# UC Santa Cruz

**UC Santa Cruz Electronic Theses and Dissertations** 

# Title

Improving TCP Using Working Memory Capacity and Network Coding

**Permalink** https://escholarship.org/uc/item/1x25m97s

**Author** Srinivasan, Ramesh

Publication Date

Peer reviewed|Thesis/dissertation

# UNIVERSITY OF CALIFORNIA SANTA CRUZ

### IMPROVING TCP USING WORKING MEMORY CAPACITY AND NETWORK CODING

A dissertation submitted in partial satisfaction of the requirements for the degree of

### DOCTOR OF PHILOSOPHY

in

## COMPUTER ENGINEERING

by

#### Ramesh Srinivasan

December 2022

The Dissertation of Ramesh Srinivasan is approved:

Prof. J.J. Garcia-Luna-Aceves, Chair

Prof. Martine Schlag

Prof. Brad R. Smith

Peter Biehl Vice Provost and Dean of Graduate Studies Copyright © by

Ramesh Srinivasan

2022

# Table of Contents

List of Figures						
A	bstra	ct		vii		
D	edica	tion		ix		
A	ckno	wledgn	nents	x		
1	Introduction					
	1.1	5G Te	chnology Use Cases	4		
	1.2	5G Te	chnology Data Transport Requirements	9		
	1.3	Resear	cch Contributions	10		
	1.4	Thesis	Outline	12		
<b>2</b>	Related Work					
	2.1	Survey	of TCP Variants	16		
	2.2	Limita	ations of TCP Prior Variants	20		
	2.3	TCP I	Enhancements for Hybrid Networks	21		
	2.4	TCP I	Enhancements with Network Coding	28		
	2.5	TCP I	Enhancements for Dynamic Topology Changes	30		
3	TC	P-EWS	SC:			
	TC	P Enha	anced-Wireless-Santa-Cruz	<b>32</b>		
	3.1	Basic	$Operation \ldots \ldots$	34		
	3.2	TCP-I	EWSC Description	36		
		3.2.1	Enhanced BEST-START Window Size	36		
		3.2.2	Enhanced RTT Correction: Link-Layer Retransmissions	44		
		3.2.3	Enhanced Fast-Retransmission: Link-Layer Reordering	45		
		3.2.4	RSSI Enhancement: Receiver Zero-Window Buffer Spoof	47		
		3.2.5	Enhanced Cluster Analysis	47		
	3.3	Perfor	mance Results	48		
	3.4	Perfor	mance Comparison	51		

	3.5	Summary	52				
4	TCP-NEWT:						
	TCP Network-Coding Enhanced Window Transformation						
	4.1	Basic Operation	55				
	4.2	Protocol Description	59				
	4.3	Performance Results	81				
	4.4	Performance Comparison	84				
	4.5	Summary	85				
<b>5</b>	TC	TCP-PNC:					
	TC	P Predictive Network Coding	86				
	5.1	Basic Operation	86				
	5.2	Protocol Description	92				
	5.3	Performance Results	100				
	5.4	Performance Comparison	101				
	5.5	Summary	104				
6	TC	TCP-RTA:					
	TCP Real-Time Topology Adaptiveness for Congestion Control						
	6.1	Basic Operation	107				
	6.2	Protocol Description	108				
	6.3	Performance Results	115				
	6.4	Performance Comparison	117				
	6.5	Summary	118				
7	Cor	nclusion	120				
Bi	bliog	graphy	123				

# List of Figures

1.1	5G Network Deployment Scenarios. [69]	4				
1.2	Mobile Cars with Multiple Cell Towers. [70]	5				
1.3	Multiple Communication Satellites in Space [85]					
1.4	Customer Physical Location Direct Drone Delivery [8] 6					
1.5	Military Communications with Ad-hoc Wireless Networks [71]	7				
1.6	Disaster Zone Emergency Rescue Crew's Communication Needs [22] 7					
1.7	IoT 5G Deployments [93]					
1.8	5G Military Deployment Scenario - Anglova [30]	9				
3.1	TCP-Cubic with No-Loss	48				
3.2	TCP-Cubic with $10\%$ Loss	49				
3.3	TCP-Cubic with $20\%$ Loss	49				
3.4	TCP-NewReno with No-Loss	50				
3.5	TCP-NewReno with $10\%$ Loss	50				
3.6	TCP-NewReno with 20% Loss	51				
4.1	TCP-NEWT header	55				
4.2	Network Coding	59				
4.3	Group Size 1	60				
4.4	Original Sample Data Seg 1 - Hex Dump	61				
4.5	Group Size 1: Coded Datagram 8.1's TCP Header	61				
4.6	Group Size 2: CE1-2 Coded Datagram 8.1's Computation by bit shifting	62				
4.7	Group Size 1: CE1-2 Coded Datagram 8.1's TCP Data	62				
4.8	Group Size 1: Coded Datagram 8.2's Relevant TCP Header fields	62				
4.9	Group Size 2: CE1-2 Coded Datagram 8.1's Computation by bit shifting	63				
4.10	Group Size 1: CE1-4 Coded Datagram 8.1's TCP Header	63				
4.11	Group Size 1: Coded Datagram 8.3's Relevant TCP Header fields	63				
4.12	Group Size 2: CE1-2 Coded Datagram 8.1's Computation by bit shifting	64				
4.13	Group Size 1: CE1-8 Coded Datagram 8.3's TCP Header	64				
4.14	Group Size 1: Coded Datagram 8.4's Relevant TCP Header fields	64				
4.15	Group Size 2: CE1-16 Coded Datagram 8.1's Computation by bit shifting	65				

4.16	Group Size 1: CE-16 Coded Datagram 8.1's TCP Header	65
4.17	Group Size 1 RX	65
4.18	Group Size 1:RTTVAR and SRTT parameters mapping	67
4.19	Group Size 2	68
4.20	Original Data Segment 2- Hex Dump	69
4.21	Group Size 2: Coded Datagram 8.1's TCP Header	69
4.22	Group Size 2: CE1-2 Coded Datagram 8.1's TCP Data	70
4.23	Group Size 2: Coded Datagram 8.2's TCP Header	70
4.24	Group Size 2: CE1-2 Coded Datagram 8.1's TCP Data	70
4.25	Group Size 2 RX	71
4.26	Group Size 2: CE1-2 Coded Datagram 8.1's TCP Data	71
4.27	Group Size 2: D1 obtained by RLC2 - RLC1 is divided by 2	72
4.28	Group Size 2:	72
4.29	Distributed Object Storage with Network Coding	75
4.30	Distributed Object Storage with Selective Network Coding	76
4.31	Distributed Object Storage with Compressed Network Coding	77
4.32	TCP Network Coding with No-Loss	81
4.33	TCP Network Coding with 10% Loss	82
4.34	TCP Network Coding with 20%Loss	82
4.35	TCP Network Coding with 30%Loss	83
4.36	TCP Network Coding with 40%Loss	83
4.37	TCP Network Coding with 50%Loss	84
5.1	Native TCP based Selective Network Coding	89
5.2	TCP-NEWT with Selective Network Coding	90
5.3	TCP-NEWT congestion window comprises only coded segments	91
5.4	LL Goodput Computation	95
5.5	An Example of Dynamic Goodput (1 - Loss-Ratio) Prediction	97
5.6	Cubic vs New Reno vs NC	102
5.7	NC - Throughput	103
5.8	Cubic vs New Reno vs NC - 20% loss	103
6.1	CWND: Adaptive vs Hybla vs NewBeno	116
6.2	TX: Adaptive vs Hybla vs NewBeno	116
6.3	CWND before transition to satellite: Adaptive vs Hybla vs NewBeno	116
6.4	TX before transition to satellite: Adaptive vs Hybla vs NewBeno	116
6.5	CWND after transition to Satellite link: Adaptive vs Hybla vs NewReno	117
6.6	TX after transition to Satellite link: Adaptive vs Hybla vs NewBeno	117
6.7	CWND after transition from Satellite link: Adaptive vs Hybla vs NewBen	5117
6.8	TX after transition from Satellite link: Adaptive vs Hybla vs NewReno	117
		•

#### Abstract

# IMPROVING TCP USING WORKING MEMORY CAPACITY AND NETWORK CODING

by

#### Ramesh Srinivasan

This thesis introduces a number of modifications and enhancements to the Transmission Control Protocol (TCP) aimed at addressing some of the known limitations of TCP taking advantage of larger amounts of working memory capacity at the nodes participating in TCP sessions, and network coding. These TCP variants are TCP-EWSC (TCP Enhanced Wireless Santa Cruz), TCP-NEWT (TCP Network-Coding Enabled Window Transformation), TCP-PNC (TCP Predictive Network Coding) and TCP-RTA (TCP Real-time Topology Adaptiveness). TCP-EWSC brings several capabilities to better adapt traditional TCP for hybrid networks including the capability to respond to sporadic temporary wireless signal outages in a resilient manner with proactive spoofing of receiver zero-window. TCP-NEWT introduces a mechanism for network coding to proactively address packet loss without retransmissions while accurately reporting all the observed TCP flow metrics back to the host. TCP-PNC augments TCP-NEWT with real-time prediction of the expected goodput (packets delivered/total packets sent) and proactive compensation for the same. TCP-RTA dynamically detects a topology change and adapts with an appropriate congestion-control strategy to maximize the effective use of the total available bandwidth. TCP-RTA also factors in the relative impact of the degree of change in topology versus that of the underlying transmission medium to identify the best congestion control strategy to ensure optimal usage of the end-to-end network infrastructure. TCP-EWSC performance results show that end-end throughput is sustained close to the prior levels even after the introduction of link-layer errors, when compared to other TCP variants which show a drop of about 8%. Performance results with TCP-PNC with network coding, show an overall increase of end-to-end TCP throughput in the range of 30-35% depending upon the number of wireless-link-layer losses, without changing the underlying "network congestion". Performance results with TCP-RTA indicate a throughput increase of more than 35% in scenarios involving dynamic topology changes during a TCP session.

#### Dedication

To my wife Srividya Ramesh for her ongoing support through this endeavor and keeping me focused on making progress and bringing a constant refreshing sense of humor and cheer along this journey.

To my parents Dr. Rajalakshmi Srinivasan and K.S. Srinivasan for their unceasing belief and faith in me through the ups and downs and always being there with boundless encouragement and inspiration.

To my daughters Shreya Ramesh and Sathya Ramesh for adjusting with my many hours of research and getting involved in technical discussions and getting an understanding of the areas where I have been spending late nights and weekends. Shreya also helped with detailed review of several chapters of this thesis as well as with the presentation. Sathya was constantly egging me on to stay focused and helped with finishing touches and edits with the presentation.

To my sister Dr. Jayanti Ravi excelling in many walks of her life and always having a cheerful disposition and encouraging me during our conversations.

To my maternal grandfather late K. Subramanian for the countless hours, he spent working closely with me decades ago when I was in high school getting introduced to science and math. He instilled in me a passion and quest for knowledge and the belief that one can make anything happen if our heart and soul is put into it.

To my maternal uncle late S. Viswanathan my local guardian through my undergraduate years at IIT-Madras and a pillar of support in my formative years.

#### Acknowledgments

Returning to graduate school as a doctoral student while also working in the industry was not a trivial undertaking. This thesis could not have been completed without the help, guidance, and support from my advisor Professor J. J. Garcia Luna-Aceves, whose technical guidance coupled with his mild and calm demeanor and subtle sense of humor, served as a tremendous source of motivation and inspiration. I would like to thank Professor Martine Schlag and Professor Brad Smith, who found time to serve as committee members and provided valuable feedback despite their busy schedules as well as Professor Marco Rolandi, who was part of my advancement committee. I would also like to thank Professor Brad Smith for working with me in a research proposal submission to Cisco Systems few years ago as well as Professor Katia Obraczka and Professor Cedric Westphal, who provided the intellectual stimulation during the many discussions in the Networking Seminar class over the years. I also would like to thank Professor Suresh K. Lodha for the warm welcome email I received, when applying for the doctoral program. I would also like to place on record my deep gratitude to Professor Pat Mantey, founding Dean of the Jack Baskin School of Engineering (SOE), for encouraging me to join UCSC. I would like to acknowledge my colleagues at CCRG over the years, James Mathewson, Nitish John, Dhananjay Sampath, Duy Nguyen, Turhan Karadeniz, Spencer Sevilla, Ehsan Hemmati, Hari Potaraju, Yali (Celina) Wang, Ashok Masilamani, Dylan Cirimelli-Low, among others, who have been incredibly supportive. The primary inspiration for my foray into academia and networking area has been my experience in networking industry and I would like to hereby acknowledge the tremendous inspiration from my ongoing networking industry experience. Starting in 1993, worked initially on development of power series communication controller card for Silicon Graphics' (SGI) Crimson machine. Thereafter, worked at networking giant Cisco Systems on every low-end access router, starting from Lan-Extender Cisco-1001 up to and including the ISR (Integrated Services Router) Cisco-1841 and Cisco-2801. I was also involved in incorporating wireless interface support on Access routers and was the architect of TCP-Receiver Zero window Spoofing, a Cisco Internetworking Operating System (IOS) feature, developed originally for the MAR (Mobile Access Router) Cisco-3200 Series Platform. I would like to thank several of my colleagues at Cisco Systems, where I worked for 14 years, including Steve Canby, Somnath Mitra, Mick Henniger, David Coli (all from erstwhile Access Business Unit) and Anantha Ramaiah (IOS Networking Group) for several engaging inspiring technical discussions. Special thanks to my team members at my startup company, Mayosys Solutions for their understanding and support of my commitment to the doctoral program.

I would like to thank Carol Mullane, Emily Gregg, Lisa Stipanovich, Thomas Cahoon and Theo-Alyce Gordon in the Graduate Student Advising office, who have been so amazing in coordinating everything.

# Chapter 1

# Introduction

The Internet is a global system of interconnected computer-networks that uses the standard Transmission Control Protocol/Internet Protocol (TCP/IP) suite to link billions of devices worldwide. It is a network of networks that consists of millions of private, public, academic, business, and government networks of local to global scope, linked by a broad array of electronic, wireless, and optical-networking technologies. The Internet carries an extensive range of informational resources and services, such as the inter-linked hypertext-documents and applications of the World Wide Web (WWW), the infrastructure to support email and peer-to-peer networks for file sharing and telephony.

In today's evolving technological world, with its ongoing continuous evolution comprising miniaturization of computer technology and increasing processor speeds, the rapid adoption and maturing of various wireless technologies, including the 802.11 a/b/c/n technologies and subsequently 60 GHz Wi-Fi 802.11ad and 802.11ay, 802.11ac, 802.11ax (IEEE 802.11 Wi-Fi standards for Wireless LANs, WLANs) for high speed local-wireless LAN communication over a Gigabit/sec speed coupled with the 5G technology coming of age and its accelerated adoption and global roll-out, has made wireless technology ubiquitous. There are innumerable numbers of wireless laptops, notebooks, tablets, smart-phones, notepads as well as a plethora of IoT (Internet of Things) devices and several mobile end-to-end devices, including but not limited to mobile cars, drones, satellites, and smart-phones carried by "mobile" people. Their numbers are growing at a rate never seen in the IT (Information Technology) industry and they are all connecting to the Internet through the last-mile wireless interface. Thus, the Internet today is a hybrid network with a substantial proliferation of the number of the last-mile-wireless links coexisting with the core of the internet using predominantly wired technologies. Additionally, certain back-haul networks are wireless and some of them use geostationary satellites for connectivity using wireless Wide Area Network (WAN) technologies. These trends make it even more important for us to consider the Internet as a hybrid network. Therefore, there is a need to take a fresh look into the design of underlying protocols and propose appropriate new, modified and/or enhanced protocols to meet the requirements and challenges of the new evolving Internet based off 5G technologies and beyond.

For convenience, Table 1.1 lists the abbreviations and definitions used in this thesis.

Abbreviations	Definitions
802.11 a/b/c/n/ad/ay/ac/ax	IEEE 802.11 Wi-Fi standards for Wireless LANs (WLANs)
ACK	Acknowledgment of receipt of TCP segment
BER	Bit Error Rate
CCN	Content Centric Network
CD <sub>i</sub>	Coded_Datagram <sub>i</sub> in TCP-NEWT window
CE1, CE2CE16	Random Linear Coefficient RLC1 RLC2 RLC16
Coded-Grp-size	Number of coded segments generated from the group in Original TCP Window
CWD/Cwnd	TCP Congestion Window
CWR	Congestion Window Reduced, acknowledgment congestion-indication echo received.
Di	Datagram <sub>i</sub> in Original TCP window
DUP-ACKS	Duplicate Acknowledgments, ACKs with same sequence number
ECE	ECN Echo
ECN	Explicit Congestion Notification - a TCP packet extension defined in RFC 3168
FEC	Forward Error Correction
Goodput	Packets Delivered/Total Packets Sent
grp_sz	Number of segments in a group in Original TCP Window
ICN	Information Centric Network
IEEE	Institute of Electrical and Electronics Engineers
IT	Information Technology
IoT	Internet of Things
LL	payLoad deLivery ratio: (packets-delivered/packets-sent)
Loss-Ratio	Packets Dropped/Total Packets Sent: (1 - goodput)
LR	Loss Ratio
M2M	Machine to Machine
NC	Network Coding
Orig-Grp-size	Number of segments in a group in Original TCP Window
OSI	Open Systems Interconnection model with TCP in layer 4
PDA	Personal Digital Assistant (hand-held PC)
R	Initial Round Trip Time Measurement
	Next Round Trip Time Measurement
RLC	Random Linear Coded
RLC (Coefficients)	Random Linear Coefficients
RLC grp packets	Random Linear Coded packets of a group
RSSI	Wireless Received Signal Strength Indicator
RTO	Retransmission Timeout
RTT	Round Trip Time (implies end-end)
RTTVAR	Round Trip Time Variation
RTTVAR <sub>8</sub>	Round Trip Time Variation for Group 8 in original TCP window
RTTVAR <sub>8.1</sub>	Round Trip Time Variation for CD8 <sub>1</sub> in TCP-NEWT
SLA	Service Level Agreement
SMSS	SENDER MAXIMUM SEGMENT SIZE
SRTT	Smoothed Round Trip Time
SRTT <sub>8</sub>	Smoothed Round Trip Time for Group 8 in original TCP window
SRTT <sub>8.1</sub>	Smoothed Round Trip Time for CD8 <sub>1</sub> in TCP-NEWT
SSCA-Multiplicative Factor	Multiplicative factor of SSCA-Window-size, default 0.5, permitted range [0.3 to 0.7]
SSCA-window-size	Window size at time of change from Slow Start to Congestion Avoidance phase
TCP/IP	Transmission Control Protocol/Internet Protocol
TcpWin-Start-sz-default	Default TCP Window Size at the start of a TCP session
WAN	Wide Area Network
Wi-Fi	Wireless Fidelity (as defined by IEEE)
Wi-Fi	WLAN products based on the IEEE 802.11 standards (as defined by WiFi Alliance)
WLAN	wireless local area network
WLR	Worst-case Loss Ratio

Table 1.1: Abbreviations and Definitions

# 1.1 5G Technology Use Cases

With the advent and adoption of 5G technology and the proliferation of mobile end-nodes, secure, reliable, and real-time data delivery has become a critical requirement. 5G technologies are maturing and deployments are currently underway, with several carriers having announced and initiated their roll-out plans and offering related services to customers. The need for an enhanced transport mechanism, incorporated during the 5G roll-out phase itself, is compelling.



Figure 1.1: 5G Network Deployment Scenarios. [69]

A sizable number of use-cases are presented that mandate the need for an alternate enhanced data transport mechanism, capable of addressing the newly identified requirements.



Figure 1.2: Mobile Cars with Multiple Cell Towers. [70]

As the cars and vehicles move, they would need to transition from one cell tower to another. Today's automated cars using 5G technology have a compelling requirement for real-time guaranteed delivery of data within a prescribed upper-bound time limit.



Figure 1.3: Multiple Communication Satellites in Space [85]

In many remote areas with no direct cell-tower coverage, communication via satellites is the only option. Fortunately, every nook and corner of the planet is connected with some satellite or the other at any given moment in time. This becomes relevant for mobile vehicles travelling through remote areas, which need sustained optimal network connection right through the hand-off phase, from a cell-tower with direct 5G access to satellite based network connectivity.



Figure 1.4: Customer Physical Location Direct Drone Delivery [8]

Amazon has already filed several patents [57], for direct consumer orders delivery by drones [47, 62, 63, 8]. Maintaining the SLA (Service Level Agreement) constraints during hand-off from cell-tower to cell-tower during the drones travel and transit is another use-case mandating the need for real-time guaranteed delivery of data.



Figure 1.5: Military Communications with Ad-hoc Wireless Networks [71]

Military personnel in a warzone with no guaranteed cell-tower infrastructure have a constant need to be able to continue to communicate immediately and reliably. This is always necessary including transition phases, when there is a link change in the ad-hoc network, due to the inherent ongoing mobility of all the nodes, comprising of vehicles and personnel.



Figure 1.6: Disaster Zone Emergency Rescue Crew's Communication Needs [22]



A similar need is seen with broken network infrastructure in disaster zones.

Figure 1.7: IoT 5G Deployments [93]

Deployment scenarios including Anglova [30] require continuous availability of services. Service providers need to attain the most efficient use of the available bandwidth over wired or wireless links with more end users being mobile. Hence, the original approach used in TCP of interpreting increases in delay as a sign of congestion must be revisited to account for the fact that a given TCP session may use diverse types of transmission media as end users move.



Figure 1.8: 5G Military Deployment Scenario - Anglova [30]

### 1.2 5G Technology Data Transport Requirements

The current observed scale of development and deployment trends in the IoT [58] and M2M (Machine to Machine) [73] communications brings an additional set of requirements in the heterogeneous 5G hybrid networks [20].

The requirements include (1) continuous availability of wireless technology for last-mile connectivity, (2) ability to transport data through the wired networks without throughput degradation for non-congestion issues, (3) ability to transport data through back-haul wireless and satellite networks to reach remote places with no wired infrastructure, (4) capability to transport data securely in real-time, while ensuring the most efficient use of the available bandwidth, by use of dynamic adjustment to the transient vagaries of the network infrastructure, and (5) detection of dynamic topology changes during an established TCP session and enabling suitable responses to ensure efficient use of the network infrastructure. All of the above mandate an urgent need for augmenting and/or replacing the original Internet design by Cerf and Kahn [18], divided into the IP [86] and TCP [68]. The new transport protocols must be capable of performing efficiently at scale in evolving hybrid 5G networks and beyond, which support mobile nodes as well.

### **1.3** Research Contributions

This thesis provides novel mechanisms in the context of TCP aimed at addressing the new challenges and requirements in evolving 5G networks and beyond with mobile end-nodes, requiring secure reliable data delivered in real-time. This is accomplished by allowing the network infrastructure to proactively identify potential topology changes as well as link-errors and dynamically adapt to an appropriate congestion control mechanism. This can ensure better utilization of the network infrastructure. With augmented proactive goodput (1 - loss-ratio) prediction, real-time data delivery is enabled through the use of optimal network coding. The performance results presented in this thesis clearly establish that the innovative approaches being proposed help meet all these objectives.

TCP-EWSC brings several capabilities to better adapt traditional TCP for hybrid networks including the following novel contributions: the BEST-START window size approach to arrive at the initial window size for a new TCP session, as proposed in TCP-WSC [77], is enhanced by a more detailed learning mechanism specific to each TCP session, arriving at "Enhanced BEST-START" starting window size from the statistically computed expected "steady state window size". Compensation for link-layer retransmission times, in the forward and the acknowledgment return path in the actual end-end RTT (Round Trip Time) values communicated to the TCP congestion control layer so that TCP's interpretation of RTT variations as network congestion works correctly, is another new contribution. Fast retransmission enhancement by initiating it once the ack (acknowledgment) is delayed more than twice the observed standard deviation from the observed mean of the RTTs for the current session. Ensuring receipt of 3 DUP-ACKs (duplicate acknowledgments) does not inadvertently trigger TCP segment retransmission, specifically when one of the packets encountered link-layer retransmissions resulting in subsequent re-ordering of the transmission of packets from the link-layer. Another novel contribution in this work is constant monitoring of the RSSI (Wireless Received Signal Strength Indicator) and initiation of proactive spoofing of 'receiver zero window" acknowledgment by layer two, so that the TCP session is not torn down just because of a sporadic transient wireless-link-layer issue. TCP-EWSC enhances the cluster analysis, as proposed in TCP-WSC [77], to ignore any one-off RTTs (outliers). They are not considered in the dynamic ongoing SRTT (Smoothed Round Trip Time) and RTTVAR (Round Trip Time Variation) computations, which in turn impact the RTO (Retransmission Timeout).

TCP-NEWT introduces a new mechanism for network coding to proactively address any potential packet loss without retransmissions while still being able to accurately report all the observed TCP flow metrics back to host and this is a novel contribution. TCP-PNC's [87] innovative contribution involves augmenting TCP-NEWT with dynamic prediction of the expected goodput and initiating apt amount of proactive compensation for the same.

TCP-RTA [88] dynamically detects a topology change and adapts with an apt topology specific congestion-control strategy to maximize the effective use of the total available bandwidth in real-time. This is a unique approach, and the protocol also factors in the relative impact of the degree of change in topology versus that of the underlying transmission medium to identify the best congestion control strategy, to ensure optimal usage of the end-to-end network infrastructure for any TCP session.

### 1.4 Thesis Outline

This thesis presents a number of TCP variants that improve on the performance of TCP taking advantage of network coding and the use of larger amounts of working memory capacity available at communicating nodes.

Chapter 2 surveys prior TCP enhancements introduced to improve the performance of TCP in different scenarios and discusses their inability to meet the previously identified requirements.

Chapter 3 presents **TCP-EWSC** (*TCP-E*nhanced *W*ireless *S*anta *C*ruz). This TCP variant includes several enhancements to TCP-WSC (TCP Wireless Santa Cruz) [77] which address the following issues comprehensively: The transmission quality of wireless channels is characterized by bursty errors with high varying BER (Bit Error Rate). It is further degraded for those channels with wireless mobile end-nodes, due to the node's mobility-induced wireless signal strength fluctuations, which is compounded during the hand-off from one access point to another [59]. It is well known that TCP does not work efficiently with mobile end-devices connected by wireless links because wireless links bring varying signal-strength and result in dynamically changing capacity. In some cases, mobile end-devices experience scenarios with sporadic breakage of wireless-linklayer last-mile communications. The initial design of TCP was designed to interpret packet losses as a sign of network congestion, which is a good indicator in a network with a static topology and reliable links but fails to account for the nature of the physical-layer of wireless links. TCP-EWSC's proposed enhancements to TCP address this limitation by proactively detecting link-layer retransmissions and by factoring in the delays due to unforeseen link-level outages. Additionally, using historical data of TCP session parameters, TCP-EWSC predicts the parameters for optimal usage of the network resources through analytics and rudimentary machine-learning.

Chapter 4 describes **TCP-NEWT** (*TCP - Network Coding Enabled Window Transformation*). This TCP variant incorporates the best of network coding and ensures seamless throughput, despite any wireless layer signal strength fluctuations and channel error issues, while ensuring that all the TCP session parameters are accurately relayed back to the higher layers in the networking stack.

Chapter 5 introduces a new mechanism for dynamic prediction of expected goodput apriori: **TCP-PNC** (*TCP-Predictive Network Coding*). This is to enable computation of the apt amount of additional unique random linear coded segments to be transported to ensure all the original segments are retrievable and available at the destination in real-time without the need of retransmission. By design, original TCP [68] has a reactive response to packet loss. With the ubiquitous advent of wireless interfaces, packet loss does not necessarily imply a true congestion in the network. TCP-NEWT along with TCP-PNC [87] enable native network coding and build on the prior work [90] [79] [89] by predicting the impending goodput during an existing TCP session, an enhancement of a prior research publication [95]. This ensures the apt number of dynamically adjusted coded segments are sent. Additionally, the coding is done in a simple manner and restricts the coding window sizes to 2, 4, 8, 16. This minimizes computation time and reduces packet delay. Thus, TCP-NEWT and TCP-PNC [87] enable proactive accurate response to impending packet loss, enabling real-time data delivery with minimal additional computational overhead.

Chapter 6 introduces TCP-RTA [88], a two-part proposal to (1) detect underlying topology changes dynamically during an ongoing TCP session, and (2) proactively switch to an appropriate congestion control strategy to ensure optimal usage of the available bandwidth. The original design goal of TCP was to transport packets reliably on an end-to-end basis, and at the same time, utilize the bandwidth available in the best possible manner. To this end, the bandwidth-delay-product is used to arrive at the overall size of the end-to-end connected pipe and thus determine the size of the sliding window of packets with outstanding ACKs. With this intent, the observed end-to-end increase in RTT [83] and packet drops were used as a metric to serve as a measure of the congestion in the network. This assumption is valid if the packet losses or delays are only due to congestion. In hybrid networks with last-mile-wireless links, the original design assumptions used for TCP are no longer accurate. Two approaches could be pursued to correct the operation of TCP in hybrid networks. The first approach involves rewrite of the TCP implementation from scratch and identify mechanisms to differentiate the delays caused by the wireless-link-layer issues from those caused by the core-network congestion. The second approach involves leveraging existing TCP implementations, while ensuring the usage of the new corrected values for RTT, which account only for those delays caused by the core-wired network and not those delays introduced by the last-mile-wireless links.

# Chapter 2

# **Related Work**

# 2.1 Survey of TCP Variants

Congestion control for TCP, particularly in the context of wireless networks, has been an area of active research. A concise comparative study of the approaches used in the different TCP variants and their implementations can be found in a couple of papers, "An Overview of Performance Comparison of Different TCP Variants in IP and MPLS Networks" [45] and "A Comparative Analysis of TCP Tahoe, Reno, New-Reno, SACK and Vegas" [23]. A survey of the various relevant variants of TCP and the way they handle the addition of wireless links is outlined in this chapter. Any issues, constraints and/or limitations with their respective implementations are also highlighted.

There are several TCP implementations including Tahoe [38], Reno [37], New-Reno [26], TCP-SACK [53], TCP-Vegas [12] [11], TCP-Jersey [97], TCP-DCR [9], TCP Santa Cruz [65], TCP-WSC (TCP-Wireless Santa Cruz) [77] which address various short-comings in the original TCP implementation [68].

For high-speed network requirements, several TCP variants have been proposed including FAST [31], HSTCP [25], STCP [46], TCPNewReno [26], HTCP [43], CUBIC [32], SQRT TCP [33], TCP-Westwood [17], BIC TCP (Binary Increase Congestion control) [98], TCP-Illinois [39], TCP-Hybla [15], YeAH-TCP [4], Compound TCP (CTCP) [91] and BBR [16] among others.

Some of the most salient features with these implementations are:

Tahoe [38]: Tahoe takes a complete-timeout interval to detect a packet loss and, in most of its implementations, it takes longer because of the coarse-grain timeout, and this is the main issue. Since it does not send immediate ACK, and instead sends cumulative acknowledgments, it follows a go-back-n approach. Thus, whenever a packet is lost, it waits for a timeout and the pipeline is emptied. This results in a major cost in links with high band-width-delay product.

**Reno** [37]: Reno performs very well over TCP when the packet losses are small. But when there are multiple packet losses in one window, then RENO does not perform very well, and its performance is almost the same as Tahoe under conditions of high packet loss. The reason is that it can only detect a single packet loss. If there are multiple packets drops, then the first information about the packet loss comes only when the duplicate ACKs are received. But the information about the second packet which was lost will come only after the ACK for the re-transmitted first segment reaches the sender after one RTT. Also, it is possible that the CWD (TCP Congestion Window) is reduced twice for packet losses, which occurred in one window. Another problem is that, if the window is exceedingly small when the loss occurs then enough duplicate acknowledgments, for a fast-retransmit trigger, would not be received and a retransmission trigger would have to wait for a coarse-grained timeout. Thus, it cannot effectively detect multiple-packet losses.

**NewReno** [26]: NewReno suffers from the fact that it takes one RTT to detect each packet loss. When the ACK for the first re-transmitted segment is received, only then it can be deduced which other segment was lost. However, it addresses some of the issues with the Reno approach and eliminates Reno's wait for a retransmit timer when multiple packets are lost within a window by responding to partial-acks, without taking the sender out of fast-recovery.

**TCP-SACK** [53]: The biggest problem with SACK is that selective acknowledgments are not provided by the receiver. If the TCP changes are localized to the sender side alone, then it is easier to deploy and support and get the needed benefits.

**TCP-CUBIC** [32]: CUBIC, introduced in 2005 [32] [49], is an improved version of the BIC (Binary Increase Congestion control) algorithm [98]. It belongs to the first group of algorithms, and unlike the standard TCP approach, CUBIC does not consider RTT to change window size. It is determined by a cubic function, depending on the time since the last packet loss.

YeAH (Yet Another Highspeed) TCP [4]: Congestion control algorithm called YeAH TCP was presented in 2006. In contrast to CUBIC, it uses a mixed approach to resizing congestion window based on losses and delays. YeAH involves two operation modes that can be called "fast" and "slow". "The "fast" mode uses the same algorithm as STCP (Scalable TCP) [46]. The congestion window size is reduced by 1/8 with each packet loss. From the other side, it is increased by 1/100 with each successful transfer. The "slow" mode is like the standard Reno TCP algorithm [38]. The size of the congestion window is determined by three phases: the slow start phase, the congestion avoidance phase and the fast recovery phase. Thresholds that are changed upon packet loss detection determine the transition between phases. However, the choice between "fast" and "slow" modes of the YeAH algorithm depends on RTT.

**BBR (Bottleneck Bandwidth and RTT) TCP [16]:** BBR is the TCP with the youngest congestion control algorithm and was introduced in 2016. The idea of BBR assumes that the maximum bandwidth of any TCP connection, formed by an arbitrarily complex default route, could be determined by two main metrics: two-way travel time (RTprop) and throughput of the bottleneck (Bottleneck Bandwidth, BtlBw). Moreover, these two values do not depend on each other. RTprop is the RTT excluding queue delays on the network devices. It can be effectively evaluated as the minimum RTT time in the  $W_R$  time window that ranges from tens of seconds to minutes.

**TCP Santa Cruz** [65]: In TCP Santa Cruz, the relative time interval delays between receipt of the ACKs for individual transmitted segments as well as estimates of delay along the forward path, is used as a measure of congestion, rather than the RTT. Subsequently, an extension to TCP Santa Cruz [65] was proposed, which improves TCP performance over lossy wireless links [64]. These are very effective enhancements relevant to hybrid networks, however they are reactive approaches. The proposed approach in this thesis TCP-EWSC, is proactive and incorporates sensing and reporting of the RSSI to facilitate proactive response to changes in the wireless signal strength and channel health.

### 2.2 Limitations of TCP Prior Variants

Packet loss in networks with wireless links may be due to bit errors, handoffs, congestion, or reordering or due to a true congestion in the network. By the nature of its design, original TCP [68] assumes packet loss is solely due to congestion in the network. TCP's congestion control responses are triggered for any wireless-linkpacket losses and for any delays (due to corresponding link-layer retransmission times), caused by inherent transient issues for packet transmission through a wireless media and not due to a congestion in the network. The corresponding adverse impact of the inadvertent congestion control response of TCP, to the effective end-end throughput, is significant. With the ongoing increase in the number of wireless clients, this is an important issue and a robust solution which will be easy to implement, will have a perceptible positive impact on a significant portion of the actual end-users of laptops, PDA (Personal Digital Assistant) and other devices which use wireless connectivity for access to the network/Internet, including mobile end-devices.

### 2.3 TCP Enhancements for Hybrid Networks

For completeness, it is pertinent to mention that RFC1185 [40] and RFC1123 [10] were among the initiatives to enable TCP extensions for high-speed networks providing for scaled windows and timestamps. The performance of TCP-IP for networks with high Bandwidth-Delay product with random losses [48], highlights some of the issues that need to be addressed for random loss networks, including networks with wireless links. The approaches proposed to improve TCP performance over networks with wireless links can be divided into two major categories:

- i. Ones that work at the transport level, and
- ii. Others that work at the link level.

Transport level proposals include Explicit Bad State Notification (EBSN) [6], Freeze-TCP [29], Indirect-TCP (I- TCP) [5], Snoop [7] and fast-retransmission[14].

Like other approaches, EBSN [6] uses local retransmission from the base station to shield wireless-link-errors and improve throughput. However, if the wireless link is in an error state for an extended duration, the source may timeout causing unnecessary source retransmission. The EBSN [6] approach avoids source timeout by using an explicit feedback mechanism. The EBSN [6] message causes the source to reinitialize the timer.

Upon detecting a poor signal strength, Freeze-TCP [29] at the mobile host throttles the sender by advertising a receive window size of zero. This causes the sender to enter persist mode and freeze all the timers and window sizes. This way, the mobile host can prevent the sender from taking any congestion control measures.

I-TCP splits the transport link at the wireline–wireless border. The base station maintains two TCP connections, one over the fixed network, and another over the wireless link. This way, the poor quality of the wireless link is hidden from the fixed network. By splitting the transport link, I-TCP does not maintain end-to-end TCP semantics. Instead, I-TCP relies on the application layer to ensure reliability.

The fast-retransmission approach does not address the issue of wireless link reliability but reduces the effect of mobile host [67] hand-off. Immediately after completing the hand-off, the IP in the mobile host [67] triggers TCP to generate a certain number of duplicate acknowledgments. This causes the source to retransmit the lost segment without waiting for the timeout period to expire. This requires modification to the TCP code at the mobile host [67].

Snoop [7] is a well-known link level proposal. In this scheme, the base station sniffs the link interface for any TCP segments destined for the mobile host [67], and buffers them if buffer space is available. Segments are forwarded to the mobile host [67] only if the base station deems it necessary.

Source retransmitted segments that have already been acknowledged by the mobile host [67], are not forwarded by the base station. The base station also sniffs into the acknowledgments from the mobile host [67]. If the base station sees a duplicate acknowledgment, it detects a segment loss and locally retransmits the lost segment if it is buffered and starts a timer. If the retransmitted segment is not acknowledged within twice the round-trip-time of the wireless link, the segment is again retransmitted. Unlike I-TCP, Snoop [7] does not completely shield the wireless link losses from the fixed network, and source timeout is still possible. If acknowledgments are lost on the wireless link, a base station retransmission cannot occur as there are no duplicate acknowledgments, and the source can timeout and source retransmission is initiated. The transmission over the wireless link resumes only after the arrival of the source retransmitted segment. Hence, in the presence of burst losses the throughput will be poor compared to I-TCP.

In WTCP [75] [76], the base station is involved in the TCP connection. WTCP [75] [76] requires no modification to the TCP code that runs in the mobile host [67] or the fixed host. Based on duplicate acknowledgment or timeout, the base station locally retransmits lost segments. In case of timeout, by quickly reducing the transmission window, potentially wasteful wireless transmission is avoided and the interference with other channels is reduced. Also WTCP [75] [76] hides the wireless-link-errors from the source by effectively subtracting the residence time of the segment at the WTCP buffer from the RTT value computed at the source, thus the RTT computation excludes wireless-link-layer retransmission delays.

However, this approach requires the base station to know the clock granularity of the source since the timestamp field in the TCP header contains a clock tick value, not the real-time clock value and this is an issue because many of the wireless-end nodes might not be in exact clock synchronization with the base station (particularly after a reboot of the end-node) and thus this approach has a big limitation. Also, in IoT and M2M domain, many wireless end-nodes might not have their clocks synchronized with the network thus making this approach not applicable in several scenarios. Thus, the WTCP approach has a number of limitations that prevent it being deployed effectively. Additionally, it does not address some of the requirements of IoT and M2M devices, which need high throughput for small bursty traffic. Leith et al [49] demonstrated that CUBIC TCP [32] suffers from slow convergence which may impede its large-scale deployment. YeAH (Yet Another Highspeed) [4] and BBR (Bottleneck Bandwidth and RTT) TCP [16] do not have mechanism to differentiate delays, losses and retransmission delays due to wireless link-layer issues, which are addressed in this thesis. Also, studies reveal that BBR may result in a salient RTT-fairness problem [92].

In TCP-DCR [9], the response to the receipt of duplicate acknowledgments is delayed by a short, bounded period  $\tau$ , to improve its robustness to channel errors in wireless networks. However, this is not an accurate way to address the delays introduced due to potential link-layer retransmissions. In TCP-WSC [77], a perfectly accurate way to precisely address this very issue was proposed.

TCP-WSC [77] incorporates improvement over existing TCP implementations in four major areas, which contribute to sub-optimal TCP throughput at various points during the lifetime of a TCP session in today's hybrid 5G Networks, particularly with mobile end nodes. These comprise (a) Starting TCP window size, (b) Increase in endend RTT, solely due to link-layer retransmissions, (c) 3 Duplicate-ACK receipt triggering fast retransmission, and (d) Transient RTT fluctuations.

In TCP-WSC [77], a communication handshake protocol between link-layer and transport layer has been proposed for a more accurate identification of a true
congestion. The number of times a given segment undergoes link-layer retransmission, is communicated by the link-layer back to the TCP layer. This helps to accurately identify and isolate the delays caused due to link-layer issues as against delays caused due to a true congestion in the network. Thus, it avoids inadvertent reduction of TCP congestion window size due to perceived longer end-end RTT. Additionally, the exact sequence numbers for the link-layer retransmitted segments, reflecting the actual order of successful transmission across the link-layer, are saved in a per-TCP-Session parameterlist, along with number of link-layer transmissions for each segment. By default, it is one for every segment. Similarly, in most prevalent TCP implementations, the fastretransmit is triggered when three duplicate acknowledgments (Dup-Acks) are received. This response assumes that network-congestion is the cause of the Dup-Acks. However, a segment undergoing a link-layer retransmission could also result in the generation of multiple Dup-Acks, until it reaches the destination, since subsequent segments (which did not need any link-layer retransmission) would have reached the TCP session endpoint before this segment, which required link-layer retransmission(s) and TCP-WSC [77] addresses this issue.

#### **BEST-START** Window Size

At the start of a TCP session, after the initial 3-way handshake, the current slow-start mechanism results in a conservative slow increase of the throughput to the steady state capacity of the TCP link. Thus, even though substantial network bandwidth could be available, TCP starts with the "slow start" mechanism for initiating the sliding window size convergence with an initial "window size of 1". This is particularly detrimental for short duration bursty traffic. In TCP-WSC [77], a mechanism to store prior TCP sessions to destination is proposed, which maintains steady state parameters of earlier sessions including steady-state congestion window size (BEST-START window size for future sessions is derived from this value). For newer TCP sessions initiated to any prior saved destinations, the knowledge of the prior session steady-state congestion window size, is used to arrive at BEST-START initial window size, to help converge very quickly to steady-state optimal use of the available bandwidth.

#### Network RTT Correction for Link-Layer Retransmissions

An increase in RTT, solely due to link-layer retransmissions, is misinterpreted by TCP as a network congestion, resulting in the inappropriate response of a reduction of TCP congestion window size. The findings of "Effect of local retransmission at wireless access points on the round-trip time estimation of TCP" [74] highlights this issue. Certain segments encounter link-layer issues and hence require link-layer retransmission. End-end RTT values for the link-layer retransmitted segments are bloated by multiples of link-layer retransmission timeouts. These larger values are interpreted as network congestion resulting in inadvertent decrease of the congestion window and corresponding throughput degradation. In this thesis, a mechanism to accurately identifying and compensating for the increase due to link-layer retransmissions in the end-end RTT is proposed.

#### Fast-Retransmission Factoring Reordering in Link-layer

Three Duplicate Acknowledgments (DUPAcks) received by TCP is interpreted as a segment lost and results in trigger of Fast Retransmit. However, this does not factor potential link-layer retransmissions for some of them, resulting in their reordering, as sent to the network. The resulting inadvertent FAST Retransmit initiation on receipt of 3 DUPAcks causes needless additional traffic and potential network congestion. Dynamic DUPack mechanism, instead of a hard-coded number (3) of DUPAcks, is proposed in TCP-WSC [77] to address this issue. In this thesis, this mechanism to account for link-layer retransmissions, is augmented to do the same in the acknowledgment receive path as well.

#### Detection Transient RTT fluctuations with Cluster Analysis

Another mechanism proposed in this thesis, to correctly identify RTT-increases solely due to network congestion, is RTT cluster analysis. Sometimes, wireless enddevices, due to their nature of being mobile devices [67], are being moved, during an ongoing TCP session and hence there are transient issues in the RSSI. Though TCP implementation does provide a weighted mechanism to minimize the impact of the transient RTT fluctuations, they affect the TCP window size and corresponding bandwidth utilization. The RTT cluster analysis mechanism proposed in TCP-WSC [77] addresses this issue. In this thesis, the cluster analysis is enhanced to incorporate removal of outliers. An observed RTT value that is more than twice the standard deviation outside of the current session history, is designated as an outlier.

#### 2.4 TCP Enhancements with Network Coding

Network Coding with TCP has been an area of active research. A concise comparative study of the actual approaches used in the different Network Coding applications to Networking can be found in paper titled "Survey of Network Coding and Its Applications" [55]. In this section, some of the most salient aspects and issues with these implementations are summarized.

Network coding meets TCP [90] TCP/NC: At the heart of this scheme is a new interpretation of ACKs. The sink acknowledges every degree of freedom (i.e., a linear combination that reveals one unit of new information) even if it does not reveal an original packet immediately. Such ACKs enable a TCP-like sliding-window approach to network coding. This scheme has the property that packet losses are masked from the congestion control algorithm. Therefore, this algorithm reacts to packet drops in a smooth manner, resulting in a novel and effective approach for congestion control over networks involving lossy links such as wireless links. However true congestion bases losses also get masked in this approach and therefore effective flow-control is inhibited. This is addressed in this thesis.

#### Redundancy adaptation scheme for network coding with TCP [78]:TCP

Vegas uses a loss predictor to decide whether the network is congested based on rate estimators [13] [54] [94], estimating the backlogged packets in the buffer of the bottleneck link and in this related work, the Vegas Loss Predictor is implemented at the Network Coding layer to know when the network experiences congestion and to adjust RTT [13]. However, the RTT values used do not factor in the additional time incurred due to potential link-layer retransmissions in last-mile wireless-links, which is incorporated in this thesis.

Comparison of TCP Congestion Control Algorithms in Network Coded Relaying Scheme [56]: The effectiveness of NC has been studied several times [90] [34] [2]. Authors investigated that NC does not give a big advantage if it used in conjunction with classic TCP, messages need to be delayed in a buffer to be able to encode them and therefore the RTT rises with every additional hop (especially where intermediate node network coding is applied). Considering that the gain decreases because TCP congestion control [3] algorithm refers to increasing RTT as congestion increases, artificially reducing the transmission rate, preventing effective use of the transmission medium. In this thesis, only end-end network coding is considered. Secondly to minimize and even eliminate the delays at the source waiting for data (at times when just a few segments are ready and waiting for transmission), the basic simplistic network coding approach that requires at least two packets to be in the transfer buffer at the same time to be able to XOR them and send them to minimize the delays, is adopted. This is further extended, when there is only a single packet and a mechanism to send it through network coding to minimize delays is also incorporated. A variant of the BBR TCP algorithm approach wherein the transmission buffers are never empty, is leveraged. However, when more buffers are full, a default fixed group size is used for choosing the RLC (Random Linear Coefficients) to generate new coded segments for transportation to the destinations.

In several of the network coding implementations, TCP layer has been augmented with a modular network coding sublayer to facilitate quick and easy adoption. However, that has resulted in many of the core intrinsic TCP session parameters and metrics like RTT, packet throughput not being accurately captured to reflect the exact status of the network. In this thesis, a mechanism for all the TCP session parameters and metrics for flow-control to be accurately captured and relayed back between the two sliding windows used in the proposed design (original packets-based TCP sliding window and random linear coded packets-based sliding window) is outlined.

## 2.5 TCP Enhancements for Dynamic Topology Changes

Mobile end-devices may undergo underlying topology changes during an ongoing TCP session, simply due to their physical movement. The resultant changes observed in end-end packet RTT would get misinterpreted as a congestion in the network by existing TCP implementations. The paper "Survey of HYBRID TCP Congestion Control Algorithms" [1] has a survey of various Congestion Control algorithms proposed for hybrid networks. None of the TCP variants in the survey address this issue of adapting to dynamic topology change. The prominent prior published research related to this specific issue includes D-TCP Dynamic TCP Congestion Control Algorithm for Next Generation Mobile Networks [42], wherein Bandwidth-Delay product is dynamically computed and a congestion metric derived off this computation, which is used to determine the response of the congestion control algorithm as regards any changes to the size of the CWND (TCP congestion window) during the RTT update and loss detection. Thus, only a single parameter is being dynamically modified and the underlying congestion control algorithm is the same for all scenarios and through the life cycle of the current session and thereafter till the TCP stack is changed.

In this thesis, TCP-RTA (TCP with Real-time Topology Adaptiveness) [88] is proposed, which dynamically recognizes potential underlying topology change in the end-end path of the current TCP session and instantaneously initiates transitions to the apt congestion control algorithm applicable for the updated new topology, thus adapting in real-time.

The goal of this thesis is to propose solutions for the above issues for throughput bottlenecks in hybrid networks. These enhancements can be leveraged across several existing TCP implementations including newer TCP versions like TCP Cubic and more effective when deployed in tandem with ECN (Explicit Congestion Notification) [72].

# Chapter 3

# **TCP-EWSC:**

# **TCP** Enhanced-Wireless-Santa-Cruz

TCP-EWSC provides five key enhancements over TCP-WSC (Wireless Santa Cruz TCP [77]): (1) Starting TCP Window Size, (2) Compensation for increase in RTT Time, solely due to Link-Layer Retransmissions, (3) 3 Duplicate Acknowledgment (DUPAck) receipt triggering fast retransmission inadvertently, (4) RSSI monitoring to proactively identify sporadic and transient wireless link issues, and (5) RTT cluster analysis to better identify and respond to transient RTT fluctuations. The details about each of the enhancements is elaborated below.

Starting TCP Window Size: The BEST-START window size approach to arrive at the initial window size for a new TCP session, as proposed in TCP-WSC [77], is enhanced by a more detailed learning mechanism specific to each TCP session. TCP-EWSC arrives at the most apt steady state window size by factoring in a weighted mechanism of past steady states. Additionally, a multiplicative factor in the range 0-1, representing a measure of how much of legroom is required to be provided to this tcp session to facilitate the most optimal convergence to its steady-state window size, is used to arrive at the "Enhanced BEST-START" starting window size from the statistically computed expected "steady state window size". With each session, TCP-EWSC uses the statistical model and choose an improved BEST-START window size; this minimizes the time spent in the slow-start phase of a TCP session, ensuring optimal use of available bandwidth while meeting fairness protocol requirements.

Compensation for Link-Layer Retransmissions: To accommodate potential link-layer issues on the return path of the acknowledgments, TCP-EWSC extends the compensation for additional transmission time and re-ordering of receipt of the acknowledgments (ACK). These additional incurred times are suitably subtracted from the RTT based on the number of link-layer retransmissions, undergone by that ACK. This correction ensures any variations in RTT that is reported, does not include those caused by wireless-link-layer issues and thus gets accurately interpreted.

**Fast retransmission trigger:** TCP-EWSC augments the fast retransmission trigger with initiation of a retransmission when the time elapsed waiting for an ACK crosses the threshold of more than twice the observed standard deviation from the observed mean of the RTTs in the existing TCP session.

**RSSI Monitoring:** A novel contribution in this thesis is constant monitoring of the RSSI and proactive initiation of spoofing 'receiver zero window" acknowledgment by layer-2, so that the TCP session is not torn down just because of a sporadic transient

wireless-link-layer issue. Sustaining the TCP session through sporadic very brief wireless connectivity outages has been found to be a necessary requirement in several mobile nodes and devices in transit as the occurrences of these transient wireless outages are observed quite often, including when the hand-offs happen in 5G networks from one base-station to another. Additionally, constant RSSI monitoring enables identification of transient wireless signal strength fluctuations especially compared to persistent wireless link capability changes, which would need to be considered as a true congestion in the wireless link.

**Cluster Analysis:** TCP-EWSC enhances the cluster analysis, as proposed in TCP-WSC [77], with regards to the interpretation of the findings by identifying outliers in observed RTT values and thereafter the new enhanced response of ignoring those values.

#### **3.1** Basic Operation

TCP-EWSC improves on BEST-START Window size as proposed in TCP-WSC [77], by recording the congestion window size(SSCA-window-size) at which the transition from slow-start(SS) to congestion-avoidance(CA) is effected during the initial start of every TCP session. A new weighted algorithm that leverages saved values of this attribute from prior sessions, to predict the apt Enhanced BEST-START window size is introduced in this thesis. In addition, a mechanism for choice of an apt default starting TCP window size (TcpWin-Start-sz-default) for each new TCP session is proposed, which is used as the initial starting value input to TCP-EWSC's weighted algorithm. This would ensure good throughput and utilization of the network and improved latency, particularly for short duration bursty TCP traffic.

The compensation for the link-layer retransmissions [77] is further enhanced as follows: Current TCP implementation interpret all re-ordering of received packets and their ACKs as solely due to network congestion. However, link-layer retransmissions with no network congestion also can cause re-ordering of transmitted segments as well as received ACKs. TCP-WSC [77] compensates for the link-layer retransmissions on the segment transmission side. As part of the enhancement proposed in this thesis, TCP-EWSC extends that to the ACK receive path as well.

TCP-EWSC accomplishes this by keeping track of (i) time and (ii) order of receipt of ACKs and (iii) the number of retransmissions required for each ACK, on the last-mile wireless access point to ensure that the original order of receipt is recovered. Using the number of ack-retransmission on the link-layer, TCP-EWSC can compensate for the RTT time to truly reflect only network transit time (and not the additional time consumed due to link-layer errors and resulting re-transmissions).

TCP-EWSC augments the inadvertent fast-transmission as follows: It dynamically adjusts the number of Dup-Acks required based on any pending segment(s) requiring link-layer retransmission(s). This was done in In TCP-WSC [77]. TCP-EWSC enhances it by triggering a retransmission if the time elapsed waiting for an ACK, goes beyond twice the standard deviation of mean observed RTT for the session, instead of waiting for timeout. The new retransmission strategy is motivated by such evidence as the Internet trace reports by Lin and Kung, which show that 85 per cent of TCP timeouts are due to "non-trigger" [51].

In addition to the ongoing wireless signal strength levels tracking, a novel contribution regarding RSSI monitoring, is the incorporation of the zero-receiver window spoofing mechanism, to ensure the TCP connection is not torn down due to brief complete glitch in the wireless link-layer connectivity, namely the wireless signal strength going to zero.

The primary enhancement of the cluster analysis approach is the removal of outliers. TCP-EWSC earmarks outliers as an observed RTT value that is more than twice the standard deviation outside of the current session history. In Chapter 6, more details on how differentiation of outliers from those caused by legitimate changes in underlying topology, is presented.

## **3.2** TCP-EWSC Description

#### 3.2.1 Enhanced BEST-START Window Size

In this thesis, the mechanism to store prior TCP sessions' parameters as proposed in TCP-WSC [77] is enhanced with additional changes. The saved TCP session parameters include steady-state congestion window size, which is leveraged for arriving at BEST-START Window size for future sessions. Another significant session parameter is **the congestion window size(SSCA-window-size)** at which the transition from **slow-start(SS)** to **congestion-avoidance(CA)** is affected during the initial start of the TCP session. The SSCA-window-size values of prior TCP sessions to a given destination are leveraged to converge in the most optimal manner to new TCP session's steady-state throughput with best use of the available bandwidth for this TCP session being initiated to same destination. The proposed learning algorithm below determines how prior session's observed parameters facilitate quicker convergence to optimal steady-state window size, namely **Enhanced BEST-START Window Size** for the new TCP session being initiated. Outlined below are the fields captured for every TCP session.

The prior history of TCP sessions is saved as a 10-tuple value, as stated below:

- Field 1: Destination Port
- Field 2: Destination IP Address
- Field 3: Date/Time of TCP Session Initiation Field 4: Duration of Session
- Field 5: SSCA-Window-size where initial Slow-Start to Congestion Avoidance transition happens
- Field 6: Time taken from session start to completion of Slow-start phase
- Field 7: Steady State Congestion Window Size
- Field 8: SSCA-Multiplicative-Factor (Multiplicative factor of SSCA-Windowsize)
- Field 9: goodput at the start of the session

• Field 10: TcpWin-Start-sz-default, which is the default starting size of the TCP window. (Result of the Enhanced BEST-START algorithm saved in this field)

For a given port-IP Address combination of a destination, only the three most recent entries are retained according to the following steps:

(a) For a new destination IP address, if 3 matches are found, then TCP-EWSC uses weighted averages as follows: 0.6 multiplication factor for (1st) most recent observations; 0.3 multiplication factor for 2nd most recent observations; 0.1 multiplication factor for 3rd most recent observations.

(b) For a new destination IP address, if 2 matches are found, then TCP-EWSC uses weighted averages as follows: 0.65 multiplication factor for most (1st) recent observations; 0.35 multiplication factor for 2nd most recent observations.

(c) if only 1 match is found, the corresponding values are directly used.

(d) in case no match was found, the traditional slow-start approach in TCP is followed, namely starting from a window size of 1.

(e) After computing the Enhanced BEST-START size by the above algorithm in steps (a), (b), (c) and (d), that value is saved as in field 10 namely as TcpWin-Start-sz-default.

TCP-EWSC's main objective is to minimize the time taken in the slow-start phase and quickly converge to optimal usage of the available bandwidth, while ensuring fairness is fully maintained with regard to independent requests for resources from other TCP sessions.

TCP-EWSC leverages learning from past sessions history to help achieve the

objective in the most efficient manner. There are two areas for potential enhancement, namely: The initial starting window size, and the gradient of the step up in window size each time when an ACK is received during the initial Slow-start phase of the TCP session. In this thesis, the focus is on determining the apt initial starting window size.

The starting default value for the SSCA-MultiplicativeFactor is 0.5. TCP-EWSC tries to keep this factor within the permitted range of [0.3 to 0.7]. It monitors the time taken from session start to transition from Slow-start to congestion avoidance and incrementally tries to fine tune the above factor till it gets close to minimizing Field 6 (*Time taken from session start to completion of Slow-start phase*).

Algorithm 1 procedure EWSC-TCP-BestStart-Basic\_WndSz(DestIPAddress, Dest-PortAddress)

```
1: best_start_wndSz = 1;
2: cnt = 0;
3: mf_3Terms_1st = 0.6;
4: mf_3Terms_2nd = 0.3;
5: mf_3Terms_3rd = 0.1;
6: mf_2Terms_1st = 0.65;
7: mf_2Terms_2nd = 0.35;
8: mf_1Term_1st = 1;
9: SSCA_Wnd_MF = 0.5;
10: while not EOF do
        m = find_addr(DestIPAddress, savedSessions);
   /* m returns the index of the first match & (-1) if there is no match till EOF*/
11:
       if (m = -1) then
          m_laddr[cnt].date_time = savedSessions[m].date_time;
12:
          m_{addr[cnt]}.duration = savedSessions[m].duration;
13:
          m_laddr[cnt].SSCA_WndSz = savedSessions[m].SSCA_WndSz;
14:
          m_laddr[cnt].timeCompleteSS = savedSessions[m].timeCompleteSS;
15:
          m_laddr[cnt].SS_WndSz savedSessions[m].SS_WndSz;
16:
          m_laddr[cnt].beginLossRatio = savedSessions[m].beginLossRatio;
17:
          m_laddr[cnt].SSCA_MF = savedSessions[m].SSCA_MF;
18:
          m_{addr[cnt].destPort} = savedSessions[m].destPort;
19:
          \operatorname{cnt}+;
20:
       end if
21:
22: end while
   sort_latestTop(m_laddr, date_time)
   \mathbf{ASSERT}((\operatorname{cnt} \ge 0) \&\& (\operatorname{cnt} \le 3));
23: if (cnt = 3) then
       return(SCA_Wnd_MF \times
24 \cdot
          (m_laddr[cnt].SSCA_WndSz×mf_3Terms_1st+
25:
          m_laddr[cnt+1].SSCA_WndSz×mf_3Terms_2nd+
26:
          m_laddr[cnt+2].SSCA_WndSz×mf_3Terms_3rd));
27:
   else if (cnt = 2) then
28:
       return(SSCA_Wnd_MF \times
29:
30:
          (m_laddr[cnt].SSCA_WndSz \times mf_2Terms_1st +
         m_{addr[cnt+1]}.SSCA_WndSz× mf_2Terms_2nd));
31:
   else if (cnt == 1) then
32:
       return (SSCA_Wnd_MF\times
33:
          m_laddr[cnt].SSCA_WndSz×mf_1Term_1st);
34:
35: else
       return(best_start_wndSz);
36:
37: end if
```

Algorithm 2 procedure EWSC-TCP-BestStart-WndSz\_factorLR(DestIPAddress, DestPortAddress, CurLossRatio)

```
1: best_start_wndSz = 1;
2: cnt = 0;
3: mf_3Terms_1st = 0.6;
4: mf_3Terms_2nd = 0.3;
5: mf_3Terms_3rd = 0.1;
6: mf_2Terms_1st = 0.65;
7: mf_2Terms_2nd = 0.35;
8: mf_1Term_1st = 1;
9: SSCA_Wnd_MF = 0.5;
10: while (not EOF) do
       m = find_addr(DestIPAddress, savedSessions);
11:
    /* m returns the index of the first match & (-1) if there is no match till EOF*/
12:
       if (m = -1) then
          m_{addr[cnt].date_{time}} = savedSessions[m].date_{time};
13:
          m_{addr[cnt]}.duration = savedSessions[m].duration;
14:
          m_laddr[cnt].SSCA_WndSz = savedSessions[m].SSCA_WndSz;
15:
          m_laddr[cnt].timeCompleteSS = savedSessions[m].timeCompleteSS;
16:
          m_laddr[cnt].SS_WndSz = savedSessions[m].SS_WndSz;
17:
          m_laddr[cnt].beginLossRatio = savedSessions[m].beginLossRatio;
18:
          m_laddr[cnt].SSCA_MF = savedTCP_SessiSSons[m].SSCA_MF;
19:
          m_{addr[cnt].destPort} = savedSessions[m].destPort;
20:
          \operatorname{cnt}+;
21:
       end if
22.
23: end while
24: sort_latestTop(m_laddr, date_time); ASSERT((cnt \ge 0) \&\& (cnt \le 3));
25: if (m_laddr[cnt].beginLossRatio) then
       if (cnt == 3) then
26:
          if (m_laddr[cnt].beginLossRatio > CurLossRatio) then
27 \cdot
28:
             tmp_bestStart_WndSz
                = SSCA_Wnd_MF \times 1.2 \times m_laddr[cnt].SSCA_WndSz \times mf_3Terms_1st;
29:
          else if (m_laddr[cnt].beginLossRatio < CurLossRatio) then
30:
             tmp_bestStart_WndSz
31:
                = SSCA_Wnd_MF \times 0.8 \times m_laddr[cnt].SSCA_WndSz \times mf_3Terms_1st;
32:
          else
33:
             tmp\_bestStar\_WndSz = SSCA\_Wnd\_MF \times
34:
                m_laddr[cnt].SSCA_WndSz×mf_3Terms_1st;
35:
          end if
36:
          if
             (m_laddr[cnt-1].beginLossRatio > CurLossRatio) then
37:
             tmp_bestStart_WndSz = SSCA_Wnd_MF \times 1.2 \times
38:
                m_laddr[cnt-1].SSCA_WndSz×mf_3Terms_2nd;
39:
          else if m_laddr[cnt].beginLossRatio < CurLossRatio) then
40:
41:
             tmp\_bestStart\_WndSz = SSCA\_Wnd\_MF \times 0.8 \times
                  m_laddr[cnt-1].SSCA_WpdSz×mf_3Terms_2nd;
42:
          else
43:
             tmp\_bestStart\_WndSz = SSCA\_Wnd\_MF \times m\_laddr[cnt-1].
44:
                SSCA_WndSz×mf_3Terms_2nd;
45:
46:
          end if
```

47:	$if (m_laddr[cnt-2].beginLossRatio > CurLossRatio) then$
48:	$tmp\_bestStart\_WndSz = SSCA\_Wnd\_MF \times 1.2 \times m\_laddr[cnt-2].$
49:	$SSCA_WndSz \times mf_3Terms_3rd;$
50:	$else \ if \ m\_laddr[cnt].beginLossRatio < CurLossRatio) \ then$
51:	$tmp\_bestStart\_WndSz = SSCA\_Wnd\_MF \times 0.8 \times m\_laddr[cnt-2].$
52:	$SSCA_WndSz \times mf_3Terms_3rd;$
53:	else
54:	$tmp\_bestStart\_WndSz = SSCA\_Wnd\_MF \times m\_laddr[cnt-2].$
55:	$SSCA_WndSz \times mf_3Terms_3rd;$
56:	end if
57:	end if
58:	$return tmp_bestStart_WndSz;$
59:	else if $(cnt = 2)$ then
60:	$\mathbf{if} (m_{\mathrm{laddr}[\mathrm{cnt}]}.\mathrm{beginLossRatio} > \mathrm{CurLossRatio}) \mathbf{then}$
61:	$tmp\_bestStart\_WndSz$
62:	$=$ SSCA_Wnd_MF $\times$ 1.2 $\times$ m_laddr[cnt].
63:	$SSCA_WndSz \times mf_2Terms_1st;$
64:	else if $m_laddr[cnt]$ .beginLossRatio < CurLossRatio) then
65:	tmp_bestStart_WndSz
66:	$= SSCA_Wnd_MF \times 0.8 \times m_laddr[cnt].$
67:	$SSCA_WndSz \times mf_2Terms_1st;$
68:	else
69:	tmp_bestStart_WndSz
70:	$= SSCA_Wnd_MF \times m_laddr[cnt].$
71:	$SSCA_WndSz \times mf_2Terms_1st;$
72:	end if
73:	If $(m_{laddr[cnt-1],beginLossRatio} > CurLossRatio)$ then
74:	tmp_bestStart_WndSz
75:	$= SSCA_Wind_WF \times 1.2 \times m_laddr[cnt-1].$
76:	sb0A_whd5z×hii_2fernis_zhd;
79.	tmp hostStart WndSz
70:	$- SSCA Wrd ME \times 0.8 \times m [addr[cnt 1]]$
79: 80.	$= SSCA_Wid_Wi^* \land 0.0 \land m_1addr[cm-1].$
80. 81.	also
82.	tmp bestStart WndSz
82.	$-$ SSCA Wnd MF $\times$ m laddr[ent-1]
84.	$= SSCA WndSz \times mf 2Terms 2nd$
85·	end if
86·	return tmp bestStart WndSz:
00.	

87:	else if $(cnt == 1)$ then
88:	$if (m_laddr[cnt].beginLossRatio > CurLossRatio) then$
89:	$tmp\_bestStart\_WndSz$
90:	$=$ SSCA_Wnd_MF $\times$ 1.2 $\times$ m_laddr[cnt].
91:	$SSCA_WndSz \times mf_1Term_1st;$
92:	else if $m_laddr[cnt]$ .beginLossRatio < CurLossRatio) then
93:	$tmp\_bestStart\_WndSz$
94:	$=$ SSCA_Wnd_MF $\times$ 0.8 $\times$ m_laddr[cnt].
95:	$SSCA_WndSz \times mf_1Term_1st;$
96:	else
97:	$tmp\_bestStart\_WndSz$
98:	$=$ SSCA_Wnd_MF $\times$ m_laddr[cnt].
99:	$SSCA_WndSz \times mf_1Term_1st;$
100:	$return tmp_bestStart_WndSz;$
101:	end if
102:	else
103:	return (best_start_wndSz);
104:	end if

#### 3.2.2 Enhanced RTT Correction: Link-Layer Retransmissions

Increased RTT, solely due to link-layer retransmissions, is misinterpreted by TCP as network congestion, resulting in an inappropriate response of a reduction of TCP congestion window size. WSC-TCP [77] had proposed a mechanism to accurately remove the increase due to link-layer retransmission on the TCP source transmission side only. TCP-EWSC factors in link-layer retransmissions of acknowledgment on the TCP source receive side as well, and this is a new contribution. Enumerated below is the additional required change on the link-layer at the wireless access router to which the end device, which is the origination of the TCP session of interest, is connected:

(1) For every datagram sent by the Wireless Access node, TCP-EWSC includes a count of number of retransmissions done for each datagram. The default value of this count will be 0, corresponding to the best case wherein no retransmissions are required.

(2) The link-layer will send this acknowledgment retransmission count to the transport layer along with the sequence number.

(3) The transport layer will then add the original packet retransmission count along with the acknowledgment retransmission count and

(4) multiply it by the Link-Layer-Retransmission-Timeout and

(5) subtract it from the end-end measured RTT to get the Actual-RTT.

(6) The above computed "Actual-RTT" is then used in TCP congestion control algorithms instead of the "End-End measured RTT".

Enhancement steps depicting the End-End measured RTT correction, factoring

link-layer retransmissions in TCP-WSC [77] is enumerated below:

- Communicate the fixed link-layer retransmission timeout (Wireless-Route\_LL\_RTO) value to the TCP layer (onetime per TCP session)
- For every Acknowledgment received:
- Keep track of the exact number (NRA NumberRetransmissionsRequired) of link-layer retransmissions required for each acknowledgment and this is included in the ACK itself.
- Sends the sequence number of successful link-layer (re)transmitted ACK, back to the TCP layer.
- TCP Layer uses the above information and computes (NRA)× WirelessRouter\_LL\_RTO
- Corrected RTT (from NW Congestion perspective) for each segment =
   Actual end-to-end observed RTT time for that segment - (n)×LL\_RTO
   - (NRA)×WirelessRouter\_LL\_RTO
- TCP Layer uses the above outlined Corrected RTT (from NW Congestion perspective) for each segment

#### 3.2.3 Enhanced Fast-Retransmission: Link-Layer Reordering

Three Duplicate Acknowledgments (DUPAcks) received by TCP is interpreted as datagram lost and results in Fast Retransmit. However, this does not factor in certain datagrams requiring link-layer retransmissions, resulting in reordering of datagrams, as sent to the network. The resulting inadvertent FAST Retransmit initiation on receipt of 3 DUPAcks causes needless additional traffic and potential network congestion. A dynamic DUPack mechanism instead of a hard-coded number (3) of DUPAcks, to account for potential link-layer retransmission of certain segments causing a reordering in actual transmission has been proposed in [77] to address this issue. TCP-EWSC factors in reordering of acknowledgment packets due to link-layer retransmissions of acknowledgment packets on the TCP-source's receive side return path and potential inadvertent re-ordering of the ACKs as they get delivered back to Source's TCP layer. This prevents any additional errors in identifying successive real duplicate acknowledgment (DUPAck) receipts to accurate count to trigger initiation of a fast-retransmit.

TCP-EWSC assigns an ACK-receipt-TCP-source-AP-side-sequence-number to every acknowledgment frame in the order in which it is received from the network by the local-Wireless-Router(Access Point). The goal is to track the order of receipt of the ACKs from the network and this is included in every ACK frame. The ACKs are reordered according to ACK-receipt-TCP-source-AP-side-sequence-number in the Transport layer as soon as the ACKs are received before they are processed by the TCP Receiver. This way it is ensured that DUPAcks, which trigger a retransmission, are indeed accurately doing so, due to a corresponding real network congestion issue (and not impacted or modified due to transient wireless-link-layer glitches).

#### 3.2.4 RSSI Enhancement: Receiver Zero-Window Buffer Spoof

Receiver Zero-Window Buffer Spoofing is initiated for sender-side transient link-layer Signal Drops. TCP-EWSC proactively responds to the dynamic temporary RSSI (*Receiver wireless Signal Strength Indicator*) signal strength reduction by spoofing TCP Receiver Zero-Window Buffer to ensure the TCP session is kept intact during this transient glitch. Subsequently, the TCP window size bounces back quickly to the optimal size using the *Enhanced BEST-START window* mechanism outlined above in subsection 3.2.1.

For every tcp session, the last ACK received is saved in the link-layer of the TCP sender. If the RSSI suddenly goes to zero, namely sporadic wireless outage is sensed by the wireless MAC layer, then the last ACK received is repurposed and using the same ACK number however with zero receive buffer size spoofed to the transport layer, so the TCP session is not torn down, though the data transmission is brought to a standstill. TCP-EWSC also augments the TCP transport layer fast-retransmit implementation so that when a DUP-ACK is received and the receive buffer size is "0", the DUP-ACK is not counted to trigger a fast-retransmission. A timer is started to a preset "timeout" value, for the duration up to which this "Receiver Zero window spoofing is maintained".

#### 3.2.5 Enhanced Cluster Analysis

When an observed RTT value is more than twice the standard deviation observed so far with respect to the given session, then it is deemed as a one-off RTT variant and ignored. TCP-EWSC uses the "corrected RTT" values after accounting for any "link-layer" retransmissions so that only the variations and variance from true network congestion are factored in, while doing the statistical cluster analysis. After review of some of the RTT statistics and data available, it was felt that ignoring those RTT values which are close to and more than twice the standard deviation of the RTT value observations might be the best approach.

## 3.3 Performance Results



The simulation was done with NS-2 and the results are shared below:

Figure 3.1: TCP-Cubic with No-Loss



Figure 3.2: TCP-Cubic with 10% Loss



Cubic BEST-START VS Slow-Start - Throughput Comparison w 20% Loss

Figure 3.3: TCP-Cubic with 20% Loss



Figure 3.4: TCP-NewReno with No-Loss





Figure 3.5: TCP-NewReno with 10% Loss



Figure 3.6: TCP-NewReno with 20% Loss

## 3.4 Performance Comparison

As is clear from the observations from the simulations, if there is any loss then the enhanced mechanisms proposed in TCP-EWSC result in a significant improvement in performance over other TCP variants. In Figure 3.2 and Figure 3.3, the transient performance right after a glitch using TCP-EWSC's mechanism is maintained at prior levels whereas with existing implementations there is a drop of about 8%. Looking at the graph it is evident that the "throughput is almost sustained" at the prior levels even when compared to other TCP variants, unlike the 8% to 10% drop observed with existing TCP implementations as seen with TCP NewReno simulations in Figure 3.5 with 10% loss and in Figure 3.6 with 20% loss.

## 3.5 Summary

The several enhancements proposed ensure every bit of available bandwidth is fully used and nothing is inadvertently left free and unused, particularly during the transient phases of a TCP session when there is a sudden increase in loss, primarily due to link-layer errors (wireless signal strength fluctuations). In today ubiquitous wireless deployments and the mission critical requirements of best connectivity and full utilization of available bandwidth at any given point in time, including in the midst of an ongoing TCP session, the enhancements proposed in TCP-EWSC in subsections 3.2.1 ensure that there is constant leverage of prior TCP sessions parameters and judicious use of that knowledge as proposed in subsections 3.2.2 and 3.2.3 to accurately and quickly converge to optimal steady state usage of the available bandwidth instead of going back to the traditional slow-start phase. Also, for momentary unforeseen wireless signal glitches, the Enhanced Receiver Zero-Window Buffer Spoof mechanism in subsection 3.2.4 is significant relief to the end-users to navigate and recover from it gracefully, seamlessly.

# Chapter 4

# TCP-NEWT: TCP Network-Coding Enhanced Window Transformation

**TCP-NEWT** (**TCP** with Network Coding Enhanced Congestion Window Transformation) transforms the original TCP congestion window into a new TCP congestion window comprising of network coded data generated from a group of segments in the original TCP window on the sender side.

Similarly, on the receiver side, on detecting the receipt of a TCP-NEWT network coded segment, TCP-NEWT transforms the TCP-NEWT receiver window into the corresponding original TCP receiver window comprising of the original TCP data segments generated by decoding of the received group of coded TCP segments. Additionally, TCP-NEWT introduces a novel mechanism for processing of the acknowledgment packets so that the path metrics like RTT measurements by the TCP-NEWT network coded segments are accurately relayed back to the original TCP stack.

The design of TCP-NEWT assumes a fixed goodput and its specification consists of: (a) the TCP packet header augmentation needed to support network coding, (b) the available choice of coefficient values that are supported by TCP-NEWT's design, (c) the enumeration of the permitted group sizes, (d) encoding of the original TCP segments on the sender side and (e) decoding of the network coded segments on the receiver side.

An example has been taken below, to illustrate how TCP-NEWT transformation works on group sizes of 1 and 2. This provides sufficient insight and clarity as to how it would work on larger group sizes. Subsequently, encoding, decoding and processing of acknowledgments including relaying of the network health (observed through RTT measurements) back to original TCP window, are elaborated in detail.

A key consideration in TCP-NEWT's approach has been to keep the computation overhead of generating the network-coded segments for transmission at the sender side as well as the subsequent decoding at the receiver side to bare minimal. This would enable TCP-NEWT deployment on mobile end-nodes with significant limitations on the computing, memory and in many cases power resources available, which is one of the objectives of this work.

# 4.1 Basic Operation

In TCP-NEWT, the header of a TCP-NEWT segment is augmented with a specific boolean field "*network-coded*" to indicate if it is a network-coded segment. Figure 4.1 illustrates the packet headers used in TCP-NEWT. The first 20 bytes of the TCP header are always used in TCP-NEWT. The options field is of variable size, and it starts from row 6 and can go up to 40 bytes. TCP-NEWT uses the **TCP Option Kind number 25** [35] [80]. The newly introduced TCP-NEWT specific option field entries are (a) kind equal to 25 (8 bits); (b) length in bytes (8 bits); (c) network\_coded:1 (8 bits); (d) group\_size equal to 1, 2, 4 or 8 (8 bits); (e) group\_id: "Grp Seq Num" (32 bits); and (f) CE*i* equal to 1, 2, 4, 8, 16 or 32 with i = 1 to 32 (6 unique values can be represented by 3 bits, however 4 bits are allocated for each CE).



CE : Coefficient

Figure 4.1: TCP-NEWT header

The "group-id" field is initialized to reflect the group-number corresponding to this coded segment and similarly the "Random Linear Coefficients" fields are assigned the RLC coefficients used to generate this segment, namely CE1, CE2....

Linear combinations of segments, part of a group, are combined to generate a network-coded segment. Hence, the number of terms being added is equal to the group size. The permitted group sizes in TCP-NEWT's implementation are 1 or 2 or 4 or 8 (though group size of 16 is also depicted as an option in the TCP header field). The permitted coefficients for generating the corresponding linear combinations in TCP-NEWT's implementation are one of six values namely 1, 2, 4, 8, 16 or 32, which ensures that multiplication with these coefficients is simply a bit-shifting operation and thus incurs minimal computational overhead. Though 3 bits would suffice for representing 6 unique values, an additional bit is also reserved for this and a total of 4 bits are allocated to represent each coefficient. The maximum number of bits consumed in the TCP header is: (number of bits per coefficient) multiplied by (number of coefficients), which is  $4 \times 16 = 64$  bits. In TCP-NEWT's implementation, the maximum group size is restricted to 8, hence  $4 \times 8 = 32$  bits are used.

Another important consideration in the choice of coefficients is that the random linear coefficients (RLC) of the segments in the original group being used to generate the coded TCP segments are such that its determinant can be evaluated without significant computation overhead at the receiver to regenerate the original packets in the group.

#### **TCP-NEWT** Permitted Random Linear Coefficient Values

TCP-NEWT restricts the choice of the random linear coefficients to the following 6 values: 1, 2, 4, 8, 16 and 32. The rationale for this choice is that multiplication with these numbers involves just a bit shift in binary numbers and thus involves minimal additional processing overhead. Also, the payload size could increase when bit shifting if the leading data field is a "1". Hence TCP-NEWT uses an additional 8 bits reserved for potential payload bloat when computing the coded segments.

#### **TCP-NEWT** Permitted Group Sizes

The group size indicates the number of segments grouped together from the original non-coded data segments. These segments are then combined (added) together after each is multiplied by suitably chosen coefficient from the set of random linear coefficients listed below to generate a coded segment. This above step is repeated till the required number of "coded segments" based on the loss-ratio (1 - goodput): packets-dropped/total-packets-sent. For design simplicity, permitted group sizes is restricted to one of the following: 1, 2, 4 or 8. The number of RLC (Random Linear Coded) unique segments that can be generated with the above permitted choices: A group size of 1 results in  $1 \times 6 = 6$  maximum RLC coded segments, as 6 RLC values (as outlined in subsection 4.1) are permissible. A group size of 2 results in  $6 \times 6 = 36$  RLC coded segments, a group size 4 results in  $6 \times 6 \times 6 \times = 1296$  RLC coded segments, and so on. However, some of the random linear combinations may be scaled version of another tuple, in which case they would not be capturing any unique information and TCP-

NEWT design avoids choice of such tuples. The worst-case additional bloat in the RLC coded segment is computed as follows: The largest group size permitted is 8. Worst-case scenario is all of them getting scaled and multiplied by the largest permitted RLC coefficient of 32, which causes a bit of shift of 5. Assuming the data in all the 8 (worst-case scenario) had a 1 in the MSB (most significant bit), it would increase the number of bits by log<sub>2</sub> 8, (where 8 is the maximum permitted group size in TCP-NEWT's design) which is 3 bits. The worst-case maximum bit shift of 5 due to multiplication by RLC, plus a bloat of 3 bits due to addition results in a total cumulative increase of 8 bits and TCP-NEWT allocates 16 bits for data payload bloat due to network coding, thus providing for sufficient room for future scaling for larger group sizes. In the figure 4.2 below, a fixed goodput is assumed and the entire set of 4 segments in the initial group from original TCP sliding window are coded using random coefficients to generate 5 coded segments for the 15% loss-ratio scenario (85% goodput). These are placed in the new TCP-NEWT window.



Figure 4.2: Network Coding

# 4.2 Protocol Description

Each generated RLC segment has the following additional fields:

- Orig-Grp-size: (≤ 16) permitted values in TCP-NEWT's design: 1 or 2 or 4 or 8, (Number of segments in a group from the original TCP window).
- 2. RLC coefficients: the number of coefficients is equal to the Orig-Grp-size, (the coefficients used for generating the new coded segments are unique tuples).
- 3. Unique Group ID: Group Sequence number for each group, common for all members in a group (like the sequence number for individual segments).
- 4. Coded-Grp-size = Orig-Grp-size/(1 WLR), (where WLR is worst-case loss-ratio and Coded-Grp-size is the number of coded segments generated from the group

in Original TCP Window).

The protocol description is presented through a couple of scenarios.

#### 4.2.0.1 Group size 1

The figure 4.3 depicts a scenario with just a single TCP segment in a group.



Figure 4.3: Group Size 1

#### Sending Side

The figure 4.3 depicts the transmit mechanism when there is a single segment. When there is a single segment in the sliding window and there are no other data/segments queuing in from higher layers for this TCP session, then group size (Orig-Grp-size) is set to 1. Depending on the loss-ratio, the number of coded segments generated could range from 2 to 4. In the above example, the network coding group-id is 8 and the number of coded segments generated has been chosen to be 4.
cf4554202f20485454502f312e300d0a 557365722d4167656e743a2057676574 2f312e31312e340d0a4163636570743a 202a2f2a0d0a486f73743a207777772e 696574662e6f72670d0a436f6e6e6563 74696f6e3a204b6565702d416c697665 0f0a0d0a

Figure 4.4: Original Sample Data Seg 1 - Hex Dump

One of the stated goals is to ensure that both processor utilization as well as memory consumption is bare minimal. This will ensure they are deploy-able in mobile end-nodes with limited processing and memory capacities. The ease with which the coded segments can be generated in TCP-NEWT's approach is demonstrated using an actual sample data comprising the following two TCP segments, segment-1 in figure 4.4 depicted above and segment-2 in figure 4.20 shown later.

Ki	nd : 25	length	n : 16	network	coded : 1	group	size		
group id : 8									
2					)				
							0000000 0000000		

Figure 4.5: Group Size 1: Coded Datagram 8.1's TCP Header

The modified TCP-NEWT header using a coefficient of 2 is depicted above.



Figure 4.6: Group Size 2: CE1-2 Coded Datagram 8.1's Computation by bit shifting

A simple left shift of all the contents by 1 bit generates the coded data.

CE1=2 1 9 8 4 0 5 ... е 8 а а 1 1001 1110 1000 1010 1010 1000 0100 0000 0101 ... .... .... 1 4 е 1 4 1 а 1 0001 1110 0001 0100 0001 1010 0001 0100

Figure 4.7: Group Size 1: CE1-2 Coded Datagram 8.1's TCP Data

The above depicts the actual contents of network coded segment with CE1=2.

Kir	Kind : 25 length : 16				network coded : 1 group				
group id : <b>8</b>									
4	- 	8		200 1	s *				

Figure 4.8: Group Size 1: Coded Datagram 8.2's Relevant TCP Header fields

The modified TCP-NEWT header with 4 Coefficient is depicted above.



Figure 4.9: Group Size 2: CE1-2 Coded Datagram 8.1's Computation by bit shifting

A simple left shift of all the contents by 2 bits generates the coded data.

CE1=4 3 3 d 1 5 5 0 8 0 b ... 11 0011 1101 0001 0101 0101 0000 1000 0000 1011  $\ldots$ .... .... 3 2 8 3 4 2 8 С 0011 1100 0010 1000 0011 0100 0010 1000

Figure 4.10: Group Size 1: CE1-4 Coded Datagram 8.1's TCP Header

The above depicts the actual contents of the network coded segment with CE1=4.



Figure 4.11: Group Size 1: Coded Datagram 8.3's Relevant TCP Header fields

The modified TCP-NEWT header using a coefficient of 8 is depicted above.



Figure 4.12: Group Size 2: CE1-2 Coded Datagram 8.1's Computation by bit shifting

A simple left shift of the contents by 3 bits generates the coded data.

CE1=8 6 7 а 2 а 1 0 1 7 ... а 110 0111 1010 0010 1010 1010 0001 0000 0001 0111 ... .... .... 7 8 5 0 6 8 5 0 0111 1000 0101 0000 0110 1000 0101 0000

Figure 4.13: Group Size 1: CE1-8 Coded Datagram 8.3's TCP Header

The above depicts the actual contents of network coded segment with CE1=8.

Kir	nd : 25	length	n : 16	network	coded : 1	group	size			
	group id : 8									
16	а <sup>н</sup>	8		ares V	<b>-</b>					

Figure 4.14: Group Size 1: Coded Datagram 8.4's Relevant TCP Header fields

The modified TCP-NEWT header for a Coefficient of 16 is depicted above.



Figure 4.15: Group Size 2: CE1-16 Coded Datagram 8.1's Computation by bit shifting

A simple left shift of the contents by 4 bits generates the coded data.

CE1=16 С 4 5 5 4 2 0 2 f .... f 1100 1111 0100 0101 0101 0100 0010 0000 0010 1111 ... .... .... f 0 а 0 d 0 0 а 1111 0000 1010 0000 1101 0000 1010 0000

Figure 4.16: Group Size 1: CE-16 Coded Datagram 8.1's TCP Header

The above depicts the actual contents of network coded segment with CE1=16.

### Receiving Side Group Size 1

The figure 4.17 depicts the receive mechanism when there is a single segment.



Figure 4.17: Group Size 1 RX

Receipt of any one coded segment would suffice to recompute the original

segment, by a simple bit shift operation to the right according to the value of the CE1, coefficient 1 in the TCP Header of the received coded segment.

If CE1=2, right shift by 1 bit; if CE1=4, right shift by 2 bits; if CE1=8, right shift by 3 bits and if CE1=16, right shift by 4 bits to generate original Segment D1.

#### Acknowledgment for Group Size 1

For a group size of one, if an acknowledgment for any one coded segment is received, it confirms receipt of the data.

One of the contributions of TCP-NEWT is ensuring full accurate health of the network is captured and relayed back as-is to the higher layer and applications by the combined TCP stack. In the example depicted, four coded segments are sent, and the receiver receives only two of them and the acknowledgments for both reach the sender. In the eventuality of more packet drops and not one reaching the receiver side or none of the acknowledgments from the received side reaching the sender, TCP-NEWT computes a new unique coded segment, simply by added two of the prior coded segments  $CD_{8.1}$  and  $CD_{8.2}$  to generate  $CD_{8.5}$  with CE1 field in TCP header for  $CD_{8.5} = CE1$  (from  $CD_{8.1}$  header) + CE2 (from  $CD_{8.2}$  header); Similarly,  $CD_{8.1}$  and  $CD_{8.1}$  header) + CE2 (from  $CD_{8.2}$  header); Similarly,  $CD_{8.6} = CE1$  (from  $CD_{8.1}$  header) + CE2 (from  $CD_{8.3}$  header).

To compute the current RTO, a TCP sender maintains two state variables, SRTT and RTTVAR. TCP-NEWT computes the RTO at the end of receipt of acknowledgment for each of the 4 coded segments transmitted using the exact method outlined in RFC 2988 [66].

When the first RTT measurement R is made, the host MUST set: SRTT  $\leftarrow$  R RTTVAR  $\leftarrow$  R/2 RTO  $\leftarrow$  SRTT + max (G, K\*RTTVAR) (Where K = 4)

For every subsequent RTT measurement R' in each NC group, the sender updates RTTVAR and SRTT for TCP-NEWT window, as follows till measurements for all coded segments are completed.





$$\begin{split} &\operatorname{RTTVAR}_{8.1} \leftarrow (1\operatorname{-beta})^*\operatorname{RTTVAR}_8 + \operatorname{beta}^* \mid \operatorname{SRTT} - \operatorname{R}_{8.1}' \mid \\ &\operatorname{SRTT}_{8.1} \leftarrow (1 \operatorname{-alpha}) * \operatorname{SRTT}_8 + \operatorname{alpha} * \operatorname{R}_{8.1}' \\ &\operatorname{RTTVAR}_{8.2} \leftarrow (1 \operatorname{-beta})^*\operatorname{RTTVAR}_{8.1} + \operatorname{beta} * \mid \operatorname{SRTT}_{8.1} - \operatorname{R}_{8.2}' \mid \end{split}$$

$$\begin{split} & \text{SRTT}_{8.2} \leftarrow (1 \text{ - alpha}) * \text{SRTT}_{8.1} + \text{alpha} * \text{R}_{8.2}' \\ & \text{RTTVAR}_{8.3} \leftarrow (1 \text{ - beta}) * \text{RTTVAR}_{8.2} + \text{beta} * | \text{SRTT}_{8.2} \text{ - R}_{8.3}' | \\ & \text{SRTT}_{8.3} \leftarrow (1 \text{ - alpha}) * \text{SRTT}_{8.2} + \text{alpha} * \text{R}_{8.3}' \\ & \text{RTTVAR}_{8.4} \leftarrow (1 \text{ - beta}) * \text{RTTVAR}_{8.3} + \text{beta} * | \text{SRTT}_{8.3} \text{ - R}_{8.4}' | \\ & \text{SRTT}_{8.4} \leftarrow (1 \text{ - alpha}) * \text{SRTT}_{8.3} + \text{alpha} * \text{R}_{8.4}' \end{split}$$

The RTTVAR and SRRT corresponding to  $CD_{8.4}$  from TCP-NEWT window are assigned to the updated RTTVAR and SRRT corresponding to completion of successful transmission of D1 and receipt of ACK.

 $RTTVAR_9 \leftarrow RTTVAR_{8.4}$ 

 $SRTT_9 \leftarrow SRTT_{8.4}$ 

## 4.2.0.2 Group Size 2



The figure 4.19 depicts the scenario where a group contains two TCP segments.

Figure 4.19: Group Size 2

#### Grp size 2 Sending Side

Figure 4.19 depicts the transmission of the two segments, D1 in figure 4.4 and D2 in figure 4.20. When there are two segments in the original TCP sliding window and there are no other segments pending queuing in from higher layers for this TCP session, then the group size (Orig-Grp-size) is set to 2. Depending on the loss-ratio, the number of coded segments generated ranges from 2 (no loss) to 4 (50% loss). In this example, the network coding group-id is 8 and the number of coded segments generated is 4.

474554202f20485454502f312e300d0a 557365722d4167656e743a2057676574 2f312e31312e340d0a4163636570743a 202a2f2a0d0a486f73743a207777772e 696574662e6f72670d0a436f6e6e6563 74696f6e3a204b6565702d416c697665 0d0a0d0a

Figure 4.20: Original Data Segment 2- Hex Dump

The linear combinations of the sample data comprising of the two TCP segments, segment-1 in figure 4.4 and segment-2 in figure 4.20, can be very easily computed.

Kir	nd : 25	length	1: 16	network	coded : 1	group 2	size	
group id : 8								
4	1°				» *			

Figure 4.21: Group Size 2: Coded Datagram 8.1's TCP Header

The modified TCP-NEWT header using a (CE1) coefficient of 4 for first seg-

ment D1 and (CE2) coefficient of 1 for 2nd segment D2 is depicted below.

RLC(2): 4\*D1+1\*D2 3 8 4 5 4 0 Ε... А А А 0011 1000 0100 0101 1010 1010 0100 1010 0000 1110 ... .... . . . 2 4 9 3 2 4 1 3 0100 1001 0011 0010 0100 0001 0011 0010

Figure 4.22: Group Size 2: CE1-2 Coded Datagram 8.1's TCP Data

 $CD_{8.1}$  is computed by adding 4\*D1 (figure 4.13) + 1\*D2 (figure 4.20).

Kir	Kind : 25 length : 16			network	coded : 1	group size 2			
group id : 8									
2	1	8		19 10	. *				

Figure 4.23: Group Size 2: Coded Datagram 8.2's TCP Header

RLC(1): 2\*D1+1\*D2 1 Е 5 С F F С 6 0 8 D ... 0001 1110 0101 1100 1111 1111 1100 0110 0000 1000 1101 ... ... . . . 2 В 1 Е 2 7 1 Е 0010 1011 0001 1110 0010 0111 0001 1110

Figure 4.24: Group Size 2: CE1-2 Coded Datagram 8.1's TCP Data

 $CD_{8.2}$  is computed by adding 2\*D1 (figure 4.13) + 1\*D2 (figure 4.20). Similarly  $CD_{8.3}$  is computed by adding 1\*D1 (figure 4.13) + 2\*D2 (figure 4.20). Similarly  $CD_{8.4}$  is computed by adding 1\*D1 (figure 4.13) + 4\*D2 (figure 4.20).

### Grp Size 2 Receiving Side

The figure 4.25 depicts the receive mechanism when there are two segments.



Figure 4.25: Group Size 2 RX

Receipt of any two coded segments out of the 4 transmitted coded segments suffices to recompute the two original segments.

(RLC2	- RLC1	.) = 2*[	22							
1	9	E	8	Α	Α	8	4	0	5	Ε
1	1001	1110	1000	1010	1010	1000	0100	0000	0101	1110
	1	E	1	4	1	Α	1	4		
	0001	1110	0001	0100	0001	1010	0001	0100		

Figure 4.26: Group Size 2: CE1-2 Coded Datagram 8.1's TCP Data

The original segments are retrieved from the received coded segments as follows: After inspecting the CE1, CE2 tuple values in the 2 received coded segments, and noting that received coded segment RLC2 has (4,1) and RLC1 has (2,1), TCP-NEWT computes (RLC2 - RLC1) and the result is depicted below, which is 2\*D1 (2 times the original data D1 of Segment-1).

1100	1111	100	101	101	100	0010	0000	0010	1111
С	F	4	5	5	4	2	0	2	F
0	1111	0	1010	0	1101	0	1010		
0	F	0	Α	0	D	0	Α		

Figure 4.27: Group Size 2: D1 obtained by RLC2 - RLC1 is divided by 2

Bit shift to the right by 1-bit (division by 2) results in original data D1 retrieval.

RLC(1): 2\*D1+1\*D2 5 1 Е С F F С 6 0 8 D ... 0001 1110 0101 1100 1111 1111 1100 0110 0000 1000 1101 ... ... ••• 2 В 1 Е 2 7 1 Е 0010 1011 0001 1110 0010 0111 0001 1110

Figure 4.28: Group Size 2:

2\*D1 is subtracted from (2\*D1 + 1\*D2) to retrieve the original data D2. Thus, both the original transmitted segments D1 and D2 are retrieved and available at receiver with minimal number of bit-shifting and addition operations.

#### Acknowledgment for Group Size 2

Ensuring that the full accurate health of the network is captured and relayed back to the higher layer by the combined TCP stack is one of the significant contributions of TCP-NEWT and this is accomplished for this scenario comprising a group size of 2 in a very similar manner to the approach, elaborated in detail earlier for group size 1. Reiterating just the key aspects below: Original TCP Stack keeps track of the RTO as well as RTTVAR and SRRT at a group level. However in TCP-NEWT, for each subsequent RTT measurement R' in each NC group, the sender updates RTTVAR and SRTT for TCP-NEWT window, as follows till measurements for all coded segments are completed as per RFC2988 [66]. Thereafter, the final TCP-NEWT RTTVAR and SRTT values are used to update the values of RTTVAR and SRTT in the original TCP before the start of the next group formation prior to its network coded segment generation. The RTTVAR and SRRT corresponding to  $CD_{8.4}$  from TCP-NEWT window are then assigned to the updated RTTVAR and SRRT corresponding to completion of successful transmission of D1 and D2 and receipt of ACKs.

 $RTTVAR_9 \leftarrow RTTVAR_{8.4}$  $SRTT_9 \leftarrow SRTT_{8.4}$ 

#### 4.2.0.3 Algorithm – TCP-NEWT

The algorithms for encoding and decoding a group of segments in TCP-NEWT are enumerated below. Once the receiver has received sufficient number of coded segments for a group, equal to the size of the group, the decoding steps are initiated. TCP-NEWT uses the Gaussian-elimination [28] procedure for solving a system of linear equations to decode and arrive at the original data sent.

#### Algorithm 3 decode(group)

1: while TCP Session is still active do

- 2: Wait for the receipt of a packet. if times out waiting, quit;
- 3: Determine the group (group\_id\_) of the received segment.
- 4: Determine the group size of the group, grp\_sz;
- 5: Determine if there is already a sink created to gather all segments of this group.
- 6: If not, create a new sink for this group and initialize grp\_rcv\_cnt, group receive count to 1;
- 7: Check If grp\_sz for this group equals grp\_rcv\_cnt for this sink
- 8: if yes pass the set of packets to the "GaussianElimination" function, which will return the original segments of this group.
- 9: end while

#### Algorithm 4 encode(group)

- 1: Determine the group (group\_id\_) of the set of packets to be encoded.
- 2: Size of the group, grp\_sz;
- 3: Determine the group\_seq\_num for each individual packet within the group
- 4: Based on LR (Loss\_Ratio), determine the number of encoded packets to be generated: numEncoded
- 5: for i = 1;  $i \le numEncoded$ ; i + do
- 6: Determine the unique set of NC Coefficients Tuples: CE[1], CE[2], ..., CE[numEncoded]

/\* (based on the group\_id\_ and group\_seq\_num for each encoded packet to be generated.) \*/

- 7: Clear RLC\_PAC[i];
- 8: Compute the contents of the coded packet.
- 9: for j = 1;  $j \leq grp_sz; j++ do$
- 10:  $RLC_PAC[i] = RLC_PAC[i] + CE[i][j] \times PAC[j]$
- 11: **end for**
- 12: Populate the packet hdr of RLC\_PAC[i] with the NC Coefficients used to generate it;
- 13: Upload RLC\_PAC[i] the newly generated coded packet into new TCP\_NWT window.
- 14: end for

## TCP-NEWT for Distributed object store (ICN [36]/CCN [41])

In this section, a proposal for a more generalized setting with receiver requiring retrieval of data as and when needed is outlined *(publish/subscribe model, applicable to ICNs (Information Centric Networks), CCNs (Content Centric Networks).* There are 2 distinct phases in this more generalized setting:



Figure 4.29: Distributed Object Storage with Network Coding



Figure 4.30: Distributed Object Storage with Selective Network Coding

#### 4.2.0.4 Transport phase

Transport phase is managed like the one outlined in 4.2.0.1 above except that approach is tweaked to account for worst-case scenarios (from OLR - observed Loss ratio which the receiver might encounter when they request for the data of interest). TCP-NEWT hard codes the WLR to 70% loss ratio for distributed object storage (corresponding in real-world to a transient flaky wireless link scenario with close to 70%



Figure 4.31: Distributed Object Storage with Compressed Network Coding

loss/drops in link-layer signal strength and interference issues, thus not really a "TRUE" congestion in the network. Secondly, the coded data is stored at many different prior identified and well-known distributed object stores. This aspect is like traditional multicast data delivery, except that the coded data is now being delivered to intermediate distributed object storage devices instead of end-consumers of the data and the coded data sent to each node is a different unique RLC for the same group. To enforce security and privacy, effort is taken to ensure that each individual distributed object storage node has less than the "minimal set of RLC packets for each group" and as a heuristic measure, TCP-NEWT proposes storing 90% of the RLC coded packets for every group at each storage node. This will ensure that it is not possible for the intermediate storage node to "recreate or regenerate" the original segments. However, there could be some losses in the transmission path. For example, if there is a 15% loss in the transmission path to a given node then (0.9 \* 0.85) = 0.765 or 76% of the RLC coded packets are available in that storage node.

#### 4.2.0.5 Receiver phase

The more generalized sub-problem statement for the receiver phase is as follows: There are 'n' (distributed storage) nodes, each having a different "loss-ratio" and a different "hop-count", "delay" in its path to a particular destination of interest.

The proposed algorithm produces that minimal optimal subset (n1) of nodes, such that the overall bandwidth usage as well as transmission delay is minimized, while ensuring that sufficient RLC datagrams reach the destination, so that zero retransmissions are needed. An algorithm for converging on the "optimal subset of individual nodes", which would be identified and requested to respond, in order to ensure 100% data available at the receiver (without having the need for any retransmission) is proposed below.

This is an example of a file being retrieved by an interested receiver. Each file has a unique object identifier specific to itself. File is an ordered collection of groups of RLC packets. It may have been split up originally by the original source node into a sequential ordered list of groups, where each group has a unique group sequence number. These RLC packets are all now saved in several distributed intermediate storage nodes and every one of these individual RLC packets will also have the "original file's unique Object identifier" in addition to the "specific group sequence number".

### 4.2.0.6 Algorithm – TCP-NEWT

- Receiver expresses interest in an object/data/file/information by broadcasting the "object ID" corresponding to it. For security purposes, the receiver should have authorization to request for this data.
- 2. Each intermediate storage node, which is a repository of some of the RLC packets corresponding to the "object ID", identifies the request and notes down the "time it took for the request to reach it: Treach" and the "hop count: hc-reach traversed from the receiver" and responds back to the receiver identifying itself as a repository of the partial data along with the "Treach" and "hc-reach" parameters as well as the lower bound of the percentage of RLC packets (rlc-percent) for each

group that it has in its storage.

- 3. The receiver would have heard back from each of the distributed data storage nodes and with the full round-trip metrics including the return path specifics "Treturn" & "hc-return". The receiver also starts with an initial estimate value of the loss-ratio for each path, which it subsequently keeps continuously updating to reflect reality as accurately as possible.
- 4. Since each RLC group would have a different number of packets, a request for a "specific unique percentage of the RLC datagrams", applicable for any given group" to each specific node, is initiated.
- 5. This problem reduces to a classic optimization problem of minimizing the delay while accounting for the potential loss while receiving from each node and factoring in the bandwidth constraints in the path to each. Thus, the send-request to each intermediate storage node has only one parameter namely: percentage of the RLC coded packets to send (send-RLC-Percent).

$$\frac{x_1}{r_1} + d_1 = \frac{x_2}{r_2} + d_2 \tag{4.1}$$

$$\frac{x_1}{r_1} + d_1 = \frac{x_3}{r_3} + d_3 \tag{4.2}$$

$$\frac{x_1}{r_1} + d_1 = \frac{x_n}{r_n} + d_n \tag{4.3}$$

$$k_1 x_1 + k_2 x_2 + \dots + k_1 x_n = x \tag{4.4}$$

For each node i, send-RLC-percent(i) is computed from LR(i), rlc-percent(i), Treach(i), Treturn(i) as well as hc-reach(i) and hc-return(i). Detailed performance analysis of TCP-NEWT for distributed object store is subject of future work.

## 4.3 Performance Results

Performance results with no-loss, 10% loss, 20% loss, 30% loss, 40% loss and 50% loss are shown below. It was ensured there were losses only on the wireless last-mile link and no real congestion in the network.



Figure 4.32: TCP Network Coding with No-Loss



TCP Network Coding with BEST-START VS Slow-Start - Throughput Comparison w 10%

Figure 4.33: TCP Network Coding with 10% Loss

TCP Network Coding with BEST-START VS Slow-Start - Throughput Comparison w 20%



Figure 4.34: TCP Network Coding with 20%Loss



TCP Network Coding with BEST-START VS Slow-Start - Throughput Comparison w 30%

Figure 4.35: TCP Network Coding with 30%Loss

TCP Network Coding with BEST-START VS Slow-Start - Throughput Comparison w 40%



Figure 4.36: TCP Network Coding with 40%Loss



TCP Network Coding with BEST-START VS Slow-Start - Throughput Comparison w 50%

Figure 4.37: TCP Network Coding with 50%Loss

# 4.4 Performance Comparison

As very succinctly evident in the results of the simulation in section 4.3, the performance has been maintained at the same level despite varying levels of errors, all the way from 10% loss in figure 4.33, 20% loss in figure 4.34, 30% loss in figure 4.35, 40% loss in figure 4.36 and finally even with 50% loss as seen in the figure 4.37. Comparing these with the loss-less scenario in figure 4.32, clearly shows TCP-NEWT can guarantee performance and throughput despite level of errors/losses with one big CAVEAT to remember, namely: these errors are ONLY due to wireless link-layer errors and NOT due to a true congestion in the network.

## 4.5 Summary

Thus **TCP-NEWT** protocol, **TCP** with a Network coding Enabled congestion Window Transformation, while incorporating network coding and creating a new network-coded congestion window housing all the newly coded data, captures all the network transient parameters and suitably passes them over to the original TCP stack. This ensures that TCP-NEWT is empowered to make all its real-time adjustments as it would in a normal TCP implementation with the observed network dynamics. TCP-NEWT proactively sends the right apt amount of additional network coded segments based on the error level scenario being simulated. The next chapter explores and proposes a mechanism for proactively detecting the error-level and adjusting to it in real-time on the fly dynamically during an existing TCP session.

# Chapter 5

# **TCP-PNC:**

# **TCP** Predictive Network Coding

In this chapter, **TCP-PNC** [87] (**TCP** with **P**redictive **N**etwork **C**oding) is introduced, which enhances **TCP-NEWT** (**TCP N**etwork-Coding **E**nhanced **W**indow **T**ransformation) by incorporating a real-time accurate goodput prediction mechanism, which is a new contribution.

## 5.1 Basic Operation

TCP-PNC [87] improves on TCP-NEWT by incorporating optimal networkcoding to proactively address packet loss without retransmission, by dynamically predicting the expected goodput (1 - loss-ratio) on an ongoing basis during the course of a TCP session. This ensures that the optimal amount of network coded packets is transmitted, nothing more and nothing less. The dynamic predictive mechanism resulting in the efficient optimal amount of network coded packets transmission is a new contribution in TCP-PNC. Additionally, TCP-NEWT already ensures all TCP session metrics are also suitably transformed and passed back to original TCP stack and this is a new contribution in this thesis. Sliding Window Protocol as used in TCP, which is being enhanced with the Sliding Window Transformation Protocol is yet another new contribution in this work.

Due to use of network coding, TCP-PNC and TCP-NEWT are in a better position to guarantee a predictable performance and SLAs (service level agreement on upper bounds for data delivery) for both data stores and writes as well as data fetch and reads. This is because retransmissions, due to transmission losses in WAN link, is absolutely minimal, if not completely absent, in the approach used. Additionally, network coding also guarantees data privacy, as only the "network coded" data is transported and not the actual data. The beauty of using network coding (NC) is that the amount of traffic it generates, and transports compared to non-coded transmission is exactly of the same order. There is, however, an increase in processing and storage. But, if the system is lossy or congested, or if diversity (e.g., multiple paths or caching proxies) can be exploited, NC can provide faster end-end data transport. Network coding of the data packets is done prior to their being dispatched for storage in the distributed storage system, which could be geographically dispersed across a WAN link. Based on the current dynamically estimated goodput of the network at a given point in time, using a new enhanced approach based off [95], the number of network coded data packets (n) to be generated from an initial dataset (m) of data packets is computed. These "n"

network coded packets are then prefixed with the same unique prefix (specific for this dataset), and they are transported across the WAN link (TCP/IP) and these network coded packets are stored in different smart storage devices (not the original data). By leveraging network coding to retrieve a data block (containing "m" units of data), it suffices if any "m" random linear combinations can be retrieved to reconstruct, decipher, and extract the actual "data-set" of interest.

A simple example helps to illustrate. If a file is sent in n chunks, the receiver must receive each chunk correctly to decode the file. If each transmission failed with probability (1-p) independently of other transmissions, each chunk takes 1/p transmissions (geometric distribution) to be received correctly and the file need n/p transmissions. If NC is used, the receiver still needs n linear combinations to decode the file. This also takes n/p transmission on average. However, with NC, any n linear combinations received enable correct decoding. Without NC, the exact n chunks must be received. Depending on the retransmission strategy, this can take much longer. The question is how much coding overhead to use in NC, so that an apt excessive number of combinations are sent, and this depends on perceived congestion or errors. If no errors are perceived, one transmission suffices with or without NC. For an example like "real-time streaming" applications, where the data is needed in real time for high-fidelity video and quality experience, there is no point in retransmission of lost packets. However, using the network coding approach, even when the network is almost perfect, with an extremely low loss-ratio, very close to ensuring 100% packets receipt at the same timeframe is a high probability and so the viewing experience becomes, with

HD video, perfect, as all frames are available at the destination in time. In summary, TCP as a mechanism ensures reliability through retransmission, but this takes time to detect a loss and subsequent retransmission. So, it is a reactive approach. TCP-PNC's approach tries to ensure reliability in real-time, using network coding and proactively sending the apt amount of random linear unique combinations (RLC) of the packets in each set, leveraging a predictive means to estimate the expected goodput. This is a new contribution in this thesis. So, it is a proactive approach. Additionally, prior NC proposals for working with Transmission Control Protocol (TCP) were incremental additions to the existing TCP implementations and so, certain intrinsic TCP metrics and parameters measuring RTT, delays, latencies as relayed back to host were not exactly accurate. TCP-NEWT and TCP-PNC incorporate a mechanism to ensure all the TCP session metrics for RLC segments are suitably returned to the original TCP session. This is another new contribution of this thesis.



Figure 5.1: Native TCP based Selective Network Coding



Figure 5.2: TCP-NEWT with Selective Network Coding

In the above approaches, all the data is not coded, the additional packets required per the identified goodput are the only network coded packets.



Figure 5.3: TCP-NEWT congestion window comprises only coded segments

Goodput for different range of the time periods "M" starting from 2 to about 32 is computed in every measurement period tm . Let  $l_M(k)$  be the payLoad deLivery (LL) ratio (goodput) of the k-th measurement, which is calculated as the number of delivered packets over the total number of packets sent during the latest **M** periods (see equation 5.1), where  $N_d(k)$  is the number of packets delivered in the k-th measurement period, and  $N_s(k)$  is the number of packets sent in the k-th measurement period.

$$l_M(k) = \frac{\left(\sum_{i=0}^{M-1} N_d(k-i)\right)}{\left(\sum_{i=0}^{M-1} N_s(k-i)\right)}, \quad M = 2, 3, 4, \cdots, 32$$
(5.1)

In real scenario M can be any value  $\geq 0$ :

$$M = 0, \ LL(k) = l_0(k) \tag{5.2}$$

$$M = 1, \ LL(k) = \frac{l_0(k) + 2^{-1}l_1(k)}{2^0 + 2^{-1}}$$
(5.3)

$$M = 2, \ LL(k) = \frac{l_0(k) + 2^{-1}l_1(k) + 2^{-2}l_2(k)}{2^0 + 2^{-1} + 2^{-2}}$$
(5.4)

$$LL(k) = \frac{2^0 \times l_0(k) + 2^{-1} \times l_1(k) + 2^{-2} \times l_2(k) + \dots}{2^0 + 2^{-n} \times l_n(k)}$$
(5.5)

# 5.2 Protocol Description

For every TCP session, TCP-PNC computes the observed Loss Ratio (**OLR: 1 - observed goodput**) at every 2 secs interval and saves it in a ObservedLossRatio, a variable array containing the values observed in the last one hour (array size is 3600/2 = 1800). Using the past saved values of the observed loss ratio, along with currently observed loss ratio, extrapolation of the gradient and prediction of the PLR (Predicted Loss-Ratio) value(s) expected in the next 2 secs (PLR(t0 + 2) as well as next 4 (PLR (t0 + 4) secs is done. Based on this trend, the max (LR(t0), PLR (t0+2), PLR (t0+4)) is chosen as the "WLR (t0)" potential "Worst-case Loss-Ratio" scenario to be addressed while deriving the number of RLC (Random Linear Coded) TCP segments. Given a TCP window size (n) at a given point in time, TCP-PNC's design restricts the max grp size of the number of TCP datagrams to be Network Coded as a group to 8. One of the main considerations in keeping a smaller max group size value is to ease the computational burden on the end-nodes (some of which may have potential processing and/or memory constraints). Ceiling function  $\lceil (N/8) \rceil$  is the number of network coded groups. There will be 2 sliding windows, one with the "original segments" (orig\_SlidingWnd) and the other with the "random linear coded segments" (coded\_slidingWnd), wherein each of the packets are enhanced with the additional fields as listed above in (3). One of the key contributions in TCP-PNC and TCP-NEWT is preserving all the TCP handshake parameters seamlessly and accurately including the transient parameters like RTT and acknowledgments received (which would correspond to the "coded\_SlidingWnd' and accurately reflect that back to the 'Orig\_SlidingWnd'). If an acknowledgment is confirmed for receipt of "K" RLC packets at the receiver, WLR at this current point is immediately computed and floor function |K\*WLR| is the corresponding number by which the Orig\_SlidingWnd is increased. Additionally, another additional pointer is maintained for start and end of each RLC group. Till the receiver acknowledges the successful receipt of the required number of coded packets needed to be able to compute and generate the original datagrams of this group, these 2 pointers are maintained. This is just in case additional RLC packets are needed at the receiver due to more than expected packet losses and to enable the receiver to have the minimal number of coded segments required to generate the original segments. The TCP header 4.1 in each RLC packet of TCP-PNC generated for a given network coded group is the same as TCP-NEWT and illustrated in figure 4.1.

#### 5.2.0.1 TCP-PNC Dynamic Goodput Computation Algorithm

#### Orig\_Grp\_size ( $\leq 8$ )

RLC coefficients (the number of coefficients is exactly equal to the Orig\_Grp\_size
– listed in (i) above)

Unique Group ID – Group Sequence number for each group (like the sequence number for individual packets) TCP-PNA ensures that the Random Linear Codes used for generating each of the new coded packets are a unique tuple of dimension Orig\_Grp\_size (max possible is 8)

 $Coded\_Grp\_size = Orig\_Grp\_size/(1 - WLR)$ . Another important consideration is to ensure that the choice of coefficients is such that – the random linear coefficients of any 8 (actually  $Orig\_Grp\_size$ ) is such that its determinant can be evaluated (at the receiver in order to be able to regenerate the Original packets in the group – in any possible scenario of the order of the actual RLC grp packets received by the receiver) Window

segments-sent, segments-acked



Figure 5.4: LL Goodput Computation

Algorithm 5 TCP-PNC Overview

```
1: LR_NUM = 0;
 2: LR_{-}DEN = 0;
 3: cum_sent = 0;
 4: cum_ack = 0;
 5: M = 32;
 6: time_slot = 1;
 7: prevLR = 0.5;
 8: LR = 0.5; LR1= 0.5; PLR2=0.5; PLR3=0.5; PLR4=0.5
 9: curLR = 1
10: do
       start = time\_slot \mod M;
11:
       j = start;
12:
       term = 0;
13:
14:
       do
           \operatorname{cum\_sent} += \operatorname{cum\_sent} + \operatorname{tm}[j].\operatorname{sent};
15:
           cum_ack += cum_sent + tm[j].ack;
16:
           LR_NUM += (1/(2 * * term))^* (cum_ack/cum_sent);
17:
           LR_DEN += 1/(2 ** term);
18:
           term +=1;
19:
20:
           j=j-1;
       while j \ge 1
21:
       j = M-1;
22:
       do
23:
24:
           \operatorname{cum\_sent} += \operatorname{cum\_sent} + \operatorname{tm}[j].\operatorname{sent};
           um_ack += cum_sent + tm[j].ack;
25:
           LR_NUM += (1/2^{**}term)^* (cum_ack/cum_sent);
26:
           LR_DEN += 1/(2^{**}\text{term});
27:
28:
           term +=1;
29:
           j = j - 1;
       while j <= start
30:
       curLR = (LR_NUM/LR_DEN);
31:
32:
       LR1 = curLR;
       PLR2 = LR1 + (LR1 - prevLR);
33:
       PLR3 = PLR2 + (LR1 - prevLR);
34:
       PLR4 = PLR3 + (LR1 - prevLR);
35:
       WLR = MIN(prevLR, LR1, PLR2, PLR3, PLR4);
36:
       CODED_GRP_SZ = ORIG_GRP_SZ/(1 - WLR);
37:
       prevLR = curLR
38:
39: while (TCP Session is still active)
```
#### **TCP-PNC - Protocol Description by Example**



Figure 5.5: An Example of Dynamic Goodput (1 - Loss-Ratio) Prediction

An actual example is taken below, to succinctly illustrate TCP-PNC's proposed mechanism for dynamically arriving at the predicted loss at the next upcoming time interval. Value of M determines the number of time periods over which the loss ratio is computed. TCP-PNC incorporates an assignment of a weight of 1 for loss ratio  $l_0(k)$ ,  $2^{-1}$  for  $l_1(k)$ ,  $2^{-2}$  for  $l_2(k)$  and so on, to ensure the data comprising just the immediate past is given a higher importance compared to the data corresponding to a slightly larger duration from the past. In the example below, with the actual sample data, it is observed that in the k-th measurement, out of 4 packets sent, 2 are successfully received and acknowledged. In the (k+1)th measurement, out of 4 packets sent, 3 are successfully received and acknowledged. Loss-Ratio(LR) = 1 - payLoad-deLivery-Ratio (LL)

$$M = 0; \quad l_0(k-1) = \frac{3}{4} = 0.75;$$
 (5.6)

$$LL(k-1) = \frac{1 * l_0(k-1)}{1} = 0.75$$
(5.7)

$$LR(k-1) = 1 - LL(k-1) = 1 - 0.75 = 0.25$$
 (5.8)

$$M = 0; \quad l_0(k) = \frac{2}{4} = 0.5 \tag{5.9}$$

$$M = 1; \quad l_1(k) = \frac{3+2}{4+4} = \frac{5}{8} = 0.625 \tag{5.10}$$

$$LL(k) = \frac{1 * l_0(k) + (1/2) * l_1(k)}{1 + 1/2} = 0.54$$
(5.11)

$$LR(k) = 1 - LL(k) = 1 - 0.54 = 0.46$$
 (5.12)

$$M = 0; \ l_0(k+1) = \frac{3}{4} = 0.75$$
 (5.13)

$$M = 1; \quad l_1(k+1) = \frac{3+2}{4+4} = \frac{5}{8} = 0.625 \tag{5.14}$$

$$M = 2; \quad l_2(k+1) = \frac{3+2+3}{4+4+4} = \frac{8}{12} = 0.67 \tag{5.15}$$

$$LL(k+1) = \frac{1 * l_0(k+1) + (1/2) * l_1(k+1) + (1/4) * l_2(k+1)}{1 + 1/2 + 1/4} = 0.70$$
(5.16)

$$LR(k+1) = 1 - LL(k+1) = 1 - 0.7 = 0.3$$
 (5.17)

Expected Dynamic Loss-Ratio: Using LR(k-1), LR(k), and LR(k+1), **TCP-PNC** predicts the **PLR** (**P**redicted Loss **R**atio) at the next three time-intervals PLR(k+2), PLR(k+3), PLR(k+4) using following simple mechanisms. TCP-PNC does a linear extrapolation of the Observed loss ratio values at (k) and (k+1) to arrive at PLR(k+2). LR(k) is 0.46 and LR(k+1) is 0.30 and therefore initial estimate for PLR(k+2) is 0.14. However, since in this example, 4 segments are sent in a timeslot, the actual possible values for  $l_0(k+2)$  are 0, 0.25, 0.5, 0.75 and 1. Since the initial estimate of 0.14 is between 0 and 0.25, TCP-PNC takes the higher of the two namely 0.25 as the PLR(k+2). Similarly taking the values of OLR(k+1) and PLR(k+2) and doing a similar linear extrapolation, TCP-PNC estimates PLR(k+3), which in this example turns out to be 0.25. Similarly taking PLR(k+2) and PLR(k+3), TCP-PNC estimates PLR(k+4), which also turns out to be 0.25. Next TCP-PNC predicts the worst-case loss-ratio by taking the minimum of the observed loss ratio in the last two measurement periods and the predicted loss ratio in the upcoming two measurement periods:

MAX(LR(k), LR(k+1), PLR(k=2), PLR(k=3), PLR(k=4)), namely MAX(0.46, 0.30, 0.25, 0.25, 0.25), which is 0.46. This is closest to 0.5, which would be the worst case

loss-ratio in the above example. For a given session, at every 2 secs interval the Observed Loss-Ratio (OLR) is computed and saved in a LossRatioTable for last hour (array size is 3600/2 = 1800). Using the past saved values of the observed loss ratio, along with currently observed loss ratio extrapolation of the gradient/trend and prediction of the PLR (Predicted Loss-Ratio) values for the next 3 time periods is done. Based on this trend, the MAXIMUM (OLR(t0-4), OLR(t0-2), PLR(t0), PLR(t0+2), PLR(t0+4)) is chosen as the WLR(t0) potential Worst-case Loss-Ratio scenario to be addressed while deriving the number of RLC TCP datagrams. Coded\_Grp\_size = Orig\_Grp\_size/(1 - WLR).

#### 5.3 Performance Results

The performance of TCP-PNC [87] was evaluated using discrete-event simulation. The NS-2 simulator [81], which provides substantial support for simulation of TCP, Routing, and Multicast Protocols over wired and wireless (local and satellite) networks, was used. The TCP implementation was modified to support the new proposed protocols. This section describes simulations from 6 scenarios:

- a. standard ns-2 [81] TCP-newReno [26]
  - (i) Using a wireless topology with an almost lossless wireless link
  - (ii) Using the same wireless topology with a substantial lossy wireless link at both the wireless end-nodes
- b. standard ns-2 [81] TCP-cubic [32]

- (i) Using a wireless topology with an almost lossless wireless link
- (ii) Using the same wireless topology with a substantial lossy wireless link at both the wireless end-nodes
- c. standard ns-2 [81] new Reno implementation modified with TCP-PNCs enhancements for networks with wireless end-nodes.
  - (i) Using the same wireless topology with an almost lossless wireless link
  - (ii) Using the same wireless topology with a substantial lossy wireless link (10% and subsequently 20%)

There was an improvement in overall throughput observed with the new implementation, especially as the transmission errors (link-layer losses) increase. Comparative results with TCP Cubic as well as TCP newReno [26] shows a significantly higher throughput with TCP-PNC.

#### 5.4 Performance Comparison

Observe an overall increase of end-to-end TCP throughput in the range of 30-35% depending upon the amount of wireless-link-layer induced retransmissions, without changing the underlying "network congestion". TCP-PNC's results show that its proposed approach significantly improves the throughput of TCP connections both for:

1. short data transfer bursts: due to its unique feature of starting remarkably close to the available bandwidth rather than the traditional slow start mecha-



Figure 5.6: Cubic vs New Reno vs NC

nism in TCP implementations as well as due to quick proactive retransmissions due to wireless-link-layer failures and hiding the wireless-link-layer delays due to retransmission coupled with Network coding.

2. (2) Longer TCP sessions: as the wireless-link-layer issues and delays are isolated from network congestion issues (without discarding the data completely when there is wireless-link-layer issue, as is done in many of other enhancements/proposals for augmenting TCP for wireless and wired networks). Significant improvement in real-time delivery of data with predicable timelines at the



Figure 5.7: NC - Throughput



Figure 5.8: Cubic vs New Reno vs NC - 20% loss

end-application level.

## 5.5 Summary

The results show that TCP-PNC [87] along with TCP-NEWT can seamlessly adapt and provide real-time guaranteed delivery of data in a 5G network with different varying wireless-link-layer error levels and signal strengths (as long as the underlying core network is not experiencing any real congestion).

## Chapter 6

## **TCP-RTA:**

# TCP Real-Time Topology Adaptiveness for Congestion Control

TCP-RTA [88] recognizes the possibility of underlying topology changes happening during the course of an ongoing TCP session. This is particularly relevant for TCP sessions with mobile end-nodes, whose mobility during a TCP session could cause a significantly impactful underlying topology change. It has been observed that different existing TCP implementations and their variants incorporate a single congestion control strategy, which is fixed and cannot be changed dynamically. The response of the current congestion control mechanisms implemented in different TCP variants, do not account for the possibility of an inherent topology change as the cause of change in one of the observed TCP parameters instead of a change in congestion in the network: for example, a sudden significant increase in RTT is seen only as an increase in congestion in the network, instead of being open for the possibility of a "change in the path taken by the packets due to a topology change involving an addition of a wireless satellite link". The above indicated limitation of TCP implementations result in a sub-optimal or in several scenarios highly inefficient usage of the available bandwidth, resulting in low throughput.

Prior related research contributions, include "D-TCP: Dynamic TCP Congestion Control Algorithm for Next Generation Mobile Networks" [42], wherein Bandwidth-Delay product is dynamically computed and a congestion metric derived off this computation, is used to determine the response of the congestion control algorithm to increase/decrease the CWND during the RTT update and loss detection. Thus, only a single parameter is being dynamically modified and the underlying Congestion Control algorithm is the same for all scenarios and through the life cycle of the current session and thereafter till the TCP stack is changed.

TCP-RTA [88] dynamically recognizes potential underlying topology change in the end-end path of an existing TCP session and incorporates the capability to adapt in real-time, to the apt congestion control algorithm applicable for this updated new topology, experienced by the TCP Session.

#### 6.1 Basic Operation

There are proven very well performing and optimized custom congestion-control strategies for specific environments as exemplified by some of the TCP variants including TCP Hybla [15] for satellite links, HSTCP [25] for networks with a large bandwidthdelay product along with low-latency, as well as some generic TCP variants like TCP NewReno [26], among others. A new mechanism TCP-RTA [88] is proposed, wherein apriori categorized values of some of the ranges of TCP parameters (primarily RTT) as corresponding to a particular underlying nature of the environment (topology) are leveraged. TCP-RTA uses the above information to help identify the actual environment encountered by a TCP Session at any given point during the given session. On detection of a significant consistent change in the TCP session parameters of interest (RTT in this scenario), which point to a topology/environment change, TCP-RTA transitions over completely to the custom congestion control strategy, which is apt for the transitioned environment/topology.

An example would be a significant consistent increase in RTT. It would imply a change of the environment to a "satellite: very slow speed link" from a regular environment. TCP-RTA's mechanism would respond as follows: (1) start with default TCP-NewReno [26], (2) detection of the significant consistent RTT increase (3) conclude that the new increased RTT values point to a transition to a path involving a satellite (4) therefore initiate a switch over of the congestion-control algorithm from that of TCP-NewReno's [26] congestion-control algorithm to that of TCP-Hybla's [15] congestion control algorithm.

#### 6.2 **Protocol Description**

TCP-RTA [88] uses the following high-level approach to effect the transition across congestion control algorithms: It dynamically adapts the congestion control strategy as enumerated below. For any TCP session, TCP-RTA starts with a default configuration including congestion control strategy, which is chosen as TCP-NewReno for this initial study and simulation. This default TCP configuration can be any variant of TCP which is apt for the environment of the TCP session as it is established. One of the proposed enhancements that is envisaged in future is the inclusion of negotiation and convergence on the apt initial default configuration during the 3-way handshake in the setup phase of any TCP session. The corresponding default RTT thresholds for each environment is also deemed to be available and configured during the initial 3-way handshake. For the default TCP-NewReno configuration, the corresponding default RTT-threshold is initialized to 300 milli-seconds (ms). In TCP-RTA's dynamic adaptive algorithm, the last three observed RTTs at any point in time, RTT-Current, RTT-Prev, RTT-Prev-Prev are saved, wherein:

RTT-Current: RTT for the most recent segment.

RTT-Prev: RTT for the segment prior to the most recent segment (prior segment).

RTT-Prev-Prev: RTT for the segment prior to the prior segment.

All the above 3 variables are initialized at the start of a session to the default-RTT-threshold (300 ms in chosen test environment). After receipt of every acknowledgment and the corresponding immediate computation of the observed RTT (RTT-new), TCP-RTA updates the value of RTT-Prev-Prev with RTT-Prev, RTT-Prev with RTT-Current and RTT-Current with RTT-new. If the three observed values of RTT are all above 800 ms, TCP-RTA infers that there must have been an underlying topology change based on some of the observed RTT times for TCP sessions going over a satellite link [15], and initiates a change in the congestion control strategy which is more apt for the newly observed dynamically changed environment, wherein the TCP session now includes a path through a significantly larger delay (typically attributed to a satellite link), namely TCP Hybla [15]. TCP-RTA repeats the above steps till the end of the TCP session, with an additional check happening after every update to the observed RTT. if TCP-RTA finds that the last three observed RTT values are all below RTT-threshold for a non-satellite link, which has chosen as 300 ms based on reported observations in [15], it reverts back the congestion control strategy to TCP-NewReno. The high-level algorithm below clearly depicts the control flow of the newly proposed dynamic Adaptive TCP and its congestion control strategy. The main underlying premise is that if there is a distinct change suddenly observed in the RTT and that change is consistently maintained for at least 3 consecutive segments back-to-back, then TCP-RTA predicts the cause of such a change should be an underlying topology change rather than a sporadic congestion in the network. As part of the proposed approach in TCP-RTA [88], it is also ensured that TCP-RTA's adaptive congestion control strategy does not

respond to any sporadic one-off outliers in the TCP parameters. TCP-RTA ensures that the hand-off from the current congestion control strategy happens seamlessly to the appropriate target congestion control strategy for the newly identified topology to which the network has transitioned to. Numerous experiments were conducted with the same thresholds for transitions across different congestion control strategies as well as having a common gray area, whose boundaries had to be crossed consistently three times by the observed-RTT values.

Algorithm 6 TcpAdaptive-Overview		
1: $Def_RTT_TCP_NEWRENO = 500;$		
2: $Def_RTT_TCP_HYBLA = 800;$		
3: $m_{adaptiveAlgProg} = TCP_NEWRENO;$		
4: $RTT$ -Current = Def_RTT_TCP_NEWRENO;		
5: $RTT-Prev = Def_RTT_TCP_NEWRENO;$		
6: $RTT$ -Prev-Prev = Def_RTT_TCP_NEWRENO;		
7: <b>do</b>		
8: <b>if</b> $(m_{\text{adaptiveAlg}} = \text{TCP}_{\text{NEWRENO}})$ <b>then</b>		
9: <b>if</b> ((RTT-Current > Def_RTT_TCP_HYBLA) &&		
$(RTT-Prev > Def_RTT_TCP_HYBLA) \&\&$		
$(RTT-Prev-Prev > Def_RTT_TCP_HYBLA))$ then		
10: $m_adaptiveAlg = TCP_HYBLA;$		
11: end if		
12: <b>else if</b> (m_adaptiveAlg == TCP_HYBLA) <b>then</b>		
13: <b>if</b> ((RTT-Current < Def_RTT_TCP_NEWRENO) &&		
$(RTT-Prev > Def_RTT_TCP_NEWRENO) \&\&$		
$(RTT-Prev-Prev > Def_RTT_TCP_NEWRENO))$ then		
14: $m_adaptiveAlg = TCP_NEWRENO;$		
15: end if		
16: end if		
17: wait till next ACK received;		
18: $\operatorname{RTT-new} = \operatorname{ComputeNewRTT}();$		
19: $\operatorname{RTT-Prev-Prev} = \operatorname{RTT-Prev};$		
20: $\operatorname{RTT-Prev} = \operatorname{RTT-Current};$		
21: $\operatorname{RTT-Current} = \operatorname{RTT-new};$		
22: while (TCP Session is still active)		

Algorithm 7 TcpAdaptive::UpdateAdaptiveAlg (const Time &rtt)

```
1: if m_adaptiveAlgProg == ALOG_INPROGRESS then
      if m_adaptiveAlgProgCnt \leq 0 then
2:
         m_{adaptiveAlgProg} = ALOG_{COMPLETED};
3:
4:
         m_adaptiveAlgProgCnt =ADAPTIVE_ALG_PROGRESS_CNT;
      end if
5:
6: end if
7: if (m_adaptiveMode) then
      if (m_{adaptiveAlg} = TCP_NEWRENO) then
8:
9:
         if (rtt > Seconds (0.800)) then
            ++m_adaptiveDetectCnt;
10:
         else
11:
12:
            m_{adaptiveDetectCnt} = 0;
         end if
13:
         if (m_adaptiveDetectCnt \geq ADAPTIVE\_SWITCH\_CNT) then
14:
            m_adaptiveAlg = TCP_HYBLA;
15:
            m_adaptiveAlgProg = ALOG_INPROGRESS;
16:
            m_{adaptiveDetectCnt = 0;
17:
            m_{adaptiveAlgProgCnt} = ADAPTIVE_ALG_PROGRESS_CNT;
18:
         end if
19:
20:
      end if
21: else if (m_adaptiveAlg = TCP_HYBLA) then
      if (rtt < Seconds (0.800)) then
22:
         ++m_adaptiveDetectCnt;
23:
24:
      else
25:
         m_{adaptiveDetectCnt = 0;
      end if
26:
      if (m_adaptiveDetectCnt \geq ADAPTIVE_SWITCH_CNT) then
27:
         m_{adaptiveAlg} = TCP_NEWRENO;
28:
         m_{adaptiveAlgProg} = ALOG_INPROGRESS;
29:
30:
         m_{adaptiveDetectCnt = 0;
         m_adaptiveAlgProgCnt=ADAPTIVE_ALG_PROGRESS_CNT;
31:
32:
      end if
33: end if
```

Algorithm 8 TcpAdaptive::SlowStart	
1:	input Ptr_SocketState, segmentsAcked
2:	procedure TCPADAPTIVE_SLOWSTART
3:	if (segmentsAcked $\geq 1$ && m_adaptiveAlg == TCP_NEWRENO) then
4:	$sndCwnd = tcb \rightarrow m_cWnd;$
5:	$tcb \rightarrow m_cWnd =$
	$\min((\text{sndCwnd}+(\text{segmentsAcked}^{tcb} \rightarrow \text{m\_segmentSize})), \text{ tcb} \rightarrow \text{m\_ssThresh});$
6:	<b>return</b> segmentsAcked-((tcb $\rightarrow$ m_cWnd-sndCwnd) / tcb $\rightarrow$ m_segmentSize);
7:	else if (segmentsAcked $\geq 1$ && m_adaptiveAlg == TCP_HYBLA) then
8:	/* slow start
9:	$INC = 2^{\rho} - 1 * /$
10:	$increment = pow(2, m_{-}\rho) - 1.0;$
11:	$incr = increment^*tcb \rightarrow m\_segmentSize;$
12:	$tcb \rightarrow m_cWnd = min (tcb \rightarrow m_cWnd + incr, tcb \rightarrow m_sThresh);$
13:	return segmentsAcked - 1;
14:	end if
15:	return 0;
16:	end procedure

Algorithm 9 TcpAdaptive::CongestionAvoidance 1: input Ptr\_SocketState, segmentsAcked, 2: procedure TCPADAPTIVE\_CONGESTIONAVOIDANCE while (segmentsAcked > 0 && m\_adaptiveAlg == TCP\_HYBLA) do 3: INC =  $\rho^2$  / W \*/ 4:  $segCwnd = tcb \rightarrow GetCwndInSegments$  (); 5:increment = std::pow  $(m_{\rho}, 2)$  / static\_cast<double> (segCwnd); 6: 7:  $m_cWndCnt +=$  increment; segmentsAcked -= 1;8: end while 9: if (segmentsAcked > 0 &  $m_adaptiveAlg = TCP_NEWRENO$ ) then 10: if (m\_adaptiveAlgProg = ALOG\_INPROGRESS) then 11:  $w = tcb \rightarrow m_cWnd / tcb \rightarrow m_segmentSize;$ 12:if (w == 0) then w = 1; 13:end if 14: if  $(m_cWndCnt \ge w)$  then 15: $m_{c}WndCnt = 0;$ 16: $tcb \rightarrow m_cWnd += tcb \rightarrow m_segmentSize;$ 17:end if 18: $m_cWndCnt +=$  segmentsAcked; 19:if  $(m_cWndCnt > w)$  then 20: 21:  $delta = m_cWndCnt / w;$  $m_{c}WndCnt -= delta * w;$ 22: $tcb \rightarrow m_cWnd += delta * tcb \rightarrow m_segmentSize;$ 23:end if 24: end if 25:26:else 27:m\_adaptiveAlgProgCnt-;  $tcb \rightarrow m_cWnd = m_bd / tcb \rightarrow m_segmentSize;$ 28:end if 29:if  $(m_cWndCnt \ge 1.0 \&\& m_adaptiveAlg == TCP_HYBLA)$  then 30: 31:  $inc = m_cWndCnt;$ m\_cWndCnt -= inc; if (m\_adaptiveAlgProg = ALOG\_INPROGRESS) 32:  $tcb \rightarrow m_cWnd += inc * tcb \rightarrow m_segmentSize;$ 33: 34: else  $tcb \rightarrow m_cWnd = m_bd / tcb \rightarrow m_segmentSize;$ 35: end if 36: 37: end procedure

Adaptive TCP incorporates the following list of enhancements:

- Slow Start Enhancement In TCP-NewReno, cwnd is increased by one segment per acknowledgment. In TCP-RTA, cwnd is changed to SegAcked \* Segment size (like Cubic [32]).
- 2. Congestion Avoidance Enhancement NewReno, cwnd is increased by (1/cwnd) In TCP-RTA, the following changes are introduced: In congestion avoidance phase, the number of bytes that have been ACKed at the TCP sender side are stored in a 'bytes\_acked' variable in the TCP control block. When 'bytes\_acked' becomes greater than or equal to the value of the cwnd, 'bytes\_acked' is reduced by the value of cwnd. Next, cwnd is incremented by a full-sized segment SMSS (SENDER MAXIMUM SEGMENT SIZE). (Similar to Linux Reno [37] implementation)
- 3. On Fast-retransmit, TCP-RTA updates ssthresh to half of current cwnd: ssthresh = bytesInFlight/2.
  To recover faster, it is enhanced as follows:

ssthresh = (bytesInFlight \* 2) /3.

4. Default boost of a factor of 10 (constant) of the Bandwidth\*Delay product while switching from LAN to Satellite and vice versa.

TCP-RTA has not impacted or changed any of the fairness with respect to other TCP sessions co-existing as the underlying congestion control strategy adopted by TCP-RTA is that of TCP-Hybla, when the topology change is detected through a consistent increase in RTT. The fairness of TCP-Hybla and earlier that of TCP-NewReno has been already established and proven and thus its applicable in this proposed solution. Even in the transition from congestion control strategies from TCP-Hybla to TCP-NewReno, the only change is TCP-RTA's non-responsiveness to transients and that too for only 3 segments. Thus, fairness is guaranteed.

#### 6.3 Performance Results

Several scenarios and options were tried out to truly validate the gains and benefits of TCP-RTA. After close analysis of the various findings, "Tcp-NewReno" and "TCP-Hybla" were chosen for simulation. The behavior for the quite significantly impactful change of a topology going through a local LAN in a home office or a corporate network to a data path involving a satellite for wireless inducing a very highly significant additional delay in the observed RTT, was studied. Unless the congestion control strategy detects a topology change and evaluates the updated "Bandwidth X Delay" product to confirm it, TCP-RTA will not tamper with the currently in-place congestion control mechanism.

As the results below succinctly indicate, a clear increase in the CWND size on transition to satellite environment is observed, though accompanied by a significantly higher RTT. NS3 was used for the simulation and a significant delay from time t=5 secs to time t=15 secs was injected, to simulate a transition to a satellite back-haul and a subsequent transition back from it.



Figure 6.1: CWND: Adaptive vs Hybla vsFigure 6.2: TX: Adaptive vs Hybla vs NewReno NewReno



Figure 6.3: CWND before transition to

Figure 6.4: TX before transition to satellite: Adaptive vs Hybla vs NewReno satellite: Adaptive vs Hybla vs NewReno





Figure 6.5: CWND after transition to Satellite link: Adaptive vs Hybla vs NewReno



260 Adaptive Tx ----

Figure 6.6: TX after transition to Satellite

link: Adaptive vs Hybla vs NewReno



Figure 6.7: CWND after transition from Satellite link: Adaptive vs Hybla vs NewReno

Figure 6.8: TX after transition from Satellite link: Adaptive vs Hybla vs NewReno

### 6.4 Performance Comparison

For the simulation studies, consistently observed RTT values in the range upwards of 800 ms were chosen to denote an environment comprising of a topology with a satellite back-haul. Similarly, RTT values consistently observed in the range downwards of 800 ms were earmarked for one set of experiments and for others a lower value 500 ms, to denote an environment/topology without a satellite back-haul. Later, various other RTT ranges could be added as needed to correspond to specific topology/environments, for which a specific apt TCP variant with its own congestion control algorithm has been identified. This would ensure that the best optimal bandwidth usage and performance for that environment is provided. It can be observed from the results that there is a significant boost in the optimal efficient usage of the network bandwidth. This idea can be extrapolated, and it does not bring any restriction to the usage of the few specific topologies used in the simulations. As soon as an underlying change to the topology is observed, not restricted to detection by only RTT changes, and the corresponding best congestion control approach for that topology is identified, TCP-RTA [88] can be leveraged and extended very easily in a pluggable manner to incorporate the corresponding congestion control strategy.

#### 6.5 Summary

As the results depict, the adaptive TCP provides a framework and mechanism for leveraging the "apt" congestion control strategy for a given dynamic scenario, thus ensuring that the network is used in the most optimal efficient manner all the time. The underlying design and approach used in Adaptive TCP lends itself to seamlessly incorporate other specific scenarios and the transition to the corresponding congestion control strategy. Currently the mechanism used to detect "topology change" in this thesis has been RTT and the RTT variations with time. However, TCP-RTA's approach does not preclude other usage of any other metrics or a combination of metrics to identify and determine any significant network change. To scale TCP-RTA proposal further, TCP-RTA's framework does not preclude and would permit machine learning as well as AI techniques for predicting proactively impending topology and environment change, so an apt congestion control strategy can be dynamically invoked in real-time to always ensure continuous ubiquitous efficient usage of network resources and bandwidth.

## Chapter 7

# Conclusion

The continuing seismic after-effects of the ALOHA channel, with its underlying simple philosophy of transmission at will and if the transmitter is unsuccessful then it will retransmit at some random time in the future over a wireless medium, along with the groundbreaking demonstration of a wireless packet-switching network based on it launched a revolution on packet switching over wireless links. The need to support wireless links and packet switching through a hybrid network involving a mixture of wired and wireless networks has been ever increasing and reached its zenith with the 5G technology coming of age and all carriers transitioning over it across the globe.

The many applications and compelling 5G technology driven use-cases as outlined in Figures 1.2, 1.3, 1.4, 1.5, 1.6, 1.7 and 1.8 mandate a compelling overhaul of the currently deployed transport mechanisms in order to provide the infrastructure and support for these very mission and time-critical applications requiring guaranteed successful data delivery with upper-bound time limits. The study and research started with outlining all these diverse use cases and arriving at a set of features, capabilities and requirements for a new transport mechanism for next-gen 5G technology and beyond, enabled-internet of the future.

Chapter 3 presented **TCP** Enhanced Wireless Santa Cruz (*TCP-EWSC*), which covers the enhancements for the seamless adaptation of the original TCP to today's 5G technology-based hybrid networks. Among the several features incorporated, include the capability to learn from history of steady state TCP session parameters of prior TCP sessions 3.2.1 and keeping a TCP-session persistent during transient temporary complete drop and absence of the wireless signal 3.2.4.

Chapter 4 introduced **TCP-NEWT** (**TCP** with **N**etwork Coding Enabled **W**indow **T**ransformation), which incorporates the best of network-coding while ensuring seamless throughput for any wireless layer signal strength and channel error issues.

Chapter 5 presented **TCP-PNC** (**TCP-P**redictive **N**etwork **C**oding) [87], which ensures that the apt optimal amount of additional network coded packets is introduced to ensure real-time delivery of data.

Chapter 6 described **TCP-RTA** (**TCP** - **R**eal-time **T**opology **A**daptiveness) [88], which proactively identifies underlying topology change and dynamically changes the congestion control algorithm to an appropriate mechanism so that the underlying available network infrastructure is put to optimal usage. This capability becomes especially important and critical for mobile end-nodes, including mobile vehicles. The systems leverage prior TCP session experiences which can be saved due to **increased working memory capacity** in today's devices and puts them to effective use for future similar situations.

There are a number of areas for future work based on the results we have introduced in this thesis. We mention below four enhancements of our work as examples.

TCP-EWSC can be further improved by incorporating support for ECN (Explicit Congestion Notification) [72] to help detect network congestion more accurately. In addition, the hard-coded weights used for the congestion window size in TCP-EWSC could be changed dynamically based on the actual observed transition window sizes.

TCP-PNC [87] uses weights corresponding to terms of binomial series for more longer historical time-based data, for predicting the impending goodput. Other models could be used for assigning weights for prior historical goodput observed values to arrive at goodput prediction.

Combining FEC (Forward Error Correction) [52] mechanisms in tandem with TCP-NEWT and TCP-PNC [87] to check if the combination of both together can get a better optimal solution is a future research area.

Finally, TCP-RTA [88] can be enhanced to dynamically determine the initial starting topology by monitoring the RTT values observed during the initial TCP 3-way handshake. This could ensure quicker convergence at the start of a TCP session. TCP-RTA can also be enhanced with an adaptive learning mechanism to dynamically tweak the RTT values used for low-water and high-water marks for a given topology based on the observations. Arriving at this new adaptive learning mechanism is a another compelling future research area.

## **Bibliography**

- Rasool Al-Saadi, Grenville Armitage, Jason But, and Philip Branch. A survey of delay-based and hybrid tcp congestion control algorithms. *IEEE Communications* Surveys and Tutorials, 21(4):3609–3638, 2019.
- [2] Khaled Alferaidi and Robert Piechocki. Tcp-mac cross layer integration for xor network coding. In *Science and Information Conference*, pages 860–875. Springer, 2018.
- [3] Mark Allman, Vern Paxson, Wright Stevens, et al. Tcp congestion control, 1999.
- [4] Andrea Baiocchi, Angelo P Castellani, and Francesco Vacirca. Yeah-tcp: yet another highspeed tcp. In Proc. PFLDnet, volume 7, pages 37–42, 2007.
- [5] Ajay V Bakre and BR Badrinath. Handoff and systems support for indirect tcp/ip.
   In Symposium on Mobile and Location-Independent Computing, pages 11–24, 1995.
- [6] Bikram S Bakshi, P Krishna, Nitin H Vaidya, and Dhiraj K Pradhan. Improving performance of tcp over wireless networks. In *Proceedings of 17th International Conference on Distributed Computing Systems*, pages 365–373. IEEE, 1997.

- [7] Hari Balakrishnan, Srinivasan Seshan, and Randy H Katz. Improving reliable transport and handoff performance in cellular wireless networks. Wireless Networks, 1(4):469–481, 1995.
- [8] Paul William Berg, Scott Isaacs, and Kelsey Lynn Blodgett. Airborne fulfillment center utilizing unmanned aerial vehicles for item delivery. https://patents. google.com/patent/US9305280, 2016.
- [9] Sumitha Bhandarkar, Nauzad Erach Sadry, AL Narasimha Reddy, and Nitin H Vaidya. Tcp-dcr: A novel protocol for tolerating wireless channel errors. *IEEE Transactions on Mobile Computing*, 4(5):517–529, 2005.
- [10] David Borman, R Braden, V Jacobson, and R Scheffenegger. Tcp extensions for high performance. *Request for Comments (Proposed Standard) RFC*, 1323, 1992.
- [11] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the conference* on Communications architectures, protocols and applications, pages 24–35, 1994.
- [12] Lawrence S. Brakmo and Larry L. Peterson. Tcp vegas: End to end congestion avoidance on a global internet. *IEEE Journal on selected Areas in communications*, 13(8):1465–1480, 1995.
- [13] Lawrence S. Brakmo and Larry L. Peterson. Tcp vegas: End to end congestion avoidance on a global internet. *IEEE Journal on selected Areas in communications*, 13(8):1465–1480, 1995.

- [14] Ramon Caceres and Liviu Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE journal on selected areas in communications*, 13(5):850–857, 1995.
- [15] Carlo Caini and Rosario Firrincieli. Tcp hybla: a tcp enhancement for heterogeneous networks. International journal of satellite communications and networking, 22(5):547–566, 2004.
- [16] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: congestion-based congestion control. *Communications of the ACM*, 60(2):58–66, 2017.
- [17] Claudio Casetti, Mario Gerla, Saverio Mascolo, Medy Y Sanadidi, and Ren Wang.
   Tcp westwood: end-to-end congestion control for wired/wireless networks. Wireless Networks, 8(5):467–479, 2002.
- [18] Vinton Cerf and Robert Kahn. A protocol for packet network intercommunication. IEEE Transactions on communications, 22(5):637–648, 1974.
- [19] Emina Soljanin Christina Fragouli. Network coding fundamentals. https://www. nowpublishers.com/article/Details/NET-003, 2007.
- [20] Erik Dahlman, Stefan Parkvall, and Johan Skold. 5G NR: The next generation wireless access technology. Academic Press, 2020.
- [21] Michiel JL De Hoon, Seiya Imoto, John Nolan, and Satoru Miyano. Open source clustering software. *Bioinformatics*, 20(9):1453–1454, 2004.

- [22] Disaster and Safety Communications Network Korea Safe-net. 5g network deployment scenarios. https://www.mois.go.kr/eng/sub/a03/bestPractices7/ screen.do, 2018.
- [23] Kevin Fall and Sally Floyd. Simulation-based comparisons of tahoe, reno and sack tcp. ACM SIGCOMM Computer Communication Review, 26(3):5–21, 1996.
- [24] Y. Feng and December 2006. G. Hamerly.. "pg-means: learning the number of clusters in data". 20th annual conference on neural information processing systems (NIPS), 2006.
- [25] Sally Floyd. Rfc3649: Highspeed tcp for large congestion windows, 2003.
- [26] Sally Floyd, Tom Henderson, and Andrei Gurtov. Rfc3782: The newreno modification to tcp's fast recovery algorithm, 2004.
- [27] Gabriel. Sliding window protocol. https://www.slideserve.com/ file-download/797317, 2012.
- [28] Carl Friedrich Gauss. Gaussian elimination method. https://mathworld. wolfram.com/GaussianElimination.html, 1850.
- [29] Tom Goff, James Moronski, Dhananjay S Phatak, and Vipul Gupta. Freeze-tcp: A true end-to-end tcp enhancement mechanism for mobile environments. In Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064), volume 3, pages 1537–1545. IEEE, 2000.

- [30] NATO IST-124 Research Task Group. 5g network deployment scenarios. https: //anglova.net/, 2022.
- [31] Cheng Jin David X. Wei Steven H. Caltechfast tcp: From theory to experiments,. Low Engineering & Applied Science, 2005.
- [32] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. ACM SIGOPS operating systems review, 42(5):64–74, 2008.
- [33] Tomoya Hatano, Masahiko Fukuhara, Hiroshi Shigeno, and Ken-Ichi Okada. Tcpfriendly sqrt tcp for high speed networks. *Proceedings of APSITT (November 2003)*, pages 455–460, 2003.
- [34] Yong Huang, Majid Ghaderi, Don Towsley, and Weibo Gong. Tcp performance in coded wireless mesh networks. In 2008 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, pages 179–187. IEEE, 2008.
- [35] IANA. 5g network deployment scenarios. https://www.iana.org/assignments/ tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1, 2022.
- [36] ICNRG. Information-centric networking. https://datatracker.ietf.org/rg/ icnrg/about/, 2012.
- [37] V Jacobson. "modified tcp congestion avoidance algorithm,". *Technical Report*,, 1990.

- [38] Van Jacobson. Congestion avoidance and control. ACM SIGCOMM computer communication review, 18(4):314–329, 1988.
- [39] Van Jacobson, Robert Braden, and David Borman. Tcp extensions for high performance. Technical report, RFc 1323, May, 1992.
- [40] Van Jacobson, Robert Braden, and Lixia Zhang. Tcp extension for high-speed paths. Technical report, RFC-1185, October, 1990.
- [41] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '09, page 1–12, New York, NY, USA, 2009. Association for Computing Machinery.
- [42] Madhan Raj Kanagarathinam, Sukhdeep Singh, Irlanki Sandeep, Abhishek Roy, and Navrati Saxena. D-tcp: Dynamic tcp congestion control algorithm for next generation mobile networks. In 2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC), pages 1–6, 2018.
- [43] P. Karn and C. Partridge. "improving round-trip time estimates in reliable transport protocols". In Computer Communication Review,, 1987.
- [44] Ghulam Kassem, Imran Ahmad, Fozia Hameed, and Asif Zakariyya. Tcp variants: An overview. In 2010 Second International Conference on Computational Intelligence, Modelling and Simulation, pages 536–540, 2010.

- [45] Madiha Kazmi, Muhammad Younas Javed, and Muhammad Khalil Afzal. An overview of performance comparison of different tcp variants in ip and mpls networks. In Simon Fong, editor, *Networked Digital Technologies*, pages 120–127, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [46] Tom Kelly. Scalable tcp: Improving performance in highspeed wide area networks. ACM SIGCOMM computer communication Review, 33(2):83–91, 2003.
- [47] Gur KimchiDaniel, Buchmueller, Scott A. Green, Brian C. Beckman, Scott IsaacsAmir, NavotFabian HenselAvi, and Severan Sylvain Jean-Michel Rault Bar-Zeev. Unmanned aerial vehicle delivery system. https://patents.google.com/patent/ US20150120094, 2015.
- [48] TV Lakshman and Upamanyu Madhow. The performance of tcp/ip for networks with high bandwidth-delay products and random loss. *IEEE/ACM transactions* on networking, 5(3):336–350, 1997.
- [49] Douglas J Leith, Robert N Shorten, and Gavin McCullagh. Experimental evaluation of cubic-tcp, 2008.
- [50] Wang Lian, David W Cheung, and SM Yiu. An efficient algorithm for finding dense regions for mining quantitative association rules. Computers & Mathematics with Applications, 50(3-4):471–490, 2005.
- [51] D. Lin and H.T. Kung. Tcp fast recovery strategies: analysis and improvements. In Proceedings. IEEE INFOCOM '98, the Conference on Computer Communications.

Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No.98, volume 1, pages 263–271 vol.1, 1998.

- [52] Sui Luyi, Fu Jinyi, and Yang Xiaohua. Forward error correction. In 2012 Fourth International Conference on Computational and Information Sciences, pages 37– 40, 2012.
- [53] S.Floyd M. Mathis, J. Mahdavi and A. Romanow. "tcp selective acknowledgment options, rfc 2018,, 1996.
- [54] Fabio Martignon and Luigi Fratta. Loss differentiation schemes for tcp over wireless networks. In International Workshop on Quality of Service in Multiservice IP Networks, pages 586–599. Springer, 2004.
- [55] Takahiro Matsuda, Taku Noguchi, and Tetsuya Takine. Survey of network coding and its applications. *IEICE transactions on communications*, 94(3):698–717, 2011.
- [56] Daniil S Meitis, Danil S Vasiliev, Albert Abilov, and Irina Kaysina. Comparison of tcp congestion control algorithms in network coded relaying scheme. In 2019 International Siberian Conference on Control and Communications (SIBCON), pages 1–4. IEEE, 2019.
- [57] Arthur Holland Michel. Amazon's drone patents. https://dronecenter.bard. edu/files/2017/09/CSD-Amazons-Drone-Patents-1.pdf, 2017.

- [58] Vasuky Mohanan, Rahmat Budiarto, and Ismat Aldmour. Powering the internet of things with 5G networks. IGI Global, 2017.
- [59] Wooseok Nam, Dongwoon Bai, Jungwon Lee, and Inyup Kang. Advanced interference management for 5g cellular networks. *IEEE Communications Magazine*, 52(5):52–60, 2014.
- [60] Liadan O'callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings* 18th International Conference on Data Engineering, pages 685–694. IEEE, 2002.
- [61] Information Sciences Institute University of Southern California. Rfc793: Transmission control protocol, 1981.
- [62] Nathan Michael Paczan. Managing unmanned vehicles. https://patents. google.com/patent/US9671791, 2017.
- [63] Nathan Michael Paczan, Michael John Elzinga, and Raphael HsiehLuan Khai Nguyen. Collective unmanned aerial vehicle configurations. https://patents. google.com/patent/US20160378108, 2016.
- [64] C. Parsa and J.J. Garcia-Luna-Aceves. Differentiating congestion vs. random loss: a method for improving tcp performance over wireless links. In 2000 IEEE Wireless Communications and Networking Conference. Conference Record (Cat. No.00TH8540), volume 1, pages 90–93 vol.1, 2000.
- [65] Christina Parsa and Jose Joaquin Garcia-Luna-Aceves. Improving tcp congestion

control over internets with heterogeneous transmission media. In *Proceedings. Sev*enth International Conference on Network Protocols, pages 213–221. IEEE, 1999.

- [66] V. Paxson and M. Allman. Rfc 2988: Computing tcp's retransmission timer, 2000.
- [67] C. E. Perkins. "ip mobility support". rfc 2002 ip mobility working group internet draft,, 1996.
- [68] Jon Postel. Rfc0761: Dod standard transmission control protocol, 1980.
- [69] Quy-Vu-Khanh. 5g network deployment scenarios. https: //www.researchgate.net/profile/Quy-Vu-Khanh/publication/ 329587507/figure/fig1/AS:703015312965634@1544623320065/ An-example-of-Mobile-Ad-hoc-Network-applied-in-5G.jpg, 2018.
- [70] Quy-Vu-Khanh. 5g network deployment scenarios. https://pdfs. semanticscholar.org/2d98/d84caddae953f1d3b2758d7de39864282bff.pdf, 2018.
- [71] Quy-Vu-Khanh. 5g network deployment scenarios. https://dl.acm.org/doi/10.1145/2248326.224834, 2018.
- [72] Kadangode Ramakrishnan, Sally Floyd, David Black, et al. The addition of explicit congestion notification (ecn) to ip, 2001.
- [73] Rapeepat Ratasuk, Athul Prasad, Zexian Li, Amitava Ghosh, and Mikko A Uusitalo. Recent advancements in m2m communications in 4g networks and evolution
towards 5g. In 2015 18th International Conference on Intelligence in Next Generation Networks, pages 52–57. IEEE, 2015.

- [74] Karunaharan Ratnam and Ibrahim Matta. Effect of local retransmission at wireless access points on the round trip time estimation of tcp. In *Proceedings 31st Annual Simulation Symposium*, pages 150–156. IEEE, 1998.
- [75] Karunaharan Ratnam and Ibrahim Matta. Wtcp: An efficient mechanism for improving tcp performance over wireless links. In *Proceedings Third IEEE Symposium* on Computers and Communications. ISCC'98.(Cat. No. 98EX166), pages 74–78. IEEE, 1998.
- [76] Karunaharan Ratnam and Ibrahim Matta. Wtcp: an efficient mechanism for improving wireless access to tcp services. International journal of communication systems, 16(1):47–62, 2003.
- [77] R.Srinivasan. Efficient tcp enhancements for hybrid networks. Master's thesis, University of California, Santa Cruz, 2016.
- [78] Hamlet Medina Ruiz, Michel Kieffer, and Béatrice Pesquet-Popescu. Redundancy adaptation scheme for network coding with tcp. In 2012 International Symposium on Network Coding (NetCod), pages 49–54. IEEE, 2012.
- [79] Hamlet Medina Ruiz, Michel Kieffer, and Béatrice Pesquet-Popescu. Tcp and network coding: Equilibrium and dynamic properties. *IEEE/ACM Transactions* on Networking, 24(4):1935–1947, 2015.

- [80] V. Paxson S. Bradner. Rfc2780: Iana allocation guidelines for values in the internet protocol and related headers, 2000.
- [81] S. Floyd S. McCanne and K. Fall. Network simulator. Public domain software, 1995.
- [82] Bhaskar Sardar and Debashis Saha. A survey of tcp enhancements for last-hop wireless networks. *IEEE Communications Surveys and Tutorials*, 8(3):20–34, 2006.
- [83] Phillipa Sessini and Anirban Mahanti. Observations on round-trip times of tcp connections, 2006.
- [84] C. E. Shannon. A mathematical theory of communication. The Bell System Technical Journal, 27(3):379–423, 1948.
- [85] spacelink. 5g network deployment scenarios. https://www.eosspacelink.com/ realizing-the-benefits-of-the-spacelink-relay-service, 2022.
- [86] DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION. Internet protocol, 1981.
- [87] R. Srinivasan and J.J. Garcia-Luna-Aceves. Optimized network coding with realtime loss prediction for hybrid 5g networks. In Proc. International Conference on Ubiquitous Networking '22. Montreal, QC, Canada, October 25–27, 2022.
- [88] R. Srinivasan and J.J. Garcia-Luna-Aceves. Tcp-rta: Real-time topology adaptiveness for congestion control in tcp. In Proc. International Conference on Ubiquitous Networking '22. Montreal, QC, Canada, October 25–27, 2022.

- [89] Jiyan Sun, Yan Zhang, Ding Tang, Shuli Zhang, Zhijun Zhao, and Song Ci. Tcp-fnc: a novel tcp with network coding for wireless networks. In 2015 IEEE International Conference on Communications (ICC), pages 2078–2084. IEEE, 2015.
- [90] Jay Kumar Sundararajan, Devavrat Shah, Muriel Médard, Michael Mitzenmacher, and Joao Barros. Network coding meets tcp. In *IEEE INFOCOM 2009*, pages 280–288. IEEE, 2009.
- [91] Kun Tan, Jingmin Song, Qian Zhang, and Murad Sridharan. A compound tcp approach for high-speed and long distance networks. In *Proceedings-IEEE INFO-COM*, 2006.
- [92] Yuechen Tao, Jingjie Jiang, Shiyao Ma, Luping Wang, Wei Wang, and Bo Li. Unraveling the rtt-fairness problem for bbr: A queueing model. In 2018 IEEE Global Communications Conference (GLOBECOM), pages 1–6, 2018.
- [93] Celplan Engineerng team. 5G/IoT Network Design and Deployment. https:// www.celplan.com/services/5g-iot-design-deployment.
- [94] Ye Tian, Kai Xu, and Nirwan Ansari. Tcp in wireless environments: problems and solutions. *IEEE Communications Magazine*, 43(3):S27–S32, 2005.
- [95] Chonggang Wang, Jiangchuan Liu, Bo Li, Kazem Sohraby, and Y Thomas Hou. Lred: a robust and responsive aqm algorithm using packet loss ratio measurement. *IEEE Transactions on Parallel and Distributed Systems*, 18(1):29–43, 2006.

- [96] Jingyuan Wang, Jiangtao Wen, Jun Zhang, Zhang Xiong, and Yuxing Huan. Tcpfit: An improved tcp algorithm for heterogeneous networks. *Journal of Network* and Computer Applications, 71, 04 2016.
- [97] Kai Xu, Ye Tian, and Nirwan Ansari. Tcp-jersey for wireless ip communications. IEEE Journal on selected areas in communications, 22(4):747–756, 2004.
- [98] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *IEEE INFOCOM 2004*, volume 4, pages 2514–2524. IEEE, 2004.
- [99] Jiong Yang. Dynamic clustering of evolving streams with a single pass. In Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405), pages 695–697. IEEE, 2003.