Performance Enhancement Proxy for Heterogeneous Networks

A Thesis Submitted to

Nirma University

In Partial Fulfillment of the Requirements for

The Degree of

Doctor of Philosophy

in

Technology and Engineering

by

Rakesh D. Vanzara (09EXTPHDE28)

Institute of Technology
Nirma University
Ahmedabad-382481
Gujarat, India

March 2017

Nirma University
Institute of Technology
Certificate

This is to certify that the thesis entitled "Performance Enhancement Proxy for Heterogeneous Networks" has been prepared by Mr Rakesh D. Vanzara under our supervision and guidance. The thesis is his original work completed after careful research and investigation. The work of the thesis is of the standard expected of a candidate for Ph.D. Programme in Computer Science and Engineering and we recommend that it be sent for evaluation.

Date:

Dr Hareshs Bhatt
Dr Rajendra Sharma

Name and Signature of the Guide

Forwarded By:

Dr S. Garg

Name and Signature of the

Head of the Department

Name and Signature of the

Dean Faculty of Technology and Engineering

29/3/2017 (Dr. M. Shah)

Name and Signature of the

Dean Faculty of Doctoral Studies and Research

Executive Registrar

Nirma University
Institute of Technology
Declaration

I, Mr Rakesh D. Vanzara, registered as Research Scholar, bearing Registration Number 09EXTPHDE28 for Doctoral Programme under the Faculty of Technology and Engineering of Nirma University do hereby declare that I have completed the course work, pre-synopsis and my research work as prescribed under R. Ph. D. 3.5.

I do hereby declare that the thesis submitted is original and is the outcome of the independent investigations / research carried out by me and contains no plagiarism. The research is leading to the discovery of new techniques. This work has not been submitted by any other University or Body in quest of a degree, diploma or any other kind of academic award.

I do hereby further declare that the text, diagrams or any other material taken from other sources (including but not limited to books, journals and web) have been acknowledged, referred and cited to the best of my knowledge and understanding.

Date: 28/03/2017

Rakesh D. Vanzara

Name & Signature of the student

---

we endorse the above declaration made by the student.

Dr Hareshbhatt Dr Hareesh S. Bhatt

Dr Priyanka Sharma Dr Priyanka Sharma

Name & Signature of the Guide

Abstract

Transmission Control Protocol (TCP) is widely used in data communications for various types of applications. Application layer protocols like Hyper Text Transfer Protocol (HTTP), File Transfer Protocol (FTP), use TCP as transport layer protocol over heterogeneous networks. Links like wired, wireless (e.g., Wi-Fi, WLAN, WiMAX, 3G), and satellite constitute the heterogeneous networks which have varied Round Trip Time (RTT) and a mixture of congestive losses and link losses due to Bit Error Rate (BER). A variety of TCP variants exist that address the issues pertaining to specific link characteristics or application environments. On a wireless link, BER is normally in the range of 10^{-6} to 10^{-11} which is very high as compared to that of wired link (i.e., 10^{-12} to 10^{-14}). RTT also varies from 1 ms to 1500 ms for different link types. However, a protocol addressing the requirements of TCP for links with variable RTT and BER is equally desirable. In the present research work, this issue is addressed by designing a new TCP variant named as Tarang and a dynamic TCP layer architecture named as ADYTIA. ADYTIA addresses the issue of single variant usage for all networking scenarios and application environments.

Tarang is a new TCP protocol designed for the link with variable RTT and BER. It improvises the start-up performance on high RTT link by using the concept of normalized round trip time and a modified approach to increase the congestion window (cwnd) in slow-start phase. When congestion occurs, instead of halving cwnd as per the traditional Additive Increase Multiplicative Decrease (AIMD) approach, Tarang uses the utilized link bandwidth to set the value of cwnd and slow-start threshold (ssthresh). This way, Tarang can provide better throughput as compared to the default TCP variant (Cubic in Linux and NewReno in Windows family operating systems) and can replace the default TCP variant in various operating systems.

Tarang outperforms other existing TCP variants in most of the cases and also

maintains fairness. However, always there is a scope for developing new protocols giving better performance for specific cases. Furthermore, usually, fairness becomes a bottleneck in utilizing a specific costly link more effectively and also when a high priority application demands higher throughput. Existing TCP/IP implementations force operating systems to fix and use single TCP variant for all applications and links. This kind of rigid binding of TCP variant with host operating system results in poor performance for various emerging applications and upcoming networking technologies. Hence, Adaptive and Dynamic TCP layer Interface Architecture (ADYTIA) is proposed in this research work. It selects the best possible variant depending on the application and its link characteristics. Depending on the changes in the link characteristics (and link usage), ADYTIA dynamically changes the variant during the lifetime of communication flow. ADYTIA allows easy plugging of newly designed TCP variant and accordingly Tarang is plugged in ADYTIA. ADYTIA addresses fairness within a homogeneous network by using single optimized variant based on the information base that is created inherently. However, fairness is affected in complex heterogeneous networks. To address this issue, ADYTIA is integrated with Performance Enhancement Proxy (PEP) at gateways of complex heterogeneous networks.

The proposed work has been thoroughly tested on both simulation environment and testbed setup. ADYTIA along with PEP has also been tested on live Internet. Tarang was able to outperform other existing variants for variable RTT and BER. In a worst-case scenario with BER of 10^{-6} and RTT of 1500 ms (i.e., multi-hop satellite link), Tarang improved performance by 20-30 times compared to other existing variants. Based on the various experiments conducted with different combinations of link parameters (i.e., RTT, BER, Bandwidth), application types (e.g., FTP) and TCP variants, ADYTIA's ability to select best suitable variant results in improved performance compared to other single variant usages. With FTP as an application type and link with high bandwidth ($>$ 10 Mbps) and RTT (e.g. 600 ms), ADYTIA selects Hybla for a link without losses (e.g. BER 10^{-11}) and Tarang for a link with losses (e.g. BER 10^{-6}). In this case, the selection of the specially crafted variant results in 20% to 80% improvement in performance.

Briefly, this research work develops a new protocol Tarang (a TCP variant) for a link with variable BER and RTT that addresses most of the issues of heterogeneous

networks. Further, ADYTIA has been developed for the dynamic selection of TCP variant based on the application and link types. It also allows the change of variant based on the variation in link characteristics and the plugging of newly designed TCP variants. Furthermore, ADYTIA has been integrated with PEP to address the fairness and deployment issues. Hence, this research work allows an individual to design a protocol or a TCP variant that is focused on futuristic applications and links or a specific scenario. TCP variants' selection based on applications and links would motivate the research community to develop and deploy each newly designed variant on operating systems.

Acknowledgments

I take the pleasure to present this research work related to "Performance Enhancement Proxy for Heterogeneous Networks" to the Almighty, for being present in all my endeavors.

I would like to thank Dr Haresh S. Bhatt, Guide and ISO,CIO, Mission Director, Space Applications Centre, ISRO, Ahmedabad for motivating me and providing continuous support throughout my Doctoral studies. His critical questions at every steps help me lot to get into the depth and focused towards the research goals.

I would like to thank Dr Priyanka Sharma, Guide and Professor, Institute of Technology, Ahmedabad for motivating me and providing much needed support for my Doctoral studies. Her continuous feedback and suggestions were great help at important juncture.

My sincere thanks to the reviewers Dr D.C.Jinwala and Dr Sanjay Chaudhary, who had given their valuable feedback during my Research Progress Committee meetings. I would like to express my sincere regards to Dr Sanjay Garg, Head of Computer Science and Engineering Department for his never-ending support and cooperation. I am also thankful to all faculty members, staff members and students of our department for providing all types of resources and support in time. I can not forget to thank Dr. Priyank Thakkar, who helped me a lot.

I would also like to express my sincere thanks and regards to Dr. Sastri Kota, President of SoHum Consultants and Adjunct Professor, University of Oulu Finland,(who has made significant contributions to satellite communications and networking in both military and commercial environments) for his valuable suggestions in writing of the thesis.

There are no words to thank the Almighty for gifting me with a wonderful kids, who bore the wrath of my journey towards this research. My wife Priti needs

special accolade for her patience and constant support provided throughout this work. Finally, my sincere most thanks to the most important and special people of my life - my parents; who have been ever motivating, endearing and highly cooperative in all my endeavors. I thank one and all, who have kept encouraging and motivating me. Without everyone's support, this thesis is beyond imagination.

Rakesh D. Vanzara

09EXTPHDE28

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ACA ..... Application Classification Algorithm
ACK ..... Acknowledgment
ACM ..... Application Classification Module
ADYTIA ..... Adaptive and Dynamic TCP Interface Architecture
AIMD ..... Additive Increase and Multiplicative Decrease
AM ..... Adaptive Module
BDP ..... Bandwidth Delay Product
BER ..... Bit Error Rate
BIC ..... Binary Increase Congestion Control
BRTT ..... Base Round Trip Time
BSD ..... Berkeley Software Distribution
CA ..... Congestion Avoidance
CCA ..... Connection Classification Algorithm
CCL ..... Connection Classification Logic
CCM ..... Connection Classification Module
CP ..... Content Provider
CTP ..... Configurable and Extensible Transport Protocol
cwnd ..... congestion window
DUPACKs ..... Duplicate Acknowledgments
ERR ..... Eligible Rate Estimation
FTP ..... File Transfer Protocol
GEO ..... Geosynchronous Earth Orbit
HLL ..... High-speed, Loss Less
HRL ..... High-speed, High RTT with variable Losses
HTTP ..... Hyper Text Transfer Protocol
HWL ..... High-speed With Losses
IP ..... Internet Protocol
ISP ..... Internet Service Provider
JFI ..... Jain's Fairness Index
LP ..... Low Priority
LTE ..... Long-Term Evolution
MIMD ..... Multiplicative Increase and Multiplicative Decrease
NIC ..... Network Interface Card
PEP ..... Performance Enhancement Proxy
PER ..... Packet Error Rate
RTT ..... Round Trip Time
SACK ..... Selective Acknowledgment
SMTP ..... Simple Mail Transfer Protocol

SRTT ..... Smoothed Round Trip Time
SS ..... Slow Start
ssthresh ..... slow-start threshold
TCP ..... Transmission Control Protocol
T-HetNet ..... Testbed for Heterogeneous Networks
UDP ..... User Datagram Protocol
WiMAX ..... Worldwide Interoperability for Microwave Access

Chapter 1

Introduction

1.1 Background

Internet based accessibility and support have become pervasive in our daily life. The growth of smart devices, wireless technologies and seamless access to the Internet has allowed individuals to contribute towards the content generation of the Internet (Yang et al.). Such communication paradigm has given birth to diversified applications and unimaginable networking scenarios. Today, large data centers amalgamate to grid or cloud via the Internet. The Internet is considered as a backbone for all types of cloud based services. These developments lead to the deployment of varieties of networking technologies and platforms. New applications across diversified platforms and links have different expectations and parameters to be optimized. Hence, heterogeneity of networking technologies, platforms, upcoming applications and variety of communication scenarios require an accurate and customized transport layer variant to have the maximum utilization of the available link capacity (Carlo, Firrincieli, and Lacamera)

Transmission Control Protocol (TCP) is the most widely deployed transport layer protocol in one or the other form across all the platforms. Each TCP variant for the specific platform is tuned to host system and designed to give an optimum performance for the wired link. In heterogeneous networks, along with wired links, there are different kinds of wireless links (Caini et al.). These wireless links could be like Wi-Fi, WiMAX, 3G, 4G, and satellite link. Each link has different characteristics. Types of links and their characteristics have been summarized in Table 1.1 (Chen, Farley, and Ye), (Bhatt et al.), (Belshe), (Grigorik), (Chen et al.). As mentioned

in (Wang et al.), (Caini and Firrincieli), and (Saverio et al.), TCP's performance gets degraded as Round Trip Time (RTT) of the connection is increased. Similarly, increase in Bit Error Rate (BER) also degrade the performance of TCP flow. A simple experiment has been performed in ns-2 (Fall and Varadhan) to measure the link utilization for different values of RTT and BER.

Table 1.1: Characteristics of each type of link

Link Type	**Bandwidth**	**BER**	**RTT**
Wired link	10, 100, 1000 Mbps	10^{-12} to 10^{-14}	10 ms to 200 ms
Optical Fiber	1, 10 Gbps	10^{-12} to 10^{-14}	10 ms to 200 ms
Wi-Fi link	11, 54, 100 Mbps	10^{-8} to 10^{-11}	60 ms to 80 ms
WiMAX	15.4 Mbps	10^{-8} to 10^{-11}	60 ms to 100 ms
2G	100-400 Kbps	10^{-8} to 10^{-11}	300 ms to 1000 ms
3G	0.5-5.0 Mbps	10^{-8} to 10^{-11}	100 ms to 500 ms
4G	1-50 Mbps	10^{-8} to 10^{-11}	<100 ms
GEO Satellite link	10, 100, 1000 Mbps	10^{-6} to 10^{-11}	500 ms per hop

Simulation topology consists of sender and receiver with varying link characteristics has been shown in Figure 1.1. TCP NewReno (i.e., default variant in window flavor operating system) (Allman, Paxson, and Stevens Floyd and Henderson) was used as the TCP variant on sender node. The throughput of NewReno for the link with varying RTT has been shown in Table 1.2. In this experiment, the link was configured as error free link. Results show that as RTT increases, throughput gets degraded. In heterogeneous networks, RTT varies significantly which affect the throughput of the TCP variant.

In another experiment, RTT was assumed to be 200 ms, and BER varied from negligible to very high. As BER increases, throughput gets degraded as depicted in Table 1.3. In heterogeneous networks, each link has different BER, and it varies with

time. Application layer protocols across the heterogeneous platforms have different expectations from the transport layer protocol. Each application layer protocol would be trying to optimize different parameters and tolerate various ranges of BER and RTT. The need and tolerance level of some of the well-known applications have been summarized in Table 1.4 (Chen, Farley, and Ye). Many popular TCP variants designed for different types of links and platforms have been shown in Table 1.5 (Yang et al.).

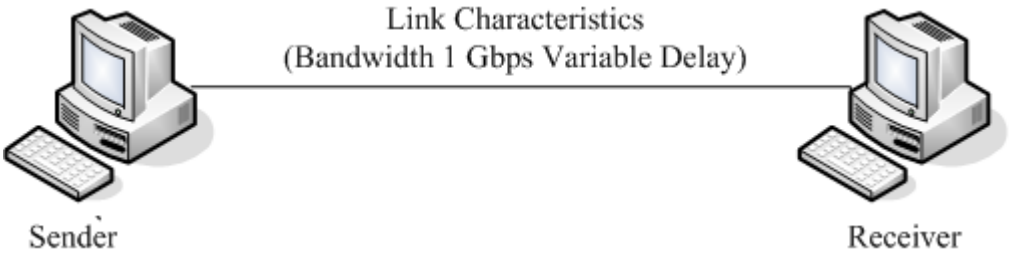


Figure 1.1: Simulation topology

Table 1.2: Throughput for different values of RTT

RTT (ms)	100	200	400	600	1000	1500
Throughput (Mbps)	950.35	919.4	853.66	784.11	409.39	0.056

Table 1.3: Throughput for different values of BER

BER	0	10^{-12}	10^{-11}	10^{-10}	10^{-9}	10^{-8}	10^{-7}	10^{-6}
Throughput (Mbps)	919	919	919	594.21	68.22	25.39	0.99	0.36

TCP variant NewReno (Henderson et al.) is most popular and used on windows family operating systems as a default variant (Yang et al.). It was designed for links with low to moderate RTT and negligible BER. Approaches like large value of initial window in slow-start phase (Allman, Floyd, and Partridge) and other variants (Padmanabhan and Katz), (Leung and Yeung),(Sally et al.), (Scharf, Hauger, and Kögel) were designed to overcome issue of slow-start phase for high Bandwidth

Delay Product (BDP) links. TCP variants like TCP-Vegas (Brakmo and Peterson), Westwood (Saverio et al.), Westwood+ (Grieco and Mascolo), TCP-Peach (Akyildiz, Morabito, and Palazzo), and Peach+ (Akyildiz, Zhang, and Fang) used bandwidth estimation technique to overcome the issue of link errors indirectly. Hybla (Caini and Firrincieli) was designed for high RTT links (e.g., satellite link), but in the presence of high BER, Hybla cannot utilize the link bandwidth effectively.

Table 1.4: Application types and expectations

Application Type	**Bandwidth Required**	**Loss Tolerance**	**Adaptability to variations in bandwidth, delay and loss**
Web browsing	medium to high	none	moderate
Email	low	very low	high
Live Interactive Voice	low	very low	limited
Live Interactive Video	medium	low	moderate
Stored Streaming Video	medium to high	low	high
Stored Interactive Video	medium to high	low	moderate
Remote Backup	high	none	high
Data Transfer	medium to high	none	low

Variants like Cubic (Ha, Rhee, and Xu), HighSpeed (Sally), Scalable (Kelly), Yeah (Baiocchi, Castellani, and Vacirca), Illinois (Liu, Başar, and Srikant), and TCP Fast (Wei et al.) were designed for the high BDP links but, performance was degraded in the presence of link losses. GridTCP (Bhatt et al.) was designed to overcome the issue of high RTT and BER. It was able to outperform other existing variants but,

Table 1.5: Few TCP variants and platforms

TCP Variant	**Platform**	**Designed for- link type**	**Characteristics**
NewReno / Compound TCP	Microsoft Window	Wired	Excellent performance with low to high RTT link. However, in presence of high BER, high RTT, its performance is degraded
Cubic	Linux kernel 2.6.19 onwards	Wired, high RTT	Able to utilize the link fully in most of the cases, but performs poorly in presence of high BER
Westwood	Linux	Wireless	Good performance even in presence of link losses. However, it performs poorly in presence of high RTT
Hybla	Linux	Satellite	It equalize the performance of flows with high RTT to that of competing low RTT flow. However, its performance is degraded in presence of high BER

it had few operational issues like, an end-user was needed to specify the number of connections to be used and also required modifications on sender and receiver both. All these variants have been presented in Chapter 2. Furthermore, each variant has been designed for a particular type of link. However, the heterogeneity of network technologies and platforms, unique characteristics of each link, and different expectations of each application suggests the need for some dynamic mechanism (Johansson). This type of mechanism must consider the application layer protocols and network link characteristics to use the most suitable TCP variant to maximize the link utilization and improvise the performance. Thus, data communication has to happen with the most suitable variant for the link and application in use and must be independent

of the platform. Further, as referred from (Sarkar and Gutie´rrez), simulations' results are not the reflection of the real system parameters and may lead to a different conclusion. Based on these justifications, the problem statement of this research work has been presented below.

1.2 Problem Statement

Heterogeneity of networking technologies and platforms, emerging new applications and its expectations, and complex communication scenarios require dynamic mechanisms to cope with challenges in the heterogeneous networks. Following key issues were observed:

- **Need of customized TCP variant:** Existing TCP variants have been designed to perform well with the specific type of link only. However, in heterogeneous networks, existing variants have the performance issues for links with variable BER and RTT.

- **Single TCP variant for all types of links and applications:** All modern operating systems support usage of default and single TCP variant at a time for all applications (i.e., uses TCP variant) and networking technologies. This kind of approach results in performance bottleneck due to the heterogeneity of platforms, applications, and networking technologies.

- **Simulations and testbed experiments:** It is important to verify the simulation results on the testbed. A newly designed mechanism tested on simulator needs to be modified for the testbed experiments that require lots of efforts in system tuning and setup because of code rework and real system parameters.

- **Deployment challenges:** A newly designed mechanism needs to be ported to each platform. Acceptance of the end-user for modifications in the operating system kernel is challenging and also it requires code rework for each platform. Hence, deployment of the mechanism has to be independent concerning the end-user platform.

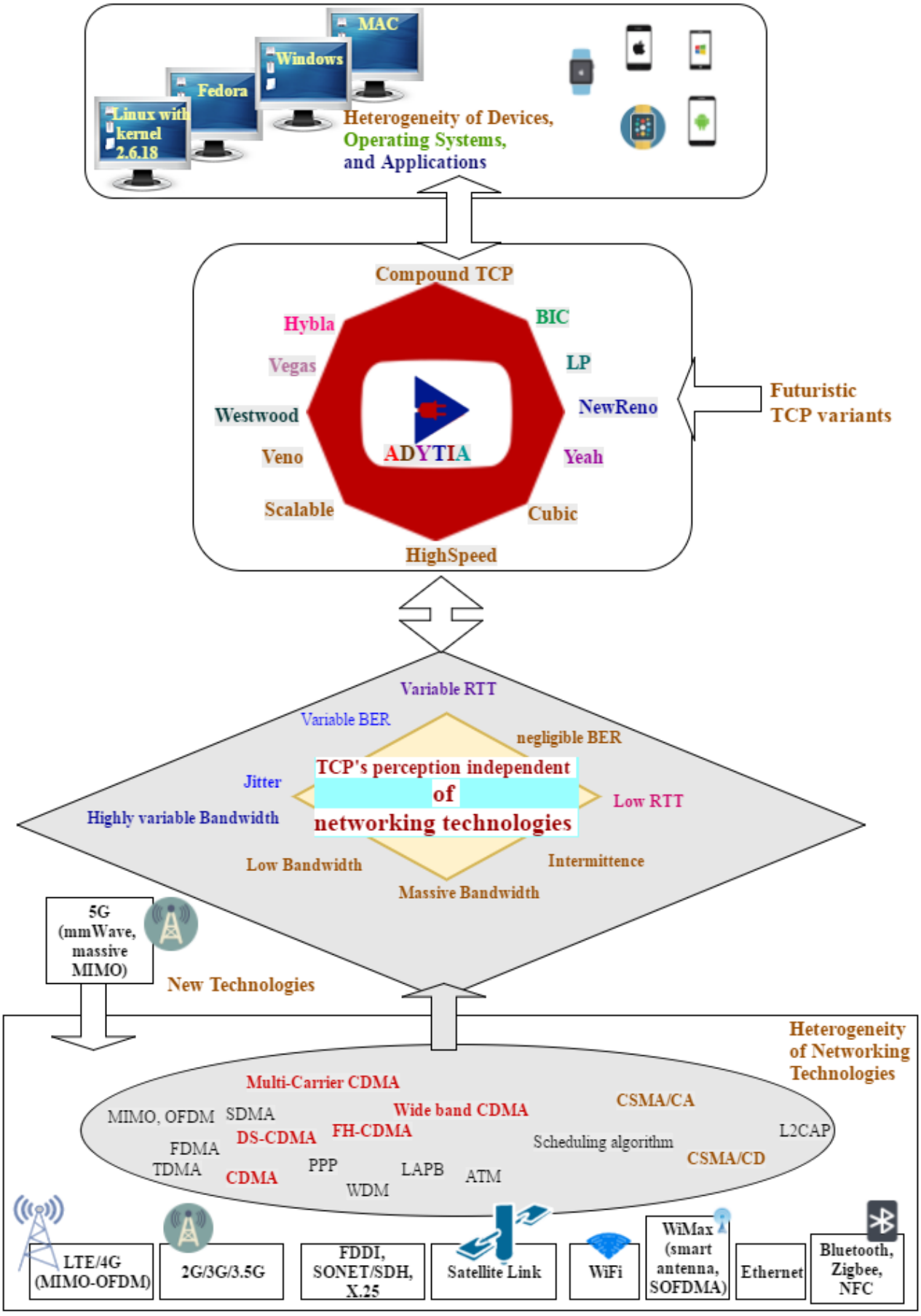


Figure 1.2: Heterogeneity of networking technologies, operating systems, devices, applications, and ADYTIA's approach

1.3 Scope & Objectives

This research work is aimed at proposing the solutions that address the issues as discussed in Section 1.2, carry out the comparative analysis of the proposed solutions with the existing approaches, and analyze the results obtained in a simulation environment, testbed setup, and on live Internet with different network dynamics using open source platform. The major objectives of the proposed research work have been presented as under:

- To design a TCP variant for a link with variable BER and RTT. The throughput should not be affected by variations in BER and RTT.

- To design an architecture that allows the usage of specific TCP variant based on link and application type. Thus, link utilization can be maximized. In heterogeneous networks, each link has different characteristics. For example, TCP NewReno was designed with an assumption that if any loss happens, it must be because of the congestion but, that is not always true for the wireless links. Each application also has different expectations from the transport layer. Applications like FTP always tries to optimize the throughput and has less tolerance for any loss whereas real time interactive video kind of application (e.g., Skype) requires minimum delay without retransmission of the lost frames. The combination of diversified applications and different networking technologies requires a specific protocol to optimize the performance. Time varying link parameters also require the protocol mechanism to be changed at runtime.

- To deploy the architecture without modifying end-user's operating system. It makes the acceptance and deployment very easy. Hence, there is a need to integrate the proposed approach on intermediate nodes (e.g., may be on ISP sides) along with Performance Enhancing Proxies (PEP).

- To allow plugging of newly designed TCP variant in the proposed architecture without any modifications in the architecture. This feature facilitates the use of futuristic TCP variants to enhance the link utilization for upcoming technologies in specific scenarios.

- Implementation in ns-2 and analysis of the results obtained. Confirmation of the benefits of the proposed approach when compared with existing TCP/IP implementations through experiments on the testbed and live Internet.

In order to realize the above objectives, it was essential to use the state of the art approach to design the protocol and mechanism. Hence, a strategy has been used that facilitates the use of the kernel level implementations in the simulations based experiments also. It reduced the code rework and allowed the experimental analysis on simulation mode as well as the testbed with real operating systems.

1.4 Research Contributions

The present section covers highlights of the research contributions made:

- Designed a new TCP variant named Tarang for a link with variable BER and RTT. Tarang has outperformed the existing variants, including Hybla, Westwood, Cubic, and NewReno. In a worst-case scenario with BER of 10^{-6} and RTT of 1500 ms (i.e., a multi-hop satellite link), Tarang has improved the performance by 20-30 times compared to other existing variants. Further, Tarang was able to achieve fairness in heterogeneous networks compared to existing variants. Tarang has been described in Chapter 3.

- **A**daptive and **DY**namic TCP **I**nterface **A**rchitecture (ADYTIA) has been designed to allow the usage of TCP variant based on link and application type. ADYTIA's ability to select best suitable variant results in improvised performance compared to existing mechanism of default and single variant usage at a time. The selection of the specially crafted variant results in 20% to 80% improvement in performance. ADYTIA has been explained in Chapter 4. It also allows the change of variant at run time for the link with time varying characteristics.

- ADYTIA allows the plugging of futuristic TCP variants. Tarang has been plugged into ADYTIA to verify this feature.

- ADYTIA accommodates the changes in networking technologies, application types, and with the use of suitable TCP variant, it allows each flow to have

maximum possible link utilization. Further, ADYTIA allows development of underlying technologies, applications, and TCP variants, independent of the considerations of matching of these three technological paradigms, and performance bottleneck. ADYTIA's approach has been depicted in Figure 1.2.

- ADYTIA has been integrated with Performance Enhancement Proxy (PEP) to address the fairness and deployment issues that are elaborated in Chapter 5. Integration of ADYTIA and PEP eases the deployment without the need for modifying end-user's kernel.

- Tarang and ADYTIA have been tested in the simulation environment. Then, ADYTIA has also been tested on the testbed with kernel level implementation and has also been tested on live Internet.

1.5 Organization of the Thesis

This report has been structured as follows:

Chapter 2 (Literature Survey & Research Challenges): This chapter introduces challenges in heterogeneous networks and categorizes the existing mechanisms for the issues that have been identified in the problem statement. Approaches like adaptive selection, TCP-kentridge (Xiuchao) and configurable transport layer solutions (Wong, Hiltunen, Schlichting, et al.) have been analyzed to clarify the status of the existing works related to the problem statement. It has also looked at the current Internet architecture, heterogeneity of the platforms and network technologies and highlights the importance of the research work. Issues discussed in the problem statement have been highlighted by the mathematical formulation. Existing and supported TCP variants in the open source platform have been compared using simulation results to investigate the problem statement in depth. It has been proved by experiments that the existing variants' (i.e., including Hybla, Westwood, NewReno, and Cubic) throughput was negligible compared to available link capacity, on links with variable-BER and RTT. Mathematical analysis has proved the advantage of split approach compared to end-to-end solutions. Further, testbed experiments for split approach have been performed to complement the mathematical analysis. Towards the end, challenges at transport layer with the current architecture have been discussed which

showed the path to go for the proposed contributions presented in the report.

Chapter 3 (Tarang- A Protocol for the link with variable BER and RTT): This chapter deals with a new protocol named Tarang, designed for a link with variable RTT and BER. The comparison of the existing protocols including Cubic, and Tarang has been carried out using ns-2 and results have been discussed. Tarang has been tested on variety of simulation scenarios with varying network dynamics. Tarang was able to outperform all existing variants including Cubic (i.e., default variant of Linux) and NewReno (i.e., default variant of Microsoft Windows). In a worst-case scenario with BER of 10^{-6} and RTT of 1500 ms (i.e., a multi-hop satellite link), Tarang has improved the performance by 20-30 times compared to other existing variants. Tarang showed the fairness in heterogeneous networks whereas existing variants have exhibited critical fairness issues. Although, Tarang could replace the default variant like Cubic, there could be a variant in future which may perform better than Tarang, in a specific scenario. Thus, there was a need to have TCP variant usage based on link and application type which has been addressed in the next chapter.

Chapter 4 (**A**daptive and **DY**namic **T**CP **I**nterface **A**rchitecture): This chapter introduces Adaptive and Dynamic TCP layer Interface Architecture named "ADYTIA" with the required modifications in the TCP/IP stack implementations. The proposed architecture consists of mainly four components: i) Connection Classification Module (CCM). ii) Application Classification Module (ACM). iii) Information Base. iv) Adaptive Module. The algorithm used by all the modules of ADYTIA has been discussed in this chapter. Next, it has listed out the challenges and implementation strategies for simulations and testbed experiments. Later, the topology and evaluation metrics used to evaluate the performance of ADYTIA have been presented, and an exhaustive simulation results analysis has been presented to highlight the strength of the proposed architecture. ADYTIA allows plugging of futuristic TCP variants. Hence, Tarang has been plugged in ADYTIA without requiring any code rework or recompilation of the kernel. ADYTIA's approach to select the specially crafted variant has resulted in 20% to 80% improvement in performance compared to other existing variants. However, ADYTIA has to be installed on each end-user. Finally, the chapter concludes with implementing feasibility options and has proposed a way ahead for ease the deployment by integrating ADYTIA with PEP.

Chapter 5 (Integrating ADYTIA with PEP): This chapter starts with the concept of Performance Enhancement Proxy (PEP) to make the implementation feasible in the current Internet architecture. The details of the selected platform and implementation strategy has been discussed. Next, the testbed experiment topology with the discussion on router configuration, link configuration, and tools used for the experimental setup has been presented. Finally, the result analysis of ADYTIA with PEP on the testbed and deployment strategy has been presented. Results of existing TCP/IP implementations and ADYTIA on live Internet have been compared to highlight the strength and need of the proposed approach on all platforms. ADYTIA along with PEP on live Internet improved the performance by 162 % compared to Cubic (i.e., default variant of Linux). This improvement is very significant and paves the way for the research work to consider other modern operating systems. Next chapter concludes the research work with scope for work in future.

Chapter 6 (Conclusions and Future Work): This chapter concludes the research work with the highlights of the results achieved. The future directions and final remarks for the path ahead have also been presented.

The Appendix-A to E cover the details of the configuration file of each machine/router of the testbed experiments, tools that have been used for parameter setting and other tools that have been used for traffic generation and analysis and important commands which have been used in testbed experiments.

Towards the end, the thesis has separate sections for *Indexes* and *Works Cited*.

Chapter 2

Literature Survey & Research Challenges

TCP is a traditional transport layer protocol used for communications on the Internet for many decades. TCP has been designed to outperform in terms of link utilization for wired networking technologies. Application layer protocols' performance depends on the TCP, and it has performed better for many standard application layer protocols (e.g., HTTP, FTP). The diversity in today's Internet based accessibility of services, the growth of smart devices, upcoming networking technologies and complex communication scenarios make the job of transport layer protocol in TCP/IP model, a challenge. This chapter elaborates on the different challenges for TCP's performance in heterogeneous networks.

This chapter has been organized as follows: Section 2.1 briefly explains the heterogeneous networks, challenges and clarifies the need of this research work. Section 2.2 discusses existing TCP variants with merits and issues. Section 2.3 presents the comparative analysis, simulation results and performance issues of the existing TCP variants. Section 2.4 identifies issues in heterogeneous networks mathematically and strongly advocates the need for some dynamic mechanism. Section 2.5 presents the testbed results of PEP to highlight the advantages of choosing this approach. Finally, Section 2.6 summarizes the transport layer challenges in heterogeneous networks.

2.1 Heterogeneous Networks

The existing Internet architecture comprises of different kinds of heterogeneous networks that are perceived by numerous technologies, which in turn results in many varieties of the path and link features (e.g., delay, bandwidth, etc.) (Barakat, Altman, and Dabbous). These varieties of links/underlying technologies can be like ethernet, optical-fiber, Wi-Fi, WiMAX, 2G, 3G, 4G, LTE-Advanced, 5G, and satellite link. In all these networking technologies, link capacity can be either lack in low-speed bandwidth links or there may be over provisioning of bandwidth in high-speed links and significant variations in latency. Latency varies for local interconnects, 10 to 200 ms to specific wireless and satellite links for 100 to 1000 ms. Thus, consequently, both, the available bandwidth and end-to-end delay on the Internet may vary immensely, and it is highly probable that the range of parameters increases further in future.

Furthermore, both, the available bandwidth for a specific flow and end-to-end delay have no stability. On the IP layer, battling traffic, traffic management in routers, and dynamic routing can lead to abrupt changes in trades of the end-to-end path. Link layer mechanisms can bring more dynamics which changes to new links due to portability, topology alteration, link-layer error correction and vigorous bandwidth provisioning scheme (Pirovano and Garcia), (Atxutegi et al.), (Zhang et al.). Thus, path characteristics are subjected to massive alteration in short time span.

A large variety of hosts with different resources (e.g., Memory, CPU, Power, etc.), like desktop machines, laptops, smart devices, are attached to the existing Internet (Xiuchao). All these hosts have different operating systems (e.g., Windows, Linux, Android, etc.) and varieties of applications (e.g., HTTP, FTP, Multimedia streaming, Gaming, etc.) run on these hosts. Heterogeneity of the present Internet has been shown in Figure 2.1. Such a heterogeneous network yields many challenges for the transport layer protocols as discussed in the following subsection.

2.1.1 Challenges

Following key challenges were identified in the heterogeneous networks:

- **Fixed TCP Variant:** A host needs to communicate with another host on the Internet (e.g., a client-server model, peer-to-peer communication) through

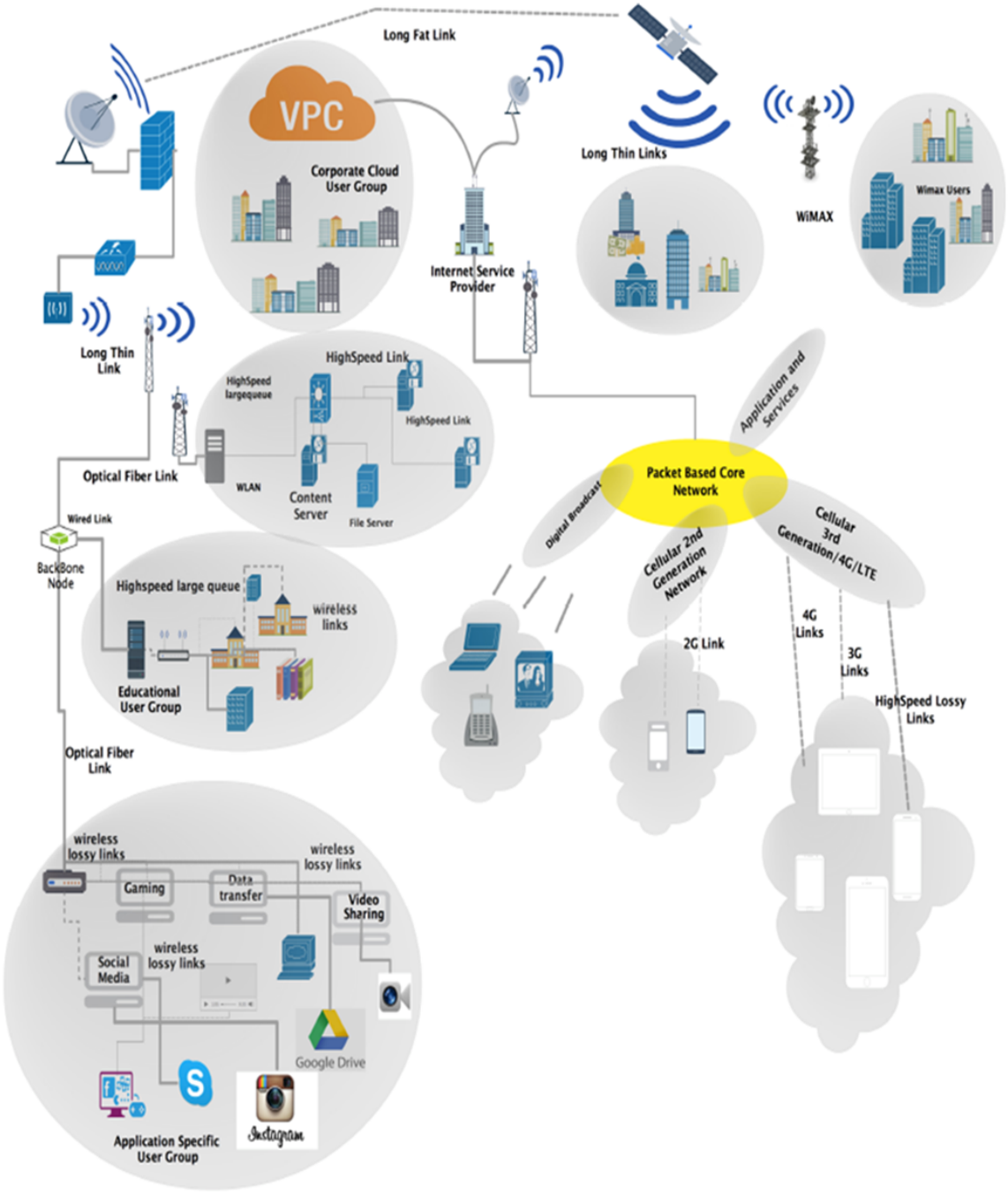


Figure 2.1: Heterogeneous network (A typical scenario)

different kinds of networks depending on the location of the peer host machine. It means different connections using a single TCP implementation of the host machine (depending on operating system) and communication takes place via different types of links. Each type of link has the varying bandwidth, RTT, and BER which affects the performance of TCP. As explained in (Caini and Firrincieli Jitendra et al.), TCP's performance has been degraded with higher

values of RTT and variations in RTT affect the throughput of TCP variant significantly. Hence, fixed TCP variant of the host machine cannot utilize the available link capacity for time varying link characteristics and different types of links.

- **Additive Increase and Multiplicative Decrease (AIMD):** AIMD mechanism of standard TCP (i.e., NewReno) cannot exploit the high bandwidth of the present networking technologies. Link with high RTT and/or high bandwidth creates the high BDP pipe which cannot be filled fully by standard TCP (Sally). Thus, there is a need to have different kinds of congestion control algorithms for varying link characteristics in heterogeneous networks.

- **TCP Congestion Assumption:** Originally, TCP was designed for the wired link in which link losses were negligible, and link capacity was low to moderate. Hence, if any losses were there, the assumption was that loss must be due to congestion which in turn triggers the slowdown of the sender machine's data sending rate. In heterogeneous networks, there could be the presence of many wireless links; hence TCP's design principle cannot be applicable. It means losses could be due to the link characteristics which necessitate the mechanism to differentiate the reason of a loss (Saverio et al.).

- **Congestion Signal:** Switching devices on the Internet (e.g., router) may have different types of queue management strategies with varying queue sizes. Hence, each switching device keeps sending the different types of signal to indicate congestion to the sender machine(Papadimitriou et al.).

- **Emerging Applications:** TCP is responsible for providing services to application layer protocols, which may have different expectations. For example, FTP tries to maximize the throughput whereas the goal of TELNET is to minimize the response time. In heterogeneous networks like the Internet, there are numerous applications with different expectations from the transport layer protocols (Chen, Farley, and Ye). Seamless connectivity and increasing bandwidth of the Internet result in varieties of applications (e.g., Facebook, Skype, online gaming, etc.). These emerging applications lead to a wide variety of

communication scenarios, which requires a particular transport layer protocol to maximize the utilization of network resources and to ensure a better user experience (Yin et al.).

- **Upcoming Communication Technologies:** The widespread use of wireless technologies (e.g., Wi-Fi, WiMAX, 3G, 4G) lead to many challenges compared to traditional wired networks. For example, transport layer protocol has to differentiate the cause of the packet loss like congestion, link error or handover (Tian, Xu, and Ansari Peng, Wu, and Leung). Thus, there is a need to measure the network link characteristics continuously to observe such reasons.

- **New Platforms:** In the present scenario, each electronic gadget designed to have the capability to get connected with the Internet(Liu and Lee). Each newly designed gadget has different capabilities in terms of processing power, memory, size, power, and operating system with a different kind of TCP variant. Hence, it is challenging for the specific TCP variant of such a variety of devices and platforms to give a better user experience.

2.2 Existing TCP Variants

Originally, TCP was designed for the wired link (i.e., low delay, moderate bandwidth and negligible BER)(Tian, Xu, and Ansari). Congestion control and flow control have been carried out by an algorithm on the sender machine using two variables, namely congestion window (cwnd) and slow-start threshold (ssthresh). Based on the mechanism of an algorithm, there exists a TCP variant. The utilization of available link capacity depends on the rate of increment of cwnd. The objective of each TCP variant was to probe the available link capacity for the ongoing communication flow and to adjust the value of cwnd to exploit link capacity fully (i.e., fair sharing among competitive flow) without causing congestion in the network. This section briefly covers the existing approaches designed for different kinds of networking technologies and platforms. TCP has facilitated the development of many variants. These variants can be broadly categorized into three different types, presented below (Wang et al.):

- **End-to-End Solutions:**

  This refers to the solutions that require only the changes in congestion control

algorithm of the sender machine. In order to use any approach of this category, it is mandatory to modify the sender machine's operating system source code.

- **End to End Solutions with the support of Network Infrastructure:**
  The variants belonging to this category of solution require changes on the sender side and or receiver side, as well as, it requires support from all intermediate networking components to implement the solution. Because of these requirements, it is difficult to deploy such kind of solution in the present Internet architecture.

- **Non End-to-End Solution:**
  This category of the TCP variants does not require any changes to be made on end host machine (i.e., sender, receiver). Hence, deployment of such kind of solution is very easy and acceptable at large. This kind of solution has been called Performance Enhancement Proxy, generally known as PEP. PEP can be configured on the gateway (where most of the connections are terminated, i.e., on ISP side). Initially, these categories of the TCP variants have been challenged by security measures at the transport layer, but with use of security measures at the application layer, it has been considered as a good candidate to resolve the challenges in heterogeneous networks.

2.2.1 TCP Taheo, Reno, NewReno

TCP Taheo was the first variant and can detect the losses using timeout mechanism. The losses can be recovered by retransmission of the segments. Reno was designed to overcome the issue of waiting for a timer to go off to detect the loss. Reno can detect the losses by three duplicate acknowledgments and retransmit the lost segment without waiting for the timer to go off. Reno could not increase the cwnd in case of multiple losses in a window (Fall and Floyd). NewReno was designed to overcome the issue of Reno and recover the multiple losses without reducing cwnd multiple times (Henderson et al.). Same as Reno, NewReno enters into the fast-retransmission state when the sender receives multiple duplicate acknowledgments. Modification in NewReno was that NewReno does not leave fast recovery state until all the packets which were outstanding at the time it entered into the fast recovery has been acknowledged.

Drawbacks: NewReno organizes from the certainty that it takes one RTT to detect each packet lost in the network. After receiving an acknowledgment for the first retransmitted segment, a sender can conclude which other segments were lost in the network.

2.2.2 Selective Acknowledgment (SACK)

The main idea of the SACK (Floyd et al.) scheme was to provide the sender with more detailed information about the state of a receiver. SACK mechanism allows a receiver to send SACK information as a part of the TCP header options, which is complementary to the existing TCP acknowledgment mechanism. The option fields used to indicate starting and ending sequence of non-contiguous blocks of arrived data at the receiver. The receiver has been shown in Figure 2.2 has just received a packet number eight and the information sent in SACK will be: first block 8761 to 10221, second block 5841 to 7301, and third block 1461 to 4381.

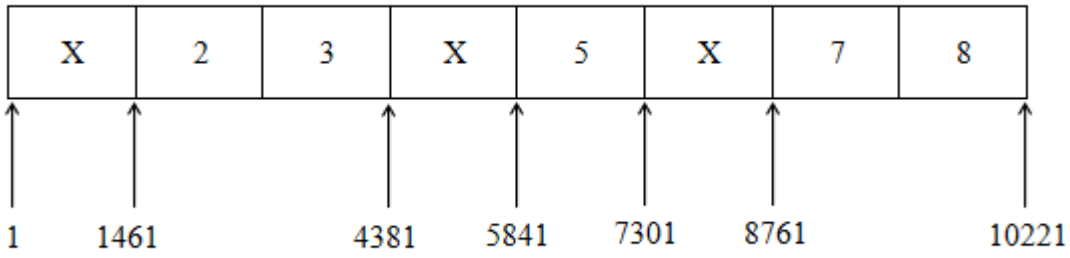


Figure 2.2: Receiver's window in case of multiple losses

The receiver sends SACK information in the TCP header options when duplicate acknowledgments have been transmitted in response to the arrival of out-of-order segments. SACK is a supportive mechanism and follows the other algorithms like TCP NewReno. SACK can be used with other TCP variants also.
Drawbacks: SACK was not able to distinguish between losses due to congestion or link errors.

2.2.3 TCP HighSpeed

As mentioned in (Sally), TCP HighSpeed has been designed for high bandwidth links that function with the large value of cwnd, in the initial stage. Unlike NewReno, increase and decrease factor were not constant in TCP HighSpeed. As a result, the available bandwidth may be exploited. TCP HighSpeed used modified response function. Three parameters namely Low_Window, High_Window, and High_P, were used to define a response function.

Drawbacks: In TCP HighSpeed the achievable congestion window could be utterly large so that sender can send the large burst of packets on receiving one acknowledgment. Thus, if there was a congestion or packet reordering on the reverse path and then sender received an acknowledgment that acknowledging thousands of fresh packets, a burst could also result.

2.2.4 Scalable TCP

Scalable TCP (Kelly) was designed to be additionally utilized and act indistinguishable to TCP NewReno when small sender window size was enough to use the low-speed links. Scalable TCP was designed for high-speed links. Scalable TCP used Multiplicative Increase and Multiplicative Decrease (MIMD) approach for the congestion control. Scalable TCP could perform well under high BDP links and can utilize network link capacity. If Scalable TCP was competing with regular TCP (e.g., NewReno), then Scalable TCP utilized a major fraction of the available bandwidth. Therefore, scalable TCP was unfair with standard TCP flows. Scalable TCP updates the congestion window per acknowledgment in an RTT as per the following rule:

Increase: cwnd = cwnd + 0.01(cwnd)

On detecting the loss it decreases the congestion window as follows:

Decrease: cwnd = cwnd - 0.125(cwnd)

Suppose a TCP connection has 250 ms RTT, 1500 bytes of segment size, 1 Gbps of the link, and if congestion occurs, then scalable TCP recovers congestion window to the original (i.e., size of cwnd when congestion occur) within 4 seconds, while TCP NewReno takes approximately 43 minutes.

Drawbacks: Scalable TCP has fairness issue with NewReno in heterogeneous networks.

2.2.5 TCP Westwood and Westwood+

TCP Westwood was designed for the wireless channel (Saverio et al.). It was modified version of TCP Reno. TCP Westwood introduced a modification of the Fast Recovery algorithm called Faster Recovery. The key idea was to continuously measure bandwidth used by the connection at the sender via monitoring arrival rate of acknowledgments. This measured bandwidth has been used to set the value of cwnd and ssthresh after the congestion episode. TCP Westwood used the following algorithms to set the values of cwnd and ssthresh after detection of congestion.

Algorithm 1 When 3 duplicate acknowledgments (DUPACKs) received

1: **if** 3 DUPACKs are received **then**
2: $ssthresh = (BWE * RTT_{min})/seg\_size$
3: **if** $cwin > ssthresh$ **then**
4: cwin = ssthresh
5: **end if**
6: **end if**
7: Enter into Congestion Avoidance (CA) phase

Algorithm 2 After TCP timer expires

1: **if** timer expires **then**
2: ssthresh = $(BWE * RTT_{min})/seg\_size$
3: **if** $ssthresh < 2$ **then**
4: ssthresh=2
5: **end if**
6: cwin=1
7: **end if**
8: Enter into Slow Start (SS) phase

In above algorithms, RTT_{min} was the smallest RTT sample observed over the duration of the connection, seg_size was the length of a TCP segment in bits, and BWE was the estimation of utilized bandwidth. TCP Westwood+ (Mascolo et al.) used a modified algorithm to estimate the bandwidth. Linux kernel has implementa-

tions of Westwood+.

Drawbacks: TCP Westwood performed poorly in the presence of high RTT.

2.2.6 TCP Hybla

TCP Hybla was designed for the high RTT link like satellite (Caini and Firrincieli). TCP Hybla equalizes the performance of connections with different RTTs. The goal of TCP Hybla was to reduce the effect of longer RTT on cwnd increase rate. It calculates the normalized round trip time ρ as:

$$\rho = RTT/RTT0 \tag{2.1}$$

RTT0 was round trip time of reference connection (i.e. low RTT connection) to which TCP Hybla tries to equalize the data sending rate. TCP Hybla's cwnd update rules are:

$$\begin{aligned} W_{i+1}^H &= W_i^H + 2^\rho - 1, SS \\ &= W_i^H + \frac{\rho^2}{W_i^H}, CA \end{aligned} \tag{2.2}$$

In above equation, W_i^H was cwnd when i^{th} acknowledgment has been received. When TCP Hybla operated in SS phase and received an acknowledgment then cwnd has been increased by $2^\rho - 1$. In CA phase, it increases cwnd by ρ^2/W_i^H after receiving acknowledgments for all the transmitted segments.

Drawbacks: TCP Hybla performed poorly for the link with relatively high loss rate in presence of high RTT.

2.2.7 TCP Binary Increase Congestion Control

TCP Binary Increase Congestion Control (BIC) was designed for high BDP links (Xu, Harfoush, and Rhee). BIC used a binary search algorithm to find the optimum value of cwnd. Previous dynamics of cwnd have been used to decide the target value of cwnd and this algorithm made the cwnd growth rate faster when it was far from the target value. In the case of congestion (or loss event), BIC could reduce cwnd with different backoff value, whereas in NewReno that backoff value has been fixed to 0.5. BIC was the default TCP variant in Linux kernel 2.6.18.

Drawbacks: TCP BIC has fairness issue with the competing flows.

2.2.8 TCP Cubic

TCP Cubic (Ha, Rhee, and Xu) was designed for high-speed network links and it was an improvement on TCP BIC. TCP Cubic was designed to be fair and friendly compared to other competing TCP flows. In Cubic the increase rate of cwnd has been governed by cubic function in terms of the elapsed time since the last loss event happened. The idea to introduce Cubic was to make the growth rate of cwnd independent of RTT. Cubic used the following equation to update the value of cwnd:

$$W_{CUBIC} = C(t - K)^3 + W_{max} \tag{2.3}$$

In above equation, C is a scaling factor, t is the elapsed time since the last instance of window reduction and W_{max} is cwnd before the last reduction. In above equation, $K = \sqrt[3]{W_{max}\frac{\beta}{C}}$ and β is constant reduction factor at the time of congestion event.
Drawbacks: TCP Cubic's performance has been degraded in presence of both, high RTT and high loss rate.

2.2.9 TCP Illinois

Efficiency, stability, and fairness were objectives of TCP-Illinois on high-speed networks (Liu, Başar, and Srikant). TCP-Illinois was loss and delay based congestion control algorithm. Hence, it used lost segment as a signal to determine the direction of cwnd change (i.e. cwnd has been increased or decreased) and queuing delay as a signal to update the value of cwnd (i.e. the value by which cwnd has been incremented or decremented).
Drawbacks: For higher values of BDP, its performance has been degraded and network responsiveness could be sluggish.

2.2.10 TCP Vegas

TCP Vegas was a delay based congestion control algorithm. TCP Vegas used expected throughput and actual throughput with other parameters derived from the set of experiments to decide growth and depletion of cwnd (Brakmo and Peterson). Vegas has been considered as a proactive approach for congestion control mechanism. TCP Vegas depends on the correct estimation of RTT.
Drawbacks: Fairness may be an issue with competing connections.

2.2.11 TCP Veno

TCP Veno observed the network congestion level by RTT variations and used this information to conclude the reason for losses (e.g congestion or random bit errors)(Fu and Liew). Veno adjusts ssthresh according to the estimation of network congestion level rather than a fixed drop factor of 0.5. Moreover, it used the modified increase algorithm of NewReno so that the connection can stay longer in an operating region in which network bandwidth was fully utilized.
Drawbacks: TCP Veno could not perform well on links with high BER, as it reduces cwnd on each loss by some value.

2.2.12 TCP YeAH

TCP YeAH was designed for high BDP links and operated using two different modes, namely, fast and slow as described in (Baiocchi, Castellani, and Vacirca). In fast mode, YeAH inflate cwnd according to an aggressive rule, while in slow mode it acts like NewReno. Estimation of number of packets in bottleneck queue has been used to decide the mode in which TCP YeAH operates.
Drawbacks: TCP YeAH could not perform well in high BER environments.

2.2.13 TCP Low Priority

TCP Low Priority (LP) (Kuzmanovic and Knightly) was sender side modification that achieves two-class service classification without any support from the network infrastructure. TCP NewReno has been a dominant protocol for best effort traffic, TCP LP was designed to realize a low priority service compared to the existing best effort service. TCP LP's main objective was to utilize the bandwidth left unused by competing TCP flows in a transparent fashion.
Drawbacks: TCP LP could not perform well in high BER environments.

2.2.14 TCP Adaptive-Selection Concept

The increasing level of heterogeneity of existing networks poses new challenges to TCP NewReno. Different TCP flows with very different characteristics like bandwidth, RTT and loss rate will have to compete in a fair way in the same network. This heterogeneity makes a choice difficult among many TCP variants, as each variant

focuses on the single impairment that results in poor link utilization for different network environments. TCP Adaptive-Selection concept (Caini et al.) has given a very simple algorithm based on bandwidth and RTT and explained an advantage of having different transport layer flavor for different kinds of links.
Drawbacks: It used very simple and rigid algorithm to select a variant. To change the variant for a specific link, it was compulsory to modify the algorithm itself.

2.2.15 TCP Kentridge

TCP Kentridge was a framework designed for the heterogeneous networks with the capability of intelligently changing TCP flavor (Xiuchao). TCP Kentridge consists of four components, Knowledge Base, Intelligent Agent, DC-TCP and Network Pipe Classification for its implementation. This framework has been implemented in Free BSD, and transport layer flavors need to be changed manually for testing. Hence, the framework has been partially implemented and not deployed on real networks.
Drawbacks: It was just an approach without details of the modules and dynamism.

2.2.16 Configurable and Extensible Transport Protocol (CTP)

CTP has been composed of micro protocols according to the need of an application (Wong, Hiltunen, Schlichting, et al.). Each micro protocol set represents the basic function of TCP and UDP protocols. This kind of protocol could be used in an application where the reliability of TCP has been needed without retransmission, for example, multimedia transfer application. CTP has been developed using Cactus, a system for developing a highly configurable protocol for networked and distributed system. In this system, each micro protocol was the collection of event and implements one function or attribute of transport layer protocol. Various attributes of transport layer protocol were reliability, ordering, performance and timeliness. The collection of micro protocols creates a composite protocol. At runtime, the composite protocol has been used to process the packet.
Drawbacks: Implementation and deployment were the biggest challenges.

2.2.17 GridFTP

GridFTP was a protocol, developed within the context of the Globus Toolkit, that supports the efficient transfer of large amounts of data on Grids (Allcock et al.). The

use of security services of Globus to assure the authentication of grid applications was a major contribution of GridFTP protocol. Other peculiar features of GridFTP were: manual setting of the TCP buffer size, use of multiple parallel streams, third-party transfer option, and partial file transfer option.
Drawbacks: It was required to start multiple streams that put overhead on the sender machine of maintaining multiple sockets.

2.2.18 GridTCP

As mentioned in (Bhatt et al.), GridTCP was based on the concept of GridFTP. GridTCP creates multiple TCP streams for desired connection to multiply the available window size. Thus, GridTCP may be viewed as a group of TCP streams acting as a single TCP connection. In GridTCP, whenever an error occurs during transmission, the only TCP connection on which an error occurred has been affected, and only the window size of affected connection has been halved. Splitting a single TCP connection into multiple TCP connections provide larger window size and hence, minimizes the performance degradation due to errors and delay.
Drawbacks: An end-user need to specify the number of connections to be used and also requires modifications on sender and receiver both.

2.2.19 Sync-TCP

Sync-TCP (Wu et al.) was a delay-based congestion control algorithm designed for high-speed networks. In Sync-TCP, competing flows could detect the same congestion signal through queue delay. In combination with synchronized congestion signal, Sync-TCP used a queue-delay-based congestion window decrease rule and a RTT-independent congestion window increase rule. These rules were designed to drive the network to operate around knee and to distribute the residual bandwidth fairly even when the number of competing flows varied and their RTTs differ significantly.
Drawbacks: It performed poorly for links with high BER.

2.2.20 Performance Enhancement Proxy (PEP)

PEPs were network agent designed for improving TCP performance over the satellite link. They could be operated on any protocol layer, but basically, they improve performance by making optimizations at the transport and application layer as men-

tioned in (Carlo.Caini, Firrincieli, and Lacamera). The advantage of PEP was that they were operated as end system without changing TCP configuration. PEP splits connection into two parts and breaks end-to-end semantics. Breaking end-to-end semantics means that the reliability of transmissions has been achieved on a hop-by-hop basis as the sender node was unable to infer data delivery information from its actual destination. PEP removed the problem of long delay and high BER from the satellite link. PEP could use the proprietary protocols on the link with peculiar characteristics.

2.3 Comparative Analysis

This section presents the comparison of the most relevant approaches, in brief, related to the research work presented in the thesis. It also covers simulation based experiments and comparison of existing TCP variants.

2.3.1 Related Approaches

In Section 2.1, major challenges in heterogeneous networks were discussed, to realize the importance of dynamic selection of TCP variants in real scenarios. A comparative analysis of three TCP variants, namely, TCP-Adaptive-Selection-Concept, TCP-Kentridge, and CTP have been presented in Table 2.1. These three approaches are highly related to the domain of the research works presented in this thesis.

2.3.2 Experimental Study of TCP Variants

TCP variants discussed in Section 2.2 have been evaluated and compared by means of simulation based experiments using ns-2 (Issariyakul and Hossain). The objective of simulation based experiment was to understand the effect of link characteristics on different TCP variants. Simulation topology used for the TCP variants' performance analysis has been shown in Figure 2.3. Simulation topology has been selected based on the existing literature work (Floyd Ha et al.) and it has been used by several researchers to do the performance analysis.

Table 2.1: Comparison of related approaches

Existing Approach	**Mechanism Used**	**Research Gaps**
TCP Adaptive-Selection Concept	-Based on the estimated bandwidth and RTT, it triggers the usage of the fixed TCP variant namely Reno, Hybla and HighSpeed. -Objective of this approach is just to highlight the primary mechanism to change the variant. -It allows the usage of multiple TCP variants for each connection.	-To add a new TCP variant, rewriting of the code and recompilation of the kernel is must. -Algorithm used for the selection of the TCP variant is very simple and rigid. -Results analysis is at primary stage and deployment issues are not considered.
Kentridge	-It focuses on the usage of multiple TCP variants for each connection. -Partially implemented in Free BSD. Require to change the TCP variant manually for the testing.	-It does not allow the plugging of new variants. -Selection of the variant is manual without any intelligence.
CTP	-It uses the micro protocol concept for each functionalities of the transport layer protocol. -As per the need of the application, it organizes the set of micro protocols to create a customized transport layer variant. It is developed using Cactus.	-It does not consider the link characteristics. -It does not allow the plugging of new variants. Deployment of such a solution is a big challenge.

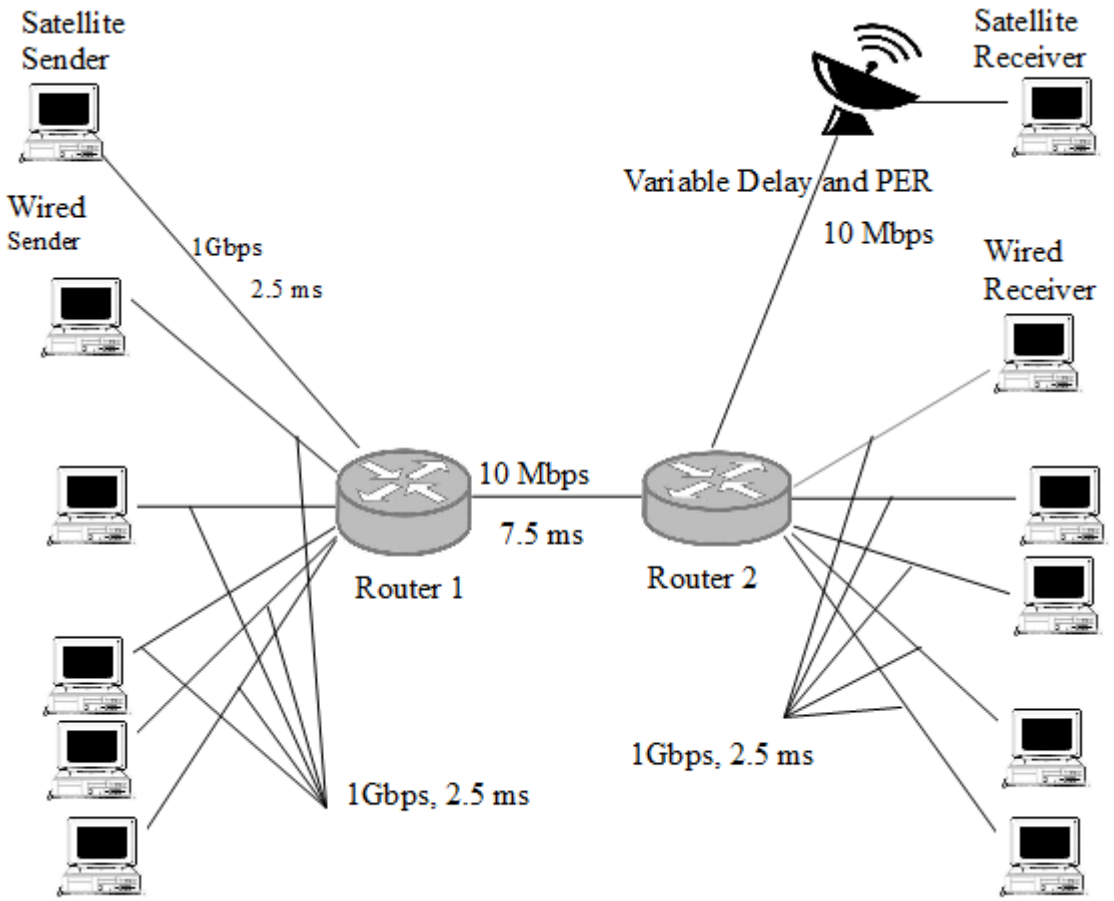


Figure 2.3: Simulation topology for TCP variants' performance comparison

TCP variants' performance for the link with RTT of 600 ms and Packet Error Rate (PER) of 0% has been shown in Figure 2.4. All variants except Hybla, Cubic, Scalable, and Yeah need a long time to probe and utilize the available bandwidth. The reason for such a start-up behavior was that the long RTT of the connection (i.e. 600 ms) slows down cwnd dynamics. TCP Hybla has been designed to cope with long RTTs and it shows the fastest acceleration and the highest throughput. A good start-up performance has also been provided by Cubic due to its nonlinear cwnd growth function. Other variants, like Scalable, and YeAH gave good link utilization when considering long transfer time (more than 60 seconds). Scalable has achieved throughput very close to the best case. However, both Scalable and YeAH performed poorly compared to Hybla in the presence of long RTT and for short transfer lengths, still the throughput of both these variants was better than the other variants.

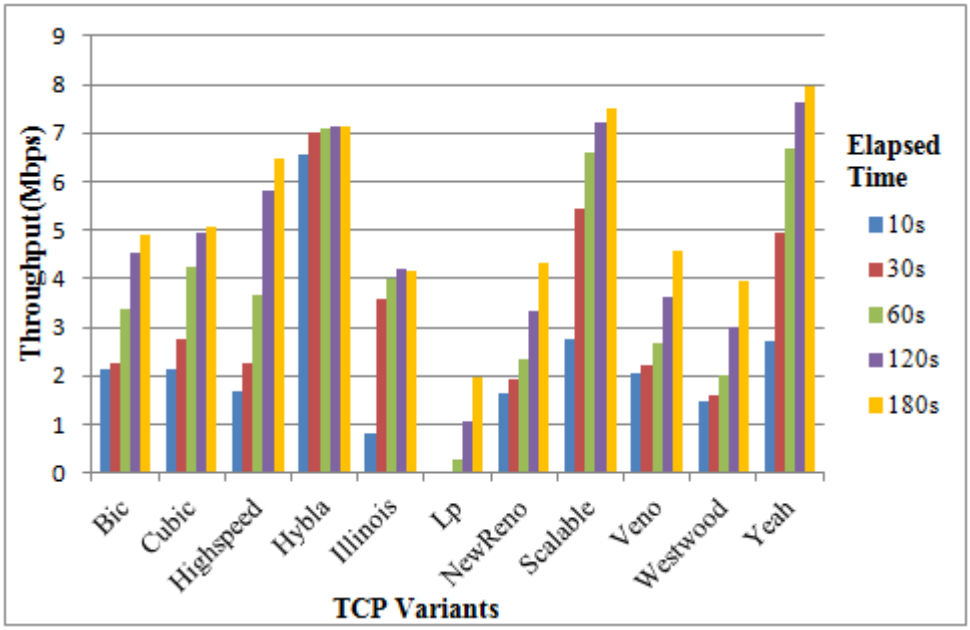


Figure 2.4: Throughput of GEO satellite link without congestion

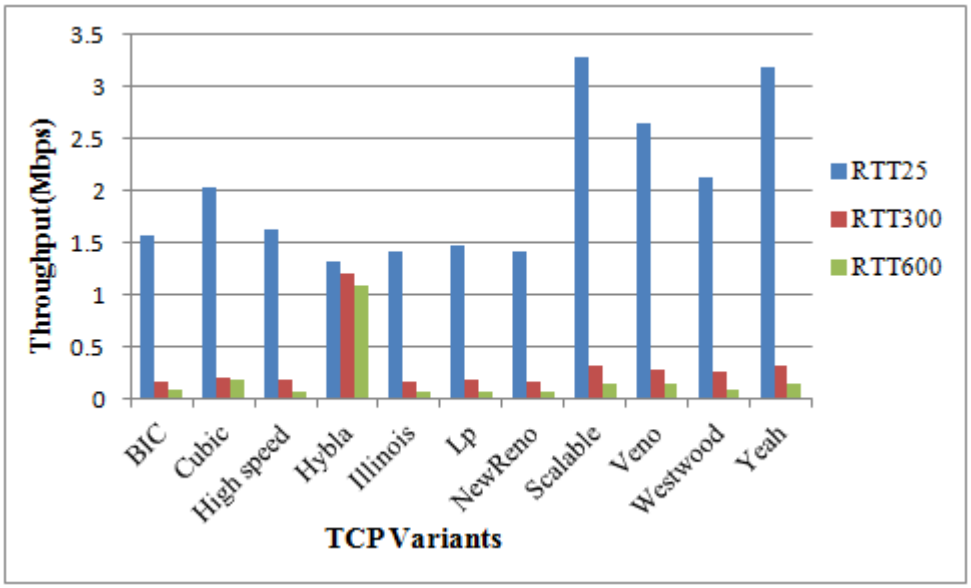


Figure 2.5: Throughput with varying RTT in presence of background traffic

The throughput on the satellite link (for varying RTT) in the presence of 5 background connections (RTT=25 ms) has been shown in Figure 2.5 and simulation has been carried out for 180 seconds. The throughput of satellite link with RTT of 25 ms represents a significant case of equal RTT for all connections. At RTT of 25 ms, all variants exhibited high throughput (i.e., near to fair share of 1.66 Mbps) except Scalable and Yeah. Scalable and Yeah operated above 3 Mbps due to its aggressive

design and showed the fairness issue. The throughput of all variants except Hybla reduced significantly (i.e., under 0.3 Mbps) for satellite link with RTT of 300 ms (i.e., connection with a GEO satellite downlink and a wired up-link). TCP Hybla maintained throughput close to the maximum fair share (i.e., 1.66 Mbps). The same conclusion holds true for the throughput with RTT of 600 ms (i.e., bidirectional GEO link), with a further degradation to under 0.2 Mbps for all TCP variants except Hybla. TCP Hybla achieved link utilization of about 1 Mbps. These results highlighted the issue of RTT unfairness in the absence of specific countermeasures. Hybla has been designed to overcome the issue of high RTT and other variants have no mechanisms to deal with high RTT link. Thus, all the variants except Hybla have performance issues on the link with high RTT.

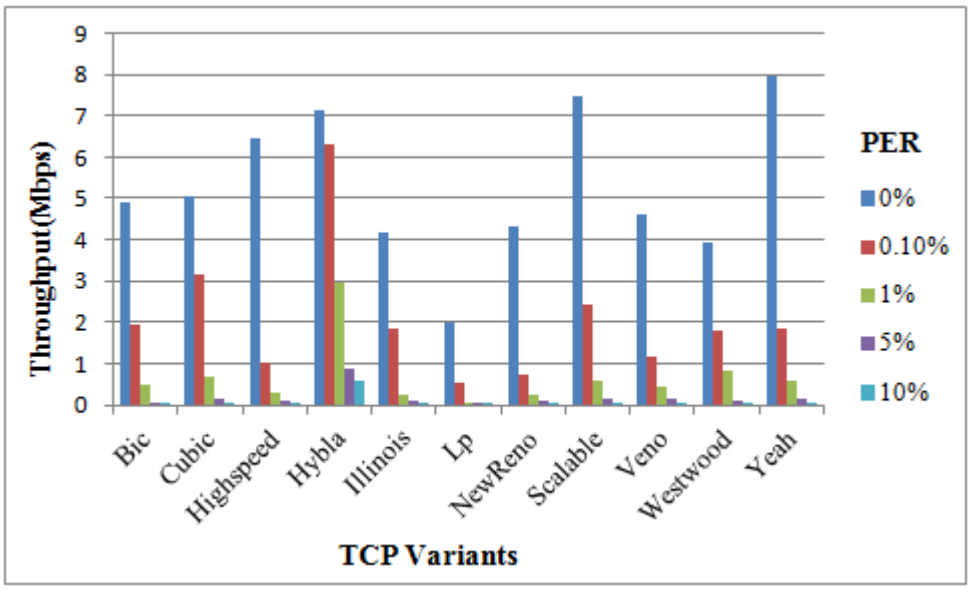


Figure 2.6: Throughput with varying PER without background traffic

The throughput on the satellite link (i.e., a link with RTT= 600 ms) without background traffic for different values of PER has been shown in Figure 2.6. PER value of 0.1% significantly affected the throughput of all variants except Cubic and Hybla. Westwood has been designed to perform well in the presence of high PER, but it could not give good performance in the presence of both, high RTT and high PER. Variants like NewReno, HighSpeed, LP were able to exploit 10% of the bottleneck bandwidth. Throughput has been reduced significantly for a very high PER value (i.e., 10% PER) for all TCP variants. The reason for such degradation in throughput

was the slow reopening of cwnd caused by the very long RTT, after loss recovery phases. Performance of Hybla and Cubic have also been degraded for PER values of 5% and 10%.

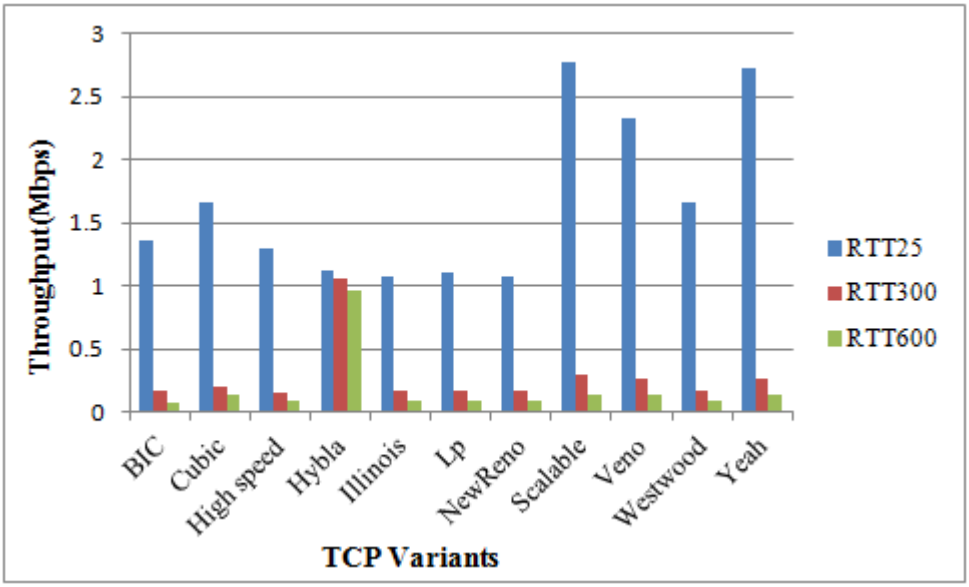


Figure 2.7: Throughput with varying RTT, PER 1% with background traffic

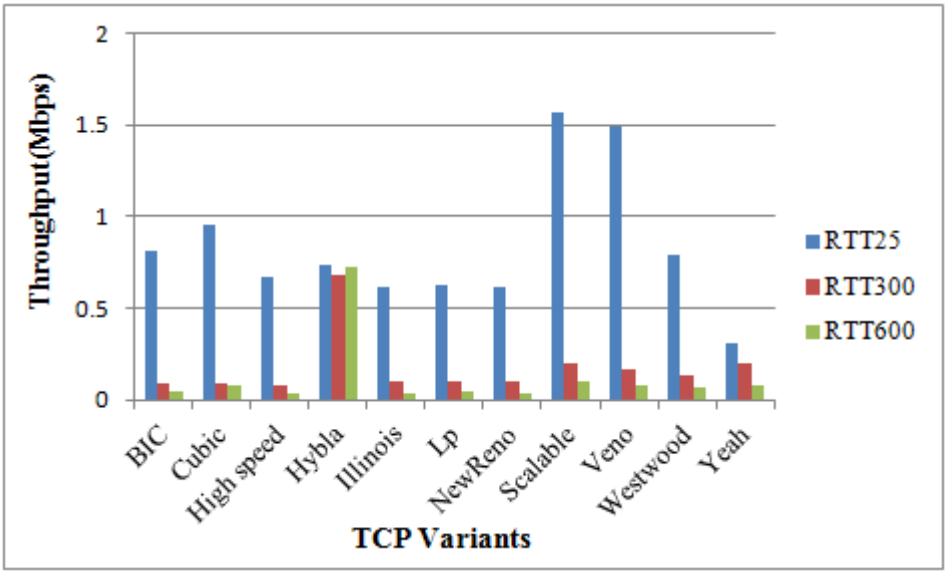


Figure 2.8: Throughput with varying RTT, PER 5% with background traffic

A satellite link with PER of 1% and different values of RTT has been configured, and correspond throughput has been shown in Figure 2.7. Five background connections were running in parallel with RTT of 25 ms, and NewReno was used as a TCP variant. The results were close to those achieved in the presence of background traffic

only, with a reduction due to the presence of link losses limited to about 15% for most of the variants. Hence, it can be concluded that in the environment considered, performance was essentially dominated by RTT unfairness problem whenever wired cross traffic was present.

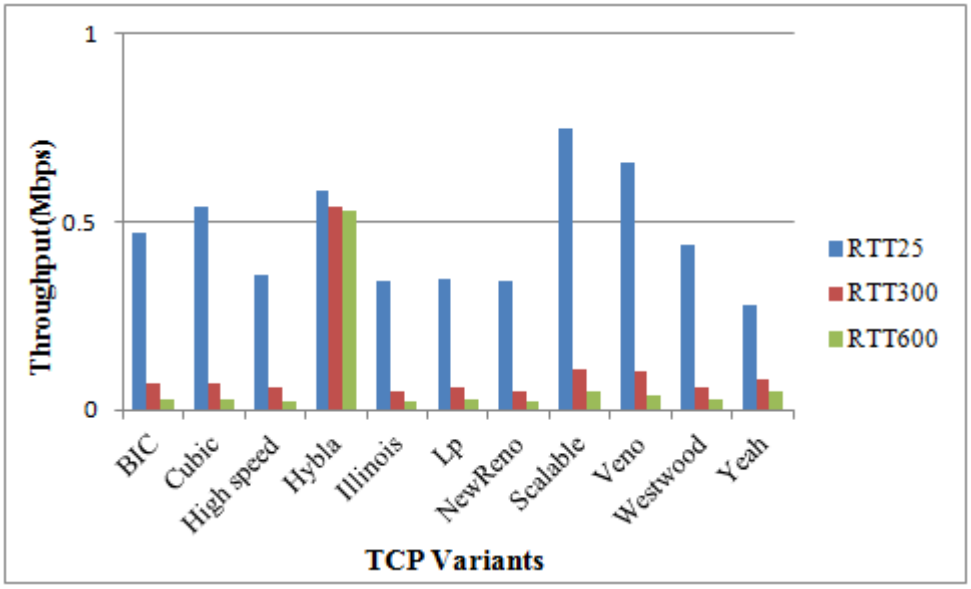


Figure 2.9: Throughput with varying RTT, PER 10% with background traffic

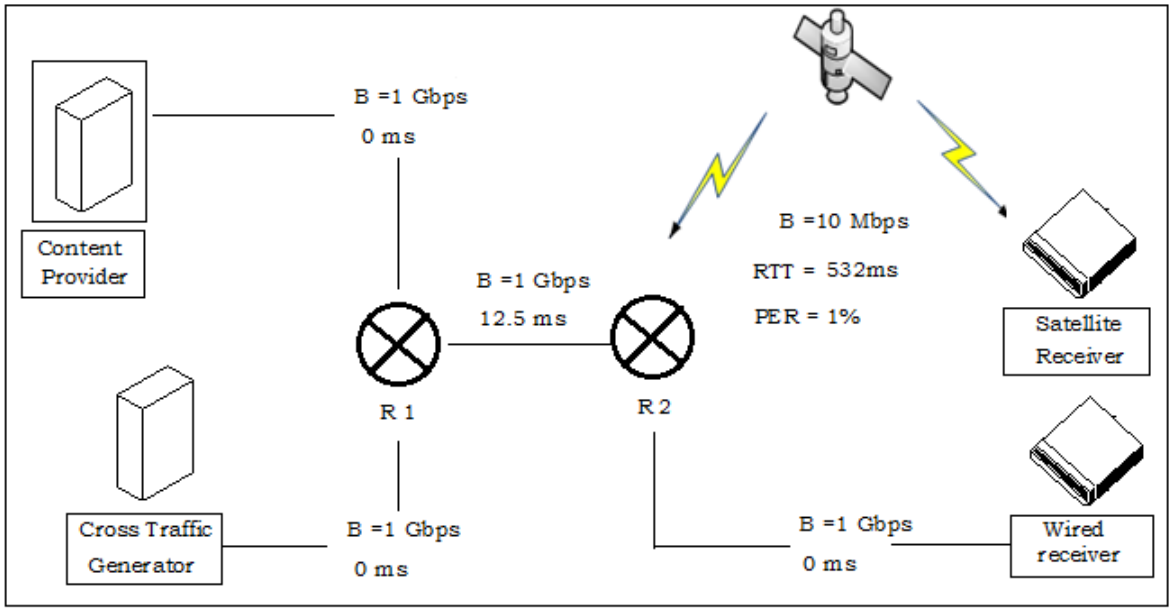


Figure 2.10: Adaptive concept simulation topology (Caini et al.)

Furthermore, a satellite link with PER of 5% with different values of RTT has been configured, and throughput has been shown in Figure 2.8. In another experi-

ment, a satellite link with PER of 10% with different values of RTT has been configured, and throughput has been shown in Figure 2.9. In these experiments, five background connections were running in parallel with RTT of 25 ms, and NewReno was used as a TCP variant. The results demonstrated that in the presence of high RTT and high PER and with background traffic, all the variants were not able to utilize the available link capacity. Hybla could utilize around 0.5 Mbps which was very less compared to fair share fraction of 1.6 Mbps. Variants like Scalable and Veno could operate around 1.5 Mbps for 5% PER and 25 ms RTT, but as RTT gets increased, the throughput of these variants also gets degraded.

Algorithm 3 Adaptive-selection concept

1: Default TCP variant is TCP Reno.
2: **if** $RTT >= 200ms$ *and* $Bandwidth >= 1Mbps$ **then**
3: TCP variant used is TCP Hybla
4: **end if**
5: **if** $RTT <= 200ms$ *and* $Bandwidth >= 100Mbps$ **then**
6: TCP variant used is TCP HighSpeed
7: **end if**

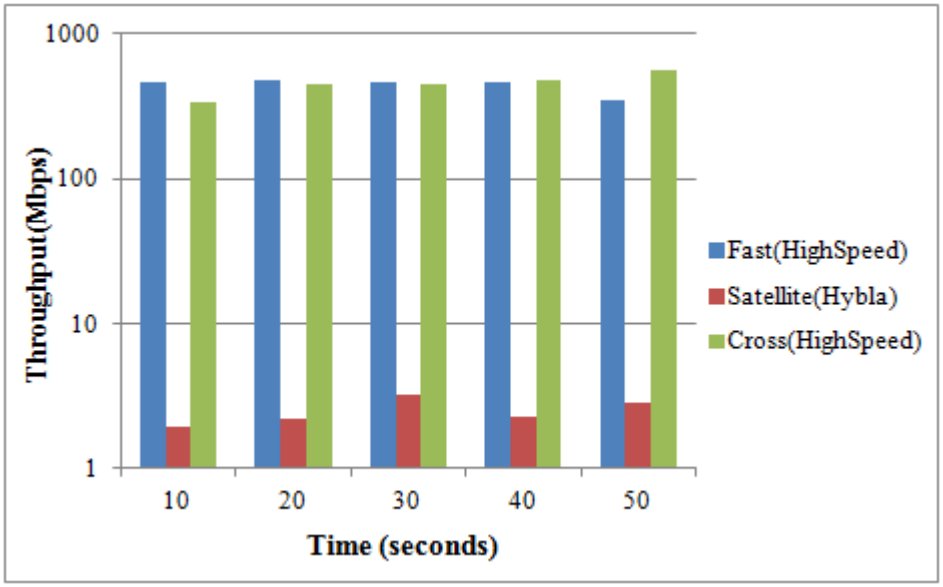


Figure 2.11: Throughput of TCP adaptive concept with 1 % PER on satellite

Simulation topology used for the simulation of adaptive selection concept has been shown in Figure 2.10. The Content Provider (CP) was the sender machine

running TCP adaptive concept and communicating with satellite receiver and wired receiver. The algorithm used in this concept was very simple as described in Algorithm 3.

When TCP adaptive concept algorithm was running on content provider and cross traffic generator, the achieved throughput has been shown in Figure 2.11. PER of satellite link was set to 1%. As per the algorithm, TCP Hybla was selected for satellite link, TCP HighSpeed for the fast connection (i.e., content provider to wired receiver) and TCP HighSpeed for cross generator traffic. Selection of variants as per the algorithm helps the flow to exploit the available link capacity.

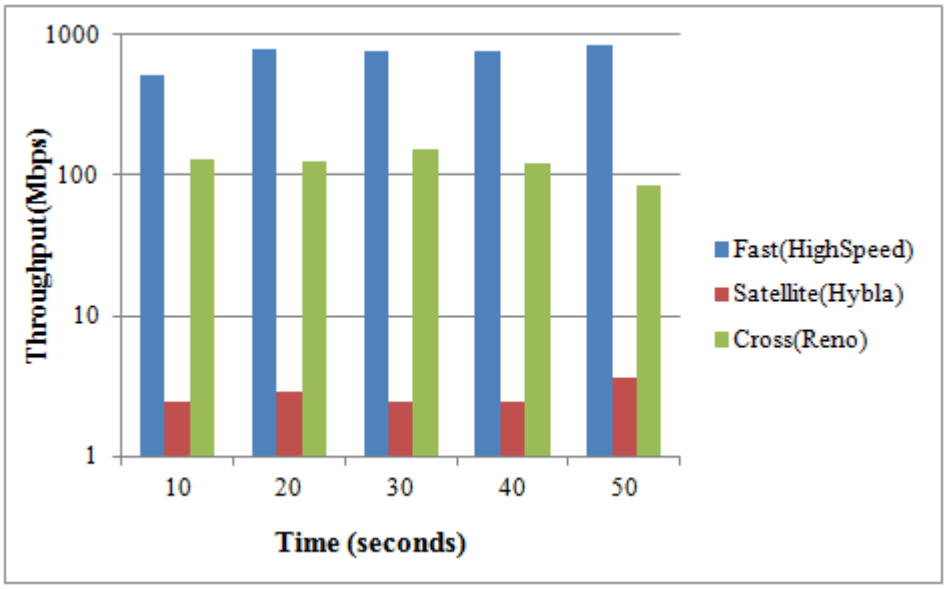


Figure 2.12: Throughput of TCP adaptive concept on CP and reno on cross, with 1 % PER on satellite

When TCP adaptive concept algorithm was running on content provider and Reno on cross traffic generator, the achieved throughput has been shown in Figure 2.12. The results show that Reno on cross connection cannot exploit the link capacity and TCP HighSpeed utilizes the major fraction of the link bandwidth. In another experiment, PER of the satellite link was set to 0% while TCP adaptive concept algorithm was running on content provider and cross traffic generator. Throughput has been shown in Figure 2.13. Compared to the results of Figure 2.11, TCP Hybla could achieve almost two times higher throughput, as there were no link losses on the satellite link.

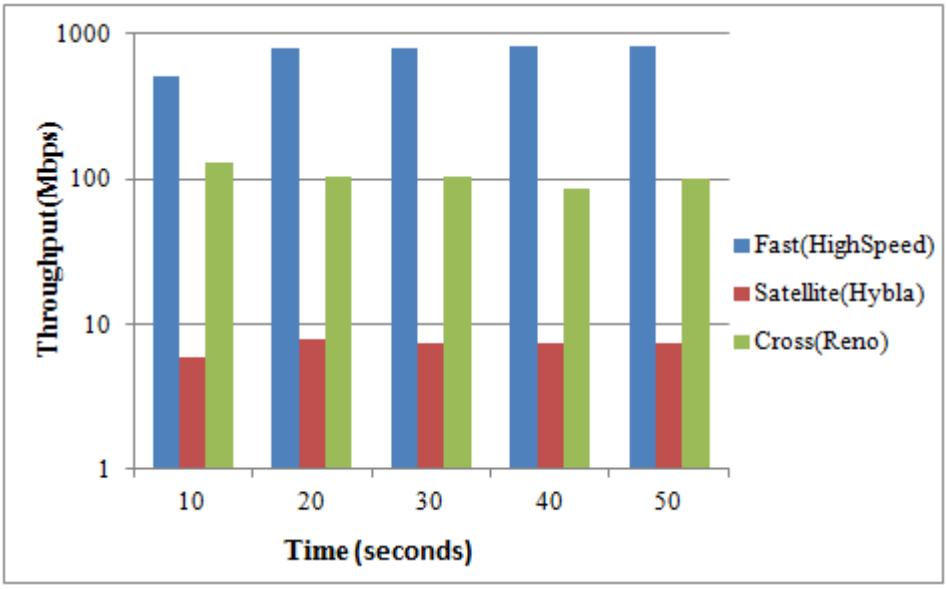


Figure 2.13: Throughput of TCP adaptive concept on CP and reno on cross, with 0 % PER on satellite

In summary, each variant was designed to complement the specific link impairments to optimize the throughput. The next section presents the mathematical analysis of network dynamics that affect the throughput of TCP.

2.4 Mathematical Formulation

The mathematical formulation has been carried out to understand the effect of RTT and PER on the increase rate of cwnd (Jitendra et al. Dordal). Another objective was to understand the effect of sender window size on the throughput of the TCP variant and benefit of using the split approach (PEP).

2.4.1 Effects of RTT and PER

In the analysis, it has been assumed that the congestion control algorithm was operated in the steady state and hence, the oscillation of cwnd remains between maximum cwnd and half of the maximum. It has also been assumed that RTT was constant, TCP retransmission timeout was not considered, and packet error probability was p.

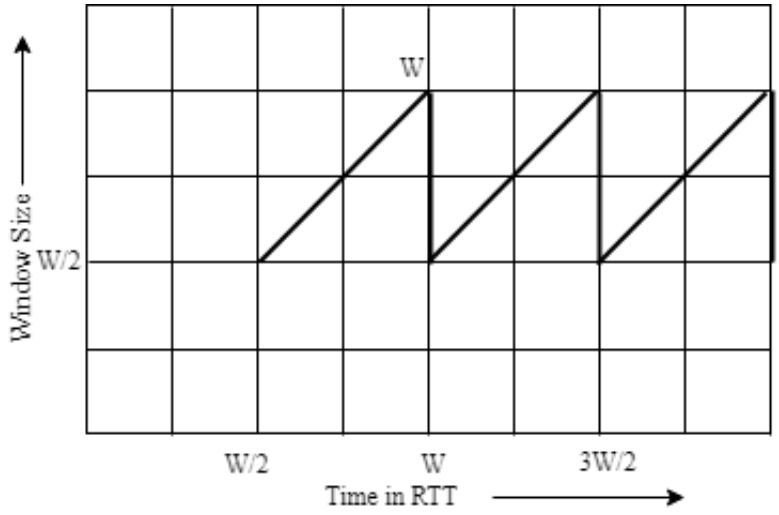


Figure 2.14: cwnd growth v/s RTT

As shown in Figure 2.14, the maximum value of cwnd is W and minimum value of cwnd is $W/2$. Hence, total change in cwnd is $W/2$ and cycle duration is

$$= \frac{(RTT)W}{2} \tag{2.4}$$

Hence, Total data delivered is

$$= (\frac{W}{2})^2 + \frac{1}{2}.(\frac{W}{2})^2 \tag{2.5}$$

$$= \frac{3}{8}.W^2 packets/cycle \tag{2.6}$$

Packets delivered in sequence without loss is $= \frac{1}{p}$

Hence,

$$\frac{1}{p} = \frac{3}{8}.W^2$$

$$W = \sqrt{\frac{8}{3p}}$$

$$Throughput = \frac{Data\_per\_cycle}{Time\_per\_cycle}$$

$$= \frac{\frac{3}{8}.W^2.(MSS)}{RTT.\frac{W}{2}}$$

$$= \frac{MSS}{RTT}.\frac{3}{4}\sqrt{\frac{8}{3p}}$$

$$= \frac{MSS}{RTT}.\frac{C}{\sqrt{p}}$$

$$\begin{aligned} Throughput &\propto \frac{1}{\sqrt{p}} \\ &\propto \frac{1}{RTT} \end{aligned} \tag{2.7}$$

2.4.2 Effects of Window Size

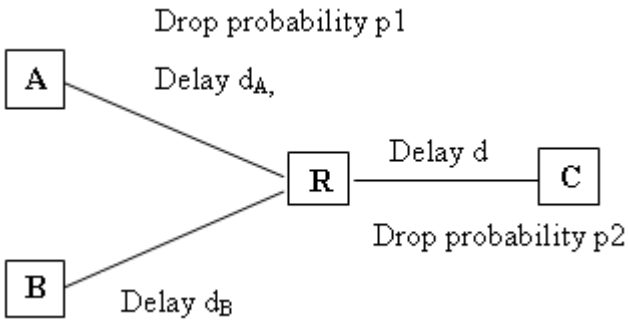


Figure 2.15: Window size and utilization of the link capacity

As shown in Figure 2.15, if fixed window size of sender A is W_A and window size of sender B is W_B, RTT_{actual} is the estimated RTT at the specific instance and RTT_{noload} is the minimum RTT observed for the connection.

$$Queue\_waiting\_time = RTT_{actual} - RTT_{noload}$$

$$Throughput = \frac{WinSize}{RTT_{actual}}$$

Let α denote the fraction of the bandwidth that $A-C$ connection receives and let $\beta = 1-\alpha$, denote the fraction that $B-C$ connection gets. If Q denotes the combined utilization of both connections queue, then queue will have about αQ packets from $A-C$ flow and about βQ packets from $B-C$ flow.

Queue Usage $= WinSize - Throughput \times RTT_{noload}$

From Figure 2.15, RTT_{noload} for $A-C$ connection $= 2(d_A + d)$

RTT_{noload} for $B-C$ connection $= 2(d_B + d)$

Hence, $\alpha Q = W_A - 2\alpha(d_A + d)$ and $\beta Q = W_B - 2\beta(d_B + d)$

$$W_A = \alpha[Q + 2d_A + 2d] \tag{2.8}$$

$$W_B = \beta[Q + 2d_B + 2d] \tag{2.9}$$

Case 1: Delay $d_A = d_B$

If we divide equation (2.8) by (2.9) then

$$\frac{\alpha}{\beta} = \frac{W_A}{W_B} \tag{2.10}$$

Thus, the bandwidth divides in exact accordance with the window size proportions.

Case 2: Delay $d_A > d_B$

In this case, greater fractions of $A - C$ packets are in transit, therefore, fewer will be in queue at R, as a result, α would be smaller than β.

From equations (2.7) and (2.10), it can be concluded that the performance of the TCP variant largely depends on its RTT, window size, packet error rate and RTT experienced by the competing flows. Hence, if we use single TCP variant for all kinds of links, especially in heterogeneous networks, performance cannot be optimal. Therefore, there is a need to have the solution, which needs to be dynamic to all discussed parameters.

2.4.3 Split Approach

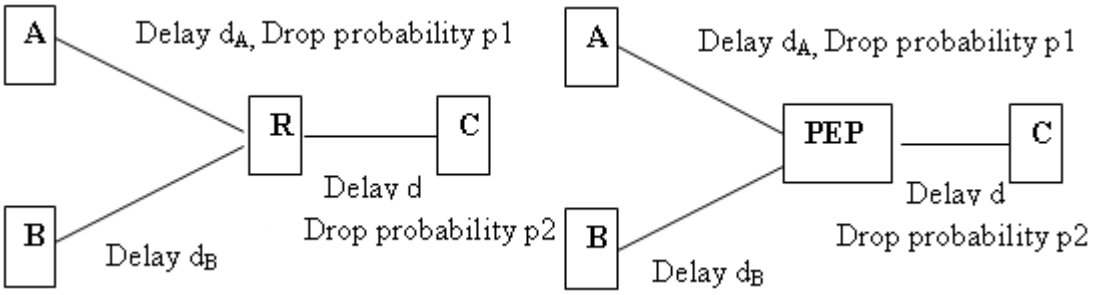


Figure 2.16: End-to-End and split approach

As shown in Figure 2.16, throughput of path $A - R$ and $R - C$ is given by

$$\begin{aligned} THP_{A-R} &= \frac{MSS}{2d_A} \cdot \frac{C}{\sqrt{p_1}} \\ THP_{R-C} &= \frac{MSS}{2d} \cdot \frac{C}{\sqrt{p_2}} \end{aligned} \tag{2.11}$$

If drop probability of p_1 and p_2 are independent of each other, the drop probability of entire path ($A - C$) is

$(p_1 + p_2) - (p_1 \times p_2)$

As long as at least p_1 or p_2 is small, $(p_1 \times p_2) << (p_1 + p_2)$
Hence, above equation can be written as
$(p_1 + p_2) - (p_1 \times p_2) \sim (p_1 + p_2)$
Hence, without the use of PEP, the throughput of the link $A - C$ is

$$THP_{A-C} = \frac{MSS}{2(d_A + d)} \cdot \frac{C}{\sqrt{p_1 + p_2}} \tag{2.12}$$

If either THP_{A-R} or THP_{R-C} is considerably larger than other, then effective throughput

$$= min \left\{ \frac{MSS}{2d_A} \cdot \frac{C}{\sqrt{p_1}}, \frac{MSS}{2d} \cdot \frac{C}{\sqrt{p_2}} \right\} > \left\{ \frac{MSS}{(2d_A + d)} \cdot \frac{C}{\sqrt{p_1 + p_2}} \right\} \tag{2.13}$$

Hence, an improvement is mathematically assured. Thus, the use of split TCP in presence of high RTT and high drop probability results in a significant performance improvement compared to the traditional end-to-end approach (Jain and Ott).

2.5 PEP's Performance Analysis on a Testbed

In above section, mathematically it has been proved that the split approach performs well for a link with high RTT and BER. Furthermore, an experimental analysis is needed to understand the advantage of split approach. The physical setup of the testbed for the performance analysis of PEP has been shown in Figure 2.17. Various networks parameters have been set at the Router 2 using netem (Hemminger et al.). PEPsal (Carlo.Caini, Firrincieli, and Lacamera) has been installed and configured on Router 1. Iperf (https://iperf.fr/) has been used to send the data between the sender and receiver. Each machine was running the Linux kernel 2.6.26 or higher. All these machines were connected through 1 Gbps switch. RTT between Router 2 and receiver was set to 600 ms to emulate the GEO satellite link. The objective of this experiment was to confirm the benefits of the split approach as discussed in the previous section. Hence, default parameters of the kernel were used without any performance tuning of local kernel parameters.

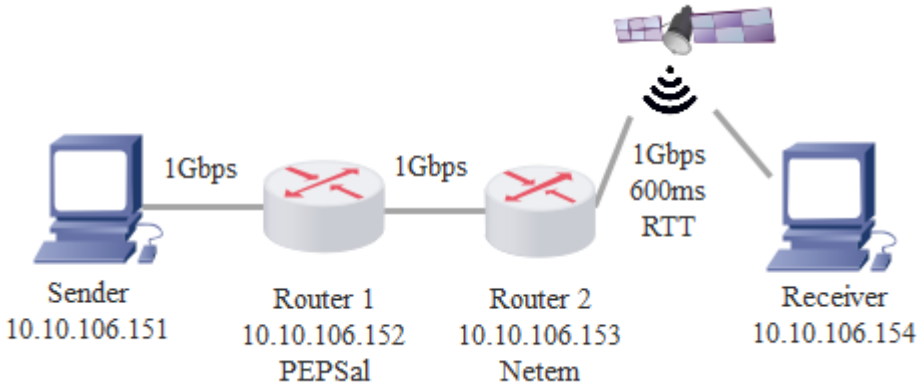


Figure 2.17: Testbed setup for performance analysis of PEP

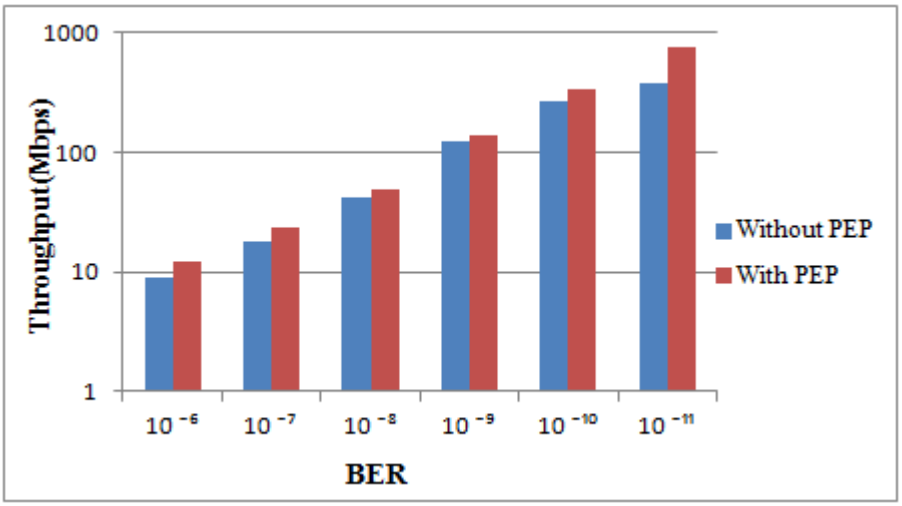


Figure 2.18: Throughput of TCP Hybla with and without PEP, RTT= 600 ms

Throughput comparison of TCP Hybla, and TCP Hybla with PEP, for different values of BER from 10^{-6} to 10^{-11} has been shown in Figure 2.18. The experiment was conducted for the duration of 600 seconds. The results indicated that the performance of TCP Hybla with PEP was 1.14 to 2 times better compared to TCP Hybla (i.e., without PEP).

Throughput comparison of TCP Cubic (i.e., default TCP variant in Linux kernel 2.6.19 onwards) and TCP Cubic with PEP has been shown in Figure 2.19. The experiment was conducted for the duration of 600 seconds, having satellite link of 600 ms RTT and BER has been changed from 10^{-6} to 10^{-11}. The results indicated that the split approach gave higher performance compared to Cubic, as proved in mathematical analysis. From the results, it could be concluded that compared to

Cubic, Cubic with PEP gave 1.15 to 3.91 times better throughput.

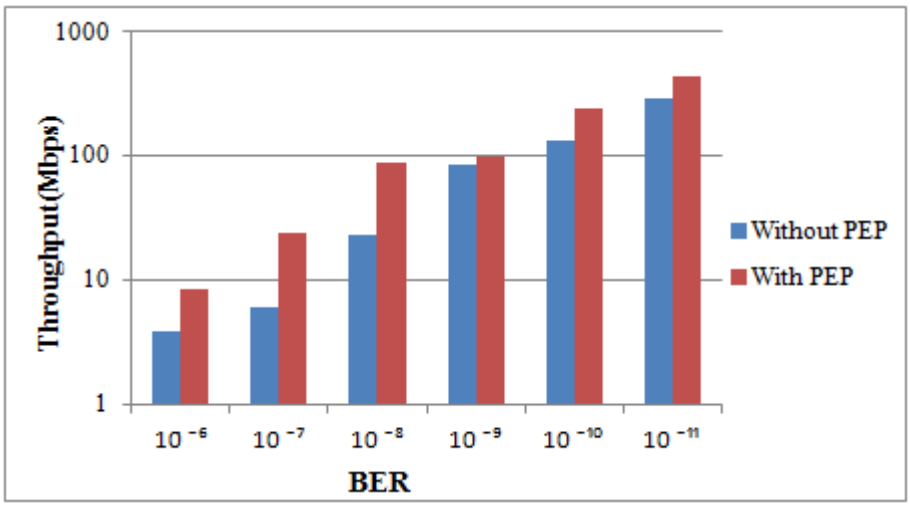


Figure 2.19: Throughput of TCP Cubic with and without PEP, RTT= 600 ms

2.6 Summary

As discussed in the above sections, present Internet is the classical example of the heterogeneous networks. Heterogeneity of such networks has not been only limited to the networking technologies but, it also consists of emerging applications, diversity of platforms and devices, and implementation issues with compatibility as a major concern. Each newly designed TCP variant can do better for the specific type of link or networking technologies. For example, NewReno has been designed for lossless links, TCP Hybla has been designed for high RTT links (e.g., satellite links), Cubic has been designed for low to high RTT and lossless links, while TCP Westwood has been designed for the links with high BER (e.g., Wi-Fi link). Every emerging application needs to utilize the default and one of the TCP variant (i.e., at any given instance) depending on the operating system (e.g., Linux kernel 2.6.19 uses Cubic) of the sender's machine.

Performance comparison of TCP variants : BIC, Cubic, HighSpeed, Hybla, Illinois, LP, NewReno, Scalable, Veno, Westwood, and Yeah carried out on heterogeneous networks. Results have been summarized below:

- In the presence of background traffic, on a link with RTT of 25 ms, Cubic was able to achieve throughput of 2 Mbps (Fair sharing bandwidth was 1.66 Mbps). On high RTT link (i.e. 300 ms and higher), in presence of background traffic

on low RTT (e.g. 25 ms) links, Cubic has achieved throughput of less than 5%. In this case, the major fraction of bottleneck bandwidth was utilized by Cubic on low RTT flows.

- NewReno has been able to achieve fair sharing of bottleneck bandwidth when all competing flows have low RTT (e.g., 25 ms RTT). However, its performance was significantly low (1% only) on high RTT links (e.g., 300 ms and higher), in the presence of background traffic flows on low RTT links. Thus, NewReno has fairness issues on heterogeneous networks.

- On high RTT link (e.g., GEO satellite with 600 ms RTT), Hybla was able to exploit more than 70% of the link bandwidth irrespective of the data transfer durations whereas all other variants exploited less than 30%. Scalable and Yeah were able to exploit more than 75% link bandwidth for data transfer durations of 2 minutes and more. Thus, existing variants have start-up performance issues on high BDP links.

- In the presence of five background traffic sources via 25 ms RTT links, throughput achieved on links with varying RTT exhibited fairness issues for all variants. Hybla was able to operate around 1.2 Mbps (Fair share was 1.66 Mbps, for 10Mbps bottleneck and 6 flows). However, in the presence of background traffic, and link errors (i.e., higher values of PER), the throughput of Hybla was significantly lower.

- In the absence of background traffic, with PER of 1% and higher, throughput achieved on 600 ms RTT link was 10% of the bottleneck link capacity for all variants. Hybla was able to exploit 30%. In the presence of background traffic with PER of 5% and higher throughput degraded to 5% and below for all variants.

- Further, experiments were carried out on simulation topology of "TCP adaptive selection concept". It has applied a very simple algorithm that permits use of three variants, namely HighSpeed, Reno, and Hybla. It has been proved that change of variant for different types of links, resulting in significant performance improvement compared to the usage of a single and fixed variant for all

types of links. However, futuristic variants' adaptiveness, application and link classifications, and experimental analysis still need to be addressed.

In Section 2.4, mathematical analysis has been presented to understand the parameters that affect the performance of TCP. These analyses suggest that the performance of TCP was inversely proportional to RTT of the link and degrades as the BER increases. Further, the window size was also an important parameter, which affects the bottleneck link's bandwidth sharing capacity among competitive flows. Finally, it was proved that compared to the traditional end-to-end approach, a split approach could always perform better. To confirm the advantage of the split approach, a testbed setup was used, and performance analysis has confirmed that the variant like Cubic (along with PEP) can improve the performance by 3.91 times. Further, this split approach made the deployment easier and feasible in the present Internet architecture.

Overall, it can be concluded that all existing TCP variants either specifically tailored for the special kind of links or the concept like TCP adaptive selection raised the issue of having some dynamic transport layer mechanism. In subsequent chapters, these issues have been addressed.

Chapter 3

Tarang-A Protocol for the link with variable BER and RTT

It was concluded in the previous chapter that all existing TCP variants (including NewReno, Cubic, Westwood, Hybla) have performance and fairness issues for the link with variable BER and RTT. Variants like HighSpeed, Scalable, Yeah, Illinois and TCP Fast were designed for the high BDP links, but the performance degraded in the presence of link losses. GridTCP was designed to overcome the issue of high BER and RTT, but it had few operational issues. Hence, there was a need to design a TCP variant which resolve the issues of: startup performance on high BDP links, performance on high BER (e.g., 10^{-6}) links, fairness in heterogeneous networks, poor performance on link with variable BER, RTT (e.g., Satellite link has very high RTT and BER. In addition, the RTT and BER of satellite link varied significantly with time due to environmental effects). In this chapter, the detailed performance comparison of Hybla, Westwood, Cubic, NewReno along with newly designed Tarang- for the link with variable BER and RTT, has been presented. Further, it has been proved that the improvement in Tarang was significantly high for the link with variable BER and RTT, and Tarang was able to achieve fairness in heterogeneous networks. Tarang has been implemented and tested in ns-2 (http://www.isi.edu/nsnam/ns/).

The remaining chapter has been organized as follows: Section 3.1 gives the introduction and highlight the need for the new protocol. Section 3.2 describes the related work for the link with high RTT and high BER. Section 3.3 presents Tarang's design and algorithm in details. Section 3.4 describes the simulation topologies and

detailed comparison of TCP Hybla, TCP Westwood, Cubic, and Tarang. Finally, Section 3.5 summarizes the chapter.

3.1 Introduction

TCP variants like Taheo, Reno, and NewReno give an optimized performance for wired networks. These protocols follow AIMD approach in which cwnd is incremented by one on receipt of an acknowledgment during Slow Start (SS) phase, and cwnd is incremented by one per RTT during Congestion Avoidance (CA) phase. The rate at which acknowledgments are received depends on RTT of the connection. Hence, the higher the RTT, the lower cwnd increase rate. If the rate of increment of cwnd is low, the link capacity cannot be utilized effectively. Compared to wired link, the satellite link has very high RTT. Hence, the performance of TCP NewReno on a satellite link is significantly poor. TCP NewReno assumes congestion by means of packet loss. Packet loss is indicated either by three duplicate acknowledgments or timeout. Further, packet loss results in the reduction of cwnd. In a wireless link (e.g., satellite, Wi-Fi) a packet loss can also occur due to the link error. As a result, standard TCP (i.e., NewReno, Cubic) cannot differentiate the reason of the loss and reduces the cwnd in case of link error. Hence, such unnecessary reduction of cwnd worsens the performance further. The next section elaborates related works for the link with high RTT and high BER.

3.2 Related Work

TCP Westwood was proposed to solve the issue of loss discrimination on the wireless link (Saverio et al.). It does so by continuous estimation of the bandwidth used by the connection. If packet loss occurs, it sets ssthresh and cwnd according to the estimated bandwidth. It solves the issue of packet error rate and improves the performance on the wireless link. In the presence of high RTT, TCP Westwood performs poorly as it was not designed for high RTT link. Hence, the performance of TCP Westwood is significantly lower in the presence of high BER and High RTT link. TCP Vegas was designed to overcome the congestion losses by means of bandwidth estimation. Still, it has performance issues for high RTT links, as maximum cwnd increase rate was same as NewReno. TCP Peach was designed with “sudden-start”, and “rapid-

recovery" mechanisms which depend on the use of dummy segments to estimate the available bandwidth. TCP Peach requires the support at IP layer of all intermediate router to discard the dummy segments in the presence of congestion.

TCP Hybla was proposed to resolve the problem of high RTT for the satellite link (Caini and Firrincieli). TCP Hybla tries to achieve the same data sending rate as of reference wired connection. It does so by making cwnd increase rate independent of RTT. TCP Hybla uses the normalized RTT to set ssthresh and cwnd after the congestion episode. In this way, Hybla was able to increase cwnd very fast, even for the high RTT link. Thus, TCP Hybla was able to utilize the available bandwidth for the link with high RTT (e.g., satellite). The performance of TCP Hybla was lower in the presence of high BER, as it was not designed to perform well in the presence of high BER. Further, variants like TCP Scalable, Yeah, and Cubic were designed for high BDP links, but performance was an issue in the presence of high RTT and BER. GridTCP was designed for the link with high RTT, BER, but it has the operational issues like user needs to specify the number of connection to be used.

Thus, there was a need to design a TCP variant for the link with variable RTT and BER.

3.3 Tarang's Design and Algorithm

Tarang has been developed for the links with variable RTT and BER. It uses the modified approach for the congestion window update in SS phase and after congestion in CA phase, as detailed in the following subsections. Tarang uses bandwidth estimation by CapProbe (Kapoor et al.) and Eligible Rate Estimation (ERR) technique similar to TCP Westwood.

3.3.1 Congestion Window Update

Tarang's cwnd update rule is the same as TCP Hybla in SS and CA phase. Standard TCP (i.e., Reno, NewReno) in SS, initializes cwnd with one segment and increases it by one on receiving an acknowledgment for the same. In CA phase, it increases cwnd by one per RTT if all the segments are acknowledged. Tarang equalizes the data sending rate of high RTT link (e.g., satellite link) with reference to low RTT

link (e.g., wired connection). It calculates the normalized round trip time ρ as:

$$\rho = RTT/RTT0 \tag{3.1}$$

Where, RTT0 is round trip time of reference wired connection to which Tarang tries to equalize the data sending rate. Tarang's cwnd update rules are:

$$\begin{aligned} W_{i+1}^H &= W_i^H + 2^\rho - 1, SS \\ &= W_i^H + \frac{\rho^2}{W_i^H}, CA \end{aligned} \tag{3.2}$$

In above equation, W_i^H is cwnd when i^{th} acknowledgment has been received. When Tarang operates in SS phase and receives an acknowledgment, then cwnd is increased by $2^\rho - 1$. In CA phase, it increases cwnd by ρ^2/W_i^H after receiving acknowledgments for all the transmitted segments. At the sender, instead of transmitting packets immediately upon receipt of the acknowledgment, the sender can delay transmitting packets to spread them out at the rate defined by the congestion control algorithm which is given by the window size divided by the estimated RTT. Thus, packet pacing was achieved by following a strategy at the sender. Alternatively, a receiver could delay acknowledgments to spread them across RTT, so that when they arrive at the sender, they will trigger spaced data packets. The goal of pacing was to evenly spread the transmission of a window of packets across the entire duration of RTT.

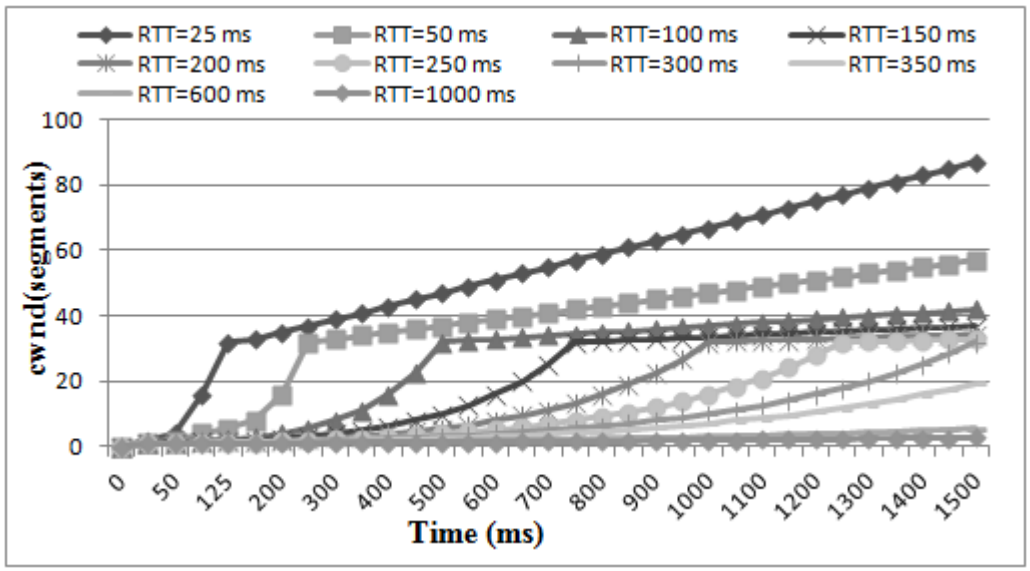


Figure 3.1: NewReno's cwnd increment with respect to time for various links with different RTT

A modified approach to increase cwnd in SS phase enabled Tarang to achieve cwnd peak value at the same time for all links, irrespective of RTT variations. In CA phase also, cwnd increased penalty for high RTT link has been avoided by the modified equations. Thus, Tarang's startup behavior for high RTT link has improved and performance of Tarang has become independent of RTT (i.e., because the rate of increase of cwnd is same for all link types irrespective of RTT variations). The rate of increment of cwnd for links with different RTTs for NewReno and Tarang has been shown in Figure 3.1 and Figure 3.2 respectively.

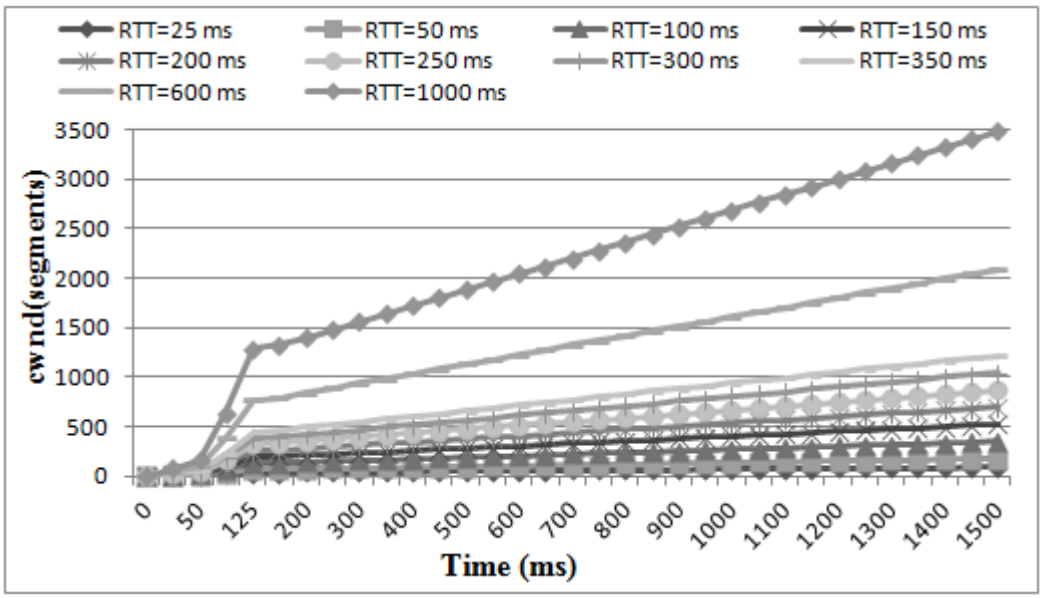


Figure 3.2: Tarang's cwnd increment with respect to time for various links with different RTT

3.3.2 Bandwidth Estimation and Congestion Episode

TCP Hybla assumes the congestion to be caused in the network due to the loss of segments. Segment loss has been detected by means of either three duplicate acknowledgments or when the timer goes off. In a case of three duplicate acknowledgments, it sets ssthresh to half of the current cwnd, and cwnd is updated to the one fourth of the current cwnd. When the timer goes off, it sets ssthresh value to same as in case of three duplicate acknowledgments, but cwnd is set to ρ. Thus, TCP Hybla assumes that the loss is due to the congestion only and so it performs poorly on wireless link (e.g., Wi-Fi, satellite) where losses can also happen due to the link characteristics. Tarang was able to increase cwnd significantly even after the loss of segments due to

the link error. This results in significant performance improvements in the presence of link losses which is the usual scenario for the satellite links.

Tarang uses the indirect way to differentiate the reason for loss of the segment. In case the loss has been due to the link characteristics, instead of decreasing, cwnd needs to be increased. To achieve this objective, Tarang tries to estimate utilized bandwidth. To estimate utilized bandwidth of the link, Tarang counts received segments by measuring the number of acknowledgments received. After that, it calculates utilized bandwidth by dividing received bytes by the time period for which acknowledgments were counted. After detecting a loss of segment (which can be triggered by either three duplicate acknowledgments or time out), new ssthresh and cwnd have been set as per the Algorithm 4 and Algorithm 5. Notations used in the algorithms have been mentioned as under:

BWE: Utilized bottleneck bandwidth by a specific flow.

CC: Bottleneck Channel Capacity.

RTT_{min}: Minimum value of RTT observed over the time.

seg_size: It is the size of segments at transport layer.

packets_in_flight: Number of segments currently in flight.

Algorithm 4 Tarang's algorithm after congestion episode (After 3 DUPACK received)

```
1: if 3 DUPACK received then
2:     if BWE < CC/2 then
3:         Residual pipe capacity = (CC − BWE) ∗ RTTmin/seg_size
4:         ssthresh= MAX (Residual pipe capacity, 0.75* packets_in_flight)
5:         cwnd= MAX (Residual pipe capacity, 0.75* packets_in_flight)
6:     else
7:         Minimum pipe capacity = (BWE ∗ RTTmin)/seg_size
8:         ssthresh = MAX(Minimum pipe capacity, 0.75*packets_in_flight)
9:         cwnd = MAX(Minimum pipe capacity, 0.75*packets_in_flight)
10:    end if
11: end if
```

According to the Algorithm 4, after receiving three duplicate acknowledgments,

if bandwidth used by connection was found to be less than half of the actual capacity of the link, then ssthresh and cwnd were set according to the difference between the actual capacity of the link and bandwidth used by the connection. This was the major modification compared to TCP Westwood. TCP Westwood used the minimum pipe capacity to set the value of ssthresh and cwnd after detecting the loss by means of three duplicate acknowledgments. This resulted in significant performance improvement.

Algorithm 5 Tarang's algorithm after congestion episode (After timeout)

```
1: if Timeout then
2:     if BWE < CC/2 then
3:         Residual pipe capacity = (CC − BWE) ∗ RTTmin/seg_size
4:         ssthresh= MAX (Residual pipe capacity, 0.75* packets_in_flight)
5:         cwnd = ρ
6:     else
7:         Minimum pipe capacity = (BWE ∗ RTTmin)/seg_size
8:         ssthresh = MAX(Minimum pipe capacity, 0.75*packets_in_flight)
9:         cwnd = ρ
10:    end if
11: end if
```

If bandwidth used by connection was greater than half of the actual capacity of the link, then bandwidth used by connection was used to set the value of ssthresh and cwnd. Hence, when used bandwidth was less, then the connection was aggressive. When a timeout occurs, Tarang uses the Algorithm 5, as described above and cwnd was set to ρ.

3.3.3 Events Details

After initialization of the variables, Tarang's function *tcp_tarang_pkts_acked* has been invoked where, if the number of packets acknowledged is greater than 0, then it sets the $w \rightarrow xrtt$ parameter. Then, it checks for the events which can be one of the four as follows:

- CA_EVENT_FAST_ACK: For acknowledgments were received in sequence.

- CA_EVENT_COMPLETE_CWE: Called when recovery of cwnd finished.
- CA_EVENT_FRTO: Called after timeout.
- CA_EVENT_SLOW_ACK: When acknowledgments were not received in sequence.

Algorithm 6 AckedCount

```
1: cumul_ack = current_ack_seqno − last_ack_seqno
2: if cumul_ack = 0 then
3:     accounted_for= accounted_for + 1
4:     cumul_ack = 1
5: end if
6: if cumul_ack > 1 then
7:     if accounted_for >= cumul_ack then
8:         accounted_for= accounted_for − cumul_ack
9:         cumul_ack=1
10:    else if accounted_for < cumul_ack then
11:        cumul_ack= cumul_ack − accounted_for
12:        accounted_for=0
13:    end if
14: end if
15: last_ack_seqno = current_ack_seqno
16: acked = cumul_ack
17: Return acked
```

After cwnd recovery has been completed, it compares bandwidth estimated using CapProbe method with that of bandwidth estimated with ERE and accordingly sets ssthresh and cwnd values as elaborated in Algorithm 4. After the timeout, Tarang uses the Algorithm 5 to set the values of ssthresh and cwnd. If acknowledgments arrive in sequence then after every acknowledgment, it calculates the Estimated Bandwidth (BWE) parameter by calculating the delta as a difference between the current time stamp and previous time stamp. Further, data sending rate has been measured as a difference between current acknowledgment number that TCP has been waiting for

and previous next byte that was waited. When acknowledgment received is not in sequence, then event SLOW_ACK occurs, and it calls Algorithms 6 to calculate the cumul_ack. Algorithm 6 removes any discrepancy due to delayed or out of ordered acknowledgments. Afterwards, the delta is measured as a difference between the current time stamp and previous time stamp to calculate BWE.

3.4 Experiments and Results

To insert the Tarang's code in ns-2, a new software module was written whereas existing modules of other variants were used for the performance comparison. Tarang's code was written for the ns-2-TCP-Linux patch (Wei and Cao). This patch provided the facility to use congestion control modules of Linux kernel in ns-2. This patch worked as a bridge between ns-2 and Linux congestion control module. Performance evaluation was carried out using ns-2 for the simulation scenarios as described below.

In the first experiment, simulation topology that used was same as the one used for the analysis of Hybla (Caini and Firrincieli) and comparative analysis was performed between Hybla, Westwood, and Tarang as detailed in subsection 3.4.1. Experiments allowed the performance evaluation for varying RTT in the presence of congestion without link losses, in the presence of link losses without congestion, and with both, congestion and link losses. It was observed that these experiments were not sufficient for the satellite link characteristics. Hence, following experiments were designed: i) single-hop satellite link with varying BER to simulate a satellite link in varying weather conditions. In these experiments, BER of 0 and 10^{-5} were considered for benchmarking and extremely high error rate link (i.e., although it probably never occurs practically on satellite link) respectively. ii) a satellite link with average BER was considered with varying RTT including multi-hop satellite link. RTT values of lesser than 500 ms were considered to show the effect of varying RTT even though it never occurs practically on a satellite link. iii) experiments with varying file sizes on single-hop satellite link with average, high and negligible (i.e., for benchmarking) BER. iv) heterogeneous network using dumbbell topology for fairness with varying BER and RTT. Results and discussion on these experiments have been presented in subsection 3.4.2.

3.4.1 Comparative Analysis between Hybla and Tarang

Simulation topology used was the same as specified in (Caini and Firrincieli) for the performance comparison with the existing variants and it has been shown in Figure 3.3. The foreground TCP connections consisted of wired and wireless links (i.e., a satellite link), and background traffic was generated on wired links. All the TCP connections shared a bottleneck link (i.e., a link between R1-R2). The RTT of the satellite link was varied from 25 ms (for the comparison with the wired link) to 600 ms (to simulate GEO satellite link). All wired links were assumed to be error free, and satellite link PER varied in the range 0 to 5 % with uniformly distributed error model. All the remaining parameters as specified in the Figure 3.3 was benchmarked as per the parameters covered in (Caini and Firrincieli) for the comparison purpose.

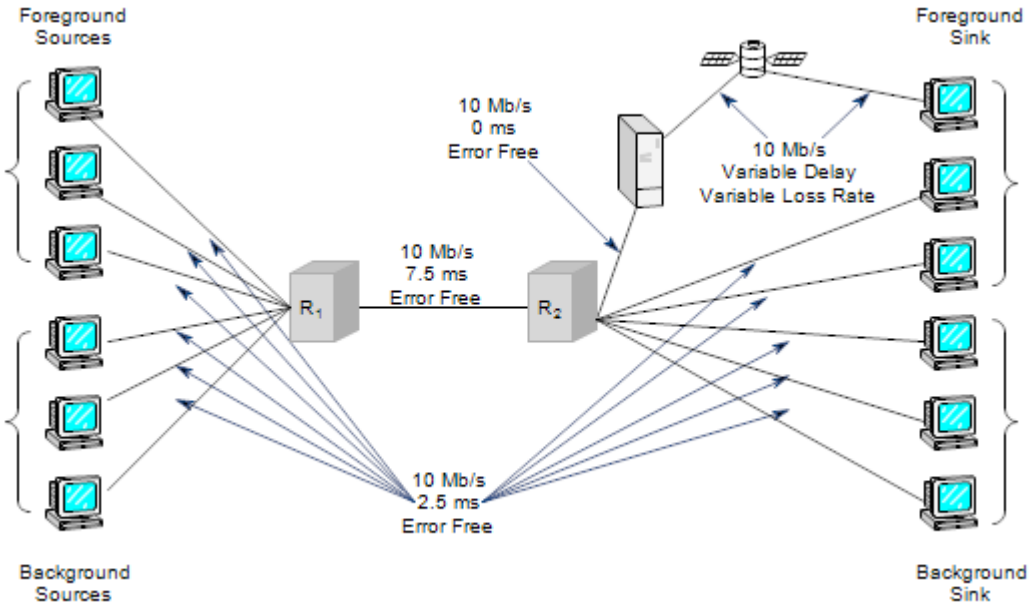


Figure 3.3: Simulation topology for performance comparison of Tarang with existing variants

For each TCP connection that originated from the sender or background source, a persistent FTP application was used to transfer the data with the simulation time of 600 seconds. The performance was evaluated in terms of throughput for variation in RTT and PER in the range of 0 to 5%. Performance of Tarang was compared with the following existing TCP variants:

- **TCP Hybla:** Variant proposed for high RTT link to improve the performance especially on the satellite link, it was used with SACK and packet spacing (Caini and Firrincieli).

- **TCP Westwood:** TCP variant designed for the wireless link to increase the performance in the presence of BER (Saverio et al.). It continuously measures utilized bandwidth by a connection for the sender. TCP Westwood code was used from Linux kernel which has implementations of Westwood+.

- **TCP Vegas:** Variant designed to prevent the losses due to congestion by estimation of the available bandwidth (Brakmo and Peterson).

- **TCP NewReno:** It was the modification of Reno that uses the concept of partial acknowledgment to achieve better throughput in the presence of multiple losses in a window (Henderson et al.).

3.4.1.1 Performance in presence of congestion

A TCP connection on satellite link with variable RTT and five TCP connections on the wired link were simultaneously sending data to corresponding receivers. TCP connection passing through the satellite link used different TCP variants for comparison and all background traffic sources were using NewReno. To measure the performance in the presence of congestion, the satellite link was assumed to be error free. Thus, all the losses occurred because of the shared link that acted as the bottleneck for all the TCP flows. In such a scenario, TCP NewReno had performance issues on satellite link with high RTT and a major fraction of the bottleneck link bandwidth was utilized by active TCP flows on wired links. To highlight this problem and to measure the effectiveness of Tarang, Figure 3.4 shows the throughput achieved on satellite link with respect to variations in RTT. Performance achieved on the wired link has not been represented as the other parameters of that link were unchanged.

As shown in Figure 3.4, as RTT increases, the performance of NewReno on satellite link degrades and this issue was mentioned in (Padhye et al.). TCP Vegas performs little better compared to NewReno. TCP Hybla was able to achieve higher throughput compared to NewReno and Vegas, but at higher values of RTT and congestion losses, its performance degrades. As Tarang uses the estimated bandwidth

to set the value of ssthresh and cwnd after the congestion, it was able to utilize the available bottleneck link capacity. TCP flows that were running on the wired link also suffer the congestion. Hence, NewReno on cross traffic sources cannot exploit the bottleneck bandwidth fully and allow Tarang to utilize the major fraction of the underutilized bandwidth of the bottleneck link. As shown in Figure 3.4, Tarang operates between 2 to 3 Mbps whereas Hybla operates between 1 to 2 Mbps.

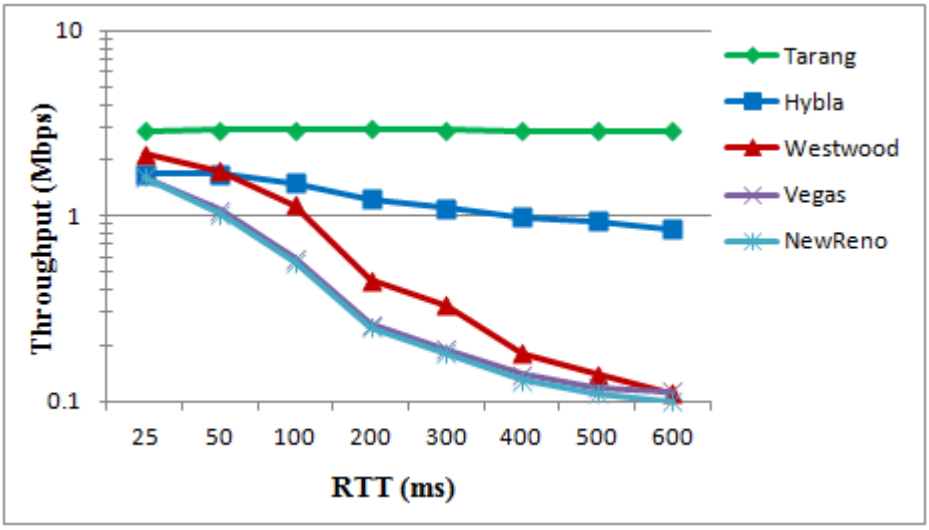


Figure 3.4: Performance comparison of Tarang and existing variants in presence of congestion

3.4.1.2 Performance in presence of link losses

In order to understand the effect of link errors on the performance of TCP variants, it was necessary to keep the congestion issue separate. Thus, in Figure 3.3, only TCP flow on satellite link was active, and any loss that happened was due to the link error. PER was set to 1%. TCP NewReno does not have the mechanism to distinguish reason of the packet loss (i.e., due to link error or congestion), and that resulted in the unnecessary reduction of cwnd in case of link losses.

Performance of different variants on satellite link with varying RTT values and 1% PER has been shown in Figure 3.5. As expected, the performance of NewReno was significantly low and become very poor as RTT increased. Performance of Vegas was slightly better than NewReno but, significantly low with respect to bottleneck link capacity (i.e., 10 Mbps). Initially, Westwood was able to operate around maximum available capacity but, as RTT increased and in the presence of 1% of PER, it was not

able to sustain and was able to achieve the throughput of less than 0.5 Mbps at RTT of 600 ms. Hybla was able to maintain the throughput of 2 to 4 Mbps even at higher value of RTT, as it uses the normalized RTT and modified equations to set the values of ssthresh and cwnd. As Hybla does not have any mechanism to differentiate the reason of the loss, it can exploit the maximum of 40% of the available link capacity in the presence of link losses. Tarang uses the estimated bandwidth to set the values of ssthresh and cwnd after congestion episode that allow it to exploit significantly high bandwidth in presence of link losses compared to other variants including Hybla. As shown in Figure 3.5, Tarang was able to maintain throughput around 6 Mbps and even at RTT of 600 ms, it was able to achieve throughput of more than 5 Mbps. Thus, Tarang was able to offer 100% higher performance compared to Hybla in the worst-case simulation scenario considered in this experiment.

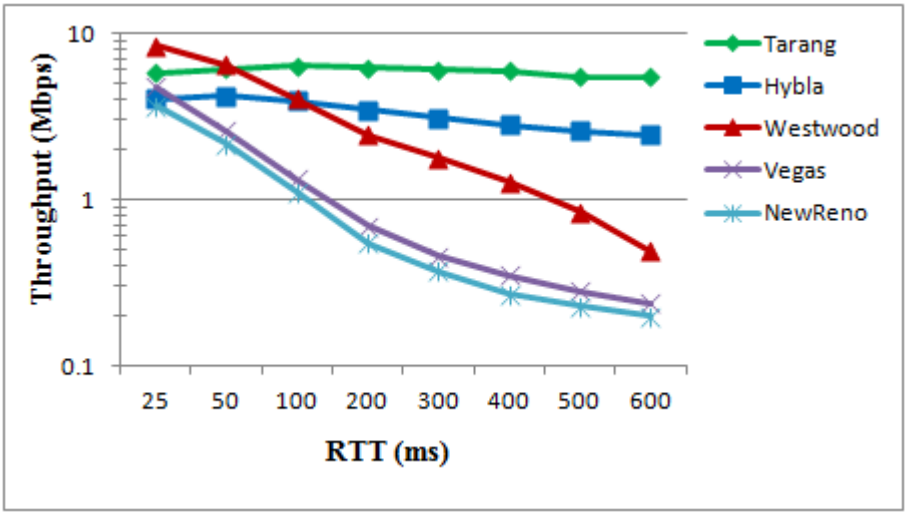


Figure 3.5: Performance comparison of Tarang and existing variants in presence of link losses (PER 1%)

3.4.1.3 Performance in presence of both congestion and link losses

As discussed in above sections, TCP variants like NewReno and Vegas were not able to utilize available link bandwidth in the presence of high RTT or link losses. Thus, in this section, the performance of Hybla, Westwood, and Tarang have been compared in the presence of link losses and congestion on satellite link for different values of RTT.

Throughput achieved on satellite link by Westwood, Hybla, and Tarang, for

PER of 0%, 1% and 5%, in the presence of congestion with varying RTT, has been shown in Figure 3.6. Performance of Hybla was reasonably good in the presence of both, congestion and link losses compared to TCP variants like NewReno and Vegas which were unable to perform even in the presence impairments considered separately (i.e., either link losses or congestion). At PER of 5% and RTT of 600 ms, throughput achieved by Westwood was less than 0.1 Mbps, and Hybla operates around 0.5 Mbps. The reason for such a poor performance was that Westwood and Hybla do not have any mechanism to cope with both, high RTT and segment losses at the same time. At PER of 5% and in the presence of congestion with higher values of RTT, Tarang was able to operate around 2 Mbps. These results demonstrated the capability of Tarang to cope with impairments like high RTT, link and congestion losses.

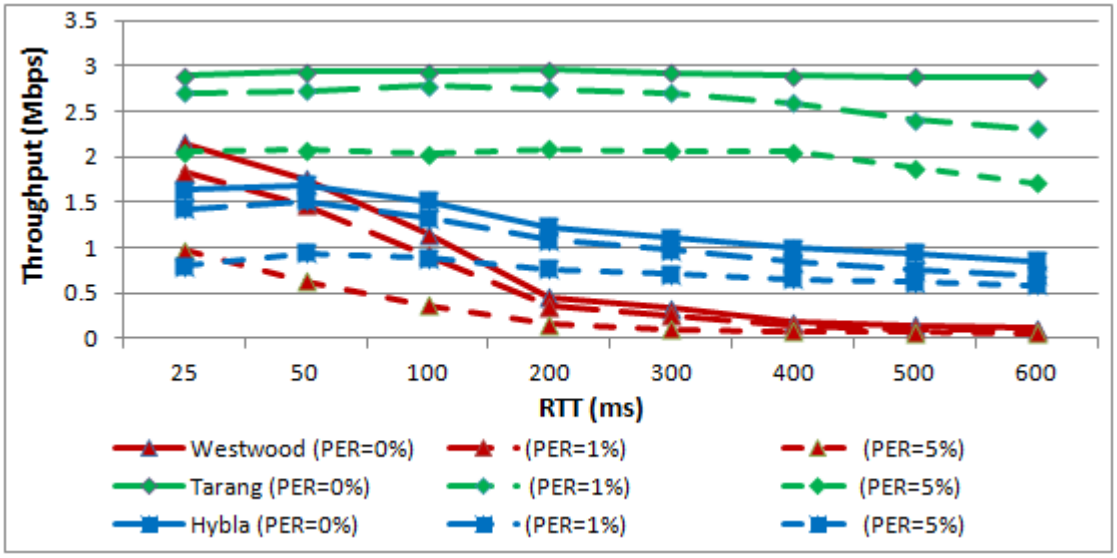


Figure 3.6: Performance comparison of Tarang and existing variants in presence of congestion and link losses (PER of 0%, 1%, 5%)

3.4.2 Enhanced Topology Testing

Simulation scenario has been shown in Figure 3.7, which was used to evaluate the performance of Tarang and its comparison with the existing variants Hybla and Westwood. This simulation scenario allows the performance evaluation and comparison with varying parameters like BER, RTT, and file-size to confirm the advantages of the Tarang compared to existing variants.

As shown in Figure 3.7, satellite link was configured to have 100 Mbps of uplink

and downlink. RTT was set to 600 ms. The objective of these experiments was to test how much link bandwidth can be exploited by the Tarang (especially for the link with varying BER, RTT) and other variants, without considering the competitive flows and fairness issues. In all simulations scenarios, segment size was set to 1500 bytes.

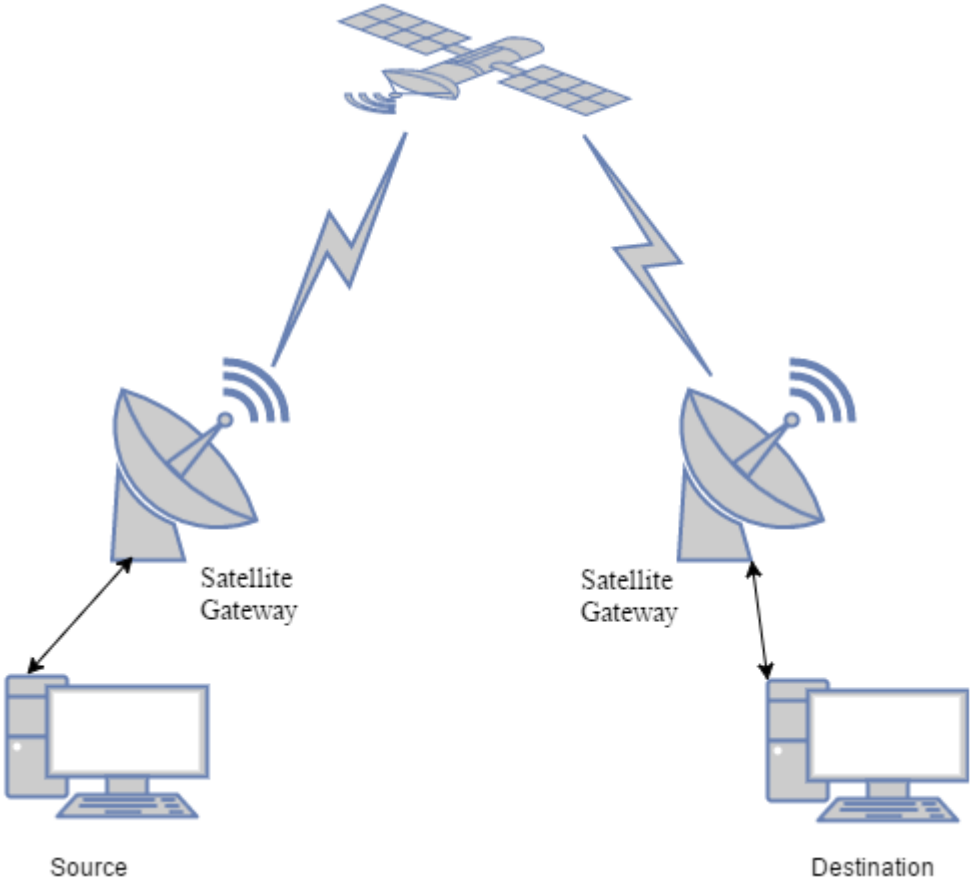


Figure 3.7: Simulation topology for performance analysis of Tarang on satellite Link

3.4.2.1 Single-hop satellite link

Throughput comparison of Hybla, Westwood, and Tarang with respect to varying BER of the satellite link has been shown in Figure 3.8. BER of 0 represents the situation where there was no loss at all (i.e., although this situation may not occur on GEO satellite link, but it has been considered for the comparison only). Hybla starts as the standard TCP, but its ssthresh value was set as per the initial estimation of the bandwidth which results in significant performance improvement in initial few seconds. As Hybla's SS and CA phases are driven by normalized round trip time, it results in faster increments of the cwnd. Hence, it starts operating at 96 Mbps within

100 seconds of the simulation time. The same result was quite apparent for Tarang because it was the no loss scenario. Westwood starts with SS and it takes the longer amount of time to increase the cwnd, as its increase rate depends on the flow of the acknowledgments. As Westwood was not specifically designed for the high RTT link, it takes the longer amount of time to operate at the peak of the available bandwidth.

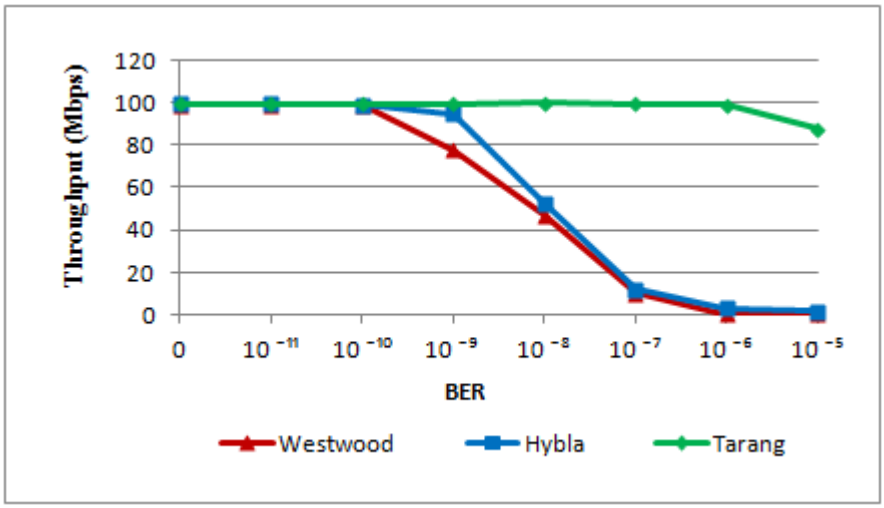


Figure 3.8: Performance comparison for single-hop satellite link with varying BER

As shown in Figure 3.8, the performance of Westwood remains below 50 Mbps for BER of 10^{-8} and RTT of 600 ms. The reason for such a performance was that Westwood increases its cwnd based on the flow of acknowledgments, and set the ssthresh and cwnd based on the utilized bandwidth after the event of an error. Hybla also operates around 50 Mbps, but due to a loss, it reduces the cwnd as it is not using any bandwidth estimation technique to set the cwnd and ssthresh after the event of a loss. Hence, its throughput gets reduced after the error and again increases the cwnd which increases the throughput. As obvious from Figure 3.8, in the presence of errors, Tarang gives the same performance as the no loss scenario (i.e., 0 BER).

As Tarang uses the residual pipe capacity rather than minimum link capacity (i.e., used in Westwood) to set the value of cwnd and ssthresh in case the utilized bandwidth is less than half of the actual capacity, it is able to increase the sending rate very fast. Compared to Hybla, Tarang was indirectly deciding that the loss was due to the link error. Hence, it was also not reducing cwnd and resulted in the same performance as the scenario of BER 0.

At BER of 10^{-7}, Westwood followed the standard approach in SS and CA phase,

and it was using only utilized bandwidth to set the cwnd. Thus, in this scenario, as BER was high, that triggered more losses and cwnd remains low which in turn keeps utilized bandwidth low. Westwood sets the ssthresh and cwnd based on the utilized bandwidth after the losses. As the value of cwnd decides the amount of data the sender can send, which in turn decides the throughput. Hence, throughput of the Westwood remains around 10 Mbps for 100 Mbps link at 600 seconds, which indicated link utilization of 10% only. Hybla keeps reducing cwnd and ssthresh on the event of loss and hence it can only utilize 12 Mbps out of 100 Mbps link.

Tarang was able to increase the cwnd and ssthresh even after the losses. It means that Tarang was able to deduce that loss was not the indication of the congestion but,the loss was due to the link error. Tarang uses the residual capacity of the link to set the cwnd and ssthresh when data sending rate was low and used the utilized bandwidth when data sending rate was high. In this way, Tarang was able to achieve the throughput as high as the capacity of the link in the presence of high BER and high RTT.

For BER of 10^{-6} and 10^{-5}, Westwood was giving the throughput of less than 0.5 Mbps for the link of 100 Mbps. The reason for such a poor performance was that due to long RTT, it cannot increase cwnd fast enough and due to high BER, after losses, it sets ssthresh and cwnd as per the estimated utilized bandwidth which was obviously very low. For the performance of Hybla the same reasons were applicable as discussed for BER of 10^{-7}. Hybla performed poorly compared to Tarang. The reason for such a poor performance was that it does not have the sense to decide the reason for the loss. Hence, it keeps reducing cwnd and ssthresh blindly in the presence of link losses.

Tarang was able to operate at the peak of the link capacity even for BER of 10^{-6} as shown in Figure 3.8. It reached the peak value at around 200 seconds whereas the performance of Westwood is less than 0.1 Mbps and Hybla operates around 2 Mbps. The performance of Tarang remains significantly higher compared to Westwood and existing Hybla approach. Another observation from Figure 3.8 was that for BER of 10^{-5}, Tarang starts operating around 90 Mbps after 200 seconds. The reason for such behavior was that due to the initial losses and timeout, cwnd remains low compared to BER of 10^{-6} or less. Thus, lower value of cwnd resulted in lower data sending rate

and in turns the throughput was low.

3.4.2.2 Multi-hop satellite link

In these experiments, RTT varied in range 100, 200, 300, 400, 500, 600, 1000, 1500 ms for a link with BER of 10^{-8} (to simulate a link with varying RTT and average BER). Lower values of RTT (i.e., 100 to 500 ms, although it never occurs in GEO multi-hop satellite environment) were considered to show the performance with varying RTT.

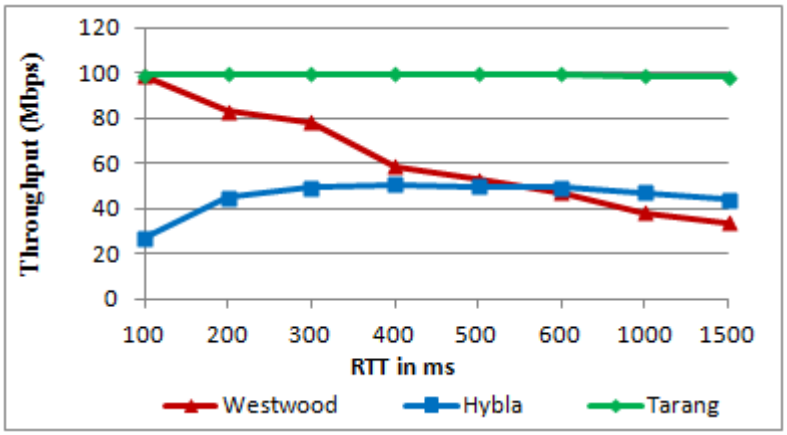


Figure 3.9: Performance comparison for varying RTT with BER of 10^{-8}

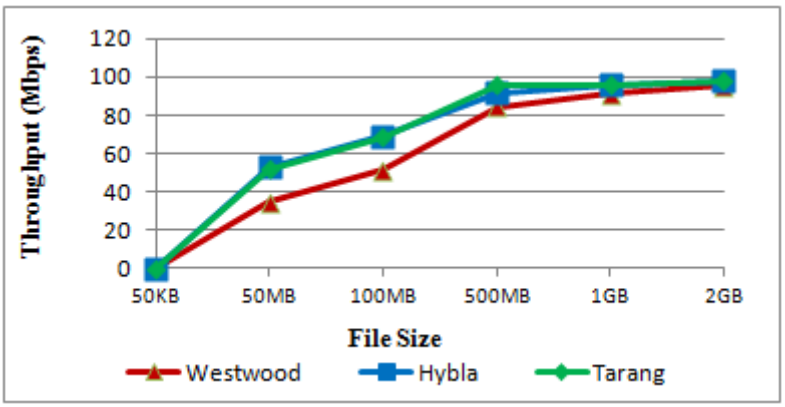
As shown in Figure 3.9, Tarang achieved throughput of higher than 90 Mbps, whereas throughput of Hybla and Westwood remained less than 50 Mbps at RTT of 600 ms and higher. Results show that performance of Tarang was not affected by the change in RTT at BER of 10^{-8} while Hybla operates around 50 Mbps. Westwood's performance decreased as RTT was increased.

3.4.2.3 Various File sizes transmission on satellite link

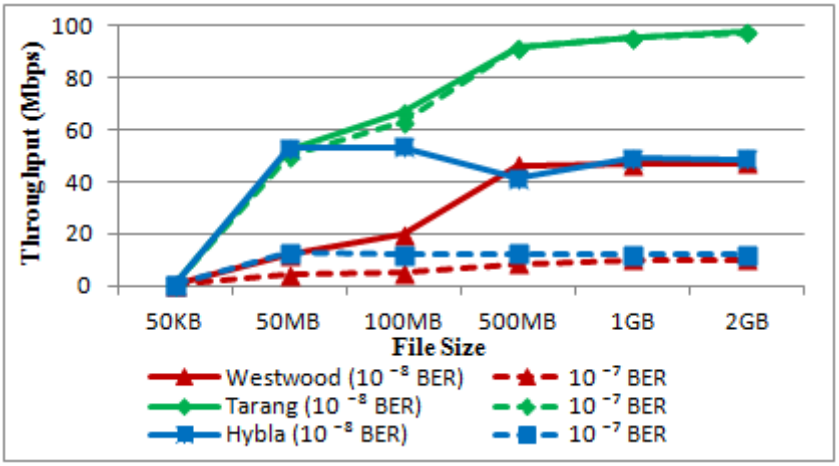
Files of sizes in range 50KB, 50MB, 100MB, 500MB, 1GB, 2GB were transmitted for a link with RTT of 600 ms and BER of 0 (i.e. negligible), 10^{-8} (i.e. average) and 10^{-7} (i.e. high). As shown in Figure 3.10-(a), for negligible BER, the performance of Hybla and Tarang were almost the same. Westwood's performance was low as the value of RTT was 600 ms. As file size was increased, performance improves as expected in case of no loss scenario. Westwood operates at the peak of the bandwidth for 2 GB file while Hybla and Tarang start operating near the peak of bandwidth for the file size of 500 MB.

Performance for average and high BER with varying file sizes has been shown in

Figure 3.10-(b). Tarang's performance remains the same for average and higher values of BER (i.e., higher than 90 Mbps) whereas the performance of Hybla and Westwood remains below 50 Mbps for average BER. Performance of Hybla and Westwood remain very low for higher values of BER.



(a) negligible BER



(b) average and high BER

Figure 3.10: Performance comparison with varying File sizes and BER

As shown in Figure 3.10-(b), for the link with BER of 10^{-8}, Westwood was able to achieve the throughput of 50 Mbps which proves that in the presence of high RTT and high BER, it cannot utilize the available bandwidth fully. Hybla operated at 53 Mbps for 100 MB file but for 500 MB file transmission, its throughput was 41 Mbps. The reason for reduced performance was that as data transmission size increases, more errors would be there and in turn on each error, Hybla keeps reducing cwnd. The same was true for 1 GB and 2 GB file transmission but, in this case Hybla took more time to transmit and was able to increase cwnd significantly during the

transmission time, which resulted in throughput of around 50 Mbps. Tarang gave the same performance as in a case of a link with BER of 0 (i.e., negligible BER).

As shown in Figure 3.10-(b), for BER of 10^{-7}, Westwood can achieve maximum throughput of 10 Mbps for 2 GB file and so transmission time was very high. Hybla cannot perform well even for the file size of 100 MB and throughput was only 11 Mbps. Thus, for BER of 10^{-7} and file size of 1 GB and 2 GB, transmission time was significantly high. The reason for such a poor performance was its inability to withstand the high BER. For the file size of 500 MB or more, Tarang was able to achieve the throughput of 90% and more. When more amounts of data are sent, Tarang was able to exploit the available bandwidth even in the presence of significant bit error rate (as high as 10^{-7}).

Hence, from the Figure 3.10, it is very clear that Hybla can perform well with high RTT but, performs poorly as BER was increased and Westwood performs better in the presence of losses but, performs poorly for the link with high RTT. Thus, Tarang can be a very useful protocol for the links like satellite where RTT was also very high and BER was moderate to very high and continuously varying.

3.4.2.4 Heterogeneous Networks- Dumbbell Topology

In this simulation scenario, three senders (S1, S2, S3) were communicating with corresponding receivers (D1, D2, D3) via a common bottleneck link of 1 Gbps. The simulation was carried out using different variants like Cubic (i.e., default variant on Linux), Hybla (i.e., designed for high RTT link like satellite), Westwood (i.e. designed for high BER link like Wi-Fi), and Tarang on all three senders. All three senders experienced different RTT (in range of 45 to 600 ms) and available link bandwidth (varies from very low value to 1000 Mbps, due to common bottleneck, different RTT and BER (varies from 10^{-6} to 10^{-11})) to simulate heterogeneous network consisting of wired link, wireless link, and satellite link. The topology used for the simulation has been shown in Figure 3.11. In this simulation topology, all senders communicated through following types of links: i) High-speed, Loss Less (HLL) ii) High-speed With Losses (HWL) and iii) High-speed, High RTT with variable Losses (HRL).

(a) High BER on HRL: S1-D1 is communicated via a link of 45 ms RTT and negligible BER to simulate data communication on a wired link. S2-D2 is communicated via a link with BER of 10^{-9} and RTT of 70 ms to simulate a Wi-Fi link. S3-D3

is communicated via a link with varying BER of 10^{-6}, 10^{-7} and RTT of 600 ms to simulate a satellite link (i.e., high BER can be due to bad weather).

(b) Average BER on HRL: S1-D1 and S2-D2 have communicated via links as described above. S3-D3 has communicated via satellite link with varying BER of 10^{-8}, 10^{-9} respectively in each set of experiments. Varying values of BER indicate the changes in weather conditions of the satellite link.

(c) Low BER on HRL: In all these experiments, S3-D3 has communicated via satellite link with varying BER of 10^{-10}, 10^{-11}. Low variation values of BER indicate the clear weather conditions for the satellite link.

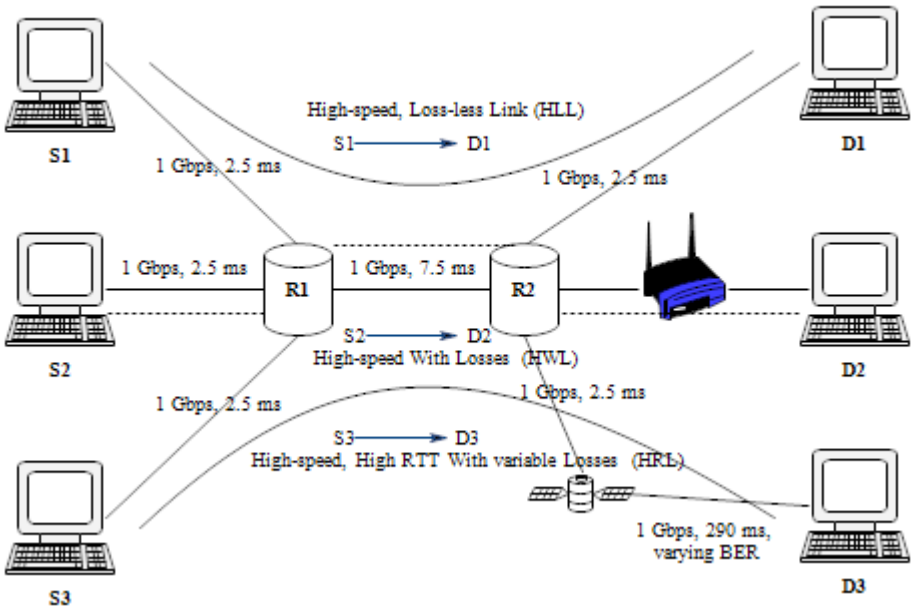


Figure 3.11: Heterogeneous network- dumbbell topology

As shown in Figure 3.12, in all above experiments, varying conditions of a link (e.g., a satellite link) affect the performance of the corresponding sender's TCP variant, and other cross traffic sender's TCP variant passing through a bottleneck link. In all these experiments, three flows were competing through a bottleneck link to maximize the link utilization for their respective flows. Cubic's performance was very poor on challenging links like HRL and HWL, and 90% of the bottleneck bandwidth was utilized by a link with 45 ms RTT and negligible BER. Westwood was able to exploit the bottleneck bandwidth for links HLL and HWL, as the design principle of Westwood was matching with these links characteristics. The performance of Hybla

for links with high BER was significantly low (i.e., links HWL and HRL). Tarang was able to share the bottleneck bandwidth almost fairly in heterogeneous networks.

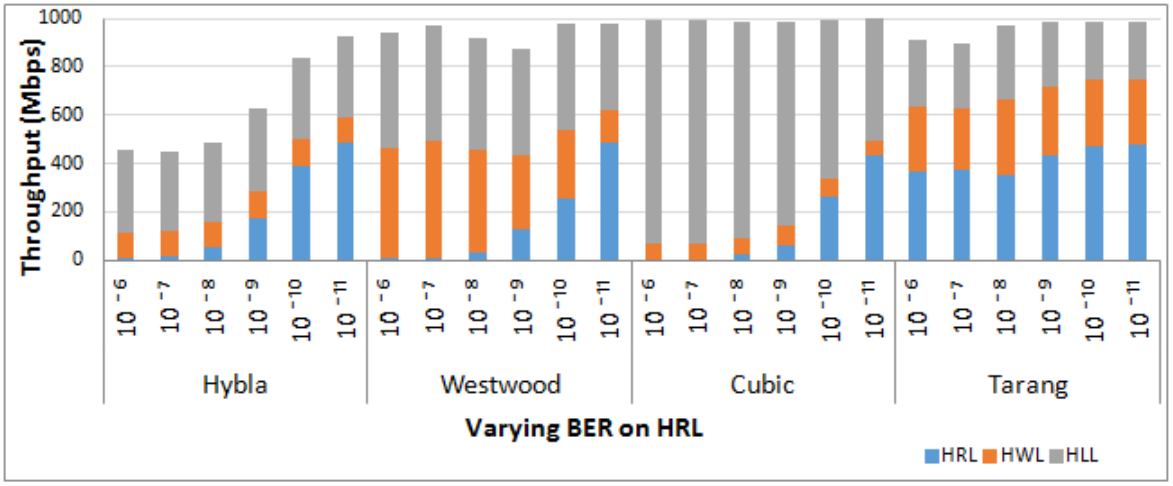


Figure 3.12: Performance comparison for varying cross traffic characteristics

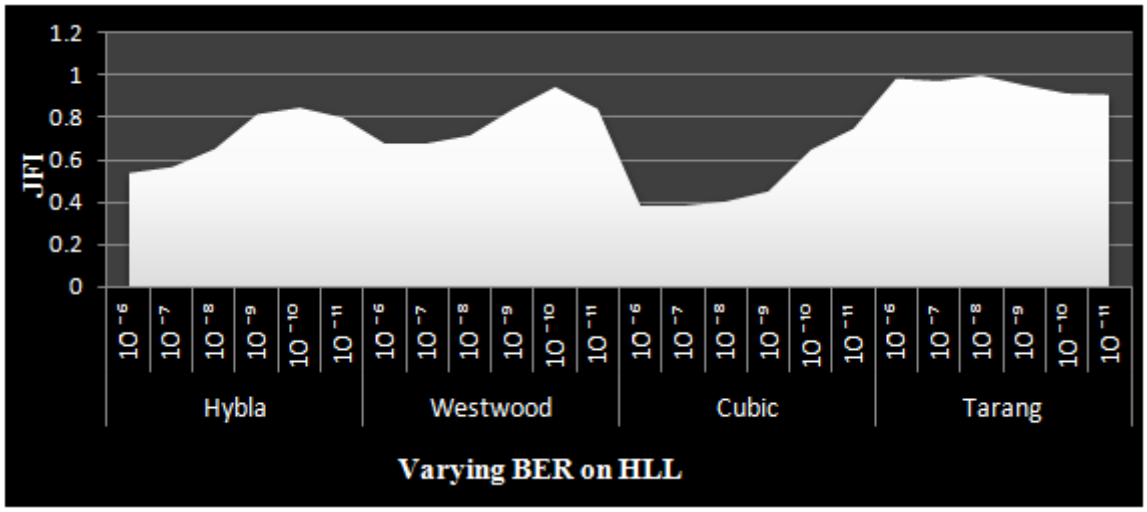


Figure 3.13: Jain's Fairness Index for varying cross traffic characteristics

To evaluate overall system fairness, Jain's Fairness Index (JFI) has been used which has been defined in Equation 3.3 (Jain, Durresi, and Babic). In this equation, b_n was the measured throughput of n^{th} flow and N was the total number of flows.

$$JFI = \frac{(\sum_{b_n})^2}{N \cdot \sum_{b_n{}^2}} \tag{3.3}$$

JFI comparison for different variants has been shown in Figure 3.13. Tarang's JFI was very near to one with very less fluctuations compared to other variants in-

cluding Cubic. Hence, Tarnag outperformed other variants for various links compared to existing variants including Cubic.

3.5 Summary

Tarang was able to offer the constant throughput with respect to varying link BER and RTT. It also exhibited fairness in heterogeneous networks. Results achieved have been summarized below:

- In an experiment, with PER of 1% and higher, varying RTT, and in the presence of background traffic, Tarang achieved 400% higher throughput compared to Westwood and Hybla.

- On a single-hop satellite link, with BER of 10^{-6} and higher, Tarang was able to exploit more than 80% of the bottleneck bandwidth whereas Hybla and Westwood could exploit only 2% and 0.5 % respectively.

- In a worst-case scenario with BER of 10^{-6} and RTT of 1500 ms (i.e., a multi-hop satellite link), Tarang improved the performance by 20-30 times compared to other existing variants.

- In heterogeneous networks, Tarang exhibited the fairness by maintaining JFI near to one whereas variants like Cubic, Hybla, and Westwood have the fairness issues.

- Tarang was able to exploit the available bandwidth on the link with variable BER, and RTT while background traffic was on low RTT, lossless links. Cubic showed 90% link utilization on low RTT, lossless link and competing flows got only 10% share of the bottleneck bandwidth.

Thus, Tarang outperformed in most of the cases, and it would be the default variant. Still, there may be a new variant in future, which may give better performance for the specific cases compared to Tarang. Furthermore, usually, fairness becomes a bottleneck in utilizing a specific costly link more effectively. Existing TCP/IP implementations force operating systems to fix and use single and fixed TCP variant for all applications and links. This kind of rigid binding of TCP variant with host oper-

ating system results in poor performance with emerging applications and upcoming networking technologies.

Hence, there is a need to modify TCP/IP implementations of modern operating systems to allow the usage of all existing and futuristic TCP variants for the effective link utilization and better end-user experience that has been addressed in the next chapter.

Chapter 4

Adaptive and Dynamic TCP Interface Architecture (ADYTIA)

In the TCP/IP implementations (Comer), all existing and upcoming applications based on TCP use the TCP variants depending on the operating system of the sender machine (e.g., FTP uses Cubic on Linux kernel 2.6.19, while FTP uses NewReno or Compound-TCP on Microsoft windows platform)(Yang et al.). Each application layer protocol has different kinds of expectations from the underlying transport layer protocol. Standard TCP like NewReno was designed to fully utilize the available link bandwidth for the wired networks. However, heterogeneous networks like the Internet, consists of wireless links in addition to wired links. Further, as proved in the previous chapter, Tarang outperformed the variants like Cubic and NewReno, but still, there could be scenarios where futuristic TCP variant may perform better. Each operating system has the different TCP variants to be used as a default variant and it has been fixed for all associated applications and underlying network technologies. Default and single TCP variant cannot utilize the full capacity of the available link for different combinations of application types and network links (Barakat, Altman, and Dabbous Papadimitriou et al. Pirovano and Garcia Yin et al.). In present chapter, an Adaptive and Dynamic TCP Interface Architecture has been proposed to resolve the issue of default and fixed TCP variant usage for different applications and network technologies. This modified TCP/IP implementations use the application types and network link characteristics to select the best possible TCP variant to maximize the link utilization. The proposed modifications were implemented in Linux kernel and

tested on ns-2.

This chapter has been structured as follows: Section 4.1 explains the related work in brief and the need for the required mechanism. Section 4.2 introduces the design of proposed modifications with all the components in details. It also presents the implementation strategies. Section 4.3 explains the experiments and analyses the results obtained. Finally, Section 4.4 summarizes the chapter with directions for the way forward.

4.1 Related Work

As elaborated in Chapter 2, the following challenges affect the performance of the transport layer protocols (i.e., especially TCP based): i) Each host has different variants of TCP based on the operating system ii) Each application that depends on TCP and originated from a host uses fixed TCP variant based on the operating system iii) Communication flow between each pair of communicating devices on the Internet needs to pass through different networking technologies which have a lot of variations in their characteristics (i.e. bandwidth, RTT, BER, queue types, mobility, etc.).

Above all, as shown by mathematical analysis in Chapter 2, the following parameters affect the performance of the TCP significantly: i) RTT ii) Sender window size iii) Link error rate (i.e. BER). These parameters have a strong impact on the variation of ssthersh and cwnd, that is responsible for effective utilization of the link. In heterogeneous networks, TCP implementations have to face many challenges because of the different link characteristics and network dynamics. Following are the parameters that affect the TCP's performance significantly, due to underlying technologies and link's dynamics.

- **Bottleneck Link Available Bandwidth:** It is the link along the path from sender to receiver with minimum available capacity (i.e., bandwidth). Due to the cross and bursty traffic, and increase and decrease in the number of communication flows through the bottleneck link, the available link capacity (i.e., Bottleneck link bandwidth divided among the number of communication flows) keep changing for each flow. This is the value that each TCP variant tries to estimate and tries to operate at that estimated value. The goal of each

TCP variant is to utilize the maximum available link capacity. The variant like NewReno (Henderson et al.) probe the link capacity using AIMD algorithm, while variant like TCP Westwood (Saverio et al.) uses the estimated bandwidth. The values of cwnd and ssthresh are set based on different approaches by each TCP variant to maximize the link utilization (Abdeljaouad et al.).

- **RTT:** The RTT comprises of two-way propagation delay (i.e., sender to receiver and vice versa), queue delay and processing delay on each node (Sun). RTT decides the speed of the flow of acknowledgments. RTT may change due to the heavy traffic on any intermediate node, retransmission at data link layer, or change in the topology and route. Smoothed average of RTT and jitter (i.e., variance in RTT) are used to set the value of rto (i.e., retransmission time out) that decides the waiting time before retransmission. Hence, RTT is the parameter which decides, how fast the value of cwnd can be increased and decreased.

- **BDP:** The value of BDP is $Link - Capacity * RTT$, which determines the minimum value of cwnd in order to utilize the link fully. Thus, the variations in BDP for different kinds of links affect the performance significantly (Lakshman and Madhow).

- **BER or PER:** It is the number of bits in error divided by the total number of transferred bits during a studied time interval. Some tools allow only to specify the PER (Packet Error Rate), in that case, $PER = 1 - (1 - BER)^K$, where K is the segment size in bits. BER affects the performance of TCP significantly as loss of segment is always treated as a sign of congestion by the TCP flow (Lakshman and Madhow).

- **Asymmetric Link:** There are links or paths between sender and receiver with different link characteristics, or there could be less available link capacity in one direction (i.e., may be due to cross traffic or configured intentionally to optimize the resources). Such an asymmetric nature may affect the rate at which acknowledgments are received that decides the rate at which sender can send the data (Armitage).

Variations in the values of above parameters and different combinations of the above parameters result in the different kinds of links on which TCP needs to operate. TCP variants discussed in Chapter 2 have been tailored to perform well on specific kind of link. Different kinds of links and specific TCP variants designed to give optimum performance have been summarized in Table 4.1 (Caini, Firrincieli, and Lacamera Akyildiz, Morabito, and Palazzo West and Vaidya Abdeljaouad et al.).

4.1.1 TCP Implementations in present Operating Systems

The operating system has implemented the TCP/IP as a part of the kernel for effectiveness and robustness. Each operating system has followed a different approach for the implementations. TCP's congestion and flow control algorithms have been implemented as a part of the TCP/IP. In this subsection, implementations of TCP variants have been presented for Microsoft windows and Linux operating systems in brief.

4.1.1.1 Windows Operating System

TCP NewReno has been implemented as a standard TCP algorithm. In addition to this, Compound TCP has been implemented (e.g., in Windows Vista) as a part of the operating system. The expert user can change the TCP variant with external commands. All communication flows originating from the sender machine uses the default variant or as per the last command executed by the expert user. Hence, a machine with windows operating system can either use NewReno or Compound TCP as a way to do congestion control and flow control.

4.1.1.2 Linux

In Linux, congestion control has been completely modularized for maintenance, extension and for providing customized settings. Each TCP variant has been implemented using a framework similar to the backplane-slots framework. Linux kernel 2.6.26 onwards, most of the variants listed in Table 4.1 have been implemented as per the framework. The major benefit of such an implementation is that core functionalities are separated (in terms of slots) from the variant specific functionalities. Cubic has been the default variant used by the latest kernel, although it supports multiple variants designed for the specific link. Still, the limitation has been that at a time all

Table 4.1: Link types and suitable TCP variants

Link Type	**Link Characteristics**	**Example of the Link**	**Specifically Designed TCP Variants**
Low Speed Very Low Delay	Link with very low- RTT and bandwidth, Low RTT link with cross traffic	Small distance wired link, dial-up connection.	Reno, NewReno, NewReno with SACK
Low Speed-Low Delay	Low bandwidth, low RTT link and low RTT link with cross traffic	Leased line connection, dial-up connection.	NewReno with SACK.
Low Speed-Low Delay Link with Losses	Low bandwidth, low RTT link with losses	Wi-Fi, 3G, 4G, WiMAX.	Westwood
High Speed Connection	Low to moderate RTT and high bandwidth	Optical Fiber	HighSpeed, BIC, Cubic, Compound, Scalable, Yeah
High Speed Connection with Losses	Low to moderate RTT, high bandwidth and with link losses	Wi-Fi link	Westwood, Cubic, Yeah
High Speed Connection with Large Queue	Low to moderate RTT and high bandwidth with large buffer on intermediate router	Optical Fiber with specific router on the path.	TCP-Sync
Long Fat Connection	Link with very High RTT and moderate to high bandwidth	Large distance Optical Fiber (e.g. Ocean Fiber)	HighSpeed, Cubic, Hybla, Compound, Scalable, Yeah
Long Fat Connection with Losses	Link with very high RTT, moderate to high bandwidth and link losses	Satellite Link	Hybla, Tarang
Long Thin Connection	Link with high RTT and low to moderate bandwidth	Large distance Optical Fiber link with cross traffic	Hybla, Cubic
Long Thin Connection with losses	Link with high RTT, low to moderate bandwidth and with losses	Satellite link, 2G, 3G	Hybla, Tarang

communication flows use the same variant (i.e. as specified using sysctl command). In this implementation, user or application has the flexibility to specify explicitly which variant has to be used but it is very difficult for the novice user or application designer to make that decision. At present, Linux kernel supports many TCP variants but at a time it can use only one, and the kernel is not using any parameters or intelligence approach for choosing the specific TCP variant.

4.1.2 Summary

As discussed above, modern operating systems support the multiple implementations of the TCP variants but at a time it can use only one, and an expert user can change as per his/her understanding. As discussed in Chapter 2, it is very important to select a specific TCP variant based on the link and application type to effectively utilize the link capacity.

4.2 Design and Implementation of ADYTIA

Existing TCP/IP implementations with the default TCP variant supported by the well- known platforms has been shown in Figure 4.1. The existing approach utilizes the specific variant based on the following:

<table>
<tr><td>Application Layer</td><td colspan="7">FTP, HTTP, TELNET, SMTP, User Application .....</td></tr>
<tr><td>Operating Systems</td><td>Free BSD</td><td>Oracle Solaris</td><td>Linux Kernel 2.6.8 to 2.6.18</td><td colspan="2">Linux Kernel 2.6.19 or later</td><td colspan="2">Windows XP/2000/Vista</td></tr>
<tr><td>Transport Layer Protocol</td><td>New Reno</td><td>New Reno</td><td>BIC</td><td colspan="2">Cubic</td><td colspan="2">New Reno/ Compound TCP</td></tr>
<tr><td>Network Layer</td><td colspan="7">IP</td></tr>
<tr><td>Data Link Layer</td><td>Wi-Fi</td><td>WiMAX</td><td>Satellite</td><td>4G</td><td>5G</td><td>Ethernet</td><td>2G,3G</td></tr>
<tr><td>Physical Layer</td><td colspan="7"></td></tr>
</table>

Figure 4.1: Existing TCP/IP implementations

- Each application layer protocol has been mapped to fix transport layer flavor (e.g., FTP always uses default TCP variant of the host machine operating system).

- The sender uses the default TCP variant depending on its operating system, irrespective of the underlying networking technologies and link types.

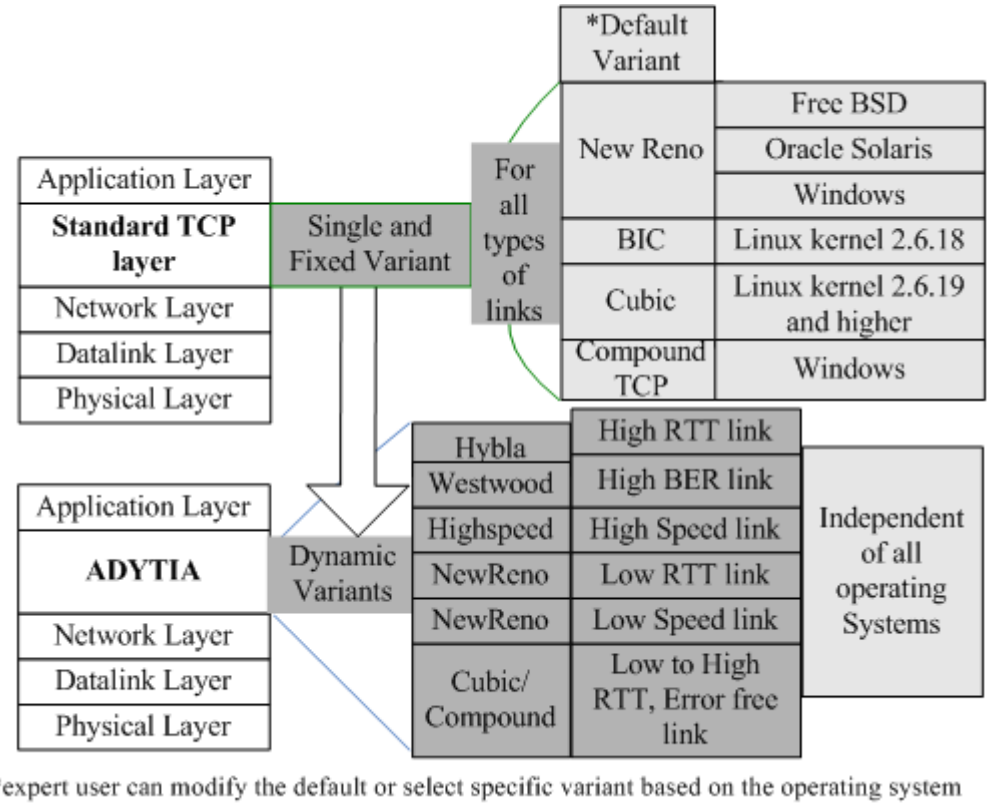


Figure 4.2: Proposed changes in TCP layer

This "blind" selection of the default variant for heterogeneous networks results in poor performance. In order to improve the performance, it is necessary to consider the application layer protocol and network link characteristics. To resolve the transport layer challenges and issues as discussed in Chapter 2, Adaptive and Dynamic TCP Interface Architecture (ADYTIA) has been designed and implemented.

The proposed changes in TCP layer (i.e., Transport layer) and its consequences in comparison with existing TCP/IP implementations in operating systems have been shown in Figure 4.2. ADYTIA has been designed to classify the network links and application types. All modules of ADYTIA and their connectivity to other modules have been shown in Figure 4.3. ADYTIA's design allows the plugging of new networking technologies, applications, and futuristic TCP variants without affecting the modules of ADYTIA and code rework. ADYTIA consists of following four modules:

- Connection Classification Module (CCM)

- Application Classification Module (ACM)
- Adaptive Module (AM)
- Information Base

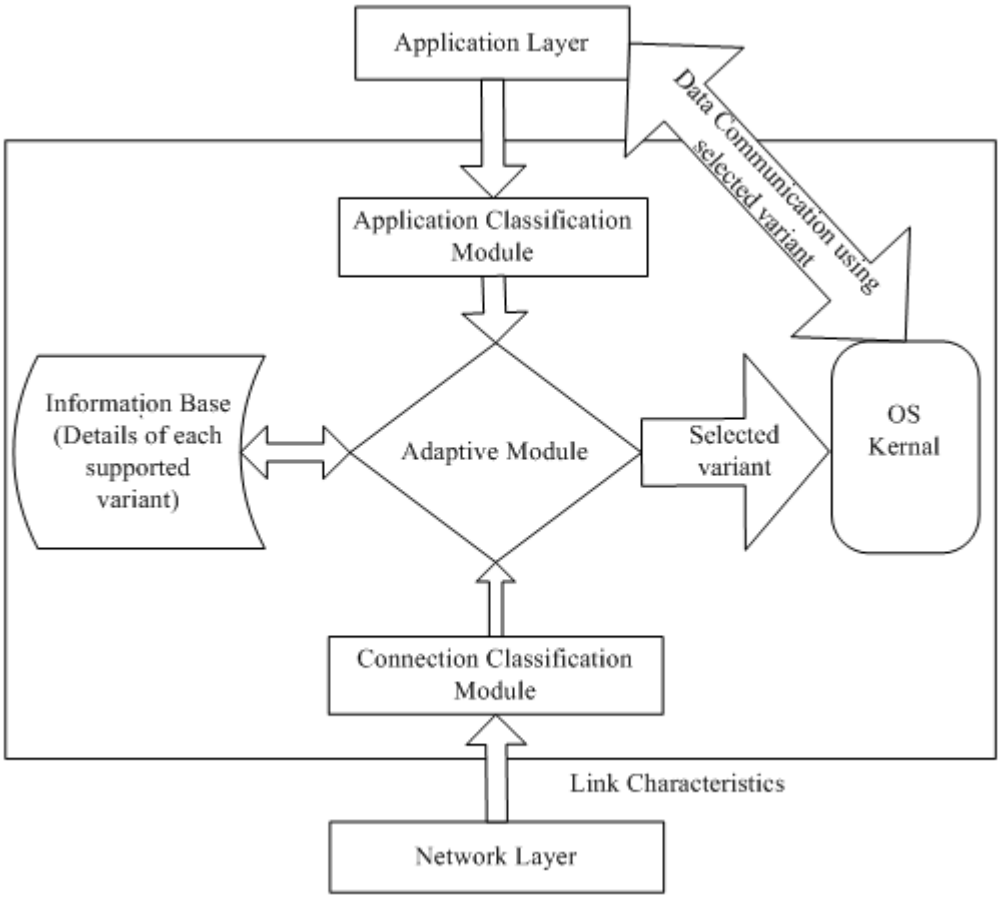


Figure 4.3: Design of ADYTIA

4.2.1 Modules and Algorithms

As shown in Figure 4.3, ADYTIA's design is based on the four modules and its communication with the kernel of the operating system. This subsection elaborates the design of each module of ADYTIA.

4.2.1.1 Connection Classification Module (CCM)

This module continuously observes the network link characteristics by observing the following four parameters and based on these parameters; CCM classifies the connection:

- **Bandwidth:** Bandwidth Delay Product (BDP) estimation technique which is based on ACK dispersion has been used to calculate the bandwidth. BDP estimation technique has been used to examine the capacity of connection used by the specific flow. Chain of the packets is sent together even if packet pacing has been enabled. Dispersion of corresponding acknowledgments is calculated and BDP is calculated using the following equation:

$$BDP = \frac{MinRTT \times \frac{TrainLength}{2}}{Dispersion(pkts)} \tag{4.1}$$

Where, MinRTT is the minimum ever seen value for the specific flow.

Table 4.2: Connection classification logic

Measured Parameter	**RTT (in ms)**	**Bandwidth (Mbps)**	**Queue delay detected**	**Packet loss detected**
Connection Type				
Low Speed-Very Low Delay link	<=100	<=100	No	No
Low Speed-Low Delay link	<=200	<=100	No	No
Low Speed- Low Delay link with Losses	<=200	<=100	No	Yes
High Speed Connection	<=200	>100	No	No
High Speed Connection with Losses	<=200	>100	No	Yes
High Speed with Large Queue	<=200	>100	Yes	No
Long Fat Connection	>200	>=10	No	No
Long Fat Connection with Losses	>200	>=10	No	Yes
Long Thin Connection	>200	<10	No	No
Long Thin Connection with Losses	>200	<10	No	Yes

- **RTT:** The value of RTT is estimated using Time Stamp option. The flow of the acknowledgments has been used to estimate the RTT for the each data segments transmitted. The instantaneous change in RTT has been smoothed using the proportional average of previously measured values and current RTT.

- **Queue Delay:** It is the difference between the average Smoothed Round Trip Time (SRTT) and Base Round Trip Time (BRTT). It is an indirect measure to see the number of competing flows through the bottleneck. The large queue delay may indicate the congestion in the networks.

- **Loss Discrimination:** A simple heuristic has been used to discriminate the reason of segment loss. The loss could be either due to congestion or because of transmission error (Xiuchao et al.). If queue delay is zero and there is a loss, it is an indication of loss due to the transmission error. Other cases of loss would be treated as a sign of congestion.

The logic used to classify the connections depending on the four parameters discussed above has been presented in Table 4.2. The logic of classifications is accessed through a file at run time, that facilitates making the changes in logic without the need for modifying and recompiling the kernel. Connection types defined in Table 4.2 are capable of accommodating any networking technologies as well futuristic networking technologies. Connection Classification Algorithm(CCA) has been used to classify the connection and has been implemented as a part of CCM module which has been shown in Algorithm 7. The connection type is communicated to the AM of ADYTIA, to select the specific TCP variant, based on the link types as perceived by the transport layer rather than actual underlying networking technologies and physical link.

4.2.1.2 Application Classification Module (ACM)

This module has been used to classify the applications based on the port number used by the standard application layer protocol (Wang, Mohapatra, and Mukherjee Sun). It can also use the other characteristics (e.g., L7 filter uses the regular expression to decided the protocol type independent of the port number) to differentiate the application expectations (e.g., FTP always expect to maximize the throughput) (Levandoski, Sommer, and Strait). Application Classification Algorithm (ACA) has been implemented as a part of the ACM, and it has been listed in Algorithm 8.

Algorithm 7 Connection classification algorithm

1: Initialization
2: **/*Estimate RTT */**
3: Sampling and smoothing of RTT sample
4: **if** $lastRTT > minRTT$ **then**
5: $minRTT \leftarrow lastRTT$
6: **end if**
7: **/*Estimate bandwidth */**
8: send burst of four packets
9: **if** $minRTT > 0$ **then**
10: $BDP \leftarrow minRTT * (train\_length/2)/dispersion(pkts)$
11: $bandwidth \leftarrow BDP/minRTT$
12: **end if**
13: **/*Estimate queue delay */**
14: **if** $minRTT > 0$ and $dispersion(pkts) > 0$ **then**
15: $queue\_delay \leftarrow DIFF(sRTT, baseRTT)$
16: $sRTT \leftarrow (7/8) * sRTT + (1/8) * current\_RTT$
17: **end if**
18: **/*Estimate the reason of the loss */**
19: **if** $CA\_State = TCP\_CA\_Recovery$ and $queue\_delay = 0$ **then**
20: Link Losses
21: **else if** $CA\_State = TCP\_CA\_Loss$ **then**
22: Link Losses
23: **else**
24: Congestion Losses
25: **end if**
26: **Repeat:**
27: Until End of Connection Classification File
28: $Connection\_Type \leftarrow search\_CCL\_File(loss\_reason, minRTT, queue\_delay, bandwidth)$
29: **end:**

4.2.1.3 Information Base

This module has been used to trigger the specific TCP variant depending on the application type and link characteristics. Information Base has been prepared after rigorous literature survey of the already existing TCP variants as detailed in Chapter 2. This Information Base can be modified, or other variants can be added in future without disturbing the other modules of the ADYTIA. The Information Base used by the ADYTIA has been presented in Table 4.3.

Table 4.3: Information Base of the supported TCP variants

Applications	**Email**	**Web Browsing**	**Real Time**	**Telnet**	**FTP**
Connection Type	**TCP variant to be used**				
Low Speed-Very Low Delay link	Default	Default	HighSpeed	Default	Default
Low Speed-Low Delay link	Default	Default	HighSpeed	Default	Default
Low Speed-Low Delay link with Losses	Westwood	Westwood	Yeah	Westwood	Westwood
High Speed Connection	HighSpeed	HighSpeed	Scalable	HighSpeed	HighSpeed
High Speed Connection with Losses	Westwood	Westwood	Yeah	Hybla-Tarang	Westwood
High Speed with Large Queue	Sync	Sync	Sync	Sync	Sync
Long Fat Connection	Hybla	Hybla	Hybla	Hybla	Hybla
Long Fat Connection with Losses	Hybla	Tarang	Tarang	Tarang	Tarang
Long Thin Connection	Hybla	Hybla	Hybla	Hybla	Hybla
Long Thin Connection with Losses	Tarang	Tarang	Tarang	Tarang	Tarang

Algorithm 8 Application classification algorithm

1: **for each** Connection **do**
2: $isk\_dport \leftarrow inet\_sock$
3: **if** $isk\_dport = port\_no$ **then**
4: return $application\_type$
5: **else**
6: continue
7: **end if**
8: **end for**

4.2.1.4 Adaptive Module (AM)

This module is the centralized entity of ADYTIA. It has been used to select the appropriate TCP variant by considering the input from CCM and ACM. Depending on the input values, it communicates with Information Base and selects the TCP variant to be used. It also interacts with the kernel of the operating system and commands it to use the selected variant. Algorithm 9 has been implemented as part of the Adaptive Module.

Algorithm 9 Adaptive module algorithm

Input: connection_type, application_type

Output: Selection of TCP variant from Information Base

```
1: while Until end of the Information Base do
2:     if Match is Found then
3:         return TCP_Variant
4:     else
5:         continue
6:     end if
7: end while
```

Algorithm 10 ADYTIA's algorithm

Input: Initiation of the transport layer connection

Output: Selection of congestion control algorithm

```
1: for each Connection  do
2:     Selected_Variant ← Cubic; /* default protocol of the platform */
3:     Connection_Type ← CCA(bandwidth, RTT, queue_delay, PER);
4:     Application_Type ← ACA(dport);
5:     Selected_Variant ← AMA(Connection_Type, Application_Type);
6:     congestion_control_alg ← Selected_Variant;
7: end for
```

ADYTIA's algorithm, as illustrated in Algorithm 10, has been used to select the specific TCP variant at the starting of the communication flow. It calls the appropriate modules and continuously triggers the changes in TCP variant in case

of changes in the link characteristics. In a best-case, the time complexity of the algorithm is $O(1)$. In a worst-case the time complexity is given by $O(RTT + C * P + C * A)$. Here C is the types of the links in the connection classification logic, P is the number of parameters used to classify the links, and A is the application types considered in ACM. As the parameters to classify the connections were increased, the worst-case complexity gets increased. Similarly, as more applications are classified by ACM, a worst-case complexity increases.

4.2.2 Challenges in Simulations

In the simulation model, the protocols have been implemented using the abstraction of the behavior of the protocols. As a result, the performance of the simulation model significantly differs with the real networks implementations. This characteristic is very important as it gives scalability, controllability, flexibility and the easiest way for the implementation and evaluation in the first phase of the development. However, for credible and successful deployments in the production environments, protocols need to be tested on the real networks with kernel level implementations. While simulation environment makes it easy to have the first cycle of the development and feedback for newly designed approach or protocol, it abstracts the low-level requirements of the real system (e.g., buffer space, memory requirements, processing delays and other systems overhead). In a nutshell, there are following challenges that need to be addressed: i) The implementation has to be as close as possible to the kernel or system level implementation ii) Simulation results need to be confirmed with the testbed experiments on the real systems.

4.2.3 Implementation Strategies

In this section, the objective was to complete the first cycle of development and performance analysis using simulation environments only. Hence, ADYTIA needs to be implemented on each node (i.e., sender nodes) for the performance analysis. The different mechanisms to analyze the TCP variants have been shown in Figure 4.4. Mathematical formulation gives the micro level insight into the protocol's behavior and parameters that affect the performance. To convert this mathematical approach into the real system protocol (i.e., a kernel level implementation) is a challenge and require lots of efforts and time. As indicated in Figure 4.4, a simulation model can

be developed faster and can be considered close to the real system implementations, but still, deployment on the real networks (e.g., like the Internet) may change the performance significantly. Another challenge in the simulation model is that the code needs to accommodate many changes (i.e., in some cases complete modifications) to migrate for the testbed experiments. Hence, one of the objectives of the research work presented in this thesis was to do the testbed experiments and deployment on the real networks like Internet. Following two subsections elaborate the traditional approach of implementation and the approach followed in this research work.

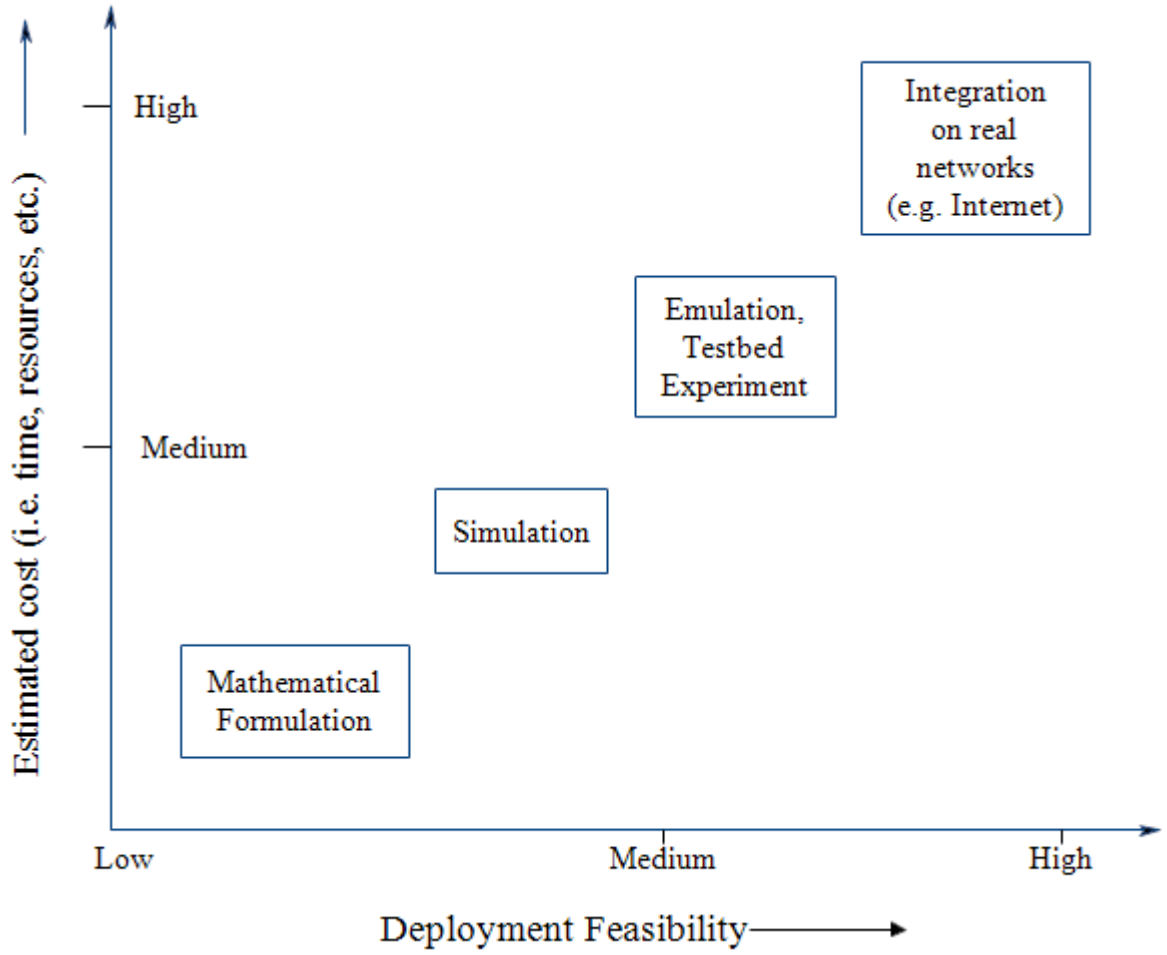


Figure 4.4: Mechanisms for performance analysis of TCP variants

4.2.3.1 Traditional Approach

The traditional way to implement ADYTIA or any newly designed TCP variants (e.g., Tarang as discussed in Chapter 3) using ns-2 has been illustrated in Figure 4.5. In this approach, the code needs to be written in 'C++' and later on for the testbed

experiments, require to be ported in 'C' for the kernel level implementations. In the latest ns-2 version (i.e., ns-2.35) all TCP variants are not available, and ADYTIA may select the variant which is not provided as a part of the standard ns-2. Thus, either all the TCP variants considered here needs to be written in 'C++' to be used in ns-2 or one needs to perform the simulation with only the available variants. To implement all TCP variants for ns-2 require a lot of efforts, and it is out of the scope of the proposed research work. However, if all TCP variants have been written in 'C++' for the simulation model and then migrated to kernel level implementations for testbed experiments, it does not guarantee the same performance, and it is hard to make a comparison between simulation and testbed results.

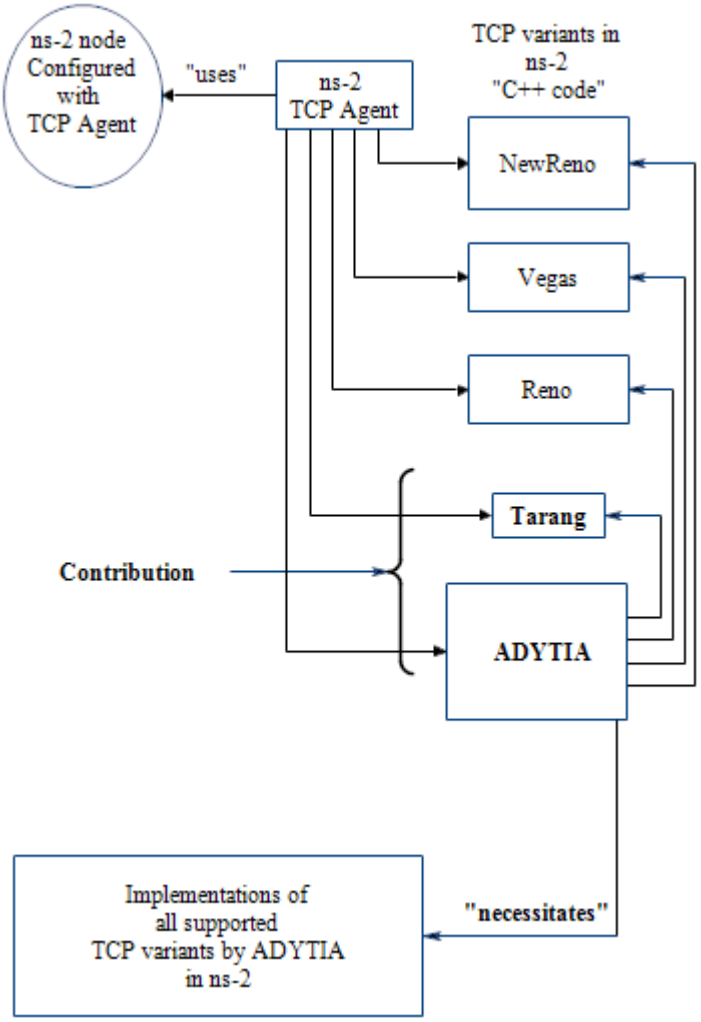


Figure 4.5: Traditional approach of protocol implementations in ns-2

In summary, the challenge has been that traditional approach may give the easiest way in terms of scalability and testing but, be difficult to implement and

migrate to the kernel of the operating system for the testbed experiments.

4.2.3.2 State of the Art

The state of the art approach for implementing and testing the TCP variants of Linux kernel in ns-2 has been shown in Figure 4.6. The advantages of this approach are: i) The first phase of the development cycle can be completed in ns-2 and migrations to Linux kernel require very less efforts ii) Performance could be compared because the same code has been used in simulation and testbed experiments iii) As per the need of ADYTIA, other TCP variants are available in Linux kernel and can be used for the comparisons.

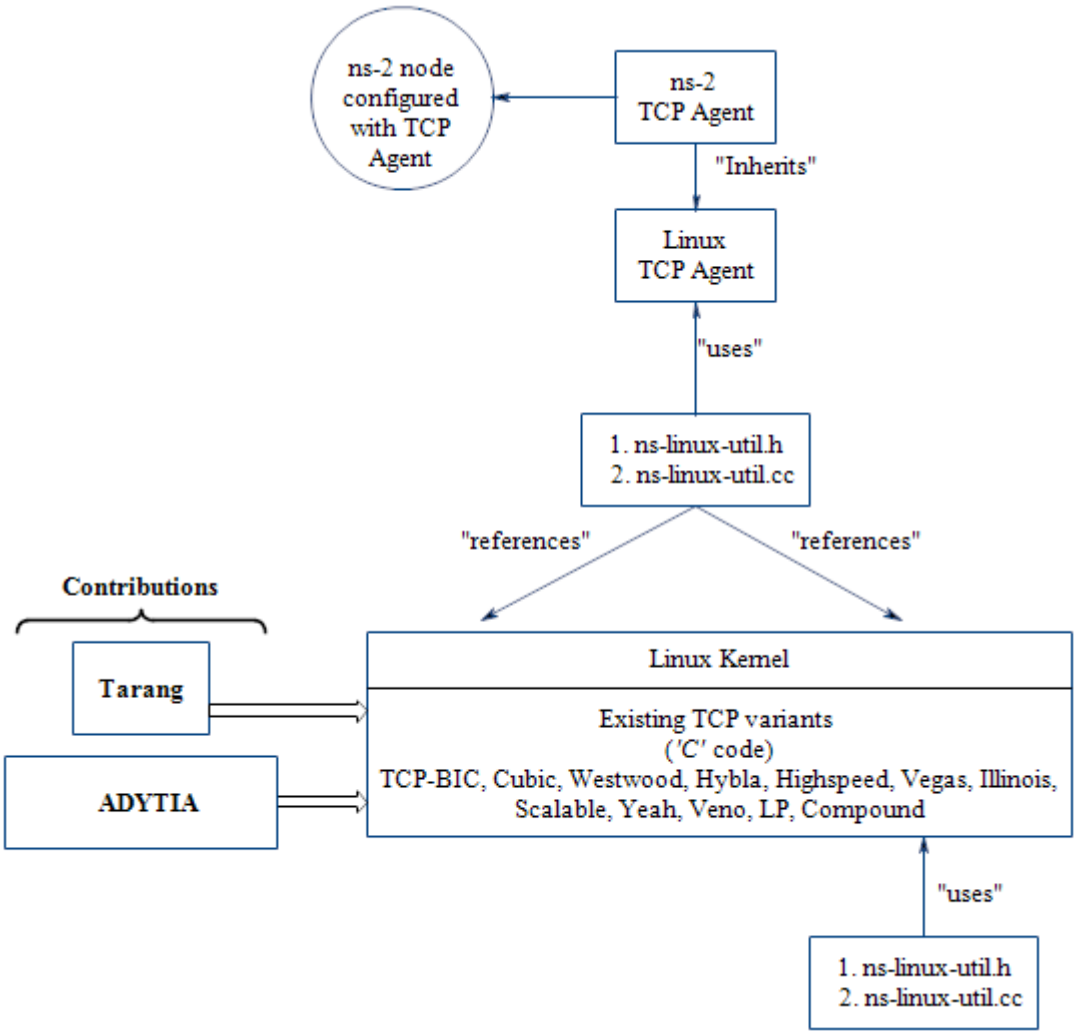


Figure 4.6: Approach used in research work for protocol implementation in ns-2

As shown in Figure 4.6, ADYTIA and Tarang have been implemented as the TCP variants. Through the various modules of ADYTIA, as per the link and appli-

cation types, appropriate TCP variant (i.e., Linux kernel implementation) have been used. When a new protocol (e.g., a TCP variant) has been designed (i.e., like Tarang) and available as a part of the Linux kernel, it can be accommodated in the Information Base of ADYTIA without modifying any module of ADYTIA. This plugging of newly designed TCP variants without any modifications is a very important feature of ADYTIA.

The result generation cycle used after the implementation has been shown in Figure 4.7. Analysis of trace files and predefined test cases help in improving the implementations of ADYTIA at micro levels. Simulation topology has been generated to trigger the different test cases and feedback of the results has been used to change the topology parameters for benchmarking and comparisons.

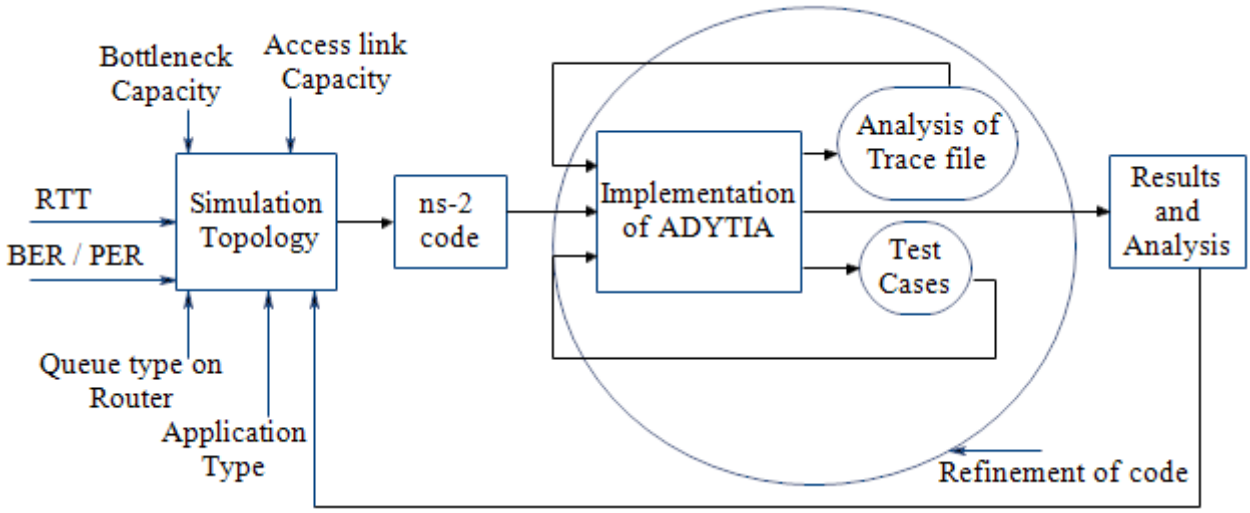


Figure 4.7: Result generation and code refinements

4.2.4 Dynamism

ADYTIA offers dynamism by selecting the variant based on link and application type rather than using the single and fixed variant. It also changes the variant depending on the changes in the link characteristics throughout the communication session.

4.2.4.1 Initial Selection of the Variant

The connection starts with the default variant based on the operating system of the sender machine. ADYTIA classifies the connection and application type using CCM and ACM module respectively as explained in Section 4.2.1. Information Base has

been shown in Table 4.3, which has been used to switch the variant from default to the specific variant based on link and application type to enhance the link utilization. For a user using FTP on a link with low RTT and high BER, Westwood has been used as per the Table 4.3, that results in better link utilization. Similarly, for a user using TELNET on a link with low bandwidth and low delay, no need to switch a variant (i.e., default variant Cubic on Linux and NewReno on Windows offer better link utilization).

4.2.4.2 Runtime Change of the Variant

During the lifetime of a communication flow, CCM of ADYTIA keeps observing the link parameters and link utilization. Depending on changes in the link characteristics, ADYTIA switches the variant to maximize the link utilization. Connection classification logic used by the CCM of ADYTIA has been shown in Table 4.2.

4.2.5 Adaptive

ADYTIA is adaptive in nature which facilitates the plugging of newly designed variants without modifying any modules of ADYTIA. It can do a self-learning to adapt the default variant based on the usage frequency.

4.2.5.1 Plugging of Tarang in ADYTIA

Information Base of ADYTIA uses a text file to read the name of the specific TCP variant to be used for a given connection and application type (as decided by CCM and ACM). For example, Tarang, as discussed in Chapter 3, has been designed for the link with variable BER and RTT. Hence, only one change is required to be made for the link having high RTT and BER by specifying a variant to be used is Tarang in place of Hybla. After plugging of Tarang, ADYTIA selects the Tarang rather than Hybla for all connections characterized with higher RTT and BER. Information Base presented in Table 4.3 illustrates the plug-in feature of ADYTIA, as Tarang is plugged-in without any code rework.

4.2.5.2 Self Learning

ADYTIA maintains the usage counter of the each supported variant based on the utilization. Once the value of the usage counter reaches a threshold value, it can be used as a default variant. Thus, default variant usage mechanism of standard TCP/IP

implementations has not been changed. When a newly designed variant is plugged into ADYTIA, an average usage count value of all supported variants is assigned to it. In a special case, a user can give the maximum value of usage count to make the newly plugged variant as a default variant.

4.2.6 Summary

As discussed in this section, it is very important to foresee at the challenges of the simulation model and decide the entire path of the development cycle to have ease and commendable migration towards the testbed implementations. State of the art approach used in the research work is very critical and very important decision to implement ADYTIA in the Linux kernel.

4.3 Experiments and Results

ADYTIA has been implemented in Linux kernel for simulation and testbed experiments. All senders were running the same TCP variant (e.g., All senders with Cubic) at a time and performance was measured in the presence of competing flows with different link characteristics. Furthermore, comparison and performance analysis were carried out by activating a specific variant on all senders. Simulation has been carried out on different topologies as discussed in the following subsections.

4.3.1 Heterogeneous Network- Dumbbell Topology

In this simulation scenario, three senders (S1, S2, S3) were communicating with corresponding receivers (D1, D2, D3) via a common bottleneck link of 1 Gbps. The topology used in this study (i.e., Dumbbell topology) is the well-known topology for the TCP variants performance analysis and has been used in similar studies (Ha et al.). Here, the aim was not to show the deployment scenario but to have the topology which allows one to know all the parameters which affect the performance. As discussed in (Floyd), dumbbell topology can be used to evaluate the performance of the transport layer protocols.

The simulation was carried out using different variants like Cubic (i.e., default variant on Linux), Hybla (i.e., designed for high RTT link like satellite), Westwood (i.e. designed for moderate to high BER link like Wi-Fi), and ADYTIA on all three senders. All three senders experienced different RTT (in range of 45 to 600 ms) and

available link bandwidth (varies from very low value to 1000 Mbps, due to common bottleneck, different RTT and BER (varies from 10^{-6} to 10^{-11})) to simulate heterogeneous network consisting of wired link, wireless link, and satellite link. The topology used for the simulation has been shown in Figure 4.8. (Miras, Bateman, and Bhatti),(Floyd). While measuring the performance on one link, other two links provided the background traffic and real network dynamics. Performance comparison of senders' with existing variants and ADYTIA has been shown in Figure 4.9. In this experiment, three senders were communicating with corresponding receivers via three types of links HLL, HWL, and HRL respectively.

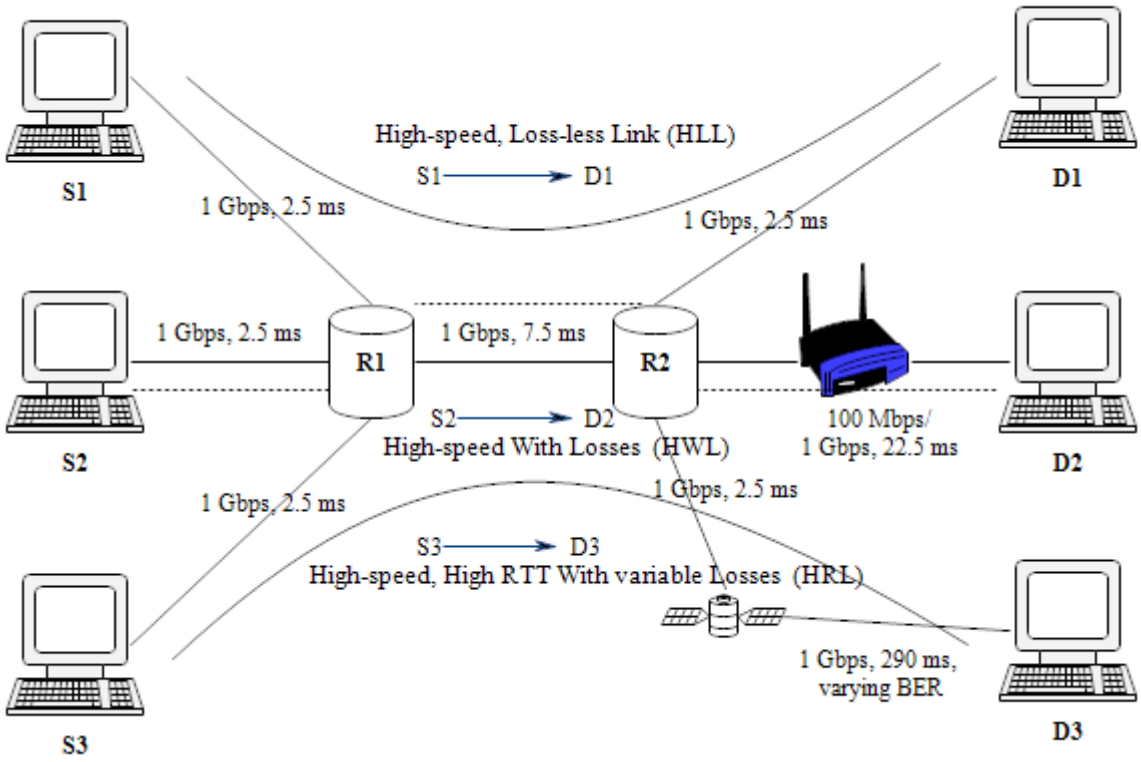


Figure 4.8: Heterogeneous network- dumbbell topology

As shown in Figure 4.9, sender with ADYTIA communicating via link HRL starts with Cubic (i.e., a default variant) and as it detects high RTT, it switches to Hybla to enhance the link utilization. Sender with Cubic (i.e., End-User with Linux), Hybla or Westwood (i.e., without ADYTIA) utilizes the fixed and single variant throughout the communication flow. A sender with ADYTIA, communicating via HWL link start with Cubic and then switches to Westwood which results in significant performance

improvement compared to the sender with Linux. ADYTIA's ability to select the most suitable TCP variant results in 158% performance improvement compared to the existing approach of the single and fixed variant usage. Throughput comparison of running Westwood, Hybla, Cubic, and ADYTIA on all three senders with varying BER of HRL link (i.e., a link with 600 ms RTT) has been shown in Figure 4.10 and discussed below.

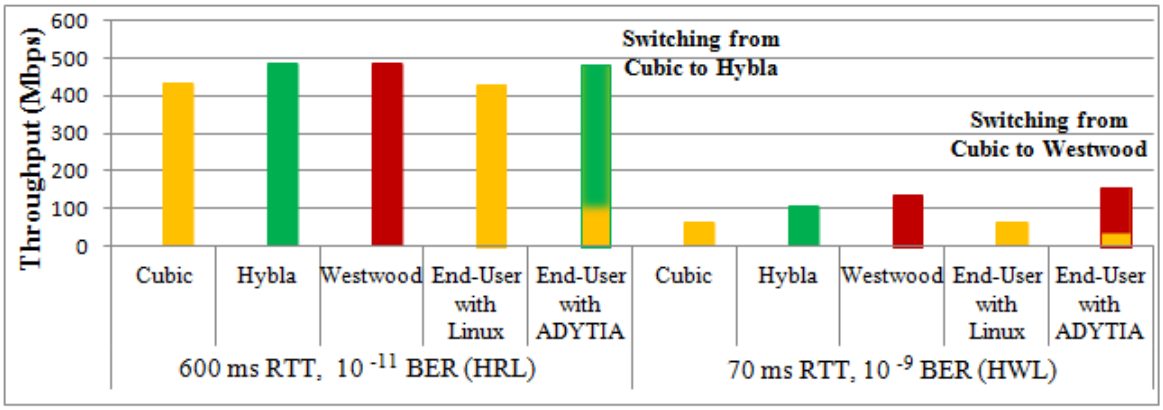


Figure 4.9: Dynamism of ADYTIA and performance comparison

4.3.1.1 Performance of Westwood

At BER of 10^{-6} and 10^{-7}, Westwood was able to achieve the throughput of more than 450 Mbps on HLL (i.e., a link with 45 ms RTT, negligible BER) and HWL (i.e., a link with 70 ms RTT, 10^{-9} BER). The reason for the high throughput was that Westwood has been specially designed to give good performance on the link with i) low RTT and low to moderate BER ii) high RTT with moderate to negligible BER. Hence, the design principle of Westwood exactly matches the characteristics of the links HLL and HWL that gives the high throughput on both the links. As Westwood uses the utilized bandwidth to set the values of ssthresh and cwnd, initially due to high utilization on HLL and HWL, it allows more exploitation of the bottleneck bandwidth (i.e., 1 Gbps). While in a case of HRL, due to high RTT and high BER, it was able to achieve very low throughput and frequent losses keep reducing the cwnd further. Thus, low utilization of the bandwidth by S3 triggered estimation of low available bottleneck bandwidth on link HRL. On HRL, for BER of 10^{-6} and 10^{-7}, throughput remained below 10 Mbps.

On link HRL with BER of 10^{-8}, initially Westwood estimated the more available

bottleneck link bandwidth and due to the high BER, it kept reducing cwnd. Thus, S3 achieved about 32 Mbps of throughput on link HRL and at the same time on HLL and HWL throughput of 425 Mbps has been achieved. As Westwood uses the estimation of available bottleneck bandwidth for setting the values of cwnd and ssthresh for all communication flows, overall link utilization remained 90% as shown in Figure 4.11. When BER of 10^{-9} was set on link HRL, S3 achieved the throughput of higher than 100 Mbps. S2 could exploit about 300 Mbps via link HWL because the available bottleneck bandwidth was less compared to BER of 10^{-6} to 10^{-8}. On HLL, S1 was able to achieve throughput of 450 Mbps due to the same reasons as discussed above. In this case, overall link utilization remained above 90% as shown in Figure 4.11.

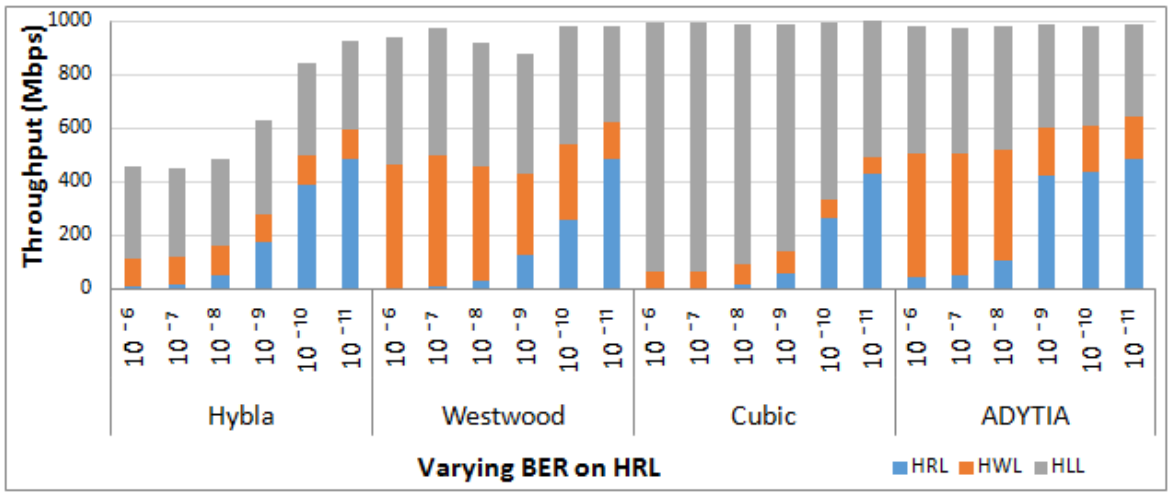


Figure 4.10: Performance comparison of ADYTIA with existing variants

When BER of 10^{-10} was set on link HRL, S3 could achieve throughput of higher than 250 Mbps as link characteristics were matching with the design principle of the Westwood. The same reason also holds true for the link HWL. In this case, overall link utilization remained about 95% as shown in Figure 4.11. At BER of 10^{-11} on link HRL, links HRL and HLL both were able to offer the significantly higher throughput as expected. At this situation, HWL (i.e. with BER of 10^{-9}) faces few losses that triggered the frequent reductions of cwnd and estimation of available bottleneck bandwidth was lower that results in low throughput on link HWL.

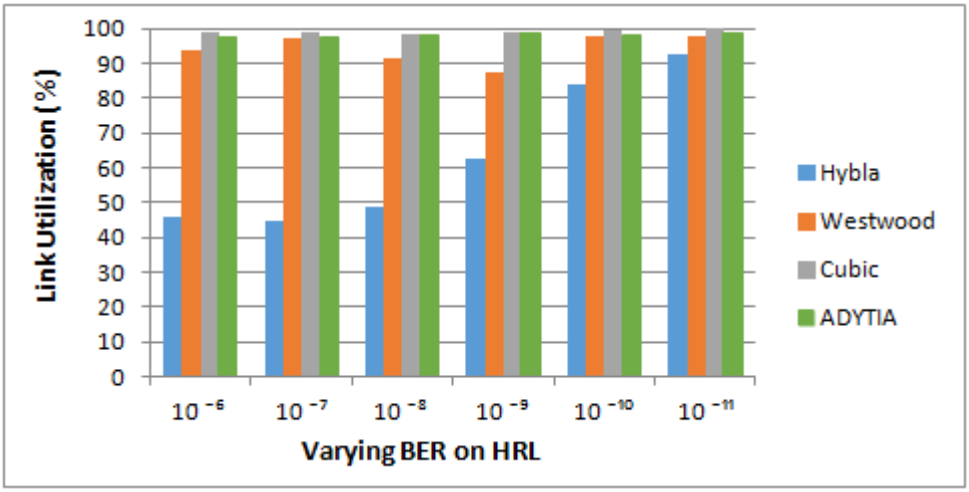


Figure 4.11: Bottleneck link utilization in heterogeneous network- dumbbell topology

4.3.1.2 Performance of Hybla

As shown in Figure 4.10, S1 was able to acquire the available bottleneck bandwidth on link HLL, and performance was noticeably well with respect to changing values of BER on link HRL. S2 was communicating via link HWL (i.e., offers BER of 10^{-9}) and Hybla does not have any mechanism to deduce the reason for the loss and hence its cwnd keep reducing in the presence of link errors. Thus, throughput on link HWL remained around 100 Mbps. Hybla has been specially designed to perform well in presence of high RTT and low link errors. In the case of BER of 10^{-6} to 10^{-8} for link HRL (i.e., offers RTT of 600 ms), the performance of Hybla remained significantly low (i.e. below 50 Mbps). The reason for such a poor performance was Hybla's inability to increase cwnd significantly and due to link errors, cwnd keeps reducing.

For the BER of 10^{-9}, on link HRL, the throughput achieved was around 120 Mbps as Hybla cannot deduce the reason of the loss and hence results in the reduction of cwnd in the presence of link errors. For the BER of 10^{-10} and 10^{-11} (i.e. significantly low BER), the throughput achieved on link HLL and HRL was around 400 Mbps, while throughput on link HWL remained around 120 Mbps because of high BER.

4.3.1.3 Performance of Cubic

As shown in Figure 4.10, when Cubic (i.e., default TCP variant on Linux) was running on all three senders, Cubic was able to achieve very high throughput on link HLL (i.e., 45 ms RTT, negligible BER link). Because Cubic has been designed to outperform

for the link with low to high RTT and negligible BER. S2 was communicating via link HWL with BER of 10^{-9}, it can be seen from the Figure 4.10 that Cubic's performance remained between 60 to 80 Mbps on link HWL. On link HRL, (i.e., 600 ms RTT link) with BER of 10^{-6} and 10^{-7}, S3 can achieve less than 10 Mbps of throughput and more than 90% of the bottleneck bandwidth has been utilized via link HLL only. With the BER of 10^{-8} and 10^{-9} on link HRL, Cubic cannot exploit more than 60 Mbps of the bandwidth. In case of BER 10^{-10} and 10^{-11} on link HLR, Cubic achieved reasonably high throughput. In summary, Cubic's throughput remained low even for the moderate BER and in a heterogeneous network as shown in Figure 4.8, the major fraction of the bottleneck link bandwidth has been utilized on the low RTT and negligible BER link (i.e., link HLL).

4.3.1.4 Performance of ADYTIA

As shown in Figure 4.10, when ADYTIA was running on all three senders, it selects the Cubic as a default TCP variant for all the links, at the starting of the communication flow. While Cubic was running, ADYTIA estimates the values of parameters as per the need of each module. Whenever there was a change in the observed and estimated values of parameters, ADYTIA uses the Information Base to decide the most suitable TCP variant to be used for the measured parameters. Once the variant has been decided, ADYTIA used the selected variant to update the values of ssthresh and cwnd.

When BER of 10^{-6} was set on link HRL, CCM estimated the available bottleneck link bandwidth value higher than 10 Mbps. The heuristic used by the CCM initially do not estimate any error on the link. Thus, CCM categorized the link as 'Long Fat Connection'. In ns-2, persistent FTP has been used for the simulation time of 600 seconds and based on this, AM triggered the selection of Hybla from the Information Base. Hence, S3 started using Hybla (i.e., switched from Cubic to Hybla) to update the values of cwnd and ssthresh. As ADYTIA was continuously observing all the parameters in CCM, it observed the link with losses and categorized the link as 'Long Fat Connection with Losses'. Thus, then onwards the algorithm used was Tarang (i.e., switched from Hybla to Tarang) as per the Information Base. Looking at Figure 4.10, it can be seen that the throughput achieved on this link was 42 Mbps. Thus, following performance improvement can be seen compared to:

i)Cubic (i.e., default variant of Linux) there is 4000% improvement ii) Hybla there is 600% improvement iii) Westwood there is 1000% improvement. It is important to observe that the performance enhancement is because of the usage of the appropriate algorithm used on the sender for updating the values of cwnd and ssthresh. In this context, ADYTIA's role is to select the variant from the Information Base depending on the link and application types. Hence, change in Information Base can give the different performance as the selected TCP variant would be different.

On link HWL (i.e. 70 ms RTT, BER of 10^{-9}), with BER of 10^{-6} on link HRL, CCM initially categorized the link as 'high speed connection' (as available bottleneck bandwidth estimated was higher than 100 Mbps) and switching of algorithm from Cubic to HighSpeed was performed. Later, CCM deduce the losses on the link (because of 10^{-9} BER) and again congestion control algorithm was switched by ADYTIA from HighSpeed to Westwood as per the Information Base. As can be seen from Figure 4.10, throughput achieved by Cubic was about 60 Mbps, while throughput achieved in case of ADYTIA was about 450 Mbps. Again, it is very critical to observe that this performance enhancement was because of the underlying algorithm used on the sender (e.g., HighSpeed, Westwood). Here, the role of the ADYTIA was to provide the selection of the appropriate variant based on the link and application types. In line with the above discussion, on link HLL, ADYTIA selected the HighSpeed (as the link was categorized as 'High Speed Connection') and throughput achieved was higher than 450 Mbps.

Looking at Figure 4.10, as BER was reduced (i.e., from 10^{-7} to 10^{-11}), throughput achieved on link HRL increased. The reason for the performance improvement was the mechanism of the selected algorithm that sets the values of cwnd and ssthresh, and reaction to the congestion and link error events. On link HWL, the achieved throughput was reduced because of the following reasons: i) Major fraction of bottleneck link bandwidth was utilized by other two connections ii) BER of 10^{-9} on link HWL, keep reducing the cwnd which results in the lower estimation of the available bottleneck bandwidth by the selected variant (i.e., Westwood).

In a nutshell, it can be concluded from the above results that in heterogeneous networks, one variant cannot achieve the efficient link utilization on all the links. Thus, ADYTIA that allows the usage of different variants based on the link type is

an important concept to achieve the better link utilization for different kinds of links. As shown in Figure 4.11, with the usage of ADYTIA, each link was able to achieve the better throughput depending on the link conditions with cumulative of 95% and more of the bottleneck link utilization.

4.3.2 Time Varying Link Characteristics

Simulation scenario with varying link characteristics at every 200 seconds has been shown in Figure 4.12. The simulation was carried out for 1000 seconds. In this scenario, it was assumed that the Internet Service Provider (ISP) was having different types of links for the backup to provide uninterrupted services to all the customers.

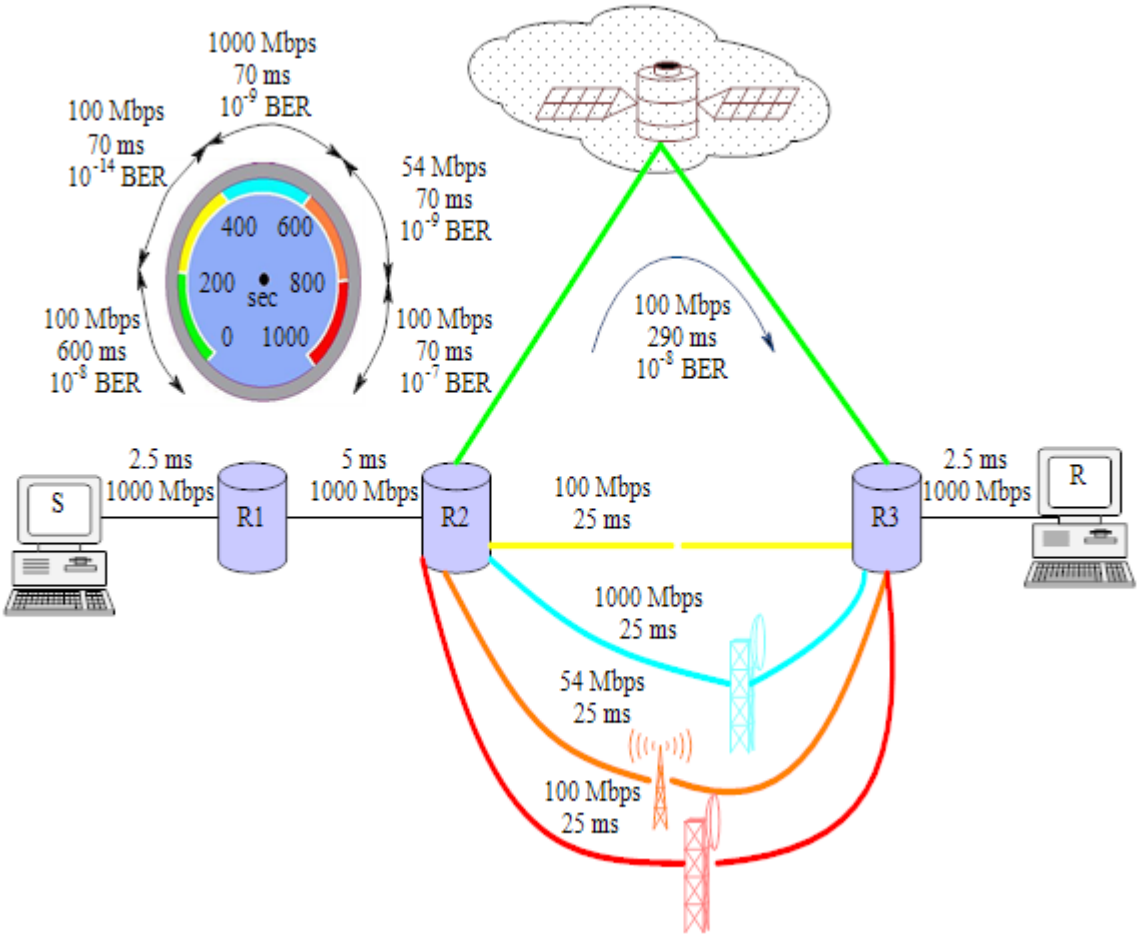


Figure 4.12: Simulation scenario with time varying link characteristics

Link characteristics in terms of bandwidth, RTT, and BER from the sender's perspective has been presented in Figure 4.12. This simulation scenario was used to test the dynamism of ADYTIA, especially its ability to change the variant at the run

time. FTP was used to generate the data at the sender for the simulation durations. The sender was running ADYTIA and communication has been started with default variant (i.e., Cubic). Based on the measured parameters, ADYTIA kept switching the variant from the Information Base to maximize the bottleneck link utilization.

The dynamism of ADYTIA that facilitates selection of different variants with respect to varying link characteristics has been illustrated in Figure 4.13. ADYTIA's ability to change the TCP variant at the run time helps to achieve maximum link utilization compared to the existing approach of the single and fixed variant for the entire duration of the communication flow. As depicted in Figure 4.13, ADYTIA uses the different variants based on the changes in link parameters. Further, it has been indicated that the ADYTIA was able to select the most suitable TCP variant depending on the link characteristics. Selection of the variant was totally in control of the Information Base which can be changed without modifying modules of ADYTIA and kernel recompilation. This feature was very innovative and facilitates the expert user or an organization to control the usage of specific variants based on application and link types.

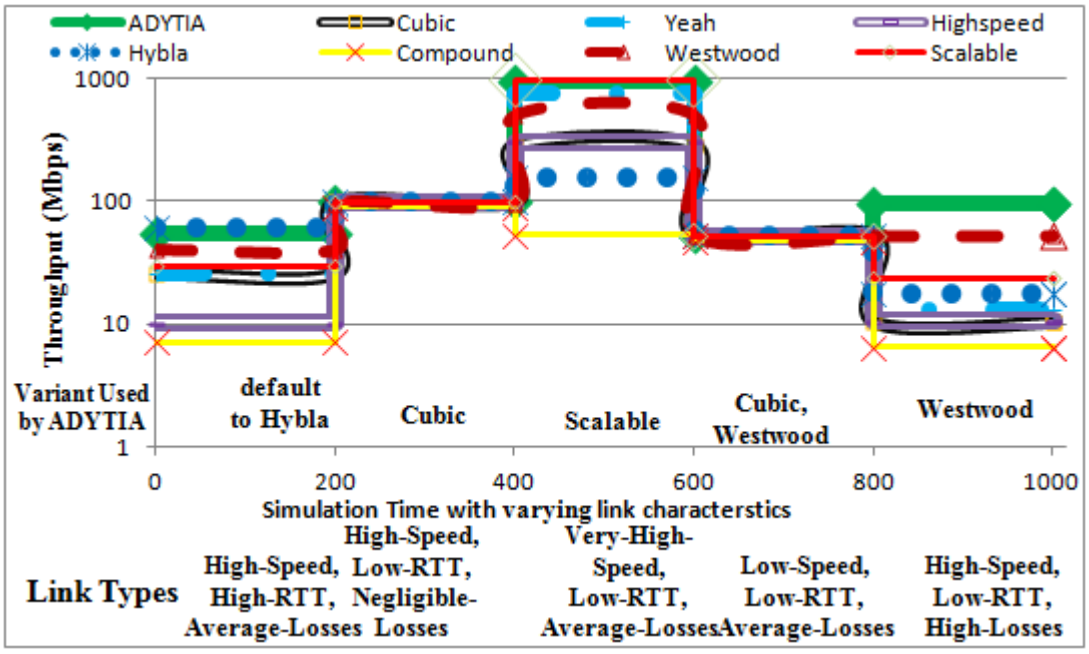


Figure 4.13: Dynamism of ADYTIA with time varying link characteristics

Throughput achieved by different variants compared to ADYTIA has been shown in Figure 4.14, and percentage (%) indicates the relative improvement of ADYTIA

with respect to the specific variant. ADYTIA outperformed Cubic (i.e., default in Linux) and Compound TCP (i.e., defaults in Window platforms) by 62.63% and 83.77% respectively which has been shown in Figure 4.14.

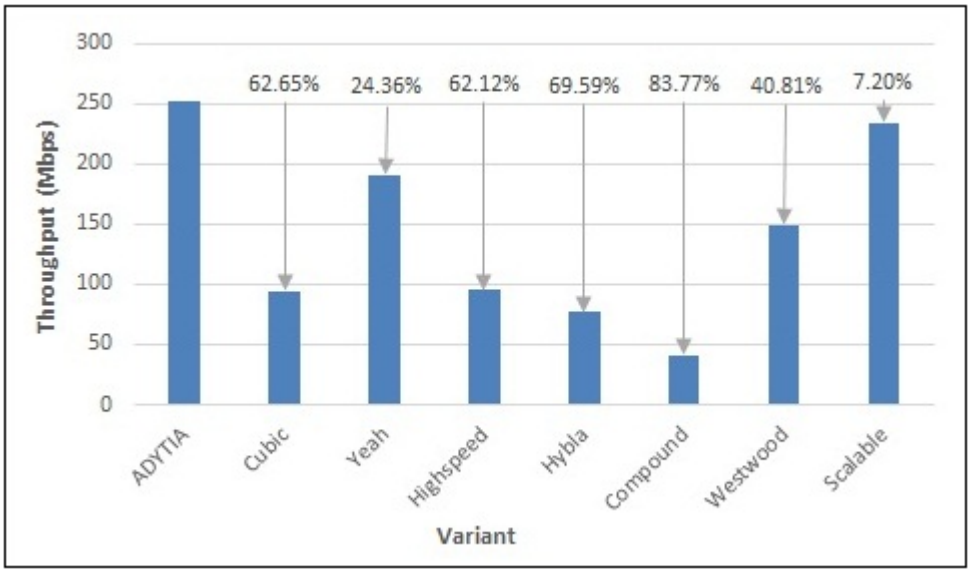


Figure 4.14: Throughput comparison of ADYTIA and existing variants with varying link characteristics

4.3.3 Internetworking

In this simulation scenario, three networks, each with three nodes communicate with corresponding receivers of the network via a common bottleneck link of 1 Gbps and all other parameters were same as described in dumbbell topology. Due to more number of senders, it creates more network dynamics for the TCP variants (e.g., different values of cwnd and ssthresh for each flow) that gave more insight into the performance analysis. All senders of each network were using the same variant at a time. The topology used for the simulation of the heterogeneous network has been shown in Figure 4.15.

In this simulation topology, 3 different types of network hosts were communicating via 3 groups of links as follow:

i)Network-1 (C1): Nodes of C1-C1 communicate via links 1-2-3 with RTT of 45 ms and negligible BER.

ii)Network-2 (C2): Nodes of C2-C2 communicate via links 4-5-6 with RTT of 70 ms and BER of 10^{-9}.

iii)Network-3: Nodes of C3-C3 communicate via links 7-8-9 with RTT of 600 ms and BER of 10^{-6} to 10^{-11}.

The objective of these experiments was to understand the dynamics of the network in the presence of more background traffic with different types of links.

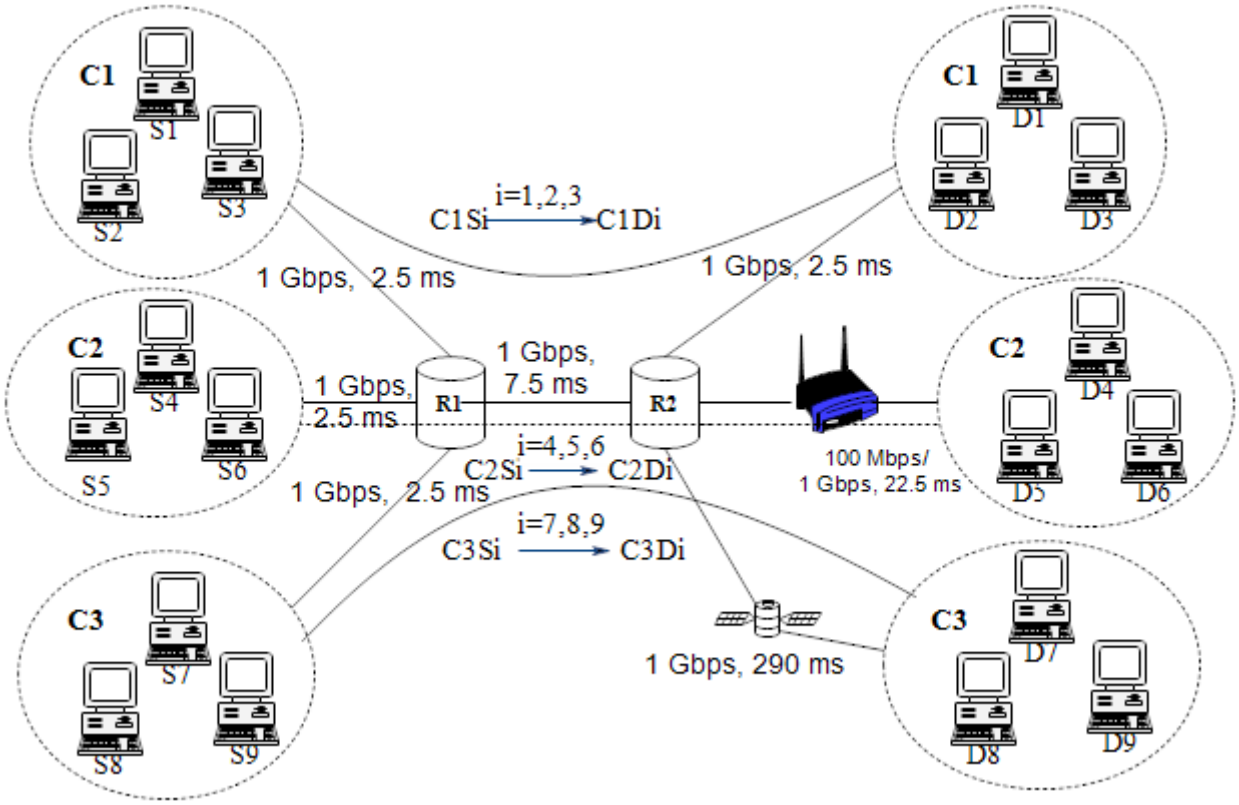


Figure 4.15: Simulation topology of internetworking

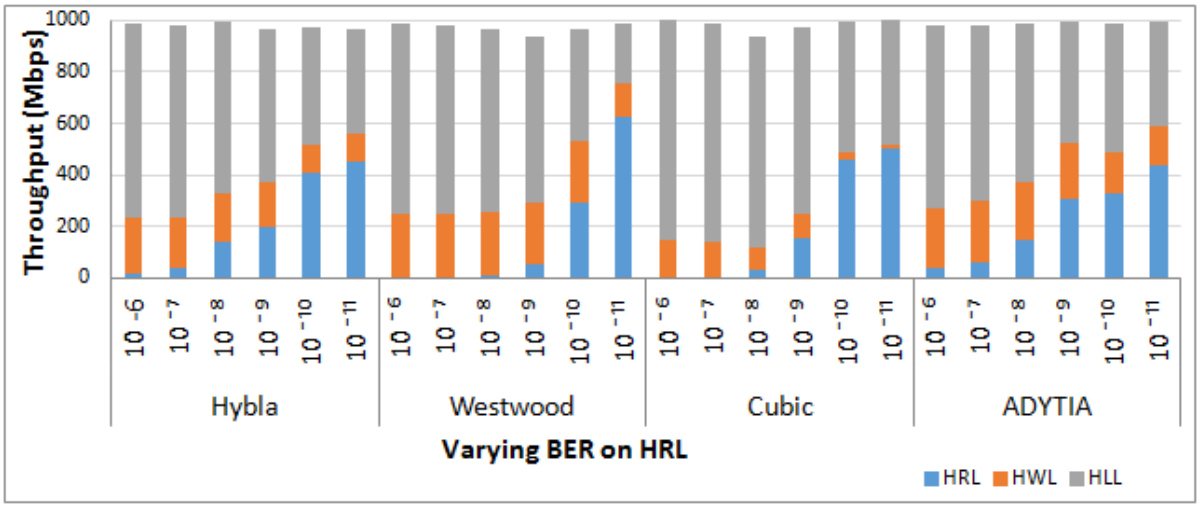


Figure 4.16: Performance comparison for internetworking scenario

As shown in Figure 4.16, when Westwood was running on all nine senders and BER on links 1-2-3 was negligible while BER of links 7-8-9 was 10^{-6} to 10^{-8} (i.e., very high BER), Westwood could estimate more available bottleneck bandwidth on all 3 senders communicating via links 1-2-3. Thus, as simulation progressed in time, 3 senders occupied the major fraction of the bottleneck bandwidth and throughput achieved on all these 3 senders collectively was about 700 Mbps. Throughput achieved via links 4-5-6 on 3 senders collectively was about 240 Mbps. Throughput achieved by remaining 3 senders via challenging links 7-8-9 (i.e., 600 ms RTT, BER from 10^{-6} to 10^{-8}) remained very low as shown in Figure 4.16. As BER was reduced on links 7-8-9, utilization of the bottleneck bandwidth by corresponding senders become relatively high and consequently utilization on other groups of links got reduced.

As shown in Figure 4.16, when Hybla was running on all 9 senders and links 7-8-9 having BER of 10^{-6} to 10^{-8}, aggregate throughput was higher as compared to Westwood, as Hybla was tailored for the high RTT links. As BER was reduced on links 7-8-9, throughput on links 7-8-9 increased and flows passing through links 1-2-3 and links 7-8-9 almost shared the bottleneck link bandwidth equally. While throughput achieved by senders 4,5,6 remained low as they were communicating via links that make an error and the senders needed to reduce the cwnd frequently.

When Cubic was running on all senders, it can be concluded from Figure 4.16 that major fraction of the bandwidth was utilized by senders S1, S2, and S3 collectively as they were communicating via low RTT and negligible BER links. Cubic has been designed for links with low to high RTT without link losses. Hence, Cubic could not exploit more bandwidth on other challenging links (i.e., links 4-5-6 and links 7-8-9). In a nutshell, it could be concluded that Cubic's throughput remains low on the challenging links in the presence of background traffic and especially when background traffic was on links with low BER.

When ADYTIA was running on all senders, it started operating with Cubic. Later, CCM of ADYTIA categorized the links 1-2-3 as the 'high speed connection' and selected the HighSpeed on all 3 senders as per the Information Base. At BER of 10^{-6} to 10^{-8} on links 7-8-9, senders of network C1 collectively acquired the major fraction of the bottleneck bandwidth as all other senders (i.e., rest of the 6 senders) were communicating via links with high BER.

Senders of C2 network started with Cubic and initially, CCM categorized the link as 'high speed connection'. Thus, senders of C2 started using HighSpeed as the congestion control algorithm. When these senders detected the loss on the links 4-5-6 (because BER was 10^{-9}), CCM categorized the links as 'high speed connection with losses'. Due to this, switching from HighSpeed to Westwood (As per the Information Base) was done, cwnd and ssthresh values updated based on the Westwood for the remaining simulation time. Westwood used the utilized bandwidth to set the value of cwnd and this value was less due to following: i) link errors ii) major fraction of the bottleneck bandwidth utilized by other senders (i.e., rest of 6). Hence, all three senders (i.e. of C2) with Westwood as the congestion control algorithm were able to achieve the throughput in the range of 200 to 250 Mbps.

When BER was 10^{-6} to 10^{-8} on links 7-8-9, as per the design of the ADYTIA, all 3 senders of network C3 started using Cubic. Initially, CCM categorized all three connections as 'long fat connection'. Thus, as per the Information Base switching of congestion control from Cubic to Hybla took place and cwnd dynamics were set as per the Hybla on all 3 senders. As BER was high on all three links, CCM detected the link with losses and categorized the connection as 'long fat connection with losses'. Hence, again congestion control algorithm was switched from Hybla to Tarang as per the Information Base. By looking at the graphs shown in Figure 4.16, it could be concluded that the combined throughput achieved by senders of C3 using ADYTIA was significantly higher compared to Cubic. Here, an important observation is that ADYTIA allows the usage of different variants on each sender and switching of the algorithm was performed as per the link characteristics. As BER was reduced from 10^{-9} onwards, throughput achieved by senders of C3 increased and throughput on links 4-5-6 was reduced (because BER was 10^{-9}).

4.4 Summary

This chapter presented the design and implementations strategies of ADYTIA. It also analyzed the performance of ADYTIA and compared the results with the existing TCP variants. Performance analysis was carried out using three scenarios: i) Three senders communicating via different types of links to simulate the heterogeneous network (i.e., using dumbbell topology) ii) Time varying link characteristics to

explore dynamic behavior of ADYTIA iii) Nine senders were communicating to simulate communication among three types of networks with different link characteristics. This scenario gave an opportunity to analyze the performance in the presence of more background traffic with diversified characteristics. Following conclusions were made about the existing approaches:

- In heterogeneous networks, in the presence of competing flows, ADYTIA's feature of selecting different variants for each type of link results in significant performance improvement compared to existing variants. In a worst-case scenario, for the link with BER of 10^{-6} and RTT of 600 ms, following performance improvement could be seen compared to: i) Cubic (i.e., default variant of Linux), there is 4000% improvement ii) Hybla, there is 600% improvement iii) Westwood, there is 1000% improvement. It is important to observe that the performance enhancement was because of the usage of the appropriate algorithm used on the sender for updating the values of cwnd and ssthresh. In this context, ADYTIA's role was to select the variant from the Information Base depending on the link and application types.

- Simulation results showed that ADYTIA was able to change the variant for each connection depending on the characteristics of the link and application layer protocol. The changing of variants per connection type allowed effective link utilization. The experimental results established that ADYTIA allowed the change of TCP variant with varying link characteristics and it always selected the best available TCP variant to maximize the link utilization. ADYTIA's performance is 62.65%, 82.87% higher compared to default variant of Linux (i.e., Cubic) and Windows (i.e., Compound TCP) respectively with time varying link characteristics. This exhibited the dynamic nature of ADYTIA, that converges to various application requirements. ADYTIA used the Information Base, which was prepared after rigorous literature survey to select the most suitable TCP variant. The Information Base of ADYTIA allowed any newly designed TCP variant to be plugged-in without disturbing the entire framework. Further, to demonstrate this feature, Tarang was plugged-in with ADYTIA, and a comprehensive analysis of the result was presented, so as to conclude its adaptive

(plug-in) and dynamic (change of variant at runtime) behavior.

- ADYTIA was able to categorize the link as per the link characteristics. Links with 600 ms RTT and average to high BER, categorized as long-fat links with losses. Links with 70 ms RTT and BER of 10^{-9} is categorized as Low-delay-link with losses. Such a categorization of link types and also the identification of the application helped the ADYTIA to select the appropriate TCP variant on each link in heterogeneous networks. In summary, the major contribution of the ADYTIA was that it allowed the use of different TCP variants on each type of link to maximize the throughput on each link. Further, ADYTIA allowed plugging of futuristic underlying technologies (e.g., 5G), TCP variants, and applications without the need for modifications in ADYTIA's modules and code rework.

In a nutshell, ADYTIA could be considered for all operating systems' kernel module as a standard. ADYTIA has been developed for the Linux kernel and tested in the simulation environment using the feature of ns-2. Although, simulation environment provides the performance details of ADYTIA, still it is required to be tested on kernel level implementation with the real operating system. Experiments through real systems (i.e., testbed experiments) give more insight for the deployment. ADYTIA's feature of selecting a TCP variant based on link and application type is quite attractive but, deployment has been a challenge as ADYTIA needs to be inserted in each end-user's operating system's kernel. ADYTIA's feature of selecting different variants for each flow may result into fairness issue in heterogeneous networks. The next chapter elaborates deployment feasibility and integration of ADYTIA with PEP, ADYTIA's experimental analysis on the testbed and live Internet.

Chapter 5

Integrating ADYTIA with PEP

As concluded in the previous chapter, to utilize the ADYTIA, it was necessary to modify the kernel of end-user's operating system. To ease the deployment of ADYTIA without modifying end-user's kernel, it was required to integrate ADYTIA with PEP. In this chapter, ADYTIA as explained in Chapter 4, has been implemented as a loadable kernel module and PEP engine has been integrated with the ADYTIA to use it on a single machine. This kind of setup allows the other sender machines to use the default operating system. This chapter also reports experimental work carried out the testbed and on live Internet.

The chapter presents the brief overview of the implementations, followed by explanations of the testbed setup used for the experiments. The following section explains the results obtained from the testbed and its analysis and comparison with the existing approach. Experimental work has also been carried out on the Internet to investigate the outcome of ADYTIA as discussed in Chapter 4. Finally, the chapter has been summarized with the benefits of ADYTIA with PEP approach and deployment feasibility. Furthermore, results presented in the chapter give more insights on the behavior of the proposed approach in realistic conditions.

5.1 Related Work

ADYTIA as presented in Chapter 4, was initially implemented in Linux kernel and used in ns-2. The same code needs to be migrated on the Linux kernel with corresponding changes as per the framework of the congestion control algorithms. In the previous version of the Linux kernel (i.e., before 2.6.13), newly designed congestion

control algorithm variants made the TCP code complex and required kernel recompilation. Since Linux kernel 2.6.13 and later versions, it supports plugging in of the congestion control modules. New congestion control algorithm has been implemented as a module in the kernel. Whenever a specific congestion control algorithm has been used, it is loaded at run time without kernel recompilation. It allows switching of the congestion control algorithms and performance analysis of newly designed congestion control algorithm in Linux.

The functions used by the congestion control algorithm were registered (by a structure tcp_congestion_ops) to tcp_register_congestion_control (i.e., part of tcp_cong.c). The structure tcp_congestion_ops (i.e defined in linux/include/net/tcp.h) contains the declaration of congestion handler interface that allows plugging of congestion control algorithm into the Linux kernel. A newly designed congestion control algorithm must implement cong_avoid and ssthresh functions. The algorithm used by every connection has been initialized by the kernel or via sysctl command. Once the congestion control algorithm has been decided, structure tcp_congestion_ops has been used to access the functions of the specific congestion control algorithm (Arianfar).

5.1.1 PEP Engine

In order to allow the usage of different TCP variants on each sender, ADYTIA needs to be implemented on each sender. Thus, it was compulsory to modify the operating system (e.g., modified Linux kernel) of each machine. Due to this requirement, deployment of the solution was not feasible as asking each user to modify the operating system code was not the logically feasible. Hence, ADYTIA has been configured and integrated on a single machine (i.e., can be called as a proxy machine) with the PEP engine. Usage of PEP engine allowed each user to use their operating system without any modifications and at the same time, ADYTIA allows the usage of different variants with different kinds of communication flows. As a PEP engine, PEPSal (Carlo.Caini, Firrincieli, and Lacamera) has been used which has been described in brief in the next subsection.

5.1.1.1 PEPSal Architecture

There are four threads working for the PEPSal named as queuer, listener, poller, and worker. Interaction of PEPSal with network layer has been shown in Figure 5.1.

PEPsal uses netfilter to pass TCP segments for queuer thread and receiving these packets back to the kernel. When netfilter received the incoming connection, two targets have been applied: i) NFQUEUE target in PREROUTING chain of mangle table for the TCP SYN segments ii) Other TCP segments have been redirected to port 5000 by REDIRECT target in the same chain of nat table, which have been handled by the listener thread.

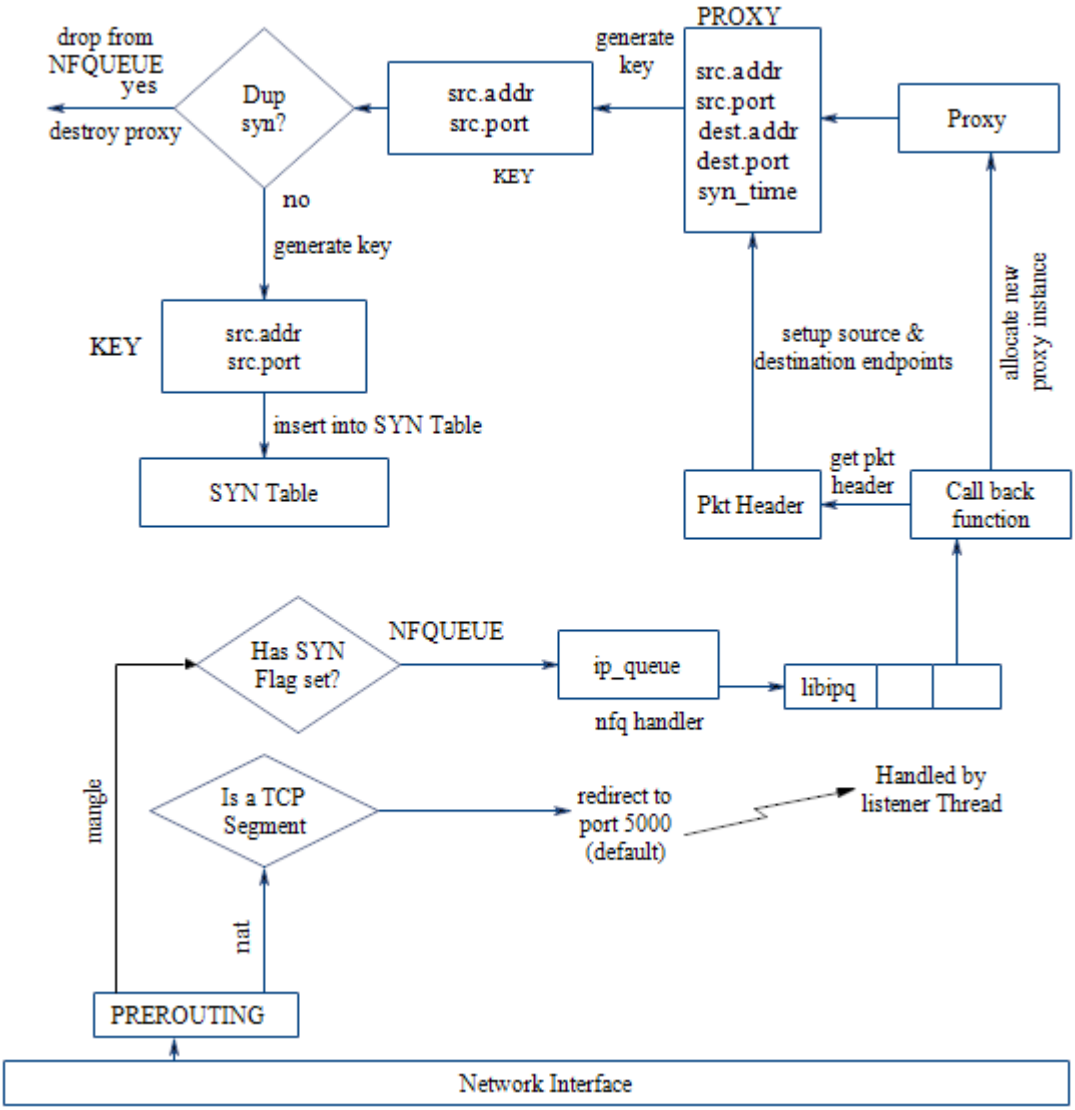


Figure 5.1: PEPSal interaction with network layer

The queuer process waits for the incoming new connection from the netfilter. IPv4 queuer handler ip_queue (implemented as libipq) has been used for kernel and user space interaction. TCP SYN segment from libipq has been handled by queuer process. New incoming connections were inserted into SYN table. It got the packet

header to know the information about two real endpoints. After allocating the new proxy instance, it sets source and destination endpoint of proxy. Next, the queuer process generates a key which is the combination of source address and source port of newly created proxy of the incoming connection. This information has been used later in PEPSal. SYN table is the part of shared memory, which contains the information about all the connections which are handled by PEPSal.

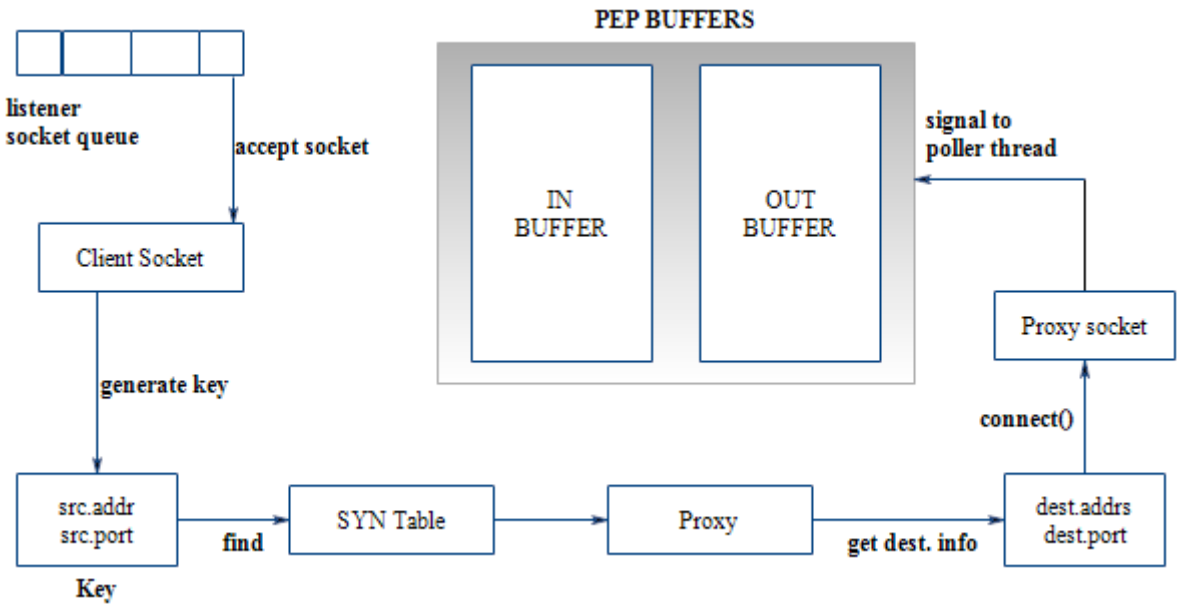


Figure 5.2: PEPSal interaction with transport layer

Interaction of PEPSal with the transport layer has been shown in Figure 5.2. The listener thread at transport layer handles all the subsequent segments of that connection which were redirected by netfilter to port 5000. All these packets were queued in listener queue. Listener thread created new client socket for each accepted connection and searched them in the SYN table using key (i.e., source IP and port). When it found the matching key, it got the destination IP address and port number from the corresponding proxy connection information. Then, a new TCP connection was established between the PEPSal and real destination, and it gave the signal to the poller thread. The poller thread initialized the PEP buffers. There were two buffers: i) IN BUFFER that was used for incoming connection ii) OUT BUFFER that was used for the outgoing connection.

At the application layer, PEPSal created the number of worker thread in PEPSal thread pool. Active queue contains information about established connections. Now,

proxy starts reading from one socket and writing all the data in the other. As shown in Figure 5.3, for incoming segments, PEPSal read data from client socket and pushed it to IN BUFFER and wrote the data from IN BUFFER to destination socket. On the other side of the interface, PEPSal uses OUT BUFFER to read and write the data. This process continued until all data were transferred. When the connection was terminated, its twin sockets were closed and its memory released.

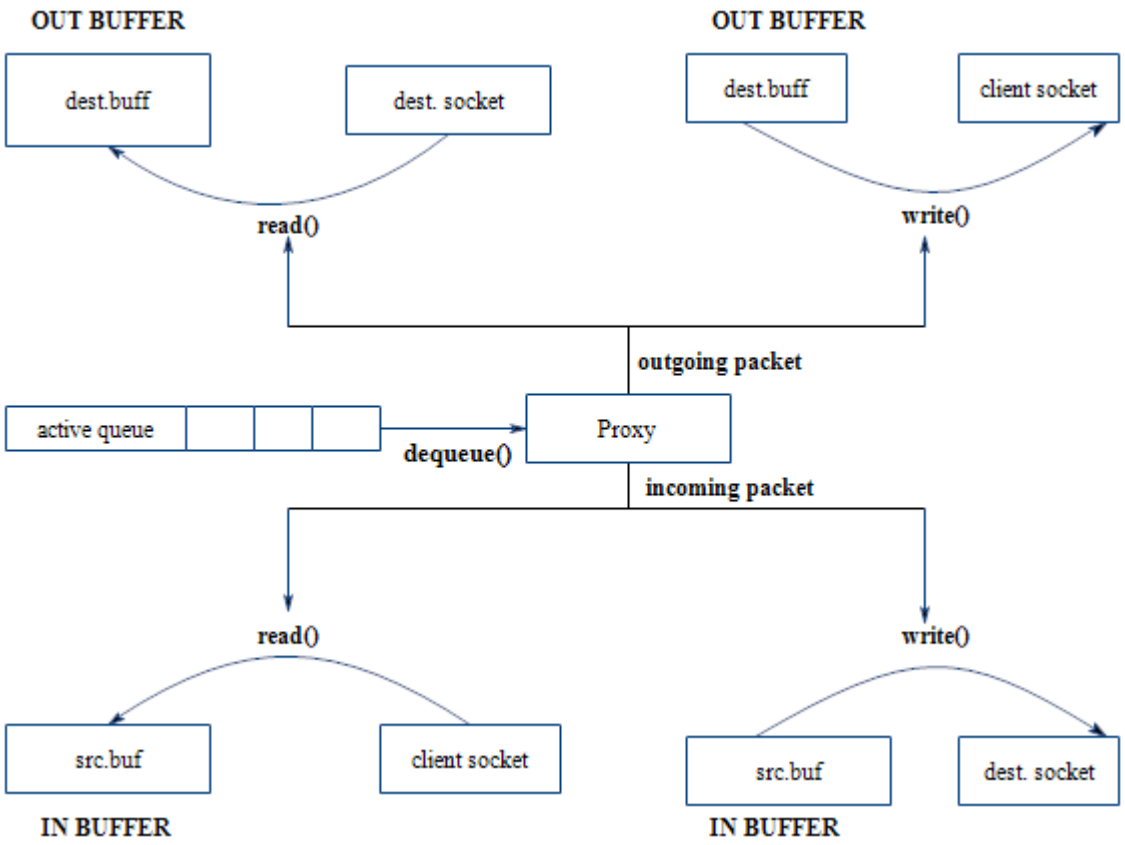


Figure 5.3: PEPSal interaction with application layer

5.2 Testbed and Result Analysis

The objective of the Testbed for Heterogeneous Networks (T-HetNet) was to show the deployment feasibility without modifying end-user's kernel and for the performance analysis of ADYTIA with PEP on the real system with kernel-level implementations. T-HetNet setup has been shown in Figure 5.4. A satellite sender was having IP address 10.10.108.193 has been connected to satellite receiver having IP address 10.10.108.199 with 1 Gbps link through Router 1 (10.10.108.196) and Router 2 (10.10.108.197). Similarly, a wireless sender (10.10.108.194) has been connected

to a wireless receiver (10.10.108.200) and a wired sender (10.10.108.195) has been connected to a wired receiver (10.10.108.201). In T-HetNet, all these machines were connected with each other using switch of 1 Gbps capacity. Link from Router R1 to R2 was the bottleneck link. Following were the configurations of the machines used in the T-HetNet:

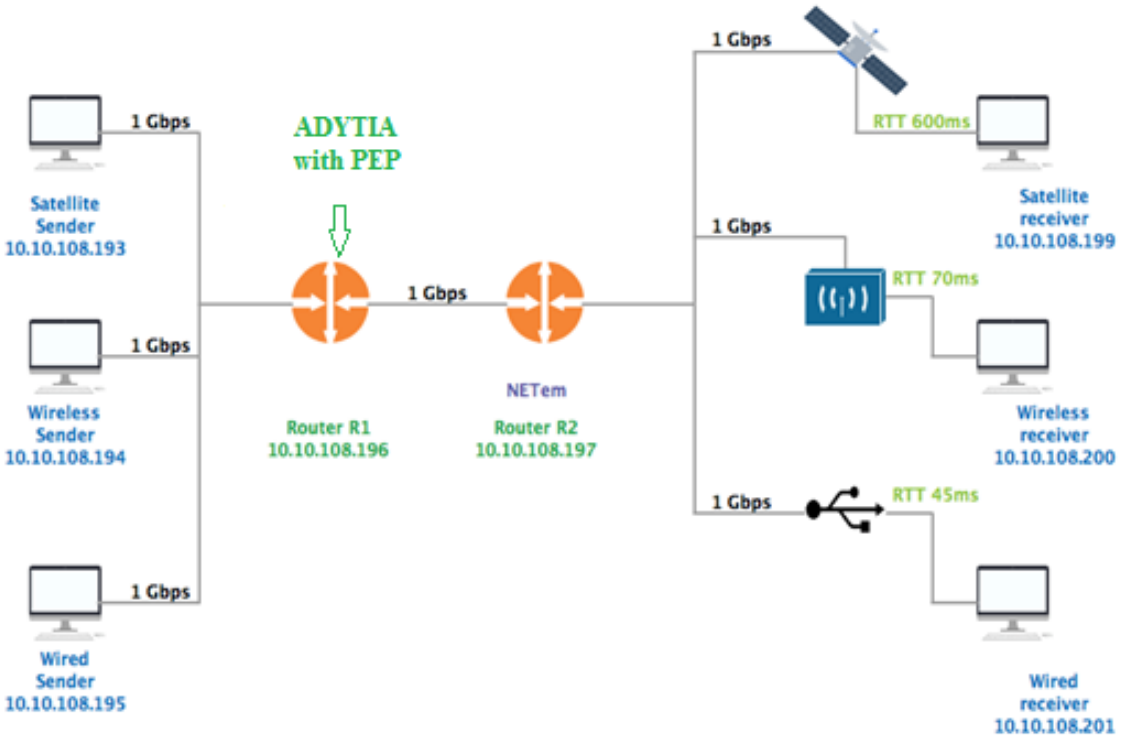


Figure 5.4: Testbed for heterogeneous networks (T-HetNet)

- **Router 1:**

  CPU: Intel(R) Pentium(R) 4, 2.4 GHz

  RAM: 8 GB

  Ethernet Card: Two cards of 1 Gbps

  Operating System (OS): Fedora 9 Linux, Kernel 2.6.26 or higher

- **All other machines:**

  CPU: Intel(R) Pentium(R) 4, 2.4 GHz or higher

  RAM: 1 GB or higher

  Ethernet Card: 1 Gbps

On every machine, there was a need to set the routing table to forward all the data via Router R1 and R2, and not directly from the sender to receiver. Every day this activity had to be repeated before starting of the testbed experiments. Thus, the shell script was prepared to be executed on each machine and has been provided in Appendix-A. On Router R1, ADYTIA and Tarang were required as a loadable kernel module. Appendix-B list the steps need to be followed to put the new TCP variant into the kernel and other commands used in testbed experiments. PEP Engine (i.e., PEPSal) required to configured on Router R1 along with ADYTIA, Appendix-C elaborates the steps to configure and start the PEPSal, and the firewall rules that need to be inserted on the machines with specific targets. To set the values of RTT and BER, emulator tools like nistnet or netem could be used. To capture the values of cwnd and ssthresh a tool like TCP probe could be used. Iperf has been used to generate the traffic on sender machines. These tools have been explained in next subsection.

5.2.1 Tools

To set up the testbed topology as shown in Figure 5.4, and to do the experimental works in the laboratory environment, it was necessary to use few tools along with some utility programs of the operating system. The tools used in the experimental setup have been discussed in the following subsections.

5.2.1.1 NISTNet Emulator

NISTNet (Carson and Santay) is a network emulation package that runs on Linux. NISTNet allows a single Linux machine to set up as a router to emulate the wide variety of network conditions. The tool is designed to allow controlled, reproducible experiments with network's performance-sensitive applications and control protocols in a simple laboratory setting. By operating at the IP level, NISTNet can emulate the critical end-to-end performance characteristics imposed by various wide area network situations. NISTNet has been implemented as a kernel module extension to the Linux operating system and an X Window system based user interface application. The tool allows an inexpensive computer-based router to emulate numerous complex performance scenarios, including tunable packet delay distributions, congestion and background loss, bandwidth limitation, and packet reordering/duplication. The X

interface allows the user to select and monitor specific traffic streams passing through the router and to apply selected performance "effects" to the IP packets of the stream.

5.2.1.2 Netem

Netem (Hemminger et al.) is a network emulator in the Linux kernel 2.6.7 and a higher version that reproduces network dynamics by delaying, dropping, duplicating or corrupting packets. Netem is an extension of tc, the Linux traffic control tool. Any Linux machine running Netem must be configured as a router. Inside the router, IP packet handling is performed as follows: Packets enter a Network Interface Card (NIC) and then classified and queued before entering Linux internal packet handling. After packet handling, packets are classified and queued for transmission on the egress NIC as shown in Figure 5.5. The details of the tc, queuing discipline and commands have been explained in details in Appendix-D.

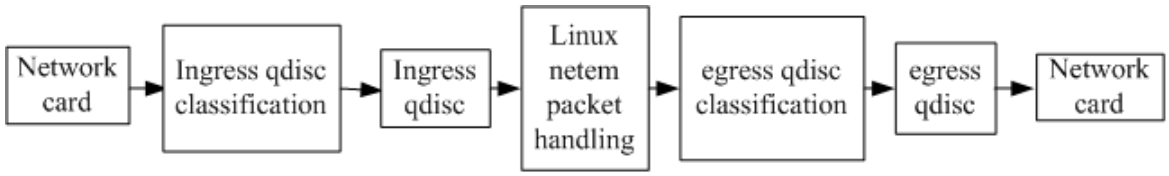


Figure 5.5: IP packets handling by Netem in Linux kernel

5.2.1.3 Iperf

Iperf was originally developed by NLANR/DAST as a modern alternative for measuring TCP and UDP bandwidth performance. Iperf is a tool to measure maximum TCP bandwidth, allowing the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay, jitter and loss of segments. By default, the Iperf client connects to the Iperf server on the TCP port 5001 and the bandwidth displayed by Iperf is the bandwidth from the client to the server. Details of Iperf usage has been illustrated in the Appendix-E.

5.2.1.4 TCP Probe

To understand the performance issues of any TCP variant, it is important to record the values of cwnd and its rate of increments. TCP probe is a module that records the state of a TCP connection in response to incoming packets. It works by inserting a hook into the tcp_recv processing path using kprobe so that the congestion window

and sequence number can be captured. To utilize the TCP probe module an entry of tcp_probe required in makefile as follows:
obj-m := module_name.o
After this change, it is required to run the command make and make install in sequence. Next, insert tcp_probe module using the command insmod tcp_probe.ko.

5.2.1.5 Linux TCP Performance Tuning

Linux has auto tuning feature to update receiver buffer size and TCP sender window size for each connection. The control of Linux auto-tuning feature is controlled by a variable /proc/sys/net/ipv4/tcp_moderate_rcvbuf. The default value of maximum 4MB buffer in Linux is not sufficient for high BDP networks. For example, a link with the bandwidth of 1000 Mbps and RTT of 600ms require the buffer size of 75 MB to fill the link fully. The memory size of TCP receiver and sender per connection was set with two variables (i.e., defined as array of 3 elements) as mentioned below:
/proc/sys/net/ipv4/tcp_rmem (for TCP receiver buffers)
/proc/sys/net/ipv4/tcp_wmem (for TCP sender buffers)
Above listed variables have three values: minimum, initial and maximum size. They are used to set the threshold on auto tuning and balance the memory usage under heavily loaded systems. This allocated memory includes memory used by socket data structure and TCP window size. Thus, the maximum value should be larger than the BDP of the link. The maximum values of TCP sender and receiver buffer size is governed by variables /proc/sys/net/core/wmem_max and /proc/sys/net/core/rmem_max respectively. It is also important to set ethernet queue size, otherwise, it can be the bottleneck on the performance of high BDP links. Hence, its value should be set to a large value. For example, the command 'ifconfig eth0 txqueuelen 100000' will set the maximum queue length in ethernet card to 100000 segments. By default, TCP saves various connection metrics in the route cache when the connection closes, so that connections established in the near future can use these values to set initial conditions. Usually, this increases overall performance, but may sometimes cause performance degradation. To disable this, a command 'sysctl w net.ipv4.tcp_no_metrics_save = 1" is used.

5.2.2 Result Analysis

This subsection elaborates the results of ADYTIA with PEP on T-HetNet as shown in Figure 5.4. A comparative analysis and investigation were also carried out with the Linux operating system's default TCP variant Cubic. Performance metrics and parameters used for the results analysis have been presented in the following subsection. Experiments were conducted on the topology shown in Figure 5.4 using Iperf and FTP client-server configurations as detailed in following subsections.

5.2.2.1 Performance Metrics

To evaluate and compare the performance of ADYTIA with PEP, and Linux's default TCP variant, following performance metrics were considered:

Throughput: It is defined as the number of bits that can be transferred in a second. Throughput is the most important parameter to represent the protocol's effectiveness on any end-to-end systems. In this experiment, throughput has been measured in the presence of background traffic for different types of links having following characteristics:

- RTT: It is the total time required for a segment to be transmitted successfully (i.e., the arrival of an acknowledgment on the sender) from sender to receiver. To emulate the heterogeneous networks, each link with different RTT has been accommodated in the topology. Links with different RTT was the real challenge for the TCP variant to achieve the best possible throughput.

- BER: It introduces the random losses on the link. To emulate the different kinds of links, different values of BER were set on each link. The link like satellite always has different values of BER at the different time (i.e., due to bad weather, interference, etc.). Throughput was measured by varying the BER values on a link (i.e., 600 ms RTT link) that offers diversified background traffic for the other senders.

Link Utilization: It indicates the combined utilization of the bottleneck link by all the communication flows passing through it. Here the bottleneck link was the link with the minimum capacity or a link which has been shared among multiple communication flows, on the path between the sender and receiver. Ideally, all

communication flows from the sender's perspective should get the equal share of the bottleneck link capacity. However, there were many parameters (e.g., link characteristics, queue dynamics, traffic patterns, etc.) including the design of the TCP variant play an important role in deciding the link utilization of the specific flow and overall link utilization by all the flows.

File Transfer Duration: In one of the scenario, an application used was FTP that transfer the different size's file on different types of links and objective was to see how fast the file has been transferred. This parameter is relevant to the throughput and also indirectly measure the quality of the end user's experience.

ADYTIA's Selection of TCP Variants: Throughout all the experiments, with the variety of link characteristics, ADYTIA selects different TCP variants based on the link and application type and as per the Information Base. Analysis of the selected variant gave insights in the following: i) Link category as per CCM and actual link ii)Application Type iii) Selected variant switching was based on Information Base.

5.2.2.2 T-HetNet with Iperf

As shown in Figure 5.4, all three senders were using Fedora 15 (i.e., Cubic is a default TCP variant) and communicated via a common bottleneck link of 1 Gbps with different receivers. ADYTIA with PEP was inserted on a gateway through which all communication flows were passing. All three senders experience different RTTs (varies from 45 ms to 600 ms), bandwidth, and BER (negligible to as high as 10^{-6}) to emulate wired, wireless and satellite link. Further, traffic was generated using Iperf. To vary the link parameters, network emulator (i.e., netem) has been used. BER of satellite link varied from 10^{-6} to 10^{-11} that created different network dynamics and ADYTIA's ability to select the variant based on application and link type was tested. Varying values of BER indicates the changes in weather conditions of the satellite link. Based on the link and application type, ADYTIA was able to operate with different variants on all three links.

Performance comparison of Hybla, Westwood, Cubic, and ADYTIA with PEP on three different kinds of links has been illustrated in Figure 5.6. When Hybla was running on all three senders, on a wired link (i.e., 45 ms RTT and negligible BER) in presence of background traffic on wireless (i.e., 70 ms RTT, 10^{-9} BER) and satellite link (i.e., with BER from 10^{-6} to 10^{-9}), Hybla was able to operate around 350 Mbps.

On a wireless link, in the presence of background traffic on other two links, Hybla's throughput remained 90 to 100 Mbps. The reason for such poor performance on the wireless link was that Hybla did not have any mechanism to differentiate the reason of the losses. Thus, in the presence of 10^{-9} BER, cwnd reduction was frequent that kept the throughput low.

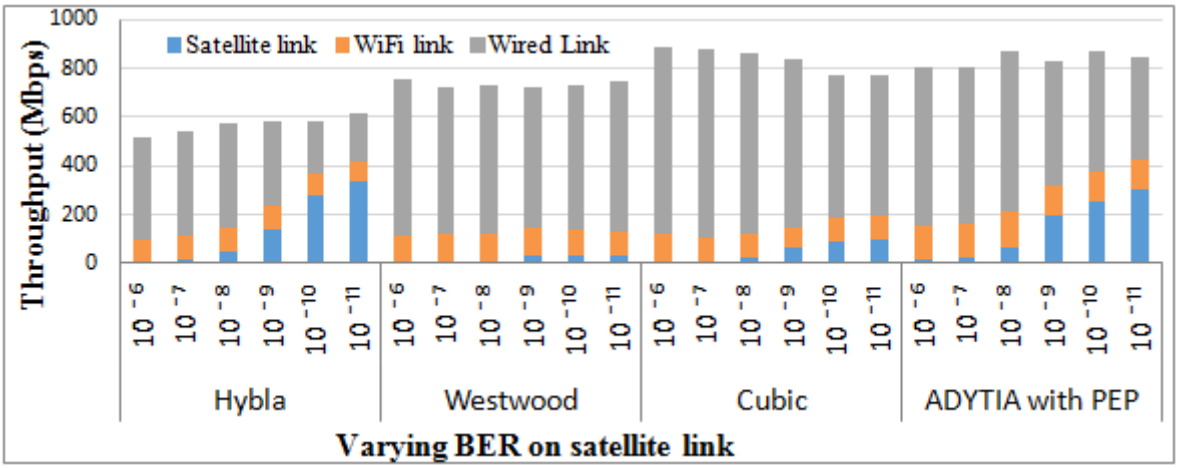


Figure 5.6: Performance comparison of existing variants and ADYTIA with PEP

On satellite link due to the high BDP, cwnd needs to be high in order to utilize the available link capacity significantly. But, high BER did not allow the sender with Hybla protocol to increase the value of cwnd. Hence, with BER from 10^{-6} to 10^{-8} on the satellite link and with background traffic on other two links, Hybla's throughput remained very low. As BER is reduced towards 10^{-11}, throughput has been increased significantly. The reason for improved throughput is that Hybla was specially designed for the high RTT links. Hence, with less number of link errors, Hybla sender was able to increase the cwnd significantly and operated around 300 Mbps at BER of 10^{-11}.

When Westwood was running on all three senders, it achieved the throughput of 600 Mbps on a wired link in the presence of background traffic on other two links. As Westwood used the available bottleneck link bandwidth to set the values of cwnd and ssthresh, the wired link got the major fraction of the bottleneck link bandwidth as satellite link have the challenging characteristics for which Westwood was not designed at all. Because of the high BER and high RTT on the satellite link, cwnd reduction frequency was very high and utilized bandwidth by sender remained very

low. Thus, this low utilized bandwidth of the bottleneck link maintained the low values of cwnd and ssthresh as per the design principle of Westwood and throughput remained very low. On wireless link as the BER was moderate and the link has low RTT that allows the Westwood to achieve the throughput of 100 Mbps.

Bottleneck link utilization when three senders running Hybla in the heterogeneous networks has been shown in Figure 5.7. At higher values of BER on the satellite link, utilization of that link remained significantly low. Hence, residual link capacity remained as high as 48%. As Hybla does not have any mechanism to detect the reason of the losses, link utilization on wireless link is about 10% only. Bottleneck link utilization when all three senders were running Westwood, has been illustrated in Figure 5.8. As Westwood was using the estimated bandwidth to set the values of cwnd and ssthresh after the congestion, residual capacity was low compared to Hybla. Westwood utilized only 3% of the bottleneck link capacity on satellite link because of high RTT of the link for which Westwood has not been tailored.

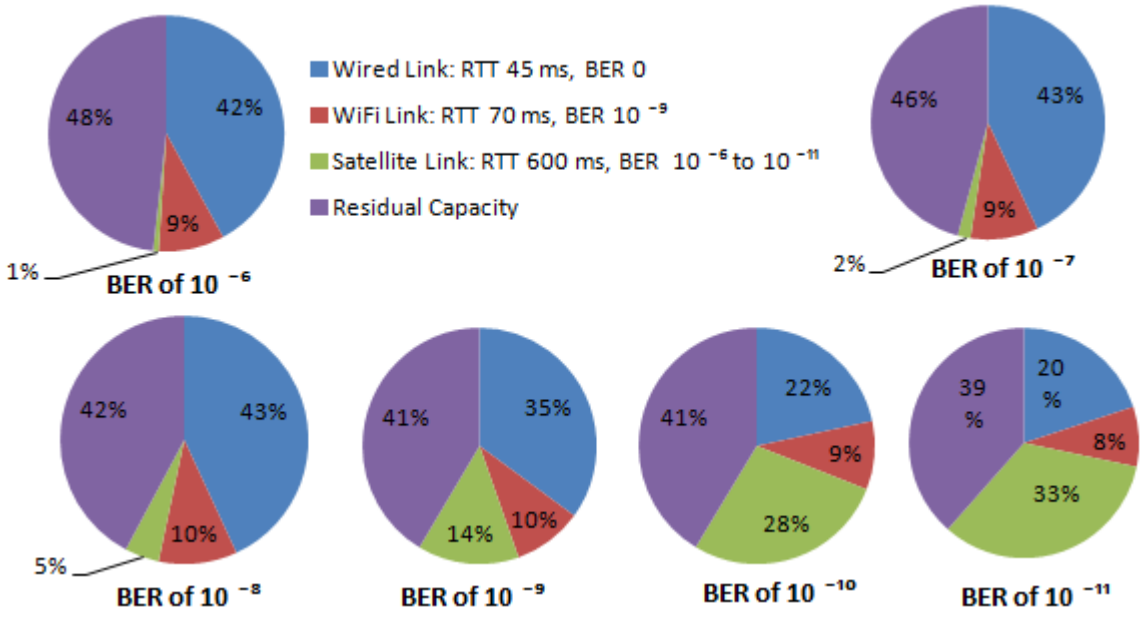


Figure 5.7: Link utilization of Hybla running on all 3 Senders

Performance comparison of Cubic (i.e., default variant of Linux kernel 2.6.19 and higher versions) and ADYTIA with PEP has been shown in Figure 5.6. When Cubic was running on all three senders of the testbed topology shown in Figure 5.4, at BER of 10^{-6} to 10^{-8} on satellite link: i) Throughput achieved by sender communicating via satellite link remained very low because Cubic was not able to increase the cwnd

in presence of link losses ii) In the presence of background traffic (i.e. via wireless and satellite link), Wired-sender that was communicating via wired link was able to capture major fraction of the bottleneck bandwidth and operated around 800 Mbps iii) Wireless link having BER of 10^{-9} could not exploit the bottleneck link bandwidth due to frequent losses and throughput achieved was about 100 Mbps. Further, as BER was reduced on satellite link, throughput achieved by satellite-sender (i.e., via satellite link) increased and it remained around 100 Mbps with a small reduction for the wired link. This experiment reveals that Cubic could not exploit the bottleneck link bandwidth for the challenging links like wireless and satellite in the presence of competitive flows on low RTT and negligible BER link (i.e., wired link).

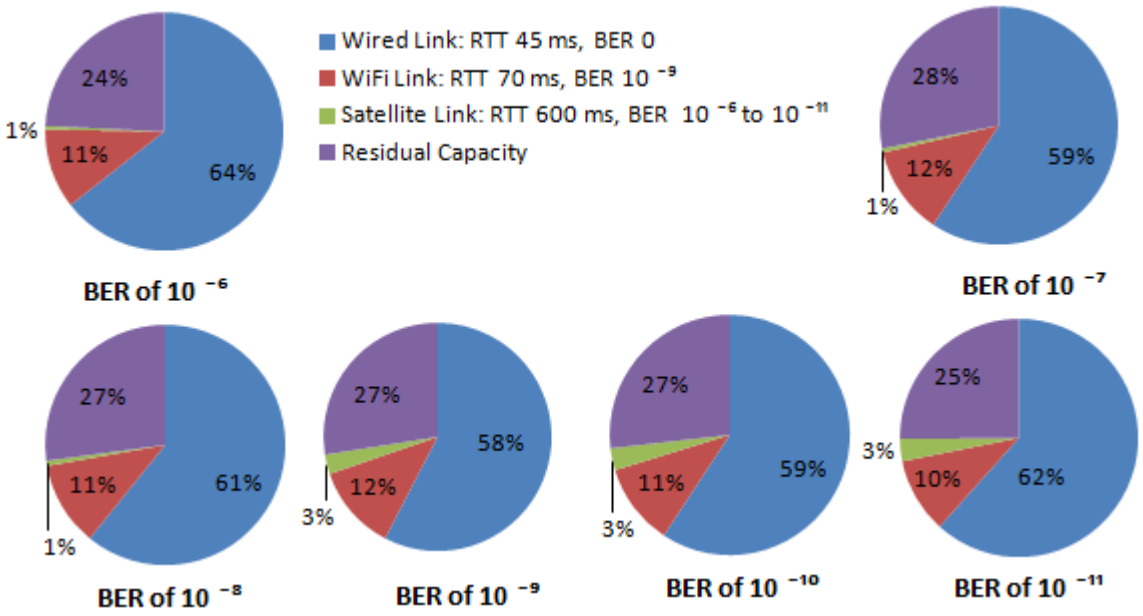


Figure 5.8: Link utilization of Westwood running on all 3 Senders

When ADYTIA along with PEP has been configured on router R1, all senders were using the default TCP variant as per the operating system of the host machine. All senders were sending the data using iperf to the corresponding receivers via R1 and R2 (i.e., where netem introduces the propagation delay and link losses). When BER was 10^{-6} and RTT of 600 ms on the satellite link, a corresponding sender (i.e., satellite-sender) initiate the connection using default TCP variant of the sender (i.e., in this case Cubic). Initially, the new connection from R1 to the corresponding receiver was started as per the default congestion control mechanism of the Linux kernel on R1 (i.e., Cubic). As per the algorithm of ADYTIA, it starts observing

the parameters and gets the link type from the CCM and application type from the ACM. Depending on the link and application type, AM checks Information Base and switch the congestion control algorithm. At 2 seconds, ADYTIA observed that the communication that was happening via satellite link have long RTT and available bandwidth was less than 10 Mbps. Hence, CCM of ADYTIA categorized the link as 'long thin connection'. As the data was generated using iperf, the application was categorized as default and based on link and application type, switching of congestion control algorithm was done from Cubic to Hybla. ADYTIA observed the link losses at 21 seconds with the estimated available bandwidth higher than 10 Mpbs and categorized the link as 'Long Fat Connection with Losses'. Hence, congestion control algorithm was switched from Hybla to Tarang as per the Information Base. Selection of the appropriate variant results in improved performance compared to Cubic as shown in Figure 5.6.

A wireless sender communicating with the corresponding receiver via wireless link along with other traffic on wired and satellite link (i.e., background traffic for the wireless link). When BER was 10^{-6} on the satellite link, ADYTIA categorized the wireless link as 'Low Speed-Low Delay Link with Losses' and application type as default in less than 1 seconds (i.e., 10 RTTs make 700 ms).

Thus, as per the Information Base, AM of ADYTIA switched the congestion control algorithm from Cubic (i.e., default variant) to Westwood. At 9 seconds, the connection was able to exploit higher than 100 Mbps of bottleneck link bandwidth and for the moment there were no losses. So, the link was categorized as 'High Speed Connection' with application type as default that triggered the switching of congestion control algorithm (CCA) from Westwood to HighSpeed. As CCM used the 100 Mbps as the threshold to categorize the link type and for the flow on the wireless link, bandwidth kept switching below and above 100 Mbps that triggered many changes of TCP variants.

On wired link (i.e., 45 ms RTT and negligible losses), in the presence of background traffic on wireless link (i.e., BER of 10^{-9}, 70 ms RTT) and satellite link (i.e., BER of 10^{-6}, 600 ms RTT) connection was started with Cubic. At 2 seconds, the link was categorized as 'High Speed' and ADYTIA switched the CCA from Cubic to HighSpeed variant. When all 3 flows were running in parallel as discussed above,

throughput achieved by satellite sender remained low in spite of selecting the appropriate variant due to following reasons:
i) Initially, the connection was started with Cubic and then switched to Hybla resulted in the lower utilization of the bottleneck bandwidth.
ii) Background traffic on the wired link (i.e., with 45 ms RTT) used the TCP variant HighSpeed (i.e., selected by ADYTIA) that captured the major fraction of the bottleneck bandwidth (because by 21 seconds, 45 ms RTT connection get about 466 RTTs). Thus, Tarang estimated the less remaining bandwidth on the bottleneck link that has been used to set the values of cwnd and ssthresh.
iii) Background traffic on wireless link was able to capture the bottleneck bandwidth in about 300 RTTs (i.e., the time at which Tarang was selected).

By looking at the graph of the Figure 5.6, it could be seen that the throughput of the satellite link has been increased as BER was reduced. When ADYTIA with PEP was configured, at BER values of 10^{-9} to 10^{-11} on satellite link, throughput achieved was in the range of 200 to 300 Mbps. Measured values of throughput depends on the following:
i)Selection of the TCP variants on all connections running in parallel.
ii)Fairness and friendliness properties of each selected variant.
iii)Information Base used in ADYTIA.

It is important to highlight that the objective of ADYTIA was to select the TCP variant based on the link and application types. In comparison to the Cubic, the performance of ADYTIA with PEP was significantly improved as shown in the Figure 5.6.

5.2.2.3 T-HetNet with FTP Client-Server

Testbed setup with FTP client and server to compare the performance of Cubic, and ADYTIA with PEP has been shown in Figure 5.9. Tools used for the testbed setup were discussed in Section 5.2.2. Three machines were configured as FTP client and other three machines were configured to work as FTP servers. Each pair of client-server communication was happening via three different types of links as follow:
i) The first link was emulated as a wired link (i.e., 45 ms RTT, negligible BER).
ii) The second link was emulated as a wireless link (i.e., 70 ms RTT, BER of 10^{-9}).

iii) The third link was emulated as a GEO satellite link (i.e., 600 ms RTT, BER varies from 10^{-6} to 10^{-11}).

The objective of this experiment was to investigate the advantage and capability of the ADYTIA as it allows the usage of TCP variant based on the link and application type. In the topology, as shown in Figure 5.9, two experiments were carried out:
i) In this experiment, all senders were configured to utilize the Cubic as the TCP variant and a file of 1500 MB was sent from all three senders to corresponding receivers.
ii) In this experiment, ADYTIA with PEP engine was used on the Router-1 and all senders were using the default variant (i.e., Cubic). A file of 1500 MB was sent from all senders.

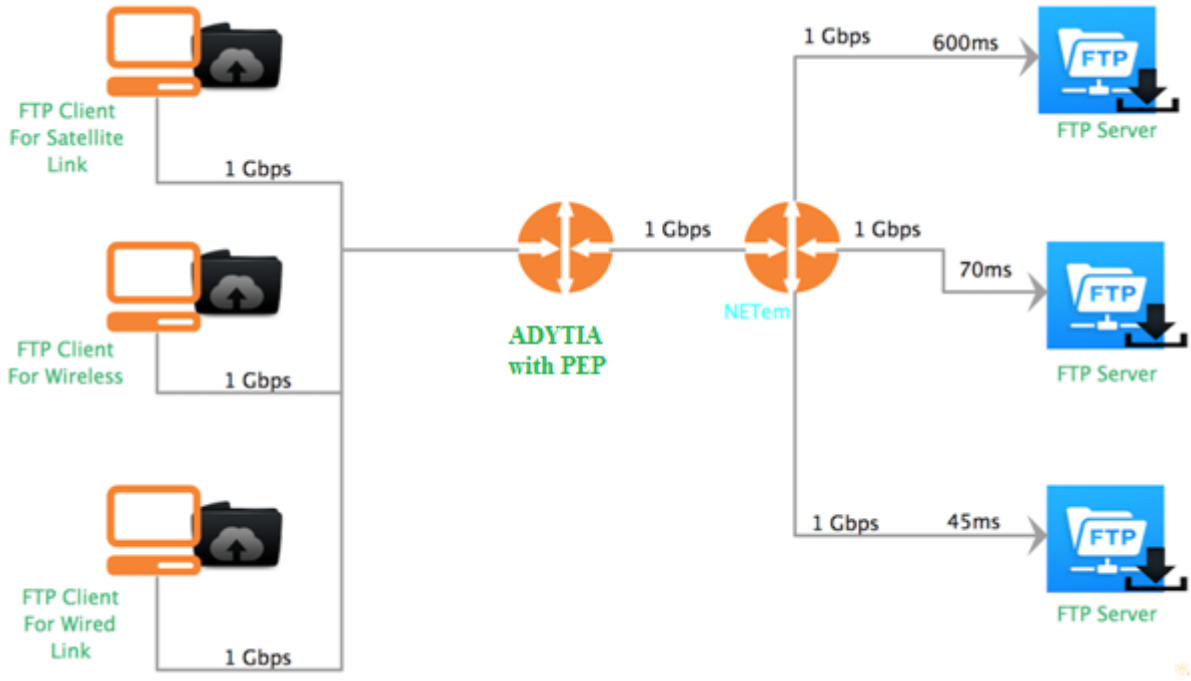
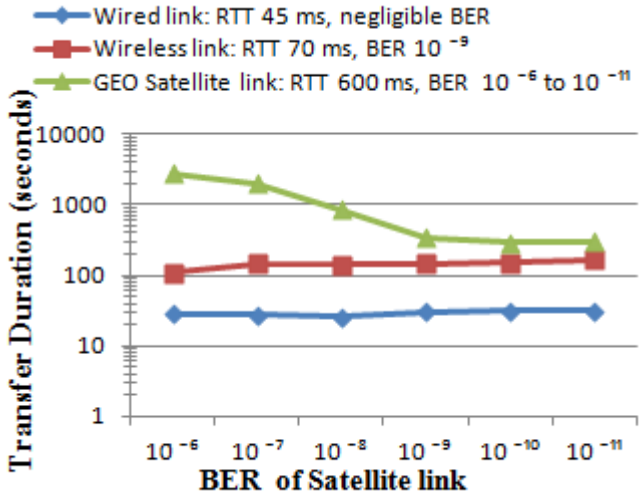
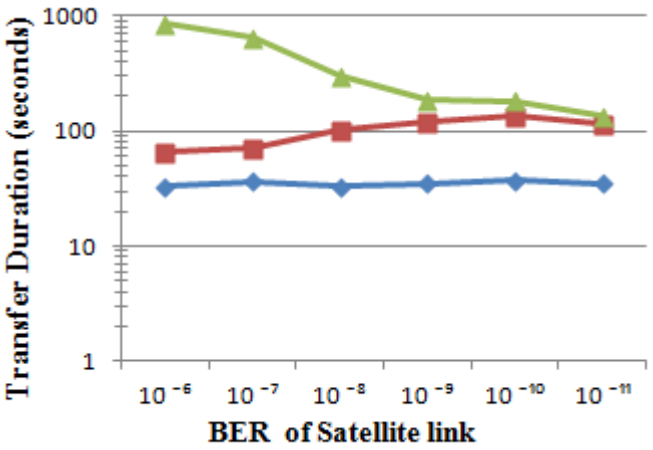


Figure 5.9: T-HetNet with FTP client and server

Performance comparison of Cubic, and ADYTIA with PEP in terms of file transfer duration when all three senders transmit the data to corresponding receivers has been shown in Figure 5.10. As can be seen in Figure 5.10-a, at BER of 10^{-6} on satellite link transfer duration was about 2600 seconds when Cubic was used by the sender. When ADYTIA with PEP engine was used on Router-1, this transfer duration reduced to 787 seconds as shown in Figure 5.10-b. It is the significant performance improvement which also gives good user experience.



(a) Performance of Cubic



(b) Performance of ADYTIA

Figure 5.10: Performance comparison of Cubic and ADYTIA with PEP for 1500 MB file

This improvement was achieved by the selection of the appropriate TCP variant depending on the link characteristics and application types. As can be seen from the Figure 5.10, there was a noticeable improvement in transfer duration for different values of BER when ADYTIA was used along with PEP engine. It is also important to observe that there was no need to alter the default variant of the sender machine's operating system.

5.2.3 Performance Evaluation on live Internet

Simulation results and testbed experiments' result demonstrated the advantages of having different congestion control algorithm based on link characteristics and application types. File transfer experiment performed on the Internet with existing TCP/IP implementations and ADYTIA with PEP helped to confirm the benefits. In this experiment, files of different sizes were transferred from our Institute laboratory setup to the Microsoft One-Drive server.

The route from the laboratory setup to the One-Drive server has been shown in Figure 5.11. RTT observed from the setup to the One-Drive server was around 280 to 300 ms. On the route from the sender machine to the One-Drive server, bottleneck link capacity was not known and probably varying with time. Objectives of the experiment were to see the ADYTIA's ability to change the congestion control algorithm dynamically and to compare the results with the existing TCP/IP implementations. To highlight the deployment feasibility and comparison with existing TCP variant, sender machine with Fedora operating system was used.

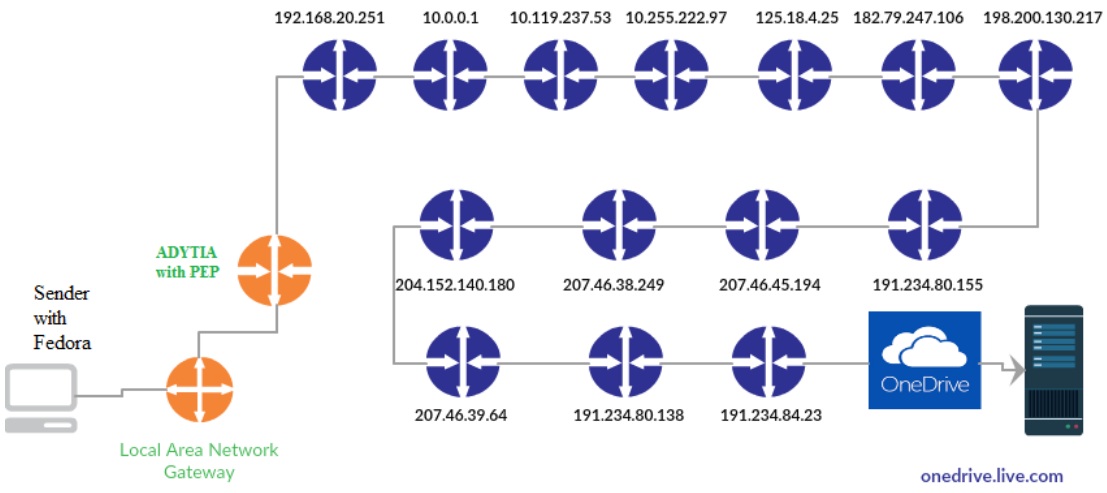


Figure 5.11: Experimental setup on live Internet

The result of the experiments carried out on the Internet for transferring the files of different sizes to the machine located 280-300 ms RTT away from the setup has been shown in Figure 5.12. Experiments were repeated 10 times with the same setup and results reported were the average of these 10 runs. It can be seen from the experimental results that a sender with Fedora 15 (using Cubic as the TCP variant)

took 49 minutes and 09 seconds to transfer a 3 GB file (i.e., throughput of 8.13 Mbps). The same file has been transferred in just 30 minutes 20 seconds (i.e., throughput of 13.18 Mbps) using the ADYTIA with PEP. Thus, improvement is by 62% and it was because of the appropriate selection of the TCP variant in ADYTIA.

Table 5.1: File transfer durations for live Internet experiments

File Size	**Sender with Fedora Operating System**	
	Transfer Duration (with Existing TCP/IP)	**Transfer Duration (with Proposed Architecture)**
10 KB	8 sec	1 sec
100 KB	9 sec	3 sec
1 MB	12 sec	3 sec
10 MB	21 sec	14 sec
100 MB	02:46 min	01:47 min
500 MB	10:29 min	07:25 min
1 GB	18:43 min	11:19 min
2 GB	36:54 min	23:47 min
3 GB	49:09 min	30:20 min

Initially, CCM detected the channel as 'low speed-low delay connection' with application type as an FTP, it selected the Westwood, and at the 16 seconds, it detected link as 'Long Fat Connection' and selected the TCP Hybla. For small file size transfer too, there was a significant advantage in terms of throughput as shown in Figure 5.12. File transfer durations for different files with varying sizes for Cubic and with the proposed approach in this research work has been shown in Table- 5.1.

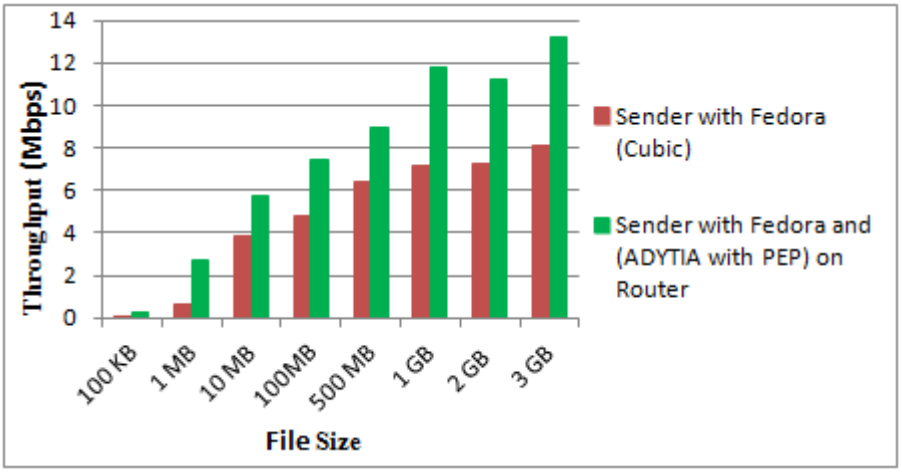


Figure 5.12: Throughput comparison of existing TCP/IP (in Fedora) and ADYTIA with PEP

5.3 Summary

ADYTIA has been integrated with PEP for deployment of this research work on live Internet. ADYTIA along with PEP has been tested on a T-HetNet testbed with extensive experiments using iperf (for artificial traffic generation with various profiles). Furthermore, T-HetNet has been configured with FTP client server application program and experimental analysis has been performed. In all these experiments, ADYTIA's dynamic nature allows the switching of TCP variant that results in 3-4 times performance improvement compared to the existing variants. Finally, ADYTIA along with PEP has been tested on live Internet by transferring the files of different sizes where it improved the performance by 62 % compared to Cubic (i.e., default variant of Linux).

The reason to utilize ADYTIA along with PEP was that end-user's kernel does not require any changes. Testbed experiments showed that the deployment of ADYTIA resulted in the improvement of throughput and good end user experience.

Results obtained in this chapter inspire towards the deployment of the proposed approach in the heterogeneous networks and especially on the Internet. Another benefit of the discussed approach in the thesis work was that the plugging in of any TCP variant available in the kernel to ADYTIA was very easy without requiring any modifications and recompilation. The conclusions of the research work done as a part of the thesis and future directions have been presented in next chapter.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The present research work incorporated an exhaustive study of various TCP variants for heterogeneous networks. These types of networks pose challenges to the transport layer protocols by offering different kinds of link characteristics. Literature survey and simulation studies depicted that the existing TCP variants have performance issues for variable RTT and BER links. Also, a fixed TCP variant is bound to the end users' operating system. This type of rigidity results in the poor performance of the TCP variants and hence user application does not work as per the application's expectations. In this research work, we have developed a protocol named as Tarang for a link with variable RTT and BER. This is primarily implemented based on bandwidth estimation and normalized round trip time on bottleneck link. Thereafter, a modified TCP/IP architecture has been designed and implemented for both simulation and testbed environments. This has been named as ADYTIA throughout this research work. It consists of four functional modules; CCM, ACM, AM and Information Base and incorporates the major working flow of the ADYTIA.

A wide range of experimental simulation has been done to establish that Tarang outperforms existing TCP variants including Cubic, the default variant of Linux based systems. On a satellite link (i.e., RTT $>$ 500ms, BER in the range of 10^{-6} to 10^{-11}) with single-hop and multi-hop configuration, the performance of Tarang has been significantly high compared to existing variants like Cubic, Hybla and Westwood. In heterogeneous networks for different link types and varying cross traffic characteris-

tics, JFI of Tarang was near by one with less fluctuation and showed the fairness; whereas other variants including Cubic have fairness issues. Although, Tarang outperformed the existing variants, it is equally important to address its existence amidst some new protocols in future; that may outperform it. Hence, ADYTIA was designed, since it enables the use of TCP variants based on link characteristics and application type.

Simulation results showed that ADYTIA was able to change the variant for each connection depending on the characteristics of the link and application layer protocol. This changing of variants per connection type, allowed for effective link utilization. The experimental results established that ADYTIA allowed the change of TCP variant with varying link characteristics and it always selected the best available TCP variant to maximize the link utilization. ADYTIA's performance was 62.65%, 82.87% better compared to default variant of Linux (i.e., Cubic) and Windows (i.e., Compound TCP) respectively, with varying link characteristics. This exhibited the dynamic nature of ADYTIA, that addresses to various application requirements. ADYTIA used the Information Base, which was prepared after rigorous literature survey to select the most suitable TCP variant. The Information Base of ADYTIA allowed any newly designed TCP variant to be plugged-in without disturbing the entire framework. Further, to demonstrate this feature, Tarang was plugged-in with ADYTIA, and a comprehensive analysis of the result was presented, so as to conclude its adaptive (plug-in) and dynamic (change of variant at runtime) behavior. Apart from the simulation study, ADYTIA was also implemented for a testbed environment by embedding its modules in Linux kernel. The results of the testbed also established that ADYTIA has been enabled with a smooth switching between protocols based on link types and application types.

The research work also explored the need to handle the fairness property of ADYTIA, in managing complex heterogeneous networks and for the ease of deployment. ADYTIA was integrated with PEP for deployment of this research work on live Internet. ADYTIA along with PEP was tested on a T-HetNet testbed with extensive experiments using iperf (for artificial traffic generation with various profiles). Furthermore, T-HetNet was configured with FTP client server application program and experimental analysis performed. In all these experiments, ADYTIA's dynamic

nature allowed the switching of TCP variant that results in 3-4 times performance improvement compared to the existing variants. Finally, ADYTIA was tested on live Internet by transferring the files of different sizes where it reduced the transfer duration significantly compared to Linux based systems (with Cubic variant).

6.2 Future Directions

Looking at the global technological scenarios and increasing number of users with hand-held devices, four future extensions of this research work are being proposed. One of the extension is related to the fact that ADYTIA can be extended for smart device operating systems. Further, ADYTIA along with PEP can also be ported to a single box as a TCP/IP module and can be used as an external device to enhance the performance. Another extension of the work could be that ADYTIA along with PEP can be integrated with cloud-based services, and can be modeled as a network service. Registered users' connections would always pass through a server where ADYTIA with PEP would be implemented to give efficient link utilization and good user experience. Furthermore, an enormous growth of mobile traffic is in place due to the widespread popularity of Internet of things (IoT), smart devices, and laptops. Most of these devices communicate with each other using heterogeneous links with constraints on the parameters such as latency, throughput, and interference from concurrent transmissions. ADYTIA can offer the good user experience and effective link utilization for these technologies. Hence, it is further concluded that the research work has scope for adaptability with the changing technological advancements and protocol development in future generations, giving its ability to effectively handle heterogeneous environments.

Appendix A

Route Configuration

Manual configurations of network for testbed setup T-HetNet (elaborated in Chapter 5) took long time before starting experiments. Obvious solution was to create script of automatically configuration. For this purposes we used shell scripting language. Configuration, routing rules, and firewall rules for each machine used for configuration of testbed are presented below.

A.1 Satellite Sender

This machine is communicating with the receiver via a link with high RTT and variable BER.

iptables -F

iptables -X

iptables -t nat -F

iptables -t nat -X

iptables -t mangle -F

iptables -t mangle -X

route add -host 10.10.108.196 gw 10.10.108.196

route add -host 10.10.108.197 gw 10.10.108.196

route add -host 10.10.108.199 gw 10.10.108.196

route del -net default netmask 0.0.0.0

route del -net link-local netmask 255.255.0.0

route del -net 10.0.0.0 netmask 255.0.0.0

route del -net 169.254.0.0 netmask 255.255.0.0

route del -net 192.168.122.0 netmask 255.255.255.0

ip link set eth1 multicast off

ip link set eth1 promisc on

A.2 Wireless Sender

This machine is communicating with the receiver via a link with low RTT and average BER.

iptables -F

iptables -X

iptables -t nat -F

iptables -t nat -X

iptables -t mangle -F

iptables -t mangle -X

route add -host 10.10.108.196 gw 10.10.108.196

route add -host 10.10.108.197 gw 10.10.108.196

route add -host 10.10.108.200 gw 10.10.108.197

route del -net default netmask 0.0.0.0

route del -net link-local netmask 255.255.0.0

route del -net 10.0.0.0 netmask 255.0.0.0

route del -net 169.254.0.0 netmask 255.255.0.0

route del -net 192.168.122.0 netmask 255.255.255.0

ip link set eth0 multicast off

ip link set eth0 promisc on

A.3 Wired Sender

This machine is communicating with the receiver via a link with low RTT and negligible BER.

iptables -F

iptables -X

iptables -t nat -F

iptables -t nat -X

iptables -t mangle -F

iptables -t mangle -X

iptables -P INPUT ACCEPT

iptables -P FORWARD ACCEPT

iptables -P OUTPUT ACCEPT

route add -host 10.10.108.196 gw 10.10.108.196

route add -host 10.10.108.197 gw 10.10.108.196

route add -host 10.10.108.201 gw 10.10.108.196

route del -net default netmask 0.0.0.0

route del -net link-local netmask 255.255.0.0

route del -net 10.0.0.0 netmask 255.0.0.0

route del -net 192.168.122.0 netmask 255.255.255.0

ip link set eth0 multicast off

ip link set eth0 promisc on

A.4 Router R1

This is a linux based machine configured to connect all senders and Router R2.

iptables -F

iptables -X

iptables -t nat -F

iptables -t nat -X

iptables -t mangle -F

iptables -t mangle -X

iptables -P INPUT ACCEPT

iptables -P FORWARD ACCEPT

iptables -P OUTPUT ACCEPT

route add -host 10.10.108.197 gw 10.10.108.197

route add -host 10.10.108.199 gw 10.10.108.197

route add -host 10.10.108.200 gw 10.10.108.197

route add -host 10.10.108.201 gw 10.10.108.197

route add -host 10.10.108.193 gw 10.10.108.193

route add -host 10.10.108.194 gw 10.10.108.194

route add -host 10.10.108.195 gw 10.10.108.195

route del -net default netmask 0.0.0.0

route del -net link-local netmask 255.255.0.0

route del -net 10.0.0.0 netmask 255.0.0.0

route del -net 192.168.122.0 netmask 255.255.255.0

ip link set eth0 multicast off

ip link set eth0 promisc on

A.5 Router R2

This is a linux based machine configured to connect all receivers and Router R1.

iptables -F iptables -X

iptables -t nat -F

iptables -t nat -X

iptables -t mangle -F

iptables -t mangle -X

iptables -P INPUT ACCEPT

iptables -P FORWARD ACCEPT

iptables -P OUTPUT ACCEPT

route add -host 10.10.108.199 gw 10.10.108.199

route add -host 10.10.108.200 gw 10.10.108.200

route add -host 10.10.108.201 gw 10.10.108.201

route add -host 10.10.108.196 gw 10.10.108.196

route add -host 10.10.108.193 gw 10.10.108.196

route add -host 10.10.108.194 gw 10.10.108.196

route add -host 10.10.108.195 gw 10.10.108.196

route del -net default netmask 0.0.0.0

route del -net link-local netmask 255.255.0.0

route del -net 10.0.0.0 netmask 255.0.0.0

route del -net 192.168.122.0 netmask 255.255.255.0

ip link set eth0 multicast off

ip link set eth0 promisc on

A.6 Satellite Receiver

This machine required to receive the data sent by satellite sender via Router R1 and Router R2.

iptables -F

iptables -X

iptables -t nat -F

iptables -t nat -X

iptables -t mangle -F

iptables -t mangle -X

route add -host 10.10.108.197 gw 10.10.108.197

route add -host 10.10.108.196 gw 10.10.108.197

route add -host 10.10.108.193 gw 10.10.108.197

route del -net default netmask 0.0.0.0

route del -net link-local netmask 255.255.0.0

route del -net 10.0.0.0 netmask 255.0.0.0

route del -net 192.168.122.0 netmask 255.255.255.0

ip link set eth1 multicast off

ip link set eth1 promisc on

A.7 Wireless Receiver

This machine required to receive the data sent by wireless sender via Router R1 and Router R2.

iptables -F

iptables -X

iptables -t nat -F

iptables -t nat -X

iptables -t mangle -F

iptables -t mangle -X

route add -host 10.10.108.197 gw 10.10.108.197

route add -host 10.10.108.196 gw 10.10.108.197

route add -host 10.10.108.194 gw 10.10.108.197

route del -net default netmask 0.0.0.0

route del -net link-local netmask 255.255.0.0

route del -net 10.0.0.0 netmask 255.0.0.0

route del -net 192.168.122.0 netmask 255.255.255.0

ip link set eth1 multicast off

ip link set eth1 promisc on

A.8 Wired Receiver

This machine required to receive the data sent by wired sender via Router R1 and Router R2.

sysctl -p

iptables -F

iptables -X

iptables -t nat -F

iptables -t nat -X

iptables -t mangle -F

iptables -t mangle -X

route add -host 10.10.108.197 gw 10.10.108.197

route add -host 10.10.108.196 gw 10.10.108.197

route add -host 10.10.108.195 gw 10.10.108.197

route del -net default netmask 0.0.0.0

route del -net link-local netmask 255.255.0.0

route del -net 10.0.0.0 netmask 255.0.0.0

route del -net 192.168.122.0 netmask 255.255.255.0

ip link set eth1 multicast off

ip link set eth1 promisc on

Appendix B

Linux Kernel Commands

B.1 Steps to insert new TCP variant in Linux kernel

- To install any modules into kernel it is required to make entry in a makefile available at path usr/src/linux version/net/ipv4 obj-m + = module_name.o
- Copy the module in a kernel at path /usr/src/linux version/net/ipv4/
- It is required to recompile kernel every time when we make any change into 'c' file.
- Kernel recompilation create/update '.o' and '.ko' file into kernel using command:
  make
  make modules_install
- Insert complied module into kernel using command
  insmod modules_name.ko
- To enable the module use command sysctl -w net.ipv4.tcp_congestion_control=module_name

B.2 Useful Commands

route - to get route information for a machine or router.

traceroute [IP address]- to get path information for the specifif IP address.

echo 1 > /proc/sys/net/ipv4/ip_forward- to configure each machine to perform routing.

sysctl -w net.ipv4.tcp_congestion_control=TCP variant (e.g. cubic) - To set congestion control algorithm to specific variant.

sysctl net.ipv4.tcp_available_congestion_control- To check available congestion control algorithms.

cat /proc/sys/net/ipv4/tcp_congestion_control- To check current congestion control algorithm.

gedit messages/log/var/ - To view the kernel log.

ethtool [interface name]- To see network card information.

netstat -i - To check promiscuous mode of NIC.

ifconfig [interface name] txtqueuelen 100000 - To set txtqueue length.

ethtool -s [interface name] speed 1000 duplex full autoneg on- To set ethernet speed or mode.

tc qdisc del dev [interface name] root- To remove tc root.

Appendix C

PEP Configurations

- Install libnetfilter_queue (version 0.0.17) and libnfnetlink (version 1.0.1) /configure, make and make install.

- Libnetfilter installation required to setup following path where .pc file exist
  export PKG_CONFIG_PATH = /user/local/lib/pkgconfig

- Install PEPsal-2.0.1 on router using command. /configure, make and make install. In case of any error execute following:

  export CFLAGS= -march = i686
  make distclean
  make

- Fire wall rules:
  iptables -N TCP_OPTIMIZATION -t nat
  iptables -N TCP_OPTIMIZATION -t mangle
  SAT_RECV="10.10.108.0/24"
  NQ=9
  OUT_IFACE="eth1"
  iptables -t mangle -F
  iptables -t nat -F
  iptables -t nat -F TCP_OPTIMIZATION
  iptables -t mangle -F TCP_OPTIMIZATION

iptables -t mangle -I TCP_OPTIMIZATION -i eth1 -s 10.10.108.0/24 -p tcp -j NFQUEUE –queue-num=9

iptables -t nat -A POSTROUTING -s $SAT_RECV -o $OUT_IFACE -j MASQUERADE

iptables -t nat -I TCP_OPTIMIZATION -i eth1 -s 10.10.108.0/24 -p tcp -j REDIRECT –to-port 6009

- Set Library path using command:
  export LD_LIBRARY_PATH =/user/local/lib

- Run PEPsal using command:
  Pepsal -v -q 9 -p 6009 -l logfilename
  If error occurs like 'can not find the connection in SYN table' while running Pepsal do the changes at PEPsal/include/syntab.h as follow:
  struct syntab_key {
  int addr;
  unsigned short port;
  } __attribute__((packed));

- PEP logger do entry of all connections in the syn table to the file specified by filename at every PEPLOGGER_INTERVAL seconds. To change time for log one can modify value of PEPLOGGER_INTERVAL in pepdefs.h as
  #define PEPLOGGER_INTERVAL (1 * 60)

Appendix D

Linux Routing and Traffic Control

Traffic control (tc) is part of the Linux iproute2 package which allows the user to access networking features. The package itself has three main features: monitoring the system, traffic classification, and traffic manipulation. The tc part in the package can be used to configure qdiscs, and packet classification into qdiscs.

D.1 tc qdiscs and classes

Queuing Discipline (qdisc) put packets in queue with an algorithm that decides when to send which packet. Following is the types of qdisc:
Classless qdisc: qdisc with no configurable internal subdivision.
Classfull qdisc: qdisc that may contain classes. Classfull qdiscs allow packet classification.
Root qdisc: a root qdisc is attached to each network interface which is either classfull or classless.
egress qdisc: works on outgoing traffic.
ingress qdisc: works on incoming traffic.
Class: classes either contain other classes, or a qdisc is attached.
Filter: classification can be performed using filters

D.2 General Commands

Following commands are used in textbed setup:
Generate a root qdisc-
tc qdisc add dev DEV handle 1: root QDISC [PARAMETER]

Generate a non-root qdisc-
tc qdisc add dev DEV parent PARENTID handle HANDLEID QDISC [PARAMETER]
Generate a class-
tc class add dev DEV parent PARENTID classid CLASSID QDISC [PARAMETER]
DEV: interface at which packets leave, e.g. eth1
PARENTID: id of the class to which the qdisc is attached e.g. X: Y
HANDLEID: unique id, by which this qdisc is identified (e.g. X).
CLASSID: unique id, by which this class can be identified (e.g. X,Y).
QDISC: type of the qdisc attached
PARAMETER: parameter specific to the qdisc attached.

D.3 Building a qdisc Tree

By default each interface has one egress (outgoing) FIFO qdisc (queuing discipline). To be able to treat some packets different than others, a hierarchy of qdiscs can be constructed. Furthermore, different kinds of qdiscs exist, each with different properties and parameters that can be tuned. To build a tree, a classfull root qdisc has to be chosen. In this example HTB (Hierarchical Token Bucket) is used, since the other qdiscs are either classless or prioritize some traffic (e.g. PRIO) or are too complicated (e.g. CBQ). At the leaves a classless qdisc can be attached. Following is an example:
First the default root qdisc is replaced:
tc qdisc add dev eth1 handle 1: root htb
Then one root class and three children classes are created:
tc class add dev eth1 parent 1: classid 1:1 htb rate 1000Mbps
tc class add dev eth1 parent 1:1 classid 1:11 htb rate 100Mbps
tc class add dev eth1 parent 1:1 classid 1:12 htb rate 1000Mbps
tc class add dev eth1 parent 1:1 classid 1:13 htb rate 1000Mbps
The parent id is equal to the classid of the respective parent. The children's class ids have to have the same major number (number before the colon) as their parent and a unique minor number (number after the colon). The qdisc is HTB with a maximal rate of 1000 Mbps.
tc qdisc add dev eth1 parent 1:11 handle 10: netem delay 600ms

tc qdisc add dev eth1 parent 1:12 handle 20: netem delay 70ms
tc qdisc add dev eth1 parent 1:13 handle 30: netem delay 45ms
tc filter add dev eth1 protocol ip parent 1:0 prio 3 u32 match ip dst 10.10.108.199 flowid 1:11
tc filter add dev eth1 protocol ip parent 1:0 prio 3 u32 match ip dst 10.10.108.200 flowid 1:12
tc filter add dev eth1 protocol ip parent 1:0 prio 3 u32 match ip dst 10.10.108.201 flowid 1:13

The parent id is the id of the class to which the qdisc is attached. The handle is a unique identifier. Netem is chosen as a qdisc. Unique numbers must be unique within an interface.

The commands for changing and deleting qdiscs have the same structure as the add command. Parameters of qdisc can be adapted using the change command. To change the 600ms delay from the qdisc with handles 10: (from the previous example) to 200ms, the following command is used:
tc qdisc change dev eth1 parent 1:11 handle 10: netem delay 200ms
To delete a complete qdisc tree only the root needs to be deleted:
tc qdisc del dev eth1 root
It is also possible to delete only a particular qdisc:
tc qdisc del dev eth1 parent 1:11 handle 10:

Appendix E

Iperf

Iperf was originally developed by NLANR/DAST as a modern alternative for measuring TCP and UDP bandwidth performance. Iperf is a tool to measure maximum TCP bandwidth, allowing the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay jitter, datagram loss.

E.1 Iperf Features

TCP

- Measure bandwidth.
- Report MSS/MTU size and observed read sizes.
- Support for TCP window size via socket buffers.
- Multi-threaded if pthreads or win32 threads are available. Client and server can have multiple simultaneous connections.

UDP

- Client can create UDP streams of specified bandwidth.
- Measure packet loss
- Measure jitter
- Multicast capable

- Multi-threaded if pthreads are available. Client and server can have multiple simultaneous connections (This doesn't work in Windows).

Where appropriate, options can be specified with K (kilo-) and M (mega-) suffices. So 128K instead of 131072 bytes. Ipref can run for specified time, rather than fixed amount of data transfer. It picks the best units for the size of data being reported. Iperf server handles multiple connections, rather than quitting after a single test. It prints intermediate bandwidth, jitter, and loss reports at specified intervals. Typical Iperf output contains a timestamped report of the amount of data transferred and the throughput measured. By default, the Iperf client connects to the Iperf server on the TCP port 5001 and the bandwidth displayed by Iperf is the bandwidth from the client to the server.

Works Cited

Abdeljaouad, I, et al. "Performance analysis of modern TCP variants: A comparison of Cubic, Compound and New Reno." *Communications (QBSC), 2010 25th Biennial Symposium on*. IEEE, 2010. 80–83.

Akyildiz, Ian F, Giacomo Morabito, and Sergio Palazzo. "TCP-Peach: a new congestion control scheme for satellite IP networks." *IEEE/ACM Transactions on Networking (ToN)* 9.3 (2001): 307–321.

Akyildiz, Ian F, Xin Zhang, and Jian Fang. "TCP-Peach+: Enhancement of TCP-Peach for satellite IP networks." *IEEE Communications letters* 6.7 (2002): 303–305.

Allcock, William, et al. "GridFTP: Protocol extensions to FTP for the Grid." *Global Grid ForumGFD-RP* 20 (2003): 1–21.

Arianfar, Somaya. "TCP's Congestion Control Implementation in Linux Kernel." *Proceedings of Seminar on Network Protocols in Operating Systems*. 2012. 16.

Armitage, Grenville. *Quality of service in IP networks*. Sams, 2000.

Atxutegi, Eneko, et al. "TCP behaviour in LTE: Impact of flow start-up and mobility." *Wireless and Mobile Networking Conference (WMNC), 2016 9th IFIP*. IEEE, 2016. 73–80.

Baiocchi, Andrea, Angelo P Castellani, and Francesco Vacirca. "YeAH-TCP: yet another highspeed TCP." *Proc. PFLDnet*. 2007. 37–42.

Barakat, Chadi, Eitan Altman, and Walid Dabbous. "On TCP performance in a heterogeneous network: a survey." *Communications Magazine, IEEE* 38.1 (2000): 40–46.

Belshe, Mike. "More bandwidth doesnt matter (much)." *Google Inc* (2010).

Bhatt, HS, et al. "GridTCP: A transport layer data transfer protocol for satellite based grid computing." *Proceedings of WoNGeN05, International Workshop on Next Generation Wireless Networks*. 2005. 18–21.

Brakmo, Lawrence S. and Larry L Peterson. "TCP Vegas: End to end congestion avoidance on a global Internet." *Selected Areas in Communications, IEEE Journal on* 13.8 (1995): 1465–1480.

Caini, Carlo and Rosario Firrincieli. "TCP Hybla: a TCP enhancement for heterogeneous networks." *International journal of satellite communications and networking* 22.5 (2004): 547–566.

Caini, Carlo, Rosario Firrincieli, and Daniele Lacamera. "Comparative performance evaluation of tcp variants on satellite environments." *Communications, 2009. ICC'09. IEEE International Conference on*. IEEE, 2009. 1–5.

Caini, Carlo, et al. "Implementation and Analysis of the TCP Adaptive-Selection Concept in ns-2 and Linux." *Advanced Satellite Mobile Systems, 2008. ASMS 2008. 4th*. IEEE, 2008. 198–203.

Carlo, Caini, Rosario Firrincieli, and Daniele Lacamera. "The TCP Adaptive-Selection Concept." *Systems Journal, IEEE* 2.1 (2008): 83–89.

Carlo.Caini, Rosario Firrincieli, and Daniele Lacamera. "PEPsal: a Performance Enhancing Proxy for TCP satellite connections." *IEEE Aerospace and Electronic Systems Magazine* 22.8 (2007): 7–16.

Carson, Mark and Darrin Santay. "NIST Net: a Linux-based network emulation tool." *ACM SIGCOMM Computer Communication Review* 33.3 (2003): 111–126.

Chen, Yan, Toni Farley, and Nong Ye. "QoS requirements of network applications on the Internet." *Information, Knowledge, Systems Management* 4.1 (2004): 55–76.

Chen, Yung-Chih, et al. "Measuring cellular networks: Characterizing 3g, 4g, and path diversity." *Annual Conference of International Technology Alliance*. 2012.

Comer, Douglas E. *Internetworking con TCP/IP*. Vol. 1. Pearson Italia Spa, 2006.

Dordal, Peter L. "An Introduction to Computer Networks." (2014).

Fall, Kevin and Sally Floyd. "Simulation-based comparisons of Tahoe, Reno and SACK TCP." *ACM SIGCOMM Computer Communication Review* 26.3 (1996): 5–21.

Fall, Kevin and Kannan Varadhan. "The network simulator (ns-2)." *URL: http://www. isi. edu/nsnam/ns* (2007).

Floyd, S, et al. "Tcp selective acknowledgment options." (1996).

Floyd, Sally. "Metrics for the Evaluation of Congestion Control Mechanisms", RFC 5166." (2008).

Floyd, Sally and Tom Henderson. "RFC 2582: The NewReno modification to TCP's fast recovery algorithm." *IETF, April* (1999).

Fu, Cheng Peng and Soung C Liew. "TCP Veno: TCP enhancement for transmission over wireless access networks." *Selected Areas in Communications, IEEE Journal on* 21.2 (2003): 216–228.

Grieco, Luigi A and Saverio Mascolo. "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control." *ACM SIGCOMM Computer Communication Review* 34.2 (2004): 25–38.

Grigorik, Ilya. *High Performance Browser Networking: What every web developer should know about networking and web performance.* " O'Reilly Media, Inc.", 2013.

Ha, Sangtae, Injong Rhee, and Lisong Xu. "CUBIC: a new TCP-friendly high-speed TCP variant." *ACM SIGOPS Operating Systems Review* 42.5 (2008): 64–74.

Ha, Sangtae, et al. "A step toward realistic performance evaluation of high-speed TCP variants." *Fourth International Workshop on Protocols for Fast Long-Distance Networks*. 2006.

Hemminger, Stephen, et al. "Network emulation with NetEm." *Linux conf au*. 2005. 18–23.

https://iperf.fr/.

http://www.isi.edu/nsnam/ns/.

Issariyakul, Teerawat and Ekram Hossain. *Introduction to network simulator NS2*. Springer Science & Business Media, 2011.

Jain, Rahul and Teunis J Ott. "Design and implementation of split TCP in the linux kernel." Diss. New Jersey Institute of Technology, Department of Computer Science, 2007.

Jitendra, Padhye, et al. "Modeling TCP throughput: A simple model and its empirical validation." *ACM SIGCOMM Computer Communication Review* 28.4 (1998): 303–314.

Johansson, Ingemar. "Congestion control for 4G and 5G access." *Internet Engineering Task Force, Internet-Draft draft-johansson-cc-for-4g-5g-00* (2015).

Kapoor, Rohit, et al. "Capprobe: A simple and accurate capacity estimation technique." *ACM SIGCOMM Computer Communication Review* 34.4 (2004): 67–78.

Kelly, Tom. "Scalable TCP: Improving performance in highspeed wide area networks." *ACM SIGCOMM computer communication Review* 33.2 (2003): 83–91.

Kuzmanovic, Aleksandar and Edward W Knightly. "TCP-LP: A distributed algorithm for low priority data transfer." *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*. IEEE, 2003. 1691–1701.

Lakshman, TV and Upamanyu Madhow. "The performance of TCP/IP for networks with high bandwidth-delay products and random loss." *Networking, IEEE/ACM Transactions on* 5.3 (1997): 336–350.

Leung, Kui-Fai and Kwan Lawrence Yeung. "TCP-swift: an end-host enhancement scheme for TCP over satellite IP networks." *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*. IEEE, 2004. 551–555.

Liu, Ke and Jack YB Lee. "Mobile accelerator: A new approach to improve TCP performance in mobile data networks." *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*. IEEE, 2011. 2174–2180.

Liu, Shao, Tamer Başar, and Ravi Srikant. "TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks." *Performance Evaluation* 65.6 (2008): 417–440.

Mascolo, Saverio, et al. "Performance evaluation of Westwood+ TCP congestion control." *Performance Evaluation* 55.1 (2004): 93–111.

Miras, Dimitrios, Martin Bateman, and Saleem Bhatti. "Fairness of high-speed TCP stacks." *22nd International Conference on Advanced Information Networking and Applications*. IEEE, 2008. 84–92.

Padhye, Jitendra, et al. "Modeling TCP Reno performance: a simple model and its empirical validation." *IEEE/ACM Transactions on Networking (ToN)* 8.2 (2000): 133–145.

Padmanabhan, Venkata N and Randy H Katz. "TCP fast start: A technique for speeding up web transfers." (1998).

Peng, Fei, Lijuan Wu, and Victor Leung. "Cross-layer enhancement of TCP split-connections over satellites links." *International Journal of Satellite Communications and Networking* 24.5 (2006): 405–418.

Pirovano, Alain and Fabien Garcia. "A new survey on improving TCP performances over geostationary satellite link." *Network and Communication Technologies* 2.1 (2013): pp–xxx.

Sally, Floyd. "HighSpeed TCP for large congestion windows." (2003).

Sarkar, Niladri and Jairo Gutie´rrez. "Revisiting the issue of the credibility of simulation studies in telecommunication networks: highlighting the results of a comprehensive survey of IEEE publications." *Communications Magazine, IEEE* 52.5 (2014): 218–224.

Saverio, Mascolo, et al. "TCP westwood: Bandwidth estimation for enhanced transport over wireless links." *Proceedings of the 7th annual international conference on Mobile computing and networking*. ACM, 2001. 287–297.

Scharf, Michael, Simon Hauger, and Jochen Kögel. "Quick-Start TCP: From theory to practice." *Proc. 6th Intern. Workshop on Protocols for FAST Long-Distance Networks (PFLDnet), Manchester*. 2008.

Sun, Zhili. *Satellite networking: principles and protocols*. John Wiley & Sons, 2005.

Tian, Ye, Kai Xu, and Nirwan Ansari. "TCP in wireless environments: problems and solutions." *Communications Magazine, IEEE* 43.3 (2005): S27–S32.

Wang, Fugui, Prasant Mohapatra, and Sarit Mukherjee. "An application based differentiated service model." *Networks, 2000.(ICON 2000). Proceedings. IEEE International Conference on*. IEEE, 2000. 424–430.

Wang, Ruhai, et al. "Protocols for reliable data transport in space Internet." *Communications Surveys & Tutorials, IEEE* 11.2 (2009): 21–32.

Wei, David X and Pei Cao. "NS-2 TCP-Linux: an NS-2 TCP implementation with congestion control algorithms from Linux." *Proceeding from the 2006 workshop on ns-2: the IP network simulator*. ACM, 2006. 9.

Wei, David X, et al. "FAST TCP: motivation, architecture, algorithms, performance." *IEEE/ACM transactions on Networking* 14.6 (2006): 1246–1259.

Wong, Gary T, Matti Hiltunen, Richard D Schlichting, et al. "A configurable and extensible transport protocol." *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. IEEE, 2001. 319–328.

Wu, Xiuchao, et al. "Sync-tcp: A new approach to high speed congestion control." *Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on*. IEEE, 2009. 181–192.

Xiuchao, Wu. "Improving TCP Performance in the Mobile, High Speed, Heterogeneous and Evolving Internet." Diss. 2009.

Xiuchao, Wu, et al. "Utilizing characteristics of last link to improve TCP performance." *Performance, Computing, and Communications Conference, 2005. IPCCC 2005. 24th IEEE International*. IEEE, 2005. 207–214.

Xu, Lisong, Khaled Harfoush, and Injong Rhee. "Binary increase congestion control (BIC) for fast long-distance networks." *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. IEEE, 2004. 2514–2524.

Yang, Peng, et al. "TCP congestion avoidance algorithm identification." *Networking, IEEE/ACM Transactions on* 22.4 (2014): 1311–1324.

Yin, Hao, et al. "Content delivery networks: a bridge between emerging applications and future IP networks." *Network, IEEE* 24.4 (2010): 52–56.

Zhang, Menglei, et al. "Transport layer performance in 5G mmWave cellular." *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*. IEEE, 2016. 730–735.