CROSS-LAYER AWARE TRANSPORT PROTOCOLS FOR WIRELESS NETWORKS

by

SUMATHI GOPAL

A Dissertation submitted to the

Graduate School-New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Prof. Dipankar Raychaudhuri

and approved by

____

____

____

____

____

New Brunswick, New Jersey

October, 2007

ABSTRACT OF THE DISSERTATION

Cross Layer Aware Transport Protocols for Wireless Networks

by Sumathi Gopal

Dissertation Director:

Prof. Dipankar Raychaudhuri

This dissertation addresses the problem of reliable file transfer over single-hop and multi-hop shared-media wireless networks which are generally characterized by fluctuating bandwidth and error characteristics. Traditional reliable file transport protocols such as TCP assume relatively slow-varying links and were not generally designed to deal with interference problems of shared media wireless networks. The large performance gap between unreliable UDP and reliable TCP motivates the investigation of new transport protocols that might achieve significantly faster file transfer than TCP on wireless media.

CLAP – a Cross Layer Aware transport Protocol has been developed as a general solution for reliable file transfer, with decoupled flow control and error control to accommodate time-varying links. Error control in CLAP was designed to minimize interference and round-trip time estimation. Flow control in the proposed transport protocol leverages MAC status information via a novel cross-layer software framework (CLF), developed to provide systematic access to intra-node and inter-node status information.

Single hop evaluations, which consider an 802.11b wireless LAN with wired backhaul, were carried out using both NS2 simulations and ORBIT test-bed experiments. In time-varying, high loss scenarios, TCP shuts down operation without MAC retries, while an early CLAP version (CLAP-*beta*) achieves over 68% of upper-bound UDP performance. In noise-free scenarios, a "skip-ACKs" TCP modification to reduce interference achieves limited gains since TCP flow control depends on regular ACKs, while CLAP-beta approaches peak UDP performance by fully using the bandwidth available.

Multi-hop evaluations with NS2 simulations consider a 3-hop primary path in a 4x4 wireless mesh over 802.11b single-channel interfaces. Occasional background flows and on-off channel noise injection produce bandwidth and error fluctuations. These simulations expose the general multi-hop wireless problem where self interference in the forward path significantly reduces end-to-end bandwidth. Increasing interference and random packet losses tend to degrade TCP performance even more significantly than in 1-hop scenarios. Here, CLAP-*final* with improvements (relative to CLAP-*beta*) to reduce dependence on RTT estimation achieves over 90% of UDP performance in a variety of time-varying conditions.

This thesis demonstrates the efficacy of reliable file transfer using CLAP to address interference and time-varying links in both single- and multi-hop wireless network scenarios. Future research opportunities include cross-layer techniques for error control, efficient inter-node protocols for CLF, and tighter integration with mesh network routing protocols.

Dedication

This dissertation is dedicated to

Vinay, my husband and best friend

For believing in me and nurturing my ambitions

Acknowledgements

Being a student at WINLAB has been a remarkable experience. The abundant knowledge, high-quality research and friendly environment helped me get over the fears of systems engineering.

I wish to sincerely thank my advisor and mentor Prof Dipankar Raychaudhuri for taking me under his guidance and identifying my potential to do systems research, long before I knew I had it. He introduced me to this challenging and fascinating research problem of data transport over wireless networks in the first month of my PhD career. His unparalleled insight of wireless and systems helped make the right design choices, and develop the various evaluation scenarios. He taught by example how to develop a top-down approach to a research problem. He helped develop a positive outlook and "use our strengths" to solve any problem. His insistence on top-notch work helped me develop a sound research roadmap for this PhD thesis. Being around him these 5 years has indeed changed my life for the better.

Indispensable to this dissertation was the role of Dr. Sanjoy Paul. I was excited when he stepped in to work with me, but little did I expect such an incredible teacher. He taught me to appreciate the intricacies of TCP design, the "how to" of protocols, and to always ask "why" and understand the operational aspects in great detail. His patient feedback for each of my papers has been of tremendous help. I will always remember his words - "Everybody wants to do big things; few realize they are made of small steps".

I sincerely thank Corporate Research, Thomson Inc., for the "Thomson Student Fellowship" award that sustained me through my PhD program. Particularly, I thank Dr. Kumar Ramaswamy for being such a good friend and mentor and for his very useful and constructive feedback at various stages. I thank them for the opportunity to visit the Thomson lab twice a week. It was of tremendous help to develop the insights I needed to come up with simpler algorithms to improve implementation efficiency.

My sincere thanks also to Prof. Wade Trappe and Prof. Roy Yates for serving on my PhD defense committee. Prof. Trappe counseled me at various times during my PhD career, particularly when I needed it the most during the qualifiers. I thank Prof. Yates for the opportunity to work with him on some of the projects. His words "Where there is a problem, there is a research opportunity", inspired me to find a way out of several challenging problems in my PhD work.

I would like to thank several well-wishers at WINLAB and Thomson for their help and insights. First of all, I would like to thank Mr. Zhibin Wu for his ingenious patches to the NS2 simulator that solved several persistent bugs. I thank Mr. Ivan Seskar for his help in understanding my results at various times. His hands-on approach to address systems problems has been a tremendous inspiration. My thanks to Dr. Hang Liu for very useful MAC protocol discussions and Mr. Haris Kremo, Mr. Hithesh Nama and Mr. Joydeep Acharya for insightful discussions on physical layer aspects. Sincere thanks to Mr. Sachin Ganu, Ms. Suli Zhao, Ms. Hongbo Liu and Mr. Xiangpeng Jing for their constructive feedback. Thanks also to Mr. Faiyaz Ahmed, Mr. Howard Edinger, Dr. Larry Greenstein, Dr. Richard Frenkiel, Mr. Saurabh Mathur and Mr. Ishan Mandrekar for their valuable insights on code implementation, error models, design of systems and state-of-the-art video devices and applications. My sincere thanks also to Dr. Daniel Reininger for his enchanting course on Computer Networking and his recommendations at various entry points in my career.

I could have never gotten this far if not for the support of my family. I thank my parents Mrs. K.R. Ramamani and Mr. P.D. Gopal, for showing by example the value of hard work. My mother has always been my inspiration for persistence and self-motivation. I thank my pati (grandmother) Smt. Kamalamma for always insisting on perfection. My sincere thanks to my parents and parents-in-law, Mrs. Anasuya Iyengar and Mr. S. K. Iyengar for their indispensable help on the home-front when we needed it the most. I thank my little son Rahul for being so good and patient with his mamma even at his tender age of two. I thank Dr. Sudha Dixit for her excellent help to take care of Rahul. I sincerely thank Dr. Seema Katiyar for her thoughtful help and advice.

Finally and most importantly, I thank my best friend and dear husband Mr. Vinay Iyengar for being an un-daunting pillar of support to me in the past decade. His resourcefulness and a general "can do" attitude have always inspired me. His pep talks, encouragement and moral support have been indispensable, particularly to complete this dissertation.

These instill the confidence to strive for tougher objectives. ***Mahatma Gandhi*** said "*Be the change you wish to see in the world*". I will end this section with a poem from ***Rabindranath Tagore***'s "Geetanjali". This collection of poems won the Nobel Prize in Literature in 1913.

Where The Mind is Without Fear

WHERE the mind is without fear and the head is held high
Where knowledge is free
Where the world has not been broken up into fragments,
By narrow domestic walls
Where words come out from the depth of truth
Where tireless striving stretches its arms towards perfection
Where the clear stream of reason has not lost its way,
Into the dreary desert sand of dead habit
Where the mind is led forward by thee,
Into ever-widening thought and action
Into that heaven of freedom, my Father, let my country awake.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

This thesis addresses the problem of reliable data transport over emerging wireless networks. The last two decades have seen tremendous advancement in high-data rate wireless network technologies. Inexpensive unlicensed band radio technologies have enabled quick deployment of wireless local area networks in homes, offices, hotspots and even entire cities. Commercial multi-hop mesh networks are emerging rapidly and are expected to gain momentum with the ratification of the 802.11s standard. Mesh networks are being deployed to enable outdoor municipal networks at relatively low cost, and also as a wireless backhaul in indoor high-performance wireless LANs.

The Transmission Control Protocol (TCP) is the most popular transport protocol on the Internet today. In the last 3 decades it has connected islands of heterogeneous networks, making data accessible from across the globe. Extending TCP performance to wireless networks has been an active research topic in the last 15 years, but several problems have been encountered. Although the first demonstration of TCP included Packet Radio and Packet Satellite networks, its subsequent optimization was driven by rapid advancement in "wired" link technologies such as Ethernet [1]. These wired links in general have high reliability (bit error rates of the order of 10^{-12}) and have very low rate of fluctuations, if any. Losses in wired networks are hence dominated by queue overflows in intermediate and end nodes, and TCP has been optimized likewise.

Wireless link characteristics are significantly different. It is now well known that wireless links have high loss rates and various TCP enhancements have tried to address this issue [31][35]. However it is less understood that wireless links are time-varying in nature with rapid fluctuations in bandwidth and error characteristics. One reason for these variations is fluctuating signal-to-noise ratio (SNR) at wireless receivers that occurs from people walking, mobility of nodes and intermediate objects, environmental factors such as rain etc or even opening/closing of doors. Fluctuating SNR affects the likelihood of packet reception, and hence affects the link reliability perceived at higher layers.

Another reason for these variations is interference. The 802.11 networks, which are the basis for most of the emerging wireless scenarios, are shared media technologies that operate using distributed medium access control mechanisms. 802.11 links thus suffer additional interference issues because of multiple contending radio nodes in close proximity. This not only results in

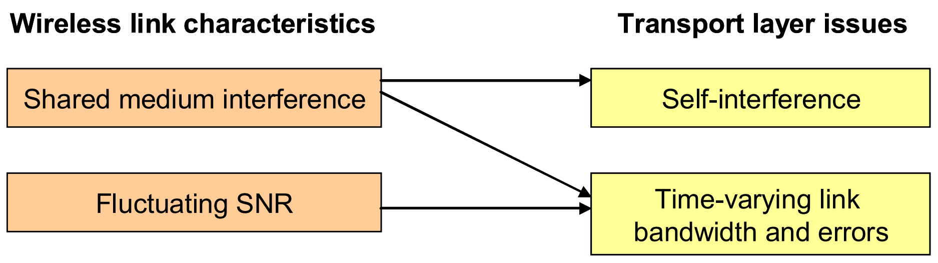

Wireless link characteristics
Transport layer issues
Shared medium interference
Fluctuating SNR
Self-interference
Time-varying link bandwidth and errors

Figure 1.1: Transport issues due to wireless characteristics

MAC collision losses, but also affects the channel time available for transmission, causing delay variations. Thus interference issues cause additional fluctuations in link bandwidth and error characteristics. 802.11 wireless cards commonly include non-standard link layer enhancements such as MAC retransmissions and auto-rate adaptation that introduce high transmission latency in changing noise scenarios, thus introducing additional link bandwidth fluctuations.

In addition to these, 802.11 networks have the unique self-interference problem for flows with bi-directional traffic. Here packets of the same flow contend with each other for channel access, often degrading overall performance, as observed with TCP DATA-ACK contention in wireless LAN scenarios [3][4][5].

The situation only worsens with increasing wireless hops, because now the transport protocol must deal with the compound effect of quality variations on all the links in a path, as well as complex external interference and self interference effects resulting from dense node placement.

Traditional transport protocols such as TCP were designed under the assumption of relatively slow-varying links with high reliability, and do not generally have to deal with self-interference that arises only in shared media networks. Experiments with TCP confirm that TCP performance is very sensitive to signal quality variations and other-user interference due to frequent protocol timeouts. TCP in these scenarios is observed to have excessive file transfer delays, despite a high goodput achieved by saturating UDP traffic. Such a large performance gap between UDP and TCP indicates the potential for new reliable transport protocols that might achieve significantly faster file transfer than TCP on wireless media [2][31][33][35].

1.1 Problem statement

Overall in 802.11 networks, fluctuating SNR and shared-medium interference translate to *self-interference* and *time-varying link characteristics* for higher layers, as depicted in Figure 1.1. In this section we describe these effects in detail and hence describe the research problem considered.

1.1.1 Self-interference in 802.11 wireless links

TCP performance evaluations demonstrate that interference among TCP DATA and ACK packets is a significant cause of throughput degradation in single hop and multi-hop scenarios. Figure 1.2 depicts one example of the instantaneous received rate of the popular TCP-Reno version in a wireless LAN environment with no other noise or other-node interference, and without using any additional link layer enhancements. TCP utilizes the available bandwidth poorly and experiences several durations of low or zero goodput. Much to the contrary, the UDP goodput plot obtained with saturating CBR traffic demonstrates much higher bandwidth available, than used by TCP. Our detailed investigation of TCP performance revealed that the various intervals of low goodput are due to TCP deadlocks that end in timeouts. The deadlocks occur after interference between the DATA and ACK packets of the same flow, result in multiple losses within a TCP congestion window. In these situations, the widely used fast-recovery algorithm of TCP, fails to recover all the lost packets, leading to the deadlock situation [5].

Next, Figure 1.3 depicts the performance achieved with increasing hops between a source-destination pair in a multi-hop wireless topology, with additional MAC retries to improve the reliability of links. Here the goodput achieved by upper-bound UDP decreases with increasing wireless hops, because of lower end-to-end bandwidth because of self-interference between DATA-DATA packets from adjacent nodes. When nodes within interference range of each other transmit packets, they cause lesser channel time available for other nodes, decreasing the overall number of DATA packets transmitted in a given interval. This multi-hop wireless characteristic is a fundamental shift from traditional transport protocol assumptions. TCP is designed to match the sending rate to the delay-bandwidth product of the route by "filling the end-to-end pipe" with as

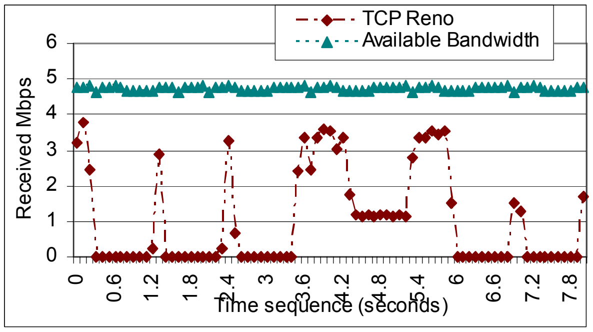


Figure 1.2: TCP self-interference in 802.11 wireless LANs

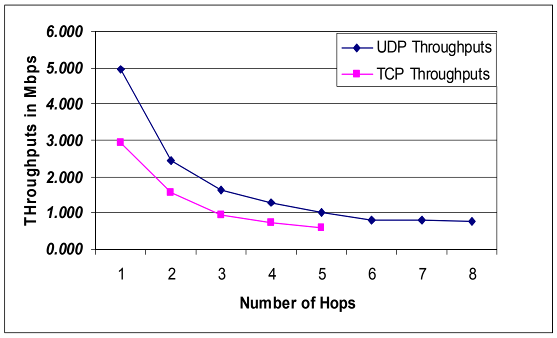


Figure 1.3: General self-interference problem with increasing wireless hops in a chain (single flow)

many packets as possible. The results here demonstrate that this approach lowers net available bandwidth and increases interference losses, and hence degrades the overall throughput of the flow.

1.1.2 Time-varying links

Wireless links are time-varying for various reasons. Changing SNR and interference in the wireless shared medium manifest as changes in link quality for higher layers. In fact, various MAC layer adaptations aggravate the flucatuations. We describe these various reasons for fluctuating link quality in 802.11 wireless links in the sections below.

1.1.2.1 Time-varying errors

A wireless link loss is caused because the wireless receiver is unable to decode at least one signal constituting a packet. When there is no interference, accurate reception depends primarily on the received signal strength relative to the received noise power. This value is commonly known as the Signal-to-Noise Ratio (SNR). The likelihood of accurate reception increases with increasing SNR. When another signal interferes, it corrupts the legitimate signal, causing the signal to be wrongly decoded by the receiver. This inaccurate decoding even for a single symbol constituting a packet often results in the loss of the entire packet. Time-varying losses are caused because of fluctuating SNR and interference at the wireless receiver.

SNR losses: SNR fluctuations occur from fading, shadowing and additive noise as revealed by several propagation studies such as those described by Rappaport[64]. Thermal noise causes an additive white Gaussian noise to always be present at the receiver. Even when the nodes are not mobile, environment changes caused by movement of people, and opening and closing of doors, have been found to cause SNR changes. These factors also cause a high loss rate in wireless links [75].

Interference losses: Changes in interference occur in 802.11 links because nodes become active randomly, and contend for channel access in a distributed manner. Particularly, when nodes have saturating traffic, the 802.11 MAC continuously contends for channel access. This increases the likelihood of MAC collisions as we showed analytically in an earlier paper [3]. We derived that that in a wireless LAN environment (all nodes within hearing range of each other), the loss likelihood ζ with saturating traffic among (N+1) active nodes is

$$\zeta = 1 - \left(\frac{(CW-1)!}{(CW-N-1)! * (CW)^N} \right)$$; where CW is the size of the contention window.

For example for two active nodes the collision likelihood is 3%, and for three nodes it is 17.5%. Hence in 802.11 wireless links, interference losses could significantly dominate SNR losses.

1.1.2.2 Time-varying bandwidth

Dynamic interference is one reason for time-varying link bandwidth. In addition, there are various 802.11 MAC features and enhancements that "translate" the fundamental wireless problems of interference and channel noise to bandwidth fluctuations.

Interference: The shared medium operation of 802.11 uses the CSMA/CA protocol where nodes contend for channel access on a per-packet basis. Here nodes only send when they sense the channel to be idle (Carrier Sense Multiple Access – CSMA). Simultaneous transmissions are minimized by transmitting in a contention window slot after random backoff (Collision Avoidance – CA). As a result the net "channel time" available to a node for transmission, is a function of number of active nodes in that interval. Since nodes are active at different times independent of each other, the channel time/interval might fluctuate rapidly affecting the number of packets sent out by the MAC in each interval. This appears like changing bandwidth to the transport layer.

Auto-rate adaptation: This is a special algorithm introduced in most wireless cards to maximize the link speed for a given SNR. The use of the algorithm is not specified in the 802.11 standard and is proprietary to each card manufacturer [70]. Wireless cards come equipped with several different modulation schemes that provide a wide range of channel rates. For example 802.11a/b/g cards support channel rates 1Mbps to 54Mbps. Higher channel rates however also cause higher loss likelihood because of decreased likelihood of accurate reception for the same SNR. Hence card manufacturers introduce auto-rate adaptation schemes to adapt channel

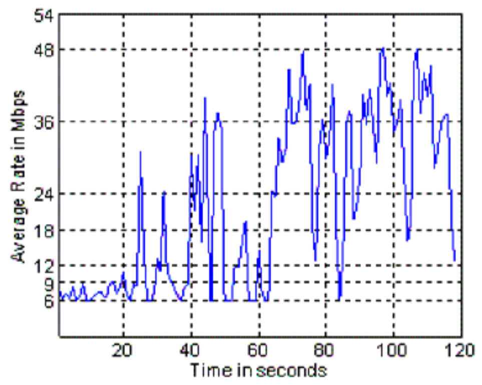


Figure 1.4: Auto-rate adaptation in wireless cards over noise-prone wireless links (result by Wu, Ganu et. al [58])

modulation to operate at the highest channel rate possible while still meeting the threshold reliability requirements. These algorithms are non-standard, but are invariably introduced in wireless cards to improve overall performance. When channel reliability fluctuates, say due to random noise, auto-rate adaptation causes the physical channel rate to also fluctuate. Wu, Ganu et. al. demonstrated this effect with experimentation on the ORBIT wireless test-bed in the 802.11g environment[58]. Their result (depicted in Figure 1.4) shows that the link bandwidth fluctuates rapidly between 6Mbps and 48Mbps in a 10 second interval, due to the combination of channel noise and auto-rate adaptation. Thus for the transport layer, auto-rate adaptation translates channel noise fluctuations to link bandwidth fluctuations.

MAC retries: MAC retries are suggested in the 802.11 standard to reduce transient interference losses, but are not mandatory [45]. However they are used by default in wireless cards mainly to hide these losses from TCP, and hence improve TCP performance over these shared medium links. When enabled, the MAC reacts to a loss by retransmitting the packet and doubling the contention window after each retransmission. This doubling halves the likelihood of MAC collisions, but also doubles the average delay to transmit the packet. For example the minimum delay (average) due to MAC random backoff in 802.11b is 310μs and the maximum delay is 10230μs since the MAC contention window can operate between 31 slots and 1023 slots. Since the 802.11 MAC does not differentiate between loss types, doubling is done even when the loss is due to noise and not interference. When the channel noise characteristic fluctuates, MAC retries could result in fluctuating MAC transmission delay for each each packet. To understand the effect at the transport layer, we conducted a simple simulation with saturating UDP traffic with and without MAC retries in a 3-hop wireless chain in a mesh topology. Figure 1.5 depicts the instantaneous received rate of UDP with and without MAC retries. Without MAC retries, the

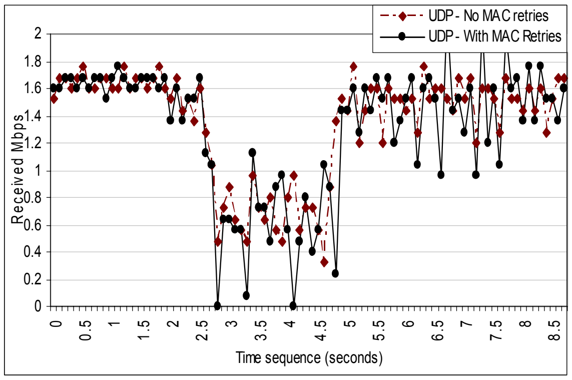


Figure 1.5: Effect of MAC retries on UDP receivd rates in a 3-hop wieless environment with bandwidth and error fluctuations

changing channel noise indeed causes some fluctuations in the received rate, but the fluctuations are significantly magnified with MAC retries. They also result in several intervals of zero received rate, while without MAC retries, there is a goodput of 0.5 – 1Mbps. Thus MAC retries cause the perception of rapid fluctuations in link bandwidth to the transport layer.

Packet sizes: Another less known result is the changing link capacity as a function of packet size. Results on video multicast experiments over wireless LANs demonstrated this insight where the 802.11 link capacity changed as the packet size distribution in video streams [10]. It happens because the 802.11 MAC introduces a large fixed time-overhead to transmit each packet, due to MAC contention. Hence the bandwidth utilization is lesser for smaller packet sizes. Hence for a VBR stream that constitutes a wide range of packet sizes, the link capacity fluctuates depending on the distribution of packet sizes in a given interval. Details of these experiments and results are in Appendix Section 6.1.

Thus the lower layer characteristics of fluctuating SNR and shared medium interference translate to time-varying link characteristics and self-interference at the transport layer. The problems for data transport due to these effects increase with increasing wireless hops. Next, we describe the research opportunity available to improve transport efficiency over these networks.

1.2 Research opportunity and our approach

Reliable transport protocols operating over wireless networks must address the fundamental issues of time-varying link quality (bandwidth and errors) and self-interference. To find available opportunities, we first explored the effects of interference and time-varying links on transport protocol performance, with in-depth evaluation of both TCP and UDP performance. The various results are presented in detail in chapters 3 and 4 where all protocols are evaluated. In this section we will summarize the derived opportunity and our proposed solution motivated.

Figure 1.6 depicts a wireless 1-hop result in the ORBIT test-bed, demonstrating the effects of fluctuating channel noise on transport protocol performance. Random Gaussian channel noise is injected in nodes at 5-second intervals, causing random packet drops at the receiver. UDP goodput shows the net received rate to change between 0 and 5Mbps at the same rate as injected noise. On the other hand, the TCP received rate plot shows that it misses several high link quality opportunities, and takes much longer to adapt after the link becomes consistently good (at 66 seconds). Here TCP achieves less than 25% of UDP goodput. These problems are even more severe over multi-hop wireless networks because of the compounding of loss and interference effects with increasing hops. The wide gap in goodput between TCP and the upper-bound

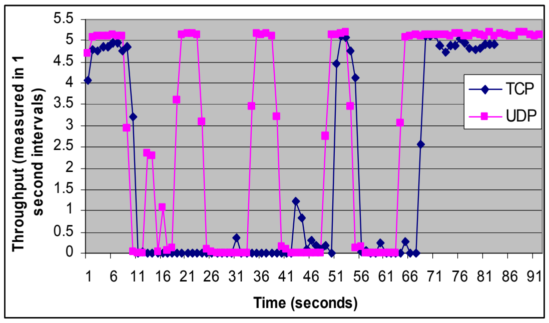


Figure 1.6: ORBIT test-bed result - TCP performance in a channel with burst noise, with MAC retries

achieved with simple unreliable UDP, shows potential for new reliable transport protocols that might achieve significantly faster file transfer than TCP on wireless media.

Link-layer enhancements (such as MAC retries or hybrid ARQ) for hiding wireless channel impairments from the transport layer have been tried earlier. These enhancements handle non-varying link characteristics well, but for fluctuating link conditions introduce large transmission latencies that even degrade the net bandwidth available as demonstrated with MAC retries in figure 1.5. Further the heterogeneity of existing and emerging wireless links, limited processing capability of cheap wireless hardware, and the diverging requirements of traffic (video, voice, data) make it very difficult, if not impossible, to address all the per-flow issues at the link layer. Transport layer protocols, on the other hand, residing on more powerful end-system hardware, are better positioned to address these issues in a link agnostic manner.

In the last 15 years, various approaches have tried to improve TCP performance over wireless networks. However several core design aspects of TCP are mismatched to the fundamental wireless characteristics, making it difficult to "fix" TCP for wireless networks or for that matter, "fix the wireless problem" to help TCP perform well. First, TCP's flow control algorithm is designed to scale back when losses occur. This design stems from wired network characteristics, where losses imply filled up queues in a bottleneck node at the source of the slowest link. However this design causes TCP to reduce the sending rate "unnecessarily" when operating over wireless links with random errors. Second, TCP's window-based flow control algorithm requires a regular pace of positive acknowledgements to adapt sending rate. But time-varying link bandwidth causes fluctuating transmission delays in each wireless link that in turn makes the acknowledgements irregular. This delay-variance effect of TCP acknowledgements are shown to degrade TCP throughput significantly [1]. Third, in 802.11 links positive acknowledgements cause self-interference that degrades TCP performance due to bandwidth sharing and MAC collision losses. But TCP needs positive acknowledgements to clock its sending rate. Hence self-interference will remain a dominant cause of packet losses and bandwidth reduction for TCP over 802.11 networks. Fourth and final, TCP's end-to-end bandwidth estimation procedure using own acknowledgements, is relatively slow to adapt compared to the time-scale of the bandwidth fluctuations. Instead, status indicators in the Phy/MAC layers in each hop more conveniently capture the changes and hence promote the case for cross-layer protocol design.

Instead, the opportunity to improve instantaneous goodput performance of reliable transport protocols lies in the cross-layer information available in lower layers in the network stack. The physical layer for example has complete knowledge of the channel rate used, and the average received signal strength – RSSI, which are together indicative of the link quality due to channel noise. The 802.11 MAC layer has information of the net channel time available for transmission after contention with other nodes. Thus a node in itself has information of link quality and interference, and we use this "cross-layer information" to determine the instantaneous link bandwidth available.

These various observations motivated CLAP – a Cross Layer Aware transport Protocol, as a general solution for reliable file transfer over wireless networks. It is developed addressing unique wireless features such as that there could be substantial bandwidth available despite high loss rates, and that control traffic must be minimized to conserve the bandwidth for legitimate DATA packets in shared media networks.

The performance of reliable transport protocols is usually measured with "bulk throughput" which is the ratio of the file size to the time take to complete the transfer. But because it evaluates the overall performance of reliable transport, combining the effects of flow control and error control algorithms, this metric does not indicate the upper-bound performance possible in the given scenario. We hence introduced "instantaneous goodput" as a performance metric, which is defined as the ratio of the number of bytes delivered to the time interval considered. This metric measures the upper-bound performance, since it removes the effects of error control and measures how well the protocol is using the available bandwidth instantaneously. We use this metric to evaluate the various protocols in several time-varying one-hop and multi-hop wireless scenarios.

1.3 Contributions

We will enumerate the contributions in this investigation categorizing them in terms of research opportunity and proposed solution.

- Identified primary wireless problems that affect file transfer end-to-end
- Identified mismatched TCP design aspects that restrict performance in wireless networks
- Developed a novel transport protocol, CLAP, which achieves significant performance improvements for reliable transfer over 1-hop and multi-hop wireless networks
- Implemented a cross-layer protocol framework for CLAP
- Validated all of the above with extensive simulation and experimental methodologies

1.4 Related Work

Data transport over wireless networks has been an active research topic in the past 15 years. Various observed issues of wireless networks such as transient packet loss, disconnections and route failures have been extensively studied in literature. However the fundamental aspects of interference and time-varying link characteristics in wireless shared media networks have received less attention.

Several enhancements to TCP have been proposed recognizing that TCP often scales back flow control "unnecessarily" over wireless networks. Some have tried 'implicit decoupling' with link layer enhancements such as with link layer retransmissions to hide wireless losses from the TCP sender [2][21][27][31][38][66][69]. Others have tried 'explicit decoupling' where, for example, the window size is explicitly frozen during disconnection avoiding errors to affect the flow rate [20][26][34][36].

Each proposed solution has addressed at most one wireless "symptom" such as transient loss in cellular networks [14][20][24][26][29][31][32][35], disconnections/route failures[20][21] and delay variance[2]. However since these "symptoms" emerge because of the core characteristics of time-varying link quality and interference – they are also tightly interconnected. For example, fluctuating SNR and/or interference cause transient packet losses, and transient packet losses result in temporary disconnections and route failures. Chun and Ramjee showed that even with link layer enhancements to handle these wireless issues "locally", the transmission latency introduced significantly degrade end-to-end TCP performance [2]. The gains are hence limited when the problems are addressed separately and when the transport protocol is not geared to handle link quality fluctuations.

	Cellular 1-hop	**802.11 link, 1-hop**	**802.11 links, Multi-hop**
Low random packet errors (due to channel noise/MAC interference)	Snoop-TCP, BA-TCP , TCPW, TCP-Triple-ACK-Recovery, RCP	addressed by MAC layer solutions	ATCP, TCP-F, TCP-DOOR, Atra
Slow-varying links (due to delay-variance/bandwidth fluctuations/ disconnections/route failures)	Freeze-TCP, Ack-regulator, window-regulator	not addressed	TCP-ELFN, TCP-BEAD, ATP
High error rates, rapidly varying links (general scenario due to MAC interference, fluctuating SNR)	addressed by physical layer solutions	not addressed	not addressed

Table 1.1: Summary of available transport protocols for wireless networks

New protocols have also been proposed, such as RCP[29] and ATP[13] in cellular and 802.11 multi-hop contexts respectively. However, they still use positive acknowledgements to clock the sending rate (RCP addresses transient losses in cellular networks, and ATP addresses route failures in multi-hop wireless networks), and hence do not operate well in rapidly time-varying and high loss scenarios.

From the wireless perspective, the available transport protocols may be categorized based on the type of wireless network and the wireless problem they address, as depicted in Table 1.1 with a sampling of protocols listed in each category. Some propose TCP enhancements for cellular networks [2][20][21][29][31][34][35], while others address 802.11 networks [22][24][25][26][32][33][38]. They broadly address low link error rates (< 5%) and/or slow-varying wireless links (occasional disconnections and route failures). In the table, protocols available in the shared medium have not considered the wireless 1-hop scenario and hence are not list in that shared medium 1-hop category. Rapidly varying bandwidth scenarios do not occur with cellular 1-hop networks, and hence that category is shaded out. To the best of our knowledge, none of the available solutions address fluctuating link bandwidths and errors, or link error rates higher than 5% that are inherent characteristics of emerging wireless network scenarios.

The various proposed wireless transport solutions are restricted to a specific type of network, and many of them require flow-specific and network-specific link layer proxies. They hence lack general applicability to heterogeneous networks. History shows that TCP enabled the Internet with widespread access, because of its ability to integrate islands of heterogeneous networks.

Such an integrating protocol is required for wireless networks more so with increasing heterogeneity of end-user devices.

The CLAP protocol presented here instead is developed “top-down” from basic principles addressing the core wireless characteristics. CLAP is applicable to any network where cross layer status information is available as an overlay network service.

The rest of this document is organized as follows. In Chapter 2 we describe our detailed investigation of TCP performance in wireless LANs affected by self-interference and time-varying noise conditions, with evaluations in the NS2 simulator and experimentation on the ORBIT test-bed. The CLAP protocol is described in detail in Chapter 3 along with details of the cross-layer software framework developed to systematically extract intra-node and inter-node status information. Chapter 4 has the CLAP protocol evaluated in wireless LAN scenarios. Chapter 5 evaluates the CLAP, TCP and UDP protocols in multi-hop wireless scenarios considering time-varying bandwidth and noise conditions. We conclude in Chapter 6 with directions for future work. The Appendix chapter 7 has additional results from experiments with video multicast over wireless LANs, detailed description of NS2 simulations and ORBIT test-bed experimentation. It also has additional design details of the CLAP protocol pertaining to the error control algorithm.

Chapter 2
TCP performance in wireless LANs

Wireless 1-hop scenarios are common in many different network technologies – 802.11 wireless LANs, 802.16 (WiMAX) for wide-area coverage, and cellular EvDO technologies. Of these, wireless local area networks have become very popular in the past few years, particularly with the ratification of the 802.11b/g/a standards and widespread availability of cheap wireless cards that enable easy Internet access from portable computers. Recent advances in physical layer technologies have resulted in steadily increasing transmission speed, for example 802.11n promises channel rates exceeding 100 Mbps.

Since TCP is the most popular reliable transport protocol used on the Internet, improvements in its performance over cellular 1-hop and multi-hop 802.11 networks have been actively considered these past several years. Much of this research is based on the premise that high loss rate links in cellular 1-hop scenarios and mobility-induced route failures are the prime reasons for poor TCP performance. Since wireless LANs have a single wireless link and short range of transmission, TCP performance in these networks has seldom been considered for evaluation assuming good performance. However, our own personal experience showed otherwise. The wireless LAN deployed in our laboratory building often did not provide the desired service experience. File download delay was unpredictable at certain times of the day (such as mid-afternoon when there was maximum activity in the lab) and some offices often experienced poor connection quality (particularly corner offices where high performance was the most required!). Since the problem happened for applications that required reliable transfer, we anticipated the problem to be with TCP and began investigating it with simulations in the NS2 simulator and wireless LAN emulation in the ORBIT wireless test-bed.

These investigations revealed that wireless links are in fact characterized by issues much different from those that were commonly considered in literature. Wireless links are time-varying in nature with rapid fluctuations in bandwidth and error characteristics, instead of just having a static loss distribution. Another significant problem for TCP was self-interference in shared-medium operation of 802.11, where DATA and ACK packets of the same flow interfered with each other.

This caused significant throughput degradation in TCP. For TCP-Reno in particular, there were several timeouts in the course of a flow when additional link layer enhancements such as MAC retries were not used. We found that these timeouts were due to increased loss likelihood during fast-recovery, and the less optimized TCP Tahoe was found to perform much better. These various TCP performance issues due to interference are presented in this chapter.

In the next section we present a brief background of 802.11 and TCP, followed by an in-depth analysis of TCP's self-interference problem considering bulk throughput effects and per-packet dynamics of various TCP versions. Next we tried to reduce the self-interference problem with a "skip-ACKs" enhancement to TCP. Here when no losses are encountered, the TCP receiver sends fewer ACKs than otherwise. However the gains achieved with this approach are still limited. We explore core reasons for poor TCP performance in all these cases. Valuable insights were derived from these TCP lessons to develop the new solutions for reliable file transfer.

2.1 Overview of protocols and simulation setup

Signal interference occurs with all wireless technologies, whether cellular or 802.11 because of the inherent broadcast nature of the wireless medium. An efficient medium access method is invariably required to minimize interference among active nodes in the neighborhood. Cellular technologies address this with centralized medium access schemes such as TDMA/FDMA/CDMA where the base station allocates a unique time-slot/frequency-band/orthogonal code for contention-free access to the wireless channel.

The 802.11 networks instead use a distributed coordination method (DCF mode), where each node contends for channel access for every packet it has to send. They implement the Carrier sense Multiple Access with Collision Avoidance (CSMA/CA) protocol, where the likelihood of collisions in the shared medium is minimized with random back-off and waiting for an idle channel before sending. However, unlike cellular networks, interference affects higher layers in 802.11, because with distributed access the net sending rate of an 802.11 MAC depends on other active nodes nearby and losses due to interference can still occur when at least one neighborhood nodes sends in the same random back-off slot.

In the next few sections we present details of the 802.11 MAC and describe the TCP protocol and its complex interaction with the 802.11 MAC. These overview discussions are required to describe TCP self-interference problems in later sections

2.1.1 802.11 MAC Overview

The Distributed Coordination Function (DCF) mode of 802.11 Medium Access Control (MAC) uses Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)[45]. Even in the infrastructure mode with a central Access Point (AP), all entities have equal priority for channel access. Channel contention happens on a per-packet basis. A node waiting to transmit a packet, first senses the channel to be idle for a certain duration (called DIFS [45]), then selects a backoff slot randomly based on a uniform distribution in [0,CW-1], where CW is the contention window. The packet is transmitted if the channel is still idle in the selected slot. If not, the node waits till the channel is idle again, backs off only for the requisite slots before attempting to transmit. The node learns of a successful transmission when it receives an acknowledgement (MAC-ACK) from the destination.

Throughput derivations of 802.11 MAC have typically assumed Poisson packet arrival per backoff slot [55]. Instead, suppose there are (N+1) nodes with a continuous supply of packets that causes them to contend for the channel far more consistently. A transmission is successful only if no other node transmits in the same backoff slot. This likelihood of all nodes selecting independent slots is

$$1*\left(\frac{CW-1}{CW}\right)\left(\frac{CW-2}{CW}\right)\cdots\left(\frac{CW-N}{CW}\right)$$

$$=\left(\frac{(CW-1)!}{(CW-N-1)!*(CW)^{N}}\right)$$

Hence the likelihood of at least two nodes selecting the same slot is

$$1-\left(\frac{(CW-1)!}{(CW-N-1)!*(CW)^{N}}\right) \qquad (1)$$

This is the likelihood of a failed transmission for nodes having a consistent supply of packets to send.

2.1.2 Brief Overview of TCP

TCP is a reliable sliding window protocol. It allows several packets to be transmitted before an acknowledgement (TCP ACK) is received. Unacknowledged data packets are maintained in a congestion window (*cwnd)*. In steady state, *cwnd* is expected to be equal to the delay-bandwidth product of the network. TCP ACKs are cumulative. Hence it is not necessary to have separate

ACKs for each packet. TCP operates in either of slow-start or congestion-avoidance modes. In slow-start, *cwnd* grows exponentially, increasing by one packet for every ACK received, while in congestion avoidance, *cwnd* increases linearly (one packet per round-trip-time) in proportion to the number of data segments acknowledged.

In case of a packet loss TCP reduces *cwnd* assuming network congestion. With TCP Reno, three or more duplicate ACKs trigger a *fast-retransmit* and cwnd drops to ½ its previous value. Then, *cwnd* grows conservatively in congestion avoidance mode. The situation is much worse in case of a *timeout,* when *cwnd* drops to 1. Here TCP backs off exponentially before trying to send the next packet. The backoff duration can be as high as 64 seconds. Sinha et. al explained the derogatory effect of timeouts on TCP throughput [30].

These various design aspects of TCP evolved in the last 30 years, mostly to cater to wired networks. We will show in later sections in this chapter that some of these optimizations (*fast-recovery* for example) that work very well over the wired Internet, in fact worsen performance over wireless LANs.

In order to explore TCP performance in the general interference and time-varying 802.11 links, we considered noise-free as well as noise-prone wireless LAN environments. The noise-free scenarios were intended to study the effects of just interference on the transport quality, while the latter explored the effects when there was both interference and fluctuating SNR. Details of the simulation setup used for TCP evaluation is described in the next section.

2.1.3 Wireless LAN system description and simulation details

The various evaluations are conducted in the NS2 simulator, version 2.1b9a enhanced with the CMU wireless module containing 802.11implementation. Figure 2.1 depicts the wireless LAN system considered here. Each node caters to a single flow, and the wireless nodes act as data

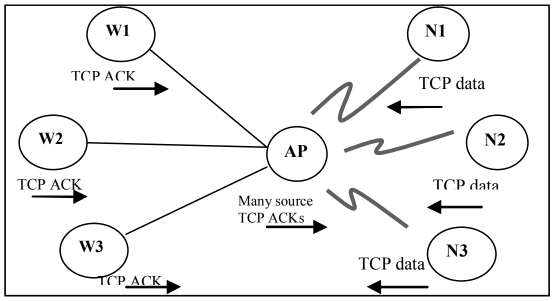


Figure 2.1: 802.11 wireless LAN topology

sources. The data sinks are wired nodes one hop away from the access point. The wired link bandwidth is chosen such that the wireless link is the bottleneck in all scenarios.

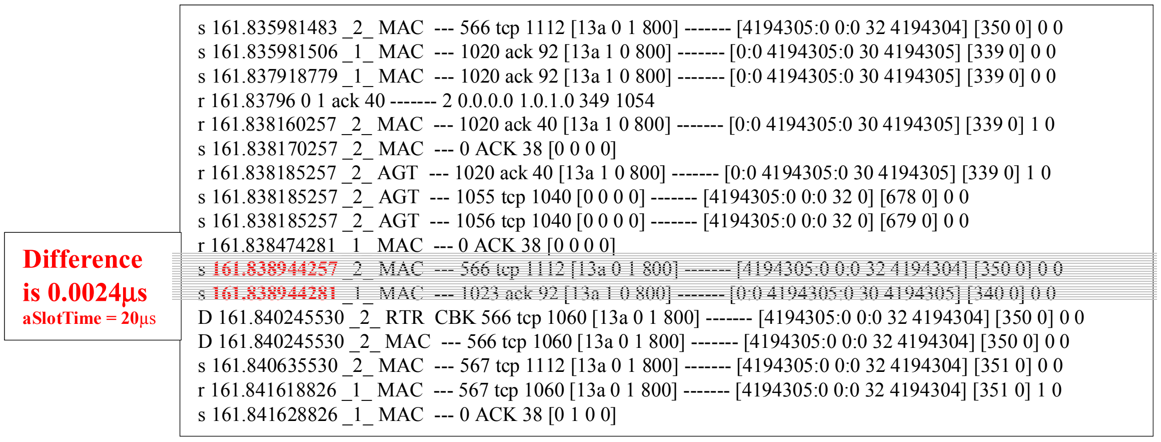
The physical channel rate is fixed at 11Mbps. However the net data rate at the transport layer is about 5Mbps due to various overheads given in Table 2.1 (in section 2.2).

2.2 Overview of TCP self-interference (noise-free conditions)

Simulation results with a single flow, showed TCP throughput to be significantly lower than the UDP throughput, even though there were no other interfering flows or channel noise that could result in packet losses. The analysis of the NS2 trace files revealed interesting insights.

2.2.1 TCP simultaneous-send problem

Detailed analysis of the traces showed that the problem in fact was happening because the send times of the DATA and ACK packet were within one slot time (20µs) of each other, i.e. they had selected to send within the same 802.11 MAC random backoff slot. The 802.11 MAC prevents nodes from transmitting when another node is already transmitting, with the CSMA requirement where nodes cannot send if the channel is sensed to be "busy". When the channel is idle, simultaneous transmission by multiple nodes is alleviated with the Collision Avoidance (CA) requirement, where they wait for a DIFS duration, and pick a random backoff slot to send the packet. If another node begins to transmit before this slot, the node waits until the channel is idle again to continue the count down before sending. However when two or more nodes select the same slot to send, they transmit in that slot without being aware of each other's transmission. This is because the 802.11 hardware is *half-duplex* and cannot detect a signal while in the transmit mode (unlike Ethernet they cannot send and receive at the same time). Hence the device cannot


Difference
is 0.0024µs
aSlotTime = 20µs

```
s 161.835981483 _2_ MAC  --- 566 tcp 1112 [13a 0 1 800] ------- [4194305:0 0:0 32 4194304] [350 0] 0 0
s 161.835981506 _1_ MAC  --- 1020 ack 92 [13a 1 0 800] ------- [0:0 4194305:0 30 4194305] [339 0] 0 0
s 161.837918779 _1_ MAC  --- 1020 ack 92 [13a 1 0 800] ------- [0:0 4194305:0 30 4194305] [339 0] 0 0
r 161.83796 0 1 ack 40 ------- 2 0.0.0.0 1.0.1.0 349 1054
r 161.838160257 _2_ MAC  --- 1020 ack 40 [13a 1 0 800] ------- [0:0 4194305:0 30 4194305] [339 0] 1 0
s 161.838170257 _2_ MAC  --- 0 ACK 38 [0 0 0 0]
r 161.838185257 _2_ AGT  --- 1020 ack 40 [13a 1 0 800] ------- [0:0 4194305:0 30 4194305] [339 0] 1 0
s 161.838185257 _2_ AGT  --- 1055 tcp 1040 [0 0 0 0] ------- [4194305:0 0:0 32 0] [678 0] 0 0
s 161.838185257 _2_ AGT  --- 1056 tcp 1040 [0 0 0 0] ------- [4194305:0 0:0 32 0] [679 0] 0 0
r 161.838474281 _1_ MAC  --- 0 ACK 38 [0 0 0 0]
s 161.838944257 _2_ MAC  --- 566 tcp 1112 [13a 0 1 800] ------- [4194305:0 0:0 32 4194304] [350 0] 0 0
s 161.838944281 _1_ MAC  --- 1023 ack 92 [13a 1 0 800] ------- [0:0 4194305:0 30 4194305] [340 0] 0 0
D 161.840245530 _2_ RTR  CBK 566 tcp 1060 [13a 0 1 800] ------- [4194305:0 0:0 32 4194304] [350 0] 0 0
D 161.840245530 _2_ MAC  --- 566 tcp 1060 [13a 0 1 800] ------- [4194305:0 0:0 32 4194304] [350 0] 0 0
s 161.840635530 _2_ MAC  --- 567 tcp 1112 [13a 0 1 800] ------- [4194305:0 0:0 32 4194304] [351 0] 0 0
r 161.841618826 _1_ MAC  --- 567 tcp 1060 [13a 0 1 800] ------- [4194305:0 0:0 32 4194304] [351 0] 1 0
s 161.841628826 _1_ MAC  --- 0 ACK 38 [0 1 0 0]
```

Figure 2.2 NS2 trace file snapshot that demonstrates self-interference between a pair of TCP DATA and ACK packets

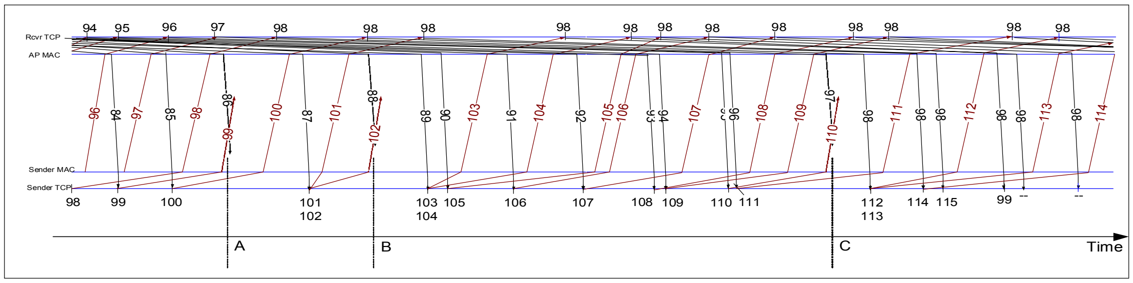


Figure 2.3: TCP self-interference over wireless LANs

detect a collision when it occurs, but realizes later that the MAC transmission failed when a MAC ACK fails to arrive.

In Figure 2.2, the highlighted lines demonstrate such a situation when two nodes picked the same slot to transmit. In this wireless LAN scenario with a single TCP flow, the two nodes are the AP and wireless client sending packets (a TCP ACK packet and a TCP DATA packet respectively) to each other, and we hence term it as the *simultaneous-send* problem of TCP over wireless LANs. Hence the AP and wireless client send signals to each other in the same 802.11 random backoff slot and fail to detect each other's transmission. The simulations demonstrated that this problem of same-slot selection happened quite frequently with TCP traffic, despite there being only one TCP flow.

This 802.11 MAC shortcoming for popular TCP traffic is a rather puzzling result, since the 802.11 MAC was specifically designed to extend Internet connectivity over wireless LANs, and TCP is the most popular transport protocol on the Internet. This motivated us to explore the core aspects of TCP and 802.11 MAC interactions that result in the poor performance. The analysis is explained in the next sub-section.

2.2.2 TCP saturates 802.11 MAC

The 802.11 MAC is designed to operate optimally for traffic that is Poisson-distributed, where packets to be sent arrive randomly and uncorrelated to each other at the MAC entities in a neighborhood. The arrivals are expected to have exponential inter-arrival times and that do not operate the MAC in saturation. We show in this section that TCP traffic is in fact quite the contrary – the traffic arrives in bursts and hence does not have exponential inter-arrival times, and this invariably operates the MAC in saturation, even while there is a single TCP flow over the wireless LAN.

The TCP sender generates data in bursts because of its window-based operation. When there is no loss, each incoming ACK triggers one or more segments to be sent. When a cumulative ACK arrives that acknowledges multiple segments, there could be a large burst of packets sent by the TCP sender to the link layer below.

When the link is a "wire" such as switched Ethernet, the packets are transmitted immediately, with little MAC overhead. On the other hand, 802.11 DCF-MAC is a stop-and-wait protocol that for each packet, first contends for channel access, sends the packet with significant MAC and Phy overheads, and then waits for a MAC ACK to confirm its reception. If the MAC ACK fails to arrive and MAC retries are enabled, the MAC retransmits the lost packet again and again until it succeeds. If a new TCP ACK arrives at this time (remember that nodes contend independent of each other for channel access and so the AP may send a TCP ACK irrespective of the sending situation at the wireless client), the TCP sender sends more segments down, even as the previous burst of packets are not yet sent. The effect of these packet bursts is hence to cause a sustained occupancy of the MAC send queue. This causes the MAC to operate in saturation, consistently contending for channel access to send packets. A consistent supply of packets occurs even at the MAC in the Access Point (AP), since in default TCP sink implementation one ACK is generated for every incoming DATA packet.

We derived the same-slot selection likelihood with saturating traffic in Section 2.2.1. Particularly when there is a single TCP flow operating, there are two nodes with saturating traffic contending for channel access. Here the likelihood of the same slot being selected by both nodes among CW slots available is simply (1/CW)*(1/CW)*CW. For the default contention window size CW=CWmin=32, the likelihood is 3%. For three nodes the likelihood is 17.6% from Equation (1). With these observations, we differ from Kamerman and Aben [55] who state that with TCP traffic the likelihood of two nodes selecting the same slot is miniscule.

Our insights are confirmed by the NS2 simulation trace of a TCP-Reno flow depicted in Figure 2.3. It captures the operation of TCP over 802.11 MAC in a wireless LAN. The TCP congestion window size (*cwnd)* in this state is 15 segments. A vertical cross section at any point shows at least one TCP packet waiting to be sent by the 802.11 MAC. This confirms our hypothesis of the 802.11 MAC operating in saturation with TCP traffic. The figure also demonstrates the *simultaneous-send* problem (losses at instants A, B and C in the figure) where the AP and wireless TCP data source send to each other within the same back-off slot and neither node detects the transmission. We have referred to this same phenomenon as TCP self-interference elsewhere in this document.

Another effect happens during the slow-start mode of TCP operation - the sender sends multiple DATA segments in response to an incoming ACK [3]. When ACKs arrive in quick succession,

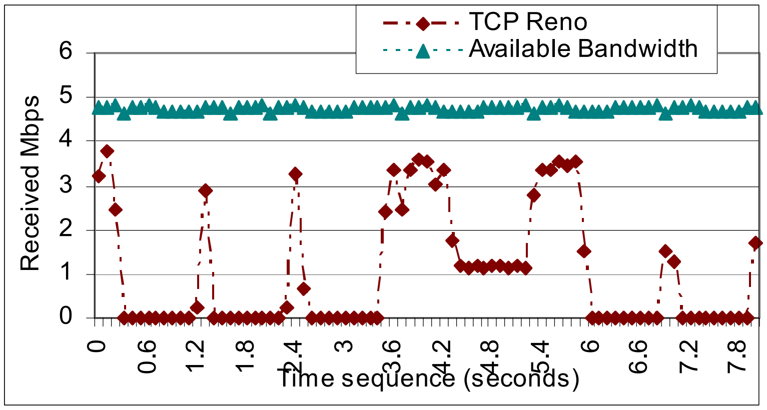


Figure 2.4: Instantaneous TCP-Reno received rate (NS2 simulations)

they result in a large burst of DATA packets at the sender-side MAC. If there is insufficient sending queue buffer in the MAC interface and the TCP burst size is not curtailed, there could be significant loss within the same node of the TCP sender at the MAC interface. This is a common reason for poor TCP performance in NS2 simulations. However in real-world implementation of TCP over 802.11 MAC, this problem of interface queue overflow seldom occurs because of the controlling role of the kernel or Operating System (OS) [4]. The OS forms the liaison between the TCP socket buffer and the interface queue. It moves a packet between them only when the interface sets an "available" flag. Most commonly the send operation in the TCP sender is implemented in a "blocking mode", and hence the sender is blocked from sending more packets down until the OS has sent packets to the interface to make more space in the TCP socket buffer.

Coming back to TCP performance during self-interference, the simulation results showed the bulk throughput for a TCP-Reno flow to be about 1Mbps during a 1 Megabyte file transfer without MAC retries. When MAC retries were used, the self-interference losses are recovered by the MAC and the throughput increased to 2Mbps. On the other hand, the goodput of a saturating UDP flow was consistently at 5Mbps (of course, no losses because there was no feedback traffic). So for no MAC retries, TCP experienced *80% degradation in throughput for a mere 3% loss rate due to self-interference.*

The instantaneous received rate plot of TCP-Reno depicted in Figure 2.3, shows why there is such a low TCP throughput. There are several durations of zero received rate at the TCP receiver, despite there being full bandwidth available in the link. Further, the peak received rate of TCP is just 70% of the available bandwidth. Clearly it is these reasons that cause the bulk throughput to be just 20% of the available bandwidth. This curious observation, motivated us to examine the

Overhead	Duration (μs)
DIFS	50
Average duration of random backoff for min. MAC contention window	310
Physical layer: short Preamble(144bits/2Mbps) + PLCP header (48bits/2Mbps)	96
MAC header + FCS duration (8*34bytes/11Mbps)	24.73
LLC + IP headers (8*(8+8)bytes/11Mbps)	11.64
Time taken by Additional SIFS + MAC-ACK, after successful delivery of the Layer-4 packet	10 + 304 = 314
40-byte TCP Header duration (40*8 bits/11Mbps) = 40 byte TCP-ACK duration	29.1
Total Overhead time for each Layer-4 packet (T_{ACK})	**835.47**
Duration of 1000-byte TCP data segment (T_{DATA}) (1000*8 bits/11 Mbps)	**727.28**

Table 2.1: 802.11 overheads incurred by a TCP packet

dynamics of TCP operation, to explain the core reasons why self-interference causes such degradation in TCP performance. The results are described in the next section.

2.3 TCP Dynamics over 802.11 wireless LANs

We conducted a detailed examination of the NS2 traces to understand the reasons behind the TCP performance of Figure 2.4. First, there were several fixed length intervals of zero received rates. Then, TCP throughput peaked at 3.5Mbps instead of the 5 Mbps of the UDP flow (available bandwidth plot). There was another fixed length interval with non-zero rate, but fixed and low TCP throughput.

Our analysis of lead to examining the dynamics of various TCP versions and the following insights were derived as a result:

(a) The peak TCP throughput is limited because of bandwidth sharing with the TCP ACK packets.

(b) *Poor Reno performance:* Losses due to self-interference often occur in quick succession (before the first loss is perceived by TCP sender). This results in multiple losses in a TCP congestion window. Reno's congestion-control is known to deadlock in such situations [72]. In Figure 2.4, the **six** "flat goodput" intervals result from deadlocks that end in timeouts.

(c) *Poor NewReno performance:* NewReno, an enhancement to TCP-Reno to recover multiple packets in a congestion window, in turn deadlocks when retransmissions are lost. Self-interference increases this likelihood, even during *fast-recovery.*

(d) *Tahoe outperforms Reno, and in some cases, NewReno:* TCP Tahoe, an earlier less optimized version of TCP gains in *throughput* over Reno and NewReno because of fewer deadlocks (that end in timeouts) in the self-interference scenarios.

We explain each of these aspects in detail in the sub sections below.

2.3.1 The Cost of TCP Acknowledgements

The TCP ACKs have two effects on performance over wireless LANs. First, they consume a portion of the link bandwidth at the cost of DATA packets, since the wireless link is not duplex – only one of TCP DATA or ACK packets can traverse it at a given time. Second, ACKs result in the loss of DATA packets due to *simultaneous-send* issues. We explore these aspects in detail below.

2.3.1.1 TCP-ACKs are expensive

A TCP-ACK comprises of 40 bytes of TCP header (without additional options). However various time overheads in MAC and Physical layers causes it to consume a significant portion of channel time. Table 3.1 specifies the various overheads for 11 Mbps channel rate (802.11b). Following is the average time consumed transmitting a TCP data/ACK (assuming previous transmission was successful):

$$\begin{aligned} T_{\text{PACKET}} &= T_{\text{DIFS}} + T_{\text{PHY\_PREAMBLE, HDRS}} + T_{\text{MAC\_BACKOFF, HDRS}} \\ &\quad + T_{\text{SIFS, MAC-ACK}} + T_{\text{IP\_HDR}} + T_{\text{TCP\_HDR}} \\ &\qquad + T_{\text{DATA}} \qquad (1) \\ &= T_{\text{ACK}} + T_{\text{DATA}} \end{aligned}$$

where T_{ACK} is the total time taken to transmit a TCP-ACK, given the same header size in TCP data and ACK packets. DIFS and SIFS are the Distributed and Slot Inter-Frame-Spaces respectively introduced by the MAC layer Each TCP packet (data/ACK) incurs an average overhead of 806.37μs and constitute **99.7%** of the channel time consumed by a TCP-ACK packet.

The bandwidth consumed by *n* ACKs in a unit interval *i*, at the expense of data packets may be calculated as follows:

Number of data packets that could have been sent: $k = n * T_{ACK} / (T_{DATA} + T_{ACK})$ (2)

Lost bandwidth $= (k*1000*8) \ / i$ bits/sec (3)

The frequency of TCP-ACKs in each unit interval in the course of the 1MB file transfer is shown in Figure 2.5. At peak operation, the number of returning ACKs is also at its peak - an average of 40 ACKs in a 0.1 second interval. At this time, the lost bandwidth from equations (2) and (3) is **1.47 Mbps**. In Figure 2 this number matches the difference between TCP's peak instantaneous throughput and the available bandwidth, corroborating the ill-effects of self-interference on TCP throughput. A more expensive effect however is packet loss that triggers congestion control in TCP.

2.3.1.2 Packet loss due to self-interference

In Figure 2.5, when the throughput is at its peak, the number of returning ACKs is also at its peak. The dynamics of TCP and 802.11 MAC during this time is depicted in Figure 2.2. It demonstrates the situations when self-interference results in three MAC collisions in quick succession of each other (instants A, B and C), before TCP-sender detects the first loss. Hence all the losses occur within a single congestion window (15 segments at this time).

A vertical cross section between a TCP and MAC process (in Figure 2.2) shows how many packets are waiting in the MAC queue (an indirect conclusion on the AP side). Clearly with a consistent supply of TCP packets, the MAC operates in saturation. We showed earlier that the collision likelihood in this MAC situation is 3% [3].

TCP sender detects a loss (the first one) only when three duplicate ACKs arrive. It then scales down the congestion window (and hence the sending rate) despite a high bandwidth availability in the wireless link at that instant. We show in the next section that the situation of multiple losses

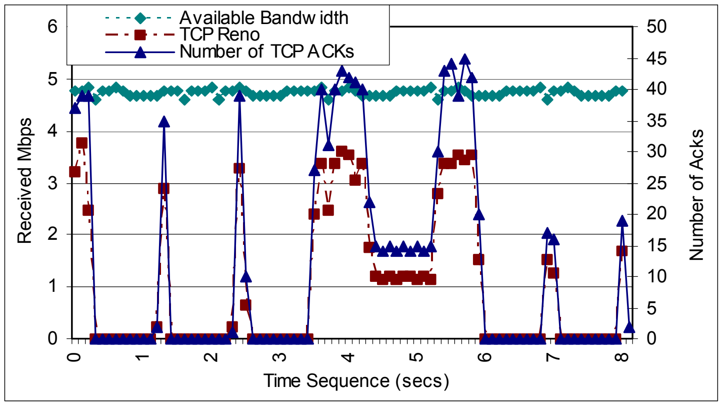


Figure 2.5: Number of TCP ACKs received compared to the data bytes

in a congestion window, often leads to timeouts in Reno and NewReno, that are a lot more expensive in terms of lost throughput than the mere loss of data packets.

2.3.2 Tahoe Outperforms Reno in 802.11 Wireless Links

Tahoe, Reno and NewReno operate identically in the normal operation mode (no losses), but differ in their congestion control algorithms. Figures 2.6, 2.7 and 2.8 depict congestion control algorithms of Reno, NewReno and Tahoe respectively, triggered in response to multiple losses of Figure 3.2. All three versions implement fast-retransmit where the segment is retransmitted upon three duplicate ACKs. (They also implement limited-transmit (RFC 3042) , where the first and second duplicate ACKs trigger transmission of up to two data segments over the congestion window.). They differ in how they adjust the congestion window after fast-retransmit

Tahoe scales it down to 1 segment, forgetting all about higher sequence number segments that were already sent (at instant A in Figure 3.8). No more packets are sent until an ACK arrives confirming the retransmitted packet. Normal operation subsequently resumes in slow-start mode.

Reno and NewReno cut down congestion window in half after *fast-retransmit* (at instant A in Figures 2.6 and 2.7) and "fill the pipe" with new data packets until a new ACK arrives. This is the *fast-recovery* mode of operation where the congestion window is incremented by one segment for each duplicate ACK. New packets are sent when congestion window exceeds the number of already outstanding packets

Reno exits *fast-recovery* and resumes normal operation when the first non-duplicate ACK arrives recovering from the first loss. In case of multiple losses (before instant A), Reno's fast-recovery ends when there are more outstanding packets than the congestion window (at instant B in 2.6). The congestion window is cut down further when recovering from the subsequent losses. In Figure 2.6, Reno enters congestion control to recover the 2nd loss at instant C, and exits at instant D. Since the congestion window is small, no new packets are sent between instants B and D. Subsequently there are no more ACKs and the deadlock situation results at instant D. It ends in a timeout.

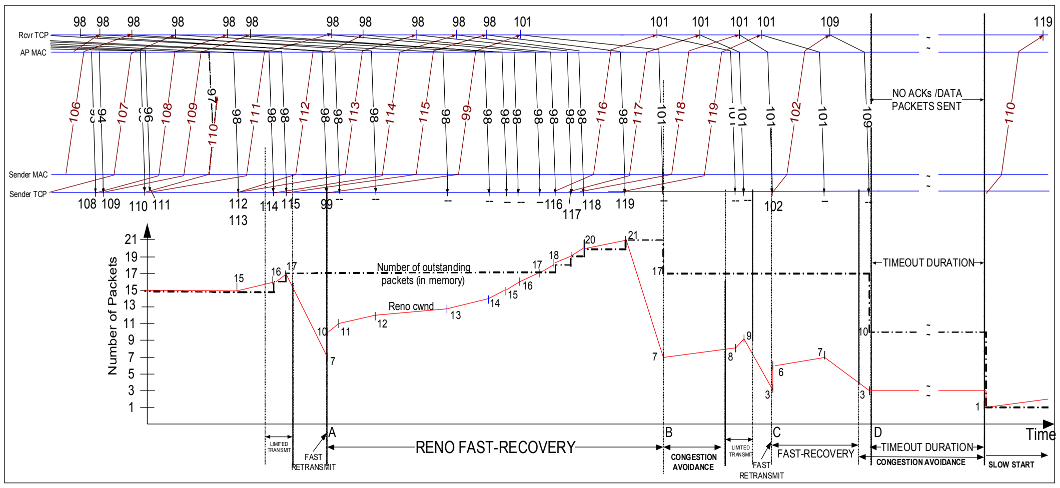


Figure 2.6: Congestion Control in TCP-Reno following multiple losses of Figure 3.2

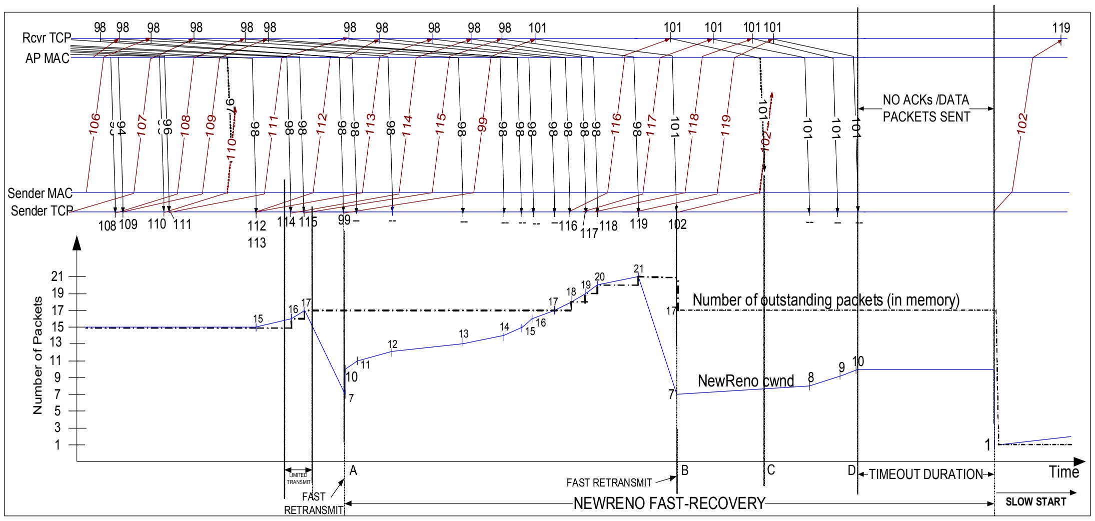


Figure 2.7: Congestion control in NewReno following multiple losses of Figure 3.2

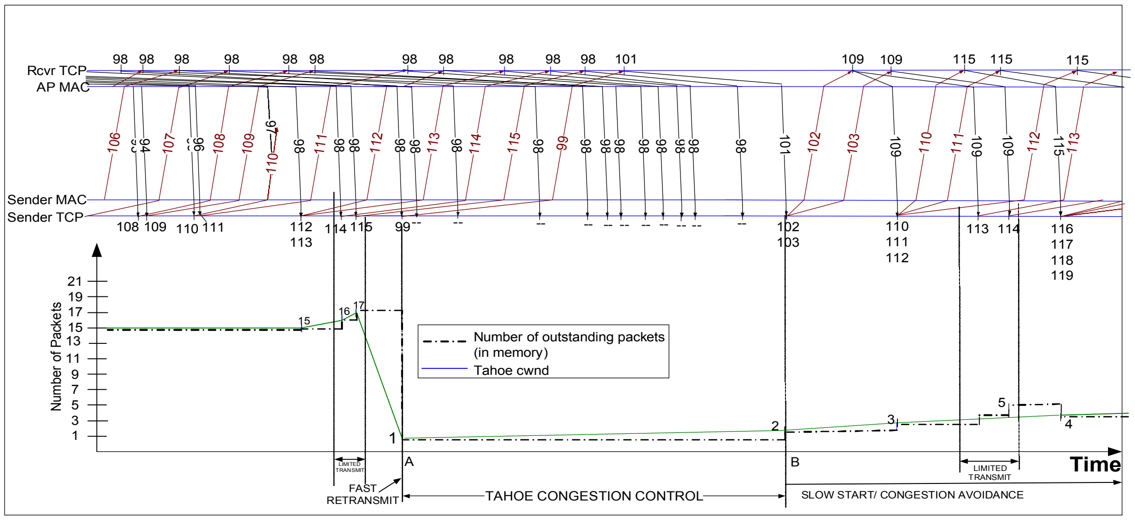


Figure 2.8: Congestion control in Tahoe following multiple losses of Figure 3.2

To overcome the deadlock, NewReno [72] continues in fast-recovery until all losses (that occurred before instant A in Figure 2.7) are recovered. The segment is retransmitted when the first ACK indicating it arrives (at instant B). The congestion window is halved after each retransmission. NewReno recovers all losses if and only if all retransmissions are successful. A timeout occurs otherwise. Figure 2.7 depicts a deadlock situation that occurs in NewReno when a packet retransmitted during fast-recovery is lost (packet #102 at instant C). Subsequent duplicate ACKs grow the congestion window, but this not sufficient to send new data packets. With no more data packets, no ACKs are triggered and a deadlock situation occurs (at instant D in Figure 2.7)

The comparison of instantaneous goodputs of Tahoe, Reno and NewReno in the course of the 1MB file transfer in the said wireless scenario is depicted in Figure 2.9. Despite being the least optimized version of TCP, Tahoe completes the file transfer in a significantly shorter time compared to Reno and NewReno and the dynamics reveal the following insights:

- *Reno suffers multiple timeouts*: With the frequent occurrence of multiple losses in a TCP congestion window, Reno *deadlocks* several times that end in timeouts. The 1-second duration of each interval is due to the *minimum retransmission timeout* setting (*minrto_*)
- *NewReno also suffers multiple timeouts:* During fast-recovery, NewReno sustains a "full transmission pipe" by sending new packets. Thus the probability of self-interference is still large during fast-recovery. In the example of Figure 2.9, deadlock situations occur three times in the course of the 1MB file transfer, and end in timeouts (after minrto_ of 1 sec). They all occur due to loss of the first retransmission, or of one of the packets sent during fast recovery
- *A low throughput is sustained in Reno/NewReno in some timeout intervals*: Figure 2.7 captures the dynamics of TCP NewReno/Reno the scenario when the retransmission at the start of congestion control is lost (at instant B). Despite this, duplicate ACKs #333 sustain the growth of

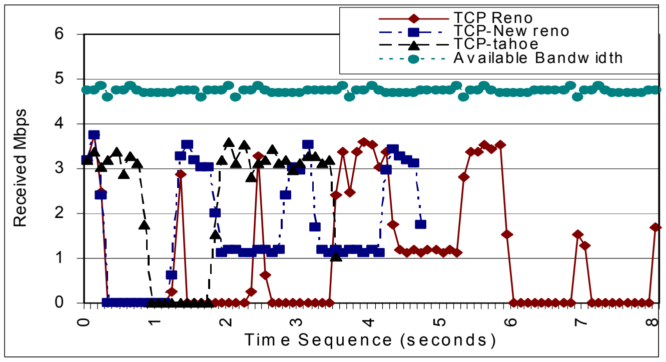


Figure 2.9: Instantaneous received rates of Tahoe, Reno and NewReno during 1MB file transfers

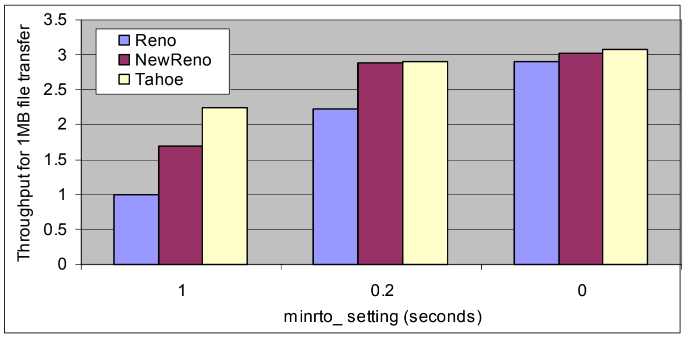


Figure 2.10: Performance of various TCP flavors for different minrto_ settings

the congestion window, and consistently trigger new data segments. In this example congestion window and the number of outstanding packets reach *154 segments* at the end of fast-recovery. With a single packet in transit at a time, the likelihood of self-interference diminishes to *zero*. But with stop-and-wait approach during this interval, the goodput drops to a minimum. In Figure 2.9, Reno starting at 4.4 seconds, and NewReno at 1.9 and 3.3 seconds experience this situation

- *Tahoe outperforms Reno. Outperforms NewReno considerably for minrto_ = 1 second.* Tahoe operation following multiple losses in a congestion window is depicted in Figure 2.8. By *not* implementing fast-recovery and resuming operation in slow-start soon after fast-retransmit (at instant A), Tahoe reduces the loss likelihood due to self-interference, improving the resilience to loss recovery. Several duplicate transmissions of data segments could ensue. But Tahoe does not deadlock and timeout as long as the retransmission is successful. Hence the likelihood of a deadlock in Tahoe is far lower than in Reno and NewReno. Tahoe's gain in goodput during the additional timeout periods of Reno and NewReno offsets the bandwidth wasted by redundant retransmissions and the lack of "pipe-filling" during loss recovery

2.3.3 Effect of minrto_ on TCP Performance

In the wireless LAN scenario considered here, the round-trip time fluctuated between 7 and 30 milliseconds. RFC 2988 [73] stipulates a minimum timeout duration (*minrto_* in NS2) of 1 second. This was to avoid spurious timeouts and retransmissions in TCP in wired nets with large fluctuating round trip times. In our scenario, Reno and NewReno both waste several 1-second intervals in a deadlock before the timeout occurs and resumes the sending rate.

More recent TCP implementations set minrto_ to 0.2 seconds. Figure 2.10 compares the net throughputs of Reno, NewReno and Tahoe for various minrto_ settings. minrto_= 0 implies that the retransmission timeout duration is completely based on the estimated round-trip time. With these settings, Reno and NewReno still experience the same number of timeouts but gain in throughput because of the shorter time spent in deadlock, and Tahoe's gain over Reno and NewReno diminishes.

2.3.4 Related Work

All TCP enhancements proposed for wireless networks, are extensions of either TCP-Reno or TCP-NewReno. To the best of our knowledge ours is the first attempt to investigate the dynamics of operation of different TCP versions in an 802.11 scenario.

The enhancement protocols may be clearly categorized into those for cellular networks and those for multi-hop 802.11 networks. Cellular networks however do not operate in a shared medium, and hence do not suffer the MAC problem. Some papers addressing TCP in multi-hop wireless have proposed limiting the congestion window in decreasing proportion to the number of hops, to reduce interference [25]. But this drastically limits TCP throughput in the multi-hop scenario, and the available bandwidth will be underutilized.

2.3.5 Summary

In this section we delved into the dynamics of TCP operation to explain the reason why TCP performs poorly when affected by merely 3% losses due to self-interference in a wireless LAN. We simulated various flavors of TCP and found that the problem was due to the incompatible operation of the popular *fast-recovery* algorithm in 802.11 wireless links. Fast-recovery greatly improves TCP performance in wired nets by maintaining a nominal sending rate while a lost packet is being recovered. However for shared medium 802.11 links, this operation of fast-recovery introduces relatively high likelihood of packet loss because self-interference continues to happen at the same rate. Losses in this period cause a deadlock situation that finally ends in a timeout. We traced the packet dynamics of TCP-Reno and TCP-NewReno that implement fast-recovery, to demonstrate these derogatory effects. Then for comparison, we analyzed the dynamics of TCP-Tahoe that does not implement fast-recovery and showed that it undergoes fewer deadlock situations and timeouts and hence gains significantly in throughput.

Basically, here we have exposed the TCP problem that its positive acknowledgements interfere with the DATA packets and degrade performance. The problem would reduce if there were fewer acknowledgements interfering with the TCP DATA packets. In the next section we describe a simple TCP modification we implemented and evaluated to reduce ACKs and hence alleviate the self-interference problem.

2.4 "Skip-ACKs" modification to TCP

TCP acknowledgements are cumulative in nature, that is, an ACK for *byte i* implies that all *bytes* until (*i-1*) are correctly received (in the TCP implementation in NS2 simulator the ACK for *i* implies all *segments including i* are correctly received). We capitalized on this cumulative nature and introduced ACK skipping to reduce the number of ACKs traversing the 802.11 link, and hence reduce self-interference.

2.4.1 Protocol Description and details

The "ACK-skipping" algorithm is rather simple and is implemented at the receiver. In the normal mode of operation where DATA packets are arriving in sequence, the TCP receiver skips sending ACKs. Say the number of ACKs to skip is *k*, the receiver only sends the first of (*k+1)* ACKs. Two tracking variables are introduced at the receiver - one maintains the number of ACKs to skip (*k*), and another maintains the sequence number of the last ACK sent (*prevACK*). The ACK sequence number is incremented every time one is generated, and the skip decision is made only at the final sending step at the receiver. An ACK is sent only if *currentACK > prevACK + k*;

To allow the congestion window to stabilize, ACK skipping is set to start only after TCP sequence number crosses a threshold (chosen as 50 here by trial and error).

ACKs were not skipped when duplicate ACKs were to be sent, or a received data packet was out of order (indicative of a possible loss). These measures were taken to ensure that the ACK-skip modification did not interfere with TCP's default error control algorithms. Further, ACK skipping started only after the received DATA sequence number reached a certain threshold (set to 50 here) since in slow-start mode, the flow rate depends on the number of ACK segments received.

Skipping alternate ACKs reduces traffic load at the AP by a factor of two and the AP contends half as much for channel access. This reduces interference with data packets, hence improving TCP throughput. However, TCP throughput is directly related to the growth of TCP congestion window, which in turn depends on regular arrival of TCP ACKs. Thus skipping too many ACKs impedes *cwnd* growth and curtails TCP throughput. Simulation and experimental results given in the next two sections confirm this insight.

2.4.2 Evaluation Methodology

This protocol was validated in NS2 simulations and also implemented in the ORBIT wireless test-bed. The following traffic parameters for both simulations and experimentation:

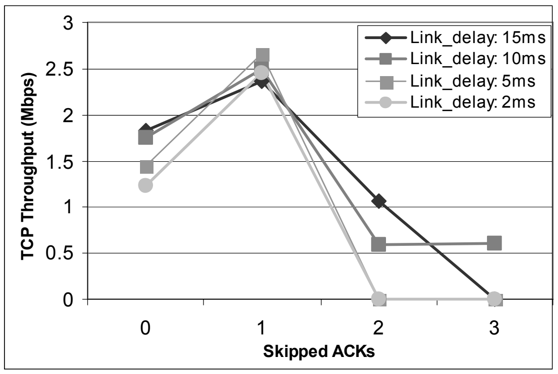


Figure 2.11: NS2 – Throughput variation with skipped ACKs and (wired) link delays; Single TCP flow; Long-lived TCP connection; NO MAC retries

1. Length of TCP flow
2. MAC retries
3. Number of ACKs skipped
4. Number of simultaneous flows.

Below we present some details of simulation and ORBIT implementation, but most of the detail is in the Appendix.

Details of NS2 simulations: Long-lived (short-lived) TCP connections are set up with 10MB (1MB) file transfers using the File Transfer Protocol (FTP). Each TCP throughput value published here is an average of 5 runs, obtained by varying FTP start times. TCP Reno is used in all experiments with the duplicate ACK parameter set to 3. Further details of setup are in Appendix section 6.2.

Details of ORBIT test-bed experiments: To reflect short-lived and long lived TCP flows, file sizes of 100kB and 6MB are used respectively. Home-grown TCP traffic generators – tcptest for TCP and NPM for UDP are used to generate traffic and collect instantaneous throughput traces in various scenarios. Details of experiment setup are described in Appendix Section 6.3

2.4.3 Performance of TCP with "skip-ACKs"

Here we will describe the NS2 simulation results and ORBIT test-bed implementation results separately, and then summarize the overall performance of the protocol enhancement.

2.4.3.1 NS2 simulations

The various results are depicted in Figures 2.11 – 2.15. Skipping TCP ACKs is found to indeed improve TCP throughput for most scenarios. All scenarios gain from skipping at least 1 ACK. Figure 3.11 compares TCP throughput of long-lived TCP connections in the absence of MAC retries for different link delays with increasing ACK skips. Since there isn't a significant difference in performance for different delays, we explain results obtained with 2ms link delay.

Figures 2.12 – 2.15 depict the main results. We explain performance observed with single and multiple flows, short-lived and long-lived flows, and with and without MAC retries.

A. Case with a Single TCP flow

The results in Figures 2.12 and 2.13 show that with no MAC retries, and default TCP implementation of one ACK per DATA packet, TCP achieves a meager throughput of around 1Mbps with both short-lived and long lived flows. One ACK skip almost doubles the throughput (98% gain) for both short-lived and long lived flows. This is because the reduced MAC traffic simply cuts MAC interference in half reducing the bandwidth consumed by the ACKs, and more importantly reduces MAC interference losses. For higher ACK skips, many sessions failed to complete despite a long simulation time. This can be explained from a combination of two effects – TCP congestion window starving from lack of regular TCP ACKs as well as the loss of cumulative ACKs due to MAC failures. Multiple timeouts severely reduce TCP throughput.

With MAC retries, the 3% loss rate due to self-interference is handled by the MAC itself. The throughput gains are because ACK skipping reduces the rate of increase of packet bursts sent by TCP to the MAC layer. We observed that TCP throughput degradation with MAC retries occurred mostly from overflow of interface queue at the TCP sender node during the slow-start mode of operation. Results depicted in Figures 2.14 and 2.15 show consistent throughput gain with ACK skipping. The gains in throughput are curiously due to TCP dynamics during slow-start. With no ACK skipping, the *cwnd* grows exponentially in slow start mode. Because of the stop-and-wait simplex characteristic of the 802.11 link, data packets are held up in the interface send queue and a large burst of TCP data packets could cause a queue overflow. When ACKs are skipped, *cwnd* experiences slower growth because of the reduced number of incoming ACK segments. Hence TCP sends smaller bursts of packets, reducing or even eliminating MAC queue overflows. This is why we see up to 30% gain in TCP throughput with ACK skipping.

B. Case with multiple TCP flows

Contention among multiple wireless nodes causes collisions in addition to the self-interference problem. Simulations showed that the most common cause of packet loss is a combination of both problems. Referring Figure 2.1, Node N1 transmits a TCP data packet to the AP at the same time when the AP transmits a packet to node N2. N2 sees a garbled signal due to simultaneous signals from N1 and the AP (collision), while the AP fails to detect N1's transmission (*simultaneous-send*). Thus the AP is found to contend for channel access far more often than other nodes. This corroborates our hypothesis that the AP will have many more packets to transmit than any one of the other wireless nodes. With MAC retries, ACK skipping results in throughput gains for the same reasons explained in the case of a single TCP flow.

C. Short-lived and Long-lived TCP connections

Figures 2.12 - 2.15 show that the pattern of throughput improvement is very similar for short-lived and long-lived TCP connections. They differ only in the extent of their gains. It may be inferred that longer the TCP connection, more will be the gain from ACK skipping. TCP congestion window achieves steady-state in the long run, when its size is close to the delay-bandwidth product of the network between source and destination. For the network setup here, it would be the product of the average RTT and the net bandwidth available. However, NS trace files show that with packets lost due to MAC contention and queue overflows, the congestion

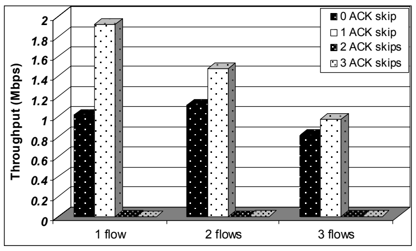


Figure 2.12: Short-lived TCP flow with NO MAC retries

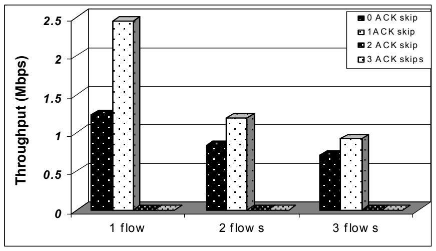


Figure 2.13: Long-lived TCP flow with NO MAC retries

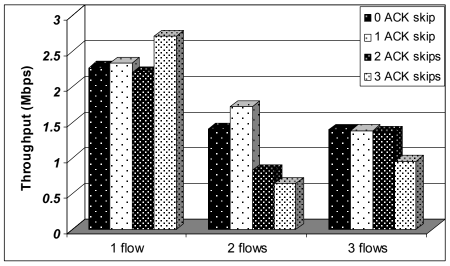


Figure 2.14: Short-lived TCP flow with MAC retries

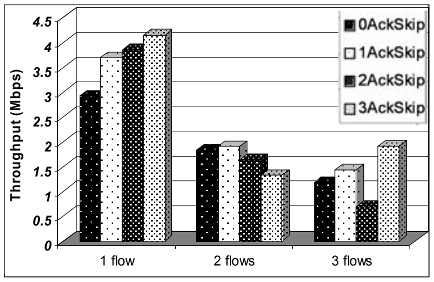


Figure 2.15: Long-lived TCP flow with MAC retries

window seldom reaches this steady state.

In the next section, we evaluate the skip-ACKs modification of TCP in the ORBIT test-bed

2.4.3.2 ORBIT test-bed implementation

Numerous research papers have been published in the area of wireless networks. Most of them test performance and new protocols with network simulators such as NS2 and OPNET. These tools are excellent sandboxes to check correctness and understand the detailed operation of protocols. However just this not sufficient validation, since these simulation tools for often fail to capture the exact characteristics of the protocol in wireless networks because of the difficulty in fully representing the physical medium. NS2 provides a relatively abstract implementation for the physical layer of the network stack thus missing many important PHY level mechanisms such as autorate and capture. Further it supports just the basic features of MAC protocols such as IEEE 802.11, hence making it essential to evaluate protocols by real world experimentation. These motivated us to consider evaluation of the skip-ACKs method in the ORBIT test-bed

The description of testbed setup and parameters are given in the methodology section in an earlier section.

Figures 2.16 - 2.19 present results from the experiments. First it is important to notice that the TCP throughput obtained with no adaptation was itself significantly higher than in simulations. The primary reason for this discrepancy is from the implementation of the interface queue in NS2. With the default setting of 50 packets, TCP packets were lost while in slow-start due to MAC queue overflows. This caused timeouts in TCP that significantly degraded overall throughput. This phenomenon did not happen while operating real-world TCP, as the operating system in the node acted as an intermediary between the TCP socket buffer and the interface queue in the network card. The OS delivered a packet from the TCP send-socket-buffer to the network interface queue, only when the interface driver set a memory availability flag. If the TCP socket buffer was full, no new bytes were accepted from the application (the send() function returned an error in the application operating the TCP socket).

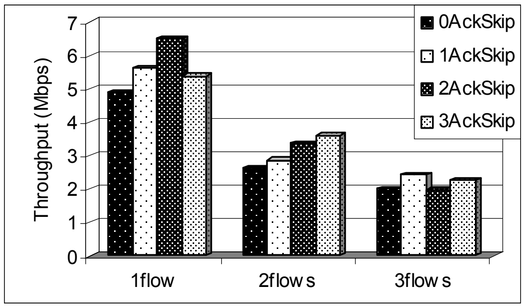


Figure 2.16: Short-lived TCP flow WITH MAC retries

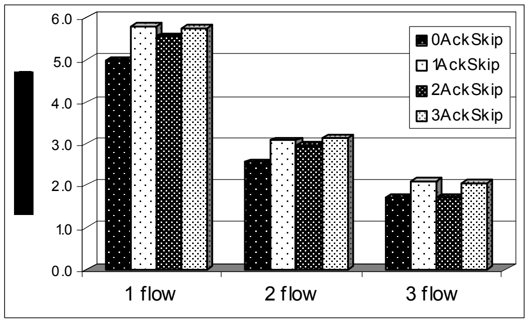


Figure 2.17: Long-lived TCP flow with MAC retries

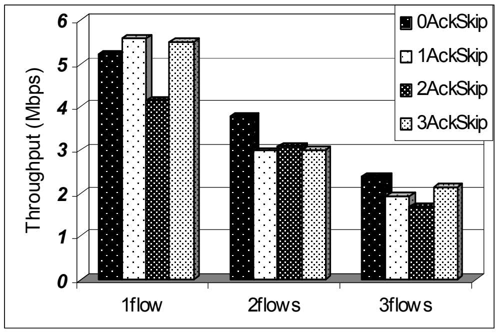


Figure 2.18: Short-lived TCP flow NO MAC retries

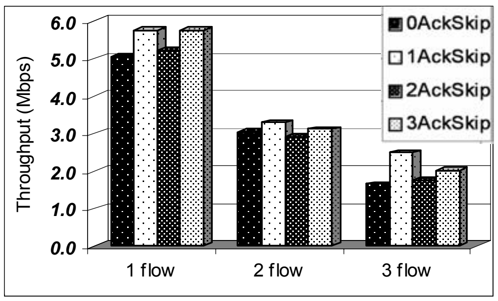


Figure 2.19: Long-lived TCP flow NO MAC retries

Since base throughputs in NS simulations differed from those in testbed experiments, we compare *patterns* in gains achieved rather than the actual gains themselves.

A. Case of enabled MAC retries

The Atheros cards had a default MAC retry setting of 16 that could not be changed. Hence that value was used even with Cisco cards. On the other hand, the maximum retries used in NS simulations was 8.

Just as in simulations, ACK skipping consistently improved TCP throughput (in this case with MAC retries) even with multiple simultaneous flows. MAC retransmissions were tracked by means of standalone sniffers. These sniffers comprised of Atheros cards in monitor mode and the *tcpdump* software. The particular gain patterns for short lived and long lived flows differed from those of NS simulations. This was probably due to the NS2 artifact of interface queue overflows during TCP slow start, something that did not occur in real experiments. The throughput gain in real experiments seemed to come directly from reduced MAC contention due to fewer TCP ACK packets.

Long lived flows saw consistent throughput gain even with 3 skipped ACKs, whereas the gain dropped with such high ACK skips for short-lived flows. This could be because short-lived flows spent a higher percentage of their operation in slow start mode. In this mode, increase in TCP congestion window was proportional to the actual number of incoming ACK segments, even if they were cumulative ACKs. In the congestion-avoidance mode on the other hand, increase in congestion window was proportional to the number of data segments acknowledged. With three or more ACKs skipped, short-lived flows experienced ACK starving and hence had reduced throughput.

Overall, ACK skipping helped the case with MAC retries. Both short-lived and long-lived flows gain from this adaptation.

B. Case of disabled MAC retries

We reiterate that it was not possible to produce the case when MAC retries were completely disabled in the wireless infrastructure network, as this feature was not supported in Atheros cards that were used for AP. However MAC retries could be disabled in non-AP wireless nodes where Cisco cards were used. For the traffic scenario considered, this meant that MAC retries were disabled for TCP data segments, while the TCP ACK segments that were relayed by the AP enjoyed MAC retries. This was also confirmed with a standalone sniffer (Described above). In liu of this, results from simulations and experiments for this case cannot be compared.

From these results we infer that ACK skipping was more favorable for long lived rather than for short-lived flows, when MAC retransmissions were available only for TCP ACK packets.

C. Other observations

The graphs indicate that for the default case with no ACK skips, TCP throughput was better with no MAC retries for TCP data segments than with MAC retries. This could be because link layer retransmissions produce variations in RTT for TCP, reducing its performance. This could possibly imply that TCP does a far better job handling MAC congestion by itself rather than with link layer retransmissions. This observation requires further study.

There was much better channel utilization with multiple simultaneous flows that with a single flow. This can be explained as due to the 802.11 backoff overhead. The 802.11 DCF backoff mechanism caused an average overhead of 300ms. With a few flows this overhead reduced as contention slots were staggered. When the likelihood of same slot selection was still reasonably small, there was a throughput improvement. However as the number of flows increased, the likelihood of same slot selection also increased, resulting in more MAC failures and subsequent degradation in throughput.

2.4.3.3 Summary of observations with "Skip-ACKs" TCP modification

Results in both simulations and test-bed experiments demonstrated consistent gains with ACK-skipping over default TCP [3][4]. The simulation results capture the case without MAC retries. Here one ACK skip produced the highest gains, while higher ACK skips degraded throughput since TCP flow control depends on the regular pace of returning ACKs. Higher ACK skips also placed a heavy burden on the occasional ACKs and their loss caused premature timeouts in the sender that in turn degraded throughput. With MAC retries, ACK skipping helped consistently. The highest gains were seen with a single long-lived flow in both NS2 simulations and ORBIT test-bed evaluation.

The single flow results confirm our hypothesis of significant throughput degradation with a 3% loss likelihood. Without MAC retries, TCP itself experienced the 3% loss rate and scaled back its sending rate. A single ACK skip produces significant gains because of lesser interference to DATA packets. When MAC retries were used, the 802.11 losses were "hidden" from TCP. The consistent throughput gain with ACK skipping was because of a controlled flow rate increase with fewer ACKs during slow-start. This reduced the packet bursts within the sender node, causing fewer overflows at the MAC send queue.

With multiple flows, cross-interference dominated over self-interference issues. In the topology we considered, there were more wireless nodes contending for the channel and the AP contended to relay all the TCP ACKs to the respective TCP source nodes. Skipping 1 ACK (in all flows) reduced the AP transmission load in half, causing the AP to contend less often for channel access, and hence reduces collisions due to ACKs. But since ACK skipping only reduced interference from the AP traffic, the overall gains with ACK skipping diminished with multiple TCP flows.

Considering all these gains, it is evident that self-interference in TCP over wireless LANs, cannot easily be mitigated because of its tightly intertwined error and flow control mechanisms. Here the window-based flow control algorithm is "clocked" by the positive acknowledgement, and hence heavily depends on their pace.

While MAC retries could hide the interference losses from TCP, they could result in delay-variance that could also degrade TCP throughput. Chun and Ramjee[1] observed this effect in the context of 3G channel scheduling in cellular networks. We demonstrate these effects in detail in the multi-hop scenario with both UDP and TCP performance in a fluctuating noise scenario.

These problems are occurring because of TCP sender's heavy dependence on regularly arriving ACKs for flow control. We show in the next section that this combined operation of flow control and error control algorithms, results in very poor TCP performance for time-varying noise in the 802.11 link.

2.5 TCP Performance in time-varying noise scenarios

We began the evaluation with emulation of such an environment on the ORBIT wireless test-bed and then created the same scenarios in NS2 simulations.

2.5.1 ORBIT test-bed experiments

Setting up a fluctuating noise-prone environment was a significant challenge on the ORBIT test-bed. We will first describe the methodology for setup and then explain performance results.

2.5.1.1 Experiment methodology

In the ORBIT 64 node test-bed we selected a pair of nodes that were close to one of the noise antennae in the grid. We evaluated various noise power levels to obtain a test scenario where noise produced partial throughput. However the granularity in the noise injection system was relatively course, and either produced full throughput (the noise did not reduce SNR much) or produced no throughput (the noise invariably reduced SNR below threshold). So we had to settle in for the latter, and introduced on-off noise that had the effect of full bandwidth or zero bandwidth link. We wrote a *Ruby* script to generate a time-varying noise scenario with changes every 5 seconds. Lower granularity on-off noise was not consistently available with the web-based control of the noise generator due to signaling delays.

When the noise was on, the antennae transmitted random signals that had an additive Gaussian noise characteristic. This noise power (transmitted by the signal generator) was selected as -20dBm. We chose a receiver node close to a noise antenna so as to be able to influence the received traffic with random noise.

We used the *tcptest* traffic generator (described in Appendix) to transfer a 1MB file over the default TCP socket in the Linux 2.6.10 kernel. The *tcptest* receiver collected traces of bytes received in 1-second intervals. Next the NPM traffic generator (described in Appendix) was run in the same noise environment and bytes received in 1-second intervals were logged along with the full trace of packet sequence number, packet size and received timestamps. NPM was configured to transmit saturating UDP traffic of constant rate and packet size, so that the link was always saturated (6Mbps is the available link bandwidth over a 11Mbps physical channel rate). The TCP and UDP traces of 1-second interval bytes received were compared to obtain insight into the efficiency of TCP operation.

2.5.1.2 ORBIT test-bed results

The effect of on-off noise on TCP and UDP throughput in the ORBIT test-bed is depicted in Figure 3.20 (same as Figure 1.5 in Chapter 1). The UDP curve shows pulses of good throughput opportunities. UDP continues to send packets at the same rate the whole time, but when the noise is on, the low SNR causes the receiver to fail to decode most of the received packets. These packets (which fail the MAC CRC check) are dropped by the MAC in the interface card, and are not seen by the host computer, let alone in the transport layer.

These low SNR cases also happen with TCP packets resulting in several losses. However unlike UDP that does not have any flow control, TCP with flow control scales back its sending rate in response to the losses. This causes TCP to not use multiple "noise-free" opportunities of the channel. TCP is also very slow to adapt after the channel returns to normalcy (takes over 5 seconds). This demonstrates the shortcoming of TCP's flow control algorithm over wireless links, where the error characteristics can rapidly fluctuate.

While real-world experimentation provides a proof of concept, it is hard to explore protocol operation characteristics here. ORBIT test-bed had other problems of time-granularity for noise, etc. Hence we simulated the same ORBIT environment in NS2 simulations to evaluate TCP over time-varying links in greater detail.

2.5.2 NS2 simulations

2.5.2.1 Methodology

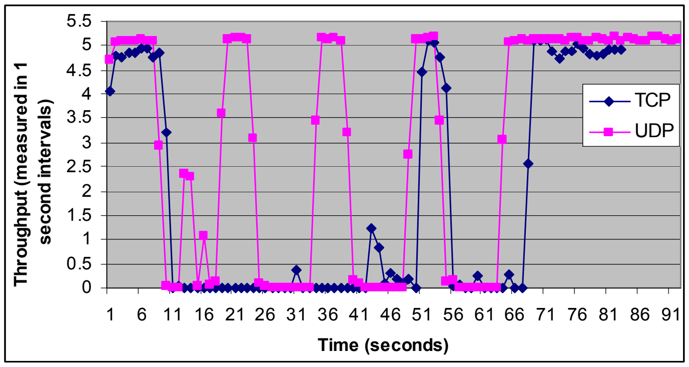


Figure 2.20: TCP and UDP performance with 5-second on-off noise in the ORBIT test-bed

The same wireless LAN topology of Figure 3.1 was used and the same settings were made as described in earlier wireless LAN experiments. In addition, random noise in wireless links were simulated to have similar effect as in the ORBIT test-bed. We created an AWGN propagation model, where for each incoming packet, a random value for noise power is generated using a bipolar Normal (Gaussian) random variable of mean 0 and variance set to the noise power value (set to $9.3X10^{-8}$W for the given results). The slow-varying wireless link is produced by introducing the random Gaussian noise in alternate 1-second intervals, and to produce the slow-varying link, noise is introduced in alternate 0.1-second intervals. TCP-Reno is used for evaluation.

2.5.2.2 NS2 simulation results

Figures 2.21 and 2.22 show the performance results over slow-varying and fast-varying noise conditions simulated in NS2, produced by injecting on-off noise at the wireless receivers. Here noise injection causes the channel noise to fluctuate between 0 and 40%. This is evident from the UDP received rate plot, which fluctuates between about 5Mbps (when noise if off) and about 3Mbps (when noise is on).

TCP performance is similar in both slow and fast-varying noise conditions. It mostly shuts down operation when the noise is on, and fails to use the "noise-free channel" opportunities. Self-interference in the noise-free channel causes spike-like operation of TCP-Reno, and the peak TCP throughput is limited to a fraction of the full bandwidth available.

Certain differences between the ORBIT result and NS2 result need mention. The ORBIT kernel uses delayed ACKs that reduce the frequency of ACKs as a result of which the interference to the data packets is reduced. Hence TCP instantaneous received rate on OBIT (plotted in Figure 3.20) achieves higher peak throughput than in the NS2 simulations.

It is thus evident that TCP shuts down operation over high error rate links. In literature, this is well known TCP behavior, where it performs poorly for error rates higher than 3%. In scenarios considered here, the error rates fluctuate between 0 and 40%.

Again here, the "shut-down" behavior of TCP is because it wrongly interprets the losses as due to network congestion, and scales back the flow rate unnecessarily. As losses continue to occur, it backs off exponentially several times to allow time for the network to recover from congestion.

During this time, it is clear that fails to use the significant bandwidth opportunity still available in the link. Note that despite changing noise in the link, the link bandwidth itself remains constant at about 5Mbps. The plots of received rate show fluctuations because of changing goodput at the transport layer, since corrupted received packets are simply dropped and not sent to the transport layer. This "bandwidth-available-despite-loss" is a significant shift in characteristic of the wireless link, from traditional wired links for which TCP is optimized.

While MAC retries could recover losses in the link, they cause large fluctuations in transmit delays for each packet. This does not help with time-varying noise, since the significant delay-variance it introduces will degrade TCP performance significantly.

In these various sections we have demonstrated that positive acknowledgements of TCP degrade its performance over wireless LANs and combined flow control and error control cause limited gains with ACK skipping, and poor performance when there is time-varying channel noise. The slow flow rate adaptation of TCP also misses several opportunities of "noise-free" channel conditions.

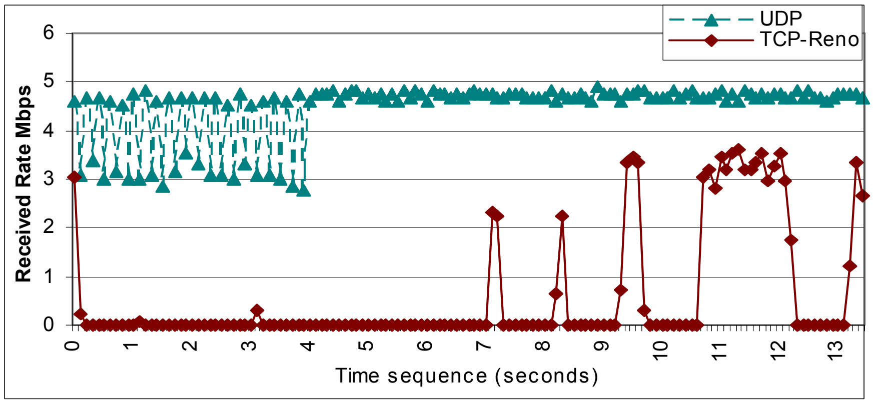


Figure 2.21: Instantaneous TCP received rate over a fast-varying wireless link

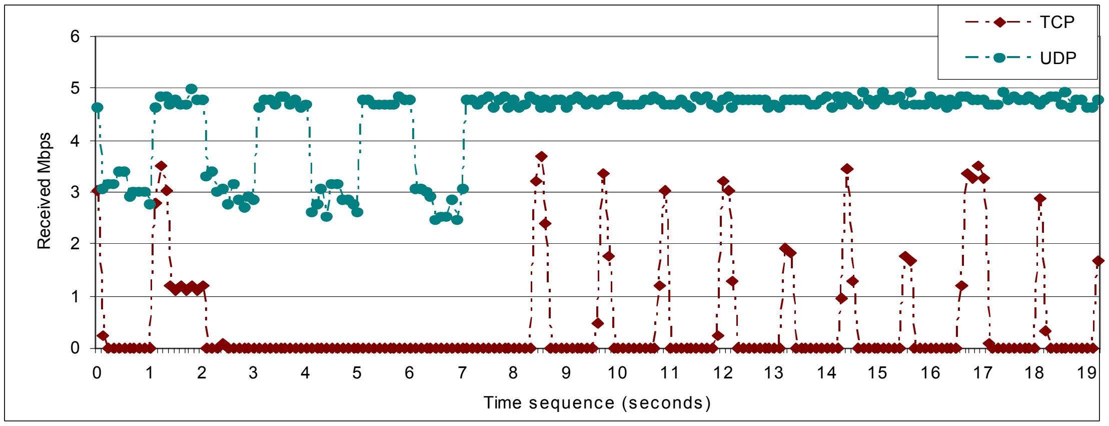


Figure 2.22: Instantaneous TCP received rate over a slow-varying wireless link

These observations motivated the design of the CLAP protocol to reduce file transfer delays over wireless LANs.

2.6 Summary

In this chapter we identified and evaluated the TCP self-interference problem. We showed that for a single TCP flow, despite a small likelihood of interference losses (3%), TCP pays a heavy price because a popular TCP algorithm (fast-recovery) used during congestion control, aggravates self-interference in wireless LANs. A simple "skip ACKs" modification to TCP to reduce ACK interference achieves limited gains because of the tight coupling of error and flow control algorithms in TCP.

We showed that this same reason causes TCP to shut down operation in time-varying high loss rate scenarios, despite bandwidth being available. Hence overall TCP performs poorly over 802.11 wireless LANs because of combined error and flow control algorithms and positive acknowledgements.

Chapter 3
Architecture and Design Considerations for Cross Layer Transport

In this chapter we describe the new cross-layer architecture and the CLAP protocol proposed as a general solution for reliable file transfer over wireless networks.

Various insights gained from TCP and UDP performance in 802.11 networks, are key to the design of the CLAP protocol. In general we observed that simple unreliable UDP always “adapts” to changing bandwidth, because channel errors do not induce flow rate reduction. On the other hand, TCP throughput suffers severely from interference losses, bandwidth changes and high error rate links. TCP performs poorly when its own positive acknowledgements cannot correctly capture the bandwidth available end-to-end. This situation arises in wireless networks because of “link anomalies” that often change the pace of acknowledgements or cause losses. Hence the following lessons are derived from an analysis of TCP performance in wireless scenarios:

(a) *Reduce dependence of flow control on positive acknowledgements*, since returning acknowledgements can often get delayed or lost in wireless networks.

(b) *Minimize dependence on round trip time estimation*, since rapidly fluctuating bandwidth in 802.11 networks also fluctuates round trip time.

(c) *Decouple flow control from error control*, since the flow rate has to sustain irrespective of losses and thus approach UDP performance.

(d) Reduce number of feedback packets as much as possible, to minimize self-interference in 802.11 networks.

3.1 Design considerations for a transport protocol

Consider the following brute-force reliable file transfer approach that reduces feedback packets to a minimum, and also separates flow control from error control. The sender sends data as UDP packets and transfers the entire file over and over again (data carousal) until the confirmation arrives that the entire file is received successfully. Assume that the end-to-end bandwidth is available by out-of-band means. It may be simple information such as that the transmission is over 802.11b with 11Mbps physical rate that translates to a peak 6Mbps transport layer bandwidth in noise-free, interference-free conditions. The flow control algorithm of the brute-force protocol is merely to send at a large enough constant bit rate (CBR) (at least 6Mbps in the 802.11b case) that saturates the route (since assuming single-path routing, the end to end bandwidth at any instant, is the bandwidth of the slowest link). The receiver sends an acknowledgement only to confirm that all packets in the file are received, and no feedback is sent before that. With this approach the wireless requirements are somewhat satisfied. Because of no self-interference all the available bandwidth is conserved for the DATA packets, and by sending at the peak data rate possible in the network, the full bandwidth is utilized at any time.

However this approach has various flaws that cause poor performance. Despite using full bandwidth, there are too many duplicates (entire file is retransmitted each time) and causes a significantly reduction in throughput, since the receiver waits for a long time for legitimate packets to arrive. Also because of its overly opportunistic approach to data delivery with a large "fixed" CBR rate, queue overflows can occur for many reasons. When there are multiple flows, the available bandwidth for a specific flow is much lesser than the offered load of 6Mbps and excessive packets are dropped due to queue overflows. These could also lead to unfair queue access by some flows. In wireless networks, such queue overflow losses could also occur when the link bandwidth is fluctuating and there is a significant difference in the offered load and link bandwidth available.

Hence while it is desirable to achieve UDP-like high-goodput over time-varying links, the following objectives are essential to maximize throughput:

(a) Design flow control to closely match the slowest-link bandwidth. This would not only improve bandwidth utilization, but will also reduce queue overflow losses.

(b) Design error control algorithm to minimize duplicates while also minimizing self-interference. This because duplicate retransmissions consume bandwidth unnecessarily at the cost of legitimate data packets.

With these insights on transport protocol design, we developed *CLAP – Cross Layer Aware transport Protocol*, with the following design characteristics:

(a) *Decouple flow control and error control algorithms*, so that there is no "unnecessary" scale-back of the sending rate in response to a loss, while there is abundant bandwidth available.

(b) *The flow control algorithm uses supplemental information to estimate bandwidth*, to compensate for not depending on the feedback packets that are now used only for error control. It obtains this supplemental information by leveraging MAC status information.

(c) *The flow control algorithm is rate-based*, so that it adapts sending rate quickly (changes every observation interval) to new bandwidth reported by the cross-layer report. In comparison, window-based algorithms such as in TCP have been found to adapt to averages rather than instantaneous changes [11].

(d) *An error control algorithm that uses aggregate Negative ACKnowledgements* (NACKs) to minimize self-interference and improve resilience to losses. Various optimizations are introduced to minimize duplicates to improve the overall throughput. Key to this improvement is minimizing the dependence on round trip time estimation.

In fact these considerations for CLAP design significantly simplify the transport protocol compared to TCP. TCP uses various adaptive timers and states, to jointly manage rate adaptation and reliability by means of the congestion control algorithm (In recent times, network congestion has dominated receiver-side buffer overload and hence congestion control in TCP has received extensive attention). TCP implements various timed algorithms to perceive losses (fast-retransmit), and scale back the sending rate in proportion to the loss level. The various timers depend on accurate round-trip time estimation. CLAP on the other hand uses the information *already* available in lower layers, instead of "guessing" the network status with complex algorithms. It only needs constant timers to regularly extract status information and send packets at a constant rate in each interval, and the dependence on round-trip time estimates is required only on the receiver side towards the end of file transmission, and does not affect the packet sending rate. Key to the simpler transport protocol design lies in *cross-layer information*. Designed to operate without it, TCP uses its own acknowledgements to match the sending rate to the delay-bandwidth product of the link. TCP's window-based AIMD (Additive Increase Multiplicative Decrease) algorithm for flow control maintains a sliding-window (called *cwnd*) that is increased additively with incoming ACKs and decreased multiplicatively when there are losses. The algorithm is sufficiently complex with an elaborate state machine requiring various timers to be maintained to track ACKs, retransmissions etc. On the other hand, CLAP's flow control constitutes only a few lines of code and does not need a state machine for efficient operation. Figure 3.1 compares conceptual representations of TCP and CLAP operation.

Today's networks are in fact quite capable of providing status information in the form of an "overlay" control plane. Moore's law has resulted in sufficient processing functionality and memory to support cross-layer status extraction. There is also sufficient bandwidth available (~Mbps) to support nominal control overheads due to inter-node status extraction (~kbps). This said, it is important to note that cross-layer techniques can achieve high performance gains only if they are implemented with reasonable complexity [51]. However given the infancy of such unified cross-layer approaches, there are few mechanisms available in literature to achieve this purpose [18].

Based on these considerations, we developed a Cross-Layer software Framework (CLF) to systematically extract intra-node and inter-node status information. A central status daemon in each node maintains general status information for each interface. For intra-node updates, the status daemon forms a liaison between status-extracting and status-providing entities in the network stack. For inter-node status updates, the status daemons run a probe-based protocol among each other. The protocol is per-flow based with the control overhead increasing with each additional flow. However the overhead is nominal at about 3.2kbps/flow for the update frequency and other parameters considered in the current implementation.

In the next two sections, we describe CLAP and CLF in detail. Their evaluation is described in Chapters 4 and 5 in single and multi-hop wireless scenarios

3.2 Cross-Layer Aware transport Protocol (CLAP)

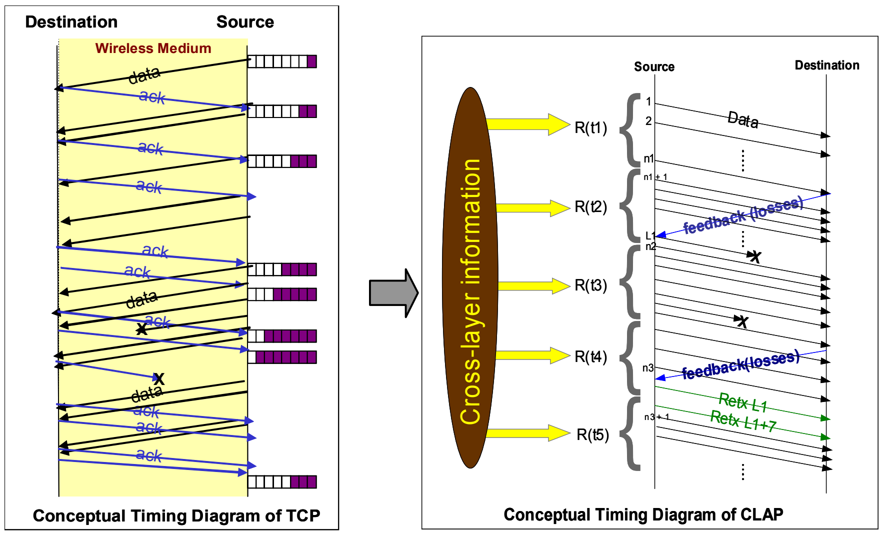


Figure 3.1: Decoupling and a systematic cross-layer design simplify transport protocol design and implementation

CLAP implements procedures for connection establishment and teardown similar to those in TCP, to ensure process-to-process communication. To establish the connection, the sender sends a SYN_ message including the start sequence number and the total number of DATA packets in the file. The CLAP receiver responds with a SYN_ACK_ message to confirm the establishment. When the receiver has received all the packets, it sends a FIN_ message to indicate completion. The sender sends a FIN_ACK_ message in response, and releases the resources allocated to the flow. The receiver waits for a FIN_ACK_ for a specific duration, and if not received by then, retransmits a FIN_ a few times before releasing the resources allocated to the flow.

The reliable transfer of the file over an established connection is handled by CLAP's flow control and error control algorithms. We discuss these in the next two sections describing how CLAP adapts its sending rate based on network conditions with its flow control algorithm, and ensures reliable delivery of all the packets with its error control algorithm.

3.2.1 Flow control

The CLAP sender adapts the packet sending rate to rapid bandwidth changes by leveraging cross-layer information. It uses status parameters from the MAC layer to estimate instantaneous bandwidth and adapts the sending rate in small intervals with a simple rate-based flow control algorithm.

3.2.1.1 Cross-layer parameters to measure instantaneous bandwidth

Our approach is to learn from link bandwidth measured in a past interval, to decide the packet sending rate in the current interval. The idea is to choose an observation interval small enough so that changes in link bandwidth can be tracked well. In general, a smaller observation interval achieves better adaptation accuracy, since any differences in actual bandwidth and offered load are noticed more quickly. In order for CLAP to adapt well, status parameters must be provisioned to represent the following information in each observation interval (a) link bandwidth (b) whether the link is in saturation (c) If the link is not in saturation, what should be the rate of increase

(a) link bandwidth: This determines the number of packets that can be sent in a given interval. In an 802.11 link it depends on the "net channel time" available for transmission because of distributed access, and the "physical channel rate" used to transmit each packet, since there can be auto-rate adaptation where the channel rate changes rapidly. A status parameter provisioned to represent "link bandwidth" must hence be inclusive of both these factors. Many papers addressing QoS in 802.11 links have used "channel busy time" as a measure of bandwidth. However, channel busy time is difficult to measure and provision since the time spent by the MAC in

random backoff mode for each packet must also be accounted. For example, for a single node serving a saturating UDP flow with 1000-byte packets, the channel appears “busy” only for 54% of the time because of various MAC and physical layer overheads (Table 2.1 has the overheads for a unicast packet, and Table 7.1 in Appendix section 7.1 has overheads for a multicast packet). Hence what is required is an easy-to-measure parameter that provides a precise measure of the link capacity in each observation interval.

Instead we provision a simple *MAC outgoing rate* parameter to measure link bandwidth. This not only captures changing channel access time due to other-node interference, but also represents changing physical channel rates since the number of packets sent in an interval is directly proportional to the physical channel rate. Further unlike the “channel busy time” parameter, *MAC outgoing rate* is easy to provision, since it merely requires the incrementing of a counter. To accommodate the possibility of MAC retries (where the same packet is sent until successful), we implemented the counter to increment at the specific location in the program where a new packet is considered for transmission, instead of when after the packet is sent.

(b) A parameter to track link saturation: The MAC outgoing rate depends on the load offered to the MAC layer by higher layers. Hence if there is insufficient load, it will fall short of measuring the full link capacity. An unsaturated link condition must hence be flagged, so that in subsequent intervals, the higher layers can offer a higher load to drive the link to saturation. We introduced the *MAC underflow indicator* to meet this purpose. It is incremented whenever the MAC cannot send because of an empty send queue.

(c) A parameter to indicate how much to increase sending rate: A MAC underflow is an indication to the transport protocol to increase its offered load. To help the transport protocol to decide by how much, we provisioned an *Interface queue space availability* parameter that indicates the space available in the send queue of the MAC interface in that observation interval (in packets).

These three MAC parameters are read and reset at the end of each observation interval. Since there can be multiple interfaces and possibly multiple CLAP processes in a node, access to these status parameters is restricted to be via the cross layer software framework (CLF). CLAP could hence extract MAC status parameters from CLF, asynchronous to the CLF-MAC interaction, and at its own preferred periodicity (lower than or equal to CLF-MAC periodicity).

When there are multiple wireless hops in data path, the same three MAC parameters can still be used to indicate the net bandwidth in the route, since the net bandwidth is the bandwidth of the slowest link. Hence these MAC parameters are general indicators of instantaneous bandwidth end-to-end (assuming single-path routing; additional network layer parameters would be required for multi-path routing). In Section 3.3 we describe implementation aspects of status collection within (intra-node) and across (inter-node) nodes it is sufficient to collect these same parameters in a multi-hop scenario to indicate instantaneous end-to-end bandwidth.

Next we describe how CLAP uses these MAC status parameters to adapt its flow rate.

3.2.1.2 CLAP rate adaptation

CLAP's flow control algorithm is chosen to be rate-based in order to adapt quickly to changing bandwidth, since window-based schemes such as in TCP have been found to adapt much more slowly than rate-based schemes [11]. Window-based schemes also send packets in bursts that cause transient queue overflows, particularly when there are bandwidth fluctuations (such as the result in Figure 4.7(b) in Chapter 3). The problem arises because of a significant difference in the offered load and the link capacity.

The CLAP sender uses status updates of the three MAC status parameters to decide its sending rate in each interval. The sending rate must be sampled more frequently than the rate of change of the link, so that bandwidth fluctuations are tracked accurately. In the current implementation, the sampling rate is fixed to 10 times/second, and is found sufficient for the fluctuating bandwidth scenarios considered (in Chapter 4 in the context of multi-hop wireless scenarios).

The CLAP sending rate in a given observation interval is decided as follows:

```
If (MAC underflow indicator ==0),
    CLAP sending rate = MAC outgoing rate.
else
    CLAP sending rate = 0.5*Interface queue availability
```

CLAP sends packets at a constant rate instead of in bursts, so as to minimize any MAC interface queue overflows. When there is an underflow, CLAP sends half the available queue space in order to be fair to other flows. This value "0.5" is heuristically chosen and achieves quick rate increase, while still maintaining fairness among multiple flows.

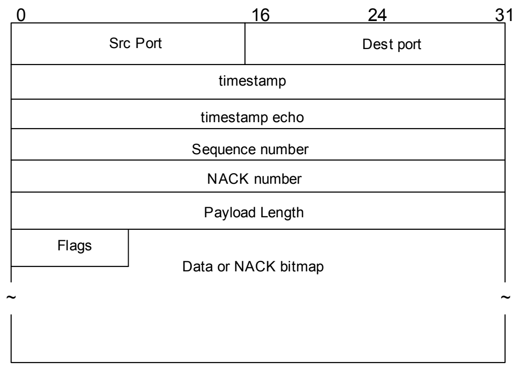


Figure 3.2: CLAP Packet format

Implementation details: The CLAP packet format is depicted in Figure 3.2. It contains a source port and destination port (*src port* and *dst port*) fields to uniquely identify a connection, and to satisfy the basic process multiplexing functionality of a transport protocol.

The *sequence number* field is used to identify each DATA packet uniquely. The *payload length* field is used to indicate the number of DATA bytes in the packet. The *flags* field is currently used only to indicate SYN_ and FIN_ messages, but may be extended for future use.

The current CLAP implementation only supports one-way data transfer in a given connection (as compared to two-way data transfer supported by Full-TCP)

The sender fills the *timestamp* field with the instant at which the data packet was generated. In the *timestamp-echo* field, the sender enters the *timestamp* of the latest feedback received from the CLAP receiver. These timestamp and timestamp fields are provisioned for future use, possibly to derive rough estimates of the round trip time.

Two timers are introduced for flow control, one to periodically extract MAC status updates, and another to send at a uniform packet rate in each interval. If an update is stale (older than 2 observation intervals), CLAP uses the local MAC status values.

Overall the CLAP flow control algorithm is rather simple compared to TCP, since the bandwidth estimation aspect is significantly simplified by using cross-layer status information.

3.2.2 Error Control in CLAP

CLAP's error control algorithm is designed to minimize self-interference and duplicate retransmissions. The CLAP receiver generates aggregate *Negative ACKnowledgements* (NACKs) to report the receipt status of a sequence of packets. Self-interference is minimized by reducing the number of NACKs and increasing each NACK information. Each NACK reports a sequence that is at least *nack_threshold_* long. Such an aggregate feedback method is also used in RMTP for bulk transfer [60].

We implemented two error control algorithms namely, CLAP-*beta* and CLAP-*final* that we describe in the next two sub-sections. They differ in the type of NACKs used. CLAP-*final* significantly reduces duplicate retransmissions compared to CLAP-*beta*, and hence improves overall throughput.

3.2.2.1 CLAP-beta

Here NACKs are sent at regular intervals to report all missing packets up to the highest sequence number received. NACKs are triggered when there is at least a *nack_threshold_* difference between a lost packet and the latest one received. However, the number of NACKs is restricted to one in each interval to allow sufficient time for the retransmissions to arrive at the receiver. Further at least one NACK is generated in a given maximum period, to ensure that the sender receives at least one feedback notifying the receipt status of DATA packets so that it may purge its buffers of successful packets. Hence in all these NACKs are sent periodically and are therefore known as *periodic-NACKs*. They also serve to resolve deadlock situations. One example of a deadlock that occurs without *periodic*-NACKs is when the sender has finished sending all new packets and is waiting for feedback to retransmit missing packets, while the receiver has lost a burst of packets towards the end of the file and there is no trigger to generate a NACK. Details of various design aspects of *periodic*-NACKs are presented in the Appendix Section 7.6.

However, *periodic*-NACKs could cause bandwidth underutilization when sent infrequently, or result in too many duplicates when sent too often because of overlapping information in adjacent NACKs. For example, in one high-loss scenario with fluctuating error rates and multiple wireless hops, *periodic*-NACKs sent much sooner than a round-trip time resulted in over 60% overhead due to duplicates. Not only do they cause bandwidth wastage, they also block queues and result in long delays for legitimate packets. On the other hand, when NACKs were sent too sparsely (i.e. the NACK interval is much larger than the actual round trip time), the bandwidth could be underutilized because of slow feedback at the sender informing of missing packets. An approach to alleviate this problem could be to introduce sophisticated round trip time estimation techniques similar to TCP to pace the rate of periodic NACKs generated by the receiver. But TCP performance is itself proof that such techniques could fail in rapidly varying wireless scenarios (various results in later chapters). Instead we introduce a novel error control algorithm that minimizes duplicates by significantly reducing the dependence on round trip time estimation.

3.2.2.2 CLAP-final

This CLAP version overcomes the limitations of *periodic*-NACKs, by significantly minimizing the dependence on round trip time estimation for NACK generation. Here duplicates are minimized by reporting only the differential vector of missing packets since the previous NACK. These NACKs are called *pivot*-NACKs and its frequency is controlled by reporting a packet sequence that is at least *nack_threshold_* in length. The sequence begins with the first unreported loss, and hence does not overlap with the previous NACK.

Such a "delta" reporting scheme is possible in CLAP, because the error control and flow control operations are decoupled. Compare this to TCP's window-based operation, where the first in the set of outstanding packets must be confirmed as successfully received, before proceeding further in the sequence. When this does not happen "on time", TCP's error control (commonly called congestion control) mechanism sets in to recover the segment. Until confirmed as successfully received, error control supersedes flow control, and only a limited number of new segments are sent in this interval (*fast-recovery* algorithm). On the other hand in CLAP, as long as there is information of which packets to send, error control *never* supersedes flow control. The sending rate sustains at the bandwidth reported as available, even when NACKs report failure of some packets. These missing packets are prioritized over new packets and transmitted at the available rate by the flow control algorithm.

It is possible that some *pivot*-NACKs and retransmissions are lost in transit, and there could be "holes" in the file sequence without the sender knowing about it. Hence another type of NACKs called *sweep*-NACKs are triggered towards the end of the file, to report all missing packets from start to the highest sequence number received. These NACKs are also triggered when no packets are received for a long time. To enable the receiver to decide when to begin generating *sweep*-NACKs, the sender conveys the total number of packets in the file in the SYN_ message during connection establishment. After the first *sweep*-NACK, subsequent *sweep*-NACKs are sent periodically until all packets in the file are received successfully. These *sweep*-NACKs however, require round trip time estimation. The interval between *sweep*-NACKs is set to the average RTT, calculated using the "request and response" of all the earlier NACKs. The RTT is estimated by using the *timestamp* and *timestamp_echo* fields in the *pivot*-NACK and the *data* packets sent in response, without maintaining any timers at the receiver.

Milestone-NACKs are optionally sent to aid the sender to purge the send buffer of the completed file portion. Figure 3.3 shows a conceptual representation of the three NACK types used in CLAP-*final*.

CLAP-*final* is evaluated in the multi-hop wireless scenarios in Chapter 5, instead of CLAP-*beta* using *periodic*-NACKs. It achieves large performance gains over CLAP-*beta*. *Pivot*-NACKs resulted in a consistent performance of CLAP, even in high loss scenarios, using over 90% of the bandwidth available (upper-bound UDP performance). On the other hand, CLAP-*beta* performance degraded with increasing loss, demonstrating a performance pattern similar to that of TCP in those scenarios. Low bandwidth utilization was seldom seen with CLAP-*final* because of *sweep*-NACKs that conveyed all the missing packets in almost the entire file. The percentage duplicates dropped to under 2% because of non-overlapping information until when over 90% of the file is transmitted. Non-overlapping information in NACKs reduced unnecessary retransmissions, conserving the available bandwidth for legitimate data packets. The cost of poor RTT estimation also reduced significantly, since the estimates were only needed towards the end of the file for *sweep*-NACKs. Thus CLAP-*final* with *pivot*-NACKs and *sweep*-NACKs proves the gains achieved with the core design objectives of CLAP outlined in section 2.1., namely – decoupled flow control and error control and minimized NACKs.

Implementation:

It is important to note that, despite various types of NACKs, they all use the same reporting format, and hence the sender side processing of NACKs does not change with NACK types and the sender merely retransmits the missing packets. NACKs use the same packet format as the data packets, as depicted in Figure 3.2. The *sequence number* field represents the start of the sequence reported in the NACK. The packet sequence is itself represented by a bitmap in the *payload* field, and its length is specified in the *payload length* field. The bitmap representation is as follows: If *seq_* is the sequence number, the i^{th} bit in the bitmap represents the status of packet (*seq_*+ *i*). A 0-bit indicates receipt, and a 1-bit indicates loss.

Hence the various NACKs differ in their *sequence number* field - it's the first missing packet in

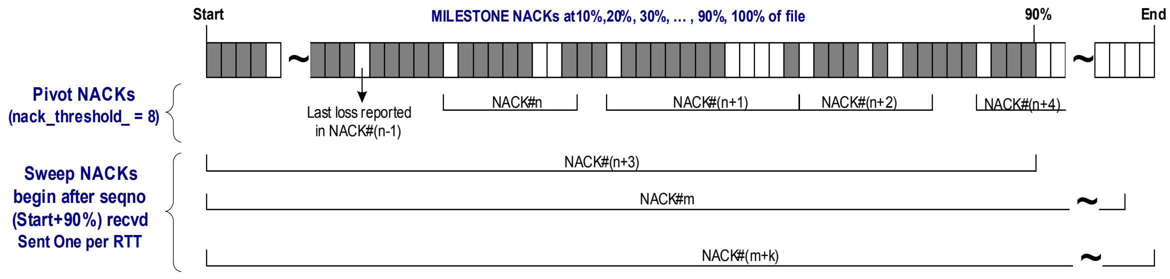

Start
MILESTONE NACKs at10%,20%, 30%, ... , 90%, 100% of file
90%
End
Pivot NACKs (nack_threshold_ = 8)
Last loss reported in NACK#(n-1)
NACK#n
NACK#(n+1)
NACK#(n+2)
NACK#(n+4)
Sweep NACKs begin after seqno (Start+90%) recvd Sent One per RTT
NACK#(n+3)
NACK#m
NACK#(m+k)

Figure 3.3: Aggregate NACK types in CLAP-*final*

the entire file in *periodic*-NACKs, its the first missing packet *after* the previous NACK in *pivot*-NACKs and the first packet in the file in *sweep*-NACKs.

The *timestamp* and *timestamp_echo_* fields serve to estimate round trip time required for *sweep*-NACKs. The receiver sets the *timestamp* field to the NACK generation time. The sender includes the timestamp of the last received NACK in the *timestamp_echo_* field of all DATA packets. When a DATA packet arrives, the CLAP receiver uses the differential of the current time and the *timestamp_echo_* field to estimate the round trip time.

CLAP's error control algorithm makes extensive use of the bitmap data-structure at both the sender and receiver sides. It significantly reduces the complexity of NACK generation and receipt algorithms, enabling efficient implementation.

This simplified error control algorithm is possible because of the separation of sending rate adaptation from CLAP feedback packets. This separation is itself enabled because of the available of cross-layer status information. In the next section we describe the framework developed to enable the network to supply this status information in a systematic manner.

3.3 Architectural considerations for cross-layer design

In this section we present the architecture for a intra-node and inter-node status collection, that may be used to benefit all layers in the network stack. Using this framework, intermediate nodes

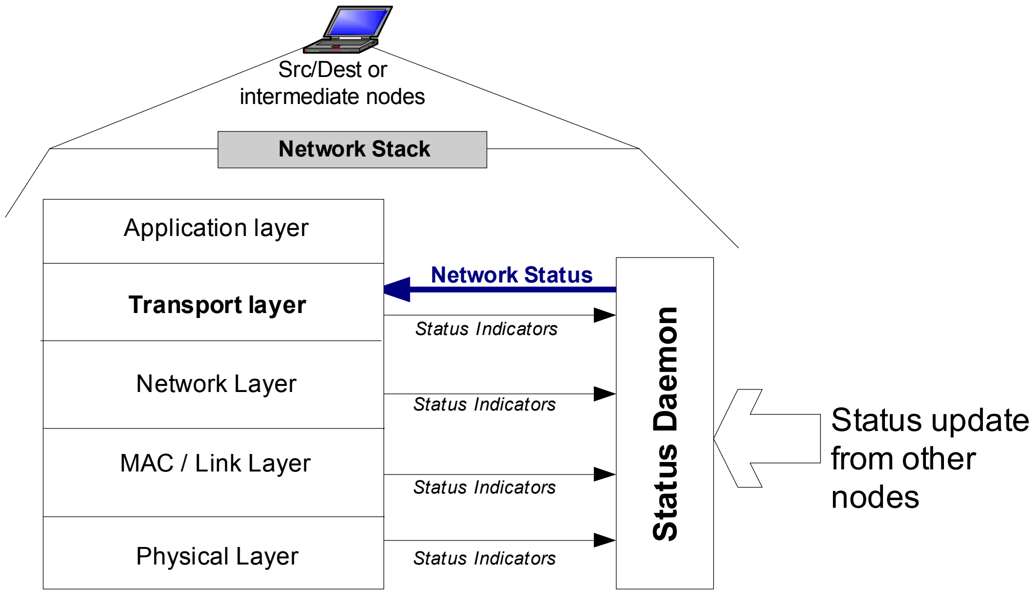


Figure 3.4: Conceptual representation of new network stack with an additional status plane

may regularly supply status information, and transport/routing protocols in edge nodes no longer need to “guess” the network status using their own feedback messages.

When the current Internet architecture came into existence in the 1970s, network entities were limited in their capability. They had low memory and storage, and the link bandwidths were too low to support the additional overhead of control information [11]. Much to the contrary however, today’s routers are equipped with high processing power, large memory and storage. There is abundant link bandwidth in core networks (terabits per second in wired links for example), often underutilized. With networks moving from shared medium technologies towards switched technologies, the core network is ceasing to be the bottleneck, and its capabilities may be put to good use to significantly improve end-to-end performance (as demonstrated with CLAP). It is hence feasible to support a parallel status/control plane even in the wired Internet.

In this section we present the architecture and implementation details of a Cross-Layer software Framework (CLF) to provision status-extraction capability in the network, and hence enable cross-layer design of transport protocols. Figure 2.4 shows a conceptual representation of a parallel status plane aiding end-to-end transport functionality. The status daemon incorporated in each node performs all the intra-node and inter-node functionality required to enable the parallel status plane in the network.

3.3.1 Cross-layer status parameters of benefit to the transport layer

In the context of wireless networks, various status parameters may be extracted to represent network status. Following is a non-exhaustive list of parameters that may possibly aid end-to-end transport.

Physical layer: This layer translates bits in a packet to symbols and then to analog signals for transmission over the physical medium. The parameters here are on a per-node, per-next-hop basis. For example, a node may select the modulation to a certain next-hop node based on the perceived bit error rate to that node. In wireless networks, the “link speed” is the physical channel rate used and is determined by the modulation. Thus the link modulation determines the link capacity. For example in 802.11b, BPSK modulation corresponds to a 1Mbps link speed, while 16-QAM modulation corresponds to 11Mbps. The signal-to-noise ratio (SNR) of the received signal affects the likelihood of accurate reception of a bit. Hence we find that channel modulation and average SNR are two parameters in the physical layer that are of interest at the transport layer, since they determine the link bandwidth the net goodput respectively. These parameters are already available as (a) Received Signal Strength Indicator (RSSI) and (b) Physical channel rate and may be extracted using libraries such as *libmac* [61], available as part of the ORBIT wireless test-bed[47][48].

MAC/Data Link Control layer: This layer concerns with sharing the transmission medium with other nodes and plays a significant role in 802.11 links because of distributed medium access. Each interface in the host can have separate set of parameters, which can further be categorized on a per-next-hop basis. The transport layer may extract information to estimate interference, link bandwidth, link loss rate etc. A possible list of parameters that may be provisioned include:

(a) Packet Error Rate (b) average packet size (c) packet sent rate (bandwidth) (d) packet loss rate (d) packet retransmission rate (e) interface queue availability (f)

Most of the MAC layer parameters are inseparable from physical layer functionality. For example, fluctuating physical channel rates can cause fluctuations in the packet sent rate.

Network layer: The status information here is on a per destination node basis. Some parameters could be tracked at end nodes as well as intermediate nodes in the route. Following is a list of possible parameters of value to the transport layer

(a) Delay jitter (variation of delay between arriving packets irrespective of order of transmission) (b) Route stability (c) Round trip time (d) bottleneck router (d) bottleneck router queue size

Transport layer: A reliable transport protocol may maintain the following status parameters:

(a) Packet loss rate (b) Packet out of order rate (c) Rate of retransmissions (d) average packet size (e) average bandwidth (f) Trigger for new route discovery (g) Average Round-trip-time (h) Variance of Round-trip-time.

3.3.2 Intra-node status collection

Within a node (intra-node), the status daemon interacts with status providing and extracting entities by means of a register-and-pull architecture, as depicted in figure 3.4. Each layer entity participating in status extraction interacts with the status daemon via standard methods defined in the Cross-Layer Framework (CLF). To the best of our knowledge, this is one of the first systematic approaches to intra-node status extraction. Other approaches include one with ICMP messaging between entities [16]-[18].

The register and pull architecture contains a "status plane participant" (SP-Participant) class, of which, all participating layer entities are a sub-class. The status daemon is an object of SP-Daemon class), and there is a single SP-Daemon object running in each node. At the start of a process, an SP-Participant registers with the SP-Daemon specifying details of the status parameters it can supply, using SP-Daemon::register_params() method. Subsequently the SP-Daemon "pulls" parameter values from the SP-Participant, using SP-Participant::get_params() method. Another SP-Participant interested in a status parameter accesses the values through the SP-Daemon::get_status() method. The SP-Daemon maintains a database of all the status parameters in the node. Each parameter is identified by a unique identifier, and the SP-Daemon is itself oblivious to their values or the entities that supply and extract them.

Implementation:

This intra-node Cross-Layer software Framework (CLF) is implemented in the NS2 simulator. The MAC layer (of class MAC802_11) is the status providing entity, and the transport layer (CLAPAgent) is the status extracting entity. A new class called CPMac802_11 is created, that sub-classes both MAC802_11 and SP_Participant classes and hence inherits both functionality. It supplies three status parameters namely., MAC outgoing rate, Interface underflow flag and Interface queue availability. The SP-Daemon extracts the values every 100ms.

The CLAPAgent is only aware of the SP-Daemon and not of the CPMac802_11 object, and extracts the MAC status parameters from SP-Daemon every 100ms. The database in SP-Daemon currently is very simple, where a new value simply replaces the old one.

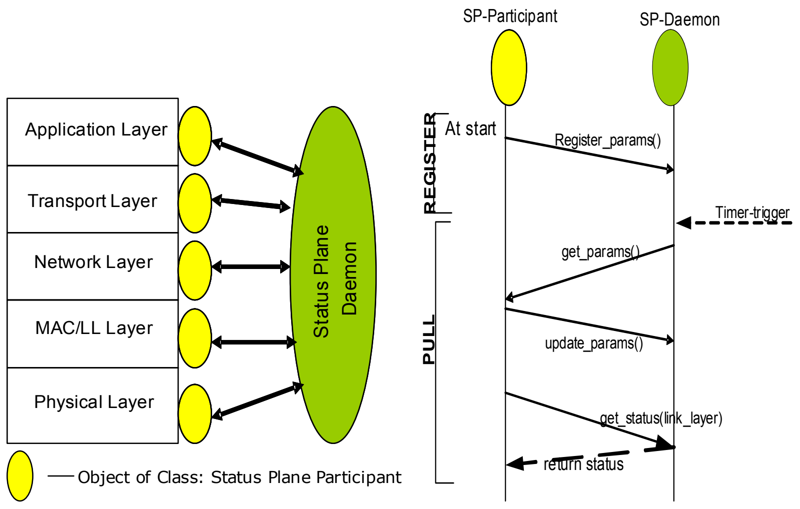


Figure 3.5: Register and Pull Architecture for intra-node updates

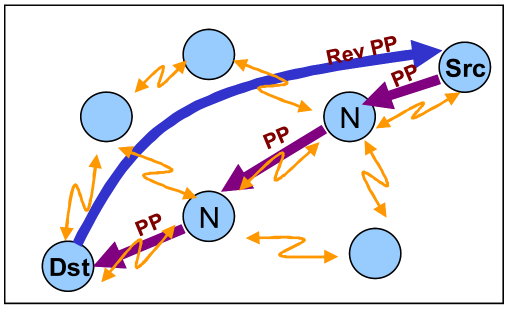


Figure 3.6: Probe-based protocol for inter-node status updates

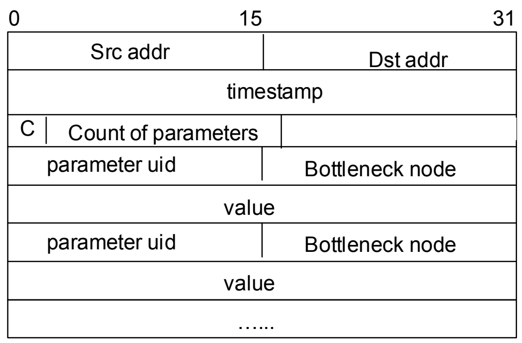


Figure 3.7: CLF *status-probe* packet format

3.3.3 Inter-node status extraction

In a multi-hop scenario, at any instant the net bandwidth available in the route is the bandwidth of the slowest link. The same intra-node MAC parameters apply, but in addition an *inter-node* update protocol is required between status daemons for extracting information about minimum bandwidth available for each flow end-to-end in the network.

At first thought the status plane appears to be an extension of the routing plane, operating just more frequently. However there are some subtle but defining differences. The routing plane collects extreme pieces of information such as node up or down. The network status plane instead assimilates information about operational quality of the nodes: information that determines channel quality such as channel rate, level of MAC contention, bit error rate etc. The time constants in the control plane are much smaller, that is, updates could be a lot more frequent and not necessarily triggered by a problem condition. Given this, the design of the control plane needs to be far more efficient so that messaging overheads are within bounds, and do not curtail operation of the data plane.

Implementation of this update protocol is flexible and can be network-specific. Here we designed a simple probe-based protocol where status daemons in CLAP senders initiate periodic *status probe messages* to collect net bandwidth information in the data path. Intermediate nodes (status daemons in them) compare MAC outgoing rate and Queue Availability parameter values with their own, and replace them if their values are smaller. The MAC underflow is updated if MAC sending rate parameter is updated, and thus tracks the bandwidth increase in the slowest link. The destination node merely tunnels the message back to the original sender. This approach introduces $f*p$ bytes/sec *per flow* in overhead, where f: update frequency in packets/sec and p: packet size in bytes/packet. In our simulation, f = 10 pkts/sec and p=40 bytes, and hence the control overhead is **3.2 kbps per flow**.

The packet format of the *status probe* message is depicted in Figure 3.6. The source and destination addresses identify the CLAP sender and receiver nodes and are required for intermediate nodes to look up the message's next hop. The 'C' (=Complete) flag indicate message status and is set by the intended destination. Parameter fields include a unique parameter id, address of the node that provided the update and the parameter value.

3.4 Summary

In this chapter we described details of the Cross-Layer Aware transport Protocol (CLAP) and the Cross-Layer software Framework (CLF) used by CLAP to extract bandwidth information end-to-end. Primarily they address self-interference and time-varying characteristics of wireless links by decoupling flow control and error control algorithms. CLAP uses aggregate NACKs for error control, many of which report non-overlapping packet sequences. The rate-based flow control algorithm leverages three MAC status parameters to estimate the bandwidth available in each interval, namely, the Outgoing MAC rate, Interface underflow indicator, Interface queue availability. The sending rate is adapted in each interval to the slowest link bandwidth available in the end-to-end route.

Two versions of CLAP are implemented – CLAP-*beta* and CLAP-*final* that differ in the type of NACKs they generate. CLAP-*beta* uses periodic-NACKs which convey the receipt status of a sequence of 128 packets (32 byte bitmap). It requires fairly accurate round-trip time estimation to maximize bandwidth utilization and minimize duplicate retransmissions, particularly in high-noise scenarios. CLAP-*final* instead uses *pivot*-NACKs that report the receipt status of non-overlapping sequences and thus significantly reduces the problem of duplicates. Towards the end of the file, the status of all the entire file is conveyed in *sweep*-NACKs providing sufficient information to the sender to fully use the available bandwidth for retransmissions.

The Cross-Layer software Framework (CLF) is developed to extract general cross-layer information systematically. A single status daemon in a node extracts both intra-node and inter-node status information and thus significantly simplifies code complexity.

In the next two chapters we evaluate CLAP performance in various single-hop and multi-hop scenarios containing 802.11 links with time-varying bandwidth and error characteristics. In these scenarios CLAP extracts MAC status information via the CLF.

Chapter 4
Cross-layer aware transport in wireless LANs

We evaluate the performance of CLAP-*beta* using periodic-NACKs in this chapter. The protocol is compared to TCP throughput and upper-bound UDP goodput, considering severe wireless LAN scenarios affected by both interference and fluctuating SNR. Fluctuating SNR is an inherent wireless problem that causes the packet loss rates to change significantly over time. In wireless LANs, SNR fluctuations are common in indoor as well as outdoor deployments because of changes induced in signal reception. This occurs from various factors including movement of people, opening/closing of doors, transient shadowing due to buildings, environmental conditions such as rain, wind etc. Receiver mobility introduces fading effects that result in additional SNR fluctuations because of changing distance and propagation characteristics of intermediate objects [64].

In this chapter, CLAP performance is evaluated and compared to TCP and upper-bound UDP performance. The UDP goodput obtained with saturating CBR traffic is the upper-bound in performance over time-varying wireless links because of unrestricted flow rate in UDP. Both throughput and goodput are used as performance metrics to evaluate overall CLAP efficiency. CLAP minimizes self-interference and sustains the sending rate even in noisy link scenarios.

4.1 CLAP performance in noise-free scenarios

We described the CLAP protocol along with methods for cross-layer status extraction in Chapter 3. In summary, CLAP decouples flow control and error control algorithms to enable rate adaptation in time-varying scenarios. Its error control uses aggregate negative acknowledgements (NACKs) to minimize interference to data packets in the shared medium. For rate adaptation, it uses a rate-based flow control algorithm that periodically adapts the sending rate, using supplemental MAC status updates instead of its own feedback packets.

In this section we evaluate CLAP performance in a wireless LAN environment with no noise. We will first consider a single flow case, and then a multi-flow scenario.

4.1.1 Methodology

CLAP is developed and validated in the NS2 simulator. Simulation details are the same as those specified in Chapter 2, Section 2.1.3. In this chapter we use CLAP-*beta* for evaluation that uses *periodic*-NACKs. MAC-layer retransmissions were disabled in these experiments.

In this scenario it is sufficient for CLAP to extract status information only from within its own node, since the wireless link is the sole bottleneck. We realize that this is not a general scenario, however the scope of this work is to prove the validity of cross layer information as an effective method to supplement flow control so that error control algorithm may be separated.

Here the CLAP sender matches its sending rate to the net sending rate reported by MAC status parameters every 100ms. TCP-SACK is used for comparison since it uses ACK aggregation and reduces self-interference to some extent.

4.1.2 Single Flow performance

Figure 4.1 and demonstrates CLAP performance compared to TCP-SACK and UDP. With reduced feedback packets, CLAP experiences negligible self-interference. Further since it uses regular updates of MAC layer information to estimate bandwidth, it uses the 802.11 link to full capacity. TCP-SACK is used here for comparison since it aggregates feedback with selective acknowledgements and hence suffers lesser self-interference. Never-the-less TCP-SACK undergoes protocol timeouts several times in the course of the flow due to interference from the ACK packets. Here, CLAP gains over 250% in throughput over TCP-SACK.

Figure 4.1 compares the instantaneous received rates of CLAP with other TCP versions with the minrto_ set to 0 seconds, to allow default RTT estimation to guide timeout intervals. A large minrto_ is not required here because of the small network considered. Since we saw in the TCP dynamics section (Section 2.3 in Chapter 2) that Tahoe performed better than the other TCP versions that used fast-recovery, we compare CLAP performance to this "best" version of TCP that combats self interference. CLAP completes the 1MB file transfer in 1.7 seconds while Tahoe with 1 ACK skip takes 2.4 seconds. The gain with 1-ACK-skip, is 12%, while CLAP gains 95% in throughput over "best-performing" TCP version in self-interference dominated scenarios.

The reasons for CLAP gains here over TCP are evident from the figures. The gain is primarily from the aggregate NACK based approach for error control instead of TCP's positive acknowledgements. In these noise-free scenarios, CLAP-*beta* sends periodic NACKs, at the rate of 10 NACKs per second. Since here of the two contending nodes (AP sending CLAP-NACKs and the wireless client sending CLAP-DATA), one node has very low traffic (AP sending CLAP-

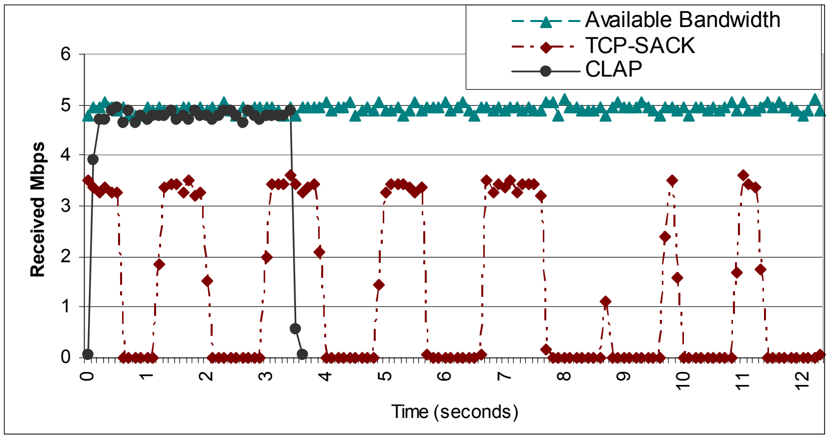


Figure 4.1: Comparison of CLAP, TCP-SACK, UDP for 10MB file transfer

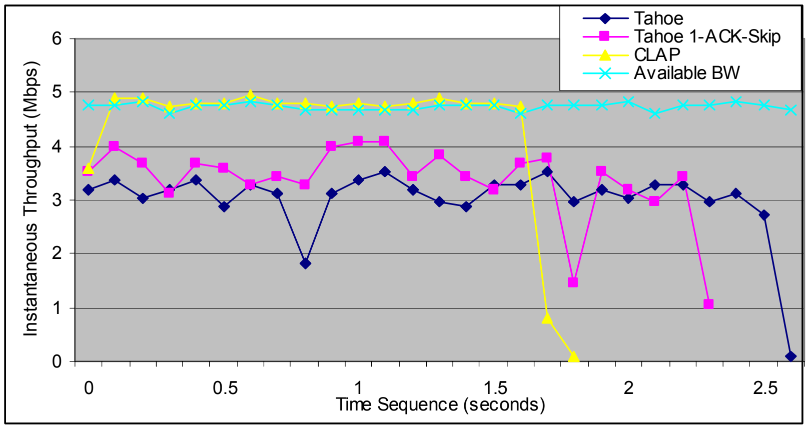


Figure 4.2: Comparing CLAP, TCP-Tahoe and Tahoe-with-1ACKSkip for 1MB file transfer

NACKs), there is low consumption of bandwidth by the CLAP feedback packets. This is evident in the Figures 4.1 and 4.2 with CLAP instantaneous received rate closely matching the available bandwidth plot.

4.1.3 Fairness with multiple flows

Figure 4.3 depicts the instantaneous received rates of 5 CLAP-*beta* flows and Figure 4.4 depicts that of 5 TCP flows operating simultaneously over the wireless LAN environment. These flows begin within 0.1 seconds of each other. The CLAP performance curves show fair bandwidth sharing among all the flows, during most part of the operation. This fairness in bandwidth sharing is a direct effect of using MAC status parameters. CLAP matches its sending rate in a given interval to the value of the *MAC outgoing rate* parameter (described in Section 3.2.1) measured in the previous interval. This value is directly proportional to the net channel time that was available to the node for transmission after sharing the time with other active nodes in the neighborhood. Hence the CLAP sending rate scales in proportion. Further with decoupled flow

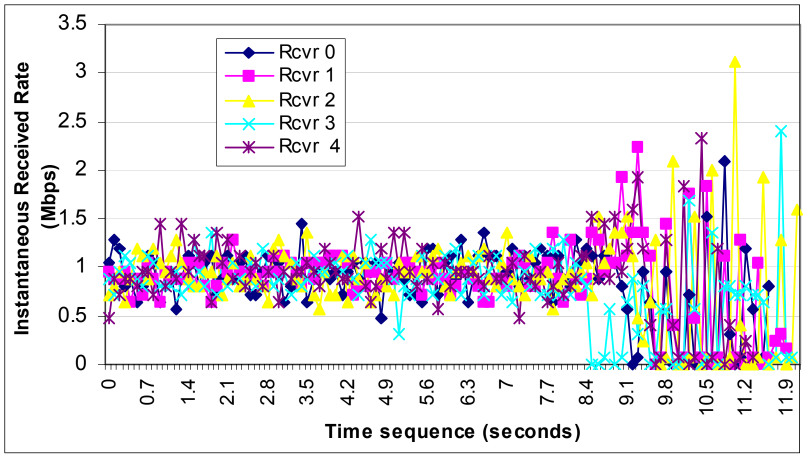


Figure 4.3: 5 CLAP flows over a *noise-free* 802.11 link

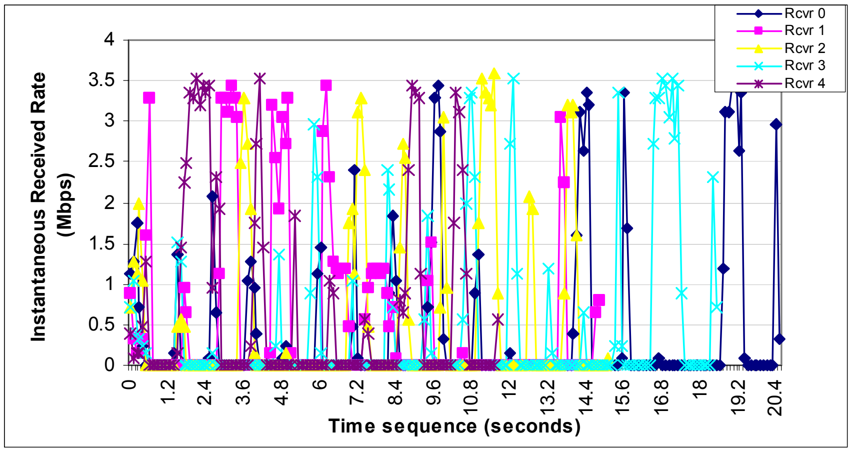


Figure 4.4: 5 TCP-Reno flows over a *noise-free* 802.11 link

control and error control, CLAP continues to operate despite the losses due to MAC collisions. From Equation (1) in Section 2.1.1 the loss rate for 5 saturating flows (plus the AP) due to failed MAC transmissions (two or more nodes select the same slot to send) is 39.03%.

The “spiky” CLAP received rates is due to the lower than required frequency of periodic NACKs, which happens because the periodicity of NACKs is significantly different from the actual round-trip time.

On the other hand, the TCP performance in Figure 4.4 shows significant unfairness among flows with them finishing at widely different times (recvr 0 flow takes about 50% longer time to complete than recvr 1 flow). Only a single TCP flow operates at a given time, i.e. the TCP flows do not co-exist. Xu and Saadawi [23] also observed this phenomenon, and termed it as a “TCP incompatibility problem” in the context of multi-hop wireless networks.

This multi-flow performance in fact is a result of the same reasons that caused timeouts in the self-interference scenarios considered in earlier sections. Again the reason for these losses is interference (in this case cross-interference occurs in addition to self-interference) and combined flow control and error control that scales back the flow rate when these losses occur. A TCP flow operates at peak rate only for a short while before it experiences interference losses and scales back. Since interference causes multiple losses in a congestion window (we proved this in Section 2.3 in chapter 2) Reno is unable to recover without a timeout. Hence interference causes a TCP flow performance to quickly slide down from peak operation. When one flow scales back, another flow gains in throughput, and hence the flows appear as if they are incompatible. The unfairness between flows happens because of the inconsistent losses among flows. The complex interplay between TCP fast-recovery algorithm (that leads to timeouts) and the losses in the MAC introduces significant randomness in operation among flows that appears like incompatibility and unfairness.

Consequently in the given example, five CLAP flows complete transmission around the same time in about 12 seconds, while the last TCP flow completes transmission only after 20.5 seconds. With each of these flows transmitting a 1MB file, the aggregate throughput of 5 simultaneous CLAP flows gains over 80% above the aggregate throughput of 5 TCP flows.

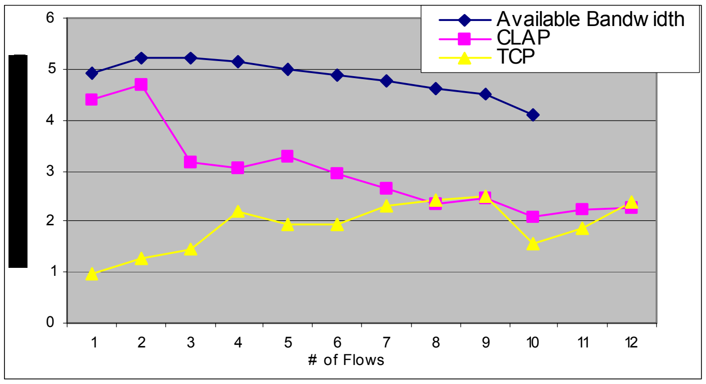


Figure 4.5: Aggregate throughputs of TCP, CLAP and UDP flows

4.1.4 Aggregate throughput with increasing flows

Clearly, CLAP utilizes the bandwidth available in the wireless medium better than TCP for 5 simultaneous flows. More importantly, CLAP appears to use the MAC bandwidth in proportion to its availability. To examine this further, we simulated increasing number of flows and found the aggregate throughput (goodput in case of UDP flows) for each of TCP, UDP and CLAP protocols.

Figure 4.5 shows these aggregate throughputs with increasing flows. First, the available bandwidth plot (that of saturating UDP traffic) peaks at 2 flows and then decreases linearly as the number of flows increase further. Second, the TCP plot shows increasing aggregate throughput but the gains are not consistent with the available bandwidth. The reason for this is the same as explained earlier for the "incompatibility" problem. Third, the CLAP plot shows aggregate throughputs to be in proportion to the bandwidth available, but the performance drops significantly after 3 flows.

Now we will explain the available bandwidth and CLAP plots in detail. The available bandwidth peaks at two flows because of the following reasons: Due to MAC random back-off, there is higher utilization of channel time two flows than with one. But for a higher number of flows, collision losses dominate the gains achieved with improved channel time utilization, and hence the UDP goodput drops for higher flows.

The drastic reduction in CLAP performance is because of the use of periodic-NACKs in the CLAP-*beta* version being evaluated here. As the number of flows increase, the loss rate due to MAC collision also increases. Lots of NACKs are generated at this time. But because of the discrepancy between the NACK periodicity and the actual RTT, there is too much overlapping information in adjacent NACKs that results in too many duplicates. At this time, even though CLAP-*beta* flows may fill the bandwidth available with sufficient packets, the throughput is low because of the large number of duplicates (similar to the problem with the brute force approach described in Section 3.1 in Chapter 3). This problem of overlapping information in adjacent NACKs is overcome in the CLAP-*final* version by using *pivot*-NACKs with non-overlapping information in adjacent NACKs, in the initial parts of file transfer.

In the next section, we evaluate CLAP performance in a time-varying noise scenario.

4.2 CLAP performance in time-varying noise scenarios

TCP evaluation in fluctuating noise scenarios showed less than 2% of bandwidth utilization, because of combined error and flow control algorithms. CLAP decouples these algorithms and uses MAC status information to estimate bandwidth for flow control. In this section we simulate slow and fast-varying noise scenarios and evaluate CLAP-*beta* performance.

4.2.1 Methodology

These scenarios were simulated in the NS2 simulator. MAC-layer retransmissions were disabled in these experiments, to observe transport protocol operation time-varying 802.11 links. The time-varying scenarios are produced by injecting channel noise. Additive Gaussian noise is introduced similar to the method described in Section 3.5.1 with respect to NS2 simulations.

4.2.2 Single flow performance

Figures 4.6 and 4.7 show the instantaneous received rate of CLAP as compared to TCP-Reno and the available bandwidth (saturating UDP flow). CLAP received rate is in proportion to the UDP goodput in the most part, while TCP simply shuts down operation. CLAP is able to operate in this manner because of the decoupling of flow control from error control and the availability of supplemental bandwidth information. CLAP gains of 275% and 317% in these slow and fast varying scenarios. These gains only increase with increasing noise affecting the flows.

The bit rate spikes in the later part of the CLAP flow in both scenarios are due to the following reason: The flow control algorithm operates independent of the error control algorithm as long as there are new packets to send. In high loss scenarios like those considered here, the sender enters a retransmit-only phase where, it waits for returning NACKs to inform it of which packets to send. With periodic NACKs, and the bitmap length maxed at 32 bytes, the information conveyed in each NACK is insufficient to “fill” the available bandwidth and hence the operation in spikes. Further, with NACKs sent too soon in some scenarios, it results in a large number of duplicates that unnecessarily delay the completion of file transfer.

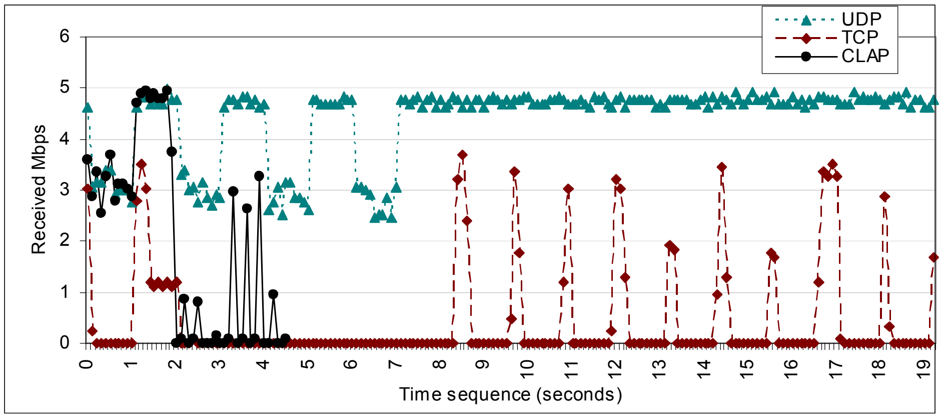


Figure 4.6: Slow-varying error characteristics: CLAP takes 4.9 seconds to transmit a 1MB file while TCP takes over 19 seconds.

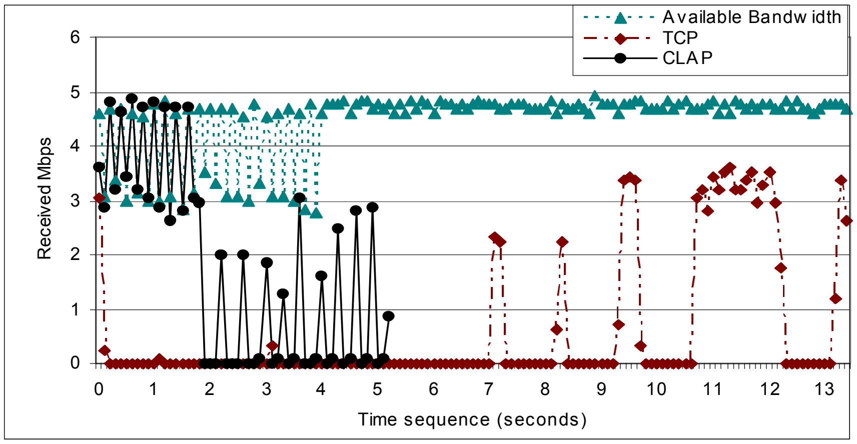


Figure 4.7: Fast-varying error characteristics: CLAP takes 5.4 seconds to reliably transfer a 1MB file, while TCP takes over 13 seconds

One approach to alleviate this problem is to increase the bitmap length in NACKs and use RTT estimation techniques to improve bandwidth utilization in the retransmit-only phase. But doing so, introduces high dependence on accurate round trip time estimation, which in highly fluctuating bandwidth scenarios, can cause a large number of duplicates or result in low bandwidth utilization due to poor estimation, and result in performance degradation similar to TCP.

Instead we overcome this problem in CLAP-*final* with a very simple NACK adaptation where NACKs only report non-overlapping packet sequences until a significant portion of the file is received. This reduces round trip time estimation requirement only towards the end of file transmission. CLAP-*final* is used for evaluation in highly dynamic multi-hop wireless scenarios with both bandwidth and noise fluctuations.

4.2.3 Multiple flows

Figure 4.8 and Figure 4.9 compare the aggregate throughputs of CLAP and TCP in noise-prone scenarios. Figure 4.8 demonstrates fair bandwidth sharing in CLAP despite channel noise. Of course here the channel noise is Gaussian distributed and occurs simultaneously across all wireless nodes. The fairness here is achieved because of the opportunistic flow control operation of CLAP, using MAC layer updates and not scaling back the sending rate despite losses. In this manner the link bandwidth is used to full capacity irrespective of the losses that occur at the wireless receiver node (at least one bit constituting the packet is not decoded correctly due to low SNR).

The overall aggregate throughput in a noise-prone scenario (fast-varying noise) is depicted in figure 4.9. TCP with increasing flows has the worse losses due to MAC contention, and its performance fluctuates quite widely. CLAP aggregate throughput is seen to be significantly higher than that of TCP. The throughput increases with increasing flows because of the overall goodput gains as the number of flows increase.

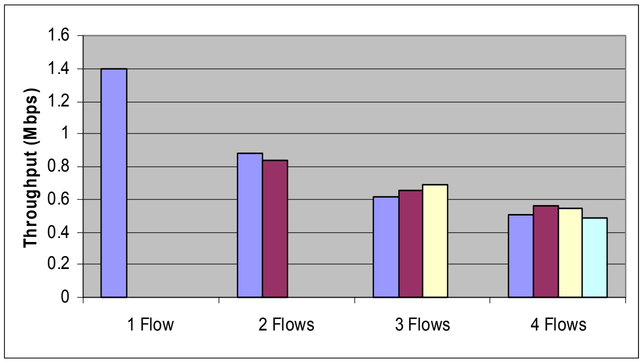


Figure 4.8: Flow fairness in CLAP in fast-varying noise scenario

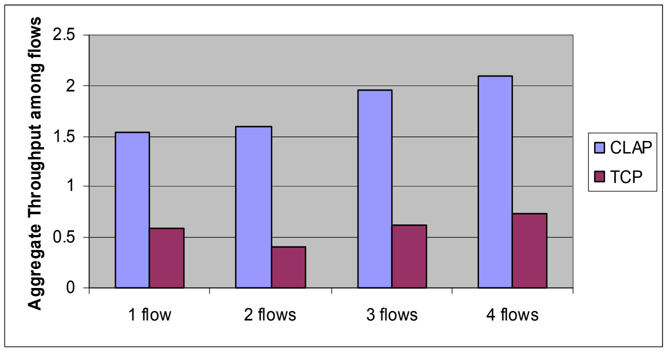


Figure 4.9: Aggregate TCP and CLAP throughputs in a noise-prone scenario.

4.3 Discussion of results and Summary

In this chapter we evaluated an early version of CLAP (CLAP-*beta*) in noise-free as well as highly fluctuating noise conditions in 802.11 wireless LANs. In noise-free scenarios, CLAP achieves over 95% of the upper-bound UDP performance compared to the 68% utilization by TCP-SACK. CLAP gains in performance over TCP because self-interference in CLAP is significantly reduced because of the low rate of feedback packets. To achieve this, CLAP uses aggregate NACKs that convey the loss status in a long packet sequence. Supplemental MAC status information enables the flow control algorithm to match sending rate to the available bandwidth irrespective of the CLAP feedback packets.

This decoupling of flow control and error control algorithms also gains significant advantage in highly fluctuating noise scenarios considered, where CLAP utilizes over 70% of the available bandwidth and completed file transfer compared to TCP which simply shuts down operation.

These results prove the efficacy of the novel algorithms in CLAP for efficient file transfer over 802.11 wireless LANs, and hence combat interference and time-varying link characteristics in these networks.

Chapter 5
Cross-layer transport in multi-hop wireless networks

Multi-hop wireless networks are expected to become increasingly important in the next few years, particularly with the emergence of mesh networks as a viable alternative in both indoor as well as outdoor settings. In outdoor environments, mesh networks are being deployed to serve as "community wireless networks" to extend broadband Internet connectivity to developing communities. Indoors they are being proposed as 2-3 hop wireless backhaul in high-performance wireless LANs. Other than for Internet connectivity, wireless mesh networks are also being considered to form dedicated municipal access networks, such as to connect neighborhood fire stations and police stations. These networks however face significant challenges for data transport, since the data path now contains not one but multiple links with shared medium interference and time-varying characteristics.

Another application of multi-hop wireless networks is with "vehicular networks" where automobiles on a highway send critical information to each other to avert fatal accidents and hence save human lives. Here there is not only end node mobility, the network itself may be mobile since intermediate hops may be constituted by other autos. Various propagation studies have demonstrated that such mobility introduces rapid SNR fluctuations due to fading, shadowing etc [64]. In shared media networks mobility also changes interference regions because of changing proximity to other nodes.

Hence, when compared with 1-hop wireless LAN scenarios considered earlier, it may be expected that reliable data transport is an even greater challenge for the emerging class of multi-hop wireless networks. The transport protocol must now deal with the compound effect of quality variations on all the links in a path, as well as complex external interference and self interference effects resulting from dense node placement.

These considerations motivated us to examine the performance of reliable transport protocols in general multi-hop wireless scenarios. We evaluate transport performance for the fundamental issue of fluctuating link quality that occurs for both static and mobile scenarios of wireless links.

Earlier papers have merely considered transport issues in the secondary dimension, where the same fluctuating link quality causes the network layer to react with route failures and rediscovery.

We set up a 3-hop wireless mesh topology with fluctuating interference and SNR and high loss rates to represent these scenarios. Fluctuating SNR and high loss rates were produced by injecting on-off additive Gaussian noise at receiver nodes, in order to emulate thermal noise that affects the sensitivity of real wireless cards. Varying interference was produced by introducing occasional background flows in the neighborhood. In this environment, the changing end-to-end capacity for a flow was assessed by measuring the instantaneous received rate of saturating UDP traffic. TCP-SACK and CLAP (CLAP-*final* version) were evaluated by comparing their instantaneous received rates with the upper bound of simple, unreliable UDP that represents best-effort since it has neither flow control nor error control restricting the throughput. Details of the CLAP-*final* protocol are in Section 2.4

Several key insights became evident with these evaluations. First, UDP performance with saturating traffic showed that there were significant changes in end-to-end goodput in the presence of other flows in the neighborhood and fluctuating channel loss rates (between 0 and 6%). Hence evaluation of reliable transport protocols in these scenarios explored how well they adapt to rapid end-to-end quality fluctuations.

CLAP evaluation showed that CLAP adapted quickly to fluctuating bandwidth and sustained the sending rate despite losses. There were fewer than 10% duplicates and the status collection protocol constituted less than 0.1% of the available bandwidth (it constituted a load of 3.2 kbps/flow). In all the scenarios considered CLAP performed consistently, utilizing over 90% of the available bandwidth (indicated by best-effort UDP throughput).

On the other hand, TCP-SACK showed significant shortcomings in performance. In channel loss scenarios, it utilized less than 2% of the bandwidth available, and it was slow to adapt when the bandwidth changed suddenly. The performance improved significantly when MAC retries were used to hide wireless losses from the TCP sender, but the throughput fluctuated rapidly due to bursty transmissions (this kind of operation is known to increase queue overflow losses).

Thus CLAP gained significantly over TCP performance and approached ideal performance. The reasons for these gains are due to all of CLAP's design considerations. Use of supplemental MAC status information for flow control achieved quick rate adaptation in fluctuating bandwidth scenarios. Efficient error control design reduced self-interference and duplicates conserving the overall bandwidth to be used by legitimate data packets. Finally decoupling of flow control from error control enabled a sustained flow rate despite link losses.

These significant gains of CLAP over TCP-SACK, and close approach to upper-bound UDP performance demonstrate the efficacy of CLAP's approach to reliable file transfer over wireless networks.

The rest of this chapter is organized as follows. We begin in Section 5.1 with a general overview of multi-hop wireless networks considering end-to-end interference and noise effects. We describe the system considered for evaluation in Section 5.2 and the evaluation methodology in Section 5.3. In Sections 5.4 ad 5.5 we present the results of TCP and CLAP evaluation. Then in Section 5.6, we discuss the results in detail and conclude this chapter.

5.1 Characteristics of multi-hop wireless networks

In this section we analytically show the effects of random noise and interference effects in individual wireless links, on end to end throughput.

5.1.1 Random noise

We assume that additive noise at each wireless receiver causes a packet loss rate of p%. Since this loss is primarily due to fluctuating SNR that arises from thermal noise, mobility etc, we assume that the link noises are uncorrelated.

Figure 5.1 depicts end-to-end random loss with increasing hops. Here Source S sends a saturating UDP data flow containing T packets, to a destination that is (k+1) wireless hops away. If p is the packet loss rate in each wireless link ($0< p< 1$), the fraction of the original number of packets that arrive at the destination is an exponential function given by $T(1-p)^{k+1}$. Hence the number of packets received – *goodput*, decreases exponentially with increasing hops due to losses introduced by additive random noise.

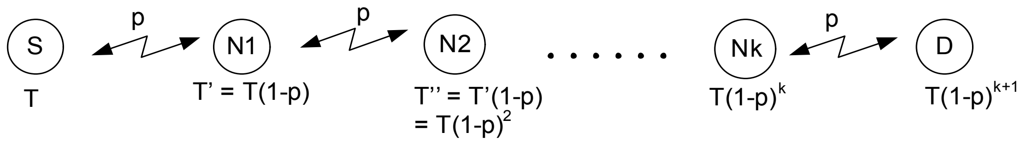


Figure 5.1: UDP packet loss due to uncorrelated random noise

5.1.2 Interference

The problem is a lot more complex with interference than with additive random noise, since interference reduces the sending rate as well as losses due to simultaneous transmissions. That is, it affects performance at both the sender and receiver nodes of wireless link, while random noise only affects at the receiver node of a wireless link.

Figure 5.2 depicts a typical multi-hop scenario where non-adjacent nodes are not in receiving range of each other. The CSMA/CA protocol of 802.11 MAC causes two types of interference ranges that are also marked in the figure – *RxThresh_* is the "Receiver threshold" range, and *CSThresh_* is the "Carrier Sense Threshold" range. These ranges are due to the wireless card sensitivity to received SNR levels, and in drawing them here, we assume that there are no SNR fluctuations. Nodes within *RxThresh_* range can decode each other's signals accurately. Nodes outside the *RxThresh_*, but within *CSThresh_* cannot decode each other's signals accurately, but can still detect the carrier.

When a node is active, all nodes within its *CSThresh_* detect a "busy" signal, and hold back transmitting until the channel is idle. This is due to the CSMA (Carrier Sense Multiple Access) aspect of 802.11 MAC that is introduced to avoid overlapping transmissions among neighborhood nodes. Hence nodes within *CSThresh_* share the channel time and hence their activity causes bandwidth fluctuations to other nodes in the range.

Nodes within *RxThresh_* range corrupt each other's signals if they transmit simultaneously. The Collision Avoidance mechanism in CSMA/CA is introduced to minimize this problem, but signals can still get corrupted if they select the same random backoff slot for transmission. We showed in Chapter 3 that for saturating load at the 802.11 MAC, same slot selection can occur frequently and hence is of concern here. Thus nodes within *RxThresh_* not only cause bandwidth fluctuations, but also result in packet losses.

In other words, the interference ranges determine how nodes in a neighborhood affect each other.

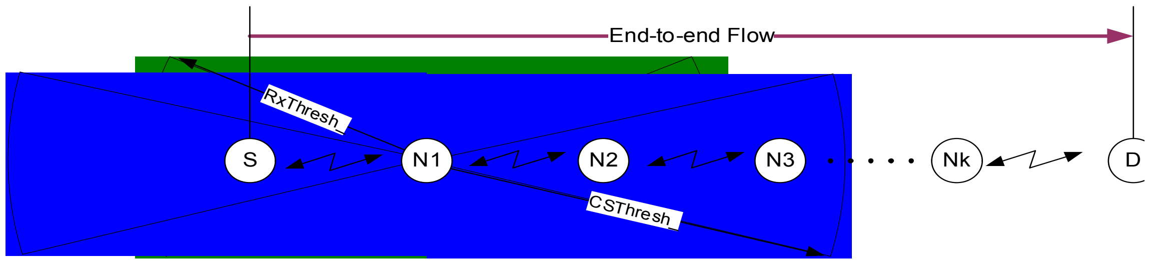


Figure 5.2: Interference ranges in typical multi-hop wireless settings

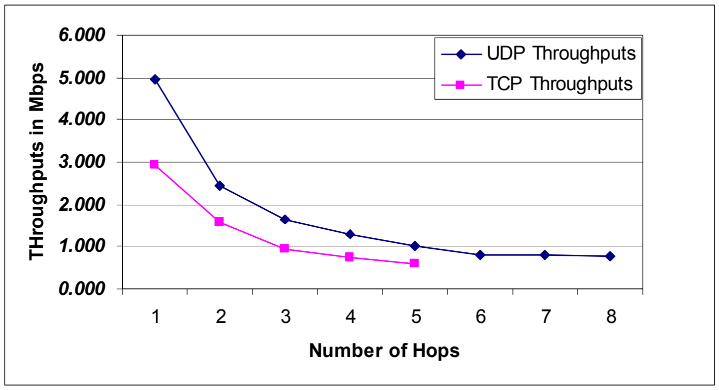


Figure 5.3: UDP and TCP performance in a noise-free scenario with increasing hops

*CSThresh_* only affect wireless sender nodes by causing bandwidth fluctuations, while *RxThresh_* affects both wireless sender *and* receiver nodes by causing both bandwidth fluctuations and MAC collision losses.

5.1.3 Effects of Interference on throughput in a chain topology

Next we describe how these interference ranges affect end-to-end throughput. We consider a multi-hop chain topology, where the wireless links are *noise-free*. Hence any packet losses are only due to interference. MAC retries are enabled to handle these losses at the link layer, so the links appear 100% reliable at the transport layer. In this particular experiment, the link queues are set to be large enough to eliminate queue overflows for TCP traffic.

Figure 5.3 shows throughput vs. increasing hops in the chain topology. Observe that the net throughput drops by half when hops increase from one to two, degrades further as the number of hops increase, but stabilizes when there are more than 6 hops. This is the effect of the interference ranges.

As the data packets traverse across nodes in the chain, all nodes within *CSThresh_* share the channel time among each other because of CSMA. When adjacent nodes (they are within *RxThresh_*) send within the same slot, these could be wasted because the receiver node may not detect the signal (remember 802.11 MAC is non-duplex, and can only do one of send or receive at a given time). i.e. when the Collision Avoidance aspect in 802.11 MAC fails (3% for a pair of nodes with saturating traffic).

For the 1-hop case, TCP throughput is significantly lesser than UDP throughput even though there are no losses, because of the bandwidth consumed by TCP ACK packets, as a result of self-interference in the shared 1-hop wireless medium. With increasing hops, self-interference affects

TCP as well as UDP, since DATA-DATA packets in the forward path now interfere with each other. The difference between TCP and UDP throughputs drops because of lesser percentage bandwidth consumed by TCP ACKs.

In this section we showed the end-to-end effects of random additive noise and interference in a multi-hop wireless network. Clearly interference is a significant issue that causes both bandwidth fluctuations and losses. Because of interference, *neighboring links* affect each other's bandwidth characteristics. This is a significant shift from the wired network characteristics where links not only have fixed bandwidth, they also operate independent of each other. These latter characteristics are what traditional transport protocols like TCP are tuned to cater to. This motivated us to explore and compare the performance of TCP and CLAP in general multi-hop wireless scenarios.

5.2 System Description

The wireless mesh topology depicted in Figure 5.4 is considered to evaluate TCP and CLAP protocols. The nodes are placed equidistant in rows and columns in a grid. The distance between nodes is selected so that in no-noise conditions, adjacent nodes on a diagonal are within *RxThresh_* of each other. In some scenarios we considered another topology, depicted in Figure 5.6, where diagonal nodes are out of *RxThresh_* but within *CSThresh_* of each other. Hence they cannot transmit directly to each other, but can detect each other's carrier during CSMA.

The source and destination operate over a 3-hop primary path. Background flows are selected so that when introduced, interfere with one or more links in the primary path to produce bandwidth fluctuations. To explain this, consider the situation when in the course of the primary flow, BG flow 1 starts. Now when node 9 transmits, all nodes in the primary path are within its CSThresh_ range. So all of them hear a busy signal and cannot transmit, when node 9 is transmitting. This results in transmission delays in all three links in the primary path. This transmission delay results in fewer packets sent in a unit interval, which results in lower bandwidth.

Then there is the issue of interference loss. Suppose node 1 sends in the same backoff slot to its next hop – node 6, node 6 sees a corrupted signal because of the interfering signal from node 9 (node 6 is within RxThresh_ of node 9). The same interference losses occur at node 14 due to node 9.

Thus the background flows affect the primary flow even though they do not use the primary path nodes. This unique aspect of multi-hop wireless networks distinguishes them from traditional wired networks where flows only interfere with each other because of shared queues in common nodes in their routes.

Random channel losses are produced by injecting on-off noise across all nodes in the network. When on the noise has an additive Gaussian characteristic and may cause a signal to be corrupted (incorrectly decoded or undetected) by lowering its SNR (signal to noise ratio). The operating

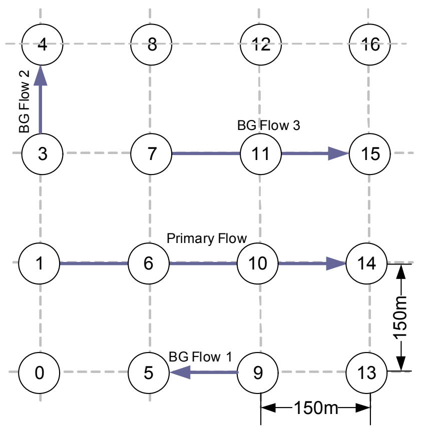


Figure 5.4: Mesh network topology including diagonal transmissions

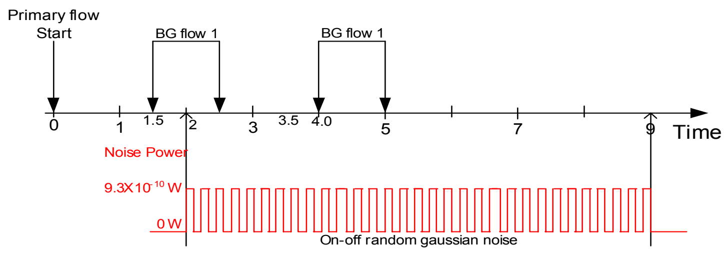


Figure 5.5(a): Scenario with significant noise and some interference

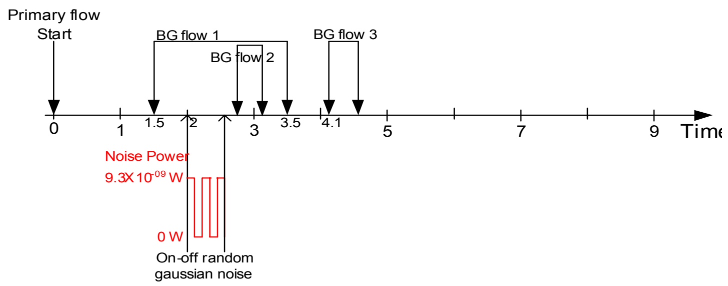


Figure 5.5(b): Scenario with significant interference and noise

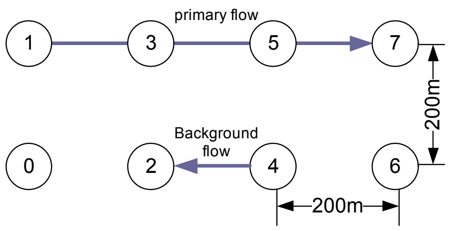


Figure 5.6(a): Mesh network without diagonal transmissions

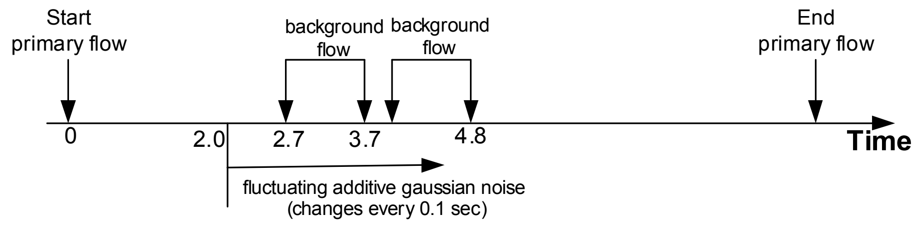


Figure 5.6(b): Time-varying scenario considered

sequences of Figures 5.5(a), 5.5(b) and 5.6(b) are created by introducing the occasional background flows and channel noise.

CLAP and TCP-SACK performance is measured in the primary path, while they transmit a 1MB file from source to destination. The received bit-rate of a saturating UDP flow is representative of the actual bandwidth available during the file transfers, since UDP is best-effort transport without any flow control or error control algorithm.

Next we explain the methodology used to implement this system and how we evaluated the transport protocols over them.

5.3 Simulation Methodology

The results in this chapter are obtained with simulations using an enhanced NS2 model, where we included several corrections to ns-2.1b9a, particularly related to 802.11 wireless modules. Some enhancements are described here.

(a) *Disabled link failure interpretation*: A default NS2 feature is to notify the "routing agent" when despite maximum MAC retries, a MAC-ACK fails to arrive confirming the successful receipt of the packet to the sender. In real implementations, such a notification mechanism would require the wireless interface to have complete knowledge of all host processes using it. Such an operation is seldom implemented in real wireless cards, since not only is it too complex and beyond the scope of 802.11 standard specifications, it also restricts card utilization.

(b) *Physical and MAC layer settings:* Physical parameters such as carrier sense and receiver sense thresholds are set based on Lucent's Orinoco card settings. A short preamble is used where the physical and MAC headers are transmitted at 2Mbps. The interface queue size is set to 100 packets. The 802.11b MAC is used with 11Mbps channel rate. RTS/CTS is disabled. All other physical and MAC parameter settings are described in Appendix section 6.2

MAC retries: MAC retries are not used unless specified. The 802.11 standard specifies optional MAC retries to reduce transient interference losses and hence improve link reliability. We presented more details in Section 1.2. Since in the multi-hop wireless scenario, there is significant interference, we introduce them to study effects of transport protocols. However only some of the results presented here use MAC retries. All the CLAP results presented do not use MAC retries.

(c) *Network layer:* The nodes are pre-configured with static routes. We made this choice since transient link losses triggered route discovery faster in some routing protocols than others, and the study of their effects is out of scope of this paper.

(d) *TCP settings*: NS2 has mature implementations of various higher layer protocols including various TCP implementations. We use TCP-SACK version since it aggregates acknowledgements, and hence experiences lesser self-interference loss. TCP-SACK operates with the following parameters:- minrto_ = 1 second. Initial slow-start threshold *ssthresh_* is set to 64 packets, instead of the default interface queue length. The latter setting invariably caused the sender-side MAC queue to overflow during TCP's initial slow-start.

(e) *CLAP and CLF*: the CLAP-*final* version was implemented for performance evaluation in multi-hop wireless scenarios. CLAP extracted cross-layer updates from CLF once in 100 ms. CLF sent status probes at the same frequency – once every 100ms.

These various simulation parameters were carefully selected to reflect real-world settings. For example, the interface queue length here is set to 100 packets, since in the real world, a TCP socket buffer is of this order by default. We showed in an earlier paper that this parameter significantly affects TCP performance in NS2 simulations, but does not matter so much in real-world because of the kernel operation [4]. Similarly the inbuilt NS2 feedback from MAC to routing agent was disabled after realizing from experimental insight that it was not feasible to implement in real-world systems. Various wireless card parameters relating to receiver sensitivity such as RxThresh, CSThresh etc are set based on Orinoco cards.

By adopting this approach to carefully select simulation parameters, our approach differs significantly from the approach adopted in most related papers that use NS2 simulations. For example, many papers identified poor TCP performance due to route failures in NS2 simulations. Their results are based on the “route failure” notification mechanism inbuilt in NS2, where the MAC notifies the routing agent when all the MAC retries fail. However since such a mechanism is not implemented in real-world systems, the validity of the results themselves becomes questionable. Other than the notification procedure, the MAC retries are themselves not mandatory in 802.11 MAC and so inconsistency arises when MAC retries are disabled.

Hence although we use simulations to evaluate protocols, with the simulation parameters reflecting real-world settings, our results are close to kernel implementation results.

5.4 Effect of MAC retries (UDP performance)

Here we describe the effect of MAC retries on end-to-end bandwidth characteristics. It is important to understand this effect since MAC retries are essential for TCP to sustain its flow rate in high-loss and time-varying bandwidth scenarios

In 1-hop wireless scenarios such as in wireless LANs, MAC retries hide most wireless losses from higher layers. For TCP in particular, this MAC behavior enables operating in saturation, as TCP scales back its flow rate when it encounters packet losses. However in error-prone multi-hop wireless scenarios with interference losses, MAC retries could result in poorer bandwidth utilization.

Figure 5.7 shows UDP goodput in the 3-hop path of Figure 5.6(a). With MAC retries, the packet receive rate at the primary receiver, fluctuates rapidly compared to the no MAC retries case. When there is interference from the background flow, the throughput in some intervals drops to zero.

Large fluctuations occur with MAC retries because of the increasing size of the random backoff

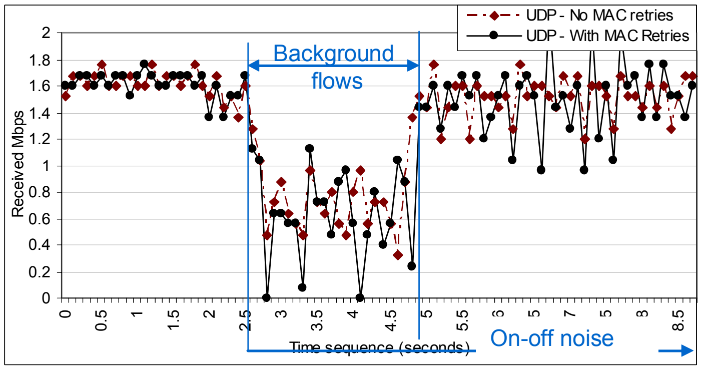


Figure 5.7: Comparison of UDP instantaneous received rate, with and without MAC retries (operating scenario of Figure 5.6(a))

window after each loss. In case of high losses and interference, the high loss likelihood often causes the contention window to reach the maximum (1024 slots), and remains at that stage until the packet is successfully sent. On the other hand, when MAC retries are disabled, the contention window size does not increase. The same base window size (CWmin) is used for random backoff before every packet, even if the previous transmission was unsuccessful.

It is evident here that in the typical time-varying multi-hop scenario, MAC retries cause bursty link performance. It is known from basic queuing theory however, that such changing link performance causes transient queue occupancy that may often lead to queue overflows.

Hence while MAC retries improve link reliability, their use has detrimental effects to overall performance because of the bursty transmissions and the perception of fluctuating bandwidth they present to higher layers.

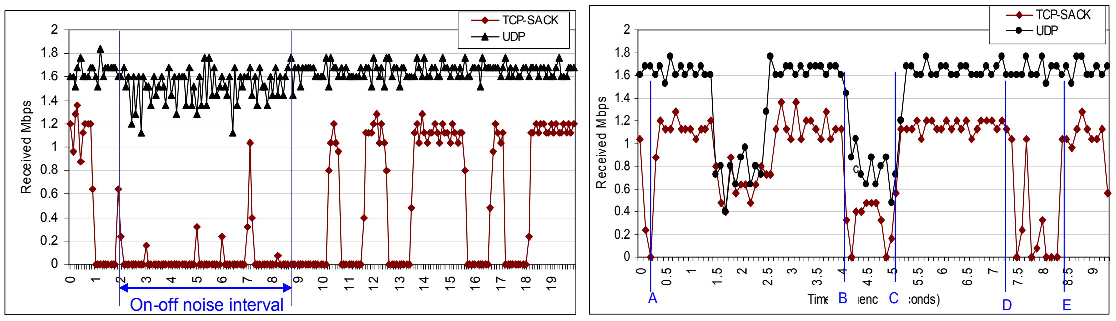
The problem with MAC retries may be expected to worsen with increasing wireless hops because of the compounding effects of interference and end-to-end noise. Hence in subsequent sections we will evaluate protocols with and without MAC retries, and try to not use MAC retries as much as possible.

5.5 TCP performance

Many papers in literature have addressed the shortcomings of TCP performance in wireless multi-hop networks. They mostly examined TCP performance considering random mobility scenarios in the NS2 simulator. Some papers showed that 80% of all TCP losses here were due to route failures and subsequently others proposed performance improvements in these route failure scenarios.

Instead here we address the more fundamental wireless problems of interference and fluctuating link characteristics in multi-hop wireless networks. These not only cause transport issues, but also cause failures in the network layer(disconnections or route failures). These same fundamental problems affect performance even with mobility, just that they occur with higher severity causing greater SNR and even interference fluctuations because of changing proximity to other active nodes.

Next is the question of with and without MAC retries. We showed the detrimental effects of using MAC retries in the previous section. However TCP evaluation in the 1-hop wireless LAN scenario showed that TCP is very sensitive to channel losses. Hence here we will evaluate TCP in both with MAC retries and No MAC retries cases and discuss the tradeoffs in both scenarios.



(a) Channel noise (b) MAC interference

Figure 5.8: TCP performance without MAC retries (operating scenario of Figure 5.5(a))

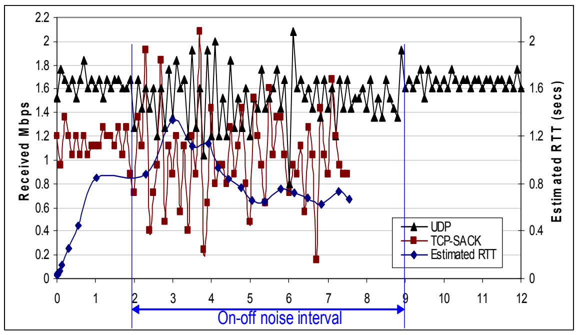
5.5.1 No MAC retries

Here the TCP sender perceives all losses that happen end-to-end including self-interference losses and those due to random channel noise. Figures 5.8(a) and 5.8(b) shows TCP performance in scenarios where MAC retries are not used. These instantaneous received rate plots are obtained for 1MB file transfers. The interference and noise scenario of Figure 5.5(a) is considered, and the occasional flows and channel noise are introduced separately in different experiments.

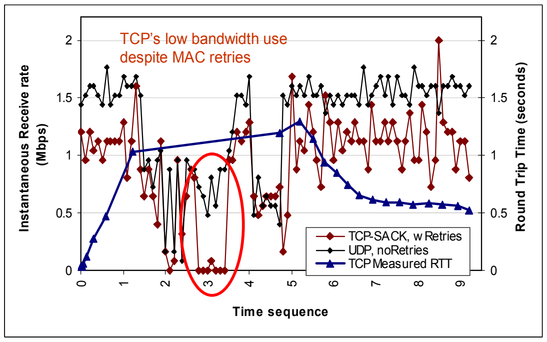
First consider TCP performance when there is fluctuating channel noise. The noise introduced is such that the loss rate fluctuates between 0 – 3% in each wireless link. End-to-end however, the compounding effect of losses over multiple wireless hops causes the loss rate to fluctuate between 6 – 31% (evident from the UDP plot in Figure 5.8(a)).

TCP performance in this scenario depicted in Figure 5.8(a) is similar to the result obtained in wireless 1-hop scenarios (Figure 2.8 and 2.9 in Chapter 2). Here TCP nearly shuts down operation and uses less than 2% of the bandwidth available. This performance is because the sender does not receive acknowledgements regularly and TCP's flow control algorithm depends on regular ACKs to clock its sending rate and a timeout occurs when an ACK does not arrive in the expected time (which is a function of estimated RTT). The TCP sender scales down the congestion window to one segment and tries again by sending just one segment. If there is no response for that segment, the next attempt is made only after an exponential backoff period. This period doubles after each attempt and can reach a maximum of 64 seconds. In Figure 5.8(a) this exponential backoff algorithm causes TCP to delay adaptation to a few seconds after the on-off noise interval has ended.

Next consider TCP performance for bandwidth fluctuations depicted in Figure 4.8(b). There are two occasions here, when the bandwidth available (UDP plot) drops suddenly to about half the



(a) channel noise only
(scenario of Figure 5.5(a))



(b) MAC interference with channel noise
(scenario of Figure 5.5(b))

Figure 5.9: TCP performance with MAC retries

peak value. During the first bandwidth drop, the TCP flow rate adapts proportionately, but during the second drop (between instants B and C), TCP received rate drops to zero multiple times, even while the UDP plot indicates sufficient availability.

In the next section, we try to eliminate these transient wireless losses by introducing MAC retries and evaluate TCP performance in the same scenarios considered here.

5.5.2 With MAC retries

MAC retries compensate for wireless link losses "locally" and thus reduce the number of wireless losses seen by the TCP receiver. Figures 5.8(a) and 5.8(b) show TCP performance with MAC retries.

Figure 5.9(a) shows the effect of MAC retries on TCP performance in the presence of fluctuating noise. TCP's bandwidth utilization improves from 2% (in figure 5.8(a)) to 68%. However MAC retries also introduce large fluctuations in the TCP received rate. These are caused by the combination of transmission latency introduced by MAC retries, and the bursty transmissions caused by delay-variance in TCP. We explained in Section 5.4 that MAC retries cause fluctuating transmission delays in the presence of fluctuating channel noise. For TCP traffic they not only affect onward DATA packets, but also the returning ACK packets. Overall there is a large fluctuation in the pace of returning acknowledgements at the TCP sender. Since TCP sender responds with packet bursts for each incoming ACK, there is a large variation of TCP data packets. Such large fluctuations in throughput often lead to transient queue overflows, particularly with multiple flows.

Figure 5.9(b) depicts performance in a general scenario where both channel noise and interfering background flows affect the flow. The received rates here are for the operating scenario of Figure

5.5(b) where there is a short duration of fluctuating noise and multiple background flows interfere in the course of the flow. The TCP received rate depicts a case where TCP poorly uses the bandwidth available, despite MAC retries. Here the background flows suddenly increase the transmission delays (lesser channel time available for transmission), and MAC retries during channel noise cause sudden bursts of TCP DATA packets. The sudden increase in transmission delay is captured in the "measured round trip time" plot of Figure 5.9(b), where around the 1.5 second instant, the RTT is seen to increase from 0.4 seconds to over 1 second.

A combination of these effects results in multiple losses in some "bottleneck" nodes such as node 6, which is within *RxThresh_* of all the background sources, and is hence affected by MAC interference losses in addition to channel time reduction. This result is a simple example where TCP fails even with MAC retries due to a fluctuating multi-hop wireless environment.

Indeed here TCP packet losses are due to queue overflows. But the overflow itself happens because of TCP's bursty traffic that cause transient filling of the queues. The bursts happen because the TCP offered load is does not match the available link capacity, indicating poor estimation of the bandwidth available by TCP's flow control algorithm, demonstrating the shortcoming of TCP's window based "pipe filling" approach to adapt the sending rate.

It is also clear that the dependence on positive acknowledgements to "clock" the sending rate is the reason for all the traffic bursts. This delay-variance effect is also observed to occur in 3G cellular networks due to channel scheduling [1]. This reiterates the detrimental effects of combined error and flow control.

5.6 CLAP performance

We began our evaluation with CLAP-*beta* that uses *periodic*-NACKs for error control. In the wireless LAN environment, CLAP-*beta* utilized over 60% of the available bandwidth in the noise scenarios, but also had a long tail where the bandwidth was poorly utilized because of the limited amount of information conveyed by the periodic-NACKs. In the multi-hop noise-prone scenarios considered here, the fat tail of periodic NACKs caused significant delay in file transfer. Further because of overlapping information of missing packets, there was a large percent of duplicate retransmissions. In one scenario, they caused a 68% overhead.

CLAP-*final* eliminated this problem in CLAP-*beta* by using *pivot*-NACKs with non-overlapping information of missing packets. *Sweep*-NACKs were used towards the end of the file which reported all the missing packets periodically. This optimization in CLAP's error control algorithm reduced the requirement for round trip time estimates to the end of the file. Since *sweep*-NACKs

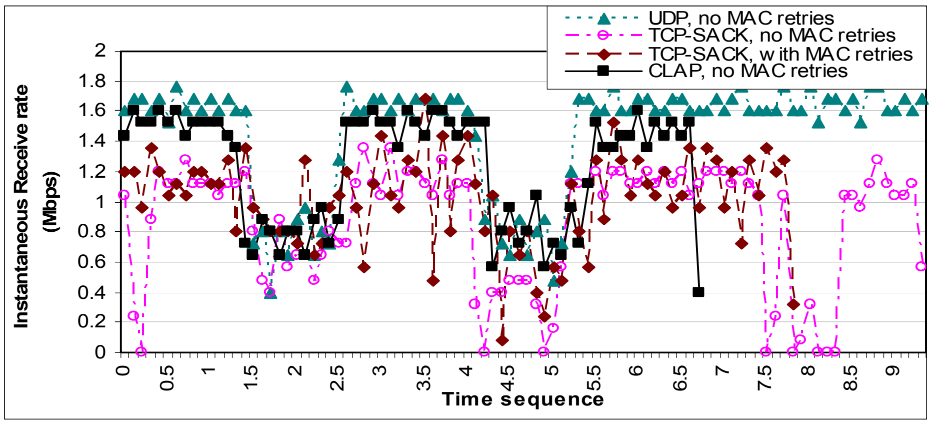


Figure 5.10: CLAP performance in a 3-hop chain topology with time-varying bandwidth (Background flows, no noise in **the scenario of Figure 5.5(a)).**

also conveyed lot of information, the CLAP sender could send the missing packets in bulk using the full bandwidth available, thus eliminating the "fat tail" of CLAP-*beta*. Here we show the results of evaluation of CLAP-*final*. We compare its performance with that of TCP that uses MAC retries for operation.

Figures 5.10 and Figures 5.11 compare bandwidth utilization of CLAP and TCP-SACK (with and without MAC retries) in the wireless scenarios depicted in Figures 5.5 (a) and (b). In all these scenarios, CLAP-*final* was seen to have fewer than 10% duplicate transmissions.

In all the figures, the UDP-no MAC retries curve is representative of the instantaneous bandwidth available in the primary path, since it always saturates traffic.

Figures 5.10 and 5.11 depict performance when bandwidth fluctuations and channel errors of Figure 5.5(a) occur separately.

5.6.1 Adapting to fluctuating bandwidth

In the fluctuating bandwidth plot of Figure 5.10, the instantaneous received rate of CLAP is seen to be very close to the upper bound UDP plot, for the entire duration of the flow. This high utilization of bandwidth for DATA packets and the relatively small fluctuations in received rate demonstrate the consistently low overhead due to CLAP's own feedback packets (the NACKs) and the status probes used by the cross-layer software framework to extract end to end status.

The use of status probes to collect requisite link bandwidth information end-to-end is key to CLAP's efficient rate adaptation. It is evident that despite their low frequency (10 probes/second, each of 40 bytes) these probes are sufficient to capture the bandwidth of the slowest link and convey it on time to the CLAP sender.

CLAP's quick rate adaptation also demonstrates the advantages of using the three specific MAC status parameters. The *MAC outgoing rate* parameter scales immediately in response to the channel access time in any given interval. When the bandwidth suddenly increases (such as at the 5.5 seconds instant in Figure 5.10(a)) the use of the *MAC underflow flag* enables CLAP to adapt to the change quickly. Further by also conveying the *Interface queue availability* value, CLAP increases its rate, just as the amount of queuing resources available, and hence minimizes transient queue overflows.

Thus CLAP's rate adaptation approach is to "fill the link to capacity" while being mindful of queuing resources. This approach differs significantly from TCP's packet-burst approach to "fill the queues", and has the advantage of minimizing queue backlogs. This minimizes the queuing delays for the status probes and enables their quick traversal across the network. This demonstrates the advantages of rate-based flow control rather than a window-based flow control such as used in TCP.

With these methods CLAP gains at least 30% in throughput over TCP-SACK, even with TCP-SACK is enhanced with MAC retries. CLAP also has far lesser fluctuations in throughput than TCP which results in fewer losses due to queue overflows.

5.6.2 Adapting to fluctuating noise

Figure 5.11 shows the performance of CLAP and other transport protocols in the presence of fluctuating noise (of the scenario depicted in Figure 5.5(a)). CLAP is seen to complete the file transfer irrespective of all the losses due to channel noise. The data received rate of CLAP is within 90% of that of upper-bound UDP.

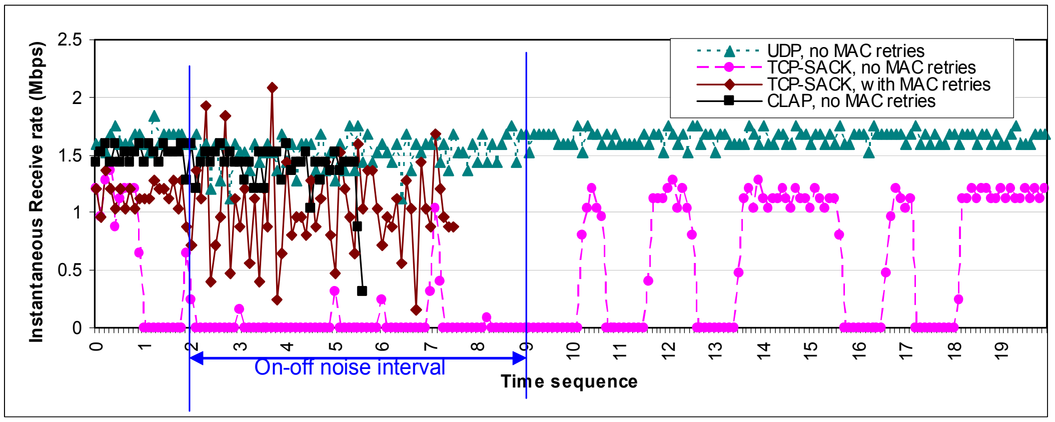


Figure 5.11: CLAP performance in a 3-hop chain topology with time-varying noise
(Channel noise, no BG flows in the scenario of Figure 5.5(a)).

Compare this result that uses the CLAP-*final* version with the result using CLAP-*beta* in figure 4.6. CLAP-*final* here is seen to eliminate the fat-tail at the end of transmission. This performance is a result of the "complete" separation of flow control and error control in CLAP-*final*. In CLAP-*beta*, flow control and error control get "coupled" towards the end of file transfer, since after sending all the new packets, the CLAP sender begins to depend on the periodic-NACKs for information of which missing packets to retransmit. The periodic-NACKs themselves convey less information because of the limited bitmap length (with 32 bytes, report up to sequence of 256 packets). Increasing the bitmap length however causes a large overlap in adjacent NACKs if they are sent sooner than the actual round trip time. This requires accurate round trip time estimation, which is a difficult task with time-varying links, as is evident from TCP performance over time-varying links. With these large overlaps, CLAP-*beta* experienced a large overhead due to duplicate retransmissions.

The figure also demonstrates the gains of CLAP-*final* over TCP-SACK. In the absence of MAC retries, TCP-SACK sent only a small fraction of the file when the noise was on. In this scenario, CLAP gained over 300% over TCP-SACK. The gains only increase with worsening noise conditions. CLAP gains 30% in throughput over TCP-SACK, when the latter uses MAC retries to reduce the error rate. But it is also evident from the plots that CLAP has far fewer fluctuations in the received rate, that gains in performance when there are multiple flows.

5.6.3 Combined bandwidth and error fluctuations

Figures 5.12 and 5.13 show CLAP-*final* performance in the general multi-hop wireless scenario where both bandwidth and error fluctuations occur. These scenarios differ in which of bandwidth or noise fluctuations dominate.

In Figure 5.12, where error fluctuations dominate, TCP throughput fluctuation (with MAC retries) is very large and uses zero bandwidth in several intervals. On the other hand, CLAP performance without MAC retries, is devoid of these rapid fluctuations. Instead it reduces the file transfer delay significantly and gains 35% in throughput over TCP-with-MAC-retries. In doing so, it also uses over 90% of the available bandwidth in the course of its flow.

In Figure 5.13 where bandwidth fluctuations dominate, CLAP outperforms TCP as expected. We explained TCP performance in this scenario in a previous section. While TCP reacts to previous losses with flow rate reduction, CLAP sustains its sending rate because of using supplemental status updates from CLF. However, the CLAP plot shows some instances when the received rate differs significantly from that of upper-bound UDP (for example, at the 5.2 second instant). This is a result of a lost status probes, which are sent best effort sent over UDP. Since these losses can happen frequently (in noise-prone scenarios for example), we optimized CLAP operation to use "local" MAC information when the end-to-end update is outdated (this "lifetime" is hard-coded to two observation intervals).

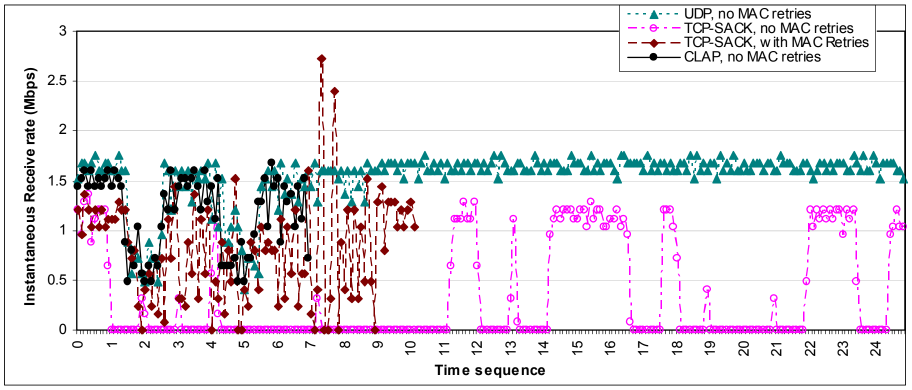


Figure 5.12: Performance in the noise fluctuations dominated scenario of 5.5(a)

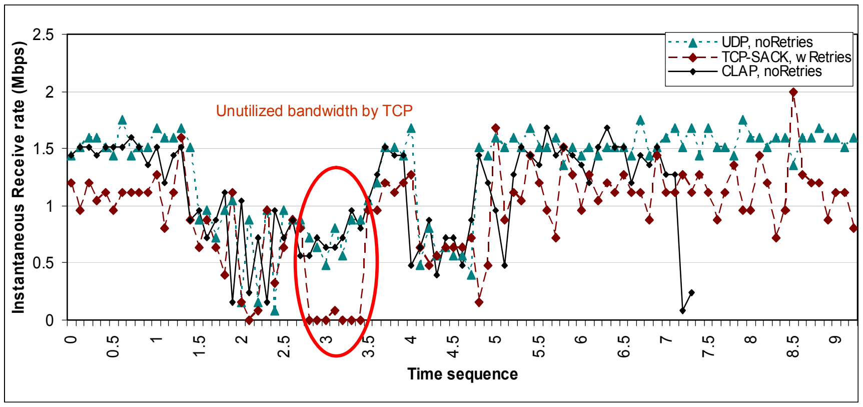


Figure 5.13: Performance in the interference dominating scenario of 5.5(b)

This use of "local" information is justified for current mesh networking environments for the following reasons: The wireless multi-hop is not expected to be longer than 2 – 3 hops. From the mesh topology and the discussion of *RxThresh_* and *CSThresh_* it is evident that interference affects a wide area, covering more than two hops in a chain. The chain is the worst case scenario since with other paths interference has more overlapping effects across nodes. Finally, it is also true that such bandwidth fluctuations due to interference have a much larger impact on performance of mesh networks than with fluctuating noise.

These two figures also lead to some curious observations of TCP performance (with MAC retries). MAC retries are suggested in 802.11 standards to reduce interference-related losses. Likewise here, MAC retries aid TCP performance much better in the interference-dominated rather than the noise-dominated scenario. MAC retries are hence more suitable to eliminate interference losses than losses due to noise, and are not an efficient enhancement for TCP operation in these wireless scenarios.

It is evident from these that CLAP without MAC retries performs better than TCP with MAC retries in terms of both reduced delay for file transfer, and reduced throughput fluctuations in the noise scenarios. CLAP without MAC retries is also seen to adapt more appropriately to link quality fluctuations compared to TCP with/without MAC retries.

From the scenario-related gains that we observed for TCP with MAC retries, it is clear that MAC retries only help in certain wireless conditions. On the other hand, CLAP performance is consistent – using over 90% of the available bandwidth irrespective of the wireless conditions.

5.7 Related Work

TCP performance in multi-hop wireless networks (more commonly known in literature as MANET – Mobile Ad-hoc NETworks) have been addressed in many papers [23][26][27][33][38], where by simulating data transfer over mobile nodes, they found that "route failures" are a predominant cause of poor TCP performance. Most have proposed link layer enhancements to notify TCP sender of an impending route failure. Some have proposed new transport protocols using cross-layer awareness to address the same route failure problem.

Solutions such as TCP-ELFN [27] and TCP-BEAD[38] do not improve transport performance in these fluctuating scenarios since they only modify TCP's congestion control algorithm to freeze the flow control state when a route failure is detected. They do not overcome the slow adaptation issues of TCP over time-varying links and do not address losses due to self-interference. Similarly the ATP protocol that uses cross-layer information from the network layer, still uses positive acknowledgements to adapt sending rate and thus still intertwines flow control and error control algorithms.

Hence all the available solutions for multi-hop wireless networks still use positive acknowledgements to clock the sending rate that increases self-interference, and intertwine error and flow control algorithms that causes poor bandwidth utilization in rapidly time-varying links.

Another observation is regarding the cross-layer approach used in these existing solutions. Many protocols [27][[38] require intermediate nodes to track packets on a *per-flow* basis, and send notification to the TCP sender when they detect impending route failures. This approach is not feasible to implement in multi-hop wireless networks because of the following reasons:-

(a) When there are a large number of flows, these proxies require elaborate state machines and large memory to support them.

(b) The low-cost radio in these emerging networks cannot support the complexity required to implement these proxies

(c) Nodes in the multi-hop network are expected to come up and go down in an ad-hoc manner. They may not have the "motivation" to track the packets of a remote TCP sender and send "altruistic" notifications.

Hence these proxy solutions will not scale with increasing flows in highly dynamic multi-hop wireless environments.

Clearly there is a lack of solutions to address the general problems in multi-hop wireless networks caused because of interference and fluctuating bandwidth and error characteristics in each link. This motivates the evaluation of CLAP in these time-varying scenarios.

5.8 Discussion of results

In this chapter, we evaluated performance of TCP and CLAP in a general mesh network setting, with multiple wireless hops from source to destination. We observed the effects of rapid bandwidth fluctuations due to background flows, and also observed performance for rapid noise fluctuations.

CLAP-*final* very closely approaches UDP performance by utilizing over 90% of the bandwidth available despite time-varying high-loss rate scenarios. The number of duplicates are greatly reduced by using *pivot*-NACKs, and constitute less than 10% of the overhead. All this gain is achieved with simple CLF status probes extracting network status information in a best effort manner. That is, a mere 3.2kbps/flow of control overhead to collect status updates, produced over 90% bandwidth utilization in highly challenging scenarios.

TCP needs MAC retries to sustain operation in time-varying scenarios, particularly when there is channel noise. Our results show that with MAC retries TCP bandwidth utilization improved from 2% to 68%. However MAC retries are intended for interference-dominated scenarios and in noise-dominated scenarios considered, MAC retries were seen to cause large fluctuations in TCP throughput over time, because of the compound effect of MAC latency and bursty operation of TCP due to delay-variance. We demonstrated a scenario where these large TCP packet bursts results in a TCP timeout due to transient queue overflows.

These results demonstrate the efficacy of using a decoupled flow control and error control approach for reliable file transfer over emerging wireless scenarios with interference and time-varying link quality. Decoupling enables CLAP to approach over 90% of the upper-bound UDP performance with a suitable choice of flow control and error control algorithms to quickly adapt to time-varying bandwidth, minimize interference, reduce dependence on round-trip time estimation and minimize duplicate retransmissions.

Chapter 6
Conclusion and Future Work

In this PhD dissertation we have addressed the problem of reliable file transfer over the emerging class of shared media wireless networks. These networks are characterized by multi-user interference and fluctuating SNR, resulting in links with low reliability and highly fluctuating bandwidth and error characteristics. We found that traditional transport protocols such as TCP perform poorly because of oversimplified assumptions about link characteristics. In particular, (a) TCP assumes relatively slow-varying highly reliable links, while wireless links are low-reliability and time-varying, and, (b) TCP assumes duplex operation of links, while many wireless networks in fact operate over a simplex shared medium. A wide gap was observed between TCP performance and that of simple unreliable UDP with a saturating flow.

These considerations motivated the design of *CLAP – a Cross Layer Aware transport Protocol* as a general solution for reliable transfer of files over wireless networks. CLAP decouples flow control and error control algorithms to accommodate time-varying links, and uses a NACK-based feedback scheme optimized to minimize self-interference and duplicate packet transmissions. The flow control algorithm adapts sending rate based on supplemental MAC and PHY cross-layer status information provided by the underlying network. A Cross-Layer Software Framework (CLF) has been implemented to systematically provision intra-node and inter-node status information.

We evaluated the performance of CLAP, TCP and UDP in various time-varying scenarios in both single-hop and multi-hop settings. For single-hop evaluation we selected a wireless LAN environment with rapidly changing error characteristics. The performance of CLAP-*beta* version that used periodic NACKs (dependent on RTT estimation) was compared with that of UDP and TCP. CLAP-*beta* improved significantly over TCP performance, approaching over 60% of upper-bound unreliable UDP performance.

For multi-hop evaluation we considered a 3-hop chain path in a mesh network topology in a time-varying environment. Occasional background flows caused bandwidth fluctuations, and receiver-injected on-off noise caused error fluctuations. The performance of CLAP-*final* version that used NACKs with non-overlapping reports of missing packets (*pivot*-NACKs) was compared with that of UDP and TCP-SACK. As expected, TCP did badly in many scenarios, while CLAP-*final* performed very well achieving over 90% of UDP performance. CLAP-*final* gained over CLAP-*beta* because of a large reduction in duplicates due to non-overlapping information in NACKs.

These experiments in various single-hop and multi-hop scenarios prove the efficacy of using the proposed cross-layer transport approach for file transfer over wireless networks.

Overall, we believe that clean-slate transport protocols optimized for wireless scenarios are practical and feasible in spite of the strong TCP legacy. Changes to transport layer protocols are strongly motivated in emerging ad hoc and mesh network scenarios where TCP performance is problematic. Unlike changes to network layer protocols, transport layer improvements like CLAP can be implemented as plug-in software at servers and mobile devices, while cross-layer control information can be provided as an overlay network service.

This work opens significant new opportunities for future research. First, CLAP, MAC status parameters and CLF need to be implemented in the Linux kernel and tested for performance, since NS2 simulations do not evaluate code complexity that could lead to sub-optimal performance. CLAP currently uses cross-layer status information only for flow control and could also be used for error control on the receiver side, possibly for a hybrid-ARQ scheme that will reduce the number of retransmissions in the interference-limited environment. Status parameters collected by the Cross-Layer Software Framework may be extended to benefit other layers in the network stack. In particular, interactions between CLAP transport and the routing layer in multi-hop ad hoc and mesh networks is an important topic for further study.

Chapter 7
Appendix

Here we describe supplemental work relevant to the thesis. In Section 7.1 we describe insights obtained from video multicast experiments in an indoor wireless LAN. In Section 7.2 we describe ORBIT experimental setup.

Then in Sections 7.3 and 7.4 we describe the TCP and UDP traffic generators we developed and used extensively for real-world experiments and ORBIT test-bed evaluations. And in Section 7.5 we describe the insights gained from various design considerations for CLAP's error control algorithm with aggregate NACKs.

7.1 Video multicast over wireless LANs

This project was carried out to experimentally evaluate the feasibility of 802.11b wireless LANs to sustain video multicast applications [10]. For multicast/broadcast applications, a common choice is the use of RTP/UDP/IP stack. The UDP/IP layers provide a best effort transmission service, with no guarantees of reliability or flow control. Further with retransmissions shut down in the link layer, there is a minimum end-to-end delay but at the expense of reduced link reliability. The RTP layer fragments large video frames to form payload of multiple packets and provides a display timestamp and sequence number for each packet.

Several video compression standards have been developed to cater to video transmission. The most popular ones include H.263 and MPEG-2 that are tailored for different applications. Emerging standards include MPEG4-SP/ASP and MPEG4-AVC (also called H.264 or JVT). It is important to evaluate the efficacy of using efficiently compressed video over a lossy channel while employing forward error correction techniques (with additional overhead), as opposed to using a hierarchical coding technique such as MPEG4-SP/ASP that is less coding efficient. In this work, we focused in general on the carriage of an efficient coding format over wireless LANs.

Experimentation in wireless networks poses distinct challenges. The inherent broadcast nature of wireless links makes them vulnerable to environmental factors such as materials composing the floor, ceiling and furniture, opening and closing of doors and even movement of people. Hence experimental results obtained in one indoor environment are rarely reproduced elsewhere. There is also the problem of repeatability in the same location. Research has also demonstrated that results even vary with the time of the day. The varying traffic characteristics of VBR streams makes it even harder to gain insight into the reasons behind observed behavior. Hence we analyzed multiple MPEG4-AVC video streams to derive some common characteristics of video compression, and then used them as input to UDP traffic generators.

MPEG4-AVC contains three types of frames - IDR, P and B. Their size is very content dependent and is influenced by a number of factors including resolution and other selected encoding parameters. IDR frames contain independent information of a macro-block, P frames contain differential information starting with the IDR frame and B frames include “future” information. We analyzed a representative set of MPEG4-AVC encoded video streams provided by [63]. They were QCIF at 30 frames/second with a global quantization parameter of 15. Some of their characteristics include:

Wide variation in size of different frame types – IDR frames may be up to 20Kbytes while a B frame may be as small as a few tens of bytes.

IDR frames may constitute almost half the compressed bit rate even though they constitute only 1/12th the number of frames.

7.1.1 Experimental setup and results

A suite of UDP streams were generated using general VBR stream characteristics and were

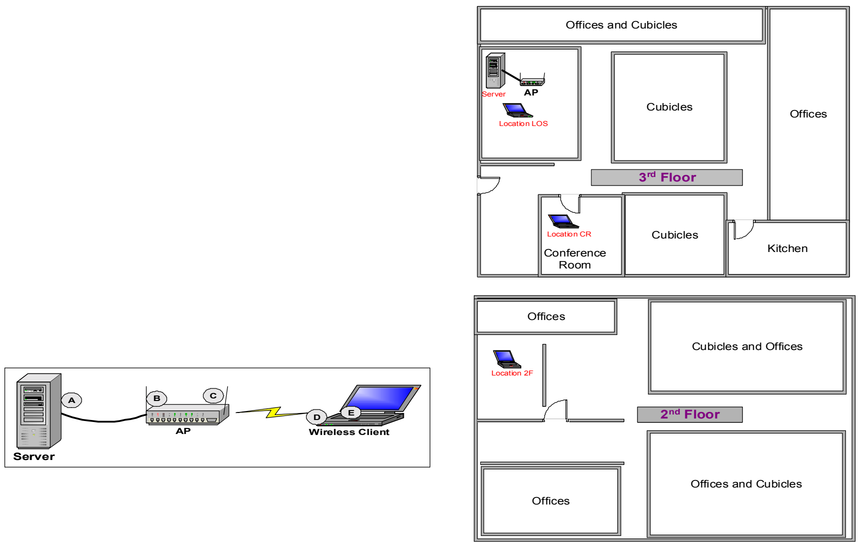
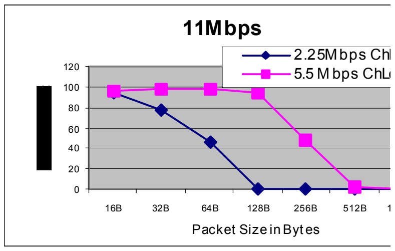
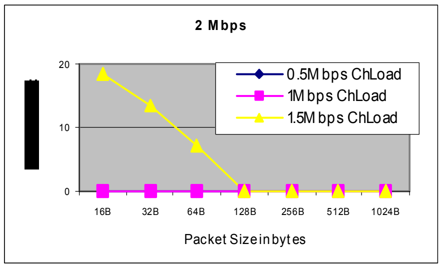


Figure 7.1: Experiment setup and network topology

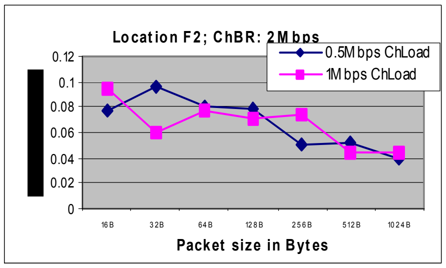
multicast in different wireless scenarios. Each experimental run had one of these parameters



(a) Loss at line-of-sight location at 11 Mbps Phy rate and different traffic loads



(b) Loss at line-of-sight location at 2Mbps Phy rate and different traffic loads



(c) Channel loss at furthest location with non-saturating traffic

Figure 7.2: Measured loss rates in different scenarios for constant UDP multicast streams with 100,000 packets

changing - packet size (16bytes to 1000 bytes), video traffic bit rate (5Mbps, 2.25Mbps, 1Mbps, 0.5Mbps), receiver location (line of sight, different room, different floor) and physical channel rate (2Mbps, 11Mbps). 802.11b MAC and a short Phy preamble (Phy header and Preamble transmitted at 2Mbps) were used.

The experiments were conducted in an office premises spanning multiple floors as depicted in Figure 7.1. The setup comprised of a server PC, access point and a laptop receiver. Each test comprised of UDP multicast stream of 100,000 packets with fixed packet sizes and constant bit rate, streamed from the server to the receiver via the access point. The access point was setup at a fixed location and receiver laptop was moved to different locations and the tests were repeated. At each location, the test was repeated for all traffic loads and all physical channel rates considered. All the experiments were conducted on weekends and holidays when there was nobody else in the building.

Figure 7.2 plots loss rates as a function of packet size in different scenarios. (a) and (b) demonstrate significant losses for small packet sizes even in the line of sight location (hence no

Overhead	Duration (µs)
DIFS	50
Average backoff with Min congestion window	310
Physical layer: short Preamble(144bits/2Mbps) + PLCP header (48bits/2Mbps)	96
MAC header + FCS duration (8*34bytes/11Mbps)	24.73
LLC+IP+UDP headers duration (8*(8+20+8)bytes/11Mbps)	26.18
RTP header duration (8*12bytes/11Mbps)	8.73
Total overhead duration per packet $T_{overhead}$	**515.63**

Table 7.1: 802.11 MAC overhead per packet

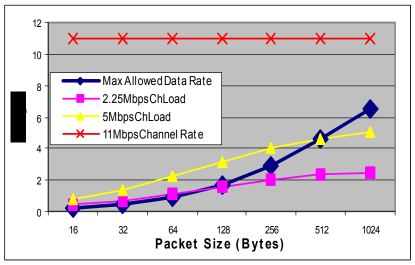


Figure 7.3: Limiting throughputs w.r.t. packet sizes

wireless channel losses). Phy rates of 11Mbps and 2Mbps produced the same effect for different sets of traffic loads. From Figure (c) it is evident that in the specific office environment, 1 floor separation did not produce any significant packet loss (< 0.1%).

Clearly, the most significant result in the paper is with changing packet sizes. Nearly all packets are lost for small size packets even with a moderate channel load. We analyzed possible losses at various points in the network, and found that the packets were being dropped in the access point, in the bridge between Ethernet and 802.11 interfaces. In these scenarios, the 802.11 link is much slower than expected causing access point queues to overflow. The reason stems from the overheads introduced by the MAC contention mechanism in the DCF mode of 802.11 operation. Each packet experiences delay in transmission due to random backoff in the shared medium. This introduces average delay of 310µs per packet. Table 7.1 gives the transmission times for various headers in the packet, and the total transmission delay overhead for each packet is 515µs, *not including the payload*. The same overhead applies irrespective of whether the payload is 16 bytes or 1000 bytes. Hence at the application layer the net link bandwidth available depends on the packet size. Figure 7.3 depicts the maximum throughput possible over an 11Mbps 802.11b link, for different packet sizes. It is a mere 200kbps for 16-byte packets, and just over 5Mbps for 1024 byte packets. The utilization efficiency improves with increasing packet sizes.

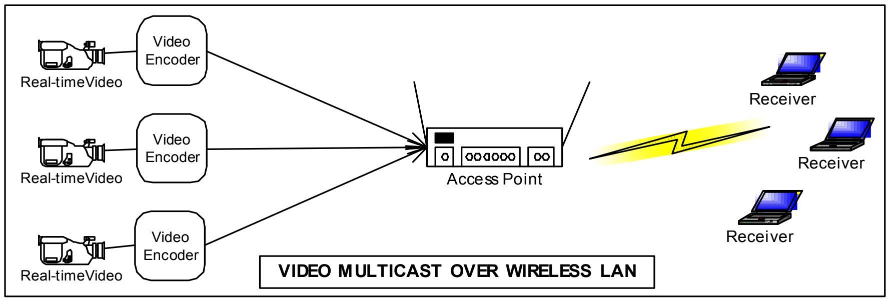


Figure 7.4: Conceptual diagram of multiple video encoders simultaneously multicasting into a wireless LAN environment

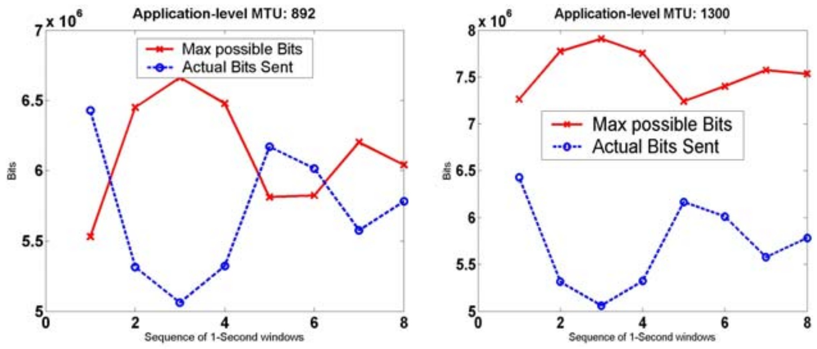


Figure 7.5: "Max possible bits" values change because of link capacity changes with packet sizes; Bigger packets use the 802.11 bandwidth more efficiently than smaller ones because of lower percentage overheads

Hence from the perspective of the application layer, *802.11 link bandwidth fluctuates with packet sizes*. For multiple VBR streams simultaneously multicast into the wireless LAN network, this means that the wireless link bandwidth changes depending on the combination of packet sizes arriving at the access point in each unit interval.

We confirmed this insight by analyzing simultaneous transmission of 3 VBR streams into the wireless LAN via the AP (conceptually depicted in Figure 7.4). These streams were staggered versions of a reference video stream called Tempete, compressed to CIF resolution in MPEG4-AVC format with QP=25. Average bit-rate per stream was 1.9Mbps and no traffic shaping. The total video bits that arrive at the AP in a 1-second interval is independent of any packet sizes. However packet sizes matter in determining the total bits serviced by the 802.11 link in that 1-second interval. Figure 7.5 depicts the difference. When the RTP MTU is 892 bytes, there are several 1-second intervals when the AP cannot send out all the packets that have arrived. On the other hand, when the RTP MTU is 1300 bytes, the 802.11 link operates much faster than the combined incoming rate of the three VBR streams, and can possibly accommodate an additional stream.

This insight may be used to adapt video multicast traffic. Small size packets must be avoided as much as possible. The compression gains achieved by small size B-frames may be offset by the large overhead incurred during transmission over a 802.11 wireless link. On the other hand, large-size packets are more vulnerable to be lost due to channel losses. Hence cross-layer feedback may also be used to indicate channel quality to traffic-shaping modules in video encoders, so that they maximize packet sizes and hence improve 802.11 bandwidth utilization.

7.2 Description of NS2 simulations

All our simulations are conducted in the NS 2.1b9a version, extended with the CMU wireless module that implements the 802.11 MAC protocol. We included several corrections to correct logical and implementation problems. One, we disabled the NS2 feature where failed MAC retries were interpreted as a “link failure”, and the routing agent is notified. In real implementations, such a notification mechanism would require the wireless interface to have complete knowledge of all host processes using it. This complexity curtails wireless network cards from implementing it. Second, we introduced a patch that appropriately reset the DeferTimer was applied to correct a persistent NS2 bug that caused an invalid uid_ in the scheduler [67].

DSDV is used as wireless routing protocol although there is no need for one in the infrastructure mode. Experiments were conducted with various link delays ranging from 2ms to 15ms. A long-preamble was used, where the Physical/MAC headers are modulated at a basic channel rate of 1Mbps. For propagation, a two-ray-ground channel model is used.

Physical parameters such as carrier sense and receiver sense thresholds are set based on Lucent’s Orinoco card settings.

Phy/WirelessPhy object was set with the following parameters - transmit power (Pt_) = 0.2818 Watts; bandwidth_ = 11Mb; dataRate_ = 11Mb; basicRate_ = 1Mb; freq_ = 2.472e9 Hz; CSThresh_ = 5.011872e-12 Watts; RXThresh_= 3.652e-10 Watts; The wired links were set to be of type duplex-link with 10Mbps bandwidth (sufficient for 802.11b where the max data rate is limited to 6Mbps due to network stack overheads), 2ms link delay and dropTail queue.

MAC queue length in wireless nodes (including AP) were set to the default NS value of 50 packets. 802.11 MAC standard suggests RTS/CTS disabled (dot11RTSThreshold = 3000; while

Maximum MAC MTU = 2304). Our experiments with RTS/CTS in this network, showed reduced TCP throughput from additional overhead per packet. Hence RTS/CTS is disabled in all simulations. For experiments with MAC retries, the default values suggested in the 802.11 std.[15] are used (dot11ShortRetryLimit=7; dot11LongRetryLimit=4).

TCP receiver advertised window is set to 1,000,000 packets - large enough to ensure that instantaneous TCP throughput is only paced by the TCP congestion window.

The link bandwidth in any scenario, is measured using a CBR flow with UDP packets between the same source-destination pair. It comprises of 1000 byte UDP packets sent (down the network stack to the wireless interface) at a constant interval of 0.001 seconds. Overheads due to 802.11 Physical and MAC layers limit the transport layer bit rate for 1000 byte packets, to less than 6 Mbps. Hence the CBR rate, which corresponds to 8Mbps is enough to saturate the wireless link.

Maximum duration of simulations was 1000 seconds. Some experiments, particularly with multiple skipped ACKs did not complete file transfers in this duration. In these cases, TCP *goodput* (net data transferred successfully in 1000s) was used as a measure of throughput.

Optional RTS/CTS in the 802.11 MAC was disabled since there were no hidden nodes in our experiments.

The interface queue length is set to 100 packets to eliminate queue overflow situations (unless otherwise specified). All packet sizes (CLAP, TCP and UDP) are set to 1000 bytes. CLAP header size is set to be 40 bytes, to match that of TCP. Multiple flows are staggered from each other, and each flow starts 0.1 seconds after the previous one.

7.3 Experimental procedures on the ORBIT wireless test-bed

The 400-node ORBIT wireless test-bed is a state-of-the-art system built to conduct repeatable and controllable wireless experiments (picture is shown in Figure 7.6 and experiment setup for wireless LAN is shown in Figure 7.7). It enables emulation of noisy wireless links by injecting random gaussian noise into the grid environment via 4 noise antennae in the grid and the noise is generated by means of a signal genarator. 3-hop topologies may be created with noise injection to lower SNR, and carefully selecting affected nodes.

We used ORBIT test-bed to implement TCP ACK skipping in a wireless LAN environment, and to also demonstrate slow TCP adaptation with fluctuating channel noise.

7.3.1 Experimental procedure

We used a 3-layer approach for ORBIT experiments:

1. Select the network topology and experiment parameters (such as noise power) and identify nodes to use in the experiment 2. Set up experiment in ORBIT, run noise and traffic generators and collect results. An average of several trial runs of the experiment is used as the final result. 3. Parse result files, and plot graphs to derive the requisite insights.

This procedure is depicted in the flow charts of Figure 7.8 and 7.9. Node selection requires prior knowledge of the grid topology, since the results obtained are somewhat node-specific. This is a result of wireless card sensitivities and asymmetric noise patterns in the grid. ORBIT allows experimenters to use their preferred operating system/kernel on test bed nodes, allowing complete root privileges. A systematic suite of Perl, Bash and Ruby scripts are used to operate the experiments. Independently developed UDP and TCP traffic generators are key to carry out the experiments among nodes.

Figure 7.6 ORBIT test-bed at WINLAB, Rutgers University located in North Brunswick, NJ

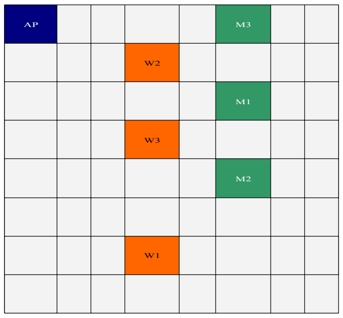


Figure 7.7: Wireless LAN setup on the ORBIT grid; Ws and Ms are wired and wireless nodes communicating via the AP

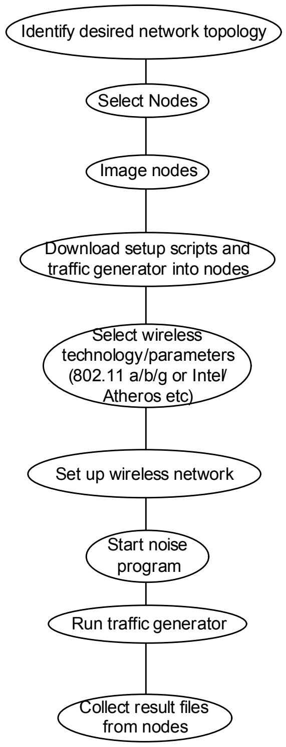

Identify desired network topology
Select Nodes
Image nodes
Download setup scripts and traffic generator into nodes
Select wireless technology/parameters (802.11 a/b/g or Intel/ Atheros etc)
Set up wireless network
Start noise program
Run traffic generator
Collect result files from nodes

Figure 7.8: Experiment procedure in ORBIT

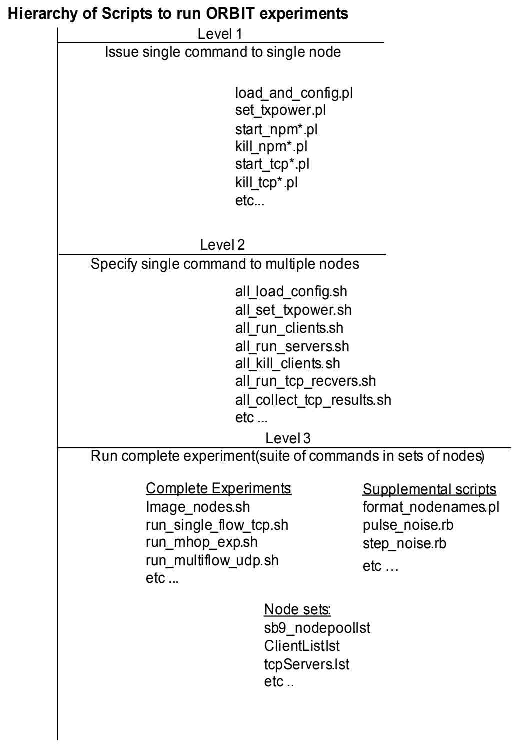


Figure 7.9: Hierarchy of scripts to run ORBIT experiments

```
#!/bin/sh
#intel-adhoc-setup.sh
modprobe ipw2200
ifconfig eth2 up
iwpriv eth2 set_mode 2
iwconfig eth2 mode ad-hoc essid sumathiAdhoc channel 1
ifconfig eth2 10.80.$1 netmask 255.255.0.0

Usage: ./intel-adhoc-setup.sh 1.1
```

```
#!/bin/sh
#ap-setup.sh
modprobe ath_pci
ifconfig ath0 up
iwconfig ath0 mode Master essid sumathi
```

in Ruby scripts and supplied to the *nodehandler* tool in ORBIT. The *nodehandler* in turn invokes requisite web services to inject AWGN noise of the desired power levels in the test bed.

7.3.2 TCP skip-ACKs setup

The main challenges experienced were with configuring certain parameters in different network cards. We explored Atheros 5212, Cisco and Intel cards. Only Cisco cards allowed disabling of MAC retries and physical rate fixation. Similarly, only the Atheros cards operated in “Master” and “Monitor” modes required for Access Point and sniffing respectively. Our experiments required MAC retries to be disabled, and no auto-rate adaptation. Hence we selected nodes with a Cisco card interface as the mobile (M) nodes. A node with an Atheros card interface was selected for AP setup. Due to this combination, MAC retries could only be disabled in the M nodes, but not in the AP. Hence for the traffic type considered here (file upload from M nodes to W nodes), MAC retries were disabled for TCP DATA, but not for TCP ACKs.

Both layer 2 (MAC) and layer 3 (Network layer with forwarding tables) settings were required to establish the wireless infrastructure network in the ORBIT grid. In layer 2, the wireless Infrastructure mode of operation between the AP and M nodes was established using the iwconfig tool in linux. The physical rate was fixed at 11 Mbps and RTS/CTS feature was turned off (no hidden nodes in the experiments so don’t need RTS/CTS). Layer 3 settings were required for wired-cum-wireless routing between the Ws and the Ms via the AP. The forwarding table in each end node was updated to ensure routing via the AP (using the *route* command) and ARP was disabled by pre-updating. IP forwarding was enabled in the AP node to forward packets between W and M nodes. There was no other interfering traffic/noise/channel fading during the experiments. Hence in all the experiments, all packet losses were due to MAC transmission failures. TCP implementation in the kernel (version 2.6.10) was modified to incorporate ACK skipping. It is important to mention that, although the kernel modification was only a few lines (less than 10 lines), one has to be very careful while making them to avoid sub-optimality. For example, a single print statement to log skipped ACKs reduced TCP throughput to a 100th of its previous value. This was due to the expensive per-packet file-write overhead in kernel operation. TCP segment sizes were set to 1000 bytes using TCP socket options.

Kernel programming with ORBIT: Transport protocols are implemented in kernel-space instead of in user-space, to maximize efficiency because of their time-critical nature of operation. The ORBIT test bed infrastructure makes it easy to do kernel modifications. ORBIT’s parallel control infrastructure may be used to track critical boot status and when required restart nodes etc. In case of erroneous implementation that leads to unbootable kernels, the kernel can be repaired via the control infrastructure.

7.4 Network Performance Monitor (NPM) – a pattern-based UDP traffic generator

The UDP traffic generator (called NPM - Network performance Monitor) was designed to suit the needs of various general applications. It generates UDP traffic based on the pattern of required bit rates and packet sizes, reading from a config file. It comprises of two programs – *npmServer* that

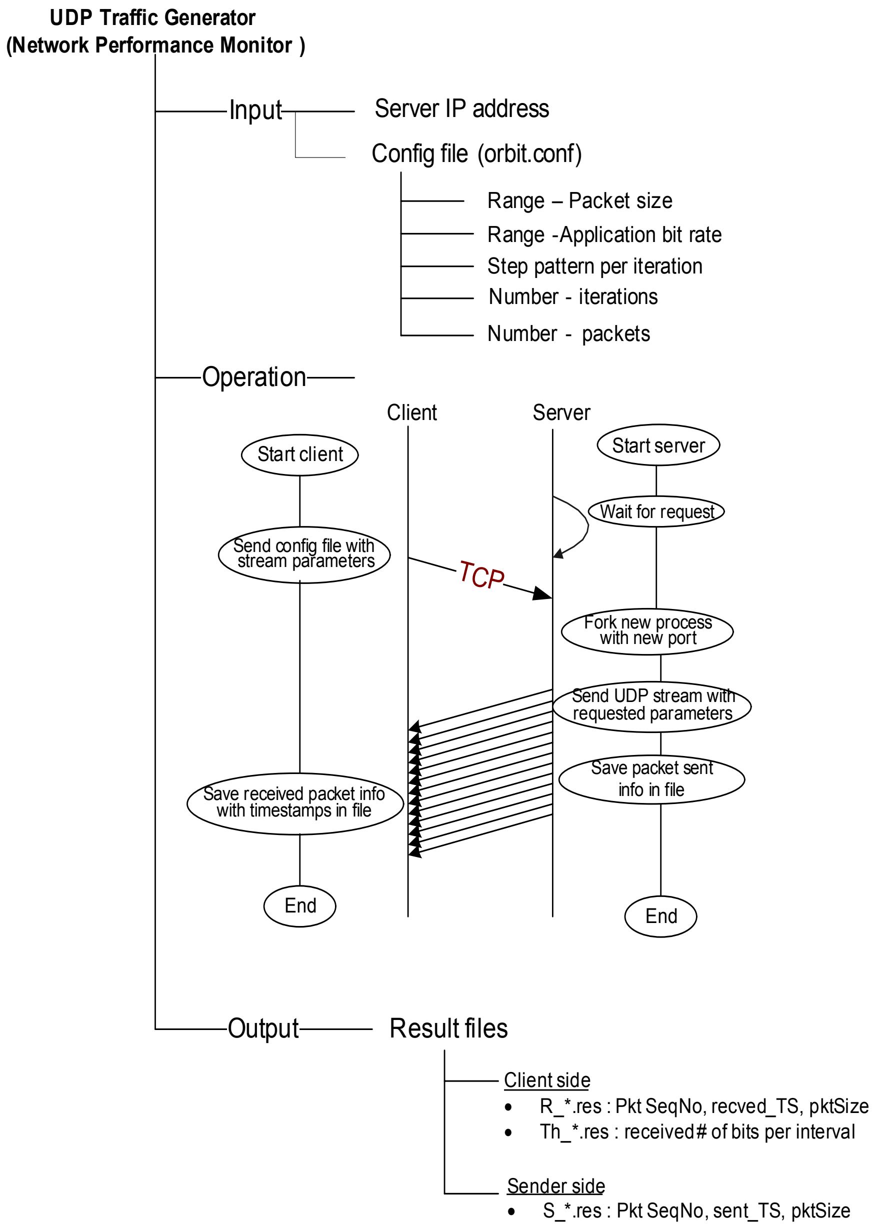

UDP Traffic Generator
(Network Performance Monitor)
Input
Server IP address
Config file (orbit.conf)
Range – Packet size
Range -Application bit rate
Step pattern per iteration
Number - iterations
Number - packets
Operation
Client
Server
Start client
Start server
Wait for request
Send config file with stream parameters
TCP
Fork new process with new port
Send UDP stream with requested parameters
Save packet sent info in file
Save received packet info with timestamps in file
End
End
Output
Result files
Client side
R_*.res : Pkt SeqNo, recved_TS, pktSize
Th_*.res : received# of bits per interval
Sender side
S_*.res : Pkt SeqNo, sent_TS, pktSize

Figure 7.11: NPM - A UDP Traffic Generator

serves requests with UDP packet streams, and *npmClient* that sends the desired traffic pattern to the npmServer and receives data. It logs information of received packets including the packet sequence number, packet size and received timestamp.

NPM can operate in unicast as well as multicast modes, The implementations for these aspects differ merely in the type of IP address specified by the npmClient that receives data. In the multicast case, the program handles subscription to the specified multicast IP address, by initiating IGMP messages in the kernel.

The general procedure is to first start the *npmServer* so that it binds to a specific port (5000) and wait for a request from an *npmClient*. The *npmClient* reads traffic parameters from a configuration file and sends the request to *npmServer* over a TCP connection. It then binds to a specific UDP port (5001) and waits for traffic from the npmServer. *npmServer* forks a new process to serve the traffic to the *npmClient.* Figure 7.10 depicts this procedure.

7.5 *tcptest* – A TCP traffic generator

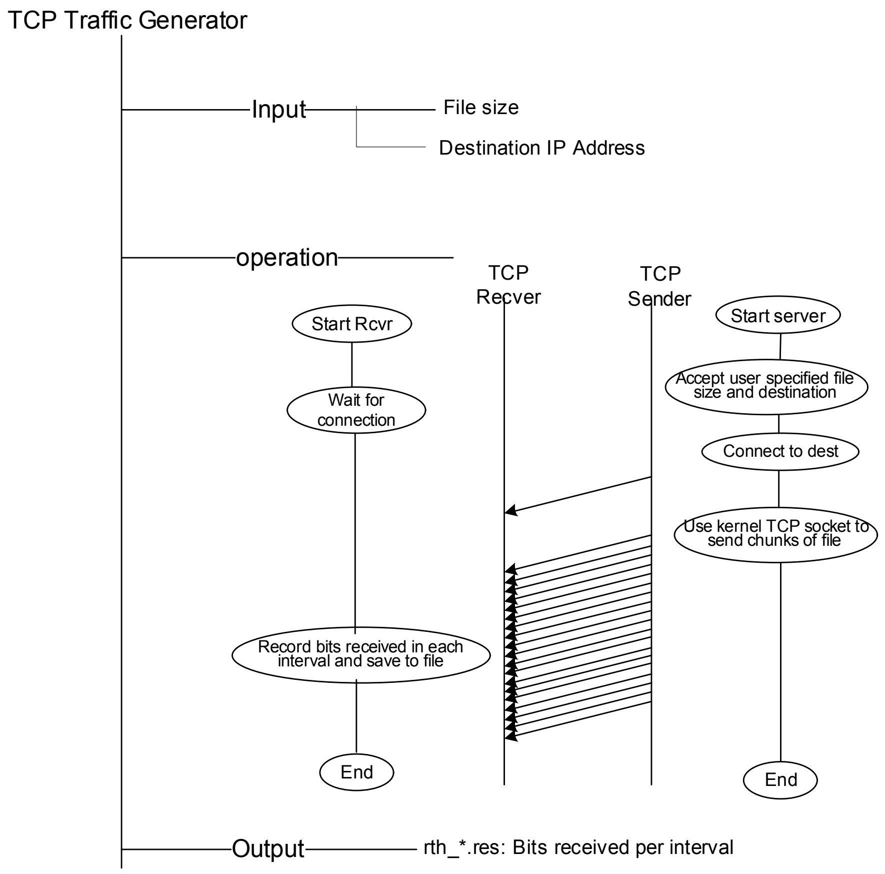

TCP Traffic Generator
Input
File size
Destination IP Address
operation
TCP Recver
TCP Sender
Start Rcvr
Wait for connection
Record bits received in each interval and save to file
End
Start server
Accept user specified file size and destination
Connect to dest
Use kernel TCP socket to send chunks of file
End
Output
rth_*.res: Bits received per interval

Figure 7.12: ***tcptest*** **- A TCP Traffic Generator**

The TCP traffic generator whose functionality is depicted in figure 7.12 is designed deliver a dummy data file of the specified size to the given destination. In essence it is a light-weight FTP application, serving the byte stream in constant packet sizes using a default kernel-supported TCP socket (Linux kernel 2.6.10 uses TCP-BIC by default).

7.6 Various considerations for CLAP error control algorithm

We considered and evaluated various different methods for CLAP's error control algorithm before finalizing the non-overlapping NACK method in the CLAP-*final* version. These various algorithms are tested in *stress-test* noise-prone scenarios, where rapid fluctuations in link errors cause significant packet losses.

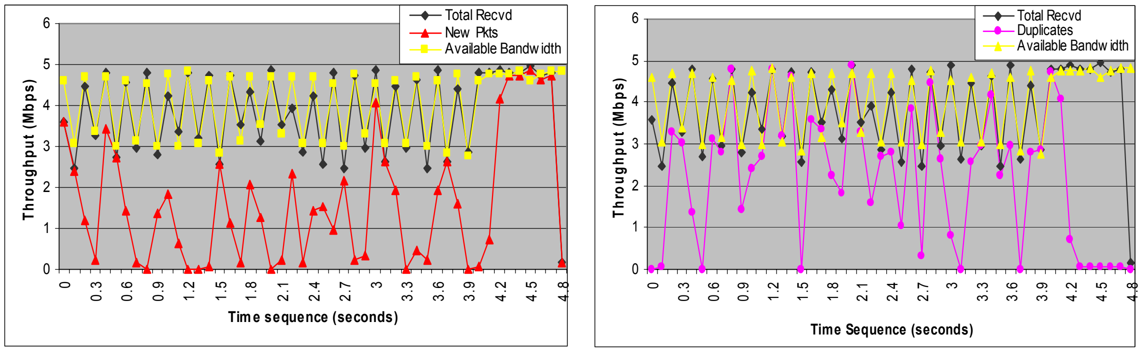
In shared medium operation, a selective reject approach using negative ACKs could achieve high gains over a positive ACKs approach, when the packet loss rate is low, because of lesser bandwidth consumed by feedback packets. However in high-loss scenarios where there are a lot of missing packets, too many NACKs may even worsen the link conditions. Hence the NACKs must deal with two possible situations: (a) When the channel is bad, its possible that there are too many aNACKs generated that would consume precious wireless bandwidth, cause self-interference and hence worsen loss of data packets. (b) In case of a very good channel, very few aNACKs could ensue. But the sender needs feedback from the receiver to ensure that the receiver is alive to periodically flush the send-window in the kernel of successful packets. Hence the number of NACKs must be controlled so that the error control operation is efficient even in high-loss scenarios. In other words, NACKs should be restricted in high-loss scenarios, but a minimum frequency is required to clear the sender side memory of outstanding packets.

We considered two possible approaches that achieve these objectives: (a) Sender-driven NACKs: The receiver sends a NACK only when solicited by the sender. (b) Receiver-driven NACKs: The receiver generates a NACK on its own accord when an incoming data packet indicates losses.

Sender-driven NACKs: Here the receiver sends a NACK only when requested by the sender. The sender maintains a transmit buffer of size *max_buf* to track outstanding packets. When the buffer fills up, the sender stops sending new packets and requests a NACK from the receiver. The receiver responds with a single NACK containing all missing packets up to the highest sequence number received.

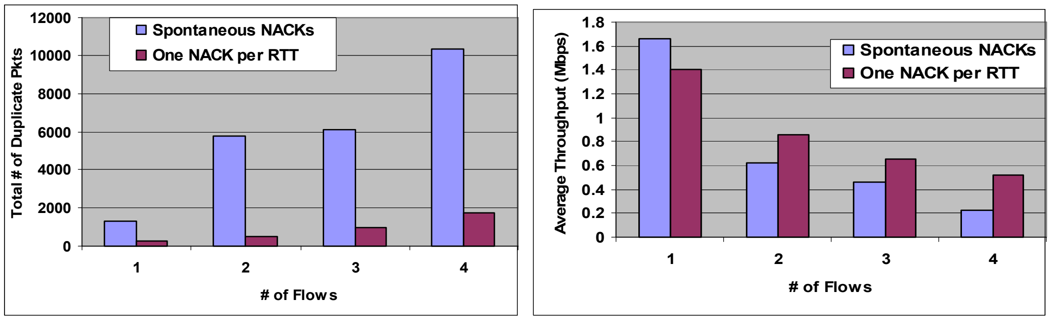
While this approach minimizes interference significantly using only few NACKs, it underutilizes available bandwidth while waiting for a NACK. This is particularly expensive in scenarios with high loss rates and long round trip times. Further the method introduces the requirement of round trip time estimation (to retransmit NACK solicit request) at the sender, which is difficult to do with few returning feedback packets and the algorithm becomes susceptible to rapid round trip time fluctuations that in turn may generate too many NACKs.

Receiver-driven NACKs: The receiver sends an *aggregate* NACK to the data sender whenever an out-of-order packet is received. NACKs report all missing packets up to the highest sequence number received, using a variable length bitmap. With occasional losses and short round trip delays, this approach works well with consecutive NACKs reporting different sets of missing packets. However if there are burst losses where and new data packets arrive before the retransmitted packets, each new data packet generates a NACK containing a list of missing packets that overlap with the previous NACK. One such situation is depicted in Figure 7.13



(a) Received rate of new packets **(b) Received rate of duplicates**

Figure 7.13: CLAP performance with spontaneous NACKs



(a) Comparing duplicates **(b) Comparing average throughput**

Figure 7.14: Comparing performance of spontaneous NACKs and periodic NACKs

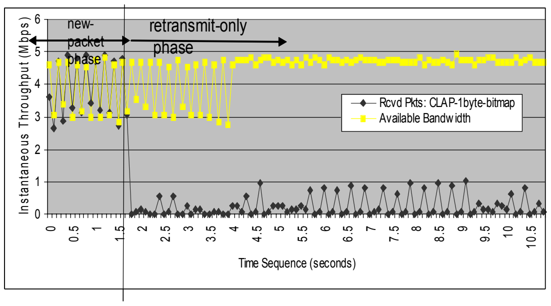


Figure 7.15: CLAP performance with periodic NACKs with a short 4-byte bitmap

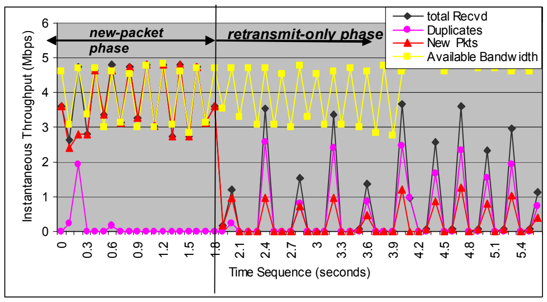


Figure 7.16: CLAP performance with periodic NACKs with a long 32-byte bitmap

(7.13(a) shows the received rate of new packets and 7.13(b) shows the received rate of duplicate packets). Here duplicate retransmissions result in a 134% overhead. It is clear from the instantaneous received rate graphs in the figure, that although the flow control algorithm utilizes the available bandwidth well, the large overhead due to duplicates degrades throughput.

Hence it is essential to reduce the number of duplicate retransmissions, to improve overall throughput achieved during reliable file transfer. The sender retransmits a packet only when requested in a NACK, so we reduced the number of NACKs by using *periodic*-NACKs instead of sending them spontaneously.

7.6.1 Periodic NACKs

Here the frequency of aggregate NACKs is restricted to one per unit interval. The length of the interval is hard-coded to be the median round trip time. This is to allow sufficient time for the sender to respond with retransmissions, before sending the next NACK, to avoid large overlaps in adjacent NACKs. We tested this approach by setting the NACK period to 100ms. The approach resulted in a significant reduction in duplicates as shown in Figure 7.14(a). This reduction produce significant gains in throughput, particularly with increasing flows as shown in figure 7.14(b).

7.6.2 Periodic NACKs with long bitmap length

However periodic NACKs could introduce significant coupling between the flow control and error control algorithms. One high loss rate scenario where this happens is depicted in figures 7.15 and 7.16. In noise-prone scenarios, many NACKs are lost due to channel noise. This causes the sender to mostly sends new packets in the initial stages of file transfer. This is marked as the "new packet phase" in figures 7.15 and 7.16. After this, the sending rate begins to be paced by returning NACKs, and the sending load only constitutes retransmissions. The duration of this "retransmission-only phase" depends on the amount of feedback conveyed in each NACK. In Figure 7.15 each NACK the reported sequence length is 32 packets (NACK bitmap length = 4 bytes), while in Figure 7.16 it is 128 packets (bitmap length = 32 bytes). Increasing the bitmap length from 4 bytes to 32 bytes produces nearly a 100% gain in throughput in the particular scenario. Still the link bandwidth is underutilized in the "retransmission-only phase".

Further increasing the bitmap length however, increases the dependency on accurate round trip time estimation. When there is a mismatch in the periodicity of NACKs and the actual round-trip time, there could either be periods of unutilized bandwidth, or there could be significant overlap in adjacent NACKs. This latter results in a large amount of duplicates. In one multi-hop wireless scenario with fluctuating bandwidth (due to occasional background flows) and high noise, periodic NACKs with 32 byte bitmap, caused a duplicates overhead of over 40%.

These issues with *periodic*-NACKs motivated the new approach with *pivot*-NACKs where adjacent NACKs reported non-overlapping packet sequences. This approach is described in detail in the context of CLAP-*final* version in Chapter 3.

Bibliography

[1] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, Stephen Wolff, "A Brief History of the Internet", http://www.isoc.org/internet/history/brief.shtml, Internet Society

[2] M C Chan and Ramchandran Ramjee, "TCP/IP performance over 3G wireless link with rate and delay variation", ACM MOBICOM 2002

[3] S. Gopal, S. Paul, D. Raychaudhuri," Investigation of the TCP Simultaneous Send problem in 802.11 Wireless Local Area Networks", IEEE ICC 2005, May 16-20, Seoul, South Korea.

[4] S. Gopal, D. Raychaudhuri, "Experimental Evaluation of the TCP Simultaneous Sent problem in 802.11 Wireless Local Area Networks", ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis (EWIND-05), August 22nd 2005, Philadelphia.

[5] Sumathi Gopal, Sanjoy Paul, "TCP Dynamics in 802.11 Wireless Local Area Networks", *To appear in* ICC 2007, to be held in June 2007 in Glasgow, Scotland, UK

[6] Sumathi Gopal, Dipankar Raychaudhuri, "Towards an adaptive transport protocol for efficient media delivery over wireless networks", Dec 2002, WINLAB, Rutgers University

[7] Sumathi Gopal, Sanjoy Paul, Dipankar Raychaudhuri "Leveraging MAC-layer information for single-hop wireless transport in the Cache and Forward Architecture of the Future Internet", To appear in The Second International Workshop on Wireless Personal and Local Area Networks (WILLOPAN) held in conjunction with COMSWARE 2007, Bangalore, INDIA, January 7 - 12, 2007

[8] Sumathi Gopal, Sanjoy Paul, Dipankar Raychaudhuri, "CLAP: A Cross Layer Aware transport Protocol for timevarying wireless links", Technical Report, TR-293, 01/2007, WINLAB, Rutgers University

[9] Sumathi Gopal, Dipankar Raychaudhuri, "Survey of adaptive transport protocols for media delivery over wireless networks", Technical Report, TR-279, 10/2005, Wireless Information Network Lab (WINLAB), Rutgers University

[10] S. Gopal, K. Ramaswamy, C. Wang, "On Video Multicast over Wireless LANs", IEEE Conference on Multimedia and Expo (ICME), Taipei, Taiwan, April 2004

[11] D.D.Clark, M.L. Lambert, L. Zhang, "NETBLT: a High Throughput Transport Protocol", ACM SIGCOMM 1987

[12] J. Liu and S. Singh, ATCP: TCP for Mobile Ad Hoc Networks, IEEE Journal on Selected Areas in Communications, 2001

[13] Sundaresan, V. Anantharaman, H.-Y. Hsieh, and R. Sivakumar. ATP: A reliable transport protocol for ad-hoc networks. In *Proceedings of 4th ACM MobiHoc*, pp. 64–75, 2003.

[14] V. Anantharaman and R. Sivakumar. A microscopic analysis of TCP performance over wireless ad-hoc networks.Presented in 2nd ACM SIGMETRICS (Poster Paper), 2002.

[15] V. G. Cerf and R. E. Kahn, "A protocol for packet network interconnection", IEEE Transactions on Communication Technology., vol. COM-22, V 5, pp. 627-641, May 1974.

[16] G. Wu, Y. Bai, J. Lai, A. Ogielski, "Interactions between TCP and RLP in wireless Internet", Proc. IEEE GLOBECOM'99, Brazil, Dec 1999.

[17] P. Sudame and B.R. Badrinath, On providing support for protocol adaptation in mobile wireless networks, Mobile Networks and Applications 2001, Vol.6, No. 1, pp. 43-55.

[18] K.Chen, S. H. Shan and K. nahrstedt, Cross-Layer design for data accessibility in mobile ad hoc networks, Wireless Personal Communications '02

[19] M. Chinta, A. Helal, ILC-TCP: An Interlayer Collaboration Protocol for TCP Performance Improvement in Mobile and Wireless Environments, WCNC 03

[20] T.Goff, J. Moronski, D.S. Phatak, V. Gupta, Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments, INFOCOM 2000

[21] A Bakre and B. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," in Proc. Of 15th International Conference on Distributed Computing Systems (ICDCS), Vancouver, Canada May 1995

[22] Xu, S.; Saadawi, T., "Revealing and solving the TCP instability problem in 802.11 based multi-hop mobile ad hoc networks", 54th Vehicular Technology Conference, 2001. Volume: 1

[23] Saadawi, T., Xu S., “Revealing TCP incompatibility problem in 802.11-based wireless multi-hop networks” , Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE , 2001, Page(s): 2847 -2851 vol.5

[24] K. Xu, M. Gerla, L. Qi, and Y. Shu, “Enhancing TCP fairness in ad hoc wireless networks using neighborhood red,” in Proc. of ACM MOBICOM, San Diego, CA, USA, Sep. 2003, pp. 16–28.

[25] Z Fu, X Meng, S Lu , "How bad TCP can perform in wireless ad hoc network" , - IEEE ISCC (IEEE Symposium on Computers and Communications), 2002

[26] M. Gerla, K. Tang, R. Bagrodia, “TCP Performance in Wireless Multihop Networks”, IEEE WMCSA, Feb99

[27] G. Holland and N.H. Vaidya, “Impact of Routing and Link Layers on TCP Performance in Mobile Ad-hoc Networks”, Proc. IEEE WCNC, Sept 99

[28] http://bbcr.uwaterloo.ca/~jpan/tcpair/tcpcdma.html

[29] H-Y Hsieh, K-H Kim, Y Zhu, R. Sivakumar "A Receiver-Centric transport protocol for Mobile Hosts with Heterogeneous Wireless Interfaces", Mobicom 2003

[30] P. Sinha, T. Nandagopal, N. Venkitaraman, R. Sivakumar, V. Bharghavan, " WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks", Wireless Networks, Volume 8, 2002

[31] H. Balakrishnan, S. Seshan, E. Amir and R. Katz, "Improving TCP/IP performance over wireless networks", Proceedings of ACM MOBICOM, Nov 1995

[32] F.Wang and Y. Zhang. "Improving TCP performance over mobile ad-hoc networks with out-of-order detection and response", In *Proc. of the ACM MOBIHOC*, 2002.

[33] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks", In *Proceedings of ACM MobiCom’99*, Seattle, Washington, Aug. 1999.

[34] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. "A feedback-based scheme for improving TCP performance in ad hoc wireless networks", *IEEE Personal Communications Magazine*, 8(1):34–39, Feb. 2001.

[35] R. Caceres, L. Iftode, "Improving the performance of Reliable Transport Protocols in Mobile Computing Environments", IEEE JSAC, Special Issue on Mobile Computing Network, 1994

[36] S Mascolo, C Casetti, M Gerla, M Sanadidi, R Wang, " TCP Westwood: Ends-to-End Bandwidth Estimation for Efficient Transport over Wired and Wireless networks", Proceedings of MOBICOM 2001

[37] L Xu, K Harfoush, I Rhee "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks", In Proceedings of IEEE INFOCOM, 2004

[38] Xin Yu, "Improving TCP Performance over Mobile Ad Hoc Networks by Exploiting Cross-Layer Information Awareness" in Proceedings of MOBICOM 2004

[39] Thomson, K., Miller, G.J., Wilder R.,"Wide-area Interent Traffic Patterns and Characteristics", Proc. IEEE Symposium on Computers and Communications, p. Athens, Greece, June 98, p. 74-78

[40] P. Gupta, P.R. Kumar, "The Capacity of wireless networks", IEEE Transactions on Information Theory, 2000

[41] J. Li, C. Blake, DSJ De Couto, H.I Lee, R. Morris, "Capacity of Ad Hoc Wireless networks", In Proceedings of MOBICOM 2001

[42] S. Bansal, R. Gupta, R. Shorey, I. Ali, A. Razdan, A. Mishra, "Energy Efficiency and Throughput for TCP traffic in Multi-Hop Wireless Networks", In Proceedings of INFOCOM 02

[43] W. Richard Stevens, "TCP/IP Illustrated, Volume 1 The Protocols", Addison-Wesley Publication 2001

[44] Larry Peterson and Bruce Davie, "Computer Networks: A Systems Approach", Second edition.. Morgan Kaufmann publishers, 2000.

[45] IEEE 802.11, 1999 Edition (ISO/IEC 8802-11: 1999) Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications

[46] IEEE 802.11b-1999 Supplement to 802.11-1999,Wireless LAN MAC and PHY specifications: Higher speed Physical Layer (PHY) extension in the 2.4 GHz band

[47] ORBIT: Open Access Research Testbed for Next-Generation Wireless Networks www.orbit-lab.org

[48] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu and M. Singh "Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols", to appear in the Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2005)

[49] The Network Simulator - ns-2 ; Website: http://www.isi.edu/nsnam/ns/

[50] The OPNET simulator ; Website: http://www.opnet.com/

[51] Vikas Kawadia and P. R. Kumar, ``A Cautionary Perspective on Cross Layer Design." IEEE Wireless Communication Magazine. pp. 3-11, vol. 12, no. 1, February 2005

[52] K Fall, S Floyd. "Simulation-based Comparisons of Tahoe, Rent, and SACK TCP", ACM Computer Communication Review, 1996

[53] S Floyd, M Handley, J Padhye, J Widmer, "Equation-based congestion control for unicast applications", ACM SIGCOMM Computer Communication Review, 2000

[54] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. "Modeling TCP Throughput: A Simple Model and its Empirical Validation". *SIGCOMM Symposium on Communications Architectures and Protocols*, Aug. 1998.

[55] Kamerman, A.; Aben, G., "Net throughput with IEEE 802.11 wireless LANs", IEEE WCNC. Sept 2000. Volume 2

[56] Y Yi, S Shakkottai , "Hop-By-Hop Congestion Control over a Wireless Multi-hop Network", IEEE INFOCOM, 2004

[57] P. P. Mishra and H. Kanakia, "A hop by hop rate based congestion control scheme," in Proceedings of ACM SIGCOMM, August 1992.

[58] Z. Wu, S. Ganu, I. Seskar, D. Raychaudhuri. " Experimental Investigation of PHY Layer Rate Control and Frequency Selection in 802.11-based Ad-Hoc Networks", ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis (EWIND-05), August 22nd 2005, Philadelphia.

[59] Praveen Gopalakrishnan, "Methods for predicting the throughput characteristics of rate-adaptive Wireless LANs", M.S. Thesis, WINLAB, ECE Dept., Rutgers University

[60] J.C. Lin, S. Paul "RMTP: A Reliable Multicast Transport Protocol", IEEE INFOCOM, 1996

[61] The libmac feature of ORBIT. See webpage http://www.orbit-lab.org/wiki/Documentation/libmac

[62] B.M. Leiner, V.G. Cerf, D.D.Clark, R.E.Kahn, L. Kleinrock, D.C.Lynch, J.Postel,L.G. Roberts, S. Wolff "A brief History of the Internet" http://arxiv.org/html/cs/9901011v1?

[63] Video traces for network performance evaluation http://peach.eas.asu.edu

[64] Theodore S. Rappaport, "Wireless Communication, Principles and Practice", Second Edition, Prentice Hall Inc., 2002

[65] Van Jacobson, "Congestion avoidance and control", ACM SIGCOMM Computer Communication Review, Volume 25, Issue 1, January 1995

[66] E. Ayanoglu, S. Paul, T. F. LaPorta, K. K. Sabnani and R. D. Gitlin, "AIRMAIL: A link-layer protocol for wireless networks, Wireless Networks", Volume 1, Number 1, March 1995

[67] Zhibin Wu, "NS2 Bug Fixes" http://www.winlab.rutgers.edu/~zhibinwu/html/network_simulator_2.html

[68] Y. Yang and R. Kravets, "Contention-Aware Admission Control for Ad Hoc Networks", IEEE Transactions on Mobile Computing, Vol. 4/4, pp. 363-338, 2005

[69] H. Wu, Y. Peng, K. Long, S. Cheng, J. Ma, "Performance of reliable transport protocol over IEEE 802.11 wireless LAN: analysis and enhancement", INFOCOM 2002.

[70] K. Ramachandran, H. Kremo, M. Gruteser, P. Spasojevic and I. Seskar, "Scalability Analysis of Rate Adaptation Techniques in Congested IEEE 802.11 Networks: An ORBIT Testbed Comparative Study". IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM) , Helsinki, Finland

[71] RFC 2581 : "TCP Congestion Control with Fast-Retransmit and Fast-Recovery" http://www.ietf.org/rfc/rfc2581.txt

[72] RFC 2582: "NewReno modification to TCP's Fast Recovery" http://www.ietf.org/rfc/rfc2582.txt , April 1999

[73] RFC 2988: "Computing TCP's Retransmission Timer", http://www.ietf.org/rfc/rfc2988.txt

[74] RFC 3042: "Enhancing TCP's Fast-Recovery Using Limited Transmit", http://www.ietf.org/rfc/rfc3042.txt

[75] A. Desimone, M.C. Chuah and O.C. Yue, "Throughput performance of transport-layer protocols over wireless LANs", In *proc. Of* Globecom'93, December 1993

Curriculum Vita

SUMATHI GOPAL

Education

09/1994 - 10/1998	B.E., Bangalore University, Bangalore, India
01/1999 - 08/2000	M.S., Rutgers, The State University of New Jersey, New Brunswick, NJ
09/2002 – 10/2007	Ph.D, Rutgers, The State University of New Jersey, New Brunswick, NJ

Professional Positions

02/2002 – 07/2002	Intern, Corporate Research, Thomson Inc., Princeton, NJ
09/2000 – 01/2002	Research Associate, C&C Research Labs (CCRL), NEC USA, Princeton NJ
05/2000 – 09/2000	Intern, C&C Research Labs (CCRL), NEC USA, Princeton NJ
01/1998 – 04/1998	Industrial Trainee, National Aerospace Laboratories (NAL), Bangalore, India

Publications

1. Sumathi Gopal, Sanjoy Paul, "TCP Dynamics in 802.11 Wireless Local Area Networks", IEEE Computer and Communications Conference (ICC) 2007, Glasgow, Scotland, UK, June 25-28th 2007.

2. Sumathi Gopal, Sanjoy Paul, Dipankar Raychaudhuri "Leveraging MAC-layer information for single-hop wireless transport in the Cache and Forward Architecture of the Future Internet", The Second International Workshop on Wireless Personal and Local Area Networks (WILLOPAN) held in conjunction with COMSWARE 2007, Bangalore, INDIA, January 12th, 2007

3. Sumathi Gopal, Dipankar Raychaudhuri, "Experimental Evaluation of the TCP Simultaneous-Send problem in 802.11 Wireless Local Area Networks", ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis (E-WIND), Conference held in Philadelphia, USA in August 2005.

4. Sumathi Gopal, Sanjoy Paul, Dipankar Raychaudhuri, "Investigation of the TCP Simultaneous-Send problem in 802.11 Wireless Local Area Networks", Proceedings of the IEEE Computer and Communications Conference (ICC) 2005, Volume 5, page(s):3594 - 3598. Conference held in Seoul, South Korea. 16-20th May 2005.

5. Sumathi Gopal, Kumar Ramaswamy, Charles Wang. "On Video Multicast Over Wireless LANs" , Proceedings of the IEEE Conference on Multimedia and Expo (ICME), Volume 2, Page(s): 1063-1066. Conference held in Taipei, Taiwan, June 2004.

6. Faulkner G., Gopal S., Ittycheriah A., Mammone R., Medl A. & Novak M., "The Aristotle Project", Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications, pages 327-332, Conference held in Montreal, Canada, June 2000.