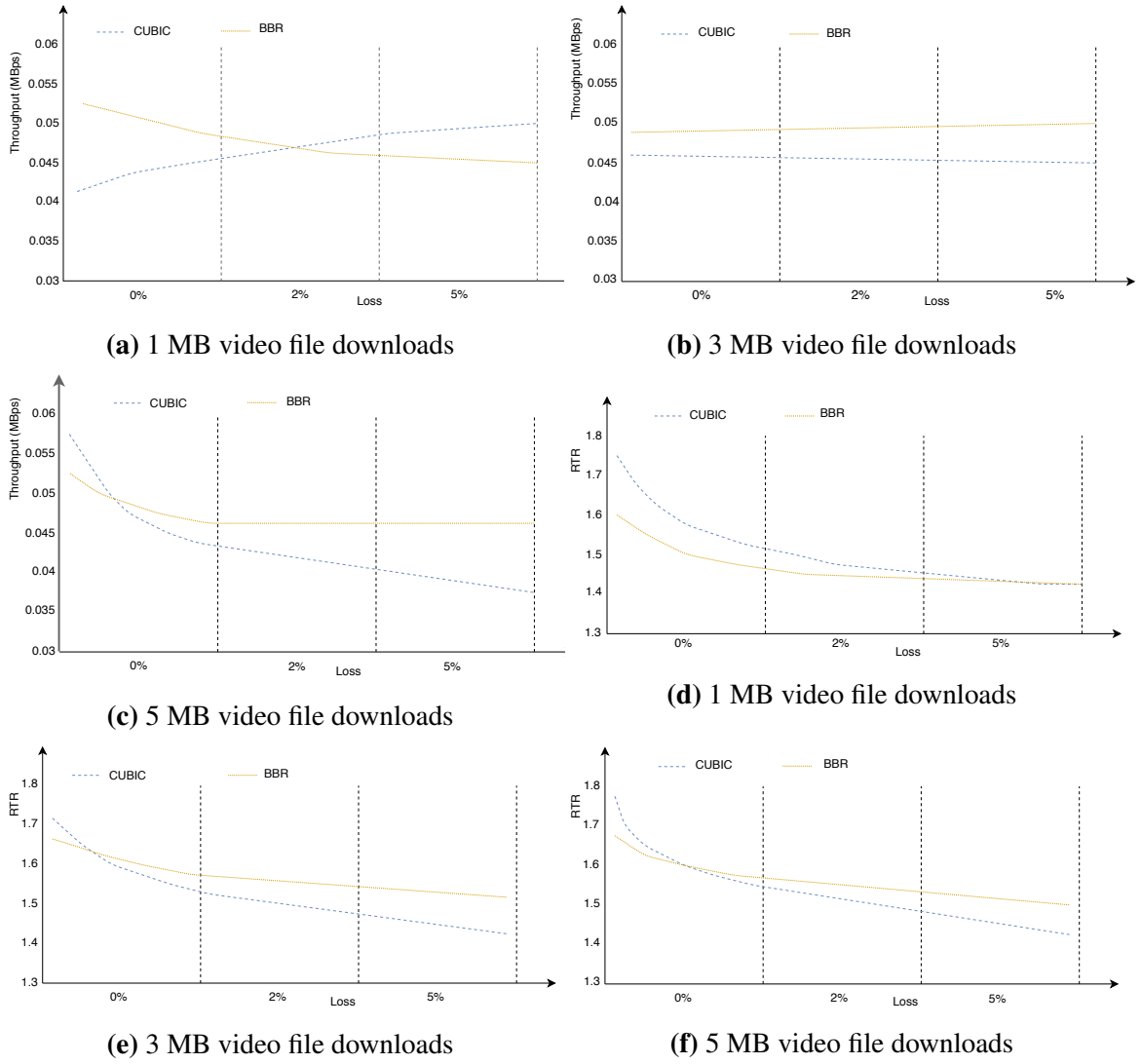
BBR (0% Loss)								CUBIC (0% Loss)							
File Size	Pkt[1] Tx[2]	Pkt Loss	Pkt Rx[3]	Pkt RTx[4]	RTR	Time (s)	Throughput (MB/s)	File Size	Pkt Tx	Pkt Loss	Pkt Rx	Pkt RTx	RTR	Time (s)	Throughput (MB/s)
1 MB	561	0	860	299	1.533	18.6	0.0538	1 MB	478	0	824	346	1.724	24.6	0.0406
	510	0	826	316	1.630	18.4	0.0543		480	0	834	354	1.737	18.5	0.0540
	549	0	846	297	1.541	34.7	0.0288		524	1	820	296	1.565	29.7	0.0336
	555	0	837	282	1.508	18.6	0.0537		466	0	818	352	1.755	18.7	0.0534
	537	0	839	302	1.562	17.7	0.0565		520	0	837	317	1.609	18.5	0.0540
Avg	542.4	0	841.6	299.2	1.552	21.6	0.0463	Avg	493.6	0.2	826.6	333	1.674	22	0.0454
3 MB	1688	0	2678	990	1.586	73.4	0.0436	3 MB	1629	0	2691	1062	1.651	65	0.0492
	1607	0	2639	1032	1.642	59.2	0.0540		1704	1	2619	915	1.537	117.7	0.0271
	1574	0	2625	1051	1.668	57.7	0.0554		1670	0	2619	949	1.568	67	0.0477
	1613	0	2609	996	1.617	61.5	0.0520		1625	0	2641	1016	1.625	64.7	0.0494
	1609	1	2649	1040	1.646	65.7	0.0487		1495	0	2588	1093	1.731	60.8	0.0526
Avg	1618.2	0.2	2640	1021.8	1.631	63.5	0.0503	Avg	1624.6	0.2	2631.6	1007	1.619	75.04	0.0426
5 MB	2605	0	4445	1840	1.706	103.7	0.0530	5 MB	2677	0	4446	1769	1.660	109.4	0.0502
	2752	0	4508	1756	1.638	101.6	0.0541		2723	0	4478	1755	1.644	105.8	0.0519
	2837	0	4487	1650	1.581	103.4	0.0532		2719	0	4489	1770	1.651	124.1	0.0443
	2814	1	4460	1646	1.585	108	0.0509		2720	0	4487	1767	1.649	106.1	0.0518
	2820	0	4491	1671	1.592	107.6	0.0511		2785	0	4471	1686	1.605	109	0.0504
Avg	2765.6	0.2	4478.2	1712.6	1.619	104.9	0.0524	Avg	2724.8	0	4474.2	1749.4	1.642	110.8	0.0496

Figure 5.4: RTR and Throughput for 2% loss, where [1]packet, [2]transmit, [3]receive, [4]retransmit

BBR (2% Loss)								CUBIC (2% Loss)							
File Size	Pkt[1] Tx[2]	Pkt Loss	Pkt Rx[3]	Pkt RTx[4]	RTR	Time (s)	Throughput (MB/s)	File Size	Pkt Tx	Pkt Loss	Pkt Rx	Pkt RTx	RTR	Time (s)	Throughput (MB/s)
1 MB	570	1	854	284	1.498	27.2	0.0367	1 MB	600	0	829	229	1.382	38.9	0.0257
	571	0	851	280	1.490	18.6	0.0537		577	0	836	259	1.448	21.3	0.0469
	588	0	859	271	1.460	18.8	0.0531		536	0	820	284	1.529	19.9	0.0502
	626	0	901	275	1.439	20.9	0.047		555	0	844	289	1.520	20.9	0.0478
	530	0	831	301	1.567	18.3	0.0546		539	0	831	292	1.541	19.6	0.0510
Avg	577	0.2	859.2	282.2	1.489	20.76	0.0481	Avg	561.4	0	832	270.6	1.482	24.12	0.0414
3 MB	1583	1	2612	1029	1.650	64.8	0.0493	3 MB	1721	0	2625	904	1.525	69.4	0.0461
	1661	0	2616	955	1.575	61.1	0.0523		1648	0	2608	960	1.582	61.3	0.0523
	1749	0	2628	879	1.502	87.6	0.0365		1656	0	2593	937	1.565	61.1	0.0523
	1649	0	2615	966	1.585	61.5	0.0520		1629	0	2606	977	1.600	58.0	0.0551
	1618	0	2609	991	1.612	61.1	0.0523		1678	0	2615	937	1.558	62.0	0.0516
Avg	1652	0.2	2616	964	1.583	67.22	0.0476	Avg	1666.4	0	2609.4	943	1.565	62.3	0.0513
5 MB	2945	1	4643	1698	1.576	115.2	0.0477	5 MB	2717	0	4436	1719	1.632	102.7	0.0535
	3071	1	4802	1731	1.563	137.3	0.0407		2896	0	4491	1595	1.550	115.3	0.0477
	3004	0	4696	1692	1.563	120.2	0.0407		2804	0	4461	1657	1.591	114.4	0.0480
	3005	0	4521	1516	1.504	135.1	0.0407		2979	0	4588	1559	1.523	138.7	138.7
	2916	0	4491	1575	1.540	110.1	0.0499		2756	0	4466	1710	1.620	105.6	0.0520
Avg	2988.2	0.4	4630.6	1642.4	1.550	123.6	0.0445	Avg	2830.4	0	4478.4	1648	1.582	115.3	0.0476

Figure 5.5: RTR and Throughput for 5% loss, where [1]packet, [2]transmit, [3]receive, [4]retransmit

BBR (5% Loss)								CUBIC (5% Loss)							
File Size	Pkt[1] Tx[2]	Pkt Loss	Pkt Rx[3]	Pkt RTx[4]	RTR	Time (s)	Throughput (MB/s)	File Size	Pkt Tx	Pkt Loss	Pkt Rx	Pkt RTx	RTR	Time (s)	Throughput (MB/s)
1 MB	609	0	859	250	1.410	21.0	0.0476	1 MB	578	0	836	258	1.446	19.4	0.0515
	551	0	819	268	1.486	20.5	0.0487		570	0	837	267	1.485	19.0	0.0502
	571	0	828	257	1.450	21.7	0.0460		562	0	835	273	1.485	19.0	0.026
	703	0	962	259	1.368	37.4	0.0267		619	0	845	226	1.365	22.7	0.044
	604	0	852	248	1.410	19.3	0.0518		555	0	820	265	1.477	19.3	0.0518
Avg	608	0	864	256.4	1.422	23.98	0.0417	Avg	576.8	0	834.6	257.8	1.446	20.1	0.0498
3 MB	1773	0	2641	868	1.489	63.4	0.050	3 MB	1875	0	2652	777	1.414	73.5	0.0435
	1834	0	2615	781	1.425	67.2	0.0476		1852	0	2626	774	1.417	79.4	0.0403
	1875	0	2705	830	1.442	66.5	0.0481		1836	0	2622	786	1.428	72.7	0.0440
	1724	0	2637	913	1.529	63.8	0.0501		1966	0	2620	654	1.332	102	0.0313
	1732	0	2616	884	1.510	61.8	0.0517		1982	0	2622	640	1.322	82.1	0.0389
Avg	1788	0	2642.8	855.2	1.478	64.54	0.0495	Avg	1902.2	0	2628.4	726.2	1.381	81.94	0.0390
5 MB	3019	0	4465	1446	1.479	129.3	0.0425	5 MB	3306	0	4481	1175	1.355	160.4	0.0342
	3046	0	4445	1399	1.459	115	0.0478		3482	0	4529	1047	1.300	175.7	0.0313
	2889	0	4452	1563	1.541	104.3	0.0527		3319	1	4459	1140	1.343	281.9	0.0237
	3038	0	4493	1455	1.478	109.8	0.0500		3224	0	4507	1283	1.398	145.7	0.0387
	2822	0	4426	1604	1.568	102.1	0.0538		3262	0	4485	1223	1.374	142.1	0.0387
Avg	2963	0	4456.2	1493.2	1.504	112.1	0.0490	Avg	3318	0.2	4492	1173.6	1.353	171.2	0.0321



(a) 1 MB video file downloads

(b) 3 MB video file downloads

(c) 5 MB video file downloads

(d) 1 MB video file downloads

(e) 3 MB video file downloads

(f) 5 MB video file downloads

Figure 5.6: Throughput and RTR performance of ModQUIC with CUBIC and BBR for different loss rates and file sizes

not had the convergence time necessary to correct the congestion. This behavior manifests itself in larger file sizes due to corrective measures being applied by BBR. A CUBIC shows constant behavior in its loss-based retransmission system for all file sizes, with a decrease from a higher starting RTR for the smallest file size, as seen in the Figure 5.6d.

There is a substantial improvement in BBR congestion control over CUBIC as BBR is not using loss as an indicator to congestion. To utilize full bandwidth, currently existing loss-based congestion control schemes such as CUBIC require the loss rate to be less than the inverse square of the BDP [Mathis *et al.* (1997)]. The performance with varying loss is

examined by Google for 60 second flow on 100Mbps/100ms link with 0.001% to 50% of random loss serviced by HTTP/2 over TCP/IP [Cardwell *et al*. (2017)]. A Google observed CUBIC's throughput to decrease tenfold at 0.1% loss and totally stall above 1%. The maximum theoretical possible throughput for a network is the link rate times fraction delivered ($= 1 - LossRate$). Google observed BBR to meet this limit asymptotically up to a 5% loss and reasonably close up to 15% loss. This performance has been verified on a live wireless 500Kbps/20ms link with 1% to 5% random loss serviced by QUIC protocol.

5.3 Summary

As the experimental analysis was carried out in a wireless environment, it was realized that packet loss becomes an incredibly significant measure of importance for quality of a connection. The FEC feature would aid in minimizing local loss due to bit error. However, due to inconclusive results as to its efficiency [Langley and Chang (2013)], the FEC functionality of QUIC was removed from Chromium by Google in 2016. This is the reason that focus has been shifted to a more efficient congestion control to correct losses, as is reflected in our attempts to analyze BBR. The result analysis shows a clear advantage in terms of Throughput, and was verified by RTR for BBR over CUBIC in situations of congestion in live wireless networks. This study recommends the use of BBR in ModQUIC as a measure, to more accurately model and predict congestion in middle boxes as opposed to react to packet loss, once there is buffer overflow. However, there were limitations in JioFi, like limited data rate else the results would have been far better analyzed as reported here.

Chapter 6

Concluding Remarks and Future Work

The research work presented in this thesis started with a complete study of transport layer protocols and congestion control mechanisms. During the study, we became familiar with the role of transport layer with congestion control dynamics in network performance. Here, the congestion window is a crucial entity while evaluating the performance of the network, whereas to analyze full functionality of the congestion window, we applied birth-death process and evaluated the congestion window growth. The QUIC protocol is the new transport layer solution to compete with TCP's dominance. Throughout from the literature survey on congestion control and QUIC protocol, we have suggested modification in the existing handshaking mechanism of QUIC protocol, called ModQUIC to improve network performance. This modification aid improvement in QUIC protocol performance in terms of throughput and delay.

For rigorous performance analysis of the developed protocol, three different testbeds were created. In first contribution (Chapter-3), the developed protocol is tested with Chromium server-client model testbed and simulation environment in Mininet. However, browser network environment is created by generating flows using Chrome, Opera and Mozila Firefox to test the fairness. The result analysis shows that the network performance has been improved significantly with ModQUIC.

In ModQUIC performance analysis, it has been noticed that the limitations of the protocol are mainly due to CUBIC congestion control mechanism. In Chapter 4, to improve CUBIC performance, we suggested a decrease factor $\beta = \beta_{TCP}/2$ for single flow and $\beta = \beta_{TCP}/n$ for n number of concurrent flows, whereas we suggested BBR as an alternate congestion control mechanism, which resolves limitations of CUBIC upto certain extent. To verify above claims, an experimental setup has been created and performance of the

ModQUIC was tested with CUBIC and BBR. The result analysis shows that overall performance of the network is better with BBR compared to CUBIC. We have extended ModQUIC performance with CUBIC and BBR with Reliance Jio wireless network in Chapter 5 and observed clear advantage with BBR.

Future Work

The present work can be extended to future research. The following could be implemented to contribute further to this area of research.

- An adaptive congestion window growth is a suitable candidate in which based on available bandwidth and type of traffic can help in further performance improvement of the transport layer.
- Fair allocation of available resources to competing flows is a simple form of QoS that can be provided to customers in packet-switched networks. The complexity of existing algorithms prevents a high-speed network implementation with the current state of router technique like Active Queue Management (AQM).
- In wireless environment, both TCP and Physical layer jointly control the congestion. The physical layer can control transmission power as per the channel condition, interference received and congestion in the network, whereas the TCP layer controls congestion.
- Dynamic packet fragmentation and packet size selection procedure based on available bandwidth or delivery window size for a particular link.

Bibliography

Afanasyev, A., Tilley, N., Reiher, P. and Kleinrock, L. (2010). Host-to-host Congestion Control for TCP. *IEEE Communications Surveys & Tutorials*, 12(3), 304-342.

Cerf, V. (1973). A Partial Specification of an International Transmission Protocol.

Cerf, V. and Kahn, R. (1974). A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 22(5), 637-648.

Cerf, V. and Postel, J.B. (1978). Specification of Internetwork Transmission Control Program. TCP Version, 3.

Jacobson, V. (1988). Congestion Avoidance and Control. *Computer Communication Review*, 18(4), 314-329.

Jacobson, V. (1990). Modified TCP Congestion Avoidance Algorithm. *ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail*.

Braden, R., Ed. (1989). Requirements for Internet Hosts - Communication Layers. STD 3, *RFC 1122*.

Willis, N. (2013). Connecting on the QUIC. *Linux Weekly News*.

Bright, P. (2018). The Next Version of HTTP won't be using TCP. *Ars Technica*.

Behr, M. and Swett, I. (2018). Introducing QUIC Support for HTTPS Load Balancing. Google Cloud Platform Blog, Google, *Retrieved 16 June 2018*.

Roskind, J., (2013). QUIC: IETF-88 TSV Area Presentation. *Retrieved 2013-11-07*.

Kuehlewind, M. and Trammell, B. (2017). Applicability of the QUIC Transport Protocol. *draft-ietf-quic-applicability-01* (work in progress).

Hamilton, R., Iyengar, J., Swett I. and Wilk, A. (2016). QUIC: A UDP-based Secure and Reliable Transport for HTTP/2. Network Working Group, Internet-Draft. *Available: http://www.connectify.me/taking-googlequic-for-a-test-drive/.*

Ha, S., Rhee, I. and Xu, L. (2008). CUBIC: A New TCP-friendly High-speed TCP Variant. *ACM SIGOPS operating systems review*, 42(5), 64-74.

Cardwell, N., Cheng, Y., Gunn, C.S. and Yeganeh, S.H. (2017). BBR: Congestion-based Congestion Control. *Communications of the ACM*, 60(2), 58-66.

Hassan, M., Jain, R. (2003). High Performance TCP/IP Networking: Concepts, Issues and Solutions, *Upper Saddle River, NJ: Prentice Hall*, 29.

Molnár, S., Sonkoly, B. and Trinh, T.A. (2009). A Comprehensive TCP Fairness Analysis in High Speed Networks. *Computer Communications*, 32(13-14), 1460-1484.

Cisco, V.N.I., (2018). Cisco Visual Networking Index: Forecast and Trends, 2017–2022. *White Paper*.

Elkhatib, Y., Tyson, G. and Welzl, M. (2014). Can SPDY Really Make the WEB Faster?. *IFIP Networking Conference, IFIP*, 1-9, IEEE.

Belshe, M., Peon, R. and Thomson, M. (2015). Hypertext Transfer Protocol version 2, (http/2), *RFC 7540*.

Floyd, S. (1994). TCP and Explicit Congestion Notification, *ACM SIGCOMM Computer Communication Review*, 24(5), 8-23.

Kwon, M. and Fahmy, S. (2004). On TCP Reaction to Explicit Congestion Notification, *Journal of High Speed Networks*, 13(2), 123-138.

Ramani, R., Karandikar, A. (2000). Explicit Congestion Notification (ECN) in TCP over Wireless Network, *IEEE International Conference on Personal Wireless Communications Networks*, 495-499.

Sukhov, A.M., Sidelnikov, D.I., Platonov, A.P., Strizhov, M.V. and Galtsev, A.A. (2011). Active Flows in Diagnostic of Troubleshooting on Backbone Links, *Journal of High Speed Networks*, 18(1), 69-81.

Hanbali, A., Altman, E., and Nain, P. (2005). A Survey of TCP over Ad-hoc Networks, *IEEE Communications Surveys and Tutorials*, 7(1-4), 22-36.

Lochert, C., Scheuermann B., and Mauve, M. (2005). A Survey on Congestion Control for Mobile Ad-hoc Networks, *Wiley Wireless communications and mobile computing*, 7(5), 655-676.

Ikeda, M., Kulla, E., Hiyama, M., Barolli, L., and Takizawa, M. (2013). Investigation of TCP and UDP Multiple-flow Traffic in Wireless Mobile Ad-hoc Networks, *Journal of High Speed Networks*, 19(2), 129-145.

Widmer, J., Denda, R., and Mauve, M. (2001). A Survey on TCP-friendly Congestion Control, *IEEE Network*, 15(3), 28-37.

Leung, K.C., Li, V.O., and Yang, D. (2007). An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions and Challenges, *IEEE Transactions on Parallel and Distributed Systems*, 18(4).

Hasegawa, G., and Murata, M. (2001). Survey on Fairness Issues in TCP Congestion Control Mechanisms, *IEICE Transactions on Communications*, 84(6), 1461-1472.

Balakrishnan, H., Padmanabhan, V.N., Seshan, S. and Katz, R.H. (1997). A Comparison of Mechanisms for Improving TCP Performance over Wireless Links, *IEEE/ACM Transactions on Networking*, 5(6), 756-769.

Jacobson, V. (1988). Congestion Avoidance and Control, *ACM SIGCOMM*, 18(4), 314-329.

Jacobson, V. (1990). Modified TCP Congestion Avoidance Algorithm. *Email to the end2end-interest mailing list*.

Floyd, S. and Henderson, T. (1999). The NewReno Modification to TCP's Fast Recovery Algorithm, *RFC-2582*.

Floyd, S., Henderson, T., and Gurtov, A. (2004). The NewReno Modification to TCP's Fast Recovery Algorithm, *RFC-3782*.

Mathis, M., Mahdavi, J., Floyd, S., and Romanov, A. (1996). TCP Selective Acknowledgment Options, *RFC-2018*.

Ludwig, R., and Katz, R.H. (2000). The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions, *SIGCOMM Computer Communication Review*, 30(1), 30-36.

Floyd, S., Mahdavi, J., Mathis, M., and Podolsky, M. (2000). An Extension to the Selective Acknowledgement (SACK), *RFC-2883*.

Wang, H., and Shin, K.G. (1996). Robust TCP Congestion Recovery, *Journal of High Speed Networks*, 13(2), 103-121.

Peterson, L.L. and Davie, B.S. (2007). Computer Networks: A Systems Approach, *Elsevier*.

Ludwig, R., Konrad, A., Joseph, A.D. and Katz, R.H. (2002). Optimizing the End-to-end Performance of Reliable Flows Over Wireless Links, *Wireless Networks*, 8(2/3), 289-299.

Acharya, P.A., Sharma, A., Belding, E.M., Almeroth, K.C. and Papagiannaki, K. (2010). Rate Adaptation in Congested Wireless Networks through Real-time Measurements, *IEEE Transactions on Mobile Computing*, 9(11), 1535-1550.

Liu, T.K. and Silvester, J.A. (1999). Congestion Control Policies for Wireless Multimedia CDMA Networks, *Journal of High Speed Networks*, 8(2), 135-147.

Ratnam, K., and Matta, I. (1998). WTCP: An Efficient Mechanism for Improving TCP Performance over Wireless Links, 3^{rd} *IEEE Symposium on Computers and Communications (ISCC'98) Proceedings*, 74-78.

Samaraweera, N.K.G. (1999). Non-congestion Packet Loss Detection for TCP Error Recovery Using Wireless Links, *IEEE Proceedings-Communications*, 146(4), 222-230.

Parsa, C., and Garcia-Luna-Aceves, J.J. (1999). Improving TCP Congestion Control over Internets with Heterogeneous Transmission Media, 7^{th} *International Conference on Network Protocols (ICNP'99)*, 213-221.

Ma, J., Ruutu, J. and Wu, J. (2000). An Enhanced TCP Mechanism-fast-TCP in IP Networks with Wireless Links, *Wireless Networks*, 6(5), 375-379.

Mascolo, S., Casetti, C., Gerla, M., Sanadidi, M.Y. and Wang, R. (2001). TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links, 7^{th} *Annual International Conference on Mobile Computing and Networking, ACM*, 287-297.

Wang, R., Valla, M., Sanadidi, M., Ng, B., and Gerla, M. (2002). Efficiency/friendliness Tradeoffs in TCP Westwood, 7^{th} *International Symposium on Computers and Communications*, 304-311.

Wang, R., Valla, M., Sanadidi, M., Ng, B., and Gerla, M. (2002). Adaptive Bandwidth Share Estimation in TCP Westwood, *IEEE GLOBECOM*, 3, 2604-2608.

Fu, C.P. and Liew, S.C. (2003). TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks, *IEEE Journal on Selected Areas in Communications*, 21(2), 216-228.

Xu, K., Tian, Y., and Ansari, N. (2004). TCP-Jersey for Wireless IP Communications, *IEEE Journal on Selected Areas in Communications*, 22(4), 747-756.

Shi, K., Shu, Y., Yang, O. and Luo, J. (2010). Receiver-assisted Congestion Control to Achieve High Throughput in Lossy Wireless Networks, *IEEE Transactions on Nuclear Science*, 57(2), 491-496.

Biaz, S., and Vaidya, N.H. (2005). "De-randomizing" Congestion Losses to Improve TCP Performance over Wired-wireless Networks, *IEEE/ACM Transactions on Networking*, 13(3), 596-608.

Rath, H.K., Sahoo, A. and Karandikar, A. (2008). A Cross Layer Congestion Control Algorithm in Wireless Networks for TCP Reno-2, *National Conference on Communication*.

Chang, B.J. and Li, Y.H. (2012). Cross-Layer-Based Adaptive TCP Algorithm in 4G Packet Service LTE-Advanced Relaying Communications, *Advances in Intelligent Systems and Applications, Springer, Berlin, Heidelberg*, 1, 451-460.

Fu, B., Xiao, Y., Deng, H., and Zeng, H. (2014). A Survey of Cross-layer Designs in Wireless Networks, *IEEE Communications Surveys & Tutorials*, 16(1), 110-126.

Pham, Q.V., and Hwang, W.J. (2017). Network Utility Maximization-Based Congestion Control Over Wireless Networks: A Survey and Potential Directives, *IEEE Communications Surveys & Tutorials*, 19(2), 1173-1200.

van den Berg, E., Sebuktekin, I., Tauil, M., Ghetie, A. and Alexander, S. (2015). Self Adaptive Robust Resource Allocation for Prioritized TCP Flows in Wireless Networks, *IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, 108-113.

Labovitz, C., Iekel-Johnson, S., McPherson, D., Oberheide, J., Jahanian, F. (2010). Internet Inter-domain Traffic, *ACM SIGCOMM Computer Communication Review*, 40(4), 75-86.

Briscoe, B., Brunstrom, A., Petlund, A., Hayes, D., Ros, D., Tsang, J., Gjessing, S., Fairhurst, G., Griwodz, C., Welzl, M. (2016). Reducing Internet Latency: A Survey of Techniques and Their Merits, *IEEE Communications Surveys & Tutorials*, 18(3), 2149-2196.

Floyd, S. (2003). High Speed TCP for Large Congestion Windows, *RFC-3649*.

Floyd, S. (2003). High Speed TCP and Quick-Start for Fast Long-Distance High Speed TCP and Quick-Start for Fast Long Distance Networks, *TSVWG, IETFF*.

Kelly, T. (2003). Scalable TCP: Improving Performance in Highspeed Wide Area Networks, *ACM SIGCOMM Computer Communication Review*, 33(2), 83-91.

Radhakrishnan, S., Cheng, Y., Chu, J., Jain, A., Raghavan, B. (2011). TCP Fast Open, 7^{th} *Conference on Emerging Networking Experiments and Technologies, ACM*, 21.

Leith, D., and Shorten, R. (2004). H-TCP: TCP for High-speed and Long-distance Networks, *PFLDnet*.

Leith, D. (2008). H-TCP: TCP Congestion Control for High Bandwidth Delay Product, *IETF Internet Draft*.

Caini, C., and Firrincieli, R. (2004). TCP Hybla: A TCP Enhancement for Heterogeneous Networks, *International Journal of Satellite Communications and Networking* 22(5), 547-566.

Xu, L., Harfoush, K., and Rhee, I. (2004). Binary Increase Congestion Control (BIC) for Fast Long-distance Networks, 23^{rd} *Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2004*, 4.

Jin, C., Wei, D., Low, S., Buhrmaster, G., Bunn, J., Choe, D., Cottrel, R., Doyle, J., Feng, W., Martin, O., Newman, H., Paganini, F., Ravot, S., and Singh, S. (2005). FAST TCP: From Theory to Experiments, *IEEE Network*, 19(1), 4-11.

Wei, D.X., Jin, C., Low, S.H., and Hegde, S. (2006). FAST TCP: Motivation, Architecture, Algorithms, Performance, *IEEE/ACM Transactions on Networking*, 14(6), 1246-1259.

Sing, J. and Soh, B. (2005). TCP New Vegas: Improving the Performance of TCP Vegas over High Latency Links, 4^{th} *IEEE International Symposium on Network Computing and Applications*, 73-82.

Shimonishi, H., and Murase, T. (2005). Improving Efficiency-friendliness Tradeoffs of TCP Congestion Control Algorithm, *Global Telecommunications Conference, GLOBECOM'05*, 1, 5.

Guan, X., Yang, B., Long, C., and Liu, Z. (2005). Improving Efficiency-friendliness Tradeoffs of TCP Congestion Control Algorithm, *Journal of High Speed Networks*, 14(3), 215-226.

Kaneko, K., Fujikawa, T., Su, Z., and Katto, J. (2007). TCP-Fusion: A Hybrid Congestion Control Algorithm for High-speed Networks, *PFLDnet*, 7, 31-36.

Grigorik, I. (2018...). High Performance Browser Networking: What every web developer should know about networking and web performance, *O'Reilly Media Inc. (https://hpbn.co/)*.

Flach, T., Dukkipati, N., Terzis, A., Raghavan, B., Cardwell, N., Cheng, Y., Jain, A., Hao,S., Katz-Bassett, E., Govindan, R. (2016). Reducing WEB Latency: The Virtue of Gentle Aggression, *IEEE Communications Surveys & Tutorials*, 18(3), 2149-2196.

De Cicco, L., Carlucci, G. and Mascolo, S. (2013). Understanding the Dynamic Behavior of the Google Congestion Control for rtcweb, 20^{th} *International Packet Video Workshop*, 1-8.

Chromium (2016). QUIC, A Multiplexed Stream Transport over UDP, *https://www.chromium.org/quic*.

Gizis, A. (2016). Taking Google's QUIC for a Test Drive, *http://www.connectify.me/taking-googlequic-for-a-test-drive/.*

Wei S., and Swaminathan, V. (2014). Low Latency Live Video Streaming over HTTP 2.0, *ACM, Network and Operating System Support on Digital Audio and Video Workshop, NOSSDAV'14*, 37:37-37:42.

Barakat, C., Thiran, P., Iannaccone, G., Diot, C., Owezarski P. (2003). A Flow-based Model for Internet Backbone Traffic, *IEEE Transactions on Signal Processing - Special Issue on Signal Processing in Networking*, 51(8), 2111-2124.

Oueslati, S. and Roberts, J. (2005). A New Direction for Quality of Service: Flow Aware Networking, *Next Generation Internet Networks*, 226-232.

Kortebi, A., Queslati, S., Roberts, J. (2004). Cross-protect: Implicit Service Differentiation and Admission Control, *HPSR'04, Phoenix*.

Fred, S.B., Bonald, T., Proutiere, A., Régnié, G. and Roberts, J.W. (2001). Statistical Bandwidth Sharing: A Study of Congestion at Flow Level, *ACM, SIGCOMM Computer Communication Review*, 31(4), 111-122.

Claeys, M., Bouten, N., De Vleeschauwer, D., De Schepper, K., Van Leekwijck, W., Latré, S. and De Turck, F. (2016). Deadline-aware TCP Congestion Control for Video Streaming Services, 12^{th} *International Conference on Network and Service Management (CNSM)*, 100-108.

Dong, M., Li, Q., Zarchy, D., Godfrey, P.B. and Schapira, M. (2015). PCC: Re-architecting Congestion Control for Consistent High Performance. 12^{th} *USENIX Symposium on Networked Systems Design and Implementation (NSDI*, 395-408.

Wang, X.S., Balasubramanian, A., Krishnamurthy, A. and Wetherall, D. (2014). How Speedy is SPDY?. 11^{th} *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 387-399.

Vernersson, A. (2015). Analysis of UDP-based Reliable Transport using Network Emulation. *Master thesis, Luleå University of Technology*.

Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C., Zhang, D., Yang, F., Kouranov, F., Swett, I., Iyengar, J. and Bailey, J. (2017). The QUIC Transport Protocol: Design and Internet-scale Deployment. *ACM Special Interest Group on Data Communication*, 183-196.

Iyengar, J., Swett, I. (2015). QUIC Loss Recovery And Congestion Control draft-tsvwg-quic-loss-recovery-01, *IETF, draft-tsvwg-quic-protocol-01*.

Swett, I. (2015). QUIC: Congestion Control and Loss Recovery.

Iyengar, J. (2016). QUIC: Redefining Internet Transport.

Iyengar, J. and Thomson, M. (2017). QUIC: A UDP-Based Multiplexed and Secure Transport. *draft-ietf-quic-transport-01*.

Chromiumblog. (2015). Chromium blog: A QUIC Update on Google's Experimental Transport. *http://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html*.

Carlucci, G., De Cicco, L., Mascolo, S. (2015). HTTP over UDP: An Experimental Investigation of QUIC, 30th *Annual ACM Symposium on Applied Computing, SAC'15*, 609-614.

Megyesi, P., Kramer, Z., Molnar, S. (2016). How Quick is QUIC?, *IEEE International Conference on Communications (ICC)*.

Biswal, P. and Gnawali, O. (2016). Does QUIC Make the WEB Faster?. *IEEE Global Communications Conference, GLOBECOM*, 1-6.

Das, S.R. (2014). Evaluation of QUIC on Web Page Performance. *Doctoral dissertation, Massachusetts Institute of Technology*.

Cook, S., Mathieu, B., Truong, P. and Hamchaoui, I., (2017). QUIC: Better for What and for Whom?. *IEEE International Conference on Communications (ICC)*, 1-6.

Srivastava, A. (2017). Performance Evaluation of QUIC Protocol under Network Congestion. *Masters Theses (All Theses, All Years), https://digitalcommons.wpi.edu/etd-theses/220*.

Kakhki, A.M., Jero, S., Choffnes, D., Nita-Rotaru, C. and Mislove, A. (2017). Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols. *IMC'17*.

Duan, Y., Gallo, M., Traverso, S., Laufer, R. and Giaccone, P. (2017). Towards a Scalable Modular QUIC Server. *ACM Workshop on Kernel-Bypass Networks*, 19-24.

De Coninck, Q. and Bonaventure, O. (2017). Multipath QUIC: Design and Evaluation. *ACM* 13th *International Conference on Emerging Networking EXperiments and Technologies*, 160-166, ACM.

Hussein, A., Kayssi, A., Elhajj, I.H. and Chehab, A., (2018). SDN for QUIC: An Enhanced Architecture with Improved Connection Establishment. 33rd *Annual ACM Symposium on Applied Computing*, 2136-2139.

Flach, T., Papageorge, P., Terzis, A., Pedrosa, L., Cheng, Y., Karim, T., Katz-Bassett, E. and Govindan, R., (2018). An Internet-wide Analysis of Traffic Policing. *ACM SIGCOMM Conference*, 468-482.

Ross, S.M., Kelly, J.J., Sullivan, R.J., Perry, W.J., Mercer, D., Davis, R.M., Washburn, T.D., Sager, E.V., Boyce, J.B. and Bristow, V.L., (1996). Stochastic Processes. *Wiley, New York*.

Persson, A., Marcondes, C.A., Chen, L.J., Sanadidi, M.Y. and Gerla, M., (2005). TCP Probe: A TCP with Built-in Path Capacity Estimation. 8^{th} *IEEE Global Internet Symposium*.

Speed Test Network Monitoring Tool. (2017). *http://beta.speedtest.net/0*.

Kharat, P.K. and Kulkarni, M., (2018). Situation-Based Congestion Control Strategies for Wired and Wireless Networks. *IEEE, In 2018 International Conference On Advances in Communication and Computing Technology (ICACCT)*, 245-250).

QUIC Test Server. (1996). *http://www.chromium.org/quic/playing-with-quic*.

Eryilmaz, A. and Srikant, R. (2007). Fair Resource Allocation in Wireless Networks using Queue-length-based Scheduling and Congestion Control, *IEEE/ACM Transactions on Networking*, 15(6), 1333-1344.

Molnar, S., Sonkoly, B., and Trinh, T.A. (2009). A Comprehensive TCP Fairness Analysis in High Speed Networks, *Computer Communications*, 32(13), 1460-1484.

Muhammad Adil Raja (2017). The Network Layer, *https://www.slideshare.net/madilraja/the-network-layer-38487922*.

Jain, R.K., Chiu, D.M.W. and Hawe, W.R. (1984). A Quantitative Measure of Fairness and Discrimination. *Eastern Research Laboratory, Digital Equipment Corporation: Hudson, MA, USA*, 2-7.

Sandvine, I., (2016). Global Internet Phenomena Report. *North America and Latin America*.

Blog, C., (2015). A QUIC Update on Google's Experimental Transport.

Gupta, A. and Mukherjee, S., (2018). The Empirical Study on The Impact of Social Media Marketing on Brand Loyalty: Case Study of Reliance Jio Infocom Limited (with Reference to Silchar Region). *Global Journal of Business Intelligence and Management Insight & Transformations*, 2(1).

Langley, A. and Chang, W., (2013). QUIC Crypto.

Mathis, M., Semke, J., Mahdavi, J. and Ott, T., (1997). The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3), 67-82.

Journal Publications

1. **Prashant Kharat** and Muralidhar Kulkarni (Feb, 2019), "Congestion controlling schemes for high-speed data networks: A survey", *Journal of High Speed Networks, Vol. 25(2019), pp. 41-60*.

2. **Prashant Kharat** and Muralidhar Kulkarni (March, 2019), "Modified QUIC protocol for improved network performance and comparison with QUIC and TCP", *International Journal of Internet Protocol Technology, Vol. 12, No. 1, pp. 35-43*.

3. **Prashant Kharat** and Muralidhar Kulkarni (under review), "ModQUIC: A Modified QUIC Protocol to Improve Network Performance", *IEEE/ACM Transactions on Networking*.

4. **Prashant Kharat** and Muralidhar Kulkarni (communicated), "ModQUIC Protocol Performance Verification with CUBIC and BBR Congestion Control Mechanisms", *International Journal of Internet Protocol Technology, Inderscience*.

5. **Prashant Kharat** and Muralidhar Kulkarni (under review), "Congestion Control Performance Investigation of ModQUIC Protocol using Jio-Fi Network: A Case Study", *Journal of High Speed Networks*.

Conference Papers

1. **Prashant Kharat** and Muralidhar Kulkarni (Feb, 2018), "Situation-Based Congestion Control Strategies for Wired and Wireless Networks", *Proceeding of International Conference on Advances in Communication and Computing Technology (ICACCT), IEEE, pp. 245-250*.

2. **Prashant Kharat**, Aniket Rege, Aneesh Goel and Muralidhar Kulkarni (April, 2018), "QUIC Protocol Performance in Wireless Networks", *Proceeding of* 7^{th} *International Conference on Communication and Signal Processing (ICCSP'18), IEEE, pp. 476-480*.

3. **Prashant Kharat**, Shantanu Vijay, Jayanth Putta, Apurva P. and Muralidhar Kulkarni (March, 2019), "ModQUIC Performance Analysis with Dynamic Packet Fragmentation in Wireless Networks", *Proceeding of* 8^{th} *International Engineering Symposium (IES 2019), Kumamoto University, Japan*.

Bio-data

Name : Kharat Prashant Krishnarao

Address : Walchand College of Engineering,
Vishrambag, Sangli, Maharashtra

Email : prashant.kharat@walchandsangli.ac.in

Mobile : 9764636875

Date of Birth : 25^{th} May 1979

Educational Qualification & Teaching Experience:

Educational Qualification		
Degree	Year of Passing	University
B.E.	2001	Shivaji University, Kolhapur
M.E.	2007	Shivaji University, Kolhapur
Ph.D.	2016 to 2019	NITK, Surathkal
Teaching Experience		
Institute	Post	Period
ADCET, Ashta, Sangli	Lecturer	2002 to 2007
DKTE's, TEI, Ichalkaranji	Assistant Professor	2007 to 2009
Walchand College of Engineering, Sangli	Assistant Professor	2009 to till date