Towards Optimal Buffer Size in Wi-Fi Networks

Thesis by

Ahmad J. Showail

In Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy

Computer Science Program

Computer, Electrical and Mathematical Sciences and Engineering Division

King Abdullah University of Science and Technology (KAUST)

Thuwal, Kingdom of Saudi Arabia

January, 2016

The thesis of Ahmad J. Showail: Examination Committee

Committee Chairperson: Prof. Basem Shihada, KAUST

Committee Member: Prof. Arif Ghafoor, Purdue University

Committee Member: Prof. Mohamed-Slim Alouini, KAUST

Committee Member: Prof. Panos Kalnis, KAUST

ABSTRACT

Towards Optimal Buffer Size in Wi-Fi Networks

Ahmad J. Showail

Buffer sizing is an important network configuration parameter that impacts the quality of data traffic. Falling memory cost and the fallacy that ‘more is better’ lead to over provisioning network devices with large buffers. Over-buffering or the so called ‘bufferbloat’ phenomenon creates excessive end-to-end delay in today’s networks. On the other hand, under-buffering results in frequent packet loss and subsequent under-utilization of network resources. The buffer sizing problem has been studied extensively for wired networks. However, there is little work addressing the unique challenges of wireless environment. In this dissertation, we discuss buffer sizing challenges in wireless networks, classify the state-of-the-art solutions, and propose two novel buffer sizing schemes. The first scheme targets buffer sizing in wireless multi-hop networks where the radio spectral resource is shared among a set of contending nodes. Hence, it sizes the buffer collectively and distributes it over a set of interfering devices. The second buffer sizing scheme is designed to cope up with recent Wi-Fi enhancements. It adapts the buffer size based on measured link characteristics and network load. Also, it enforces limits on the buffer size to maximize frame aggregation benefits. Both mechanisms are evaluated using simulation as well as testbed implementation over half-duplex and full-duplex wireless networks. Experimental evaluation shows that our proposal reduces latency by an order of magnitude.

ACKNOWLEDGEMENTS

All praise is due to Allah, the Lord of the worlds. The Entirely Merciful, the Especially Merciful. I thank him for all his bounties. Peace and blessing of Allah be upon our beloved Prophet Muhammad and his family and companions.

Having a PhD was a dream. It would never be true with out the support, patience and motivation of many people who are close to my heart. Truly, the secret to my success is my wife Doha, who promised me to be supportive and indeed she was. She showed patience at difficult moments.

The PhD is a long journey. It requires a lot of dedication, motivation and last but least inspiration. My kids, Elyas and Batool, missed me a lot as a father in the past several years. They were, and always are, a source of joy and prosperity in my life. To them I dedicate this dissertation.

My parents nurture in me the passion to be a life long learner. They were always pushing me to accept nothing but the best. Their guidance, support and prayers were invaluable for me. I would like to gift them a warm kiss on their foreheads.

This achievement was not possible with out the guidance of Prof. Basem Shiahada. To me, he is more than supervisor, he is a friend, colleague and a kind mentor. His usual and continuous support was vital to my success.

Many more are in my thank you list, my twin Dr. Mahmood, my friends Hatim and Majid, and my direct mentor Dr. Kamran and many more. These people were stars when I felt that the night is too dark.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

A-MPDU	Aggregate MAC Protocol Data Unit
A-MSDU	Aggregate MAC Service Data Unit
AARF	Adaptive Auto Rate Fallback
AC	Access Category
ACK	Acknowledgement
AIMD	Additive Increase Multiplicative Decrease
ALT	Adaptive Limit Tuning
AODV	Ad-hoc On Demand Distance Vector
AP	Access Point
AQM	Active Queue Management
ARQ	Automatic Repeat Request
BA	Block Acknowledgement
BDP	Bandwidth Delay Product
BER	Bit Error Rate
BQL	Byte Queue Limit
BSD	Berkeley Software Distribution
CBR	Constant Bit Rate
CDF	Cumulative Distribution Function
CoDel	Controlled Delay
CPU	Central Processing Unit
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Clear To Send
DCF	Distributed Coordination Function
DIFS	Distributed Inter-Frame Space
DMA	Direct Memory Access

DNB	Distributed Neighborhood Buffer
DQL	Dynamic Queue Limit
DRWA	Dynamic Receive Window Adjustment
DSL	Digital Subscriber Line
eBDP	emulating Bandwidth Delay Product
EDCA	Enhanced Distributed Channel Access
FCS	Frame Check Sequence
FIFO	First In First Out
FTP	File Transfer Protocol
GI	Guard Interval
GigE	Gigabit Ethernet
HT-SIG	High Throughput Signal
HWMP	Hybrid Wireless Mesh Protocol
IEEE	Institute of Electrical and Electronics Engineers
IFQ	Interface Queue
IP	Internet Protocol
IPTV	Internet Protocol Television
ISP	Internet Service Provider
JFI	Jain's Fairness Index
LAN	Local Area Network
LFN	Long Fat Network
MAC	Media Access Control
MCS	Modulation and Coding Scheme
MIMO	Multiple Input Multiple Output
MPDU	MAC Protocol Data Unit
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit

NRED	Neighborhood Random Early Detection
NS-2	The Network Simulator
OFDM	Orthogonal Frequency-Division Multiplexing
OS	Operating System
PHY	Physical layer
PIE	Proportional Integral controller Enhanced
qdisc	queueing discipline
QoE	Quality of Experience
QoS	Quality of Service
RED	Random Early Detection
RFD-MAC	Relay Full-Duplex MAC
RTS	Request To Send
RTT	Round Trip Time
SIFS	Shortest Inter-Frame Space
SISO	Single Input Single Output
TCP	Transmission Control Protocol
TCP-AP	TCP with Adaptive Pacing
TXOP	Transmission Opportunity
txqueue	transmit queue
U-NII	Unlicensed National Information Infrastructure
UDP	User Datagram Protocol
VoIP	Voice over IP
WaRP	Wearable Reference Platform
WLAN	Wireless Local Area Network
WMN	Wireless Mesh Network

WQM	Wireless Queue Management

Chapter 1

Introduction

It is difficult to believe that the time to send a packet between two wireless nodes that are several meters away from each other may be longer than the time to send the same packet to the moon? Earth-to-Moon communication delay is around one second, whereas many packets in today's Internet experience delay of several seconds due to waiting in deep queues until they get transmitted [3, 4].

1.1 Problem Statement and Motivation

Nowadays, many real-time applications such as Voice over IP (VoIP) and online gaming are latency-sensitive. To achieve a satisfactory performance, these applications need to operate under controlled latencies in the order of milliseconds or even microseconds. Unfortunately, a large file download from the Internet may increase the network end-to-end delay upto several seconds. One reason for this behavior is over buffering in the network stack.

Buffers are designed to absorb transient traffic bursts. However, arbitrarily sized buffers can degrade network performance. Large buffers lead to long queuing delays, while very small buffers may result in network under-utilization. Ideally, the buffers need to be sized just large enough to keep the link saturated at close to full utilization. The purpose of all buffer sizing techniques is to find the optimal buffer size that

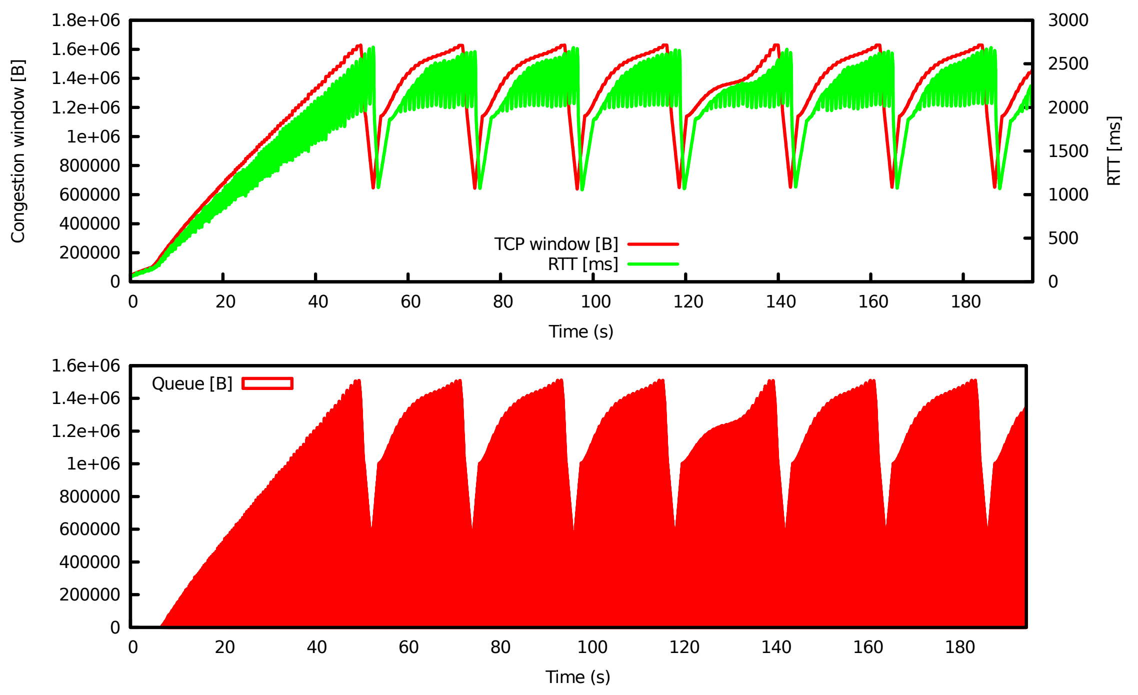


Figure 1.1: TCP congestion window, RTT, and egress queue utilization with a one-hop TCP flow in our 802.11n wireless testbed with 6.5 Mb/s link rate. Buffer size values correspond to values in the stock Linux kernel.

maximize network capacity while minimizing queueing delays.

With declining memory prices and the fallacy that ‘more is better’, network devices are increasingly being over provisioned with large buffers that aim to improve throughput by limiting packet drops. While throughput is the dominant performance metric, packet forwarding latency also impacts user experience. This includes not only real-time traffic such as VoIP, video conferencing, and networked games, but also web browsing, which is sensitive to latencies in the order of hundreds of milliseconds. Recent studies indicated that a one second delay in page load times of e-commerce websites can significantly impact customer conversion [5, 6]. Further, large queueing delays also impact the stability of core internet protocols such as TCP, which rely on timely notification of congestion information to respond effectively.

The goal of the following experiment is to show the significance of high latency in today’s wireless networks. In this experiment, a large file is transferred between two

Linux hosts connected wirelessly via Institute of Electrical and Electronics Engineers (IEEE) 802.11n (Wi-Fi) wireless interfaces. The two hosts are connected at a link rate of 6.5 Mb/s, which is the lowest rate supported by IEEE 802.11n. The detailed experimental setup is described in Chapter 3. The growth of the Transmission Control Protocol (TCP) congestion window as well at the RTT between the two hosts are monitored. The queue utilization of the File Transfer Protocol (FTP) server is also measured. Our results are shown in Fig. 1.1. The TCP congestion window peaks at 1.6 million bytes (window scaling [7] is enabled by default in Linux hosts), with RTT peaking at around 2.4 s. Most of these 'in-flight' TCP segments are queued up at the txqueue interface (Linux default size of 1000 packets), contributing to large queueing delays that lead to the high RTT delays. Most of the widely-deployed operating systems use some variant of loss-based TCP congestion control algorithms. Having large buffers prevents a timely dropping of a packet that is required for conveying network congestion to the TCP sender, leading the TCP congestion window to shoot up to the high values observed in our experiment. It could be noted that with these large buffers, the queue utilization never drops to 0 despite the TCP congestion window halving multiple time over the course of the experiment.

1.2 Thesis Objectives

The main goal of this work is to attain certain Quality of Service (QoS) guarantees in wireless networks. Basically, we are interested in minimizing wireless network latency. One way to achieve this goal is to limit the queueing delay by distinguishing good and bad buffers. The former are buffers that absorbs bursty traffic. On the other hand, bad buffer only contribute to network latency without any noticeable improvement in throughput. In fact, finding the optimal buffer size in any wireless network is challenging due to the special features of wireless networks. For example, the wireless link is often shared among several nodes, hence only a single node can transmit at any given time. According to the widely used Media Access Control (MAC) protocol named Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), the node is going to reschedule its transmission after a random amount of time in case the the medium is found busy. As a result, packet service time varies from one node to the other. Moreover, the capacity of this shared link is not fixed as it varies over time in response to sporadic noise and interference from neighbor wireless nodes. Also, recent MAC enhancements such as frame aggregation allow transmission of large frame aggregates creating further challenges in efficient managing of buffer sizing techniques in wireless networks. Our contributions to address the above mentioned challenges are as follows:

1. Investigate the major buffering layers encountered by a packet through a Linux host and discuss the performance implications of their size. In fact, managing these buffers in a commodity Operating System (OS), such as Linux, is challenging because they are scattered over multiple layers in the software stack. The major sources of buffering encountered by a packet through a Linux kernel have been described. Through careful sizing of the right buffer, the goal is to achieve high utilization of the bottleneck spectrum while maintaining low queueing delays.

2. Describe in details the challenges of buffer sizing in wireless data networks. Not only that we present a summary of buffer sizing solutions available in the literature, but also classify these solutions based on where and how buffer sizing is done. Similarly, we discuss recent Active Queue Management (AQM) techniques in the wired domain. Moreover, we discuss their limitations and why we believe they are not suitable for today's wireless networks.

3. Account for the shared nature of wireless spectrum by formulating the buffer siz-

ing problem in wireless multihop networks as sizing a distributed queue spread over multiple nodes constituting the bottleneck radio neighborhood. Moreover, using a simple cost function, a mechanism for sizing the transmission buffer at individual nodes in the bottleneck neighborhood queue has been proposed. This approach is called Distributed Neighborhood Buffer (DNB). Using simulation and experimental evaluation, it has been verified that DNB achieves close to full network utilization with a significant reduction in the end-to-end delay.

4. Account for wireless channel variable capacity by adjusting the buffer size based on the time needed to drain the buffer. This buffer draining time is recalculated periodically using the current transmission rate. Determining the buffer size adaptively reduce the queueing delays significantly while allowing sufficient buffers to saturate available network capacity. This approach is called Wireless Queue Management (WQM). WQM is both practical and incrementally deployable; it uses existing data traffic as probe for network measurements and does not incur any additional overhead such as time-stamping the packets at their arrival to the queue. Furthermore, WQM improves network fairness as it limits the ability of a single flow to saturate the buffers. WQM is implemented and evaluated in Linux based wireless routers.

5. Analyze the effects of frame aggregation in the IEEE 802.11n/ac standards standard on the optimal buffer size selection. The impact of over-buffering is characterized using both single-hop and multi-hop topologies on an IEEE 802.11n testbed. Also, it is shown that a suitably designed aggregation scheduler can substantially reduce delays, while simultaneously increasing throughput. This is important since it helps us understand the impact of aggregation schedulers in balancing efficient use of channel utilization while minimizing queueing delays. Our proposed queue management scheme, WQM, factors in frame aggregation

to get accurate estimates of queue draining time when deciding about the optimal queue size. Also, WQM enforces high and low limits on queue size based on the length of the transmitted aggregates.

6. Tackle the problem of buffer management in wireless full-duplex systems. In this kind of networks, radios are able to transmit and receive simultaneously using the same channel. This poses additional challenges when trying to optimize the buffer size in the network. We modify WQM to suite wireless full-duplex networks. Performance evaluation shows that our proposed scheme reduces the end-to-end delay significantly at the cost of minimal packet dropping.

1.3 Thesis Organization

The main theme of this dissertation is limiting network latency by optimizing the buffer size in wireless networks. This thesis proposal is organized as follows. In Chapter 2, the "bufferbloat" concept is introduced and a real life example of bloated buffers and their effects is given. After that, buffer sizing challenges in the wireless domain are detailed. This chapter is concluded with a survey of the state-of-the-art solutions to battle bufferbloat and their limitations. Our testbed and experimental methodology are described in Chapter 3. Also in this chapter, various buffering layers in both The Network Simulator (NS-2)[8] and Linux software stack are examined to identify which buffer to tune. Chapter 4 presents the proposed queue sizing mechanism for multihop wireless networks that is called DNB. It starts by describing the notion of a wireless network bottleneck. Then it shows the model for sizing the distributed buffer collectively and allocating this buffer among the contending nodes. Towards the end, it demonstrates the efficacy of our model *via* simulations and experimental evaluation on an IEEE 802.11s testbed. WQM is introduced in Chapter 5, which is a customized queue management and scheduling scheme that is compatible with recent enhancements in the wireless standards. This chapter discuses the design and implementation of WQM on real wireless routers. It also shows the experimental evaluation of this approach compared with the default Linux scheme. Our proposal to solve the problem of buffer management in wireless full-duplex networks is described in Chapter 6. This chapter starts by giving some background material on wireless full-duplexing. After that, the proposed approach is detailed. Towards the end, performance evaluation is shown. This dissertation is concluded in Chapter 7. Finally, our planned future work is listed in Chapter 8.

Chapter 2

State-of-the-Art of Buffer Sizing in Wireless Networks

In this chapter, we start by illustrating the "buferbloat" phenomenon [9] and its implication on wireless networks. After that, we describe the challenges of buffer sizing in wireless data networks. We then present a summary of buffer sizing solutions available in literature. We also provide a taxonomy of other mechanisms proposed in the literature to limit queuing delays in wireless networks. These can be broadly classified as follows:

1. Network-centric techniques used on routers and intermediate network nodes.
2. End-to-end techniques which require tuning higher-layer protocols at the end-hosts to limit the amount of buffering in the network.

We also discuss recent Active Queue Management (AQM) techniques. Since these operate at a different control point, they may be used to complement direct manipulation of buffer sizes. To ground our discussion, we present some performance measurements from our wireless network testbed.

2.1 Introduction to Bufferbloat

Packet-switched networks use buffers to accommodate transient traffic bursts. These buffers aim to prevent packet loss and maintain high output link utilization. Buffer sizing is an important network configuration parameter: under-buffered networks lead to frequent packet loss and subsequent under-utilization of network resources, while over-buffered networks lead to increased queueing delays.

The impact of 'persistently-full', large buffers on network performance have been known for many years. These buffers build up at network bottlenecks along the routing path of a flow. Recently, the term 'bufferbloat' [9] has been used to describe the performance impact when these large buffers are used with simplistic First In First Out (FIFO) queue management with drop tail packet scheduling. Several research papers discuss bufferbloat root causes and the scope of this phenomena in today's Internet [10, 11, 12]. While large buffers may potentially increase throughput by limiting packet drops, big queues can result in high latency. With falling memory prices and the fallacy that 'more is better', this performance degradation from large buffers can be observed in many networking devices, including end-user equipment such as Digital Subscriber Line (DSL), wireless Access Point (AP) or cable routers. In today's networks, bloated buffers may create delays in the order of seconds [3, 12]. Alternatively, some researchers believed that bufferbloat created unnecessary buzz in the community [13, 10, 14]. In a systemic evaluation of bufferbloat, *Cardozo et al.* [13] suggested that bufferbloat might not be a significant problem in practice. Considering the microscopic view of the buffer architecture of typical network devices, they drew attention to the impact of varying the buffer size of various buffers in the network transmit stack. They found out that the occurrence of the bufferbloat phenomenon is not common. This is in agreement with what *Allman* found in his empirical evaluation of bufferbloat [10]. In his study, he concluded that although bufferbloat might happen, it does not happen that often. Also, Hohlfeld *et al.* [14] evaluated bufferbloat from a Quality of Experience (QoE) prospective. They concluded that bloated buffers degrade users QoE only when they are persistently filled.

Most of the existing work on buffer sizing in the literature has been studied in the context of core Internet routers with a large number of flows (*e.g.,* [15, 16, 17], among others). With the network core increasingly being over-provisioned and popularity of wireless hand-held devices, the bottlenecks often lie in the access network. For example, many of the current users, who are accessing server applications via their corporate Wireless Local Area Network (WLAN), are bottlenecked by the wireless link capacity. Similarly, users accessing the Internet via a Wi-Fi based Wireless Mesh Network (WMN) or a 3G/4G cellular network, are likely to be bottlenecked by the slow wireless link capacity. Thus, it is important to study the impact of bufferbloat on wireless network performance.

The wireless environment brings new challenges to our understanding of buffer sizing requirements [18]. In particular, wireless networks have time-varying link rates and variable packet inter-service time. Furthermore, wireless nodes experience interference from other devices sharing the same frequency spectrum. In addition, various enhancements for improving efficiency, such as frame aggregation, also impact packet scheduling dynamics that need to be considered while designing buffer sizing mechanisms. These challenges are discussed in more details in Sec. 2.2. As discussed in Sec. 2.3, there is currently very limited work in the literature on the impact of buffer size on wireless network performance.

To motivate the study of bufferbloat in wireless networks, we conduct several experiments using our Linux-based wireless testbed. The first experiment includes a large file transfer between two hosts connected wirelessly via IEEE 802.11n radios. To simulate the dynamic rate selection of the wireless nodes, the wireless link rate has been changed every 50 seconds: it starts at 144.4 Mb/s, then it drops to 65 Mb/s, 6.5 Mb/s, and finally 13 Mb/s. The growth of the TCP congestion window as well as the RTT between the two hosts has been monitored. Also, we measure the queue utilization of the FTP server and the amount of dropped packets by the sender. Results are shown in Fig. 2.1. It could be observed that the TCP congestion window peaks at 1.6 million bytes (window scaling [7] is enabled by default on Linux hosts), with RTT peaking at around 2.6 seconds. Most of these ‘in-flight’ TCP segments are queued up at the Linux transmit queue (*txqueue*) interface (default size of 1000 packets), contributing to large queueing delays and hence long RTTs. Nowadays, most OSs use some variant of loss-based TCP congestion control algorithms. Having large buffers prevent timely dropping of packets that is required for conveying network congestion to the TCP sender, leading the TCP congestion window to shoot up to the high values observed in our experiment. We note that with these large buffers, the queue utilization never drops to 0 although the TCP congestion window halves multiple times over the course of the experiment. As expected, slower links lead to the longest queueing delays. The huge variation in RTT values clearly suggests that a uniform static buffer size cannot be used for wireless networks that are fundamentally dynamic in nature. Similar performance degradation due to bloated buffers has also been reported for cellular networks [19]. Fig. 2.1 also shows that number of dropped packets increases with network load. This is in agreement with what Fu *et al.* found earlier [20].

Buffers build up at network bottlenecks where the egress rate is below the ingress rate. One example is last mile access for a typical home network: wireless clients connect to a home AP over hundreds of Mb/s using IEEE 802.11n and emerging IEEE 802.11ac radios, but then the uplink to the network of the Internet Service Provider (ISP) is throttled to only tens of Mb/s. These bottlenecks also build up in multihop wireless networks with a mix of radios configurations. This is illustrated using a 4-hop topology in our 802.11n testbed which is shown in Fig 3.1. Frame aggregation, described in details in Sec. 2.2, is enabled at the source and the first hop

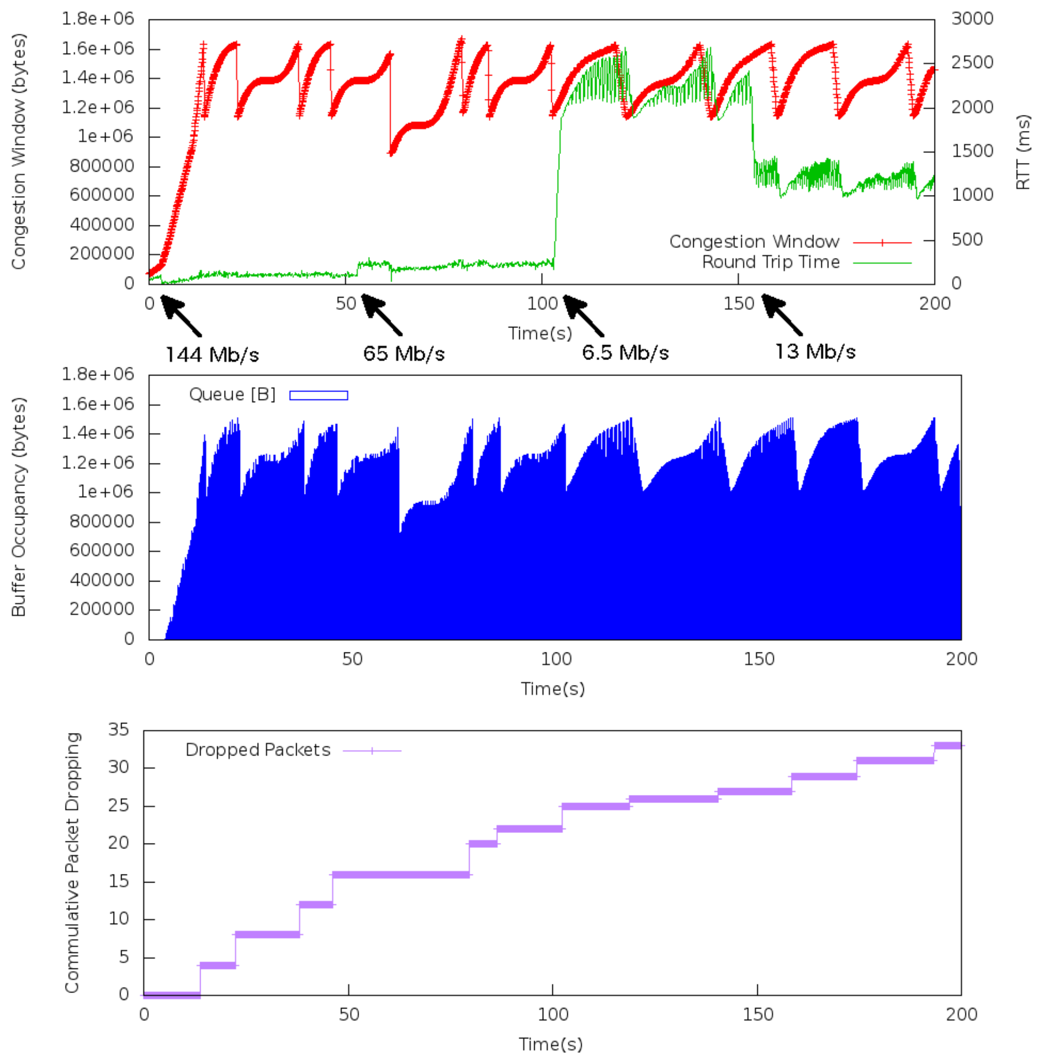


Figure 2.1: TCP congestion window, RTT, egress queue utilization and the number of dropped packets for a TCP flow in an IEEE 802.11n wireless testbed with varying link rates over time. The buffer size corresponds to the default value in Linux.

node, whereas aggregation is disabled in all remaining nodes. Obviously, nodes with aggregation enabled will be transmitting with higher rates. Fig. 2.2 shows that with aggregation disabled on the second relay node, it forms a bottleneck with large queue buildup compared to other nodes in the network. Similar queues also build up when IEEE 802.11n radios are used in conjunction with legacy IEEE 802.11a/b/g radios as the latter are not able to support frame aggregation.

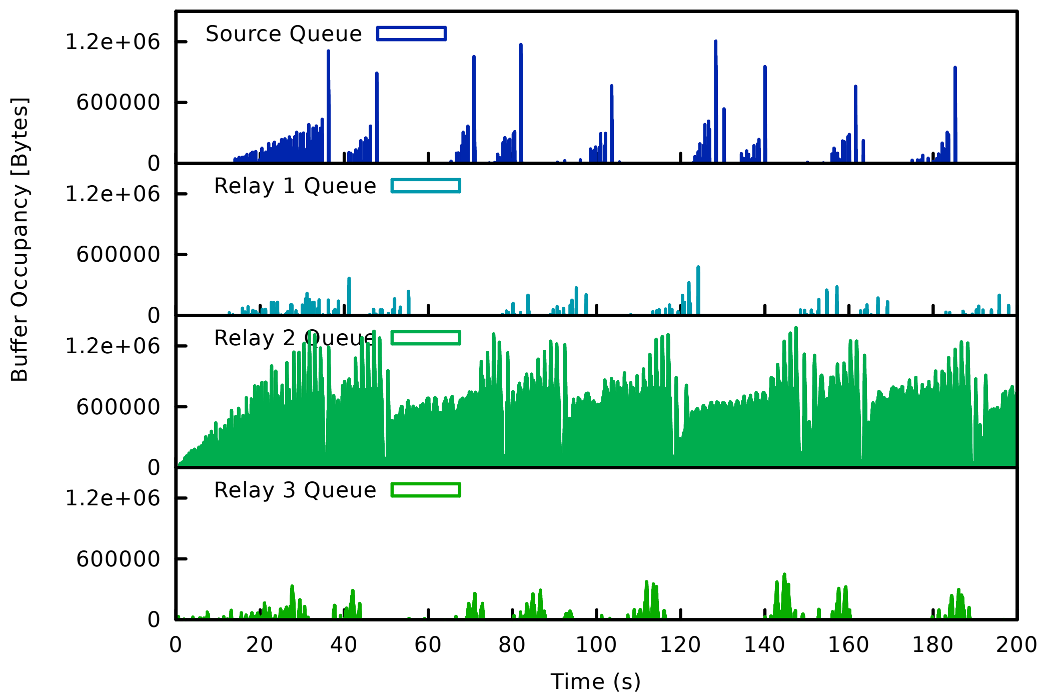


Figure 2.2: Queue utilization for a TCP flow over 4-hop topology in our IEEE 802.11n wireless testbed with 144.4 Mb/s links. A-MPDU frame aggregation is enabled at the source and the first hop only.

2.2 Buffer Sizing Challenges

Buffer sizing techniques and their impact on performance of wired networks is well-understood [15] [16] [17] [21]. However, these techniques cannot be directly applied to the wireless domain because of several unique challenges described below.

2.2.1 Link scheduling

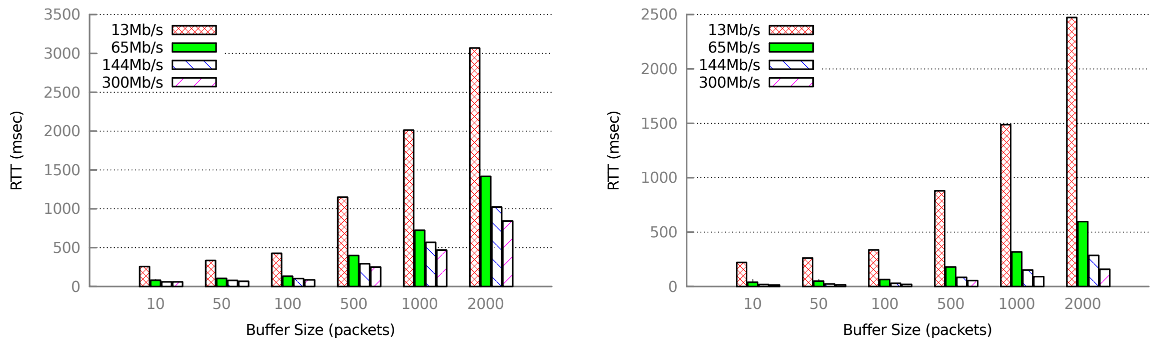
The wireless spectrum is considered a shared resource between a set of neighboring nodes. Interference considerations may require that only one of these nodes transmit at a time. The number of transmit opportunities available to a node is partly dependent on the number of neighboring nodes that are also actively contending for channel access. Thus, unlike a wired link, a wireless link cannot be scheduled independently of its neighboring nodes. This limits the available, usable capacity of a wireless link, as it now varies depending on the network topology and the number of competing flows. Thus while the physical wireless link rate may reach 600 Mb/s, the actual rate achievable by a flow may be significantly less and would further vary over time based on the link scheduling constraints.

2.2.2 Adaptive link rates

Wired link rates are constant and often known apriori. In contrast, link rate adaptation algorithms dynamically set the wireless link rate in response to changing network conditions. These link rates may exhibit significant variations over time, *e.g.*, the link rate for an IEEE 802.11n radio may vary from 6.5 Mb/s to 600 Mb/s. Depending on the link rate adaptation algorithm, these link rates may vary on time scales ranging from seconds to minutes. This has serious implication on the network Bandwidth Delay Product (BDP) which determines the buffer size required for saturating the link.

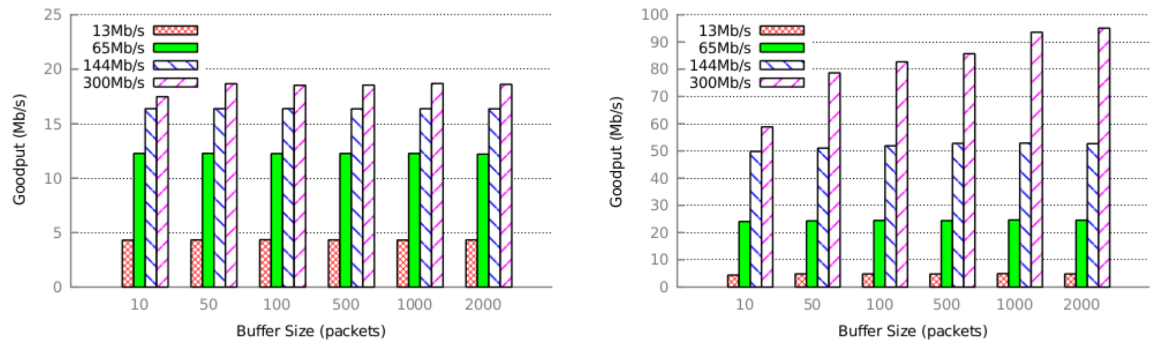
The default rate control algorithm in stock Linux kernel is Minstrel [22]. Minstrel relies on active measurements to select the appropriate link rate. The basic idea behind Minstrel is to search for the best rate by sending packets over fixed rates every 100 ms and decide which one to use based on the packet transmission success rate. The success rate is calculated by dividing the amount of data transmitted by the time for a single try of one packet to be sent on the air. This should be then multiplied by the probability of successful transmission which is the percentage of packets sent successfully out of all packets sent in a given look around.

We perform various experiments in order to evaluate the impact of variable link rate on wireless network dynamics. Our IEEE 802.11n testbed detailed hardware specification could be found in Sec. 3.1. The basic idea is to transfer a large file between two wireless nodes at fixed rate while monitoring goodput as well as other TCP statistics. To examine the effect of dynamic link rates, this experiment has



(a) without A-MPDU aggregation (b) with A-MPDU aggregation

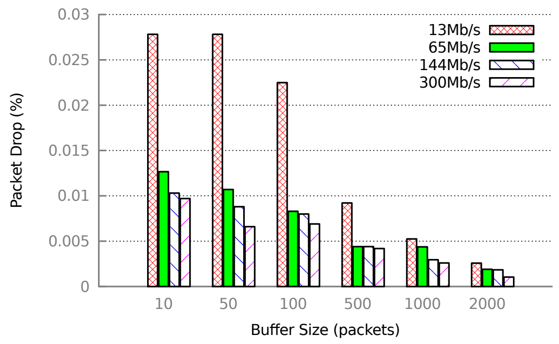
Figure 2.3: Delay experienced by a large file transfer while varying link rate and buffer size.



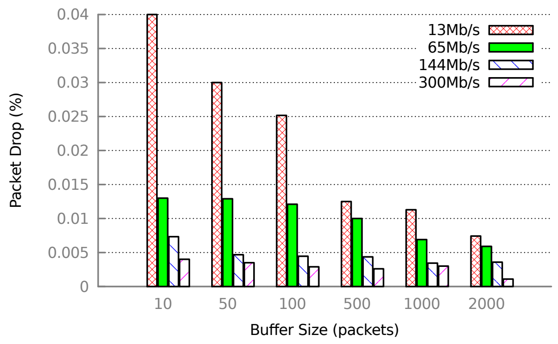
(a) without A-MPDU aggregation (b) with A-MPDU aggregation

Figure 2.4: Goodput of a TCP large file transfer over various link rates and buffer sizes.

been repeated at multiple static link rates and (*txqueue*) buffer sizes while enabling and disabling wireless frame aggregation. Fig. 2.3 and Fig. 2.4 show the end-to-end delay and network goodput respectively over a single-hop wireless network. It can be observed that there is no optimal buffer size that works across the four link rates used in the experiment. Large buffers work well with fast links where they can saturate the link capacity while maintaining acceptable RTT. Small buffers are better suited for slow links, where they limit the queueing delays while giving similar throughput as large buffers. As illustrated in Fig. 2.3b and 2.4b, changing the buffer size from 10 to



(a) without A-MPDU aggregation



(b) with A-MPDU aggregation

Figure 2.5: Packet drops of a TCP large file transfer over various link rates and buffer sizes.

50 packets result in minor throughput improvement for 13 Mb/s to 144.4 Mb/s link, yet come with a 30% increase in throughput for the 300 Mb/s link. However, this buffer size cannot be used across all link rates as the RTT with 13 Mb/s link already exceeds 250 ms over a single wireless hop. Such delays are unacceptable when these queues are shared with real-time traffic. Fig. 2.5 shows the percentage of packet drop for each buffer size used in the experiment. We observe that shrinking the buffer size increases the number of dropped packets. This is in agreement with the results of Dhamdhere and Dovrolis [23], who showed that extremely small buffers lead to high loss rates. Given that even real-time applications require a bounded loss rate to perform well, use of fixed-sized buffers may be infeasible in this case. We also observe that faster links experience lower packet loss as fast links can deflate the queue more quickly, resulting in reduced packet dropping. It is worth noting that bigger buffers always result in higher goodput. For example, we observed a 10% increase in goodput for the 13 Mb/s link moving from 10 to 1000 packet buffer, though this may not be visually apparent in Fig. 2.4b because of the scale of the y-axis on the graph.

2.2.3 Frame aggregation

While the previously mentioned challenges are generally common across wireless networks, standard-specific enhancements introduce additional complexity. The MAC layer in IEEE 802.11n/ac standards introduces two types of frame aggregation techniques [2]: Aggregate MAC Service Data Unit (A-MSDU) and Aggregate MAC Protocol Data Unit (A-MPDU). As shown in Fig. 2.6 [1], an A-MSDU aggregates multiple Internet Protocol (IP) packets into a single frame (up to a maximum implementation dependent A-MSDU frame size of 3,839 B or 7,935 B), and appends Physical layer (PHY) and MAC layer headers as well as a Frame Check Sequence (FCS) trailer. On the other hand, an A-MPDU gathers multiple IP packets back-to-back. A-MPDU is illustrated in Fig. 2.7 [1] and is limited in size to 65,535 B (bound by the 16-bit length field in the High Throughput Signal (HT-SIG) field in the headers). A single A-MPDU can carry a maximum of 64 subframes (limited by the Block Acknowledgement (BA) frame). Each MAC Protocol Data Unit (MPDU) has its own FCS field. This allows the receiver to request a retransmission of corrupted MPDUs by transmitting a BA frame containing a bitmap to identify the status of individual MPDUs. The actual A-MPDU size used for communication may further be limited by the receiver, as advertised in its HT capabilities element. An important difference between these two aggregation methods is the support of QoS traffic classes. A-MPDU sub-frames may belong to various traffic classes. Alternatively, all the sub-frames in an A-MSDU must belong to the same traffic class. As A-MSDU aggregation is not implemented in Linux wireless drivers yet, the experimental work described in this document will be limited to A-MPDU aggregation only.

The impact of packet aggregation in IEEE 802.11n is discussed in prior work [24, 25]. The authors showed that frame aggregation is capable of increasing the network capacity. Also, the authors in [26] proposed a scheme to adaptively change the aggregate size based on nodes mobility patters. However, the impact of frame aggregation

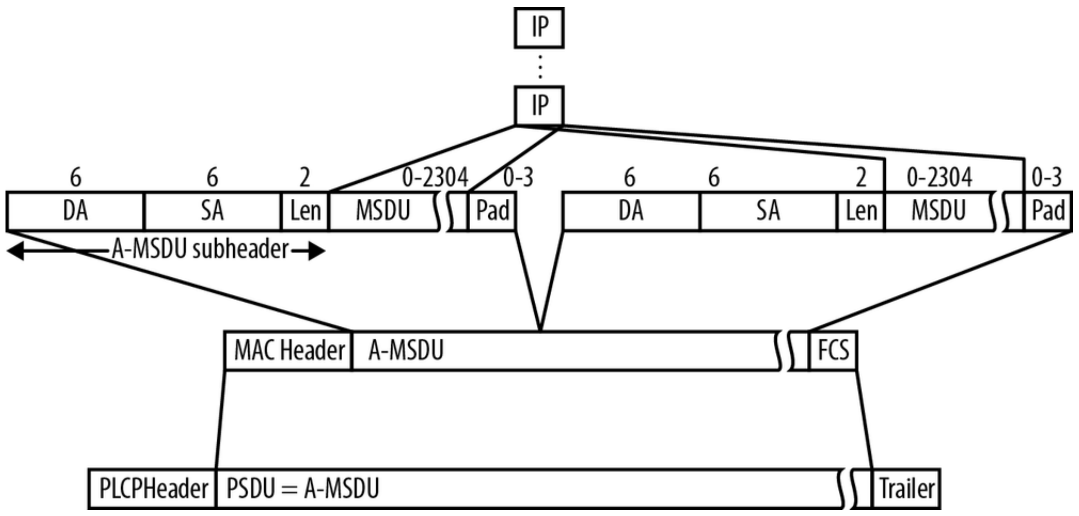


Figure 2.6: Frame format for A-MSDU aggregation in IEEE 802.11n networks [1].

on end-to-end delays, and in particular, its relation to buffer size is not considered. Fig. 2.4 shows that A-MPDU aggregation increases the network goodput by 5× for 300 Mb/s link with large buffers, and up to 3× with small buffers. Fig. 2.5 shows that big buffers lead to slightly higher packet drop rate when A-MPDU frame aggregation is enabled. Both of these observations are attributed to the fact that big buffers allow large aggregates, as shown in Fig. 2.8.

802.11ac, the emerging standard from the IEEE, also supports frame aggregation. The maximum size of an A-MPDU aggregate in IEEE 802.11ac is 1MB whereas an IEEE 802.11n based device is capable of sending aggregates as big as 64KB only. Another major difference between aggregation in IEEE 802.11n and 802.11ac is the fact that the latter always sends frames as aggregates even if the sender has only a single frame to send. Hence, the use of constant small buffers will be inefficient as it is going to increase the MAC overhead of sending out these aggregates.

The A-MPDU aggregation logic is not specified in the standard and is implementation dependant. Ideally, these algorithms need to balance the requirements between making efficient use of channel resources using large frame aggregates when the channel quality is good while trying to minimize queueing delays by processing packets

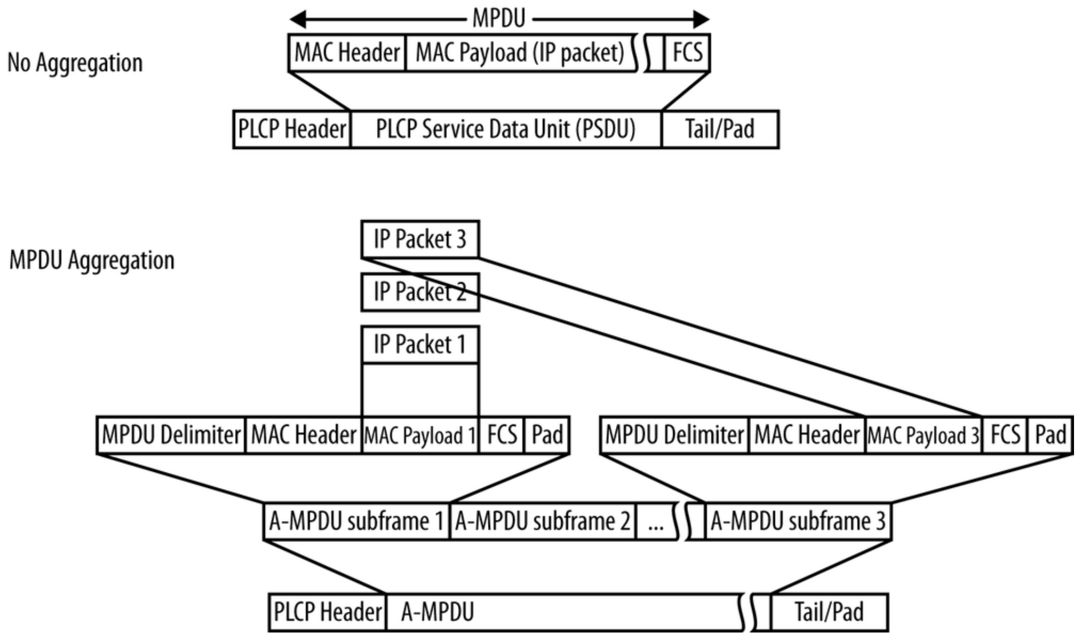


Figure 2.7: Frame format with and without A-MPDU aggregation in IEEE 802.11n networks [1].

quickly. A naïve packet scheduler that always waits to assemble a maximum sized A-MPDU may maximize network capacity, but is going to increase the latency despite this. Furthermore, our experimental analysis confirms that using such scheduler does not allow end nodes to establish TCP connection due to long delays. In fact, the ath9k [27] driver scheduler, used in our setup, prefers timeliness over capacity; instead of waiting to assemble maximal allowable A-MPDU aggregates which may maximize throughput, it aggregates as many MPDUs as available at that time in the buffer subject to the regulatory and receiver constraints. This A-MPDU aggregation logic is listed in Algo. 1. Thus, while it may not use optimal A-MPDU sizes, the fact that it never spends time waiting for new frames to arrive from higher layers can result in minimizing end-to-end delays. Fig. 2.3 shows that this technique may reduce the delay by up to 5×.

To understand the relation between buffer sizing and A-MPDU length, we fix the transmission rate in our IEEE 802.11n wireless testbed and monitor the A-MPDU length over a large file transfer between two nodes while varying the length of Linux

```
Input: Number of frames in buffer ($Q$), Regulatory A-MPDU size limit ($\delta_1$),
       Receiver A-MPDU limit advertised in its HT Capabilities element ($\delta_2$),
       Number of frames in this A-MPDU ($n$)
Output: Assemble A-MPDU for transmission
1  n = 0, A-MPDU = 0;
2  While $Q \neq 0$ do
3      // Check for regulatory or receiver limits;
4      if ($n > \delta_1$ or $n > \delta_2$) then
5          break;
6      Add padding (if necesary) to align A-MPDU frame boundry;
7      Link this frame to the aggregate A-MPDU;
8      Q- -; // Decrement buffer count by 1;
9      n++; // Increment frame count by 1;
10 Deliver assembled A-MPDU to driver transmit function;
```

Algorithm 1: A-MPDU aggregation logic as implemented in ath9k [27] driver.

Transmit Queue (*txqueue*) after each run. Fig. 2.8 shows that higher transmission rate and bigger buffer allow the sender to transmit longer A-MPDUs. Error bars in this figure, which represent maximum and minimum aggregate length, prove that IEEE 802.11n devices do not always send aggregates with the same size. In fact, this figure also shows that link rate directly determines the maximum A-MPDU size. For example, aggregation is disabled at the 6.5 Mb/s link rate as transmitting a large A-MPDU at this rate may violate the 4 ms frame transmit duration regulatory requirement in the 5 GHz band.

User Datagram Protocol (UDP) flows are used in real-time communication, such as online games, Internet Protocol Television (IPTV), and VoIP. Hence, it is important to compare such flows to other flows that favor reliable delivery over timely delivery. We repeat the same experiment with UDP instead of TCP to evaluate the interaction of frame aggregation with UDP flows. The only difference in the experiment setup is enabling the default rate control algorithm in Linux (Minstrel) instead of fixed link rates. Latency and goodput results are shown in Fig. 2.9. UDP consistently achieves higher goodput compared to TCP. This is due to multiple factors: Firstly,

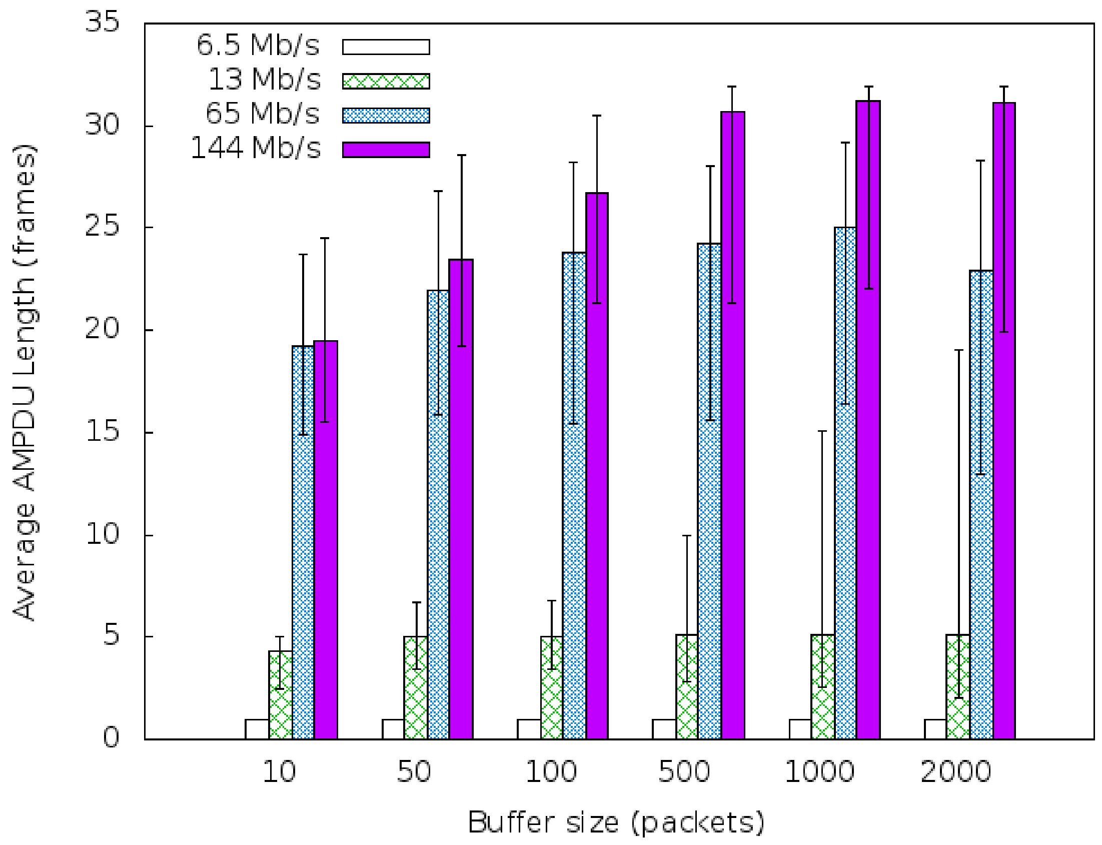
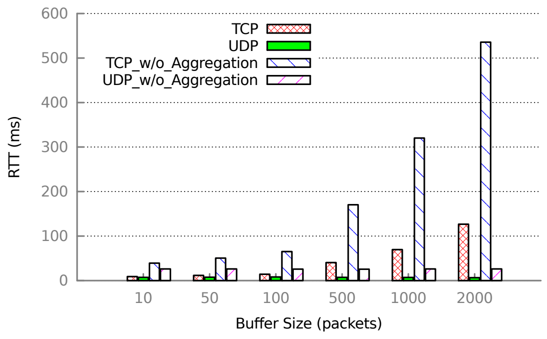
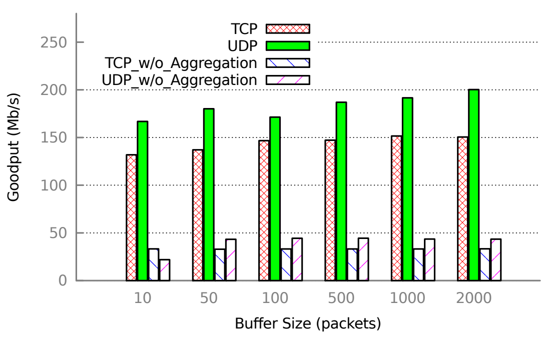


Figure 2.8: Average A-MPDU length of a TCP large file transfer for various link rates and buffer sizes.

UDP does not incur the overhead of transmitting TCP Acknowledgement (ACK) segments, and thus the capacity spared can be used to send additional data packets. Secondly, TCP employs congestion control algorithms, while UDP can saturate the medium with a sustained traffic rate. In fact, the only case where TCP goodput outperforms UDP happens when using the 10 packets buffer; we attribute this to the high UDP drop rate (around 9%) which limits the performance of A-MPDU frame aggregation. Fig. 2.9b shows that UDP goodput stabilizes when the aggregation is disabled, though we observe slight variation in results with aggregation for buffers larger than 10 packets. This is because large buffers allow longer aggregates. For example, the average number of frames per aggregate increases from 16.1 for the 50 packets buffer to 17.6 for a 2000 packets buffer. Fig. 2.9a shows that UDP delays



(a) Latency analysis



(b) Goodput analysis

Figure 2.9: Delay and goodput comparison of both TCP and UPD flows with and without A-MPDU frame aggregation.

are always smaller than TCP; this is because UDP does not incur extra delays for connection management and reliability.

2.2.4 Variable packet inter-service rate

The packet inter-service rate of a wired link is deterministic for a given packet size. In contrast, the packet inter-service rate of a wireless link is variable due to several reason. First, MAC protocols such as CSMA/CA use random backoffs to reduce the probability of a collision. Second, the Bit Error Rate (BER) of a wireless link is typically orders of magnitude higher than a wired link (BER of 10^{-5} to 10^{-3} for a wireless link vs. 10^{-15} to 10^{-12} for a wired link) [28]. Wireless MAC protocols use Automatic Repeat Request (ARQ) to maintain packet transmission reliability. As a result, a packet may be transmitted multiple times (*e.g.*, up to 7 retries per IEEE 802.11 standard specifications [2]) before it is successfully received, contributing to variation in inter-service delays.

2.2.5 Power management

Power management strategies of portable wireless devices can also affect buffer sizing. Transmit power control can reduce co-channel interference as well as conserve device battery. However, transmit power also impacts link reliability and subsequently the link rate selected by the rate control algorithm, which then determines the appropriate buffer size as described earlier. Other features, such as Power Save Mode in IEEE 802.11, also impact buffer sizing as the transmitter needs to factor in the sleep cycle of the receiver.

2.2.6 Network management

Resource-constrained wireless networks, enforced by wireless network management layer, use mechanisms for QoS provisioning, which may produce additional delays before a packet is processed. Other middleboxes in the network path can further delay packets, such as security inspection performed by some cellular network providers. Finally, mobile devices may also encounter large delays during the hand-off process between base stations. These requirements introduce additional considerations for optimal buffer sizing in wireless networks.

2.2.7 Multi-hop challenges

Multi-hop wireless networks further exacerbate the challenges described above. Due to the shared nature of wireless spectrum, a flow not only competes for transmission opportunities with other flows (*inter-flow contention*), but also contends with its own packet transmissions along the hops to the destination (*intra-flow contention*). This adds to the link scheduling and variable packet inter-service time challenges described above. Further, the abstraction of a ‘bottleneck’ in a shared wireless medium translates to a set of nodes in the network that experiences high channel contention. Since a traffic flow may traverse multiple hops in the network, the bottleneck is going to be scattered over multiple nodes [29]. It is unclear how to size buffers in this distributed environment. Moreover, a node in a multi-hop network may need to relay traffic to other nodes in the network. Hence, additional measures is required to provide

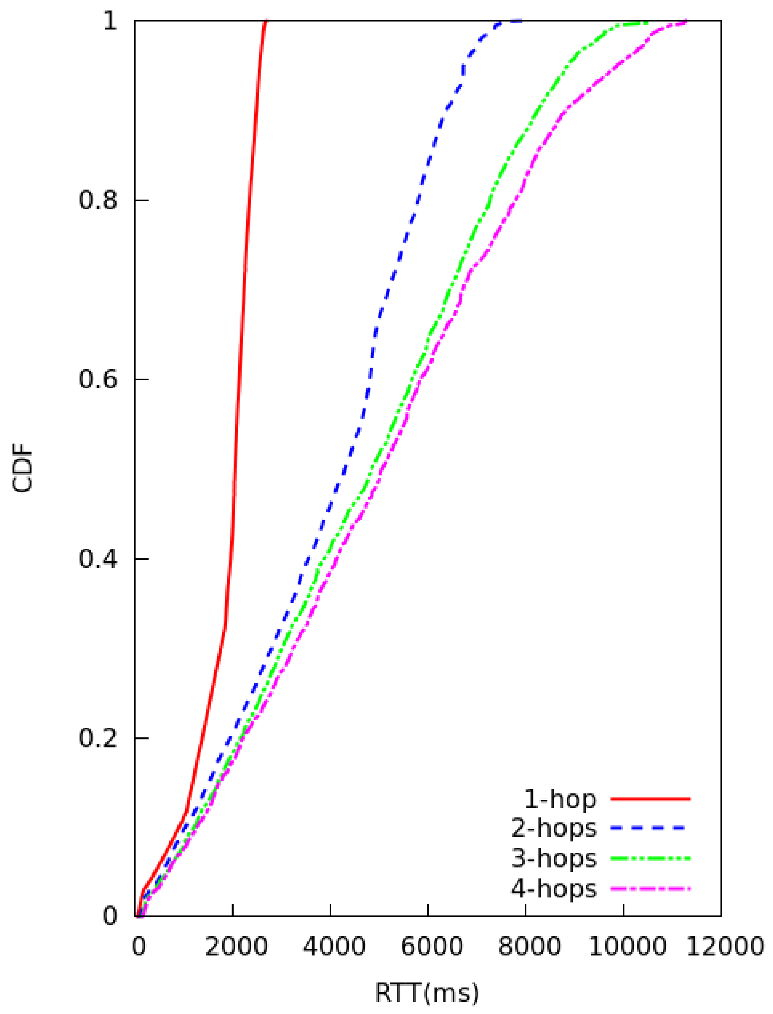


Figure 2.10: End-to-end delay CDF of a large file transfer over topologies with increasing number of hops.

isolation and fairness between flows in this kind of networks.

In order to illustrate the effects of mulit-hop topologies on network dynamics, a large file is transferred between two hosts in our IEEE 802.11n testbed while varying the number of intermediate hops from one to four. We use static routing in this experiment. We observe from Fig. 2.10 that the peak RTT increases by 3$\times$ (from 2.69 to 7.95 seconds) when the hop count changes from one to two while the network goodput decreases by half (from 4.87 to only 2.41 Mb/s). It could be observed from Fig. 2.11 that while the network capacity is decreased by half when moving from 1-hop to 2-hop topology, the TCP congestion window is actually increased to fill up the

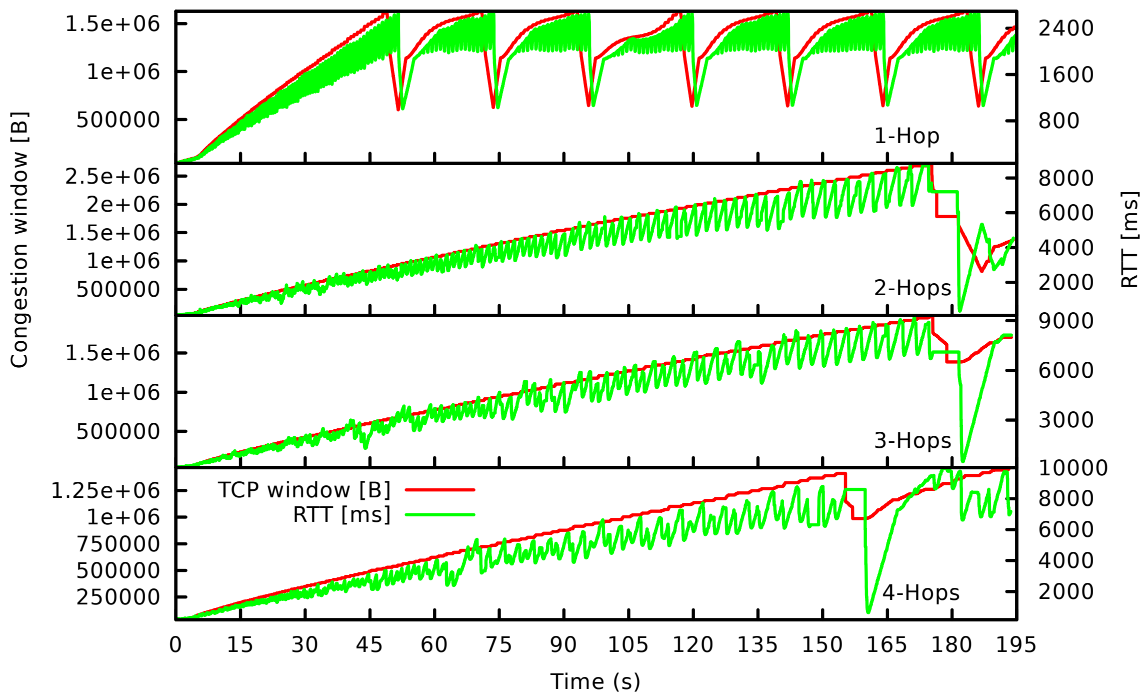


Figure 2.11: TCP congestion window and end-to-end delay of a large file transfer over topologies with increasing number of hops.

Flow #	1 hop	2 hops	3 hops
Flow 1	18.53 Mb/s	16.98 Mb/s	9.85 Mb/s
Flow 2	18.45 Mb/s	0.10 Mb/s	0.77 Mb/s

Table 2.1: Per flow goodput for a bidirectional file transfer over a multihop wireless network.

additional available buffer on the intermediate relay node, reflecting this large increase in RTT. Similar behavior, *i.e.,* longer delays and lower goodput, is experienced when we look at the 3-hops and 4-hops topologies. To study the effect of these persistently full buffers on the network fairness characteristics, we repeat the same experiment with a bidirectional file transfer instead of a single file transfer. Table 2.1 lists the goodput per flow over various hops. We observe severe unfairness between the two flows, with the flow starting first starving out the flow starting later in the experiment. This is because the flow starting first quickly saturates intermediate hosts buffers, resulting in dropped packets and timeouts for the flow starting later.

Routing protocols play a significant role in the performance of multi-hop wireless networks. Adaptive load-aware routing protocols that route around the congested parts of the network may yield better performance than static routing. Multi-path routing protocols can also be used, where a data stream is distributed as multiple data flows that can take link-disjoint or even node-disjoint paths. However, the performance gains of these protocols may be limited by the network topology. For example, in infrastructure WMNs where the bulk of traffic is routed either towards or away from a single gateway router providing Internet connectivity, load-aware or multi-path routing has limited leeway. Routing protocols can also be used to address channel contention issues such as hidden terminal or exposed terminal problems. To address some of these issues, Dousse [30] proposed a hole routing scheme. The main idea behind this scheme is to reduce the queue size on relay nodes to only one packet to mitigate the problem of low goodput in multi-hop networks. Hence, every node in this scheme has either a packet or a hole. In fact, this routing scheme helps solving bandwidth allocation problem. Similarly, Xue and Ekici [31] used adaptive routing, among other techniques, to increase energy efficiency in multi-hop networks. Finally, Draves *et al.* [32] tackled the problem of routing multi-radio devices. They came up with a routing protocol that takes into consideration loss rate and channel bandwidth to be able to choose a high throughput path.

2.2.8 Implementation challenges

Implementing buffer sizing mechanisms on modern operating systems represents another challenge as buffers exist on multiple layers in the software stack. It is unclear as to which of these buffers should be tuned. For example, the Linux network stack uses *txqueue* to buffer packets between the kernel network subsystem and the device driver. *txqueue* may be scheduled using a variety of queueing disciplines. Its typical size is 1000 packets in order to support networks with high BDP. In addition to *txqueue*, packets may also be queued at the device driver ring buffers (also called

Tx/Rx descriptors). These buffers are used to hide the latency of the interrupt processing overhead. One of the main issues with device driver ring buffers is the fact that it is sized by the number of descriptors which vary in size. As a result, the actual time to empty the buffer cannot be estimated precisely. Moreover, IEEE 802.11 Enhanced Distributed Channel Access (EDCA) implementation uses a different ring buffer for each separate traffic class, making the buffer sizing problem even more complicated. For example, small buffers for real time voice and video provide the required QoS, whereas background and best effort traffic prefers bigger buffers to achieve high throughput. In reality, the size of ring buffers are driver dependent. For example, Tx ring size is 200 packets for ath5k driver [33], and 512 packets for ath9k [27]. There is an inherent trade off between small and large device driver rings. Small device driver buffers help in lowering memory consumption. However, they result in packet dropping if the system is Central Processing Unit (CPU) bound or the memory bus is saturated. Linux buffering layers are discussed in greater details in Sec. 3.2.

Unsurprisingly, a significant disparity has been found in the buffer size used in various research platforms. Many papers that presented results using the NS-2 [8] simulator use a buffer size of 50 packets which is the default size for the queue object in this simulator. The legacy open source MadWifi driver [34] for Atheros chipset uses a transmission buffer of 200 packets. The newer ath5k driver [33] for the same hardware divides this 200 packet buffer equally among four queues representing the traffic Access Category (AC) as defined in EDCA mechanism. The ath9k [27] drivers for IEEE 802.11n Atheros radios use a buffer of 512 packets equally divided among the four ACs. This shows that buffer sizing is a platform dependent problem in the wireless domain.

2.3 Bufferbloat Solutions

Wireless networks buffer sizing efforts in the literature could be categorized based on how and where to tackle the problem of excess buffering into two categories: Network centric and end-to-end centric methods. Actual sizing of buffers inside routers and middleboxes is what meant by network centric methods. The latter includes efforts to control the amount of buffering in the network by tuning TCP parameters on network edges. To put it another way, they modify the transport layer of the end hosts.

2.3.1 Network centric methods

Buffer sizing has been extensively studied for core Internet routers. A widely used rule-of-thumb is to set the buffer size to be larger than the BDP of the network [21], *i.e.,* $B \geq RTT \times C$, where C is the *bottleneck* link capacity along the path, and RTT is the effective round-trip propagation delay through the bottleneck link. This buffer size B represents the number of packets required in order to fully utilize the bottleneck link while the TCP source recovers from a loss-induced window reduction. This rule-of-thumb holds for a single TCP flow in a wired network. When a large number of flows share a bottleneck router, the window size is going to be synchronized in lockstep. As a result, the aggregate window size is still a sawtooth pattern, and the BDP guideline for sizing the bottleneck buffer still holds. However, when there are N desynchronized flows and the window processes are independent, the buffer size B can be reduced to $B = RTT \times C/\sqrt{N}$ while still achieving near 100% utilization [15]. Enachescu *et al.* [16] suggested that B can be further reduced to $O(log\ W)$, where W is the window size of each flow, resulting in buffer sizes of only $10 - 20$ packets while achieving $85 - 90\%$ of link utilization.

Van Jacobson drew attention to the importance of persistently full buffer in 1989, leading to the development of Random Early Detection (RED) [35] algorithm. This technique prevents the formation of large queues at the bottleneck by dropping packets probabilistically when the queue size reaches certain threshold. RED represents one of the early AQM techniques. These techniques attempt to avoid large queue build-up at intermediary network hosts through proactive, probabilistic packet drop. In spite of the fact that several versions of RED were proposed in the literature, none of them succeeded to gain traction because they tend to be hard to configure. Not to mention their slow response to fast changes in the environment [36].

Recently, a no-knobs AQM technique called Controlled Delay (CoDel) [36] has been proposed. This algorithm was essentially designed to detect bad queues, which are defined as the queues that last longer than one RTT resulting in a constantly high buffering latency. CoDel is a self-configurable algorithm and has shown good performance over other AQMs [37]. Unlike traditional AQM techniques, CoDel does not monitor the queue size or queue occupancy directly. Instead, it keeps track of the packet sojourn time through the queue. In other words, it monitors how long each packet stays in the queue. This makes the algorithm independent of link rate and has clear reflection of user experience. For a given interval, the algorithm finds the lowest queuing delay experienced by all packets. Once the minimum queuing delay exceeds a predefined value for a fixed amount of time, the algorithm goes into the dropping phase. Packet dropping is going to be stopped only once the queuing delay falls under the predefined value. Furthermore, CoDel keeps track of the minimum queue length for a period that is longer than the nominal RTT. This is important because the algorithm does not allow packet dropping if the queue has less than one Maximum Transmission Unit (MTU) worth of bytes. On eof the limitations of CoDel is that all the packets are going to be dropped at the queue egress which is clearly a waste of network resources. Another key point to remember is that CoDel allows the buffer to be as small as one frame which will restrict aggregate formation resulting in lower utilization.

Another no-knobs AQM variant, called Proportional Integral controller Enhanced (PIE) [38], that combines the benefits of both RED and CoDel has been proposed in the literature recently. Similar to RED, PIE randomly drops a packet when experiencing congestion. However, congestion detection in PIE is based on the queuing delay instead of the queue length. PIE determines the level of network congestion based on latency moving trends. Upon packet arrival, the packet may be dropped according to a dropping probability that is determined by the dequeue rate and the length of the queue. Both PIE and CoDel target queuing delay directly without necessarily restricting the buffer size. However, unlike CoDel, PIE does not keep track of the per packet timestamp. Moreover, it decides whether or not to drop a packet before actually queuing it.

While neither CoDel nor PIE are specifically designed for wireless networks, simulation results show that they manage to respond to changes in link rates while achieving a utilization similar to the traditional tail drop approach. This, however, may not be enough to support fast mobility in wireless devices such as vehicular speed mobility. Furthermore, it is unclear how AQM based techniques can be effectively used in multi-hop wireless networks where the bottleneck spans multiple distributed nodes [39]. A recent study compared several AQMs in terms of latency over wired and wireless networks and showed that both CoDel and PIE perform worse than the well-known RED technique with auto-tuning capabilities [40].

Another effort that also targets bufferbloat and is considered complementary to CoDel is Network Transmit Queue Limits [41] designed by Tom Herbert from Google. This patch deals with device internal transmit queue which is the last step in Linux buffering architecture. Device driver queue usually supports a ring of descriptors that hold several packets. These ring descriptors are used to keep the transmitter busy instead of waiting for the kernel before transmitting each and every packet. Network Transmit Queue Limits is composed of two complementing parts: Byte

Queue Limit (BQL) and Dynamic Queue Limit (DQL). Each one of these parts is targeting a specific problem. BQL solves the issue of ambiguity of queue limits in the device driver. Traditionally, queue limits are specified by the number of hardware descriptors which vary in size. Instead, BQL specifies the size of the queue in bytes, rather than packets, which indicates the time to empty the queue more accurately. On the other hand, DQL tires to set the limit on queue size dynamically in order to adapt to changes in the system load. It does so by monitoring the queue occupancy and tuning the queue size accordingly. Unfortunately, these schemes are implemented for wired networks only and could not be extended directly to wireless networks due to the variable link capacity nature of the wireless channel [42]. Finally, both of these schemes never take frame aggregation into consideration when selecting the optimal buffer size.

Calculating the BDP of a wireless network is different from a wired network because of coupling between the bandwidth and delay of a wireless link [43]. In long-haul wired links, the transmission delay is small compared to the propagation delay; thus a source can often inject multiple packets back-to-back into the pipe. The same is not true for CSMA/CA based wireless links; first, the transmitter has to contend for channel access for *each* transmission, and second, the IEEE 802.11 transmitter requires an ACK before it can transmit the next frame. As a result, the delay of transmitting a packet over wireless network is strongly coupled with the link's effective bandwidth.

There is very limited amount of work on buffer sizing for wireless networks. For single-hop IEEE 802.11 WLANs, the AP buffer can be sized dynamically to strike a balance between channel utilization and delay requirements of various flows. Li *et al.* [44] studied adaptive tuning of IEEE 802.11 AP buffers using three algorithms: emulating Bandwidth Delay Product (eBDP), Adaptive Limit Tuning (ALT), and A* algorithm.

eBDP extends the classical BDP rule to AP buffers. Due to the variation in wireless link rates, eBDP adaptively sets the buffer size limit based on the current mean service time of the packet, T_{serv}. T_{serv} is nothing but the time difference between the packet getting to the head of the queue and its successful transmission. The goal is to limit T_{serv} to some predefined maximum T_{max}. The algorithm decreases the AP buffer size, Q_{eBDP}, when T_{serv} is increased, and vice versa. eBDP could be formalized according to the following equation:

$$Q_{eBDP} = min(T_{max}/T_{serv} + c, Q_{max}^{eBDP}) \tag{2.1}$$

where Q_{max}^{eBDP} is the maximum allowable buffer size and c is a constant added to accommodate short-term packet bursts.

Although simple in concept, eBDP has a fundamental limitation. Despite that packet service time is a good indication of channel contention, it does not capture queueing delays. To put it another way, an AP with multiple download flows and with no (or few) competing upload streams experiences small packet transmission service time, allowing eBDP to build up large buffers that will never be empty, thus increasing queuing delays. ALT feedback algorithm improves on eBDP as follows: it monitors buffer occupancy and modifies the size accordingly. Let $t_b(k)$ and $t_i(k)$ represent the queue busy and idle durations respectively during the measurement interval k. Then the queue size for the interval $k+1$, $q(k+1)$, is set as follows:

$$q(k+1) = q(k) + a\,t_i(k) - b\,t_b(k) \tag{2.2}$$

where a and b are design parameters. The algorithm tries to balance the time the queue is idle (t_i) and the time it is busy (t_b). That is to say, if $a\,t_i(k) = b\,t_b(k)$, then the buffer size in the $k+1$ interval is kept unchanged. Furthermore, the maximum queuing delay T_{max} is set to 200 ms instead of the mean RTT. According to the authors, this value is considered to be an approximate upper bound of usual Internet flows. However, this raises a question on how valid this assumption will be on other types of wireless networks. ALT essentially operates based on a feedback loop: the buffer size depends on measured link utilization, which in turn depends on the buffer size. However, it suffers from low convergence rate. ALT needs around three minutes to converge to small buffer size when the number of competing upload flows increase from 0 to 10.

A* is a hybrid approach that combines the two previously mentioned methods. This algorithm calculates two queue sizes: (1) Q_{eBDP}, by monitoring the mean service time of packet transmissions, and (2) Q_{ALT}, by monitoring the buffer occupancy percentage. It then simply chooses the minimum of these two values.

$$Q_{size} = min(Q_{eBDP}, Q_{ALT}) \tag{2.3}$$

Thus A* uses eBDP in order to react quickly to sudden changes in the packet service time. Statistical multiplexing makes further reduction in AP buffer size feasible. The ALT part of A* can be used to further tune the buffer size.

A* is evaluated through NS-2 simulation and testbed implementation. The simulation results show that A* is able to reduce the buffer size from 350 packets to only 100 packets when the number of concurrent flows increases from 0 to 10. By varying the number of concurrent flows and link rates in simulation runs, it was shown that A* is able to maintain good balance between throughput and delay under various network conditions. Real testbed implementation of A* recommends the use of buffer sizes that is smaller than the standard 400 packets buffer. These smaller buffers result in significant reduction in queuing delay while achieving similar throughput compared to fixed buffer scenarios. This reduction in delay holds also in the case of short-lived TCP flows. It was also shown that the mean completion time for these short-lived flows is reduced by a factor of two which in-turn results in better user experience.

One of the main limitations of the A* algorithm is that it only works on AP buffers; it is unclear if this scheme can also be implemented on client devices to manage queueing delays for uplink flows. Moreover, A* attaches a timestamp to each packet entering the queue to be able to calculate the service time of the packet. This is clearly an overhead that affects the overall network performance. Further, A* performance was not evaluated using real IEEE 802.11n devices and hence it is unclear if the small buffer sizing approach recommended in this work will scale with frame aggregation (such as in IEEE 802.11n radios), where sufficient buffers may be required to assemble the large aggregates supported by the standards. Indeed, some results suggest that eBDP, which forms the basis of A*, yields sub-par performance for some practical IEEE 802.11g/n networks [45]. Above all, A* was not evaluated over multi-hop environments. In fact, extending this scheme for multi-hop networks is not straightforward due to the fact that each node in the A* algorithm selects its buffer size independently. As the bottleneck in multi-hop networks spans multiple nodes, some coordination between nodes may be needed to find the optimal buffer size.

Bruno *et al.* [46] suggested having a large buffer at the AP to improve fairness between upstream and downstream TCP flows in single-hop wireless networks. The intuition behind this idea is to increase the queuing delay for the ACK's in order to limit the rate of the upstream wireless source. However, TCP stability will be disturbed by ACK's long queuing delays. For multi-hop wireless networks, a number of publications (*e.g.* , [47] among others) proposed using the queue size for detecting network congestion and enforcing subsequent rate adjustments. Moreover, Xu et al. [48] targeted the unfairness problem using buffer sizing methods. They solved the unfairness issue by monitoring the size of the distributed neighborhood queue. They extended RED into Neighborhood Random Early Detection (NRED). NRED drops packets probabilistically if the size of the distributed neighborhood queue exceeds certain threshold. Although the performance of our proposed scheme was not compared to NRED, we show that it outperforms the state of the art AQM techniques in the literature such as PIE [38], which is similar to NRED in the fact that it drop packets randomly. Furthermore, PIE showed better performance compared to Random Early Detection. Also for mobile adhoc networks, Dousse [30] proposed the reduction of queue sizes on relay nodes to only one packet to mitigate the problem of low throughput in multi-hop networks. Obviously, this is infeasible for IEEE 802.11n networks with A-MPDU size of tens of subframes.

Finally, we would like to highlight that several other papers studied the problem of wireless networks buffer sizing from other aspects. Thottan and Weigle [49] studied the effect of tuning several AP parameters to meet various application QoS as well as ensuring flow level fairness. One of the studied parameters is queue size of EDCA different access categories. Chen et al. [43] investigated the BDP in mobile adhoc networks as well. In fact, they did not focus on the buffer sizing problem. Instead, they come up with an upper bound for BDP that is determined by the number of round-trip hops on the path. This upper bound is also used in calculating the TCP congestion window limit to avoid TCP's congestion window overshooting problem. Since this upper bound on BDP is based on the coupling of delay and bandwidth of the wireless link, it is not going to hold in case of packet aggregation that is introduced in IEEE 802.11n standard.

2.3.2 End-to-end methods

Various enhancements to end hosts, including modifications to the TCP stack, have been proposed over the years. In particular, delay-based TCP variants, such as Vegas, try to avoid congestion while maximizing network capacity. Unlike loss-based TCP flavors, such as Reno, delay-based schemes use the difference between the actual and expected flow rate to infer the amount of congestion in the network and adjust the window size linearly. However, TCP Vegas suffers from several limitations. Firstly, the accuracy of base RTT in TCP Vegas is effected by path re-routing. Based RTT is used for indicating network congestion and over-estimating this value due to network persistent congestion only prolongs the congestion. Secondly, Vegas leads to unfair rate allocation due to over-estimation of the connection propagation delay [50]. Finally, mixing Vegas with other loss-based based TCP flows may further exacerbate the fairness issues.

TCP's window-based congestion control algorithm can trigger a burst of packet transmissions. To give an example, when a node receives a cumulative ACK or several back-to-back ACKs. Sustained bursts can lead to packet losses with subsequent drop in throughput. TCP pacing [51] addresses this by spacing out the transmission of a congestion window (`cwnd`) worth of packets over the estimated RTT interval, *i.e.,* the packets are injected into the network at a rate `cwnd`/RTT. While this minimizes traffic burstiness (and subsequent queueing delays), it may reduce throughput compared to unmodified TCP because of delayed congestion signals and synchronized losses [52]. ElRakabawy and Lindemann [53] observed that the inherent variability in the RTT of multi-hop wireless flows may offset these synchronization issues. They proposed a rate-based transmission algorithm over TCP called TCP with Adaptive Pacing (TCP-AP): instead of spacing the transmission of `cwnd` worth of packets over the RTT of a flow, the transmission rate is computed using 4-hop propagation delay, determined by spatial reuse in a chain of wireless nodes, and coefficient of variation of recent RTTs to identify incipient congestion along network path. TCP-AP improves goodput by reducing collisions at the link layer and also improves fairness between flows. However, paced traffic fares poorly when competing with non-paced traffic for network resources, creating a significant impediment in wide-spread deployment of pacing-based protocols [52]. Moreover, all the benefits of TCP-AP are linked with the accurate estimation of the 4-hops propagation delay which is difficult to estimate in wireless networks [54].

Warrier *et al.* [55] proposed a differential backlog congestion control for wireless networks. The idea is to throttle the flow control of the sender based on the queue back pressure. Hence, if the queue grows beyond certain threshold, the sender is going to reduce its transmit rate and vice versa. One limitation of such approach is the need of the queue occupancy information from several hops in the multi-hop networks. Transferring this information over multiple hops is a considerable overhead that will affect the practicality of the proposed approach. Alternatively, our proposed method works with local knowledge even for multi-hop networks [56].

Recently, Jiang *et al.* [19] proposed a receiver based TCP solution targeting bufferbloat in cellular networks called Dynamic Receive Window Adjustment (DRWA). It adaptively adjusts the size of receive window which indirectly manages the amount of buffering in the network. The authors showed that DRWA is more efficient than setting a fixed upper bound on sender's congestion window which may be suboptimal under varying channel conditions. They claimed that DRWA outperforms delay-based TCP versions that suffer from throughput degradation. However, requiring changes to the client network stack limits the application of this scheme to networks where the devices are managed by the service provider such as operator-locked smartphones in cellular networks. Also for cellular networks, Chan *et al.* proposed a scheme called Sum-of-Delay (SoD) [57] that tackles the problem of link buffer size estimation in 3G/4G mobile data networks. The main idea behind (SoD) is to modify TCP congestion control to function based on estimated queue length instead of packet loss events. Similarly, this scheme has practical limitations because of the large deployed base of TCP.

Chapter 3

System Description

This chapter is dedicated to the description of our experimental testbed. Both hardware and software setups are described in details. Furthermore, packet buffering layers in Linux networking stack are analyzed. This is an important milestone towards designing an optimal buffer sizing scheme targeting real hardware implementation.

3.1 Testbed Specifications

We deploy a 10-node WMN testbed in our campus. Node locations are shown in Fig. 3.1 and were determined, in part, by availability of power and Ethernet sockets. Our testbed consists of small form-factor Shuttle [58] computers. Each one of these computers, or Shuttle boxes, have an Intel E7500 Core 2 Duo processor, and 1 GB of memory. This testbed was initially equipped with IEEE 802.11 a/b/g radios, represented as blue radio icons in Fig. 3.1. Later, the wireless interface cards have been upgraded to support several PHY and MAC enhancements introduced by the IEEE 802.11n standard. This upgrade requires several testbed design modifications in both hardware and software sides. The main differences between the two testbed versions are highlighted in Table 3.1.

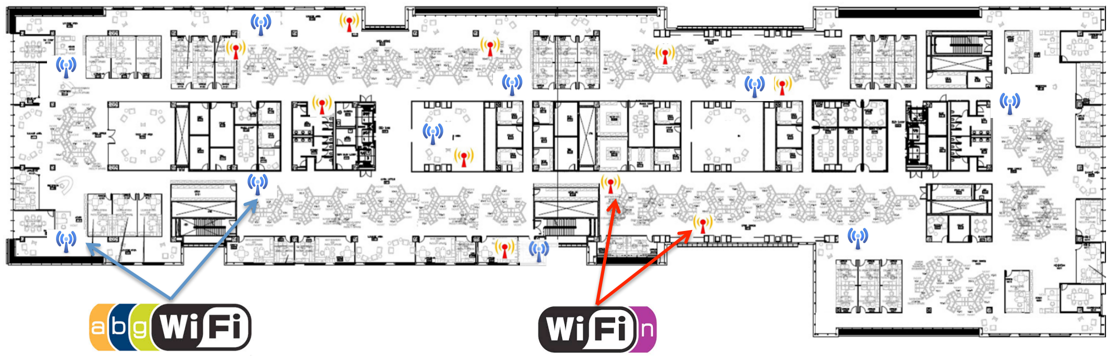


Figure 3.1: Our testbed node positions are identified by radio icons.

3.1.1 IEEE 802.11a/b/g testbed

Wireless nodes in this testbed are equipped with TP-Link WN350G (Atheros AR2417) IEEE 802.11 a/b/g cards. Our network uses the 2.4 GHz band and shares the spectrum with the production wireless Local Area Network (LAN) in our campus. Nodes run a custom 2.6.35 Linux kernel with web100 [59] instrumentation to monitor the state characteristics of the TCP streams. We use ath5k drivers [33] to configure the nodes' wireless interfaces. The platform runs the open80211s [60] implementation of the IEEE 802.11s standard. This was partly driven by our desire to understand and contribute to the ongoing development of the framework. This testbed uses Hybrid Wireless Mesh Protocol (HWMP) routing protocol with airtime metric [61] for path selection. Wireless link rates are fixed to 11 Mb/s to avoid flow rate fluctuations due to link rate adaptation algorithm that may try to adapt to interference from our campus production WLAN.

In the software side, iperf [62] is used to generate test traffic. The segment size of the generated TCP streams is 1448 bytes. The kernel then couples this with a 20-byte TCP header, a 10-byte TCP timestamp option header, a 2-byte padding field, and a 20-byte Internet Protocol version 4 (IPv4) header for a total Maximum Segment Size (MSS) of 1500 bytes. Cubic [63] is the default TCP congestion control algorithm in Linux kernels since 2.6.18. It replaces the linear window growth function of TCP variants such as NewReno and SACK by a cubic function so as to better improve the utilization of Long Fat Network (LFN)s.

3.1.2 IEEE 802.11n testbed

The testbed wireless cards have been replaced with TP-Link WDN4800 (Atheros AR9380) IEEE 802.11a/b/g/n wireless cards in order to utilize IEEE 802.11n MAC-layer enhancements. This chipset supports three spatial Multiple Input Multiple Output (MIMO) streams for a maximum wireless link rate of 450 Mb/s. It also supports dual band radio operation. Hence, we use the 5 GHz Unlicensed National Information Infrastructure (U-NII) radio band as it does not interfere with the production WLAN on our campus which uses the 2.4 GHz spectrum. Since the transmission range of the 5 GHz radios is shorter than the 2.4 GHz radios, the testbed nodes have been brought closer to each other. The new locations of the nodes are represented by red radio icons in Fig. 3.1. These locations are again determined by the availability of power and Ethernet drops. Unless otherwise stated, the nodes are placed around 10 m apart from each other. The experiments are repeated along various source and destination pairs across the testbed to offset any location-specific wireless idiosyncrasies.

Adding IEEE 802.11n wireless interface cards requires several modifications to the software setup of the tesbed. Wireless nodes in the testbed run a custom 3.17.7 Linux kernel with web10g [64] instrumentation to monitor the state characteristics of our TCP streams. Web10g uses an efficient Netlink-based kernel Application Binary Interface to make the TCP statistics available in userspace. We use ath9k drivers [27] to configure the wireless interfaces. Minstrel [22], the default rate control algorithm in stock Linux kernel, is enabled in the boxes. For some experiments, we disable Minstrel and vary the link rate manually in the testbed in order to monitor the effect of adaptive link rate on various network dynamics .Both iperf [62] and netperf [65] are used to generate traffic. The Linux distribution in our testbed nodes uses TCP Cubic

Parameter	Value	
Standard	IEEE 802.11a/b/g	IEEE 802.11n
Link rates	11 Mb/s	6.5 Mb/s, 144.4 Mb/s, 300 Mb/s
Radio band	2.4 GHz ISM	5 GHz U-NII
Spatial streams	SISO stream	3 MIMO streams
Frame aggregation	Not supported	A-MPDU
Linux kernel	Custom 2.6.35 with web100	Custom 3.17.7 with web10g
Traffic source	iperf	iperf, netperf
Packet size	1500 Bytes	1500 Bytes
txqueue size	1000 packets (Default size)	1000 packets (Default size)
TCP Flavor	Cubic	Cubic
Routing	HWMP	Fixed path routing

Table 3.1: Experimental setup for both testbed versions.

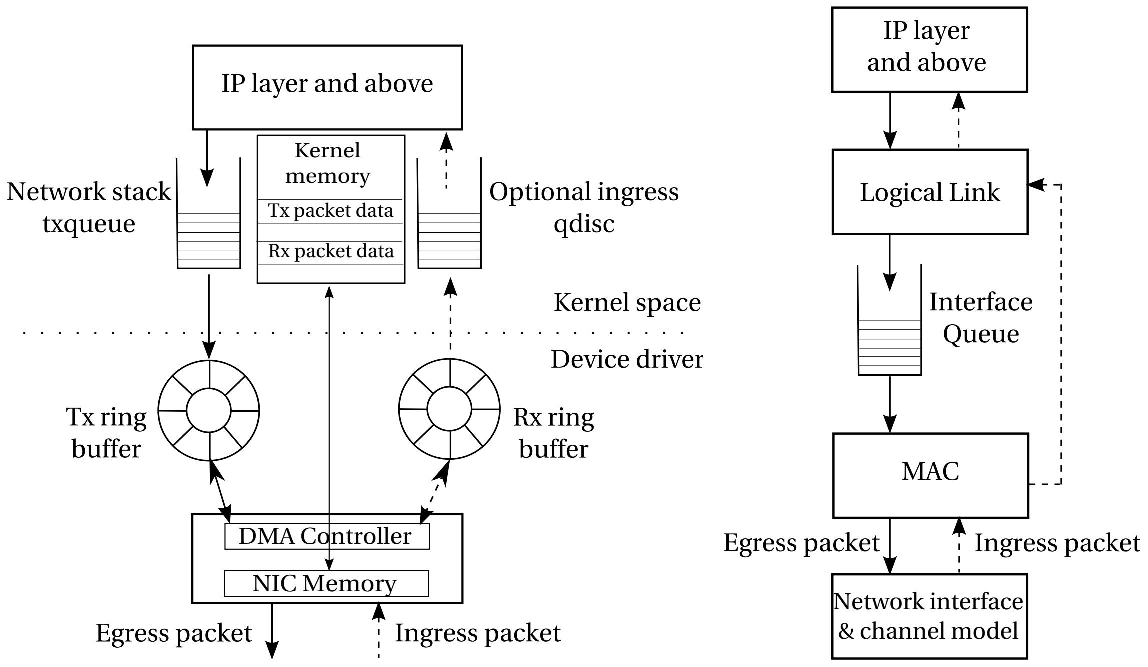
by default. Likewise, window scaling [7] is enabled, as per the default configuration on all recent Linux kernels. This allows TCP to support large receive window sizes.

3.2 Buffering Layers

In this section, an overview of various layers of buffering is provided. This is important as it helps identifying which buffer to tune when trying to control the bufferbloat phenomenon.

3.2.1 Buffering in Linux network stack

Packets in a contemporary modern OS encounter buffers at multiple layers in the software stack. Each of these buffers may introduce queueing delays. In this section, we describe the major buffering layers encountered by a packet as it traverses a Linux host. Furthermore, we discuss the performance implications of sizing various types of buffers. Although the focus in this document is on the Linux kernel, the impact of multi-layered buffering is also present in others OSs such as the Berkeley Software Distribution (BSD) and Windows. Thus, the design guidelines we are providing are



(a) Linux packet transmission and reception (b) ns-2 mobile node

Figure 3.2: Simplified architecture of buffers used in packet transmission/reception in Linux hosts as well as simulator mobile nodes.

also applicable to these systems.

A packet transmission through Linux host encounters buffers at multiple layers. At the radio interface side, there are hardware queues inside the physical adapter. Without these queues, the communication between the host and the adapter necessarily follows a 'stop-and-wait' approach, which is inefficient considering the large transfer latency between the host and the adapter. At the software side, the Linux network stack includes multiple layers of buffers.

Network stack's transmit queue: The Linux kernel introduces a transmit queue (txqueue) between the kernel network subsystem and the device drivers. This queue can be scheduled by a rich set of queueing discipline (qdisc) tools, providing a flexible traffic control framework.

Typical size: Most current Linux distributions set the default transmit queue length to 1000 packets. Older distributions generally used a 100 packet transmit queue for

Fast Ethernet; the queue size has since been scaled up for 1 Gigabit Ethernet (GigE) interfaces.

Impact of size: A large enough transmission queue is necessary to support line rate transfers for networks with high BDP. The value should be lowered for slower devices with a high latency as to prevent fast bulk transfers from disturbing real-time interactive traffic.

Device driver ring buffers: Device drivers use queues to amortize interrupt processing overhead. This includes the Tx and Rx ring buffers (also called Tx/Rx descriptors). Each descriptor represents a data structure that describes a buffer and its attributes to the network controller. The controller then uses this information to transfer data between the controller and the host memory. Typically, device drivers use a single Tx ring buffer. However, drivers with multiple queues are increasingly being used. For instance, IEEE 802.11 EDCA implementation uses separate queues for each of the four traffic ACs in the standard: Background, Best effort, Video, and Voice. Typically, these traffic classes have different buffering requirements: real-time voice and video traffic prefers timeliness over reliability, and hence small buffers; background and best effort traffic can build relatively longer queues to take advantage of lull in transmission of higher priority traffic.

These queues may also be involved in performing MAC-specific tasks. For example, packet aggregation in IEEE 802.11n is performed at the MAC layer and thus needs a separate aggregation buffer. This buffer size is usually limited to a maximum of 64 MPDUs as per the limitation of the BA response frame as defined in the IEEE 802.11 standard specifications, though some drivers use smaller values.

Typical buffer sizes: The Tx/Rx ring size is driver dependent. Table 3.2 shows the default ring buffer size for ath5k and ath9k drivers in the 2.6.35 kernel. The mac80211 Linux subsystem supports drivers with up to 4 queues per EDCA ACs. The ath5k driver has a total buffer size of 200 packets divided equally amongst the four ACs.

Driver	Tx descriptors	Rx descriptors
ath5k (Atheros 802.11 a/b/g)	200 (50 for each AC)	40
ath9k (Atheros 802.11n)	512 (128 for each AC)	512

Table 3.2: DMA ring buffers in stock 2.6.35 Linux kernel.

Similarly, the ath9k driver equally distributes a buffer size of 512 packets amongst the ACs.

Impact of size: These values reflect various design and performance considerations. On one hand, memory consumption can be lowered by using small buffers. However, if the system is CPU bound or has a saturated memory bus, then small buffers may result in dropped or missed packets. The routers used in our testbed deployment are not CPU or memory bound (our system hardware specifications are listed in Sec. 3.1), and thus buffer sizing considerations ignore hardware bottlenecks.

3.2.2 Which buffer to tune?

One of the main issues with device driver ring buffers is the fact that it is sized by the number of descriptors which vary in size. As a result, the actual time to empty the buffer cannot be estimated precisely. Hence, in all our experiments the device driver Tx ring buffer is kept at a constant value and the txqueue size is tuned per the proposed framework requirements. Furthermore, this approach helps in maintaining some buffers at the transmit queue layer so as to allow the flexibility of using Linux's traffic control framework, if required. The size of txqueue buffer is varied using the `ifconfig` userspace utility. In a similar manner, the size of the Tx ring descriptor is fixed to a static value by extending the Linux `ethtool` utility.

3.2.3 Simulation Buffering

To make the discussion about buffering layers complete, buffering in network simulators should be considered. A simplified architecture of the NS-2 [8] wireless node is shown in Fig. 3.2b. The Interface Queue (IFQ) sits between the logical link layer and the MAC layer. The default queue object is a FIFO queue of size 50 packets. The MAC layer uses callback functions to pull packets out of the IFQ when it is ready to transmit. A software framework has been instrumented to monitor the queue size of the IFQ and to modify it at run time.

Chapter 4

Distributed Neighborhood Buffer

4.1 Overview

In this chapter, we study the impact of buffer sizing in IEEE 802.11-based WMNs providing backhaul connectivity for last mile Internet access [66]. These multihop networks have stationary mesh routers affixed to rooftops and building structures, forming a multi-hop wireless backhaul. Client devices connect to their preferred mesh router via wire or an orthogonal wireless channel. The capacity of this link is higher than the capacity achievable along the multi-hop path of wireless mesh routers, and thus buffers are more likely to build up at the mesh routers. Most of the traffic in this network is directed towards or away from the *gateway* mesh router that bridges traffic to the wired Internet.

We solve the buffer sizing problem in WMNs using the notion of distributed queues spread over a contention neighborhood. For a TCP flow traversing multihop wireless links, the end-to-end rate is determined by the contention neighborhood that allocates the minimum transmission time to member links. For optimal network utilization, this bottleneck contention neighborhood needs to be fully utilized. Thus, the buffer sizing problem has been mapped as a distributed buffer management problem for nodes sharing the bottleneck contention neighborhood. A packet transmission by *any* of these nodes fully utilizes the bottleneck spectral resource for the duration of this transmission. A novel mechanism for sizing and distributing these buffers has been proposed to keep the bottleneck close to full utilization while minimizing queueing delays. This mechanism considers the network topology and wireless link rates, and is thus adaptive to changing network conditions. In contrast, a fixed, pre-set buffer size does not suffice across the range of configurations achievable with IEEE 802.11 platform [44]. That is to say, a single IEEE 802.11n radio interface can connect at link rates varying from 600 Mb/s (with 4 spatial streams) to 1 Mb/s (for backward compatibility with IEEE 802.11b), and thus requires widely different buffer sizes for saturating the wireless channel. Using analysis and performance evaluation with simulations and a testbed prototype, It has been shown that the proposed scheme reduces the end-to-end delay of a TCP flow by a factor of $6\times$ or more for the evaluated scenarios, while maintaining close to full network utilization.

4.2 Design

First of all, we identify the bottlenecks that limit the end-to-end rate of a multihop wireless network. Then, we show how distributed buffers are associated with this bottleneck. Finally, we discuss the practicality of the proposed design.

4.2.1 Bottleneck collision domain

The wireless medium is a shared resource. This limits the set of nodes that can transmit concurrently. The notion of collision domain [67] has been used to identify these interfering links. The *collision domain* of link l_i is defined as the set of all links that contend with link l_i. In general, identifying the interfering links is a hard problem, as links exhibit varying degree of interference [68]. Additional mechanisms requiring active or passive monitoring may have to be used [69]. In this work, the two-hop interference model [70] has been utilized to identify interfering links; two

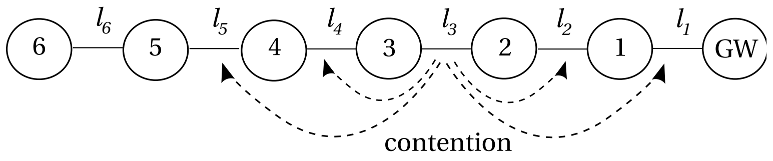


Figure 4.1: With a two-hop interference model, the collision domain of link l_3 includes links $l_1, l_2, l_4,$ and l_5.

links interfere if they operate on the same channel and one endpoint of one link is within the transmission range of one endpoint of the other link. The two-hop model approximates the interference avoidance in an IEEE 802.11 network with Request To Send (RTS)/Clear To Send (CTS) enabled. While the model has limitations in representing complex topologies, it has simply been used it for ease of exposition and the proposed buffer sizing mechanism is independent of the underlying interference model. In Fig. 4.1, the collision domain of link l_3 includes links $l_1, l_2, l_4,$ and l_5. It could be noted that this notion of interference domain overestimates the impact of interference by limiting the number of nodes that can concurrently transmit. However, the actual interference between nodes is determined by the interference range, which is typically longer than the 1-hop communication range. The combined model offers acceptable accuracy with computational simplicity and practical feasibility.

The *utilization* of a collision domain is the sum total of transmission times for all links in a collision domain. The feasibility constraints on scheduling require that this utilization cannot exceed 1. Mathematically, it could be represented as follows: Let $R_{(m,n)}$ be the link rate between neighboring nodes (m, n) and let $r_{(m,n)}$ be the traffic carried by this link. Let r_i be the end-to-end rate for flow f_i. Then $r_{(m,n)} = \sum_{i: f_i \text{ traverses } (m,n)} r_i$. Let $\mathbf{C} = \{C_1, C_2, ..., C_j\}$ be the set of j collision domains in this network. Ignoring physical and MAC layer headers, the feasibility constraints require

$$\sum_{\forall\,(m,n)\,\text{in}\,C_p} \frac{r_{(m,n)}}{R_{(m,n)}} \leq 1, \ \forall p \in \{1, 2, ..., j\}$$

A *saturated* collision domain is defined as a collision domain which is fully utilized. A saturated collision domain becomes a *bottleneck* for a flow if that flow has a maximal rate amongst all other flows using this collision domain. A multi-hop flow may be part of one or more collision domains; its end-to-end rate is then bound by the collision domain that assigns it the lowest rate.

4.2.2 Distributed neighborhood buffers

Since the bottleneck collision domain limits the end-to-end rate of a multi-hop flow, it is important that the spectral resource in this bottleneck be fully utilized. From a buffering perspective, this is achieved if at least one node in the bottleneck always has packets to send. A packet transmission by any of these nodes fully utilizes the available spectral resource for the duration of the transmission. Thus, instead of considering buffers at individual nodes, sizing the collective buffer of the bottleneck collision domain has been proposed so as to saturate its available spectral resource. This neighborhood buffer is distributed over the set of nodes constituting the bottleneck. Its size is the sum of the buffer sizes of all the constituent nodes. Similarly, its packet arrival rate is the sum of the arrival rates at individual nodes. However, this neighborhood buffer does not exhibit the FIFO characteristics due to distributed queues and stochastic scheduling of IEEE 802.11 MAC protocol. Fig. 4.2, shows the distributed FIFO buffer at nodes constituting the bottleneck collision domain for a 6-hop chain topology.

It is important to note that the definition of distributed FIFO buffers includes all packets queued at a node. For instance, packets queued at node 5 for transmission to node 6 via link l_6 in Fig. 4.2 are included. This can be mitigated by maintaining a *virtual* per-link queue at each node, and including only the virtual queues in the distributed buffer that directly correspond to the links in the collision domain. In this current work, however, the entire FIFO buffer at a node in the distributed queue has

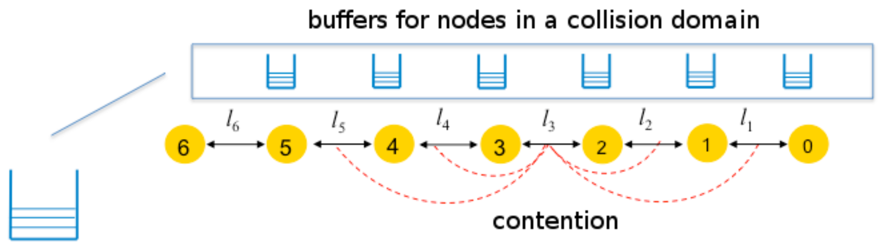


Figure 4.2: The distributed neighborhood buffer of a collision domain includes the buffers associated with the constituent nodes in that collision domain.

been included. This allows node 6 to transmit to 5 (but not vice versa) when there is an active transmission on link l_3. Limiting node 5's transmission to 6 over-estimates the impact of interference when 3 transmits to 2, but may be necessary to prevent interference at 3 when it receives a packet from 2 (as interference range is typically larger than transmission range, a concurrent transmission from node 5 may corrupt reception at node 3).

One simple (though naive) way to enhance the utilization of bottleneck spectrum is to make the bottleneck neighborhood buffer arbitrarily large, and then hope that it will always have sufficient packets in queue to saturate the bottleneck even when the TCP source recovers from a packet loss. However, it is easy to visualize that if this buffer is any larger than the rate supported by the network, it will only add to the queueing delay experienced by the flow. At the same time, if the bottleneck is under-buffered, it will not always be fully utilized. The challenge, thus, is to determine the size of this neighborhood buffer that maintains the bottleneck near 100% utilization while minimizing the queueing delay.

4.2.3 Determining network parameters

A mesh node needs current network information to compute its buffer per our proposed framework. Below are some guidelines to determine this information in an efficient, distributed manner.

Efficient techniques for building interference maps [69, 71] can be used to determine collision domains. Simpler models, like the two-hop interference model considered in this work, can be captured using local information at a node. For example, we can use routing information to determine one and two-hop neighbors both up and down the routing tree, or monitor the relay of traffic by neighboring nodes to determine two-hop neighbors. The utilization of each collision domain can then be monitored locally (*e.g.* Atheros hardware maintains a 32-bit register counter for medium busy utilization), and the collision domain with a utilization higher than some threshold (*e.g.* say 90%) constitutes the bottleneck for a flow.

Once the collision domain is identified, wireless link rates can be inferred as follows: Device drivers maintain various wireless link statistics (including PHY rate information) between a node and its neighbors. Two-hop link rate information can be determined through capture and analysis of PHY frame header when the neighboring nodes relay the packet up or down the routing tree. This link rate information along with the packet size for a flow (again, known locally) is used to compute the RTT for propagating the packet through the collision domain.

The available BDP of a wireless network changes over time. Nodes recompute their buffer when network conditions change, such as changes in routing path or link rates, or when there are changes in flow activity information. This change in flow activity can be detected by a node using packet analysis for flows that originate locally or monitoring the wireless medium for flows in the neighborhood. If a flow is considered as a sum of various subflows originating from client devices associated with a mesh node, then this flow activity information may be stable for durations of tens of seconds (the large delay values discussed in this work are observed for long-lived flows such as large file transfers). A mesh node monitors its state for changes in network state or flow information, and recomputes the buffer when necessary. A system flowchart for this cycle is shown in Fig. 4.3.

Finally, it could be noted that inaccurate estimation of various network parameters

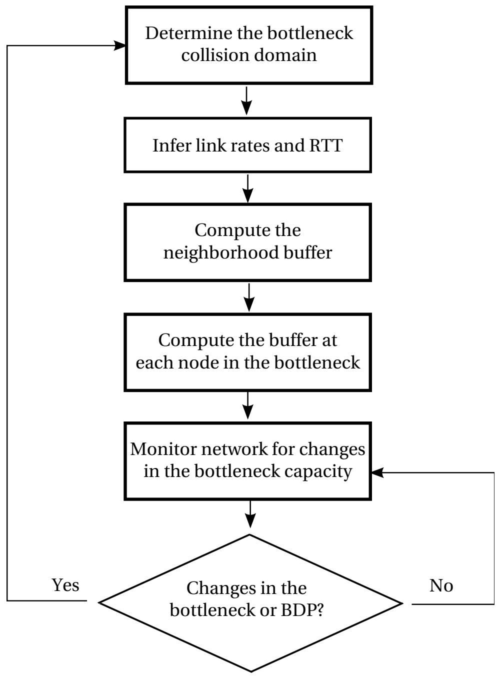

Determine the bottleneck collision domain
Infer link rates and RTT
Compute the neighborhood buffer
Compute the buffer at each node in the bottleneck
Monitor network for changes in the bottleneck capacity
Changes in the bottleneck or BDP?
Yes
No

Figure 4.3: Flowchart for DNB buffer sizing heuristics.

yields suboptimal BDP for the bottleneck collision domain which is used to estimate the neighborhood and the per-node buffer. This suboptimal buffer size impacts the bandwidth-delay tradeoff performance for a flow. The approximations considered in Sec. 4.3 err on the side of overestimating BDP, thus preferring bandwidth at the cost of slight increase in end-to-end delay.

4.3 System Model

Consider a WMN with $\mathbf{N} = \{n_1, n_2, ..., n_N\}$ be the set of wireless mesh routers. Assume that the set $\mathbf{M}$, where $\mathbf{M} \subseteq \mathbf{N}$, represents the set of nodes in the bottleneck collision domain. Let $M = |\mathbf{M}|$, *i.e.,* a node in a bottleneck collision domain contends with $M-1$ other nodes for channel access. The node n_i has a buffer of size b_i packets. Let $B = \sum_{\forall\, i \,\mathrm{in}\, \mathbf{M}} b_i$ be the cumulative distributed buffer for this collision domain.

The first step in the analysis below is to determine B, and then propose a model for distributing it as b_i units amongst the M contending nodes.

4.3.1 Neighborhood buffer size B

Consider a single multi-hop TCP stream. Assume that the wireless network can support this stream at a maximum rate of λ packets/s.[1] Let the stream be in Additive Increase Multiplicative Decrease (AIMD) congestion avoidance phase. The TCP window size reaches a value of W_{max} packets before experiencing a loss. As a result, the sender halves the window size to $W_{max}/2$ packets. Since the window size limits the number of unacknowledged packets in flight, the sender, on average, waits for a time interval $\frac{W_{max}/2}{\lambda}$ before starting to retransmit. At the same time, the distributed buffer B takes B/λ s. to drain. For full utilization of the bottleneck spectrum, the source should start retransmitting before the buffer is fully drained, *i.e.,* $\frac{W_{max}/2}{\lambda} \leq B/\lambda$, or

$$B \geq \frac{W_{max}}{2} \tag{4.1}$$

When the TCP source starts retransmitting, its rate (*i.e.,* `cwnd`/RTT) should be higher than λ, as otherwise the network capacity is under-utilized. Thus $\frac{W_{max}/2}{RTT} \geq \lambda$, or,

$$\frac{W_{max}}{2} \geq RTT \cdot \lambda \tag{4.2}$$

From (4.1) and (4.2),

$$B \geq RTT \cdot \lambda \tag{4.3}$$

This equation is similar to that of wired networks [15]. The main difference is in λ, the maximum carrying capacity of the network. In wired networks, λ is determined by the bottleneck link capacity. In multi-hop wireless networks, it is limited by the bottleneck collision domain. The exact values of λ and RTT depend on the

[1] We use packets instead of bits for ease of exposition.

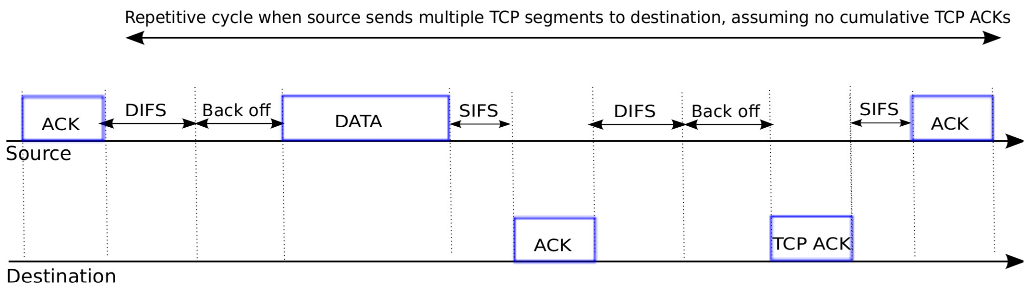

Repetitive cycle when source sends multiple TCP segments to destination, assuming no cumulative TCP ACKs
ACK
DIFS
Back off
DATA
SIFS
DIFS
Back off
SIFS
ACK
Source
ACK
TCP ACK
Destination

Figure 4.4: 802.11a/b/g MAC overhead per TCP segment transmission.

network topology and wireless channel characteristics. As per Eq. (4.3), however, the distributed buffer B should be sized higher than these instantaneously precise values. This is necessary to account for the time-varying capacity of the wireless channel, and non-deterministic IEEE 802.11 Distributed Coordination Function (DCF) scheduling. In this work the loose upper bounds have been estimated on λ and RTT for a given network. The resulting larger-than optimal buffer trades-off queueing delay for higher channel utilization. Experimental results in Sec. 4.4.2 confirm that the impact of this trade-off is minimal, while maintaining buffers as small as 2-3 packets at most mesh nodes.

For a given flow, the upper bound on λ is obtained when there is no competing traffic. Let the wireless link rates for the M nodes in the collision domain be represented by the vector $\mathbf{W} = \{w_1, w_2, ..., w_M\}$. The bottleneck bandwidth of the path is determined by $w_{min} = \min(w_1, w_2, ..., w_M)$. Let $\lambda = w_{min}$ be a loose upper bound on the network carrying capacity.

The RTT in (4.3) should only include the propagation delay, and not the queueing delays, for the wireless links. The propagation delays are dominated by transmission delays in wireless networks. Where applicable, other delays, such as those introduced by access links bridging to the wired Internet (*e.g.* DSL), may also need to be included. To compute the wireless transmission delays, consider the MAC overhead for transmitting a TCP segment and its associated ACK in Fig. 4.4. Let T_{d-DATA} and T_{d-ACK} represent the *total* transmission time for TCP segment and ACK, respectively.

Parameter	Value
T_{slot}	20 μs
T_{SIFS}	10 μs
T_{DIFS}	50 μs
CW_{min}	31
CW_{max}	1023

Table 4.1: System parameters of IEEE 802.11b [2].

$$T_{d-DATA} = T_{BO} + T_{DIFS} + T_{DATA} + T_{SIFS} + T_{ACK} \tag{4.4}$$

$$T_{d-ACK} = T_{BO} + T_{DIFS} + T_{TCP-ACK} + T_{SIFS} + T_{ACK} \tag{4.5}$$

where T_{BO} is the backoff interval. The average T_{BO}, $\overline{T_{BO}} = (CW_{min} - 1) \times T_{slot}/2$, where CW_{min} is the minimum MAC contention window and T_{slot} is the slot duration. T_{SIFS} and T_{DIFS} are the Shortest Inter-Frame Space (SIFS) and the Distributed Inter-Frame Space (DIFS) deferral, respectively. T_{DATA}, $T_{TCP-ACK}$, and T_{ACK} are the transmission times to transmit a TCP segment, ACK, and MAC-level ACK, respectively. Based on the configured RTS threshold, additional time of $T_{RTS} + T_{CTS} + 2 \times T_{SIFS}$ may be added to T_{d-DATA} as well as T_{d-ACK}, where T_{RTS} and T_{CTS} is the time to transmit an RTS and a CTS frame, respectively. Various IEEE 802.11b system parameters are listed in Table 4.1. For transmitting a TCP segment of size 1460 B and its ACK as per the exchange shown in Fig. 4.4, it takes approximately 15 ms at 1 Mb/s link rate and 2.7 ms at 11 Mb/s link rate.

RTT through the bottleneck collision domain can then be computed as follows:

$$\begin{aligned} RTT &= \sum_{i=1}^{M} T_{d-DATA} + T_{d-ACK} \\ &= M \cdot (T_{d-DATA} + T_{d-ACK}) \end{aligned} \tag{4.6}$$

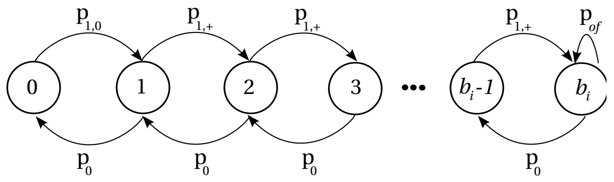


Figure 4.5: Queue occupancy state transition for a wireless node n_i with a buffer of size b_i.

4.3.2 Distributing the neighborhood buffer among nodes

Once the neighborhood buffer size B is computed, it will be distributed amongst the set of nodes in the bottleneck collision domain. One simple strategy is to divide B equally amongst the nodes. However, this does not consider the fact that a queue drop closer to the source has consumed fewer network resources than a queue drop near the destination. The proposed model uses a cost function to capture this bias.

Fig. 4.5 shows the queue occupancy state transition for a mesh node. A node n_i can queue at most b_i packets corresponding to its allocated buffer. Let π_k represent the steady state probability that the node queue size is k, for $0 \leq k \leq b_i$. In particular, π_0 and π_{b_i} represent the probabilities that the buffer is empty or full, respectively. The node successfully transmits a packet with probability p_0, transitioning to a state with one less packet queued. With probability $p_{1,+}$, a relay node receives a packet from its parent in the routing tree, and then queues it locally for transmission to the next hop. The transition probability $p_{1,0}$ is a special case for packet reception when the current queue size is zero. Finally, probability p_{of} represents a packet drop due to buffer overflow.

A node can transmit a packet with probability 1 if all other nodes have an empty buffer (*i.e.,* they are in state π_0); else it contends fairly for the channel with other nodes that also have packet(s) buffered for transmission. Thus,

$$\begin{aligned}
p_0 &= 1 \cdot \pi_0^{M-1} + \frac{1}{2} \cdot \binom{M-1}{1} \cdot \pi_0^{M-2}(1-\pi_0) \\
&+ \frac{1}{3} \cdot \binom{M-1}{2} \cdot \pi_0^{M-3}(1-\pi_0)^2 + \ \ldots \ + \\
&+ \frac{1}{M} \cdot \binom{M-1}{M-1} \cdot (1-\pi_0)^{M-1} \\
&= \sum_{i=1}^{M} \frac{1}{i} \cdot \binom{M-1}{i-1} \cdot \pi_0^{M-i} \cdot (1-\pi_0)^{i-1} \qquad (4.7)
\end{aligned}$$

The probability of receiving a packet from a parent is 0 if the parent buffer is empty. If it is not empty (*i.e.,* in state $1-\pi_0$), the parent contends fairly with the other nodes that also have packet(s) buffered for transmission.

$$\begin{aligned}
p_{1,0} &= 0 \cdot \pi_0 + (1-\pi_0)\left[1 \cdot \pi_0^{M-2} + \frac{1}{2} \cdot \binom{M-2}{1} \cdot \pi_0^{M-3}(1-\pi_0)\right. \\
&+ \frac{1}{3} \cdot \binom{M-2}{2} \cdot \pi_0^{M-2}(1-\pi_0)^2 + \ \ldots \ + \\
&+ \left.\frac{1}{M-1} \cdot \binom{M-2}{M-2} \cdot (1-\pi_0)^{M-2}\right] \\
&= (1-\pi_0)\sum_{i=1}^{M-1} \frac{1}{i} \cdot \binom{M-2}{i-1} \cdot \pi_0^{M-1-i} \cdot (1-\pi_0)^{i-1} \qquad (4.8)
\end{aligned}$$

If the child queue is not empty, it always contends for transmission with its parent. Thus $p_{1,+}$ is computed as follows:

$$\begin{aligned}
p_{1,+} &= 0 \cdot \pi_0 + (1-\pi_0)\left[\frac{1}{2} \cdot \pi_0^{M-2} + \frac{1}{3} \cdot \binom{M-2}{1} \cdot \pi_0^{M-3}(1-\pi_0)\right. \\
&+ \frac{1}{4} \cdot \binom{M-2}{2} \cdot \pi_0^{M-4}(1-\pi_0)^2 + \ \ldots \ + \\
&+ \left.\frac{1}{M} \cdot \binom{M-2}{M-2} \cdot (1-\pi_0)^{M-2}\right] \\
&= (1-\pi_0)\sum_{i=1}^{M-1} \frac{1}{i+1} \cdot \binom{M-2}{i-1} \cdot \pi_0^{M-1-i}(1-\pi_0)^{i-1} \qquad (4.9)
\end{aligned}$$

The buffer at a node overflows when a node in state π_{b_i} receives a packet. Thus,

$$p_{of} = \pi_{b_i} \cdot p_{1,+} \tag{4.10}$$

The cost associated with this drop due to network resources already consumed by this packet increase with the hop distance of n_i from the source along the routing path, *i.e.,* packet dropped closer to source wastes fewer network resources compared to a packet dropped closer to destination. Assume the nodes n_i are ordered such that they represent an increasing hop distance from the source, *i.e.,* the hop count from the source for $n_i < n_{i+1}$. Thus, the index i could be used for $i = \{1, 2, ..., M\}$ as a simple cost function to represent the bias associated with a packet drop at node n_i.

Thus the optimization problem can be formulated as follows:

$$\begin{aligned} \min_{b_i} \quad & \sum_{i=1}^{M} \pi_{b_i} \cdot p_{1,+} \cdot i \\ \text{subject to} \quad & \textstyle\sum_{i=1}^{M} b_i = B \\ \text{and} \quad & b_i \geq 0, \forall i \in M \end{aligned} \tag{4.11}$$

The steady state flow balance leads to the following:

$$\begin{aligned} \pi_0 \cdot p_{1,0} &= \pi_1 \cdot p_0 \\ \pi_1 \cdot p_{1,+} &= \pi_2 \cdot p_0 \\ &\cdots \\ \pi_{b_{i-1}} \cdot p_{1,+} &= \pi_{b_i} \cdot p_0 \end{aligned} \tag{4.12}$$

Therefore,

$$\pi_k = \pi_0 \left(\frac{p_{1,+}}{p_0}\right)^{k-1} \cdot \frac{p_{1,0}}{p_0}$$

when $k > 0$. Substituting this in $\sum_{k=0}^{b_i} \pi_k = 1$, π_0 can be calculated since $p_{1,0}, p_{1,+}$, and p_0 can also be represented by π_0.

The above analysis is based on the assumption that all nodes experience identical transition probabilities. This assumption holds, irrespective of the buffer b_i allocated to node n_i, as long as the probability of a node buffer being empty is small, *i.e.,* $\pi_0 \approx 0$. This allows p_0, $p_{1,0}$, and $p_{1,+}$ from (4.7), (4.8), and (4.9) to converge to $\frac{1}{M}$ (for simplicity, $p_{1,0}$ is approximate from $\frac{1}{M-1}$ as $\frac{1}{M}$). This behavior is consistent with the IEEE 802.11 MAC that provides an equal Transmission Opportunity (TXOP) to all nodes in a collision domain.

From Eq. (4.12) it could be observed that $\pi_{b_i} = \frac{1}{b_i+1}$. Substituting into Eq. (4.10),

$$p_{of} = \pi_{b_i} \cdot p_{1,+} = \frac{1}{b_i + 1} \cdot \frac{1}{M} \tag{4.13}$$

The optimization problem then becomes

$$\min_{b_i} \sum_{i=1}^{M} \frac{1}{b_i + 1} \cdot \frac{1}{M} \cdot i \tag{4.14}$$

$$\text{subject to} \quad \textstyle\sum_{i=1}^{M} b_i = B$$

$$\text{and} \quad b_i \geq 0, \forall i \in \mathbf{M} \tag{4.15}$$

The objective function in (4.14) can be expanded as follows:

$$\frac{1}{M} \min_{b_1, b_2, \ldots, b_M} \left(\frac{1}{b_1 + 1} + \frac{2}{b_2 + 1} + \ldots + \frac{M}{b_M + 1} \right)$$

Let

$$f \triangleq \left(\frac{1}{b_1 + 1} + \frac{2}{b_2 + 1} + \ldots + \frac{M}{b_M + 1} \right) \tag{4.16}$$

From analytical algebra, the optimal b_i satisfies

$$\frac{\partial f}{\partial b_i} = 0, \forall i \in \mathbf{M}, \tag{4.17}$$

or the optimal b_i are the boundary values, *i.e.,* $b_i = 0$ or B. Using substitution, it could be verified that the boundary values do not minimize the cost function f. The last equation, (4.17), is composed of a set of M linear equations with M unknowns whose solution is as follows:

$$\begin{aligned} b_1 &= \frac{B+5}{1+\sqrt{2}+\sqrt{3}+\ldots+\sqrt{M}} - 1 \\ b_2 &= \frac{\sqrt{2}(B+5)}{1+\sqrt{2}+\sqrt{3}+\ldots+\sqrt{M}} - 1 \\ \ldots & \\ b_M &= \frac{\sqrt{M}(B+5)}{1+\sqrt{2}+\sqrt{3}+\ldots+\sqrt{M}} - 1 \end{aligned} \tag{4.18}$$

When B is large, *i.e.,* $B >> 1$, this could be approximated as:

$$b_1 : b_2 : \cdots : b_M = 1 : \sqrt{2} : \ldots : \sqrt{M} \tag{4.19}$$

It could be noted that the results in (4.19) hold for any cost function that increases linearly with the hop count. We believe that our proposed cost function should be viewed as an example of a buffer allocation for a specific interference model and a cost function that we considered. In fact, the core contribution of the dissertation is more general: the distributed bottleneck in a wireless network should consider the cumulative buffer size of contending nodes to balance throughput and delay trade-off. It's possible to come up with a more accurate albeit complicated interference model and cost function, but the underlying contribution of the thesis is more general and would still apply.

4.4 Performance Analysis

DNB is evaluated using both simulation and real testbed implementation.

4.4.1 Simulations

DNB performance results have been benchmarked against the following: (1) A 50-packets buffer at all mesh routers. This is the default NS-2 configuration commonly used in research publications; (2) TCP-AP [72], a TCP pacing mechanism for multi-hop wireless networks (see Sec. 2.3 for a brief description of TCP-AP). Pacing TCP traffic minimizes bursts that lead to large queueing delays. TCP-AP paces its traffic using 4-hop propagation delay, thus minimizing queueing along the interfering links. Consistent with [72], a 50-packets buffer for TCP-AP has been used, though we find that queues hardly build up beyond a few packets. A large file transfer is used to simulate backlogged TCP traffic. The TCP segment size used in this simulations is 1460 bytes. Wireless link rates are 11 Mb/s, unless specified otherwise. RTS/CTS control frames are disabled.

Single flow topologies

First, two variants of a 4-hop chain are analyzed: (1) when all nodes are within mutual carrier sense range, and (2) when the edge nodes are outside carrier sense range. The two scenarios experience different sources of packet loss: queue overflows in the first topology (as most collisions are resolved by MAC-layer ARQs) and collisions due to asymmetric information about the channel state [66] in the latter. The neighborhood buffer size B computed per (4.3) depends on the wireless link rates. For the testbed experimentation, $B \approx 15$ packets for 54 Mb/s links (IEEE 802.11a/g) or $B \approx 85$ packets for 600 Mb/s links (IEEE 802.11n). For 11 Mb/s links (IEEE 802.11b) used in the simulations, $B \approx 10$ packets. Per (4.19), this is distributed as buffer of size

Scheme	Goodput Kb/s	Mean RTT (Std. Dev.) ms
50 pkts buffer	930	316 (68)
TCP-AP	850	15 (0.5)
Buffer per (4.3) & (4.19)	878	22 (2)

Table 4.2: Performance comparison for a 4-hop TCP flow with 11 Mb/s wireless links. All nodes are within mutual carrier sense range.

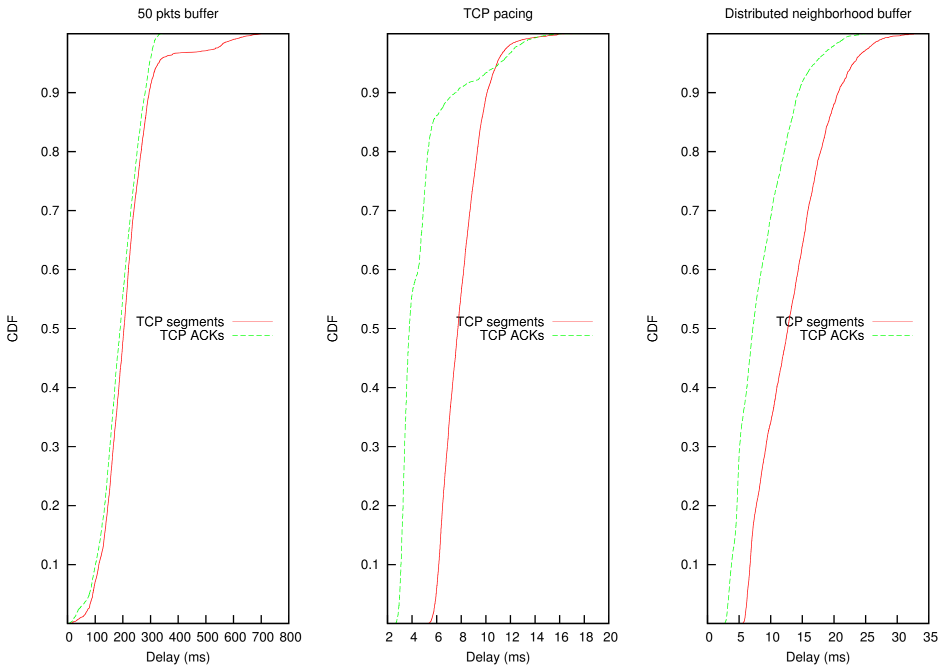


Figure 4.6: Delay distribution CDF for the parking lot topology.

2, 2, 3, and 3 packets at 1, 2, 3, and 4-hop nodes from the source, respectively. For symmetric wireless links, the performance for upload flows is similar to that obtained for download flows between the gateway and a given mesh node.

Simulation results are summarized in Tables 4.2 and 4.3. Also, the Cumulative Distribution Function (CDF) plots for the delay distribution are shown in Fig. 4.6. These results confirm that the base case with a 50-packets buffer yields the highest aggregate throughput, albeit with large RTT delays. This may be immaterial for large file transfers where the objective is to maximize throughput. Regardless, a network

Scheme	Goodput Kb/s	Mean RTT (Std. Dev.) ms
50 pkts buffer	910	200 (49)
TCP-AP	825	15 (0.5)
Buffer per (4.3) & (4.19)	830	23 (1.3)

Table 4.3: Performance comparison for a 4-hop TCP flow with 11 Mb/s wireless links. The two end nodes outside mutual carrier sense range.

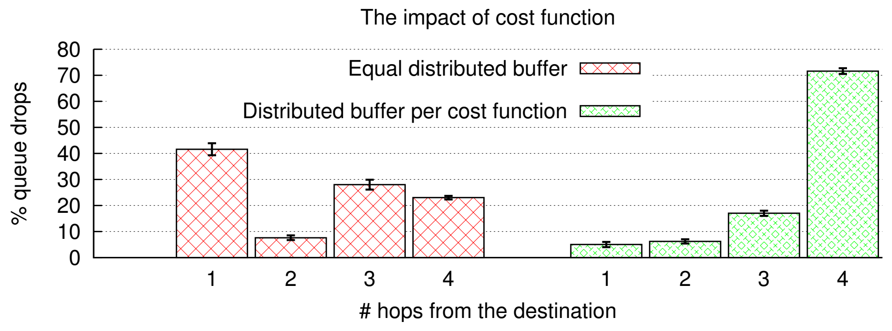


Figure 4.7: Queue drops at mesh routers in a 4-hop chain.

should be able to provide low-delay services, especially when TCP flows share buffers with real-time flows. DNB lowers the base case RTT by a factor of 15 and 9 across the two topologies, at the cost of about 6% to 9% drop in throughput. TCP-AP achieves the lowest delay among the three schemes. In general, however, there are practical limitations of deploying pacing in real networks: First, while WMN service providers can configure the buffers on their mesh routers, but they often have no control on the TCP/IP stack on the client devices. Second, paced flows fare poorly when sharing a network with non-paced flows [52], thus limiting the incentive for a user to migrate. We evaluate this further for multi-flow topologies in Sec. 4.4.1 below.

Further, the performance improvements of the proposed cost function is analyzed. Using a 4-hop chain topology, a 12-packet neighborhood buffer has been distributed equally among the relay nodes (a 3-packet buffer at each relay) and compare its performance to a buffer distribution using the proposed cost function. The experiment have been repeated 5 times; the average queue drops (with the standard deviation)

Scheme	Normalized Goodput	Normalized RTT
50 pkts buffer	1	20.3
TCP-AP	0.90	1
Buffer per (4.3) & (4.19)	0.96	2.2

Table 4.4: Performance statistics averaged over multiple topologies with varying hop count. Goodput results are normalized to the goodput achieved with the base case for a 50 packet buffer for that simulation. RTT results are normalized to the RTT measured with TCP pacing for that simulation. Averages computed over multiple different topologies.

at each node is shown in Fig. 4.7. With equal-sized buffers, up to 42% queue drops happen at the node 1-hop away from the destination. In contrast, with the proposed cost function, over 70% of the queue drops occur at the node closest to the source. In this set of simulations, the proposed scheme further reduced RTT by about 10% with a slight improvement in throughput of 2-3%.

The proposed buffer sizing approach has been validated on multiple other topologies by varying the flow hop count, including topologies where packet losses occur due to both buffer overflows as well as collisions. In all simulations, the base case with the 50-packets buffer yielded the highest goodput, while TCP-AP yielded the least delay. The goodput and RTT for a given simulation run have been normalized to these values, and then average those up over different topologies. Results are tabulated in Table 4.4. In general, DNB achieves about 96% goodput of the base case with a 50-packets buffer, while reducing the RTT by a factor of 10.

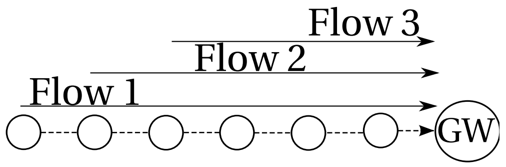
Multi-flow topologies

The problem formulation in Sec. 4.3 considers a single TCP stream. Let us investigate if the underlying behavior is also sustainable in a multi-flow network. The motive here is not to determine the optimal cumulative neighborhood buffer in multi-flow environments; instead, the goal is to evaluate whether small buffers also work well with multiple flows. Two illustrative scenarios have been considered: (1) a parking lot topology where multiple flows pass through shared relay nodes, and (2) a cross topology when multiple flows share the bottleneck spectrum but not the relay nodes. Each flow in the performed experiments represents either (i) a large file transfer with TCP, or (ii) a bundle containing a large file transfer with TCP *and* a VoIP stream. VoIP streams has been simulated with G.729 codec characteristics using Constant Bit Rate (CBR) UDP traffic: a stream bit rate of 8 Kb/s with a packet size of 40 bytes each. For toll quality service, VoIP streams should maintain a one-way latency (mouth to ear) of less than 150 ms and a delay jitter of less than 30 ms [73].

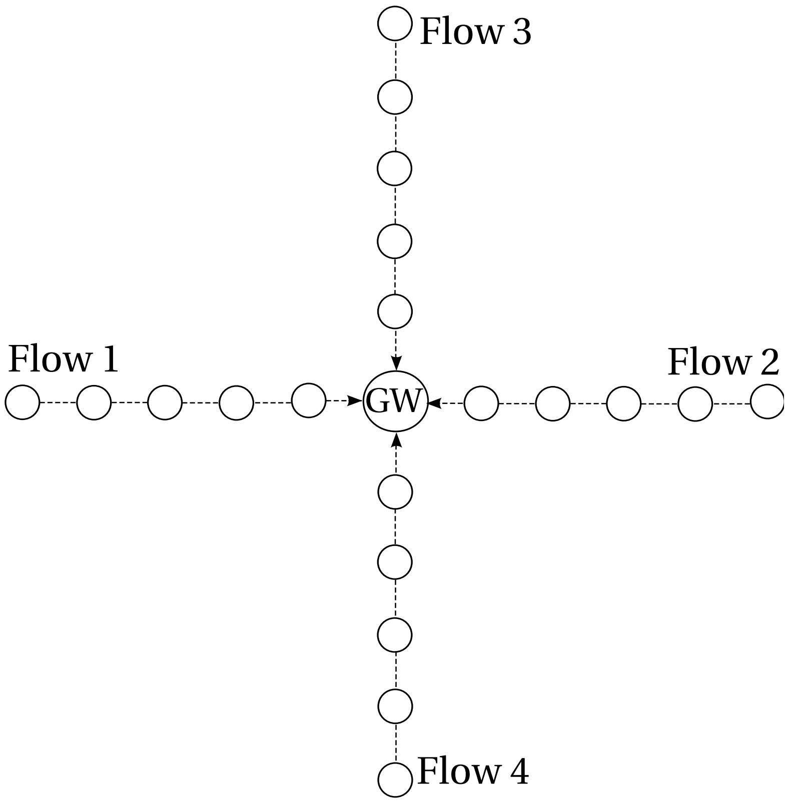
Parking lot topology

The first set of experiments have been done with the parking lot topology illustrated in Fig. 4.8a. It has data flows originating from nodes 4, 5, and 6-hop away from the gateway. These flows share a common bottleneck collision domain. Considering each flow in isolation, The proposed buffer sizing strategy in Sec. 4.3 can be used to compute a neighborhood buffer size B_1, B_2, and B_3 corresponding to each flow. Then, these buffers can be distributed such that a mesh node in the collision domain has a *per-flow* buffer of size determined by (4.19). Per-flow queueing requires flow classification at relay nodes; it has performed using source node IP address and port numbers. These queues are using fair queueing [74] on top of DCF scheduling. DNB considerably overestimates the cumulative buffer required for saturating the bottleneck spectrum; however, the per-flow buffer (of 1–3 packets) at each relay node is still small. Determining the optimal buffer size in such multi-flow networks is left for future work.

The results for the three large file transfer streams are summarized in Table 4.5. The mean and the standard deviation are listed for goodput and RTT averaged over the three flows. TCP pacing lowers RTT by a factor of 9 at a cost of 8%



(a) Parking lot topology



(b) Cross topology

Figure 4.8: Multi-flow topologies

drop in goodput. In contrast, DNB improves RTT by a factor of 6 at the cost of less than 4% drop in average goodput. These improvements may appear relatively modest compared to the results for networks with single TCP streams. This is partly because multiple streams sharing a 50 packet buffer do not let TCP congestion window for individual streams grow quickly to unsustainable large values. Finally, DNB improvements with distributed buffers are obtained despite the suboptimal (though small) buffer sizes used in this experiment.

As mentioned earlier, paced TCP traffic fares poorly when competing with non-

Scheme	Avg. Goodput (Std. Dev.) Kb/s	Mean RTT (Std. Dev.) ms.
50 pkts buffer	285 (68)	354 (35)
TCP-AP	264 (32)	38 (3)
Buffer per (4.3) & (4.19)	275 (22)	58 (3)

Table 4.5: Parking lot topology. 3 FTP streams.

Scheme	FTP		VoIP	
	Mean Goodput (Std. Dev.) Kb/s	Mean RTT (Std. Dev.) ms.	Mean Bit Rate Kb/s	Mean Delay (Jitter) ms.
50 pkts buffer	261 (58)	388 (32)	7.8	239 (8)
TCP-AP	240 (17)	54 (6)	8	37 (5)
Buffer per (4.3) & (4.19)	250 (33)	87 (6)	8	40 (6)

Table 4.6: Parking lot topology. 3 FTP and 3 VoIP streams.

paced TCP streams. This has been highlighted using a TCP-AP stream for the 4-hop flow and non-paced TCP for the 5 and 6-hop flows in Fig. 4.8a. Simulation results show that the paced flow achieves less than $\frac{1}{10}$th of the throughput of the other flows. No such unfairness is observed when none of the three flows are paced. Note that this behavior is very different from the unfairness observed in prior work [66], as it is the stream *closest* to the gateway that is losing throughput. This performance challenge is an impediment in wide-spread deployment of pacing-based protocols.

Table 4.6 shows the results when each flow in Fig. 4.8a represents a bundle containing a large file transfer with TCP and a VoIP stream. For the VoIP traffic, the mean bit rate (stream goodput), one way delay, and delay jitter averaged over the three streams are listed. With large network buffers, TCP streams can quickly build a large congestion window. This increases the queueing delay for the VoIP traffic that shares the buffer with the TCP streams. With 50 packet buffers, the average delay for the three VoIP streams far exceeds the 150 ms delay bound. In contrast, both TCP-AP and our proposed distributed buffer sizing mechanism limit this unchecked

Scheme	Mean Goodput (Std. Dev.) Kb/s	Mean RTT (Std. Dev.) ms.
50 pkts buffer	465 (44)	260 (53)
TCP-AP	417 (24)	28 (5)
Buffer per (4.3) & (4.19)	455 (27)	56 (13)

Table 4.7: Cross topology. 4 FTP streams.

Scheme	FTP		VoIP	
	Mean Goodput (Std. Dev.) Kb/s	Mean RTT (Std. Dev.) ms.	Mean Bit Rate Kb/s	Mean Delay (Jitter) ms.
50 pkts buffer	382 (17)	309 (72)	7.8	187 (30)
TCP-AP	339 (15)	33 (6)	7.9	24 (4)
Buffer per (4.3) & (4.19)	368 (5)	71 (8)	7.9	35 (4)

Table 4.8: Cross topology. 4 FTP and 4 VoIP streams.

growth of the TCP congestion window. TCP-AP throttles the TCP streams more aggressively, roughly incurring a loss of 5% in goodput compared to our proposed buffer sizing mechanism. As a final observation, we note that the combined goodput of the FTP and VoIP streams in Table 4.6 is less than the corresponding entries in Table 4.5. This is because the IEEE 802.11 overhead is better amortized over the larger 1460 byte TCP segments used in the FTP simulations. Various optimizations for reducing this overhead have been discussed in prior work [75].

Cross topology

Next, the cross topology has been considered in Fig. 4.8b. A source node 5-hops along each edge sends a data flow to the gateway in the center. Using a 2-hop interference model, the 1 and 2-hop nodes around the gateway along each edge constitute the shared bottleneck collision domain between the four flows. With uniform 11 Mb/s links, this distributed buffer size is $B = 18$ packets per (4.3). Since the number of hops (and consequently the cost of packet drop) are similar for each edge, B should be distributed as 2 and 3 packets at the 3 and 4-hop nodes from source along each edge.

Simulation results for the FTP streams are summarized in Table 4.7. The mean and the standard deviation for the four flows are listed. TCP pacing lowers RTT 10-fold at a cost of 10% drop in goodput. DNB improves RTT by 5$\times$ with 2% drop in average goodput of the four flows.

Table 4.8 shows the simulation results when each flow in Fig. 4.8b represents a bundle containing a large file transfer with TCP and a VoIP stream. Consistent with the previous observations for the parking lot topology, TCP fills up available network buffers, leading to unacceptable VoIP delays when large network buffers are shared with the TCP stream. Right sizing the network buffers per our proposed mechanism reduces the VoIP delays by more than a factor of 5, at a cost of less than 3% loss in goodput for the TCP streams.

4.4.2 Testbed

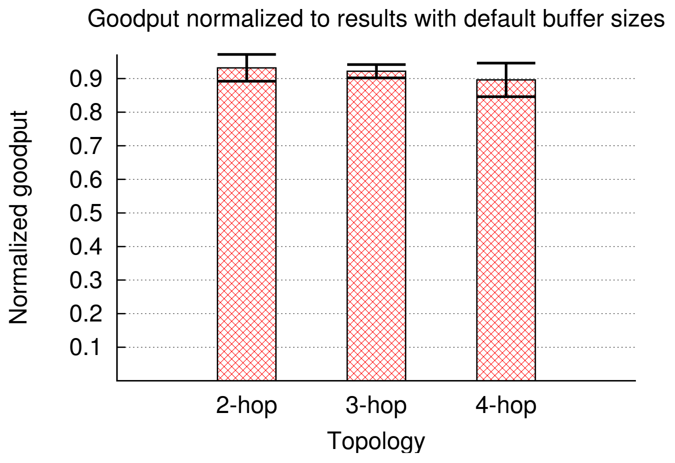
DNB has also been evaluated using our IEEE 802.11a/b/g testbed over multiple scenarios. The testbed hardware setup and software parameters are described in details in Sec. 3.1. For comparative analysis, the performance results have been benchmarked with the default buffer sizes used in our Linux distribution. In the first set of experiments, the hop length of a path has been varied from 2 to 4 hops. These experiments were repeated along various source and destination pairs across the testbed to offset any location-specific wireless idiosyncrasies. For a 3-hop chain and 11 Mb/s wireless links, the neighborhood buffer size B computed from (4.3) is approximately 8 packets. As per (4.19), this is distributed as a txqueue buffer of size 2, 3, and 3 packets at 1, 2, and 3-hop nodes from the source, respectively. The buffer sizes for other topologies were similarly calculated. Testbed results are shown in Fig. 4.9. The goodput measured with DNB is normalized to the goodput for the same topology with default buffer sizes in the Linux distribution. For RTT, the

Scheme	Avg. Goodput (Std. Dev.) Kb/s	Mean RTT (Std. Dev.) ms.
Default, large buffers	786 (45)	1653 (327)
Buffer per (4.3) & (4.19)	712 (6)	91 (19)

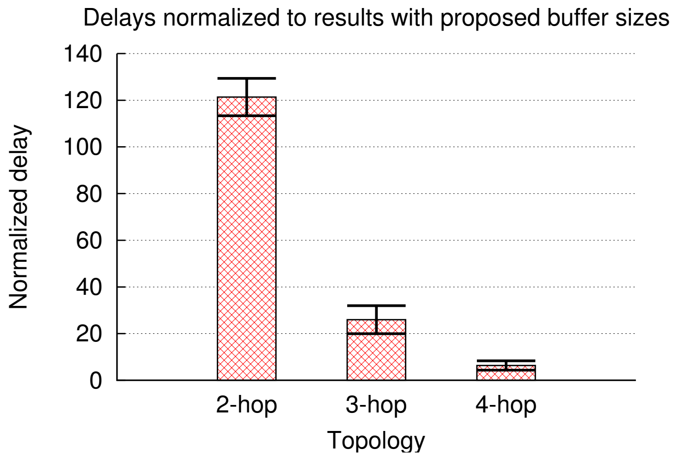
Table 4.9: Multi-flow experimental results. A 3-hop and a 4-hop flow together share the network along disjoint paths.

delays with large buffers is normalized to the delay values measured with DNB. The error bars in these graphs are the 95% confidence intervals. Testbed results show a two-orders of magnitude improvement in two-hop and 25× improvement in three-hop delays while obtaining upwards of 92% goodput achieved with large buffers in both cases. The results with a 4-hop chain show a 6× improvement in delay while achieving 90% of the large-buffer goodput. These results show that wireless specific losses are minimal in smaller topologies (thanks to MAC-level retransmissions in IEEE 802.11) where most nodes can carrier sense each other, and hence TCP congestion avoidance algorithms mostly reacts to losses when large buffers overflow. In contrast, in larger topologies wireless losses tend to dominate and prevent TCP congestion window from completely filling up intermediate router buffers. However, the window still grows to suboptimal large values that increases queueing delays beyond those acceptable for real-time traffic. To give an example, the average RTT with large buffers in our 4-hop topology is 431 ms as compared to 67 ms with DNB.

Now it is time to investigate if the underlying performance improvements are also sustainable in a multi-flow network using small buffers. The next set of experiments considers a 3-hop and a 4-hop flow in our 10-node IEEE 802.11a/b/g testbed. The two flows traverse disparate (though interfering) nodes along their path to the destination. Considering each flow in isolation, the buffer sizing strategy in Sec. 4.3 can be used to compute a neighborhood buffer size B_1 and B_2 corresponding to each flow. Then distribute these buffers such that a mesh node in the collision domain has a per-flow buffer of size determined by (4.19).



(a) Flow goodput



(b) RTT

Figure 4.9: Flow goodput and delay across various testbed topologies. Goodput results are normalized with default buffer sizes to the goodput achieved with DNB. RTT measurements are normalized with default buffers to the RTT measured with DNB. Error bars are the 95% confidence intervals.

Testbed results are summarized in Table 4.9. The mean goodput and the RTT for the two flows, as well as their standard deviation are listed. IEEE DNB reduces the average RTT of the two flows by a factor of 20, at the cost of about 9% drop in aggregate goodput.

Finally, the performance of DNB is also validated using our IEEE 802.11n testbed. In this set of experiments, TCP download flows (file transfer from the server to the wireless client) are used over a 4-hop chain topology. Our results are summarized in

Scheme	Avg. Goodput (Std. Dev.) Mb/s	Mean RTT (Std. Dev.) ms.
Default, large buffers	8.8 (0.3)	1113 (91)
Buffer per (4.3) & (4.19)	8.1 (0.04)	153 (9)

Table 4.10: A 4-hop 802.11n mesh topology connected via 65 Mb/s links.

Num. of hops	Equal Buffers		Cost Function Buffers	
	Goodput (Mb/s)	RTT (ms)	Goodput (Mb/s)	RTT (ms)
1-hop	61.76	16.628	69.05	14.128
2-hops	28.14	42.55	29.45	39.674
3-hops	17.02	69.47	17.53	64.077

Table 4.11: Evaluating goodput and RTT for the proposed buffer distribution scheme over various number of hops

Table 4.10. With 65 Mb/s links, the default buffer size in Linux yields an average throughput of 8.8 Mb/s with a mean RTT of 1113 s over the 4-hop topology. With DNB, the neighborhood buffer size is approximately 14 packets. Using the linear cost function in Sec. 4.3 to distribute the buffers over the bottleneck, DNB reduces RTT by 7× at the cost of about 8% drop in flow throughput. Using the same testbed, we also evaluate the performance of our proposed method to distribute the optimal buffer over nodes in the bottleneck collision domain. We enable the rate control algorithm in the testbed and repeat the same experiment twice. In the first run, the optimal buffer is distributed equally among the nodes in the neighborhood whereas the buffer is distributed according to our cost function as per (4.19) in the second run. Table 4.11 shows the goodput and RTT of the two schemes over an increasing hop count. It is clear from these results that the proposed methodology in distributing the buffer achieves higher goodput and lower delay compared to the equal distribution method over all scenarios. This is because our scheme favours dropping packets near to the source by assigning slightly bigger buffers to nodes closer to the destination.

Chapter 5

Aggregation-Aware Queue Management

5.1 Overview

The IEEE 802.11n standard specifications introduced MAC-layer frame aggregation to improve network performance. Using A-MPDU, a wireless node can transmit up to 64 MPDUs or sub-frames. The aggregation logic or scheduler is left open to the vendor's implementation. Always transmitting a maximum sized A-MPDU may maximize throughput, but this will increase delays if a node needs to wait to assemble 64 MPDUs from higher layers. In contrast, the current Linux implementation sends as many frames as currently available in the buffers, resulting in A-MPDUs with variable frame sizes. This variability in frame size poses a new challenge to accurately estimate the queue draining time based on the current transmission rate. Other enhancements in IEEE 802.11n, such as channel bonding and MIMO streams, allow IEEE 802.11n radios to operate at link rates as high as 600 Mb/s. Thus, there is a huge variation in the queue draining time between the highest and lowest possible rates. For example, assume a single sender and receiver, both configured with the default Linux buffer size of 1000 packets. The 600 Mb/s link needs only 20 ms to empty the buffer; however, this buffer drain time is two orders of magnitude higher for a 6.5 Mb/s link.

To address these challenges, we have designed and implemented WQM, an aggregation-aware queue management scheme for wireless networks. WQM identifies and distinguishes between 'good' and 'bad' buffers. Good buffers are buffers needed to absorb bursty traffic, while bad buffers only contribute to network latency without any noticeable improvement in throughput. WQM is both practical and incrementally deployable; it uses existing data traffic as probe for network measurements and does not incur any additional overhead. To account for channel variability, WQM periodically recalculates the time needed to drain the buffer based on the current transmission rate. It then adjusts the buffer size to maintain the network QoS guarantees, reducing queueing delays where necessary, while allowing sufficient buffers to saturate available network capacity. Further, WQM incorporates MAC behavior, such as A-MPDU frame aggregation, to get accurate estimates of queue draining time. WQM has been implemented and evaluated in our IEEE 802.11n testbed and the experimental results in Sec. 5.4 show that it reduces the end-to-end delay by by an order of magnitude compared to the default buffer size used in stock Linux. Moreover, it reduces end-to-end latency by up to 7$\times$ compared to state of the art bufferbloat solutions in the literature. In the worst case, this reduction comes at the cost of 8% drop in goodput.

To the best of our knowledge, this is the first attempt to address the queue sizing problem for IEEE 802.11n networks. Compared to other efforts in the wired and the wireless domain, WQM is considered more practical as it does not involve time-stamping the packets at their arrival to the queue. It also accounts for the MAC-layer behavior when deciding about the optimal buffer size.

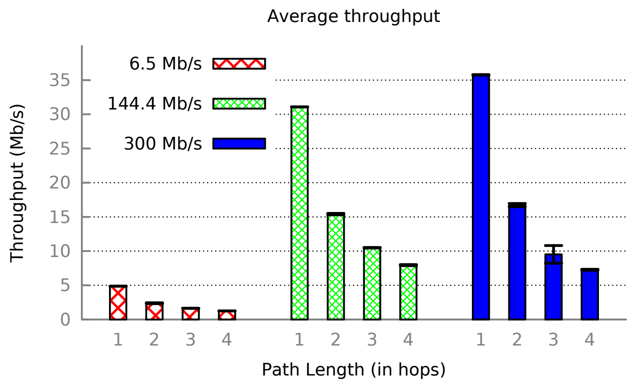
5.2 Motivation

To motivate WQM, we have evaluated the impact of A-MPDU frame aggregation on bufferbloat over multiple fixed link rates using our IEEE 802.11n wireless testbed deployed in our campus. The detailed experimental setup is illustrated in Sec. 3.1. The maximum aggregate size achievable in practice in ath9k is limited to a frame duration of 4 ms to comply with regulatory requirements for operation in the 5 GHz U-NII band. Thus the actual density of MPDUs in an A-MPDU is partly dependent on the wireless link rate which determines the frame transmit duration. Experiments both with and without A-MPDU aggregation have been conducted to understand the impact of aggregation on wireless network performance. Each experiment for a given link rate and network topology is performed 5 times. The throughput and RTT results are then averaged across these runs. Three link rates have been used for these experiments: 6.5 Mb/s (Modulation and Coding Scheme (MCS) index 0, 20 MHz channel bandwidth, 800 ns Guard Interval (GI)), 144.4 Mb/s (MCS 15, 20 MHz channel, 400 ns GI), and 300 Mb/s (MCS 15, 40 MHz channel, 400 ns GI). This configuration parameter set allows us to experiment with varying channel bandwidth as well as GI values.

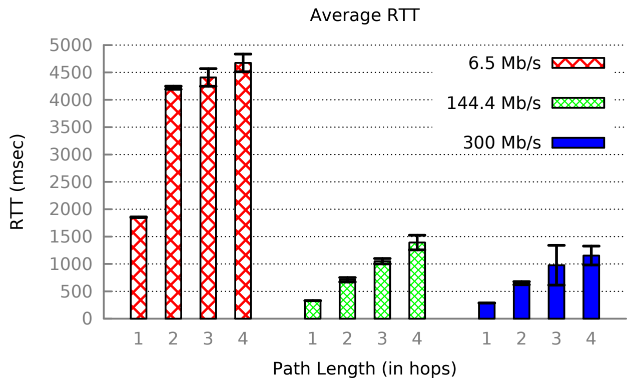
Single-flow topologies

In the first set of experiments, transmit A-MPDU aggregation has been disabled. Then, the path length of the flow has been varied from a single-hop network to a 4-hop chain topology. A given experiment uses a uniform wireless link rate between adjoining nodes. This rate is varied from 6.5 Mb/s, 144.4 Mb/s, and 300 Mb/s in different experiments. Throughput and delay for various hop-counts are shown in Figs. 5.1a and 5.1b respectively. The error bars represent the standard deviation.

As expected, throughput increases with the link rate, and shows a decreasing trend with the hop-count. The throughput drops by $\frac{1}{2}$, $\frac{1}{3}$, and $\frac{1}{4}$, for 2, 3, and 4-hop networks, suggesting sparse spatial reuse in our topologies. The difference in throughput between 144.4 Mb/s links and 300 Mb/s links is small, and almost within error bounds of the measurements for 3 and 4-hop topologies. The RTT delay



(a) Throughput



(b) RTT

Figure 5.1: Flow throughput and RTT for wireless links without A-MPDU aggregation.

measurements clearly show the impact of bufferbloat. The average 1-hop RTT delay measurements are 1853 ms, 328 ms, and 286 ms for 6.5 Mb/s, 144.4 Mb/s, and 300 Mbp/s links. These delays show that a single file-transfer between two wireless nodes can saturate the device buffers, even at the 300 Mb/s link rate, as shown in Fig. 5.2. It could be observed that the buffer size grows up to its limit of a 1000 packets (each packet of 1500 bytes), before registering a queue drop and triggering TCP's congestion control algorithm.

In general, it has been found that while RTT measurements increase with the hop count, they do not always exhibit a proportionate increase, *e.g.* , for 6.5 Mb/s links, a 3-hop and a 4-hop network shows an increase of only 4.5% and 10% over the

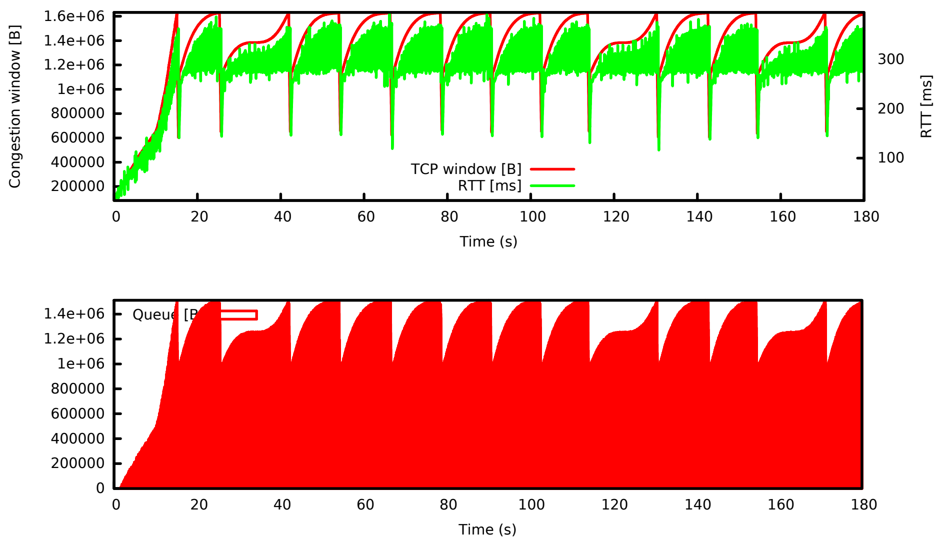


Figure 5.2: TCP congestion window, RTT, and egress queue utilization for a 1-hop TCP flow over a 300 Mb/s wireless link.

2-hop delays. This suggests that with slow speed links, TCP's feedback mechanism is unable to saturate all buffers along the multi-hop path. This is also validated by the buffer utilization plots shown in Fig. 5.3, where none of the buffers fills up to the capacity. In such networks, TCP's congestion control mechanisms are triggered by losses other than queue drops, such as wireless collisions. In contrast, topologies using 144 Mb/s or 300 Mb/s links show a more uniform, consistent increase in RTT with increasing hop-count.

Next, A-MPDU transmit aggregation has been enabled. Measurement results are shown in Figs. 5.4a and 5.4b. The ath9k release used in our testbed nodes does not support A-MPDU aggregation at 6.5 Mb/s, as transmitting a large A-MPDU at this link rate may violate the 4 ms frame transmit duration regulatory requirement in 5 GHz band. Thus, only the results for 144.4 Mb/s and 300 Mb/s link rates have been shown. A-MPDU aggregation significantly increases the network throughput. For a 1-hop network, 144.4 Mb/s link shows a throughput improvement of 3×, while a 300 Mb/s link shows an improvement of 5×. For multi-hop networks using 144.4 Mb/s, the throughput roughly decreases by $\frac{1}{2}$, $\frac{1}{3}$, and $\frac{1}{4}$ for 2, 3, and 4-hop chain topologies.

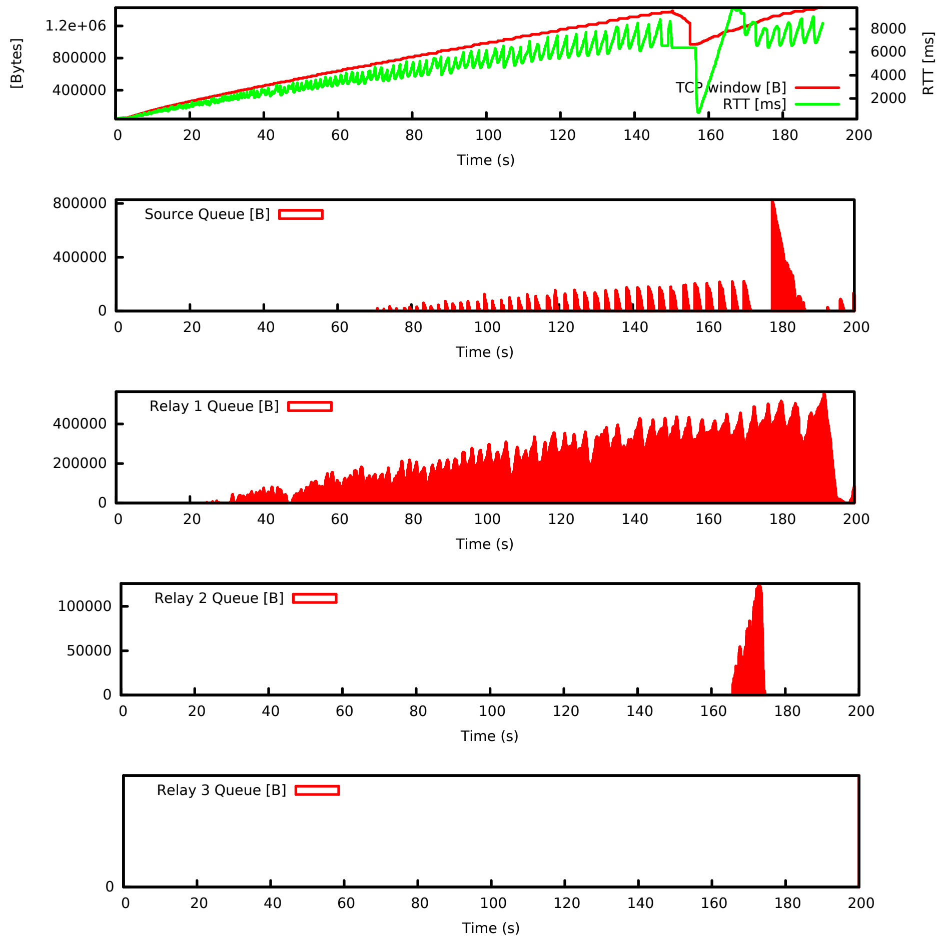
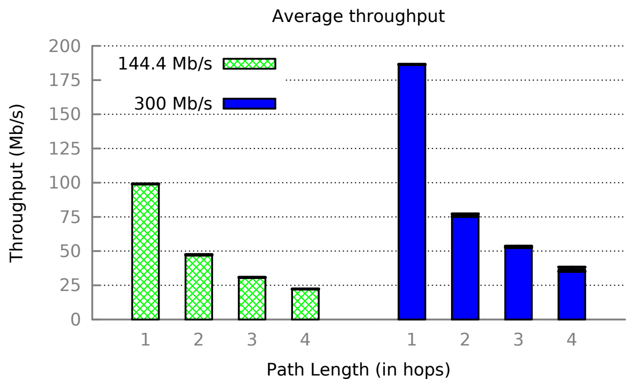
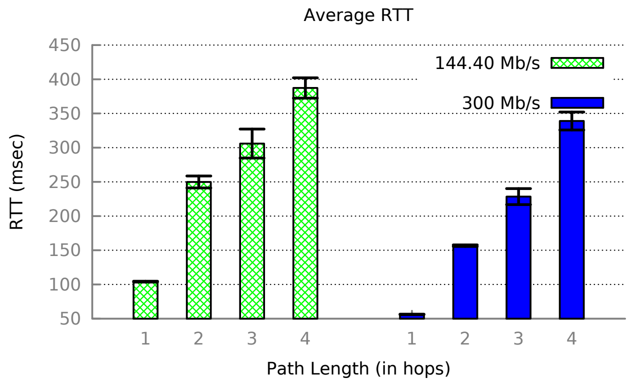


Figure 5.3: TCP congestion window size, RTT, and txqueue size distribution for a TCP flow in a 4-hop chain topology with 6.5 Mb/s wireless links.

However, the throughput drop is higher when using the 300 Mb/s link rate, averaging across 60% for each additional hop. To investigate further, the A-MPDU size (in terms of MPDUs per A-MPDU) at each hop along the path to the destination have been measured . The average A-MPDU size is shown in Fig. 5.5. For 1-hop networks, the A-MPDU size approaches the maximum limit of 32 MPDUs imposed by the device driver. However, the average A-MPDU size is smaller in multi-hop networks. This is because the packets are dispersed in queues at multiple nodes along the path to the destination, and thus a given node may not always have the maximum number of MPDUs ready to transmit together. In particular, 300 Mb/s links use a smaller A-MPDU size compared to 144 Mb/s links for the 2nd, 3rd, and 4th relay nodes in



(a) Throughput



(b) RTT

Figure 5.4: Flow throughput and RTT for wireless links with A-MPDU aggregation.

the multi-hop topologies, leading to higher than 50% drop in throughput over a 1-hop 300 Mb/s link.

In addition to the increase in throughput, A-MPDU aggregation also lowers the RTT across all topologies as shown in Fig. 5.4b. To give an example, the 4-hop RTT is improved by over 3× for both link rates. As illustrated in Fig. 5.5, the A-MPDU scheduler does not always transmit a maximum sized A-MPDU. Many A-MPDUs were transmitted with a smaller size based on the number of frames available in the buffer at a given time. Transmitting multiple MPDUs together debloats the txqueue size, leading to smaller queueing delays and the subsequent reduction in RTT.

Finally, additional set of experiments have been conducted for multi-hop topologies where only part of the nodes used transmit A-MPDU aggregation. Such networks

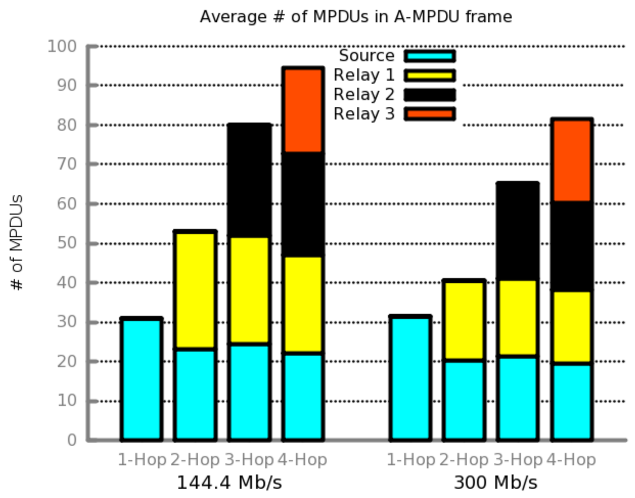
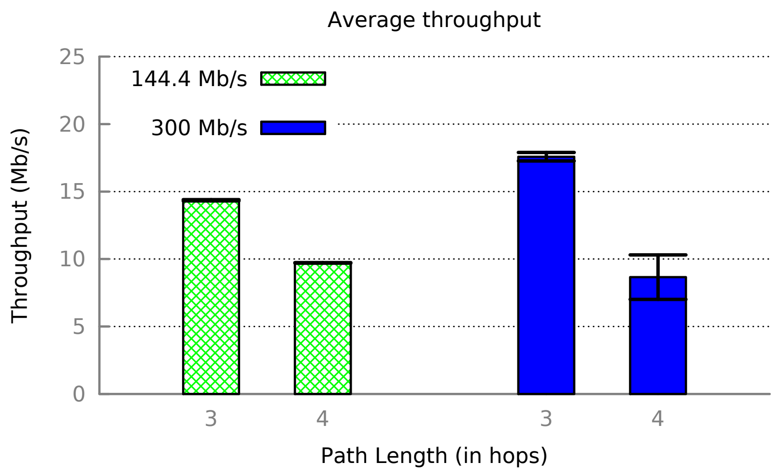


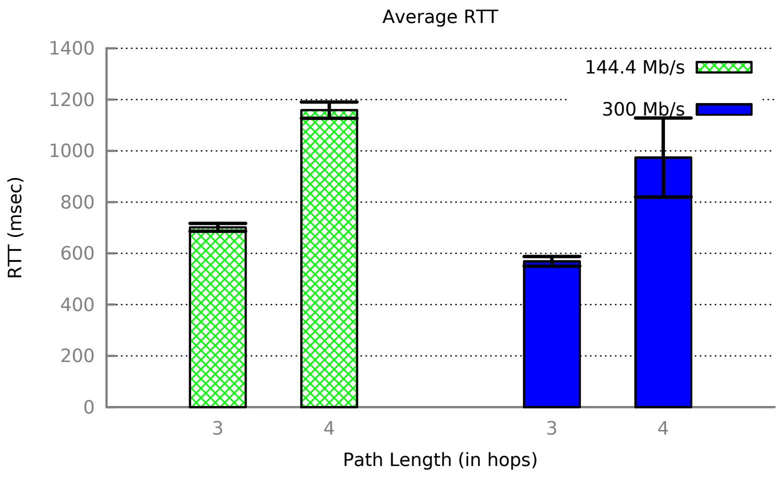
Figure 5.5: Average A-MPDU size. For multi-hop networks, A-MPDU size is measured at each hop along the path to the destination. ath9k does not support Tx A-MPDU aggregation at 6.5 Mb/s link rate.

are likely in heterogeneous environments with a mix of equipment from different vendors or support for backward compatibility with IEEE 802.11 a/b/g technologies. The source node and the first relay node had A-MPDU aggregation enabled, while all subsequent relay nodes had A-MPDU aggregation disabled. Throughput and RTT measurement results are shown in Figs. 5.6a and 5.6b respectively. 1-hop and 2-hop results have been omitted as those are similar to results in Figs. 5.4a and 5.4b.

It could be observed that throughput drops below the values observed when all links used transmit A-MPDU aggregation (Fig. 5.4a), though its still higher than values obtained when A-MPDU aggregation is disabled for all links (Fig. 5.1a). Similarly, RTT measurements also fall in between the values obtained with these two cases. This suggests that even having some partial nodes using A-MPDU transmission in a network can increase the throughput and reduce delays. In such scenarios, the network is bottlenecked by the nodes that do not transmit using A-MPDU aggregation. The queue utilizations across these topologies are shown in Fig. 5.7 for a 4-hop chain topology using 144.4 Mb/s wireless links. This analysis shows that sustained queues mostly build up at the first node that does not support A-MPDU



(a) Throughput



(b) RTT

Figure 5.6: Flow throughput and RTT for a network with partial links supporting A-MPDU aggregation.

transmit aggregation. For the 4-hop chain topology, this is the relay node 2-hops from the source. A buffer sizing strategy for reducing queueing delays would need to identify and manage similar set of bottlenecks in the network.

Multi-flow topologies

In this set of experiments, the goal is to characterize the impact of bufferbloat in wireless networks with multiple backlogged TCP flows. A 4-hop parking lot topology has been used with TCP flows sourced at each successive nodes in a chain and ter-

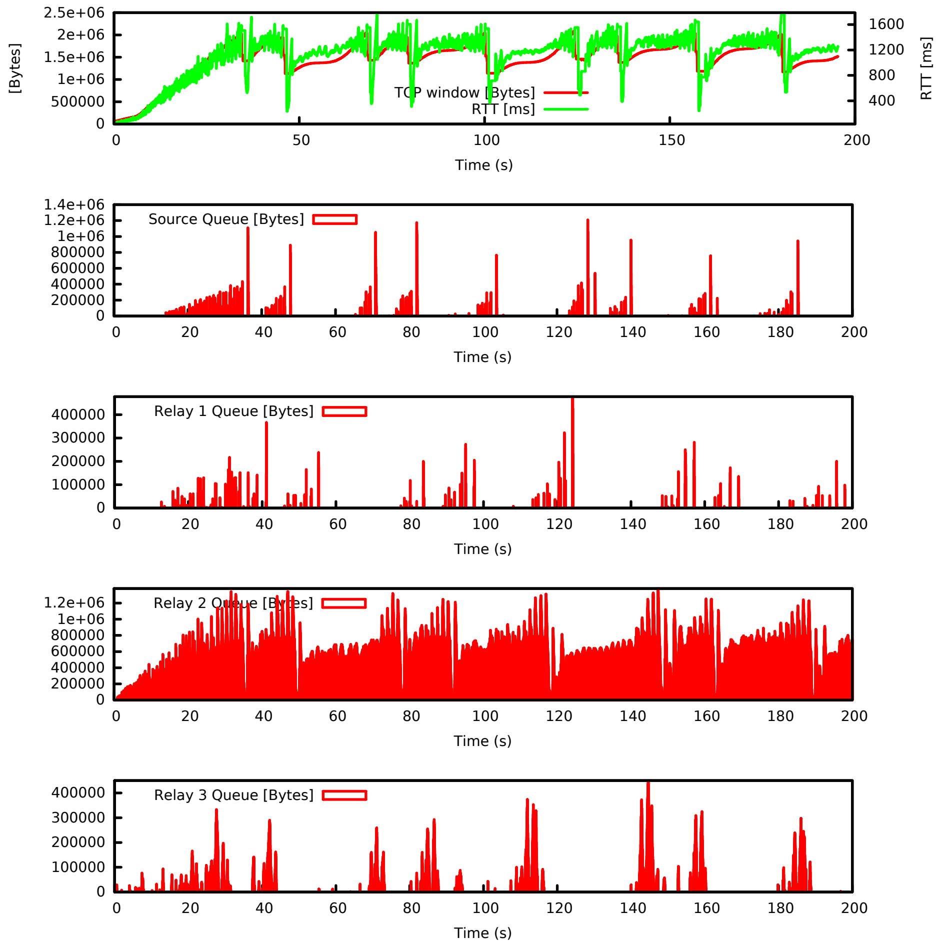


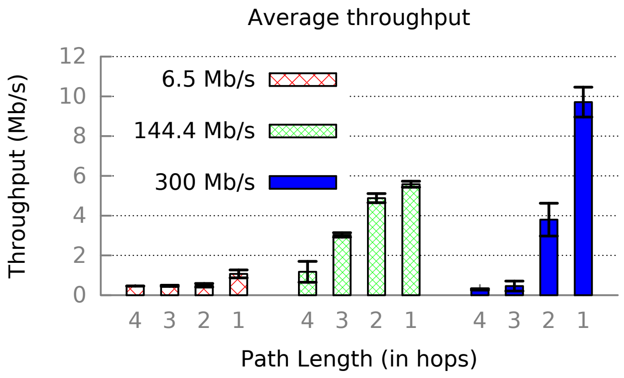
Figure 5.7: TCP congestion window size, RTT, and txqueue size distribution for a TCP flow in a 4-hop chain topology with 144.4 Mb/s wireless links. The source and 1-hop relay node use Tx A-MPDU aggregation.

minating at the final node. Similar to the single-flow experiments, uniform link rates have been used for a given experiment, varying the link rates from 6.5 Mb/s, 144.4 Mb/s, and 300 Mb/s in different experiments.

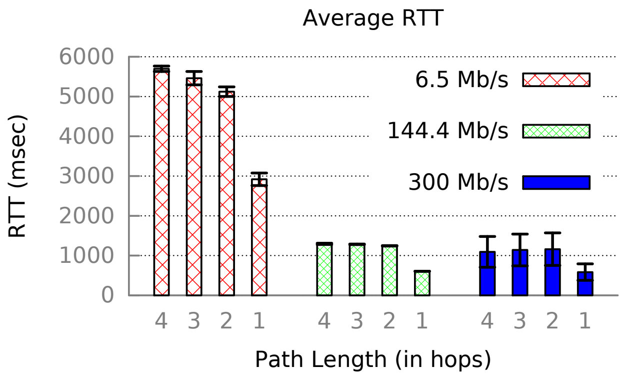
The first set of experiments have been performed with transmit A-MPDU aggregation disabled. Throughput and RTT results are summarized in Fig. 5.8. This figure shows the flow throughput dropping with increasing hop-count. Across all scenarios, the 1-hop flow obtained the highest throughput, followed by the 2-hop flow, and so on. This is the well-known fairness problem in WMN [76]. To quantify the degree of fairness in rate allocation, Jain's Fairness Index (JFI) is computed for the

6.5 Mb/s, 144.4 Mb/s, and 300 Mb/s links as 0.86, 0.81, and 0.46 respectively. It could be observed that the rate allocation becomes more unfair with faster link rates. At these high rates, the 1-hop node can quickly build up a large TCP congestion window, saturating its buffers and starving out flows that traverse more hops. The propagation delay for a 4-flow hop is at least 4 times larger than that of a 1-hop flow. Although that some unfairness in flow rates is expected since TCP allocates rates in proportion to the RTT, results revealed that the throughput is significantly smaller, *e.g.* the 1-hop flow throughput is approximately 2.3×, 4.7×, and 33× the 4-hop flow throughput with the 6.5 Mb/s, 144.4 Mb/s, and 300 Mb/s link rates, respectively. This throughput imbalance persists because of the disproportionate queueing delays experienced by different flows (we would like to note that nodes in our network do not suffer from the hidden terminal or related wireless challenges).

Next, the same experiments have been repeated with transmit A-MPDU aggregation enabled. Results are shown in Fig. 5.9. Experiments with 6.5 Mb/s link rates are omitted since the ath9k driver does not support transmit A-MPDU aggregation at that rate. It could be observed that the unfairness in rate allocation persists, as expected. The JFI for 144.4 Mb/s and 300 Mb/s link rate is 0.77 and 0.50, respectively. The 1-hop flow throughput is 3.38× and 150× the 4-hop flow throughput for the 144.4 Mb/s and 300 Mb/s, respectively. This reinforces the observation that unfairness in rate allocation increases with faster link rates, as both 1-hop and 2-hop flows can quickly saturate the local node buffers. Indeed, in some of the experimental runs, the distant hop flows took a long time just to establish a single TCP connection, with initial TCP setup control messages encountering full buffers along the routing path. This unfairness problem may be addressed using the proposed queue management scheme size as shown later in this chapter.



(a) Throughput

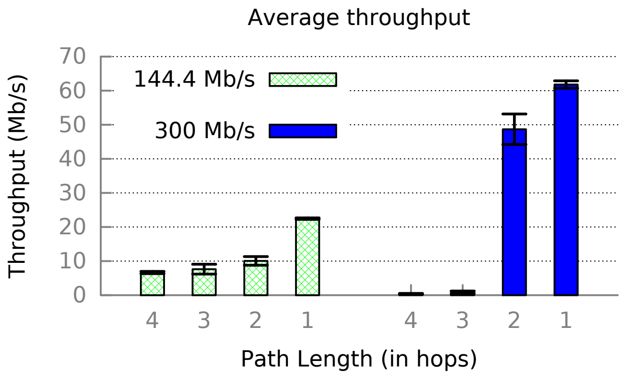


(b) RTT

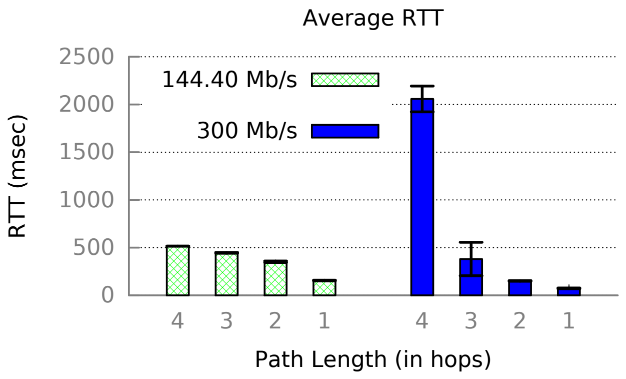
Figure 5.8: Flow throughput and RTT for parking lot topologies without A-MPDU aggregation.

Experiments Summary

In this empirical evaluation the impact of A-MPDU frame aggregation on wireless network performance has been studied over various link rates. It has been shown that the default Linux configuration with A-MPDU aggregation enabled can produce large queueing delays, with RTT values averaging 1700 ms for a single backlogged TCP stream on a 1-hop network with a 6.5 Mb/s wireless link rate. Multi-hop wireless networks have additional buffers at each hop. These large buffers further increase the queueing delays, with RTT values approaching 4600 ms for a 4-hop chain topology with uniform 6.5 Mb/s wireless link rates. The queue utilization analysis shows that while RTT measurements increase with the hop count, they do



(a) Throughput



(b) RTT

Figure 5.9: Flow throughput and RTT for parking lot topologies with A-MPDU aggregation.

not always exhibit a proportionate increase, especially at low wireless link rates where TCP's feedback mechanism is unable to saturate all buffers. Also, it has been shown that A-MPDU aggregation can be used to reduce RTT values, while simultaneously improving throughput. However, the RTT values still approach 350 ms over a 4-hop network. Such delays are catastrophic when queues are shared with real-time flows such as VoIP with strict latency and jitter requirements. Furthermore, aggregates detailed analysis showed a smaller A-MPDU size in multi-hop networks; here, packets are dispersed over multiple nodes, and thus a given node may not always have the maximum number of MPDUs ready to transmit together. As a result, the buffer should be sized carefully taking into consideration A-MPDU size variability in order

```
1  Set the max. acceptable queuing delay $limit$
2  Calculate the initial $B_{initial}$ based on the current Tx rate ($R$) and the round
   trip delay for a single A-MPDU transmission ($ARTT$):
3  $B_{initial} = R * ARTT$
4  for every measurement interval do
5      Calculate queue drain time $T_{drain}$ based on the total number of bits in the
       queue ($BL$) and the percentage of time the channel is not busy ($F$)
6      $T_{drain} = \frac{BL/R}{F}$
7      Adjust the queue size $B$ based on whether the network is bloated or not
8      if $T_{drain} > limit$ and $B > B_{min}$ then
9          if $alarm_{high}$ is ON then
10             decrease queue size $B$
11         else
12             set $alarm_{high}$ to ON and $alarm_{low}$ is OFF
13     else if $T_{drain} < limit$ and $B < B_{max}$ then
14         if $alarm_{low}$ is ON then
15             increase queue size $B$
16         else
17             set $alarm_{low}$ to ON and $alarm_{high}$ is OFF
```

Algorithm 2: WQM OPERATION PSEUDO CODE.

to maximize flow throughput and minimize the overall end-to-end delay. Moreover, this analysis revealed that large buffer sizes can also impact the fair rate allocation. The presented experimental results showed that 1-hop and 2-hop flows can quickly saturate their local buffers, especially at high wireless link rates, starving distant flows. As shown later in this chapter, the proposed buffer sizing technique is able to limit the unbridled growth of the TCP congestion window for small hop-count flows to improve flow rate fairness.

5.3 Approach

In this section, we describe WQM operation and show how we select various WQM parameters.

5.3.1 WQM Operation

WQM algorithm is described in Algo. 2. One of the key design principles in WQM is to make minimal assumptions and take decisions based on measured statistics. For example, WQM uses the link rate to calculate the expected queueing delay. The operation of WQM can be divided into an initial stage and an adjustment stage. In the initial stage, WQM selects a starting buffer size. This size is calculated based on a variation of BDP [21]: the buffer size should be greater than or equal to the product of the bottleneck link capacity with effective end-to-end delay over that link. However, this rule was designed initially for wired networks, and cannot be used directly in IEEE 802.11n based networks as it does not account for A-MPDU frame aggregation. It is obvious that the transmission time, and hence RTT, for a single frame will be less than the transmission time of a single A-MPDU. In fact, it is not always possible to obtain the end-to-end delay. Hence, WQM initializes the buffer using a single-hop RTT that is known. Then, it uses an adaptation algorithm to increase the buffer size if the actual RTT is found larger. This should minimize latency for short bursts of traffic and maximize utilization without sacrificing latency for longer-running traffic. Hence, the initial queue size can be calculated as:

$$B_{initial} = R * ARTT \tag{5.1}$$

where $B_{initial}$ is the initial queue size, R is the current Minstrel transmission rate, and $ARTT$ is the aggregate round-trip delay of a single A-MPDU transmission as illustrated in Fig. 5.10. $ARTT$ can be calculated as per equations 5.6 and 5.7.

After assigning the initial buffer size, the adjustment phase of WQM kicks in. In this phase, the buffer size is tuned to match the current network load. Periodically, the queueing delay is calculated using:

$$T_{drain} = \frac{(BL/R)}{F} \tag{5.2}$$

where T_{drain} is queue drain time, BL is the queue backlog in bits, and F is the percentage of time the channel is free for the sender to transmit *i.e.,* channel is not busy. We divide the queue draining time by the estimate of channel free time to account for the fact that the wireless channel is a shared medium. To illustrate, if three stations are simultaneously contending for the channel, each of them will roughly get 1/3 of the time to transmit. Hence, the time to drain the queue is approximately $3\times$ higher compared to the case where only a single node is transmitting.

If the time to drain the queue T_{drain} exceeds the predefined maximum *limit* for two consecutive *measurement intervals*, then this is an indication that the buffer is bloated. As a result, the buffer size is decreased to limit the amount of buffering and hence limit the queueing delay. On the other hand, if T_{drain} is lower than *limit* for two consecutive *measurement intervals*, then the queue size is increased allowing more frames to be buffered. Observing the network statistics over two consecutive cycles before taking a corrective action helps in accounting for temporary bursty traffic. This corrective action cannot alter the buffer size beyond a minimum and a maximum value, (*i.e.,* $B_{max} > B > B_{min}$ as described in Sec. 5.3.2 below).

Finally, we highlight that a larger $B_{initial}$ may be needed to achieve maximum utilization for multi-hop networks. However, we prefer low latency by starting with a smaller than optimal buffer size that will eventually grow in the adjustment phase. As shown in our experimental analysis in Sec. 5.4, this approach works well with both long-lived and short-lived flows.

5.3.2 WQM Analysis

In this section, we define an upper and lower bound for the buffer size (B_{max} and B_{min}, respectively) and define the maximum allowed queueing delay (*limit*). To find an upper bound on the buffer size, B_{max}, let us consider an IEEE 802.11n network with a single TCP stream. Assume that the maximum possible transmission rate is λ packets[1]/s. Assuming that the TCP stream is in the congestion avoidance phase, the congestion window is going to grow until it reaches W_{max} when a packet loss happens. As a result, the sender halves its TCP congestion window. It will wait for $\frac{W_{max}/2}{\lambda}$ before going to the transmit phase again. The buffer drain time is B/λ s. Ideally, the sender always transmits just before the buffer gets empty in order to make sure the link is fully utilized, (*i.e.,* $\frac{W_{max}/2}{\lambda} \leq B/\lambda$), or

$$B \geq \frac{W_{max}}{2} \tag{5.3}$$

Also, to maintain full link utilization, the sender TCP transmission rate (*i.e.,* $\texttt{cwnd}/ARTT$) should be at least λ . Hence, $\frac{W_{max}/2}{ARTT} \geq \lambda$, or,

$$\frac{W_{max}}{2} \geq ARTT \cdot \lambda \tag{5.4}$$

From (5.3) and (5.4),

$$B \geq \lambda \cdot ARTT \tag{5.5}$$

Hence, the maximum buffer size B_{max} is equal to the BDP using the maximum possible transmission rate and the corresponding packet RTT. Mainly, $ARTT$ represents the transmission delay as propagation delay is considered negligible in wireless networks. Fig. 5.10 shows the MAC overhead of a single A-MPDU transmission over

[1]We use packets instead of bits for ease of exposition.

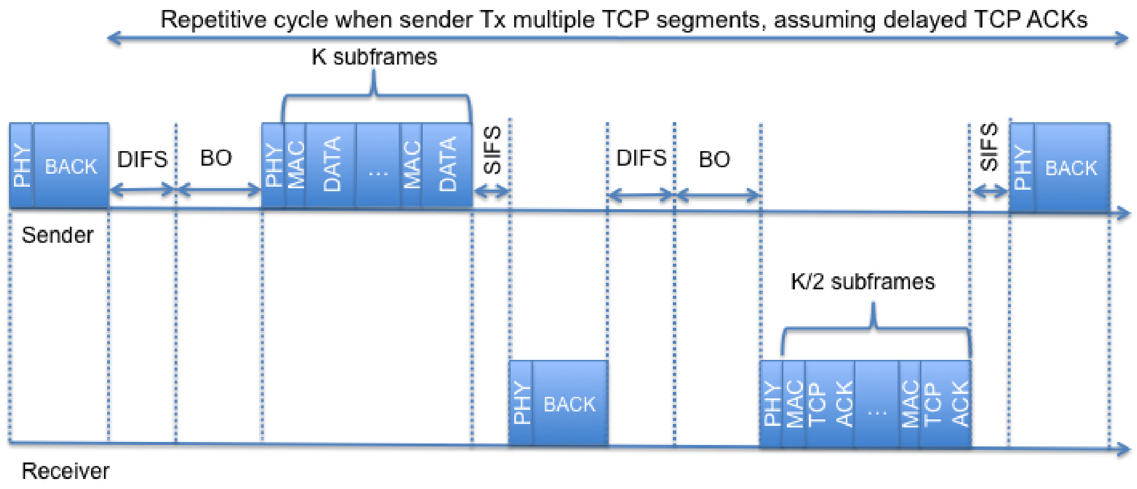


Figure 5.10: 802.11n MAC overhead per A-MPDU transmission.

IEEE 802.11n network. As per this figure, $ARTT$ is the sum of the TCP segment transmission time T_{d-DATA} and the TCP ACK transmission time T_{d-ACK} which can be calculated as per the following equations:

$$\begin{aligned} T_{d-DATA} =& T_{BO} + T_{DIFS} + 2 * T_{PHY} + T_{SIFS} + T_{BACK} \\ &+ K * (T_{MAC} + T_{DATA}) \end{aligned} \tag{5.6}$$

$$\begin{aligned} T_{d-ACK} =& T_{BO} + T_{DIFS} + 2 * T_{PHY} + T_{SIFS} + T_{BACK} \\ &+ K/2 * (T_{MAC} + T_{TCP-ACK}) \end{aligned} \tag{5.7}$$

The system parameters for the used IEEE 802.11n network are listed in Table 5.1. T_{BO} is the backoff interval in case the channel is found to be busy. Based on both MAC contention window and slot duration T_{slot}, the average backoff time could be calculated as $\overline{T_{BO}} = (CW_{min} - 1) \times T_{slot}/2$. SIFS time interval is T_{SIFS} and DIFS time interval is T_{DIFS}. T_{DATA} and T_{MAC} are aggregate frame and MAC header transmission times respectively. A single A-MPDU may contain K TCP segments, potentially as large as 64kB or a total of 64 segments whichever is smaller, each with its own MAC header. Hence, a transmission duration of $K * (T_{DATA} + T_{MAC})$ is added per A-MPDU. The TCP ACK transmission time is $T_{TCP-ACK}$ whereas T_{BACK} is the time to

Parameter	Value
T_{slot}	slot time = 9 μs
T_{SIFS}	shortest inter-frame space = 16 μs
T_{DIFS}	distributed inter-frame space = 34 μs
T_{PHY}	PHY preamble and header time = 33 μs
CW_{min}	minimum contention window size = 15
CW_{max}	maximum contention window size = 1023
T_{BO}	average back-off interval = $(CW_{min} - 1) * T_{slot}/2$
R	physical rate based on rate control algorithm (Mb/s)
R_{basic}	basic physical rate = 6 Mb/s
K	maximum A-MPDU length
T_{MAC}	MAC header time = L_{MAC}/R
L_{MAC}	MAC overhead = 38 bytes = 304 bits
T_{DATA}	data frame time = L_{DATA}/R
L_{DATA}	data frame length = 1500 bytes = 12000 bits
$T_{TCP-ACK}$	TCP ACK time = $L_{TCP-ACK}/R$
$L_{TCP-ACK}$	TCP ACK length = 40 bytes = 320 bits
T_{BACK}	block ACK frame time = L_{BACK}/R_{basic}
L_{BACK}	block ACK frame length = 30 bytes = 240 bits

Table 5.1: System parameters of IEEE 802.11n [2].

transmit a MAC-level block ACK frame. Assuming that TCP delayed acknowledgement is used, only $K/2$ frames are acknowledged. T_{PHY} is the transmission duration of both PHY preamble and header.

The maximum buffer size is needed when the sender transmits with the highest possible Tx rate, 600 Mb/s for IEEE 802.11n, and all frames are sent with maximum A-MPDU length, *i.e.,* $K = 64$ subframes. The delay for transmitting an A-MPDU with maximum length and its block ACK (per the exchange shown in Fig. 5.10) over a single hop is about 1.9 ms using a 600 Mb/s link. According to (5.5), the upper bound on the buffer size B_{max} should be 95 packets. As a lower bound, the minimum buffer size B_{min} should be equal to the maximum A-MPDU length allowed by the link rate. This is because permitting the queue size to be smaller than the number of subframes in a single A-MPDU will result in sending shorter aggregates which in turn limits the network throughput.

We now compute a lower bound on the allowed queueing delay (*limit*). As shown earlier, the maximum allowable aggregate length varies with the link rate. Consider a wireless channel with high interference where the rate control algorithm chooses to transmit at the lowest possible link rate (6.5 Mb/s for IEEE 802.11n radios). As per the ath9k [27] aggregation implementation logic, A-MPDU aggregation is disabled when transmitting at 6.5 Mb/s. As a result, *limit* should be greater than or equal to the transmission time of one frame at the lowest possible rate. As per (5.6) and (5.7), *limit* should be greater than or equal to 2.5 ms. Although different sessions may require different value of *limit*, we prefer to fix the value of *limit* in order to enhance the practicality of our algorithm. In fact, we test the value of $limit = 2.5$ ms in our testbed experiments over multiple scenarios and find that this value allows for huge reduction in latency while preserving the network throughput across a wide range of bandwidths, RTTs, and traffic loads.

5.4 Experimental Analysis

5.4.1 Implementation Details

WQM is implemented as a daemon running in Linux user space. The source code is available at [77]. WQM controls the length of *txqueue* using the *ifconfig* utility and gathers channel related information using the *iw* utility. An important parameter in WQM is the frequency of obtaining channel statistics. WQM uses the same look-around interval as Minstrel [22], which is the default rate control algorithm in Linux, because the link rate is guaranteed to be fixed over this interval. Every 100 ms, WQM obtains a sample of the current transmission rate, the buffer backlog, the average A-MPDU length, and the percentage of time the channel was busy in the last look-around. In reality, this look-around interval is found to be sufficient to collect meaningful samples without wasting too many CPU cycles. Further, our experimental

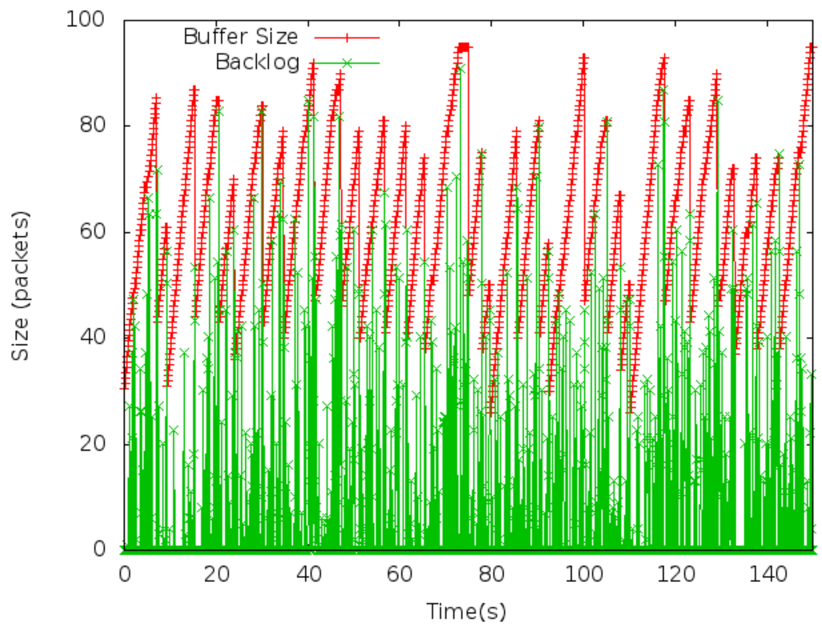


Figure 5.11: WQM buffer size adaptation in response to variation in queue occupancy. This figure represents the AIMD behaviour of WQM.

analysis in Sec. 5.4.2 shows that this interval is sufficient to respond quickly to changes in the environment.

As explained earlier, if WQM estimates the queueing delay to be longer than the desired target, then it is going to reduce the buffer size to lower the delay. However, the buffer should not be smaller than the maximum number of subframes per aggregate as this may only limit the utilization without any reduction in the latency. On the other hand, if the queue draining time is less than target, WQM can safely increase the buffer size. Our WQM implementation uses a conservative approach in which the buffer size grows linearly (*i.e.*, the buffer size is increased by one packet) in response to measurements outlined in Algorithm 2. When required by the algorithm, WQM decreases the buffer size by half to make sure the network latency remains bounded. Thus, WQM strictly prefers low latency over high goodput. Fig. 5.11 illustrates this behavior by showing the variation of buffer size over time. This AIMD behavior of the algorithm is chosen as it experimentally outperforms all other possible alternatives, namely AIAD, MIAD, and MIMD in terms of end-to-end delay as shown

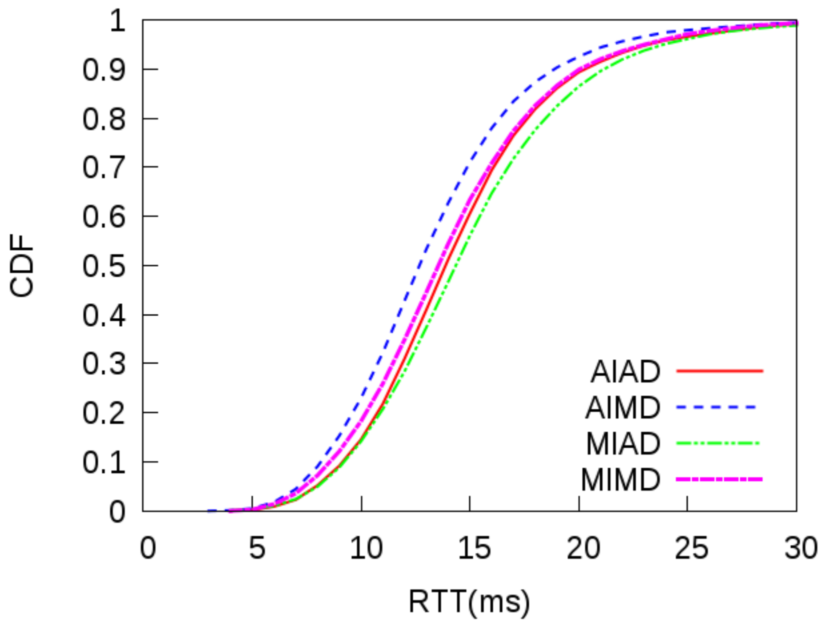


Figure 5.12: Comparing the round trip delay of various versions of WQM algorithm.

in Fig. 5.12. Our testbed specification are summarized in Table 3.1.

We compare the performance of WQM to the default Linux configuration, where the buffer size is set to 1000 packets, and two state of the art bufferbloat solution in the literature, namely CoDel [36] and PIE [38]. Both CoDel and PIE are considered to be parameterless with auto-tuning functionalities. Hence, we do not need to specify any additional parameters for them. Also, both of them were evaluated only over wired networks using the Network Simulator NS-2. It is well known that network simulators do not represent precisely the real world because of many unrealistic assumptions. It is good for preliminary results and/or for checking scalability on very large networks. Similar to WQM, CoDel was evaluated while running several FTPs using TCP CUBIC over various link rates. In a similar way, PIE was evaluated while running 5 TCP NewReno flows over a 10 Mbps link. PIE was also evaluated while running 20 TCP NewReno flows for 100 seconds in a small testbed. This is similar to the mutiflow experiments we did using our testbed. We also compare the performance of WQM to DNB which was proposed in the previous chapter.

Hops	Goodput (Mb/s)				
	Default	WQM	CoDel	PIE	DNB
1-hop	155.7	135.78	134.47	145.735	58.05
2-hops	66.67	62.64	63.89	59.226	27.8
3-hops	41.17	39.176	41.876	38.203	15.66

Table 5.2: Average goodput of WQM, CoDel, PIE, DNB, and Linux default settings over various number of hops.

Hops	RTT (ms)				
	Default	WQM	CoDel	PIE	DNB
1-hop	61.51	12.98	22.19	75.7	18.33
2-hops	169.44	30.84	57.1	185.027	40.89
3-hops	224.387	49.471	90.428	261.256	74.44

Table 5.3: Mean RTT of WQM, CoDel, PIE, DNB, and Linux default settings over various number of hops.

5.4.2 Experimental Evaluation

In this section, we present our experimental performance evaluation of WQM. WQM performance is evaluated in a wireless testbed on our campus. The testbed hardware ans software specifications are detailed in Sec. 3.1. Our experiments could be classified into two main groups based on the number of concurrent flows used in the experiments, namely single flow scenarios and multi flow scenarios.

Single Flow Scenarios

In this set of experiments, we run a single flow between the sender and the receiver and measure its goodput and latency. We compare the performance of WQM to CoDel, PIE, DNB, and the default static buffer size. We start by evaluating WQM in multi-hop wireless scenarios, where we vary the number of hops between the sender and the receiver from one to three. Hence, packets are queued multiple times before reaching their destinations. The CDF for RTT over various topologies is shown in Fig. 5.13 and the corresponding goodput is shown in Fig. 5.14. We would like to

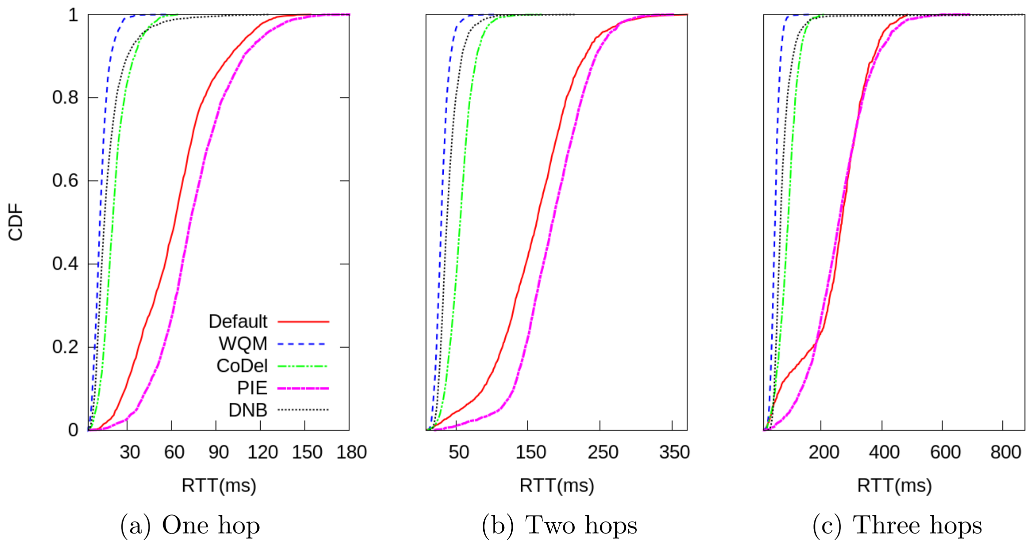


Figure 5.13: RTT CDF for a single flow while varying the hop count.

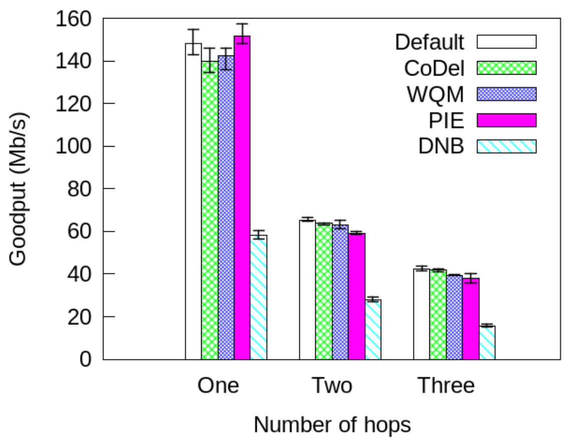
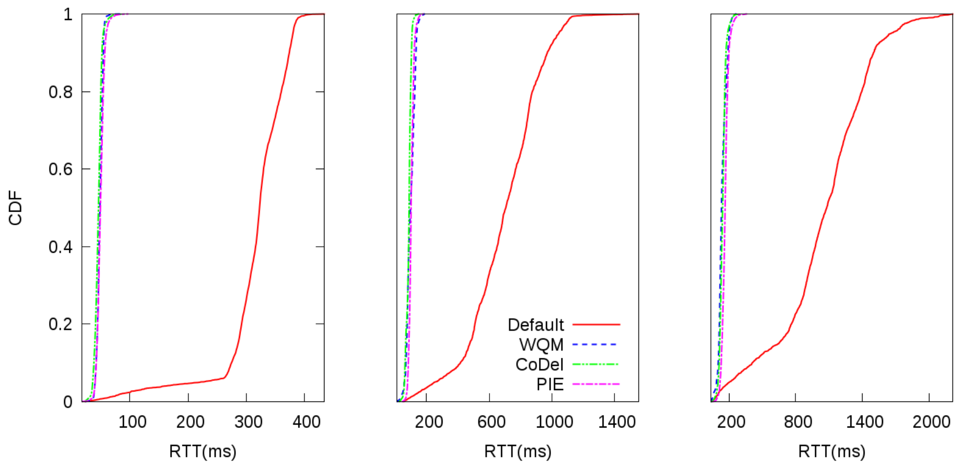


Figure 5.14: Goodput of a single flow while varying the hop count.

note that unless otherwise stated, all the results in this section are averaged over at least three runs and the error bars represent the minimum and maximum values. To make the comparison easier, we list average goodput and latency results for the all the schemes in Tables 5.2 and 5.3, respectively. For various number of hops between the sender the receiver, WQM reduces the network latency by at least 5× compared to both PIE and the default buffer size and 2× compared to CoDel. To give an example, WQM manages to reduce the mean delay in the three hops scenario from 261.3 ms in the case of PIE to only 49.47 ms at the cost of less than 5% goodput reduction in the worst case. As expected, DNB suffers from low goodput compared to WQM as well as other schemes. This is due to the fact that DNB chooses very small neighbourhood buffer size which prevents building large frame aggregates. Hence, the utilization of the network is going to be affected due to low level of frame aggregation in the network. We attribute the ability of WQM to outperform other schemes in controlling the queuing delay to the achieved level of frame aggregation. Suppressing frame aggregation increases the end-to-end delay under all scenarios as shown in Table 5.3. This is in agreement with our earlier findings in [78]. Furthermore, it is clear that the needed buffer size in wireless networks is much lower than the buffer size limit in PIE which is 1000 packets.

In order to support backward compatibility, IEEE 802.11n devices disable frame aggregation if the receiver is not capable of dealing with aggregates. To test this scenario, we repeat the previous set of experiments after disabling A-MPDU aggregation. For various hop counts, the RTT CDF and average goodput are shown in Fig. 5.15 and 5.16 respectively. Compared to the default case with 1000 packets buffer, WQM achieves upto 7$\times$ reduction in latency while having similar goodput. As expected, WQM performs as good as CoDel and PIE in terms of delay and goodput when aggregation is disabled. This set of experiments show that even if the Wi-Fi devices are not deployed in green field mode, *i.e.,* the network is not solely composed of IEEE 802.11n devices, WQM can still maintain an acceptable network latency. It is also worth noting that these experiments show that A-MPDU frame aggregation helps deflating the buffer which results in significant queueing delay reduction from about two seconds to only half a second in the worst case.

To evaluate WQM under various channel conditions, the same set of experiment are repeated three times while varying the distance between the testbed nodes. We start with the default distance in our testbed which is 10 m, then increase it to 20 m, and finally to 30 m. Delay and goodput results are shown in Fig. 5.17 and 5.18



(a) One hop (b) Two hops (c) Three hops

Figure 5.15: RTT CDF for a single flow while varying the hop count after disabling A-MPDU frame aggregation.

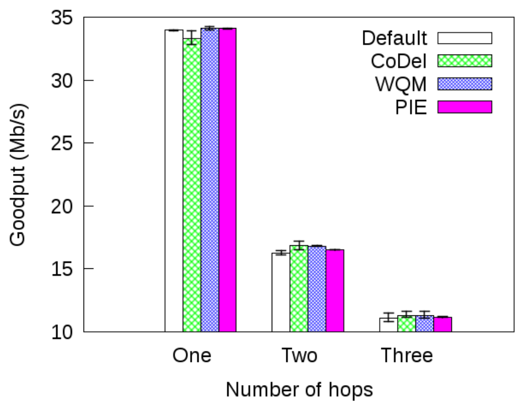
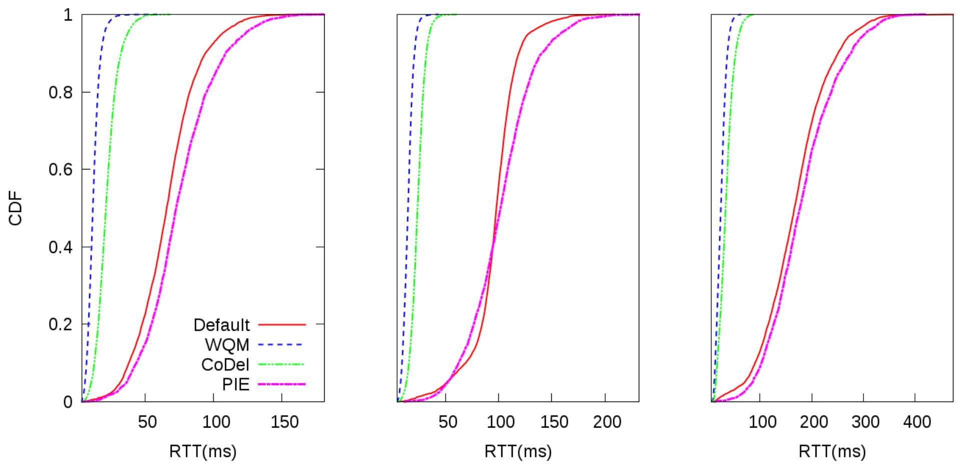


Figure 5.16: Goodput of a single flow while varying the hop count after disabling A-MPDU frame aggregation.

respectively. As expected, nodes that are far from each other suffer from longer delay compared to closer nodes. In all the three cases, WQM outperforms CoDel, PIE and the default buffer size in terms of RTT at the cost of 7% drop in goodput in the worst case. It is interesting to note that the gap in goodput between WQM and the other schemes shrinks as the distance between the nodes gets longer. This happens because the rate control algorithm, which is enabled by default in our testbed, reduces the transmission rate when the transmitter and the receiver are far from each other in order to increase the link reliability. This in turn lowers the BDP *i.e.*, the



(a) 10m (b) 20m (c) 30m

Figure 5.17: RTT CDF while varying the distance between nodes in the testbed.

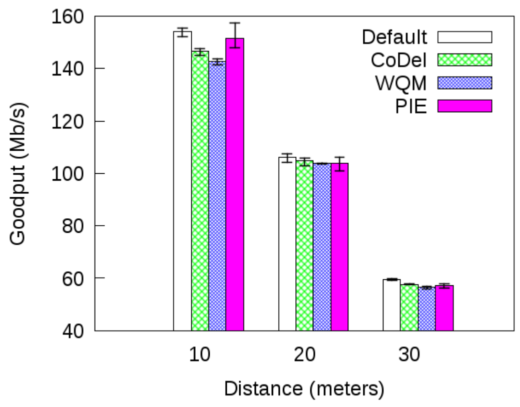
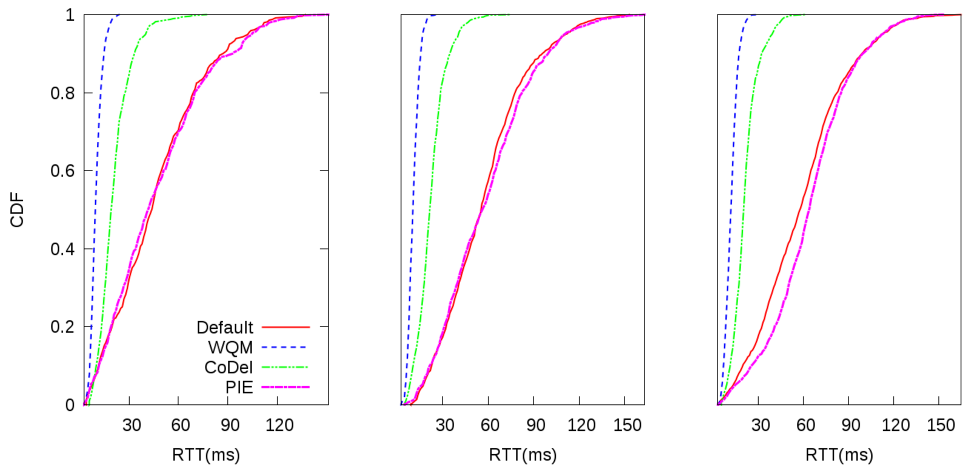


Figure 5.18: Average goodput achieved while varying the distance between nodes.

needed buffer in the network. As a result, the effect of WQM small buffer on network utilization is going to be minimal in this case.

Finally, we evaluate the performance of WQM using short flows. In all previous experiments, long flows of 100 s are used to simulate large file transfers that are able to saturate the channel and fill the buffers. However, short flows are also common in real life. To test this scenario we repeat the same set of experiments while varying the flow duration from 5, 10, to 15 s and observe the delay and goodput. To get accurate results, we repeat every experiment 10 times and report the average. The CDF of RTT is shown in Fig. 5.19 and the average goodput is shown in Fig. 5.20. It can be



(a) 5s (b) 10s (c) 15s

Figure 5.19: RTT CDF while varying the flow duration.

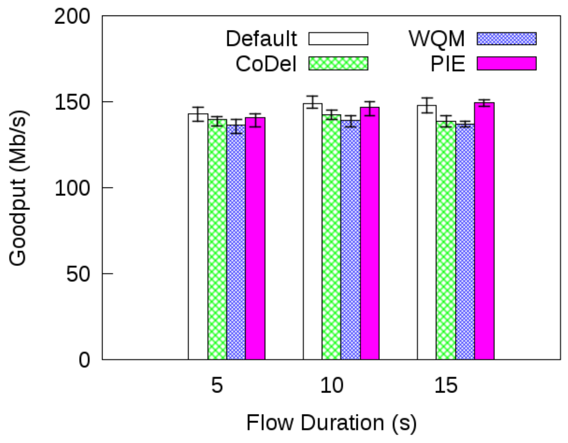
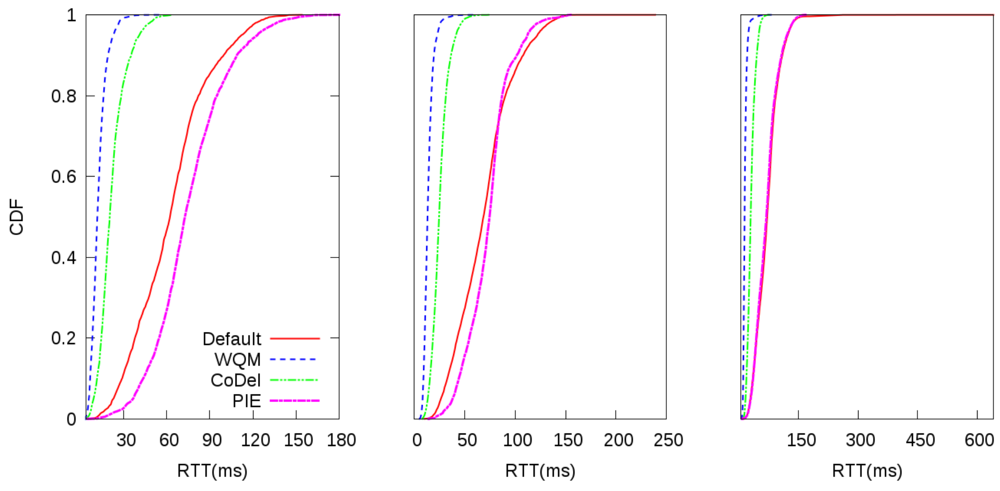


Figure 5.20: Average goodput achieved while varying the flow duration.

observed that in all cases the delay is bounded to 150 ms which is a direct implication of not building the buffers. However, WQM is still achieving the best reduction in RTT compared to all other schemes. In the worst case, this reduction comes at the price of 8% drop in goodput.

Multi Flow Scenarios

In this section, we evaluate the performance of WQM while running multiple concurrent flows instead of only one between the transmitter and the receiver. In the first



(a) One flow (b) Three flows (c) Five flows

Figure 5.21: RTT CDF of various concurrent flows over a single hop topology.

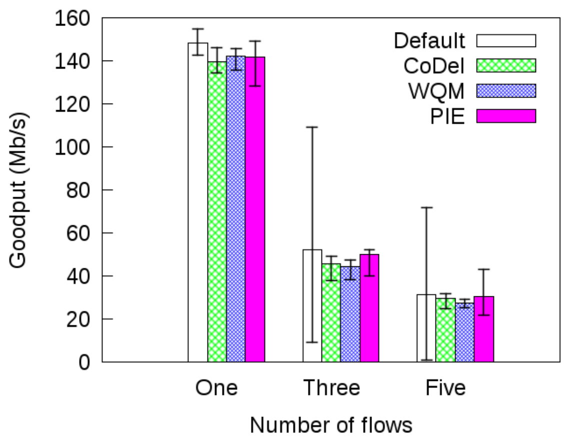
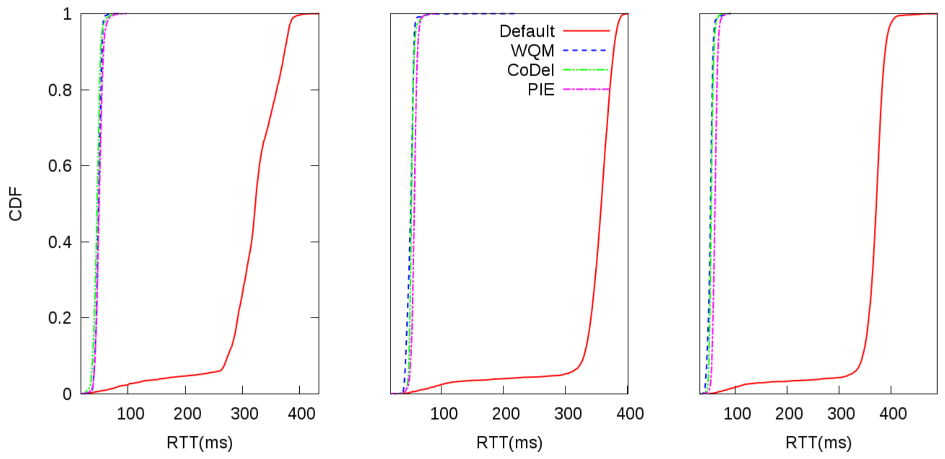


Figure 5.22: Average goodput with multiple flows over a single hop topology.

set of experiments, we increase the number of concurrent flows from one to five, and measure the overall network goodput and latency for WQM, CoDel, PIE as well as the default static buffer. The RTT CDF for 1, 3, and 5 flows are shown in Fig. 5.21 and the average per flow goodput for the three cases are shown in Fig. 5.22. For all scenarios, WQM reduces the network latency by at least 5× compared to both PIE and the 1000 packets buffer at the cost of 13% goodput reduction in the worst case. Compared to CoDel, WQM reduces the delay by almost 2×. Goodput results of WQM and CoDel are within error bounds of each other. Note that as the number



(a) One flow (b) Three flows (c) Five flows

Figure 5.23: RTT CDF of various concurrent flows over a single hop topology after disabling A-MPDU aggregation.

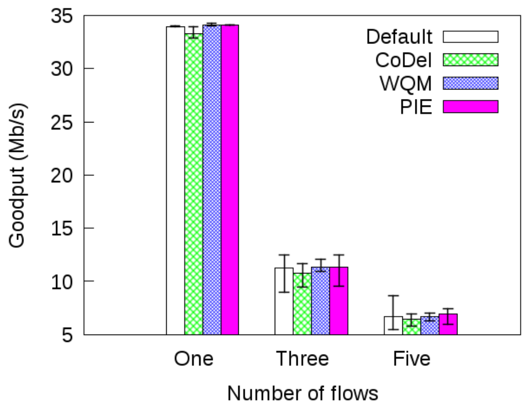


Figure 5.24: Average goodput with multiple flows over a single hop topology after disabling A-MPDU aggregation.

of flows increases, the default scheme suffers from severe unfairness (including starvation) between the flows, as reflected by the error bars. For example, JFI value for the default case is 0.7 for the 3 flows case, compared to 0.99 for all other schemes. Large buffers in the default scheme lead to severe unfairness because one or more flows can fill up the buffer quickly while starving others. WQM prevents this behavior by simply controlling the number of buffered packets.

As mentioned earlier, frame aggregation might be disabled in certain cases. To test the performance of WQM under these scenarios, we repeat the previous set

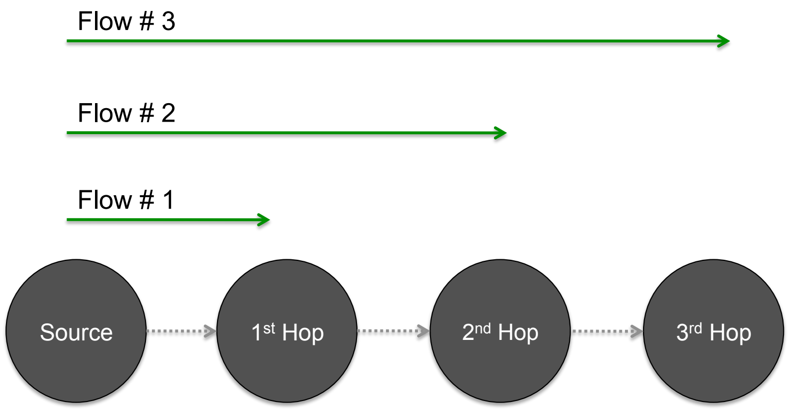
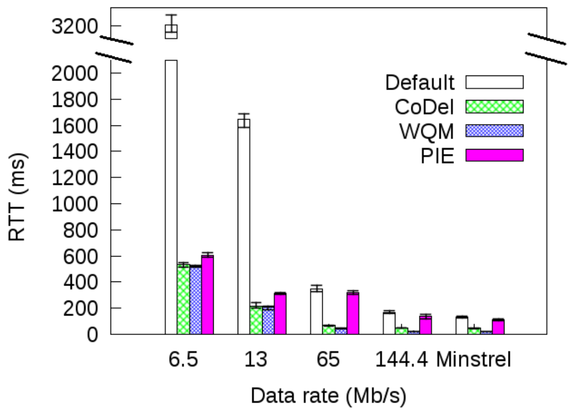


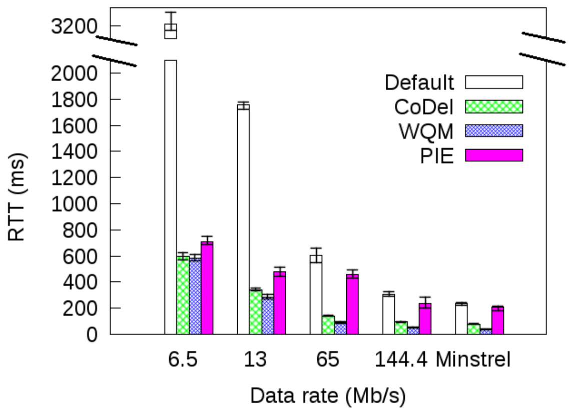
Figure 5.25: Illustration of the parking lot topology used in our experiments.

of experiments while disabling A-MPDU aggregation. The latency performance is shown in Fig. 5.23 and the average per flow goodput is shown in Fig. 5.24. When compared to the default static buffer size, WQM reduces the delay by around 7× while getting similar goodput. WQM is performing as good as both CoDel and PIE in this case. This is happening because WQM accounts for frame aggregation when selecting the optimal buffer size. As illustrated in Sec. 5.3.2, the number of subframes per aggregate, which is one packet in this case, will be used as a lower bound to buffer size in WQM. In fact, this one packet limit is what CoDel and PIE use in their buffers. After all, this set of experiments show that WQM still performs as good as the state of the art even at border cases.

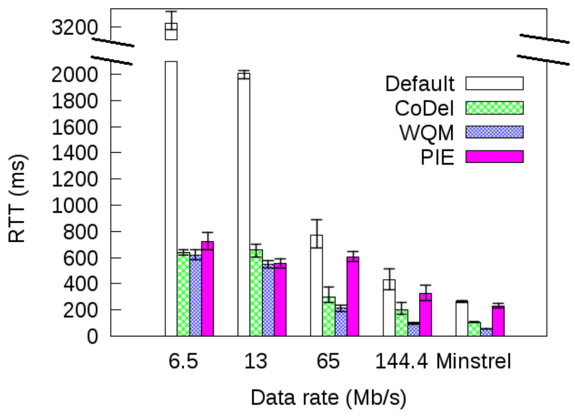
Finally, we analyze the performance of WQM over both multi-hop and multi-flow scenarios. In this experiment, nodes are organized in a parking lot topology, as shown in Fig. 5.25, where three flows starts simultaneously from the same source but are destined to different nodes in the network. In our case, we use three flows in a three hop topology where the 1st flow ends at the first hop from the destination, the 2nd flow stops at the second hop and the 3rd flow is the only one that reaches the third hop. This experiment is repeated several times while enabling and disabling Minstrel which is the default rate control algorithm. When disabled, the rate is set manually to either 6.5, 13, 65 or 144.4 Mb/s. The mean RTT per flow as well as the total



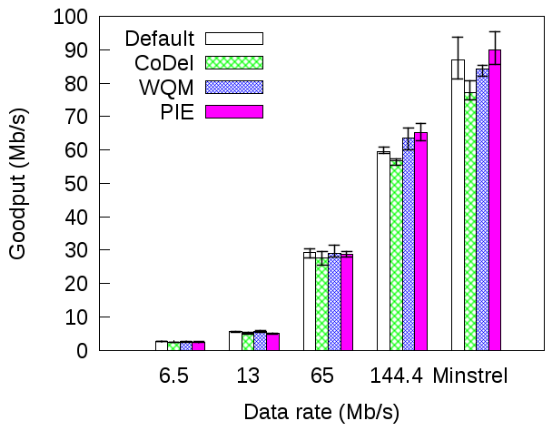
(a) One hop flow



(b) Two hops flow



(c) Three hops flow



(d) Net goodput

Figure 5.26: Average end-to-end delay per flow and total goodput in the parking lot topology.

goodput achieved by the three flows combined are shown in Fig. 5.26. Similar to the previous experiments, error bars represent maximum and minimum values over at least three runs. When compared to the default buffering scheme, WQM reduces end-to-end delay by 8× for the one hop flow, 6× for the two hops flow and at least 4× for the three hops flow regardless of the transmission rate. This reduction in queuing delay does not come at the price of significant goodput reduction. This is another proof that such a large buffer is not always needed. Also, WQM outperforms CoDel in terms of delay and goodput in all the cases. For example, WQM reduces queueing delay by 2× compared to CoDel when Minstrel is enabled while achieving slightly better goodput. Since aggregation is disabled at 6.5 Mb/s link rate, PIE performance is close to WQM performance. As the rate increases, the gap between PIE and WQM gets bigger as longer aggregates are being transmitted. In the worst case, PIE suffers from 7$\times$ more latency compared to WQM.

Chapter 6

Buffer Management in Wireless Full-Duplex Systems

6.1 Introduction

The long-held assumption that wireless devices can only operate in half-duplex mode is not true after the appearance of the full-duplex wireless systems. Traditionally, a radio is not able to transmit and receive simultaneously using the same channel because the antenna at the receiver side is going to hear its own transmission which is hundreds or thousands of times stronger than the signals coming from other nodes. In fact, full-duplex systems [79] [80] have succeeded in challenging this assumption by using cancellation techniques to cancel self-interference and eliminate the noise created by transmit signal.

Recent research efforts proved the feasibility of wireless full-duplexing as illustrated in Fig. 6.1, thanks to interference cancellation techniques. *Choi et al.* [79] achieved a single channel full-duplex wireless communication by introducing a novel self-interference cancellation scheme called "Antenna Cancellation". The insight behind antenna cancellation is that the transmissions from multiple antennas could be added destructively when placing the receive antenna in a specific location. However, this design suffers from many limitations. First of all, the prototype requires 7

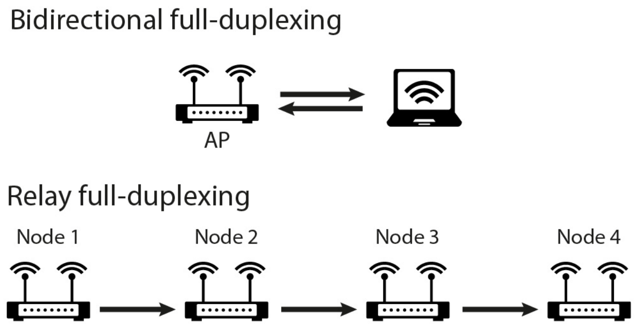


Figure 6.1: Bidirectional full-duplexing and relay full-duplexing.

inches of spacing between antennas which makes it unsuitable for today's tiny wireless cards. In addition, it supports neither wide bandwidths, such as the 20 MHz IEEE 802.11 Wi-Fi signals, nor high transmit powers. Moreover, antenna cancellation is very sensitive to antennas placement mismatch.

To overcome these limitations, *Jain et al.* [80] proposed a full-duplex radio design based on a balanced/unbalanced (Balun) transformer. The mechanism known as "balun cancellation" exploits signal inversion using a balun circuit in an adaptive manner to match the self-interference signal. This design, unlike the antenna cancellation based design, eliminates the bandwidth constraint and supports high transmit powers. While it solves many problems related to wireless full-duplex systems, this cancellation mechanism is very sensitive to delays and needs very sophisticated electronic components. Further, TX and RX antennas are separated by 20 cm which represents an engineering limitation especially for mobile devices like tablets and mobile phones.

In 2012, a group of researchers form Rice University [81] implemented a practical 20 MHz IEEE 802.11 multi-antenna full-duplex system. Their design achieves almost the intended doubling of throughput. In addition, *Hong et al.* [82] introduced a transparent spectrum slicing scheme called "Picasso" which allows simultaneous transmission and reception on separate and arbitrary spectrum fragments using a single antenna. Picasso solves the problem of leaking of interference into adjacent spectrum especially in Wi-Fi Orthogonal Frequency-Division Multiplexing (OFDM) signals. After that, several other implementations of full-duplex systems appeared such as the implementation of full-duplex IEEE 802.11ac radio using a single antenna for both transmit and receive [83]. This achievement was made possible thanks to the use of an analog cancellation board and a circulator.

The research about full-duplex did not stop at this point. *Bharadia & Katti* [84] demonstrated that full duplex can be combined with MIMO putting an end to comparisons between the performance of MIMO half-duplex and full-duplex systems. Conceiving a MIMO full-duplex design was not a simple task because a single antenna in the MIMO full-duplex system may suffer not only from the ordinary self-interference but also from a very strong cross-talk coming from neighboring antennas in the TX chain. In fact, the use of multiple Single Input Single Output (SISO) full-duplex replicas to implement full-duplex MIMO is ineffective due to several complexity and scalability issues. In order to reduce complexity, the proposed design exploits the fact that neighboring MIMO antennas share a similar radio environment. Hence, this solution is based on creating a cascaded filter structure.

Full-duplex systems have shown great potential to solve important challenges in wireless networks such as hidden terminals, loss of throughput due to congestion, and large end-to-end delays [79]. In fact, the idea of receiving and forwarding simultaneously can reduce the large end-to-end delays in multi-hop networks since a full-duplex node can simultaneously start forwarding a packet to the next hop while receiving it. However, full-duplex relaying is not sufficient to completely solve the problem of high latency in today's networks. For instance, wired networks operate in full-duplex mode and still suffer from unacceptable delays. If we take into consideration the fact that wireless spectrum is a shared resource between a set of neighboring nodes even in full-duplex mode, the situation will be worse. In order for existing wireless full-duplex designs to have large scale deployments, they should address such challenges. To address the issue of bufferbloat in wireless full-duplex systems, we propose using WQM mechanism to manage the buffers in the full-duplex nodes. As explained in Ch. 5, WQM dynamically adjusts the buffer size according to queue draining time and current size. To the best of our knowledge, this is the first attempt to address the buffer management issue in wireless full-duplex systems. We demonstrate through simulation that WQM has succeeded to decrease latency by two orders of magnitudes while achieving better throughput compared to the conventional Drop Tail mechanism.

6.2 Approach

Typically, wireless devices have two types of buffers, namely transmit buffers and receive buffers. The latter usually do not get bloated due to good processing capabilities in today's wireless devices. In the case of multi-hop networks, the wireless node processes incoming packets and may forward some of them to the transmit queue in case this node is not their final destination. Since bloated transmit buffer may cause long queuing delay, we focus on managing this kind of buffer in full duplex systems. We believe that adaptive buffer sizing techniques for half-duplex systems could be reused in the full-duplex domain. However, in order to do so we must address several challenges in designing a buffer sizing method that is suitable for wireless full-duplex. Moreover, adaptive buffer sizing can be used to improve the energy efficiency of radio design.

The majority of AQM based buffer management techniques that we discuss previously in Sec. 2.3 are not initially designed for wireless networks. Hence, they are not capable to support various challenges related to the wireless medium such as adaptive transmission rate, frame aggregation, link scheduling *etc.* In the previous chapters, we present two buffer management methods that are specifically designed to comply with the requirements of wireless networks. We show that these techniques performs well over wireless half-duplex systems. Nevertheless, there is a lack in the literature about the suitable methods to manage the buffer of a full-duplex node. Moreover, the impact of such techniques on network metrics such as latency and throughput needs to be analyzed. Our primary goal in this chapter is to study the interaction between buffer management techniques and wireless full-duplex systems. To be able to analyze the performance of buffer management methods in a wireless full-duplex environment, we need to modify the buffer module in our wireless full-duplex testbed in order to stimulate the adaptive buffer management behavior.

In Chapter 5, we propose a queue management scheme for wireless networks called WQM. It uses an adaptive buffer sizing algorithm that estimates periodically the buffer draining time using current transmission rate, backlog queue and the number of neighboring nodes. In fact, WQM varies the buffer size depending on buffer draining time. Once this draining time exceeds a predefined value, an alarm is raised and the buffer size is going to be reduced. In comparison to CoDel and PIE, WQM demonstrates better performance using an ordinary IEEE 802.11n half-duplex testbed. The novelty of WQM lies in the fact that it is designed to address the unique challenges of buffer sizing in wireless networks [85]. This is what motivates us to implement WQM on top of our wireless full-duplex deployment with minor changes as detailed in the next section.

6.3 Implementation

We based our work on a simulation of full-duplex communication in a multi-hop environment using the discrete-event network simulator NS-3 [86]. In our implementation, we deploy Relay Full-Duplex MAC (RFD-MAC) protocol [87] by extending the Wi-Fi module in NS-3.20 [88]. RFD-MAC is a media access control protocol

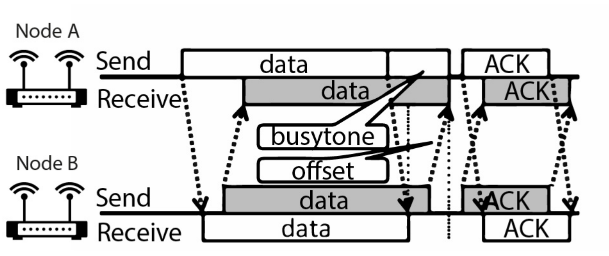


Figure 6.2: RFD-MAC time sequence.

that is specifically designed for relay full-duplexing. Multi-hop networks can handle bidirectional full-duplexing as well as relay full-duplexing. The difference between these two schemes is illustrated in Fig. 6.1. Bidirectional full-duplexing means that the wireless node is able to send and receive to/from the same node simultaneously. In Relay full-duplex scenario, node 2 can receive a frame from node 1 and forward a frame to node 3 at the same time. To maximize full-duplex capability, RFD-MAC must choose a secondary transmission node properly. For example, node 2 has two candidates for a secondary transmission node: nodes 1 and 3. If it selects a node that does not have a frame, then relay full-duplexing does not occur.

RFD-MAC takes into account the possibility of collisions between primary and secondary transmissions. A collision may occur at the receiver node when a primary transmission node is placed within the transmission range of a secondary transmission node and vice versa. Every node builds its own surrounding node table and exploits this table to choose a secondary transmission node. By doing so, RFD-MAC avoids collisions between primary and secondary transmissions. The algorithm of selecting the secondary node is based on a priority set. Whenever a node completes the transmission of a frame before the receiving is done, RFD-MAC uses a busytone until the reception is complete. Then, the primary and secondary transmission nodes exchange ACK frames to finalize the full-duplex transmission session as shown in Fig. 6.2. This scheme does not completely eliminate collisions because the receiver is susceptible to collisions until the reception of the packet header is complete.

We choose to implement our proposed method using NS-3 because it was built to improve the realism of the transmit stack of Linux based computers [89]. In the Wi-Fi module of NS-3, the internal queues of Wi-Fi Net Devices have different architecture compared to the traditional buffers used elsewhere. In fact, to implement WQM in NS-3, we need to modify the source files of the Wi-Fi module. The insight behind our work is to adaptively size the transmit packet queue and enforce upper and lower limits on its size. After doing the required analysis, we find that the maximum needed buffer size based on our network setup is 24 packets and the initial buffer size should be set to 2 packets. In WQM, the buffer size is not allowed to be less than one packet. We drew the attention of the reader that the default buffer size for Drop Tail in NS-3 is 400 packets. Also, it is important to mention that WQM does not operate at the physical layer. Hence, it is not going to be directly affected by the level of self-interference in the network.

We would like to note that our current implementation is slightly different from the original WQM implementation. First, in our implementation we use an Ad-Hoc Wi-Fi network based on IEEE 802.11a standard [90] in which the link rate can vary between 6 Mb/s and 54 Mb/s. All devices are configured with Adaptive Auto Rate Fallback (AARF) rate control algorithm [91] instead of Minstrel. AARF attempts to increase the transmission rate after a predefined number of successful transmissions at the current rate. To ensure the stability of the channel, AARF increases its success threshold before attempting to switch to a higher rate by remembering the number of failed probes. In case of two consecutive packet losses, it lowers the transmission rate one step and resets the success threshold to 10. In its original version, WQM is synchronized with Minstrel which is tuned every 100 ms. This is not the case for our current setup since AARF is based on packet transfer status. Thus, we choose to tune the buffer size for every incoming packet to the transmit queue. Second, the simulation of full-duplex communication [86] is not QoS-enabled and hence it doesn't

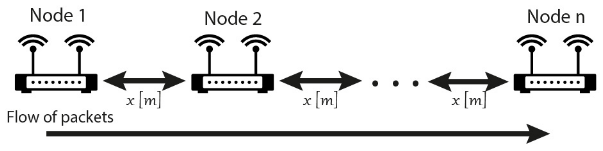


Figure 6.3: Topology of single flow scenario.

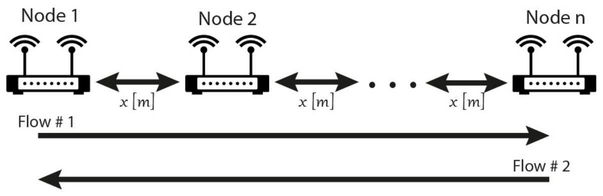


Figure 6.4: Topology of bidirectional flows scenario.

support frame aggregation. So, we modify the WQM algorithm to deal with disabled frame aggregation.

6.4 Performance evaluation

In this section, we compare the performance of WQM to Drop Tail mechanism in the full-duplex environment. We choose Drop Tail because it is the default buffering scheme in NS3. We test our implementation over two scenarios: single flow scenario as illustrated in Fig. 6.3 and bidirectional flows scenario as illustrated in Fig. 6.4. These topologies represent typical relay full-duplex networks in the Ad-Hoc mode. The distance between nodes in the network is fixed to 90 m. The routing protocol used in the simulation is Ad-hoc On Demand Distance Vector (AODV). We repeat all the experiments multiple times while varying the number of nodes in the network and the sender transmission rate. The simulation parameters are summarized in Table 6.1.

Prameter	Value
Wi-Fi Standard	IEEE 802.11a
Radio Band	5 GHz
Packet size	1500 Bytes
Default m_queue size	400 packets
Amount of data transferred	100 MB
Distance between nodes	90 m
Routing protocol	AODV
Rate control algorithm	AARF

Table 6.1: Simulation parameters summary.

6.4.1 Single flow scenario

As shown in Fig. 6.3, when the simulation starts, node 1 transmits a flow of packets towards node n, which is the last node in the network. This setup tries to mimic the transfer of a large file between nodes in the network. Every packet carries 1500 bytes worth of data. The simulation ends when node n receives 100 MB worth of bytes or when 1000 seconds is elapsed from the start of simulation, whichever happens first. We start collecting various network metrics after the receiver node receives the first 100 packets. In all our experiments, we vary the source rate gradually from 100 packets/s to 1000 packets/s. We repeat each experiment multiple times while varying the number of nodes in the network from three to five. To increase the reliability of our results, we run every experiment at least twice and report the average of the results. The end-to-end delay is shown in Fig. 6.5, goodput results are shown in Fig. 6.6 and the collision rate is shown in Fig. 6.7. We would like to note that we use the logarithmic scale for the y-axis in the latency figures to be able to show the difference between the two schemes.

As expected, WQM outperforms Drop Tail in terms of latency reduction. When the buffer is bloated, WQM reduces the latency by an average of 109$\times$ for the three nodes case, 208$\times$ for the four nodes case and 49$\times$ for the five nodes case. In compari-

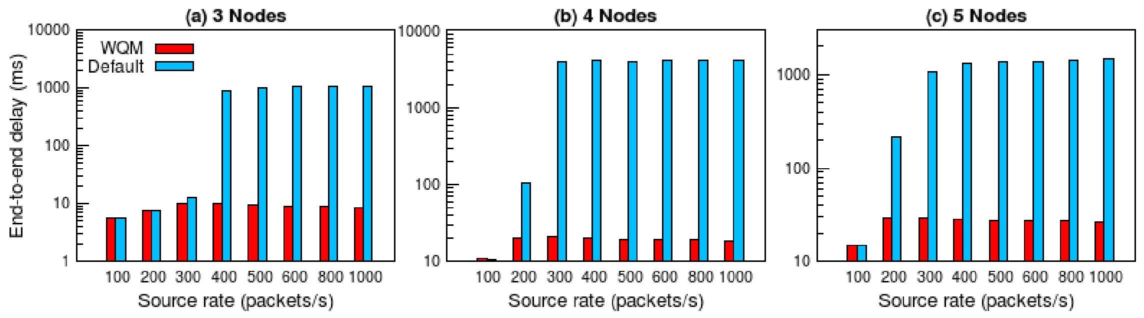


Figure 6.5: End-to-end latency while varying source rate for the single flow scenario.

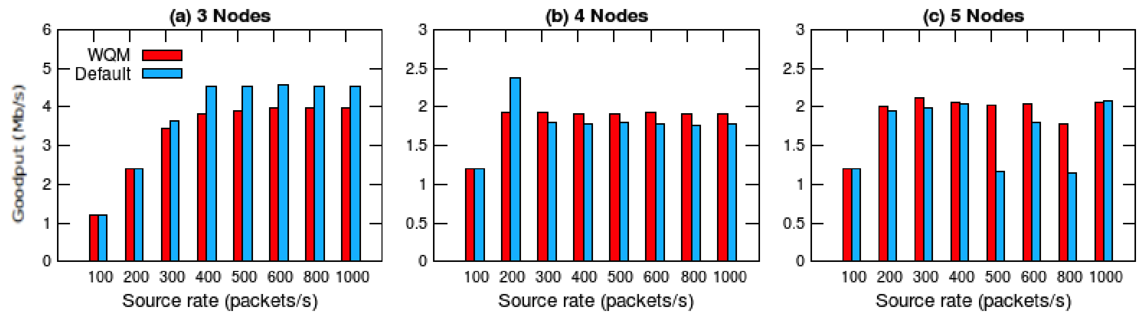


Figure 6.6: Goodput while varying source rate for the single flow scenario.

son to Drop Tail, WQM manages to reduce up to 99% of the encountered latency. For example, in the four nodes case and using a transmission rate of 800 packets/s, WQM reduces the end-to-end delay from 4094.51 ms to only 18.91 ms, which represents two orders of magnitude reduction. In fact, the default buffering scheme leads to latency in the order of seconds in all scenarios. When the source rate reaches more than 300 packets/s, the end-to-end latency reaches 1 s for the three nodes case, around 4 s for the four nodes case and approximately 1.5 s for the five nodes case. On the contrary, WQM maintains a delay less than 10.27 ms for the three nodes topology, less than 20.69 ms for the four nodes topology and less than 29.32 ms for the five nodes topology. It is obvious from these figures that selecting the optimal buffer size results in significant queuing delay reduction.

As shown in Fig. 6.6, the goodput results of WQM suffer from less variation compared to Drop Tail. In the case of three nodes topology, WQM drops the goodput by an average of 9% compared to the default buffering scheme in NS-3. However,

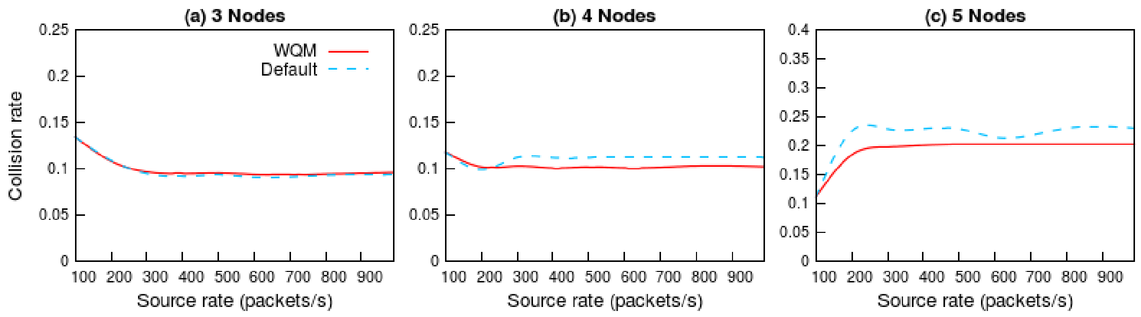


Figure 6.7: Collision rate while varying source rate for the single flow scenario.

when the number of hops is increased, WQM improves the network goodput. On average, it increases the goodput by 7.7% and 25% for the four and five nodes scenario respectively when the network is congested. Furthermore, We attribute this to the ability of WQM to reduce the collision rate between the primary and secondary transmissions when the number of hops is increased as illustrated in Fig. 6.7. In fact, as the source rate increases, the buffer in the case of Drop Tail fills up quickly leading to extra latency and higher collision rates. The bufferbloat point in our experiments is located between 300 packets/s and 400 packets/s which is very close to the default static buffer size in our implementation (400 packets). As expected, the situation gets worse when there are more nodes in the network. This is a clear proof that static buffers are not suitable for full-duplex wireless networks.

6.4.2 Bidirectional flows scenario

In this section, we evaluate the performance of WQM in the presence of bidirectional flows in the network. We run two 50 MB flows simultaneously in opposite directions between the edge nodes in the network as illustrated in Fig. 6.4. Similar to the single flow scenario, we vary the number of nodes from three up to five and vary the source rate gradually from 100 packets/s to 1000 packets/s. The end-to-end latency, goodput and collision rate between the primary and the secondary transmission are shown in Fig. 6.8, 6.9 and 6.10 respectively.

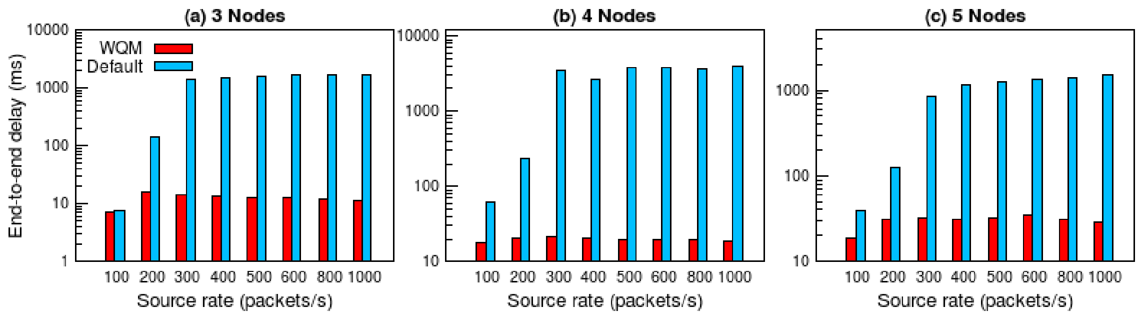


Figure 6.8: End-to-end latency while varying source rate for the bidirectional flows scenario.

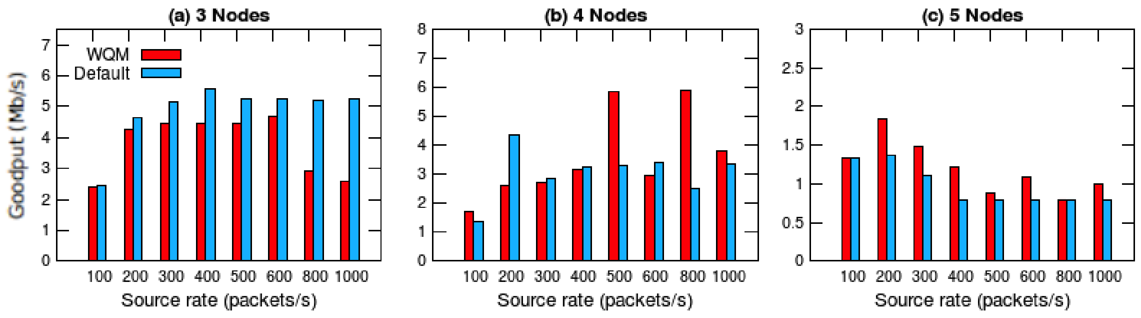


Figure 6.9: Goodput while varying source rate for the bidirectional flows scenario.

Similar to the single flow scenario, when the source rate is equal to or higher than 300 packets/s, WQM reduces network latency by an average of 126× for the three nodes case, 181× for the four nodes case and 40× for the five nodes case. For instance, when the network consists of three nodes and both the sources operate at 600 packets/s, WQM manages to drop the end-to-end latency from 1646.15 ms in Drop Tail case to only 12.61 ms which represents two orders of magnitudes delay reduction. This achievement comes at the cost of around 10% reduction in goodput. It could be noticed that when the sources are operating at high transmit rates, WQM suffers from 50% drop in goodput in the three nodes case. We attribute this to the high collision rate in this case as shown in Fig. 6.10.

As we increase the number of nodes in the network, we notice that WQM outperforms Drop Tail in terms of network goodput. In the five nodes case, the average increase in goodput is about 24%. We would like to note that in the four nodes

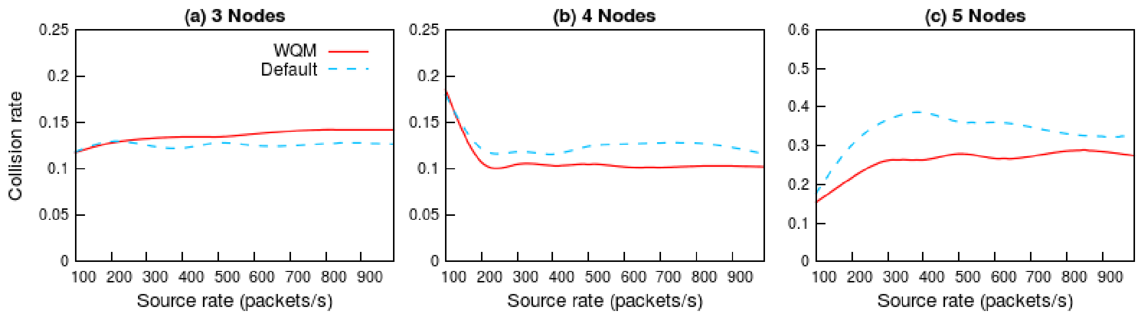


Figure 6.10: Collision rate while varying source rate for the bidirectional flows scenario.

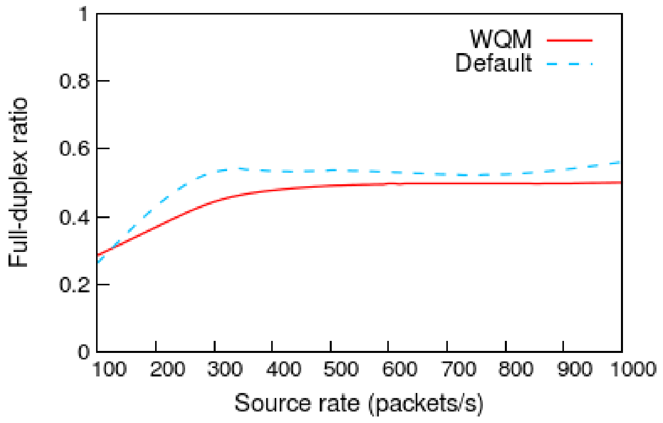


Figure 6.11: Full-duplex Ratio for the four nodes scheme.

topology, Drop Tail have slightly better goodput than WQM in several cases even though the latter achieves lower collision rate as shown in Fig. 6.10. We investigate this issue and find that WQM have inferior full-duplex ratio in this case as illustrated in Fig. 6.11. This reduction varies between 2.87% and 8.81% in comparison to Drop Tail and may limit the ability of WQM to utilize the drop in collision rate. Despite this limitation, WQM manages to enhance goodput from 2.48 Mbps to 5.92 Mbps when both sources are transmitting 800 packets/s.

The collision rate for both WQM and Drop Tail are shown in Fig. 6.10. For the three nodes case, WQM has higher collision rate compared to Drop Tail. On average, WQM suffers from 1.15% more collisions. This fact combined with limited buffer size may explain the significant drop in goodput mentioned before. However, with larger topologies, WQM achieves an average of 1.46% and 7.63% reduction in collision rate for four and five nodes topology, respectively. The ability of WQM to increase the goodput with larger networks compared to Drop Tail is undoubtedly a great achievement.

Overall, WQM keeps the end-to-end latency below 15.57 ms for the three nodes topology, below 20.81 ms for the four nodes topology and below 34.84 ms for the five nodes topology which is an acceptable delay for real time applications such as VoIP, online gaming, video streaming, *etc.* As mentioned earlier, WQM manages the buffer in an effective manner and prevents building up large buffers at the bottleneck links.

Chapter 7

Concluding Remarks

In the last few years, the world witnessed a dramatic growth in using mobile and wireless networks. Mobile devices such as smart phones and tablets are replacing the traditional desktops and becoming principal computing devices. Recent studies reveal that the number of wireless users in a any given country exceeds its own population [92]. As a result, improving the performance of wireless systems in terms of latency and capacity is extremely important.

Buffer sizing has extensively been studied in the context of core Internet routers handling a large number of flows. These networks have a large BDP. On the other hand, wireless networks usually have a much smaller BDP, translating to small buffer sizes. However, very small buffers may degrade the network utilization. In fact, configuring these buffers in commodity OS (such as Linux) is challenging because buffers are spread over multiple layers in the software stack. Moreover, new enhancements in the IEEE 802.11n/ac standard, such as frame aggregation, complicates this problem even further. We show that optimally sizing buffers is not only important for real-time traffic, but also for TCP flows sharing the bottleneck buffer as well. We classify wireless buffer sizing schemes into two categories: network centric and end-to-end centric methods. As shown in this dissertation, it is very difficult to have a single optimal buffer that suites all types of wireless networks. Through careful sizing of various buffers, the goal is to achieve high utilization of the bottleneck spectrum while maintaining short queueing delays. In this context, two buffer sizing schemes have been proposed, namely DNB and WQM. The former focuses on solving the buffer sizing for WMNs whereas the latter accounts for frame aggregation while deciding about the optimal buffer size to be used.

DNB introduces the concept of a neighborhood buffer. This buffer is going to be distributed over multiple nodes in order to determine the cumulative buffer size that can saturate the spectral resource constituting the network bottleneck. It uses heuristics for sizing the cumulative neighborhood buffer, and for distributing it amongst the contending nodes. Performance evaluation using simulations and testbed experiments show that DNB can effectively maintain high network utilization with short delay. There are interesting avenues for future work in this area. DNB distributes the neighborhood buffer among the nodes using a simple cost function where the cost of a packet drop along a path increases linearly with the hop count. With multi-rate hops along a path, this cost may not increase linearly. Other alternatives, such as dropping packets only on the first hop, may perform better. Further, DNB uses a loose upper bound on achievable network capacity to size the neighborhood buffer. A possible extension to DNB is to add an adaptive component in which the nodes in a radio neighborhood measure the current flow rates to estimate the network carrying capacity and adjust their allocated buffer sizes in response. This adaptive scheme would fare better in multi-flow networks as well as in wireless networks susceptible to large variability in channel noise and interference.

Enhancements in IEEE 802.11n/ac standards, such as frame aggregation, exacerbate the challenges of optimal buffer sizing in wireless networks. Large buffers may lead to long end-to-end delays in the order of seconds. To address this issue, we propose a practical, adaptive, and lightweight queue management scheme called WQM. It chooses the buffer size based on network load, channel condition, and frame aggregation level. WQM is implemented in Linux and tested on a wireless testbed.

We prove through experiments over various single-hop and multi-hop scenarios that WQM can reduce the end-to-end latency by a factor of 8× compared to the default Linux configuration. Further, WQM outperforms other state of the art bufferbloat solutions such as CoDel and PIE by a factor of 7× in terms of delay reduction. In the worst case, this reduction comes at the cost of 8% drop in goodput. Finally, we show that WQM improves flow fairness as it limits the ability of one of the flows to saturate the buffers. We are pursuing a number of interesting avenues for future work. We are currently evaluating WQM using a combination of both short and long-lived flows across multiple topologies. Further, we plan to evaluate WQM using flow separation as well as by replacing its drop tail approach with selective drop algorithms. Moreover, we would like to study the interaction between TCP pacing and frame aggregation and its consequences for bufferbloat in wireless networks. Also, it would be interesting to evaluate the effect of having an adaptive look-around interval in our algorithm. Finally, we would also like to test WQM using wireless devices with IEEE 802.11ac compatible radios.

Recently, wireless full-duplexing imposes itself as a reality turning on very promising area of research. We tackle the problem of buffer management in full-duplex systems and analyse the gains in terms of latency of implementing AQM on top of such systems. We prove through simulation over multiple scenarios that WQM can decrease latency in relay full-duplex networks by two orders of magnitudes. We believe that this opens a new research direction as it evaluates the interaction between buffer management and full-duplex design. In the future, we will consider other methods to evaluate WQM using real testbed such as Wearable Reference Platform (WaRP) boards and also test the performance using the most recent Wi-Fi standard. Further, we aim to design a novel buffering scheme for full-duplex devices that takes into consideration internal queues as well as ring buffers. Finally, we would like to investigate the effect of buffer management on the energy efficiency of full-duplex systems.

Chapter 8

Future Research Work

We believe that fixing bufferbloat at the wireless edge requires work along multiple lines, creating complementary solutions that, taken together, may address the myriad challenges described in this dissertation.

8.1 Frame aggregation schedulers

IEEE 802.11 standard specifications have left the design of A-MPDU aggregation schedulers open to vendor implementation, creating the space for well-designed schedulers that can balance the various performance tradeoffs in a wireless network. Our analysis shows that efficient design of A-MPDU aggregation schedulers can boost goodput while simultaneously reduce end-to-end delays [78]. This can be attributed to two factors: (1) Each A-MPDU includes a single PHY preamble and header, significantly reducing this overhead as these headers are usually transmitted at base rate for backward compatibility with 802.11 a/b/g nodes. (2) A single channel access can transmit as many as 64 subframes, and in response receive a single block ACK. However, even with aggregation enabled, RTT values can still exceed approximately 100 ms over a single wireless hop. We anticipate that the performance will deteriorate further in noisy radio environment as well as in multi-hop networks. These delays may potentially further exacerbate with the emerging 802.11ac standard which also supports A-MPDU frame aggregation with the following main differences from 802.11n:

1. Supports aggregates as large as 1 MBytes compared to 64 KBytes for 802.11n.
2. Always transmits frames as A-MPDUs, even if the sender is transmitting a single frame only.

We believe some of these challenges may be addressed through a cross-layer approach where the impact of frame aggregation scheduler is considered at multiple layers in the protocol stack. In particular, buffer sizing mechanisms may need to account for the current A-MPDU sizes, among other wireless channel characteristics. It may need to ensure that the wireless driver will always have sufficient packets in queue to maximize the gains that can be achieved through frame aggregation.

8.2 Wireless compatible active queue management

Buffer sizing and AQM algorithms may be considered as complementary solutions that can be used in conjunction. As such, the combined effect of the two schemes needs to be studied through both analyses and experimentation. Traditional AQM algorithms may fail in a wireless environment where the queue size may not always be the best indicator of network congestion. Newer algorithms, such as CoDel, address this challenge by using the packet sojourn time to interpret congestion. Thus, the optimal queue backlog is a function of the buffer drain time, and this varies in response to changing channel conditions. Analyzing and adapting the behavior of AQM algorithms with dynamic buffer sizing under this environment needs to be studied in more details.

8.3 Virtual queueing

The AP or BS transmits data to multiple client devices, each experiencing different channel conditions. As a result, the buffer size suitable for one client device may deteriorate the performance of another. One solution is to implement a per station virtual queue to segregate the traffic for different nodes. We believe that some variant of fair queuing is necessary to isolate the impact of one wireless device from the other. This can also help improve fairness between flows with different congestion window sizes.

8.4 Fine-tuning TCP

End-to-end solutions may be easier to deploy in controlled networks, such as cellular networks. This is particularly beneficial when the operator cannot access/configure bottleneck router buffers along the traffic route. The TCP stack on client devices can be modified through updates pushed out to smartphones locked by the operator. It is more beneficial to implement these changes at the client side (than at the BS) as the client has more accurate information about the last-hop wireless link.

REFERENCES

[1] M. Gast, *802.11n: A Survival Guide.* O'Reilly Media, Inc., 2012.

[2] IEEE LAN/MAN Standards Committee, *IEEE 802.11 Wireless LAN medium access control (MAC) and physical layer (PHY) specifications*, IEEE, 2012.

[3] C. Staff, "Bufferbloat: what's wrong with the internet?" *Commun. ACM*, vol. 55, no. 2, pp. 40–47, Feb. 2012.

[4] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: illuminating the edge network," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, ser. IMC '10, 2010, pp. 246–259.

[5] G. Linden, "Make data useful," 2006.

[6] S. Stefanov, "Exceptional website performance with YSlow 2.0," China Software Developers Network (CSDN), 2008.

[7] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," Internet Engineering Task Force, RFC 1323, May 1992. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1323.txt

[8] The network simulator - ns-2. http://www.isi.edu/nsnam/ns.

[9] Bufferbloat. http://www.bufferbloat.net/.

[10] M. Allman, "Comments on bufferbloat," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 1, pp. 30–37, Jan. 2012.

[11] A. Araldo and D. Rossi, "Bufferbloat: passive inference and root cause analysis," Telecom ParisTech, Tech. Rep. TECHREP-13b, 2013.

[12] J. Gettys and K. Nichols, “Bufferbloat: dark buffers in the internet,” *Commun. ACM*, vol. 55, no. 1, pp. 57–65, Jan. 2012.

[13] T. Cardozo, A. da Silva, A. Vieira, and A. Ziviani, “Bufferbloat systematic analysis,” in *Telecommunications Symposium (ITS), 2014 International*, Aug 2014, pp. 1–5.

[14] O. Hohlfeld, E. Pujol, F. Ciucu, A. Feldmann, and P. Barford, “A QoE perspective on sizing network buffers,” in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ser. IMC ’14, 2014.

[15] G. Appenzeller, I. Keslassy, and N. McKeown, “Sizing router buffers,” in *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM ’04, 2004, pp. 281–292.

[16] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, “Routers with very small buffers,” in *Proc. of the IEEE INFOCOM ’06*, Apr. 2006.

[17] G. Raina and D. Wischik, “Buffer sizes for large multiplexers: Tcp queueing theory and instability analysis,” in *Next Generation Internet Networks, 2005*, April 2005, pp. 173 – 180.

[18] K. Jamshaid, B. Shihada, L. Xia, and P. Levis, “Buffer sizing in 802.11 wireless mesh networks,” in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, Oct. 2011, pp. 272 –281.

[19] H. Jiang, Y. Wang, K. Lee, and I. Rhee, “Tackling bufferbloat in 3g/4g networks,” in *Proceedings of the 2012 ACM conference on Internet measurement conference*, ser. IMC ’12, 2012, pp. 329–342.

[20] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, “The impact of multihop wireless channel on TCP throughput and loss,” in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3, March 2003, pp. 1744–1753 vol.3.

[21] C. Villamizar and C. Song, “High performance TCP in ANSNET,” *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 5, pp. 45–60, Oct. 1994.

[22] D. Smithies and F. Fietkau, “Minstrel rate control algorithm,” http://wireless.kernel.org/en/developers/Documentation/mac80211/RateControl/minstrel.

[23] A. Dhamdhere and C. Dovrolis, “Open issues in router buffer sizing,” *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 87–92, Jan. 2006.

[24] D. Skordoulis, Q. NI, H. Chen, A. Stephens, C. Liu, and A. Jamalipur, “IEEE 802.11N MAC frame aggregation mechanisms for next-generation high-throughput WLANs,” *IEEE Wireless Communications*, pp. 40–47, February 2008.

[25] J. Friedrich, S. Frohn, S. Gubner, and C. Lindemann, “Understanding IEEE 802.11n Multi-hop Communication in Wireless Networks,” in *Workshop on Wireless Network Measurements*, May 2011, pp. 321–326.

[26] S. Byeon, K. Yoon, O. Lee, S. Choi, W. Cho, and S. Oh, “MoFA: Mobility-aware frame aggregation in Wi-Fi,” in *Proceedings of the Tenth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT ’14, 2014.

[27] Ath9k FOSS drivers. http://wireless.kernel.org/en/users/Drivers/ath9k.

[28] K. Jamshaid, “Centralized Rate Allocation and Control in 802.11-based Wireless Mesh Networks,” Ph.D. dissertation, University of Waterloo, Jan. 2010.

[29] K. Jamshaid, B. Shihada, A. Showail, and P. Levis, “Deflating link buffers in a wireless mesh network,” *Ad Hoc Networks*, vol. 16, no. 0, pp. 266 – 280, 2014.

[30] O. Dousse, “Revising buffering in CSMA/CA wireless multihop networks,” in *Proc. of the IEEE SECON ’07*, Jun. 2007.

[31] D. Xue and E. Ekici, “Optimal power allocation in multi-hop wireless networks with finite buffers,” in *Communications (ICC), 2011 IEEE International Conference on*, June 2011, pp. 1–5.

[32] R. Draves, J. Padhye, and B. Zill, “Routing in multi-radio, multi-hop wireless mesh networks,” in *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom ’04, 2004, pp. 114–128.

[33] Ath5k FOSS drivers. http://wireless.kernel.org/en/users/Drivers/ath5k.

[34] Madwifi: Multiband Atheros driver for WiFi. http://www.madwifi-project.org.

[35] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, Aug. 1993.

[36] K. Nichols and V. Jacobson, "Controlling queue delay," *Queue*, vol. 10, no. 5, pp. 20:20–20:34, May 2012.

[37] T. Hiland-Jrgensen, "Battling bufferbloat: An experimental comparison of four approaches to queue management in linux." *Project Report*, Dec. 2012. [Online]. Available: http://rudar.ruc.dk/handle/1800/9322

[38] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," in *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*, 2013.

[39] K. Jamshaid, P. Ward, M. Karsten, and B. Shihada, "The efficacy of centralized flow rate control in Wireless Mesh Networks," *EURASIP journal on Wireless Communications and Networking*, vol. 2013, no. 63, pp. 1–17, 2013.

[40] N. Khademi, D. Ros, and M. Welzl, "The new AQM kids on the block: Much ado about nothing?" Dept. of Informatics, University of Oslo, Norway, Tech. Rep. TR-434, October 2013.

[41] T. Herbert, "Byte queue limits," Linux Plumbers Conference, 2011.

[42] "Linux 3.3: Finally a little good news for bufferbloat." http://www.cringely.com/2012/03/25/linux-3-3-finally-a-little-good-news-for-bufferbloat/.

[43] K. Chen, Y. Xue, S. Shah, and K. Nahrstedt, "Understanding bandwidth-delay product in mobile ad hoc networks," *Elsevier Computer Communications*, vol. 27, no. 10, pp. 923–934, June 2004.

[44] T. Li, D. Leith, and D. Malone, "Buffer sizing for 802.11-based networks," *IEEE/ACM Transactions on Networking*, vol. 19, no. 1, pp. 156 –169, Feb. 2011.

[45] D. Taht, "What I think is wrong with eBDP in debloat-testing," https://lists.bufferbloat.net/pipermail/bloat-devel/2011-November/000280.html.

[46] R. Bruno, M. Conti, and E. Gregori, "Analytical modeling of TCP clients in Wi-Fi hot spot networks," in *Proc. of the IFIP Networking '04*, May 2004, pp. 626–637.

[47] B. Hull, K. Jamieson, and H. Balakrishnan, "Mitigating congestion in wireless sensor networks," in *Proc. of the ACM SenSys '04*, Baltimore, MD, Nov. 2004.

[48] K. Xu, M. Gerla, L. Qi, and Y. Shu, "Enhancing TCP fairness in ad hoc wireless networks using neighborhood RED," in *Proc. of the ACM MobiCom '03*, Sep. 2003, pp. 16–28.

[49] M. Thottan and M. C. Weigle, "Impact of 802.11e EDCA on mixed TCP-based applications," in *Proceedings of the 2nd annual international workshop on Wireless internet*, ser. WICON '06, 2006.

[50] C. Boutremans and J.-Y. Le Boudec, "A note on the fairness of TCP Vegas," in *Broadband Communications, 2000. Proceedings. 2000 International Zurich Seminar on*, 2000, pp. 163–170.

[51] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic," in *Proceedings of the conference on Communications architecture & protocols*, ser. SIGCOMM '91, 1991, pp. 133–147.

[52] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of TCP pacing," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, Mar 2000, pp. 1157 –1165 vol.3.

[53] S. ElRakabawy and C. Lindemann, “A practical adaptive pacing scheme for TCP in multihop wireless networks,” *Networking, IEEE/ACM Transactions on*, vol. 19, no. 4, pp. 975 –988, Aug. 2011.

[54] Y. Xu, Y. Wang, J. Lui, and D.-M. Chiu, “Balancing throughput and fairness for TCP flows in multihop ad-hoc networks,” in *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks and Workshops, 2007. WiOpt 2007. 5th International Symposium on.* IEEE, 2007, pp. 1–10.

[55] A. Warrier, S. Janakiraman, S. Ha, and I. Rhee, “DiffQ: Practical differential backlog congestion control for wireless networks,” in *INFOCOM 2009, IEEE*, 2009, pp. 262–270.

[56] A. Showail, K. Jamshaid, and B. Shihada, “WQM: An aggregation-aware queue management scheme for IEEE 802.11n based networks,” in *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, ser. CSWS ’14, 2014, pp. 15–20.

[57] S. Chan, K. Chan, K. Liu, and J. Lee, “On queue length and link buffer size estimation in 3G/4G mobile data networks,” *Mobile Computing, IEEE Transactions on*, vol. 13, no. 6, pp. 1298–1311, June 2014.

[58] Shuttle Inc. http://www.shuttle.com.

[59] The web100 project. http://www.web100.org/.

[60] Open80211s. http://open80211s.org.

[61] J. Camp and E. Knightly, “The IEEE 802.11s extended service set mesh networking standard,” *Communications Magazine, IEEE*, vol. 46, no. 8, pp. 120–126, 2008.

[62] Iperf. http://dast.nlanr.net/Projects/Iperf/.

[63] S. Ha, I. Rhee, and L. Xu, “CUBIC: a new TCP-friendly high-speed TCP variant,” *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[64] The Web10g Project. http://www.web10g.org/.

[65] Netperf. http://www.netperf.org/netperf/.

[66] V. Gambiroza, B. Sadeghi, and E. Knightly, "End-to-end performance and fairness in multihop wireless backhaul networks," in *Proc. of the ACM MobiCom '04*, Sep. 2004, pp. 287–301.

[67] J. Jun and M. L. Sichitiu, "The nominal capacity of wireless mesh networks," *IEEE Wireless Communications*, pp. 8–14, Oct. 2003.

[68] J. Padhye, S. Agarwal, V. N. Padmanabhan, L. Qiu, A. Rao, and B. Zill, "Estimation of link interference in static multi-hop wireless networks," in *Proc. of the ACM/USENIX IMC '05*, Oct. 2005, pp. 305–310.

[69] A. Kashyap, U. Paul, and S. R. Das, "Deconstructing interference relations in WiFi networks," in *Proc. of the IEEE SECON '10*, Jun. 2010, pp. 73–81.

[70] H. Balakrishnan, C. L. Barrett, V. S. A. Kumar, M. Marathe, and S. Thite, "The distance-2 matching problem and its relationship to the MAC-layer capacity of ad hoc networks," *IEEE Journal on Selected Areas in Communication*, vol. 22, no. 6, pp. 1069–1079, Aug. 2004.

[71] L. Qiu, Y. Zhang, F. Wang, M. K. Han, and R. Mahajan, "A general model of wireless interference," in *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, ser. MobiCom '07, 2007, pp. 171–182.

[72] S. M. ElRakabawy, A. Klemm, and C. Lindemann, "TCP with adaptive pacing for multihop wireless networks," in *Proc. of the ACM MobiHoc '05*, May 2005, pp. 288–299.

[73] T. Szigeti and C. Hattingh, *End-to-end QoS network design: Quality of Service in LANs, WANs, and VPNs*, 1st ed. Cisco Press, 2004.

[74] A. J. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proc. of the ACM SIGCOMM '89*, Sep. 1989, pp. 1–12.

[75] S. Ganguly, V. Navda, K. Kim, A. Kashyap, D. Niculescu, R. Izmailov, S. Hong, and S. R. Das, “Performance optimizations for deploying VoIP services in mesh networks,” *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 11, pp. 2147 –2158, 2006.

[76] K. Jamshaid, P. Ward, and M. Karsten, “Mechanisms for centralized flow rate control in 802.11-based wireless mesh networks,” *Computer Networks*, vol. 56, no. 2, pp. 884–901, February 2012.

[77] WQM source code. http://www.shihada.com/download.php?file=wqm.zip.

[78] A. Showail, K. Jamshaid, and B. Shihada, “An empirical evaluation of bufferbloat in IEEE 802.11n wireless networks,” in *IEEE Wireless Communications and Networking Conference (WCNC)*, April 2014, pp. 3088–3093.

[79] J. I. Choi, M. Jain, K. Srinivasan, P. Levis, and S. Katti, “Achieving single channel, full duplex wireless communication,” in *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, ser. MobiCom ’10, 2010.

[80] M. Jain, J. I. Choi, T. Kim, D. Bharadia, S. Seth, K. Srinivasan, P. Levis, S. Katti, and P. Sinha, “Practical, real-time, full duplex wireless,” in *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom ’11, 2011, pp. 301–312.

[81] M. Duarte, A. Sabharwal, V. Aggarwal, R. Jana, K. Ramakrishnan, C. Rice, and N. Shankaranarayanan, “Design and characterization of a full-duplex multi-antenna system for WiFi networks,” *Vehicular Technology, IEEE Transactions on*, vol. 63, no. 3, pp. 1160–1177, March 2014.

[82] S. S. Hong, J. Mehlman, and S. Katti, “Picasso: Flexible RF and spectrum slicing,” in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM ’12, 2012, pp. 37–48.

[83] D. Bharadia, E. McMilin, and S. Katti, “Full duplex radios,” in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM ’13, 2013, pp. 375–386.

[84] D. Bharadia and S. Katti, “Full duplex MIMO radios,” in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI’14, 2014, pp. 359–372.

[85] A. Showail, K. Jamshaid, and B. Shihada, “Buffer sizing in wireless networks: Challenges, solutions, and opportunities,” *IEEE Communications Magazine*, Accepted, 2014. [Online]. Available: http://hdl.handle.net/10754/348555

[86] Implement wireless full-duplex communication in the ns-3 network simulator. https://github.com/yusuke-sugiyama/ns-3-fdwifi, accessed April, 2015.

[87] K. Tamaki, H. Ari Raptino, Y. Sugiyama, M. Bandai, S. Saruwatari, and T. Watanabe, “Full duplex media access control for wireless multi-hop networks,” in *Vehicular Technology Conference (VTC Spring), 2013 IEEE 77th*, June 2013, pp. 1–5.

[88] Wi-Fi Model Library - ns-3. https://www.nsnam.org/docs/models/html/wifi.html, accessed April, 2015.

[89] G. F. Riley and T. R. Henderson, “The ns-3 network simulator.” in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Gnes, and J. Gross, Eds. Springer, 2010, pp. 15–34.

[90] “IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (MAC) and physical layer (PHY) specifications,” *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1–2793, March 2012.

[91] M. Lacage, M. H. Manshaei, and T. Turletti, “IEEE 802.11 rate adaptation: A practical approach,” in *Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWiM ’04, 2004, pp. 126–134.

[92] CTIA-The Wireless Association. http://www.ctia.org/.

APPENDICES

A Papers Published and Under Review

Journal Publications

• A. Showail, K. Jamshaid, and B. Shihada, “Buffer Sizing in Wireless Networks: Challenges, Solutions, and Opportunities”, *IEEE Communication Magazine*, to appear, 2015. I.F. (4.46)

• A. Daghistani, A. Ben Khalifa, A. Showail, and B. Shihada, “Green Partial Packet Recovery in Wireless Sensor Networks, *Journal of Network and Computer Applications*, SI: Green Network Protocols and Algorithms, to appear, 2015. I. F. (2.229)

• A. Showail, and B. Shihada, “Batteling Bufferbloat in Wi-Fi Based Networks”, *IEEE Transactions on Networking (TON)*, under review, 2015.

• A. Showail, A. Elrasad, A. Meer, A. Daghistani, K. Jamshaid, and B. Shihada, “iFrag: Interference-Aware Frame Fragmentation Scheme for Wireless Sensor Networks”, *ACM journal of Wireless Networks*, Vol. 20, No. 4, pp. 1-18, 2014.

• K. Jamshaid, B. Shihada, A. Showail, and P. Levis, “Deflating Link Buffers in a Wireless Mesh Network”, *journal of Wireless AdHoc Networks*, Vol. 16, pp. 266-280, 2014. I. F. (1.957).

Conference Proceedings

• N. Bouacida, A. Showail, and B. Shihada, "Buffer Management in Wireless Full-Duplex Systems", *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications.*, accepted, 2015.

• M. Alaslani, A. Showail, and B. Shihada, "Green Frame Aggregation Scheme for Wi-Fi Networks", *IEEE International Conference on High Performance Switching and Routing (HPSR)*, accepted, 2015.

• A. Showail, K. Jamshaid, and B. Shihada, "WQM: An Aggregation-aware Queue Management Scheme for IEEE 802.11n based Networks", in Proc. *ACM Sigcomm Capacity Sharing Workshop (CSWS)*, pp. 15-20, 2014.

• A. Showail, K. Jamshaid, and B. Shihada, "An Empirical Evaluation of Bufferbloat in IEEE 802.11n Wireless Networks", in Proc. *IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 3088-3093, 2014.

Patent

• B. Shihada and A. Showail, "Buffer sizing for multi-hop networks", Continuing U.S. Provisional Patent no. US20140140209 A1. 2014.