Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

MASTER THESIS

Achieving Low Delay & High Rate in 802.11ac Edge Networks

HAMID HASSANI
School of Computer Science and Statistics
Trinity College Dublin

Supervisor: **Prof. DOUGLAS LEITH**

A thesis submitted in fulfillment of the requirements for the degree of Master by Research

2020

Declaration of Authorship

"As you start to walk on the way, the way appears."

Jalal ad-Din Muhammad Rumi

TRINITY COLLEGE DUBLIN

Abstract

School of Computer Science and Statistics

Master by Research

Achieving Low Delay & High Rate in 802.11ac Edge Networks

by **HAMID HASSANI**

Provision of connections with low end-to-end latency is one of the most challenging requirements in 5G. In most use cases the target is for $<$ 100ms latency, while for some applications it is $<$ 10ms. In part, this reflects the fact that low latency is already coming to the fore in network services, but the requirement for low latency also reflects the needs of next generation applications such as augmented reality, virtual reality and the tactile internet.

In this thesis we analyze the end-to-end latency in an edge network where an 802.11ac wireless hop is the bottleneck and queueing delay at the AP is the main source of latency. We demonstrate that queueing delay is coupled to the aggregation level in 802.11ac WLANs and that we can manage the delay by controlling the aggregation level. We implement this algorithm with a simple feedback loop on Linux using MAC timestamps. We also propose and implement a machine learning technique to infer aggregation level from kernel timestamps on Android OS where we do not have access to MAC timestamps. We demonstrate that the aggregation-based rate control policy selects a rate between that of Cubic and BBR. Importantly, the end-to-end one-way delay is more than 20 times lower than that with Cubic and BBR while it induces very few losses. We also propose a passive technique using logistic regression to detect the location of the path bottleneck, i.e. whether the bottleneck is the backhaul link or the wireless hop, and show how measurement of the aggregation level can be used for this purpose. We show that this approach has more than 90% accuracy across a range of different network configurations.

Acknowledgements

I would like to express my deepest appreciation to my supervisor, Prof. Douglas Leith, for the unparalleled support, valuable advice, and extensive knowledge. His willingness to give his time so generously has been very much appreciated.

I gratefully acknowledge Dr. Francesco Gringoli, at University of Brescia, Italy, for his helpful collaboration and constructive advice to conduct this research.

I would also like to extend my deepest gratitude to my parents and to my sister for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them.

In addition, there are my friends in the laboratory, Pavlos, Kariem and Apostolos, who were of great support in deliberating over our problems and findings, as well as providing happy distraction to rest my mind outside of my research during these years.

My special thanks should also go to Mr. M. Nazeri for his valuable guidance.

Finally, I would like to thank Aisling, Barney, Darragh 😎, and Niamh 😄, the lovely Irish family that I lived with during my studies abroad, for sharing me with the warm and welcoming atmosphere at home.

This research was supported by Science Foundation Ireland under Grant No. 13/RC/2077.

Contents

List of Figures

Chapter 1

Introduction

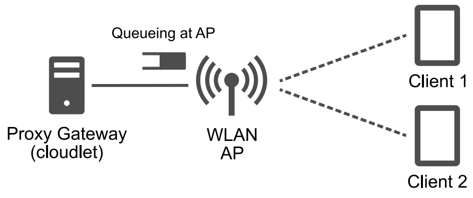
While much attention in 5G has been focussed on the physical and link layers, it is increasingly being realized that a wider redesign of network protocols is also needed in order to meet 5G requirements. Transport protocols are of particular relevance for end-to-end performance, including end-to-end latency. For example, ETSI has recently set up a working group to study next generation protocols for 5G [1]. The requirement for major upgrades to current transport protocols is also reflected in initiatives such as Google QUIC [2], Coded TCP [3] and the Open Fast Path Alliance [4].

In this thesis we consider next generation edge transport architectures of the type illustrated in Figure 1.1(a). Traffic to and from client stations is routed via a proxy located close to the network edge (e.g. within a cloudlet). This creates the freedom to implement new transport layer behavior over the path between proxy and clients, which in particular includes the last wireless hop. One great advantage of this architecture is its ease of rollout since the new transport can be implemented as an app on the clients and no changes are required to existing servers.

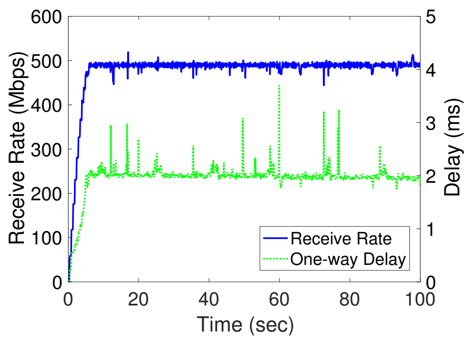
Our interest is in achieving high rate, low latency communication. One of the most challenging requirements in 5G is the provision of connections with low end-to-end latency. In most use cases the target is for <100ms latency, while for some applications it is <10ms [5, Table 1]. In part, this reflects the fact that low latency is already coming to the fore in network services, for example Amazon estimates that a 100ms increase in delay reduces its revenue by 1% [6], Google measured a 0.74% drop in web searches when delay was artificially increased by 400ms [7] while Bing saw a 1.2% reduction in per-user revenue when the service delay was increased by 500ms [8]. However, the requirement for low latency also reflects the needs of next generation applications such as augmented reality and the tactile internet.

As can be seen from Figure 1.1(a), the transmission delay of a packet sent over the downlink is composed of two main components: (i) queueing delay at the AP and (ii) MAC channel access time. The MAC channel access time is determined by the WLAN channel access mechanism and is typically small, so the main challenge is to regulate the queueing delay. We would like to select a downlink send rate which is as high as possible yet ensures that a persistent queue backlog does not develop at the AP.

While measurements of one-way delay might be used to infer the onset of queueing and adjust the send rate, measuring one-way is known



(a)



(b)

FIGURE 1.1: (a) Cloudlet-based edge transport architecture with bottleneck in the WLAN hop (therefore queueing of downlink packets occurs at the AP as indicated on schematic) and (b) Illustrating low-latency high-rate operation in an 802.11ac WLAN (measurements are from a hardware testbed located in an office environment, see Section 4.6).

to be challenging[1] as is inference of queueing from one-way delay[2]. Use of round-trip time (RTT) to estimate the onset of queueing is also known to be inaccurate when there is queueing in the reverse path. In this thesis we avoid these difficulties by using measurements of the aggregation level of the frames transmitted by the AP. Use of aggregation is ubiquitous in modern WLANs since it brings goodput near to line-rate by reducing the relative time spent in accessing the channel when transmitting several packets to the same destination. As we will see, the number of packets aggregated in a frame is relatively easy to measure accurately and reliably at the receiver. Intuitively, the level of aggregation is coupled to queueing. Namely, when only a few packets are queued then there are not enough to allow large aggregated frames to be assembled for transmission. Conversely, when there is a persistent queue backlog then there is a plentiful supply of packets and large frames can be consistently assembled. We show that by regulating the downlink send rate so as to maintain an appropriate aggregation level it is possible to avoid queue build up at the AP and so to realize high rate, low latency communication in a robust and practical fashion.

Figure 1.1(b) shows typical results obtained by regulating the aggregation level. These measurements are from a hardware testbed located in an office environment. It can be seen that the one-way delay is low, at around 2ms, while the send rate is high, at around 500Mbps (this data is for an 802.11ac downlink using two spatial streams and MCS 9 [9]). Increasing the send rate further leads to sustained queueing at the AP and an increase in delay, but the results in Figure 1.1(b) illustrate the practical feasibility of operation in the regime where the rate is maximized but the onset of sustained queueing is avoided.

[1]For example, due to clock offset and skew between sender and receiver.

[2]For example, the transmission delay across a wireless hop can change significantly over time depending on the number of active stations, e.g. if a single station is active and then a second station starts transmitting the time between transmission opportunities for the original station may double, and it is difficult to distinguish changes in delay due to queueing and changes due to factors such as this.

1.1 Contributions

The main contributions of this work are as follows:

1. Analyzing end-to-end delay in 802.11ac edge networks using aggregation
2. Proposing a machine learning approach to infer aggregation level using kernel timestamps on non-rooted Android devices
3. Introducing a novel transport layer approach for achieving high rate, low delay in 802.11ac edge networks using aggregation
4. Designing a logistic regression model to detect the bottleneck location
5. Modeling aggregation and delay behavior at access points in 802.11ac edge networks
6. Proposing a proportional fair low delay rate allocation scheme in 802.11ac edge networks using aggregation
7. Constructing a nonlinear inner-outer feedback controller to achieve low delay and high rate in 802.11ac edge networks

Some of the NS3 simulations presented in this thesis have been carried out by Prof. Francesco Gringoli from University of Brescia, Italy.

1.2 Structure

The structure of this thesis is as follows:

- In Chapter 2 we provide a brief overview of some of the concepts and technical terms that are used throughout this thesis.
- In Chapter 3 we review the literature in userspace transport protocols, congestion control and network monitoring methods in wireless networks and address the limitations of proposed algorithms.
- In Chapter 4 we analyze the end-to-end delay in 802.11ac edge networks where the bottleneck lies in the last wireless hop. We show that the aggregation in 802.11ac is coupled to delay and can be measured precisely using MAC timestamps. We also propose a machine learning approach to infer aggregation level using kernel timestamps on non-rooted Android devices. We then introduce a novel transport layer approach based on aggregation to have low delay and high rate communications in 802.11ac edge networks. We evaluate the performance of proposed algorithm in NS3 network simulator and implement it in our testbed on Linux and Android and compare the performance to Cubic and BBR.
- In Chapter 5 we model the aggregation and delay behavior at access points in an edge network where an 802.11ac wireless hop is the bottleneck. We propose a proportional fair rate allocation algorithm

to achieve high rate while maintaining low delay using aggregation in form of a convex optimization problem. We then construct a non-linear inner-outer feedback controller to solve the online solution of the optimization problem and find the optimum send rates. Following analyzing the stability of the proposed closed-loop controller, we implement and evaluate the performance on NS3 and in our testbed with multiple Linux and Android clients.

- In Chapter 6 the concluding remarks are given along with a discussion on limitations and future work.

1.3 Publications

The following journal and conference papers are under submission during the course of this master by research program.

1. Hamid Hassani, Francesco Gringoli, Douglas J. Leith, "Quick & Plenty: Achieving Low Latency & High Rate in 802.11ac Edge Networks", Submitted to IEEE Transactions on Mobile Computing.
2. Hamid Hassani, Francesco Gringoli, Douglas J. Leith, "Regulating Queueing Delay in 802.11ac WLANs: Nonlinear Controller Analysis & Design", Submitted to IEEE Transactions on Control Systems Technology.
3. Hamid Hassani, Francesco Gringoli, Douglas J. Leith, "Analyzing Delay in 802.11ac Edge Networks", Prepared to be Submitted to ICC 2020 Conference.

Chapter 2

Background

In this chapter, we provide an overview of the preliminary knowledge of networking that would be help understanding the concepts that we present in this thesis.

2.1 Brief Overview of 802.11ac

Wireless LANs based on the 802.11 family of standards are ubiquitous at the network edge. They use CSMA/CA to share access to the wireless channel amongst the stations in a WLAN. Briefly, in CSMA/CA each station maintains a contention window (CW) variable. Time is slotted and when a station wishes to transmit it initializes a counter to a value uniformly at random in interval [0, CW - 1], decreases this counter for each slot that the channel is detected to be idle and then transmits once the counter reaches zero. This countdown is paused when the channel is detected to be busy, and so the MAC slots at which the counter is decremented are variable size and tend to become longer as more stations share the channel.

Most modern WLANs now use 802.11ac, an extension of 802.11n that allows use of MIMO (transmission and reception using multiple antennas), higher modulation and coding scheme (MCS) rates (the rates at which data in a wireless frame is transmitted) and wider wireless channel widths of up to 80MHz. With three antennas, a transmitter can use up to three spatial streams for transmission i.e. the number of spatial streams (NSS) can vary from 1 to 3. The MCS rates are indexed form 0 to 9, e.g. with an 80MHz channel and NSS = 3 the MSC rates vary from 87 to 1170Mbps.

2.2 Aggregation in 802.11n, 802.11ac etc

Since 2009, when the 802.11n standard was introduced, all WLANs make use of aggregation to improve efficiency. That is, transmitted frames may carry multiple packets, thereby amortizing the fixed PHY/MAC overheads over multiple packets. This is essential for achieving high throughputs. Since the PHY overheads are largely of fixed duration, increasing the data rate reduces the time spent transmitting the frame payload but leaves the PHY overhead unchanged. Hence, the efficiency, as measured by the ratio of the time spent transmitting user data to the time spent transmitting an 802.11 frame, decreases as the data rate increases unless the frame payload is also increased, i.e. several data packets are aggregated and transmitted in a single frame. To facilitate aggregation packets destined for different stations are queued separately at the transmitter. When a transmission opportunity occurs a frame is typically formed by aggregating the queued packets, up to the maximum allowed level N_{max} (typically 64 packets or 5.5ms frame duration, whichever is smallest), and so the level of aggregation used is closely linked to the size of the queue backlog at the transmitter.

2.3 Edge Computing

Increasing demand for low delay and high rate is encouraging movement of latency-critical services from the cloud to the network edge. This architecture, referred as to edge computing, is considered to be a key technology for realizing various visions for the next-generation Internet, such as Tactile Internet and Internet of Things (IoT) [10]. In edge computing, computational and data storage resources are located close to the end users in order to decrease the propagation delay and save bandwidth.

In this paradigm, a cloudlet is a concept used for the network equipment located at the network edge which provides computing resources for use by applications. A cloudlet can be viewed as a small cloud data center used for achieving low end-to-end delay and high rate communications for edge services, such as virtual reality and augmented reality that offload computation to the cloudlet [11].

Chapter 3

Literature Review

In this chapter, we discuss some of userspace transport protocols and congestion control mechanisms. We also review algorithms proposed for queue build-up detection, capacity measurement and bottleneck detection in wireless networks.

3.1 Userspace Transport Protocols

In recent years there has been an upsurge in interest in userspace transports due to their flexibility and support for innovation combined with ease of rollout. This has been greatly facilitated by the high efficiency possible in userspace with the support of modern kernels. Notable examples of new transports developed in this way include Google QUIC [2], UDT [12] and Coded TCP [3, 13, 14]. ETSI has also recently set up a working group to study next generation protocols for 5G [1]. The use of performance enhancing proxies, including in the context of WLANs, is also not new e.g. RFC3135 [15] provides an entry point into this literature. However, none of these exploit the use of aggregation in WLANs to achieve high rate, low delay communication.

Interest in using aggregation in WLANs pre-dates the development of the 802.11n standard in 2009 but has primarily focused on analysis and design for wireless efficiency, managing loss etc. For a recent survey see for example [16]. The literature on throughput modeling of WLANs is extensive but much of it focuses on so-called saturated operation, where transmitters always have a packet to send, see for example [17] for early work on saturated throughput modeling of 802.11n with aggregation. When stations are not saturated (so-called finite-load operation) then for WLANs which use aggregation (802.11n and later) most studies resort to the use of simulations to evaluate performance due to the complex interaction between arrivals, queueing and aggregation with CSMA/CA service. Notable exceptions include [18, 19] which adopt a bulk service queueing model that assumes a fixed, constant level of aggregation and [20] which extends the finite load approach of [21] for 802.11a/b/g but again assumes a fixed level of aggregation.

3.2 Detecting Queue Build-up

While measurement of round-trip time is relatively straightforward since it uses the same clock (at the sender) to measure when a packet is transmitted and when its corresponding acknowledgement is received[1], due to queueing in the reverse path and delay asymmetry between forward and reverse paths the measured RTT can be an unreliable indicator of queueing in the forward path, see for example [22]. Furthermore, using RTT to detect queueing is known to give inaccurate results in 802.11 networks [23], due to ACK aggregation strategy, aggregation in uplink etc. Accurately measuring one-way delay is also known in general to be challenging[2].

Recently, [24] proposes an elegant Ping-Pair method for detecting queue build up in 802.11ac APs. In this approach, which is now used by Skype, a wireless client sends a pair of back-to-back ICMP echo requests to the AP with high and normal priorities specified by DSCP value, respectively. The high priority packets are queued separately from the normal priority packets at the AP, and serviced more quickly. Hence, the difference in RTTs between the two pings can provide an indication of queueing at the AP and [24] proposes thresholding this difference based on a predefined threshold inferred from a decision tree classifier in order to detect the queue build-up. However, this active probing approach creates significant load on the network and so itself can cause queue buildup. For example, Figure 3.1(a) plots the measured one-way delay and loss with and without ping-pairs as the send rate at which UDP data packets are sent from server to client is varied. Figure 3.1(b) plots the corresponding goodput (the rate at which packets are received at the client, i.e. after queue overflow losses at the AP). It can be seen that use of ping-pairs causes packet loss to start to occur for send rates above 200Mbps compared to send rates above 400Mbps without packet pairs, and similarly the one-way delay is roughly doubled with ping-pairs for send rates above 400Mbps.

3.3 Congestion Control

In computer networks, congestion control is needed when network capacity is less than the resource demands [25]. Various congestion control algorithms have been proposed since 1988 [26] which mostly use two indicators to detect congestion in the network: loss and delay.

3.3.1 Loss-based Congestion Control

In loss-based congestion control protocols, the congestion window size (cwnd) is reduced following one or more packet losses then increased using a window growth function. For example, TCP NewReno [27],

[1]Although still subject to "noise" due to use of client powersave, interrupt mitigation by the network interface, TSO and other offload to the NIC, kernel scheduling delays, and so on.

[2]The impact of clock offset and skew between sender and receiver applies to all network paths. In addition, when a wireless hop is the bottleneck then the transmission delay can also change significantly over time depending on the number of active stations e.g. if a single station is active and then a second station starts transmitting the time between transmission opportunities for the original station may double.

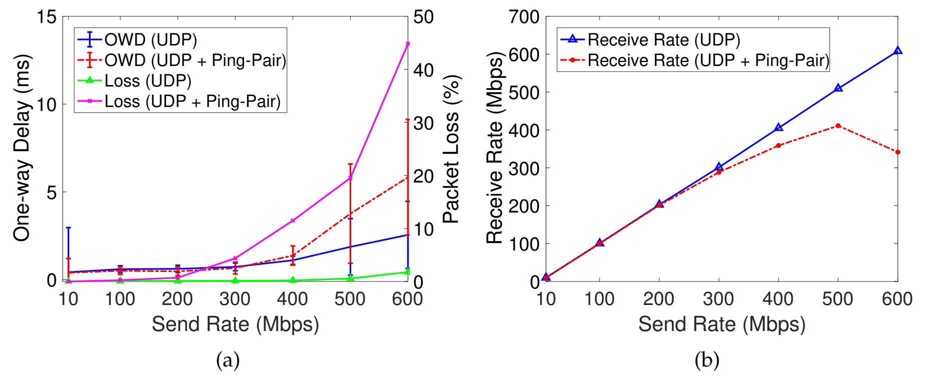


FIGURE 3.1: Performance of the Ping-Pair algorithm [22] in an 802.11ac WLAN as the send rate of a downlink UDP flow to a mobile handset is varied. Experimental data, setup in Section 4.6, AMDSU aggregation.

adopted by Google QUIC [2], uses the received ACKs to detect the packet loss and reduce the send rate. Then the growth function of TCP NewReno climbs linearly until detecting the next congestion event [28]. TCP Cubic [29] is the current default congestion control algorithm in Linux. TCP Cubic adjusts cwnd based on a cubic function of time since the last congestion event rather than RTT. H-TCP [30] also uses the elapsed time since the last packet loss event and also the ratio of minimum and maximum RTTs to adjust the send rate. However, the window growth function of H-TCP is a quadratic function of the elapsed time since the last congestion event.

3.3.2 Delay-based Congestion Control

In delay-based congestion control mechanisms, round-trip time is used to estimate the onset of queueing and adjust the send rate. As we discussed in Section 3.2, measurement of round-trip time can be unreliable indicator of queueing in the forward path. For example, TCP Vegas [31] uses the minimum observed RTT during the connection as the base delay then estimates the queue size using the current cwnd and the RTT of previous packet. A similar concept is applied in other variants of TCP such as the TCP Veno [32] and TCP FAST [33].

TCP BBR [34] is currently being developed by Google to achieve high rate and low delay communications.The BBR algorithm updates the minimum observed RTT in specific probing intervals and tries to estimate the bottleneck bandwidth and adapt the send rate accordingly to try to avoid queue buildup. The delivery rate in BBR is defined as the ratio of the in-flight data when a packet departed the server to the elapsed time when its ACK is received. This may be inappropriate, however, when the bottleneck is a WLAN hop since aggregation can mean that increases in rate need not correspond to increases in delay plus a small queue at the AP can be beneficial for aggregation and so throughput. Currently, Google is developing the next version of BBR to address some issues of the earlier version such as weak performance in

802.11ac WLANs reported in the BBR community forum [35] and also observed in our testbed, see Section 4.4.5.

3.4 Measuring Network Capacity

A variety of models have been proposed for measuring end-to-end network capacity. Pathchar [36] (and its Linux reimplementation Pchar [37]) sends probe packets with different sizes and TTLs then waits for the ICMP Time-Exceeded messages from different Layer-3 nodes along the path. Then Pathchar uses the minimum observed RTT to calculate the capacity of each hop. A key issue with this algorithm is that it needs a long time to measure the bandwidth since it involves sending a significant number of packets [38] to successfully find at least one packet that does not face any queuing delays. Furthermore, Layer-2 store-and-forward devices that do not decrease the TTL value [39], but can increase the delay due to queueing (serialization latency) [40] can cause Pathchar to misunderstand the location of the path bottleneck. Another problem of this method is that some routers either rate-limit or filter ICMP messages [41].

Pathchar is quite an old algorithm so to investigate its use in modern edge networks we implemented Pchar in the testbed illustrated in Figure 3.2, where the wireless link is the bottleneck. This testbed uses an Asus RT-AC86U Access Point (which uses a Broadcom 4366E chipset and supports 802.11ac MIMO) and configured to use the 5GHz frequency band with 20MHz channel bandwidth. A Linux server is connected to this AP via a gigabit Ethernet link to communicate with a Linux client which uses a Broadcom BCM4360 802.11ac NIC that supports MCS11. Using root privilege, we ran Pchar with default settings in the server to estimate the capacity of the edge network. Following 20 minutes of probing, the estimated capacity was 114Mbps. Note that the maximum receive rate in this setup is around 180Mbps (tested by iperf 2.5).

Packet-pair techniques send pairs of probe packets back-to-back to the client and measure the inter-packet arrival time to estimate the capacity of a path. This solution can cope with invisible Layer-2 devices since it does not rely on ICMP messages. However, the packet-pair approach is not robust to cross-traffic because it assumes that the transmitted probe packets are always queued together. Therefore, in the presence of cross traffic, it is probable that other packets are placed between the probing packet pairs in a queue and so reduce the accuracy of this method. To solve this issue, some algorithms have been proposed such as Nettimer [42] and Pathrate [43] which mostly use statistical approaches to remove the effect of cross traffic but these algorithms do not always give accurate results [41].

To investigate performance of the packet-pair idea, we also applied the Pathrate algorithm in the testbed shown in Figure 3.2 Note that Pchar is a server-side application, but Pathrate needs to run on both the server and the client. Using a similar setup and following 18 minutes of probing without any cross-traffic, the estimated capacity by Pathrate was in the range of 265Mbps to 320Mbps.

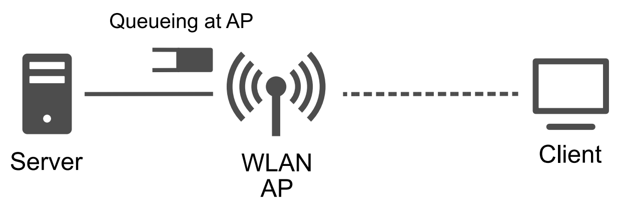


FIGURE 3.2: Schematic of a cloudlet-based edge transport architecture used for experimental setup.

An important reason why these algorithms, i.e. Pchar and Pathrate, have inaccurate results in our testbed is due to the aggregation in the downlink and uplink introduced in modern 802.11n/ac routers. In the downlink, the AP may aggregate probe packets in the packet-pair technique into the same frame so that these packets reach the destination NIC in the same time. Therefore, the inter-packet arrival time measured in the application layer is just the OS noise (when kernel reads the packets from NIC's buffer). Also aggregation in the uplink and ACK aggregation can increase the delay in the reverse path so that using RTT to measure the capacity in the one-packet technique may not give precise results.

3.5 Detecting Bottleneck Location

Having information about the location of queue build-up is important for network monitoring. For example, measurements show that the cellular link is the bottleneck in 68.9% cases for 3G users and 25.7% for LTE users [44]. In this section, we review the literature on detecting the location of the bandwidth bottleneck in computer and cellular networks.

3.5.1 Computer Networks

The open-source firmware proposed in [45], includes an algorithm to implement in APs in order to detect whether the bottleneck is in the backhaul link or in the WLAN. It measures two parameters, i.e. the coefficient of the variation of the inter-packet arrival time of packets received from the backhaul link and the round-trip time of packets transmitted over the wireless hop. As a passive measurement tool, it requires enough data traffic to be transmitted over the network to accurately detect the location of path bottleneck. This sensing time can be as high as a few seconds. Furthermore, the necessity for updating the firmware of routers to probe the network traffic, makes this technique not widely implementable. As described in [45], the predefined thresholds in this approach depend on the system configuration such as the device driver of NIC and also some other hardware configurations.

3.5.2 Cellular Networks

The BurstTracker algorithm [38] assumes that the scheduler in LTE prefers to assign all of the resource blocks in a millisecond to a single user when the LTE network is congested. Hence, assigning more than 90% of resources to a single user is considered as an indication of sustained queueing in the base station by this approach. In addition, the pattern of assigned resource blocks to a user in each millisecond is an important element to detect the queue status. For example, if a silent millisecond follows a millisecond of transmission with less than 40% of resources allocated to that user, the BurstTracker algorithm will assume that the queue is empty with 93% accuracy. However, this algorithm needs root privilege to access the debug information in the mobile handset which includes the trace of resource blocks allocated to the user during active milliseconds (when the queue is not empty in the base station).

QProbe [44] is an active probing algorithm that sends small probe packets in a burst and takes advantage of the proportional fair (PF) scheduler in cellular networks to detect the location of path bottleneck on the user side, i.e. backhaul or cellular last hop (3G or LTE). Here, the elegant idea is that when the probe packets are queued in the backhaul, then the inter-packet arrival time could be large but if congestion occurred in the base station then inter-packet arrival time would be very small. This happens because when the PF scheduler selects a UE for downlink transmission, multiple packets can be scheduled back-to-back in consecutive time slots during congestion events. The results show that the QProbe technique can detect the location of path bottleneck with more than 85% accuracy for both 3G and LTE networks.

Chapter 4

Quick & Plenty: Achieving Low Delay & High Rate in 802.11ac Edge Networks

4.1 Introduction

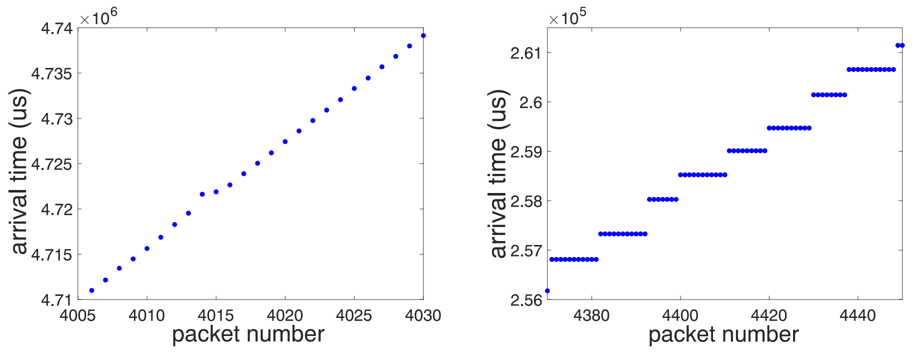
We consider transport layer approaches for achieving high rate, low delay communication over edge paths where the bottleneck is an 802.11ac WLAN. We first show that by regulating send rate so as to maintain a target aggregation level it is possible to realize high rate, low delay communication over 802.11ac WLANs. We then address two important practical issues arising in production networks, namely that (i) many client devices are non-rooted mobile handsets/tablets and (ii) the bottleneck may lie in the backhaul rather than the WLAN, or indeed vary between the two over time. We show that both these issues can be resolved by use of simple and robust machine learning techniques. We present a prototype transport layer implementation of our low delay rate allocation approach and use this to evaluate performance under real radio conditions.

In summary, our main contributions in this chapter are as follows. Firstly, we establish that regulating send rate so as to maintain a target aggregation level can indeed be used to realize high rate, low latency communication over 802.11ac WLANs. Secondly we address two important practical issues arising in production networks, namely that (i) many client devices are non-rooted mobile handsets/tablets and (ii) the bottleneck may lie in the backhaul rather than the WLAN, or indeed vary between the two over time. We show that both these issues can be resolved by use of simple and robust machine learning techniques. Thirdly, we present a prototype transport layer implementation of our low latency rate allocation approach and use this to evaluate performance under real radio channel conditions.

4.2 Preliminaries

4.2.1 Measuring Aggregation

The level of aggregation can be readily measured at a receiver using packet MAC timestamps. Namely, a timestamp is typically added by the NIC to each packet recording the time when it is received. This



(a) 10Mbps send rate

(b) 200Mbps send rate

FIGURE 4.1: MAC timestamp measurements for UDP packets transmitted over an 802.11ac downlink to two client stations. Packets transmitted in the same frame have the same MAC timestamp, and it can be seen from (a) that while there tends to be only one packet per frame at a 10Mbps send rate this increases in (b) to around 10 packets per frame at 200Mbps. The same downlink send rate is used for both client stations, data is shown for one client station. Experimental data, setup in Section 4.6.

timestamp is derived from the WLAN MAC and has microsecond granularity[1]. When a frame carrying multiple packets is received then those packets have the same MAC timestamp and so this can be used to infer which packets were sent in the same frame.

For example, Figure 4.1 shows measured packet timestamps for two different downlink send rates. The experimental setup used is described in Section 4.6. It can be seen from Figure 4.1(a) that when the UDP arrival rate at the AP is relatively low each received packet has a distinct timestamp whereas at higher arrival rates, see Figure 4.1(b), packets start to be received in bursts having the same timestamp. This behavior reflects the use by the AP of aggregation at higher arrival rates, as confirmed by inspection of the radio headers in the corresponding tcpdump data.

4.2.1.1 Link Layer Retransmission Book-keeping

The 802.11ac link layer retransmits lost packets. Our measurements indicate that these retransmissions usually occur in a dedicated frame in which case the aggregation level of that frame is often lower than for regular frames, e.g. see Figure 4.2. Losses also mean that the number of received packets in a frame is lower than the number transmitted. Fortunately, by inserting a unique sequence number into the payload of each packet we can infer both losses and retransmissions since they respectively appear as "holes" in the received stream of sequence numbers and as out of order delivery. We can therefore adjust our book-keeping to compensate for these when estimating the aggregation level.

4.3 Low Delay High-Rate Operation

Figure 4.3 shows measurements of the mean aggregation level, packet delay and loss vs the send rate to a client station for a range of network

[1]Note that, as will be discussed in more detail later, a second timestamp is also added by the kernel but this is recorded at a later stage in the packet processing chain and so is significantly less accurate.

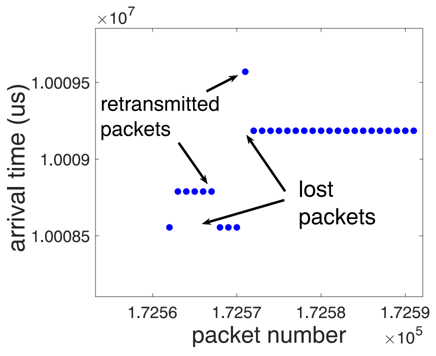


FIGURE 4.2: Experimental measurements illustrating typical 802.11ac link layer packet loss and retransmission behavior. In the first frame a burst of five packets are lost and retransmitted in the second frame. In the third frame the first packet is lost and retransmitted in the fourth frame.

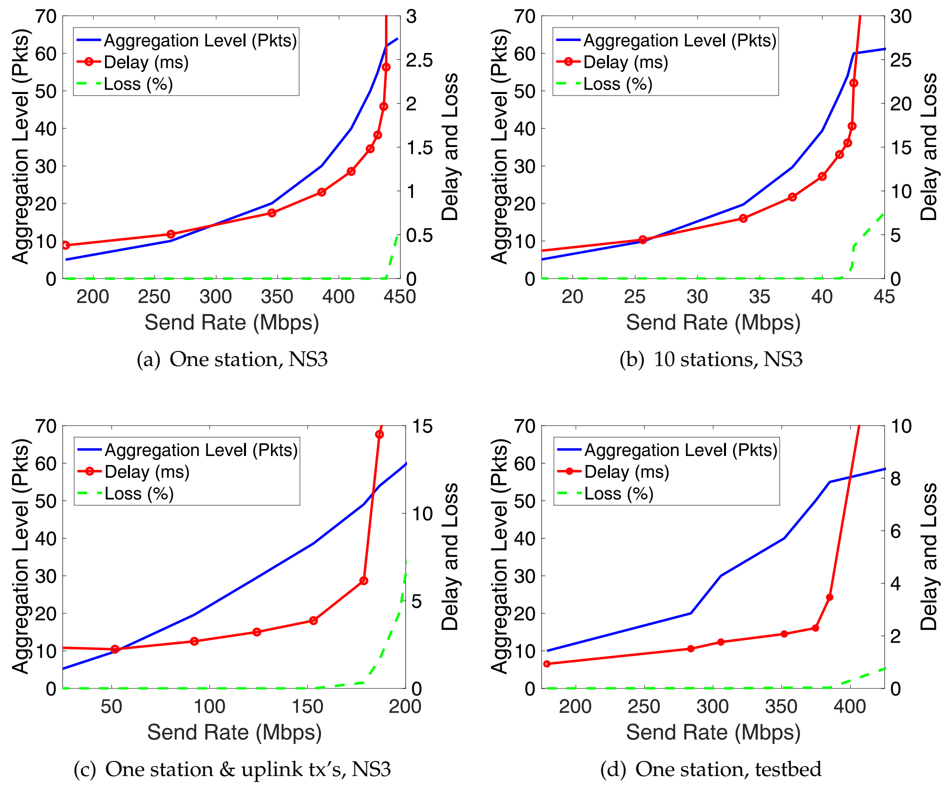


FIGURE 4.3: Measurements of average aggregation level, one-way packet delay and packet loss vs the send rate for a range of network conditions. (a) downlink flow to one client station, (b) 10 downlink flows to each of 10 client stations, data shown is for one of these flows, (c) setup as in (a) but with contention from an uplink flow, (d) setup as in (a) but measurements are from a hardware testbed located in an office environment.

configurations. A number of features are evident. Firstly, as the send rate is increased the aggregation level increases monotonically until it reaches the maximum value N_{max} supported by the MAC (for the data shown $N_{max} = 64$ packets). Secondly, the packet delay increases monotonically with send rate, initially increasing slowly but then increasing sharply as the send rate approaches the downlink capacity. Observe that the sharp increase in delay coincides with aggregation level approaching its upper limit of 64 packets and with the onset of packet loss. Note that all packet loss in this data is due to queue overflow since we verified that link layer retransmissions repair channel losses.

We can understand the behavior in Figure 4.3 in more detail by reference to the schematic in Figure 4.4. Packets are transmitted by the sender in a paced fashion. On arriving at the AP they are queued until a transmission opportunity occurs. The queue occupancy increases roughly linearly since the arriving packets are paced (have roughly constant inter-arrival times). Upon a transmission opportunity the queued packets are assembled into an aggregated frame and transmitted. Provided the queue is less than N_{max} the queue backlog is cleared by this transmission. For example, consider the shaded frame in Figure 4.3. This frame is transmitted at the end of time interval T_2 and the packets indicated by the shaded area on the queueing plot are aggregated into this frame. The oldest packet in this frame could have arrived just after interval T_1 and so may have waited up to T_2 seconds before transmission. Later arriving packets will, of course, experience less delay than this. The intervals T_1, T_2 etc between frame transmissions are random variables due to the randomized channel access mechanism used by 802.11 transmitters. Importantly, these intervals depend on the aggregation level, i.e. the duration T_2 depends on the time taken to send the frame aggregated from packets arriving in interval T_1 etc, and in turn the aggregation level depends on the interval duration since more packets arrive in a longer interval. The delay and aggregation level are therefore coupled to one another and this is what we see in Figure 4.3. Note that the intervals between transmissions may also vary due to contention with other transmitters (uplink transmissions by clients, transmissions by other WLANs sharing the same channel etc), link layer retransmissions, transmissions by the AP to other clients (recall modern APs use per station queueing so the coupling is only via these intervals) and so on but the basic setup remains unchanged and this is also reflected in Figure 4.3.

The data in Figure 4.3 suggests that if we could operate the system at an aggregation level of, for example, around 32 packets then we can obtain a high transmit rate while maintaining low delay. It is this observation that underlies the approach we propose here. Note that the AP transmit efficiency increases with the aggregation level since the overheads on a frame transmission are effectively fixed and so sending more packets in a frame increases efficiency. Hence, operating at less than the maximum possible aggregation level N_{max} incurs a throughput cost and there is therefore a trade-off between delay and rate. However, Figure 4.3 indicates that this trade-off is quite favorable, namely that low delay comes at the cost of only a relatively small reduction in rate compared to the maximum possible.

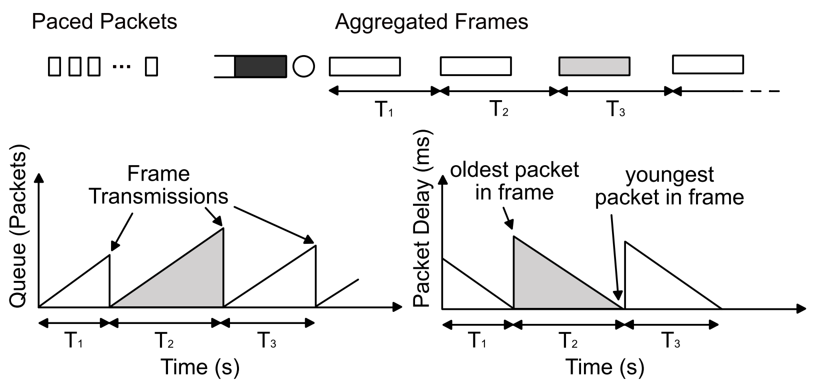


FIGURE 4.4: Illustrating connections between queueing, packet delay and frame aggregation at the AP. Packets arriving at the AP are queued for transmission, the queue growing roughly linearly over time as packets arrive in a paced fashion. When a transmission opportunity occurs an aggregated frame is constructed from the queued packets. Provided the number of queued packets is less than the maximum frame aggregation level N_{max} then the queue backlog is cleared by the transmission. The delay of the oldest packet in a frame is upper bounded by the time between transmission opportunities.

4.3.1 Controlling Delay

We proceed by introducing a simple feedback loop that adjusts the sender transmit rate (corresponding to the AP arrival rate, assuming no losses between sender and AP) to maintain a specified target aggregation level. Namely, time is partitioned into slots of duration Δ seconds and we let $\mathcal{T}_{i,k}$ denote the set of frames transmitted to station i in slot k. Station i measures the number of packets $N_{i,f}$ aggregated in frame f and reports the average $\mu_{N_i}(k) := \frac{1}{|\mathcal{T}_{i,k}|}\sum_{f\in\mathcal{T}_{i,k}} N_{i,f}$ back to the sender. The sender then uses proportional-integral (PI) feedback[2] to increase its transmit rate x_i if the observed aggregation level $\mu_{N_i}(k)$ is less than the target value N_ϵ and decrease it if $\mu_{N_i}(k) > N_\epsilon$. This can be implemented using the pseudo-code shown in Algorithm 1. Note that this feedback loop involves three design parameters, update interval Δ, feedback gain K and target aggregation level N_ϵ. We consider the choice of these parameters in more detail shortly but typical values are $\Delta = 500$ms or 1000ms, $K = K_0/n$ with $K_0 = 1$ (where n is the number of client stations in the WLAN) and $N_\epsilon = 32$ packets.

Algorithm 1 Feedback loop adjusting transmit rate x_i to regulate aggregation level μ_{N_i}.

```
$k = 1$
**while** 1 **do**
    $\mu_{N_i} \leftarrow \frac{1}{|\mathcal{T}_{i,k}|}\sum_{f\in\mathcal{T}_{i,k}} N_{i,f}$
    $x_i \leftarrow x_i - K(\mu_{N_i} - N_\epsilon)$
    $k \leftarrow k + 1$
**end while**
```

We implemented this feedback loop in our experimental testbed, see Section 4.6 for details, and Figure 1.1(b) shows typical results obtained by regulating the aggregation level.

[2]While design of more sophisticated control strategies is of interest, this is an undertaking in its own right and we leave this to future work.

4.3.2 Multiple Stations: Equal Airtime Fairness

When there are multiple client stations we can modify Algorithm 1 as follows to allocate roughly equal airtime to each station. Recall that the airtime used to transmit the payload of station i is $T_i = \mu_{N_i} L / \mu_{R_i}$, where L is the number of bits in a packet, μ_{N_i} is the number of packets in a frame and μ_{R_i} is the MCS rate used to transmit the frame in bits/s. So selecting $\mu_{N_i} = \mu_{R_i} / \mu_{R_{i^*}}$ makes the airtime equal with $T_i = T_{i^*}$ for all stations. Letting $\boldsymbol{x}$ denote the vector of downlink send rates to ensure equal airtimes we therefore increase the rate x_{i^*} of the station i^* with highest MCS rate $\mu_{R_i}(k)$ when its observed aggregation level $\mu_{N_i}(k)$ is less than the target value N_ϵ and decreases x_{i^*} when $\mu_{N_i}(k) > N_\epsilon$, i.e. at slot k

$$x_{i^*}(k+1) = x_{i^*}(k) - K(\mu_{N_i}(k) - N_\epsilon) \tag{4.1}$$

The rates of the other stations are then assigned proportionally,

$$x_i(k+1) = x_{i^*}(k+1)\frac{\mu_{R_i}}{\mu_{R_{i^*}}},\ i = 1, \ldots, n \tag{4.2}$$

Note that the update (4.1)-(4.2) only uses readily available observations. Namely, the frame aggregation level $N_{i,f}$ and the MCS rate $R_{i,f}$, both of which can be observed in userspace by packet sniffing on client i.

4.3.3 Selecting Design Parameters Δ and K_0

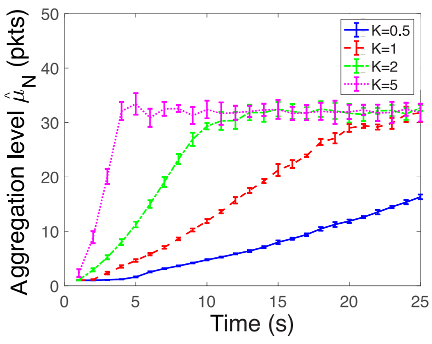
4.3.3.1 Convergence Rate

We expect that the speed at which the aggregation level and send rate converge to their target values when a station first starts transmitting is affected by the choice of feedback gain K_0 and update interval Δ. Figure 4.5 plots measurements showing the transient following startup of a station vs the choice of K_0 and Δ.

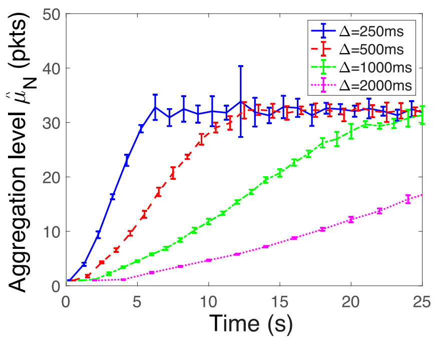
It can be seen from Figure 4.5(a) that as gain K_0 is increased (while holding Δ fixed) the time to converge to the target aggregation level $N_\epsilon = 32$ decreases. However, as the gain is increased the feedback loop eventually becomes unstable. Indeed, not shown in the plot is the data for $K_0 = 10$ which shows large, sustained oscillations that would obscure the other data on the plot. Similarly, it can be seen from Figure 4.5(b) that as the update interval Δ is decreased the convergence time decreases.

4.3.3.2 Disturbance Rejection

Observe in Figure 4.5 that while the convergence time decreases as K_0 is increased the corresponding error bars indicated on the plots increase. As well as the convergence time we are also interested in how well the controller regulates the aggregation level about the target value N_ϵ. Intuitively, when the gain K_0 is too low then the controller is slow to respond to changes in the channel and the aggregation level will thereby

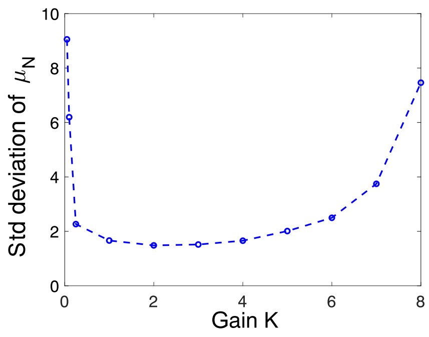


(a) Impact of K_0 (with $\Delta = 1000$ms)

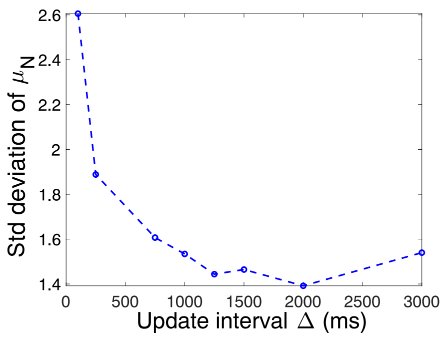


(b) Impact of Δ (with $K_0 = 1$)

FIGURE 4.5: Convergence rate vs feedback gain K_0 and update interval Δ. Mean and standard deviation from 10 runs at each parameter value. One client, 802.11ac, $N_\epsilon = 32$. Experimental data, setup in Section 4.6, $N_{max} = 64$.



(a) Impact of K_0 ($\Delta = 1000$ms).



(b) Impact of Δ ($K_0 = 1$).

FIGURE 4.6: Noise rejection (as measured by standard deviation of μ_N) vs feedback gain K_0 and update interval Δ. One client, 802.11ac, $N_\epsilon = 32$. Experimental data, setup in Section 4.6, $N_{max} = 64$.

show large fluctuations. When K_0 is increased we expect feedback loop is also able to respond more quickly to genuine changes in channel behavior.

This behavior can be seen in Figure 4.6(a) which plots measurements of the standard deviation of the aggregation level μ_N (where the empirical mean is calculated over the update interval Δ of the feedback loop) as the control gain K_0 is varied. When the update interval Δ is made smaller we expect that the observations $\mu_{N_i}(k)$ and $\mu_{R_i}(k)$ will tend to become more noisy (since they are based on fewer frames) which may also tend to cause the aggregation level to fluctuate more. However, the feedback loop is also able to respond more quickly to genuine changes in channel behavior. Conversely, as Δ increases the estimation noise falls but the feedback loop becomes more sluggish.

Figure 4.6(b) plots measurements of the standard deviation of the aggregation level μ_N as Δ is varied. It can be seen that due to the interplay between these two effects the standard deviation of μ_N increases when Δ selected too small or too large, with a sweet spot for Δ around 1250-2000ms.

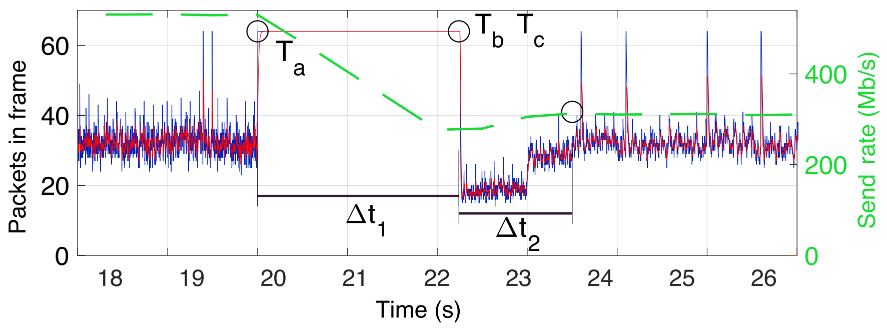


FIGURE 4.7: Illustrating adaption of send rate by feedback algorithm in response to a change in channel conditions (from use of 2 spatial streams down to 1 spatial stream). NS3 simulation, single client station, MCS 9, $K_0 = 1$, $\Delta = 500$ms, $N_\epsilon = 32$, $N_{max} = 64$.

4.3.3.3 Responding to Channel Changes

The feedback algorithm used by the sender regulates the send rate to maintain a target aggregation level. It therefore adapts automatically to changing channel conditions. To investigate this we change to using NS3 simulations since this allows us to change the channel in a controlled manner (we also have experimental measurements showing adaptation to changing channel conditions, not shown here, but in this case we do not know ground truth).

Fig. 4.7 illustrates typical behavior of the feedback algorithm. Initially the AP uses 2 spatial streams and then at $t = T_a = 20$s it switches to 1 spatial stream. For $\Delta t_1 = 2.24$s all AMPDUs hit the maximum aggregation level of 64 packets and we start observing losses. During this time it can be seen that the algorithm, which updates the send rate twice per second ($\Delta = 500$ms), is slowing down the sending rate. It takes four rounds to reach a rate compatible with the channel, but it takes a little bit more to stabilize the aggregation level at the AP in $t = T_b$. After another three rounds (for approximately $\Delta t_2 = 1.26$s) it can be seen that the sending rate settles at its new value in $t = T_c$.

4.3.4 Fairness With High Rate & Low Delay

We now explore performance with multiple client stations. We begin by considering a symmetric network configuration where client stations are all located at the same distance from the AP. We then move on to consider asymmetric situations where the channel between AP and client is different for each client. Again we use NS3 simulations in this section since this facilitates studying the performance with larger numbers of clients.

4.3.4.1 Clients Same Distance From AP

We begin by considering a network configuration where client stations are all located two meters from the AP. Fig. 4.8 (top) shows measurements of the aggregated application layer goodput and average delay vs the number of receivers.

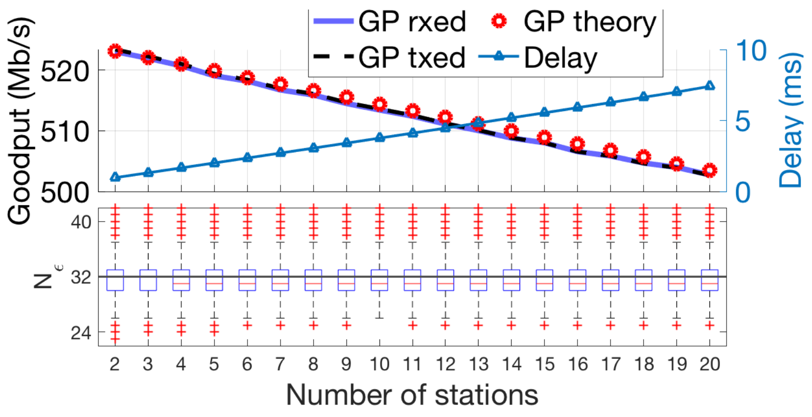


FIGURE 4.8: Sum-goodput and delay vs number of receivers (top) and corresponding distribution of aggregation level about the target value of $N_\epsilon = 32$ (bottom). In the top plot the *GP theory* line (GP, goodput) is a theoretical upper limit computed by assuming an AMPDU with $N_\epsilon = 32$ packets, 10 feedback packets per second per receiver, 10 beacons per second, and no collisions. NS3 simulations, $K_0 = 1$, $\Delta = 500$ms, $N_{max} = 64$.

It can be seen that the aggregated goodput measured at the receivers is close to the theoretical limit supported by the channel (MCS) configuration, being only a few Kb/s below this for 20 receivers. This goodput is evenly shared by the receivers (the measured Jain's Fairness Index is always 1). The average delay increases almost linearly at the rate of 350μs per additional station. The lower plot in Fig. 4.8 shows the measured distribution of frame aggregation level, with the edges of the boxes indicating the 25th and 75th percentiles. It can be seen that the feedback algorithm tightly concentrates the aggregation level around the target value of $N_\epsilon = 32$. As expected, since the delay is regulated to a low level we did not observe any losses.

4.3.4.2 Randomly Located Clients

We now consider a scenario where the client stations are randomly located in a square of side 40m and the AP is located in its centre. We configured MinstrelHT algorithm as the rate controller, this adjusts the MCS used by each client station based on its channel conditions (better for stations closer to the AP, worse for those further away). To ease visualization we use NSS=1 which helps to reduce the MCS fluctuations generated by Minstrel. We ran experiments with eight receivers until we collected 200 points where the rate controller converged to a stable choice for all receivers, i.e., with more than 85% of frames received with the same MCS. We group receivers by MCS and report statistics on N_ϵ for each group as boxplots in the top-left plot in Figure 4.7. The thick circles indicate the choice of rate allocation that assigns equal airtime to all receivers, and it can be seen that the measured rate allocation is maintained close to those values by the feedback algorithm.

In clockwise order Figure 4.7 then shows the ECDF of losses, aggregated goodput and delay. Observe that losses occur in this configuration because of far away nodes not being able to correctly decode all packets. The aggregate goodput can drop as low as 150Mb/s when MinstrelHT selects MCS4 for all receivers, but converges to 300Mb/s with MCS 9

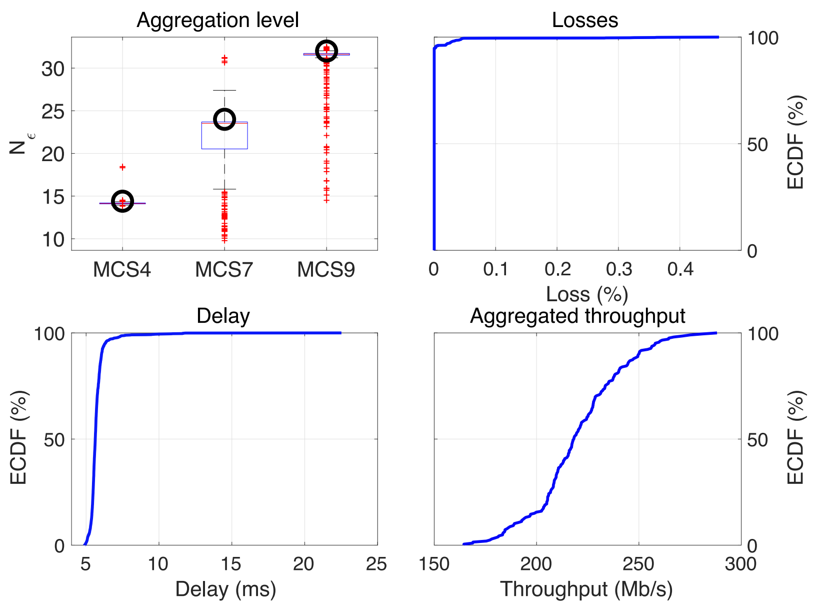


FIGURE 4.9: Performance with 8 receivers placed randomly in a square of side 40m: MCS is chosen by MinstrelHT algorithm, NSS = 1. NS3 simulations, $K_0 = 1$, $\Delta = 500$ms, $N_{max} = 64$.

(the theoretical maximum goodput with MCS 9, NSS 1 and $N_\epsilon = 32$ is 307Mb/s). Delay is consistently less than 6ms.

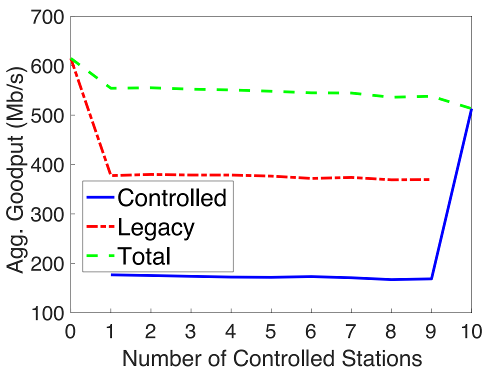
4.3.4.3 Coexistence With Legacy WLANs

We next analyze the performance of stations which regulate the aggregation level when they co-exist with legacy stations.

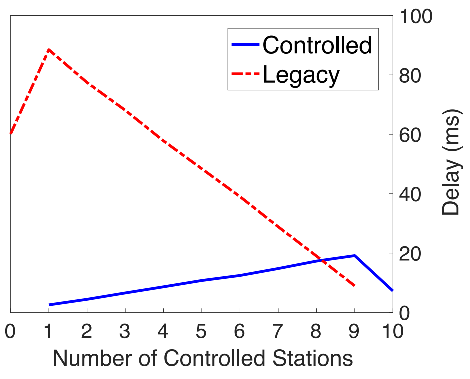
Figures 4.10(a)-4.10(b) show the measured delay and performance for a setup with two WLANs sharing the same channel. The first WLAN contains stations that regulate the aggregation level on the downlink while in the second WLAN the AP has a persistent backlog and so always has packets to send to each station. We hold the total number of client stations constant at 10 but vary the fraction which regulate the aggregation level, as indicated on the x-axis of the plots.

When the number of new controlled stations is zero, i.e. there are 10 saturated legacy stations sharing the same AP, then the aggregated goodput shown in Figures 4.10(a) is close to the theoretical prediction for when the aggregation level is 64 packets (approximately 615Mb/s). When the number of controlled stations is 10, i.e. there are 10 controlled stations sharing the same AP, then it can be seen that the aggregated goodput is around 515Mbps, close to the theory value when the aggregation level is 32 packets (which is the target value for the controlled stations).

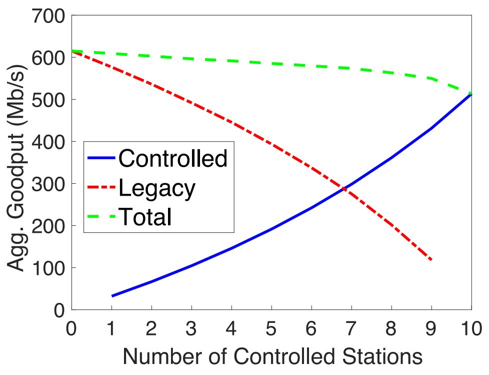
When there is a mix of controlled and legacy stations it can be seen that the legacy stations gain a higher fraction of the total goodput than the stations which control aggregation level, as expected since the legacy stations use the maximum possible aggregation level $N_{max} = 64$. However, this gain in goodput comes at the price of higher delays for the legacy stations. If we compare the delay achieved for the same number of stations in the two groups, it can be seen that the delay of the legacy



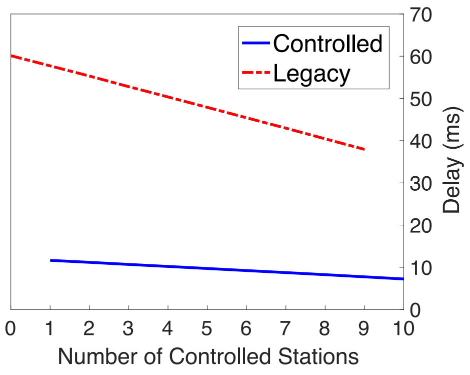
(a) Goodput Comparison, 2 BSSs



(b) Delay Comparison, 2 BSSs



(c) Goodput Comparison, 1 BSS



(d) Delay Comparison, 1 BSS

FIGURE 4.10: Top: comparison of Goodput (left) and Delay (right) when Controlled and Legacy stations form two BSSs with separate APs. Bottom: same comparison when all stations are joined to the same AP. NS3 simulations, $K_0 = 1$, $\Delta = 500$ms, $N_{max} = 64$.

stations is approximately four times that of stations that regulate the throughput. The goodput fraction is (almost) invariant with the number of stations since the 802.11 MAC assigns an equal number of transmission opportunities to each AP regardless of the number of clients in each WLAN.

This data confirms that WLANs where stations regulate their aggregation level can coexist successfully with legacy WLANs. Namely, legacy WLANs are not penalized and the new controlled stations are still able to achieve low delay at reasonably high rates.

Figures 4.10(c)-4.10(d) show corresponding measurements when the legacy stations share the same WLAN as the new controlled stations.

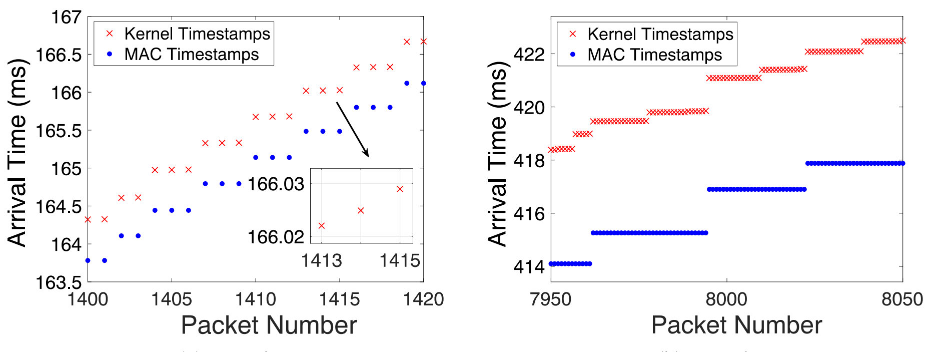
Now the fraction of goodput allocated to each class of station changes as the number of controlled stations is varied. This is because both legacy and controlled stations share the same AP and this uses round-robin scheduling. Once again, as expected legacy stations achieve higher goodput than stations which regulate the aggregation level. The high aggregation level used by the legacy stations means that their transmissions take more airtime. This induces delay for the new controlled stations since they must wait for the legacy station transmissions to complete due to the round-robin scheduling used by the AP: still the delay for the controlled stations is always less than 12ms even in the worst case with a single controlled station against 9 legacy stations and it falls to 7ms when all stations are controlled.

4.4 Non-rooted Mobile Handsets

The results in the previous section make use of MAC timestamps to measure the aggregation level of frames received at WLAN clients. However, access to MAC timestamps is via prism/radiotap libpcap headers and typically requires root privilege. While this is fine for devices running operating systems such as Linux or OS X, it is problematic for mobile handsets and tablets since root privilege is generally not enabled for users on Android and iOS. Mobile handsets/tablets are, of course, the primary means of accessing the internet for many users and so for our low latency approach to be of practical use it is important that it is compatible with these.

Potentially we can sidestep this constraint by use of a separate network sniffer with root access. But this is unappealing for at least two major reasons. Firstly, it entails installation of additional infrastructure, with associated cost and maintenance and unfavorable economics as cell sizes shrink. Secondly, sniffing in monitor mode is itself becoming increasingly complex due to use of MIMO (the directionality makes it difficult to achieve monitor coverage) and very high rate PHYs (making sniffing liable to error/corruption). Similarly, deploying measurement mechanisms on the AP may not be an option: manufacturers, in fact, restrict access to their devices and the update cycle can be much slower than in the case of a proxy software running on an edge- or cloud- server.

With the above in mind, we note that the kernel adds timestamps to received packets and these are accessible on mobile handsets via the



(a) 100Mbps (b) 300Mbps

FIGURE 4.11: Kernel and MAC timestamp measurements for UDP packets transmitted to a mobile handset over 802.11ac. In (a) the arrival rate of packets at the AP is 100Mbps and in (b) 300Mbps. Experimental data, setup in Section 4.6, AMDSU aggregation ($N_{max} = 128$).

`SO_TIMESTAMP` or `SO_TIMESTAMPNS`[3] socket options without root privilege. However, the kernel timestamp for a packet is recorded when the received packet is moved to the receive ring buffer and so is subject to significant "noise" associated with driver scheduling e.g. interrupt mitigation may lead to a switch to polled operation when under load.

Note that we tried `tcpdump` on a rooted Google Pixel 2 phone but found that monitor mode does not enable. We also tried the `SO_TIMESTAMPING` socket option with root privilege to capture MAC timestamps using the `recvmsg` Linux utility with related flags such as `SOF_TIMESTAMPING_RX_HARDWARE` and `SOF_TIMESTAMPING_RAW_HARDWARE` but the MAC timestamps are not reported. Therefore, MAC time-stamping is not supported by all handsets and having root access does not guarantee accessing this information on mobile phones.

Typical kernel timestamp "noise" is shown, for example, in Figure 4.11. When the arrival rate at the AP is relatively low, it can be seen from Figure 4.11(a) that two or three packets share each MAC timestamp and so are aggregated into the same frame. While the kernel timestamps differ for packets transmitted in the same frame (see plot inset), there is nevertheless a clear jump in the kernel timestamps between frames and this can be used to cluster packets sharing the same frame. Figure 4.11(b) shows corresponding measurements taken at a higher network load. It can be seen that now many more packets are aggregated into each MAC frame, as might be expected. However, there is now no longer a clear pattern between the jumps in kernel timestamp values and the frame boundaries: sometimes there are large jumps within a frame. Although not shown in this plot, it can also happen that no clear jump in kernel timestamps is present at boundary between frames. We believe this is due to the action of NAPI interrupt mitigation, which causes the kernel to switch from interrupt to driver polling at higher network loads.

In this section we explore whether, despite their noisy nature, kernel packet timestamps can still be successfully used to estimate the aggregation level of frames received on a mobile handset.

[3]Same timestamping mechanism as SO_TIMESTAMP, but reports the timestamp in ns resolution rather than μs. [46]

4.4.1 Estimating Aggregation Level: Logistic Regression

To estimate the aggregation level using noisy kernel timestamps we adopt a supervised learning approach. For training purposes we have ground truth available via the sniffer, although we require the resulting estimator to be robust enough to be used across a wide range of network conditions without retraining.

As our main input features we use the inter-packet arrival times of last m received packets, derived from their kernel timestamps, plus their standard deviation. Parameter m is a design parameter that we will select shortly. The input feature vector $X^{(i)}$ associated with the ith packet is therefore:

$$X^{(i)} = [t_{i-m+1} \quad t_{i-m+2} \quad \dots \quad t_i \quad \sigma_i]^T \tag{4.3}$$

where t_i is the arrival time difference (in microseconds) between ith and $(i-1)$th packets received and σ_i is the standard deviation of the last m inter-packet arrival times. We define target variable $Y^{(i)}$ as taking value 1 when the ith packet is the first packet in an aggregated frame and 0 otherwise.

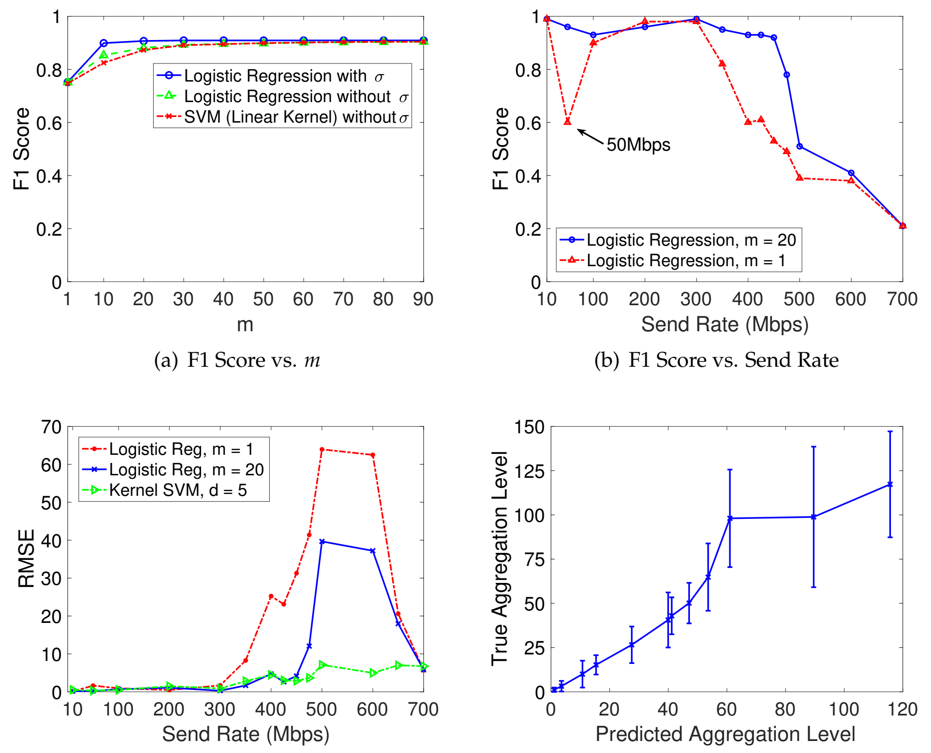
While the OS polling noise is challenging to model and the relation between MAC and kernel timestamps is complex, surprisingly it turns out that we can quite effectively estimate $Y^{(i)}$ using the following simple logistic model:

$$P(Y^{(i)} = 1 | X^{(i)} = X) = \frac{1}{(1 + e^{-\theta_0 - \boldsymbol{\theta}^T X})} \tag{4.4}$$

where $\boldsymbol{\theta} = (\theta_1 \dots, \theta_{m+1})^T \in \mathbb{R}^{m+1}$ plus θ_0 are the $m+2$ (unknown) model parameters.

To train this model we use timestamp data collected for a range of send rates from 100Mbps to 600Mbps where at each rate 250,000 packets are collected. We use the F1 metric, which combines accuracy and precision, to measure performance at predicting label $Y^{(i)}$ for each packet. We use the Scikit-learn library [47] to train the model using this data, applying 20-fold cross-validation to estimate error bars. We found the standard deviation to be consistently less than 0.01 and since it is hard to see such small error bars on our plots these are omitted.

Figure 4.12(a) plots the measured F1 score vs the number m of input features. Data is shown both for logistic regression and SVM cost functions and also when the standard deviation σ_i is and is not included in the feature vector. From this data we can make the following observations. For m greater than about 40 features the performance of all of the estimators is much the same, but for m less than 40 addition of σ_i boosts performance by around 5%. Note that use of a small value of m is desirable since we need to wait for m packets in order to generate an estimate for $Y^{(i)}$ and so the latency of the estimator increases with m. The performance with the logistic regression and SVM cost functions is similar, as might be expected, but we adopt the logistic regression choice of parameters as the predictions have slightly better performance.



(a) F1 Score vs. m

(b) F1 Score vs. Send Rate

(c) RMSE of Predicted Aggregation Level vs. Send Rate

(d)

FIGURE 4.12: Performance of logistic regression estimator vs (a) number m of features and (b), (c) send rate; (d) actual vs predicted aggregation level. Experimental data, Samsung Galaxy handset, setup in Section 4.6, AMDSU aggregation (so $N_{max} = 128$).

Unless otherwise stated, hereafter we use $m = 20$ plus feature σ_i. Note also that the same set of parameter values $(\boldsymbol{\theta}, \theta_0)$ is used for all send rates i.e. a single estimator is used across the full range of operation.

Figure 4.12(b) plots the performance of this estimator vs the downlink send rate. It can be seen that the prediction accuracy is high for rates up to about 500Mbps, but then starts to drop sharply. This is the accuracy of predicting the label $Y^{(i)}$ of each packet, but of course our real interest is in predicting the aggregation level. The aggregation level can be directly derived from the $Y^{(i)}$ labels (its just the number of packets between those labelled with $Y^{(i)} = 1$, capped at N_{max}). Figure 4.12(c) shows the measured aggregation level prediction accuracy vs the downlink send rate. It can be seen that, as might be expected, it shows quite similar behavior to Figure 4.12(b). A notable exception is at send rates above 700Mbps where the accuracy of the aggregation level improves. This is because at such high rates the aggregation level has hit the upper limit N_{max} and the estimator simply predicts N_{max} as the aggregation level. We will consider the causes for the drop in accuracy at rates between 500-700Mbps in more detail shortly, but note briefly that it is directly related to the load-related "noise" on kernel timestamps (recall Figure 4.11(b)).

As a baseline for comparison Figures 4.12(b)-(c) also shows the performance of the logistic regression estimator when $m = 1$ (and without

σ_i). The latter corresponds to an estimator that labels packets by simply thresholding on the inter-packet arrival time i.e. when the time between the current and previous packet exceeds a threshold we label the current packet as the first in a new MAC frame. The threshold level used is optimized to maximize prediction accuracy on the training data. From Figure 4.11(a) we can expect that under lightly loaded conditions this approach is quite effective, but Figure 4.11(b) also tells us that it is likely to degrade as the load increases and indeed it can be seen from Figures 4.12(b)-(c) that the performance of this baseline estimator degrades for rates above about 300Mbps (compared to rates above about 500Mbps with $m = 20$). It can also be seen that the accuracy drops sharply at a rate of 50Mbps. This is because at this send rate the inter-packet send time is close to the simple threshold used in the baseline estimator. Hence the logistic regression estimator with $m = 20$ offers significant performance gains over this baseline estimator.

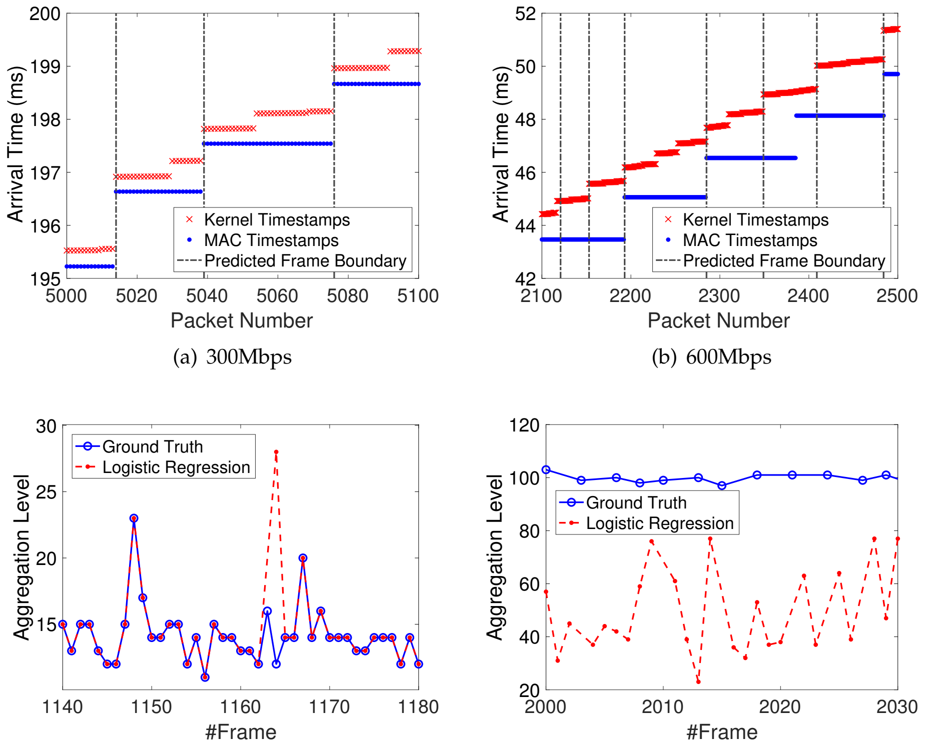
4.4.2 Improving Accuracy At High Network Loads: SVM

As already noted, the accuracy of the logistic regression estimator falls for send rates in the range 500-700Mbps, see Figures 4.12(b)-(c). The reason for this can be seen from Figure 4.13. Figure 4.13(a) shows the frame boundaries predicted by the estimator when the arrival rate at the AP is 300Mbps. The true frame boundaries can be inferred from the MAC timestamps, which are also shown in this plot. Observe that even although there are jumps between the kernel timestamps of packets sharing the same frame the estimator is still able to accurately predict the frame boundaries. Figure 4.13(b) shows the corresponding data when the arrival rate is increased to 600Mbps. It can be seen that there are now many jumps in the kernel timestamps of packets sharing the same MAC frame and sometime no jump in timestamps between packets transmitted in different frames (e.g. see the frame towards the right-hand edge of the plot). As a result the estimator makes many mistakes when trying to predict the frame boundaries.

Figures 4.13(c)-(d) show time histories of the estimated aggregation level. Observe in Figure 4.13(d) that there is a fairly consistent offset between the predicted and actual aggregation level. While this figure is for a single send rate of 600Mbps, Figure 4.12(d) plots the relationship between predicted and actual aggregation level for a range of send rates. Since the error is fairly consistent the potential exists to improve the estimator for send rates in the 500-600Mbps range. We explored various approaches for this and found the most effective is to combine the logistic regression estimator with a radial-basis function kernel SVM with the following input features,

$$X^{(i)} = [\mu_{\widehat{N}_{\delta_{i-d+1}}} \quad \cdots \quad \mu_{\widehat{N}_{\delta_i}} \quad \sigma_{\delta_i} \quad A_{\delta_i}]^T \tag{4.5}$$

where we partition time into 100ms slots and $\mu_{\widehat{N}_{\delta_i}}$ is the empirical average of the aggregation level predicted by the logistic regression estimator over the i slot, σ_{δ_i} the empirical variance and A_{δ_i} the number of frames. The averaging over slots reduces the noise and significantly improves



(a) 300Mbps

(b) 600Mbps

(c) 300Mbps

(d) 600Mbps

FIGURE 4.13: Frame boundaries predicted by the logistic regression estimator for (a) medium and (b) high network loads, and corresponding kernel and MAC timestamp measurements. Also time histories of estimated aggregation level for (a) medium and (b) high network loads. Experimental data, Samsung Galaxy handset, setup in Section 4.6, AMDSU aggregation ($N_{max} = 128$).

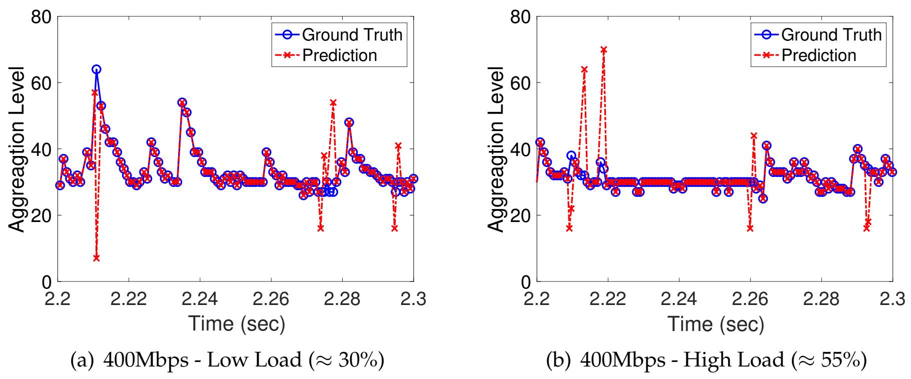


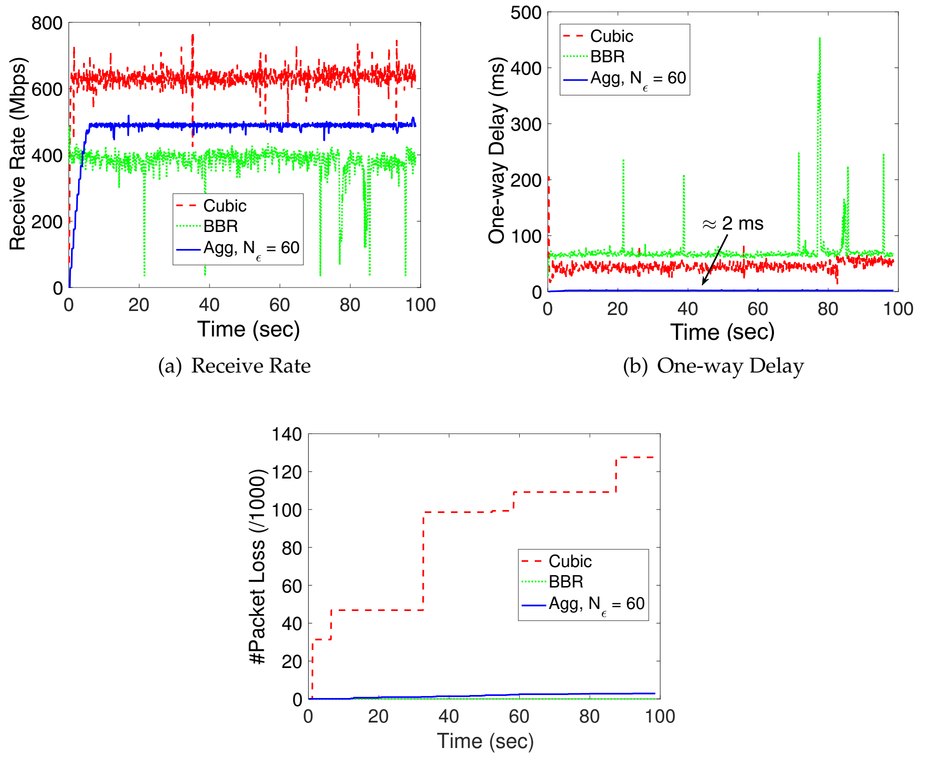
FIGURE 4.14: Illustrating the accuracy of prediction for different CPU loads. Experimental data, Samsung Galaxy handset, setup in Section 4.6, AMDSU aggregation ($N_{max} = 128$).

performance. The output of the SVM is the predicted aggregation level. We trained this SVM using the measured aggregation level averaged over each slot (again to reduce noise during training) and using cross-validation selected $d = 5$. Figure 4.12(c) plots the RMSE of the predictions vs send rate when the logistic regression estimator is augmented with this SVM. It can be seen that the performance is considerably improved for send rates in the 500-600Mbps range, with the the RMSE now no more than 8 packets compared with a value of around 40 packets when the logistic regression estimator is used on its own.

Note that since it operates over 100ms slots the SVM estimator is less responsive that the logistic regression estimator, but since the controller only updates the send rate every Δ seconds with Δ typically 0.5 or 1s then the 100ms delay introduced by the SVM estimator is minor. However, since the SVM estimator is relatively computationally expensive and the logistic regression estimator is sufficiently accurate for the low delay operating regime of interest here (where the rates are less than 500Mbps), so in the rest of the chapter we confine use to the logistic regression estimator unless otherwise stated.

4.4.3 Effect of CPU Load On Estimator Performance

To understand whether the noise on kernel timestamps is affected by system CPU load as well as network load we collected packet timestamp measurements while varying the CPU load by playing a 4K video in full screen mode. We found that CPU load makes little difference to the accuracy of the aggregation level estimator. For example, Figure 4.14 shows two typical time histories of measured and estimated aggregation level. Figure 4.14(a) is when the CPU load is around 30% and Figure 4.14(a) when the CPU load is around 55%. Even with a fairly high network load of 400Mbps it can be seen that the aggregation levels predicted by the estimator agree well with the actual aggregation level regardless of the CPU load.



(a) Receive Rate

(b) One-way Delay

(c) #Packet loss

FIGURE 4.15: Compare the performance of aggregation-based rate control algorithm with TCP Cubic and BBR. The one-way delay in (b) is averaged over 100ms intervals. $K_0 = 1$, $\Delta = 1000$ms, $N_\epsilon = 60$. Experimental data, Samsung Galaxy handset, setup in Section 4.6, AMDSU aggregation ($N_{max} = 128$).

4.4.4 Robustness of Estimator

While the foregoing measurements are for a Samsung Galaxy tablet we obtained similar results (using the same trained estimator, without changing the parameter values) when using a Google Pixel 2 handset. We also obtained similar results when there are multiple WLAN clients, as might be expected since per client queueing is used by 802.11ac APs i.e packets to different clients queued separately.

4.4.5 Performance Comparison With TCP Cubic & BBR

We extended the client-side code in our prototype implementation of the rate allocation approach in Section 4.3 to make use of kernel timestamps and the logistic regression estimator of aggregation level. The code is written in C and could be directly cross-compiled for use on the Samsung Galaxy tablet.

As might be expected, we obtain similar results to those shown in Section 4.3 when using MAC timestamps and so do not reproduce these here. Instead we take the opportunity to compare the performance of our proposed aggregation-based rate control algorithm with TCP Cubic [29], the default congestion control algorithm used by Linux and Android. In addition, we compare performance against TCP BBR [34] since this is a state-of-the-art congestion control algorithm currently being developed by Google and which also targets high rate and low latency.

Since TCP Cubic is implemented on Android we use the Samsung Galaxy as client. However, TCP BBR is not currently available for Android and so we use a Linux box (Debian Stretch, 4.9.0-7-amd64 kernel) as the BBR client.

Figure 4.15 shows typical receive rate and one-way delay time histories measured for the three algorithms. It can be seen from Figure 4.15(a) that Cubic selects the highest rate (around 600Mbps) but from Figure 4.15(b) that this comes at the cost of high one-way delay (around 50ms). This is as expected since Cubic uses loss-based congestion control and so increases the send rate until queue overflow (and so a large queue backlog and high queueing delay at the AP) occurs. As confirmation, Figure 4.15(c) plots the number of packet losses vs time and it can be seen that these increase over time when using Cubic, each step increase corresponding to a queue overflow event followed by backoff of the TCP congestion window.

BBR selects the lowest rate (around 400Mbps) of the three algorithms, but surprisingly also has the highest end-to-end one-way delay (around 75ms). High delay when using BBR has also previously been noted by e.g. [48] where the authors propose that high delay is due to end-host latency within the BBR kernel implementation at both sender and receiver. However, since our focus is not on BBR we do not pursue this further here but note that the BBR Development team at Google is currently developing a new version of BBR v2.

Our low delay aggregation-based approach selects a rate (around 480 Mbps), between that of Cubic and BBR, consistent with the analysis in earlier sections. Importantly, the end-to-end one-way delay is around 2ms i.e. more than 20 times lower than that with Cubic and BBR. It can also be seen from Figure 4.15(c) that it induces very few losses (a handful out of the around 4M packets sent over the 100s interval shown).

4.5 Detecting Bottleneck Location

The foregoing analysis applies to edge networks where the wireless hop is the bottleneck. For robust deployment, however, we need to be able to detect when this is violated i.e. when the bottleneck is the backhaul link. In this section we show how measurement of the aggregation level can be used for this purpose also. Note that this is of interest in its own right for network monitoring and management, separately from its use with our aggregation-based low delay rate control approach.

The basic idea is as follows. When the AP is the bottleneck then a queue backlog will develop there and so an elevated level of aggregation will be used in transmitted frames. Conversely, when the bottleneck is the backhaul link then we will observe delay and loss but a low level of packet aggregation. Hence we can use aggregation level and loss/delay as input features to a bottleneck location classifier. Note that this passive probing approach creates no extra load on the network (unlike active

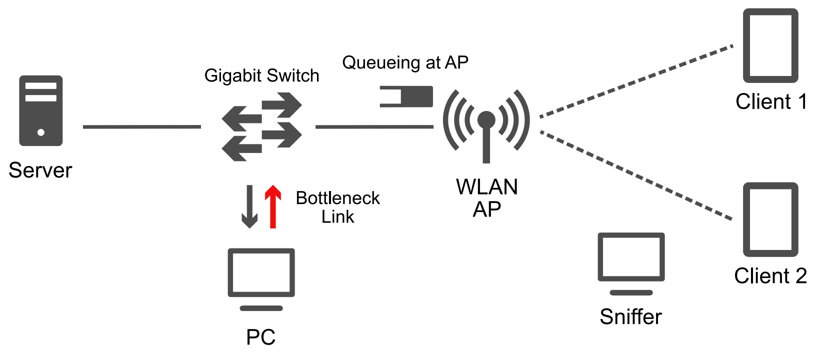


FIGURE 4.16: System model used in our testbed to do measurement when the backhaul is bandwidth bottleneck, 802.11ac.

probing methods) and can also cope with bridged Layer-2 devices (such as a bridged AP) which are invisible to ICMP probes.

4.5.1 Experimental Setup

To adjust the bottleneck location we modify the experimental setup described in the Section 4.6 so that packets between the sender and the AP are now routed via a middlebox which allows us to adjust the bandwidth of the backhaul link, see Figure 4.16. We adjust the bandwidth using two different techniques: (i) by forcing the ethernet link to operate at either 100Mbps or 1000Mbps (using `ethtool`) and (ii) by generating cross-traffic on the backhaul link. When adjusting the link rate we also adjust the link queue size corresponding e.g. when changing to 100Mbps we set `txqueuelen` to 100 packets.

As client stations within the WLAN we use a Samsung Galaxy Tab S3 (Client 1) and a Google Pixel 2 (Client 2).

4.5.2 Bottleneck Classification: Ethernet Rate Limiting

4.5.2.1 Feature Selection

We begin by considering when the bandwidth is limited by the ethernet link rate. We use a supervised learning approach to try to build a bottleneck classifier. To proceed we collect training data for a range of send and link rates (the link ethernet rate is varied between 100Mbps and 1000Mbps using `ethtool`). Using timestamps for each 802.11ac frame we extract the aggregation level and the number of packets lost. For the latter we insert a packet id into the body of each packet and count "holes" in the sequence of received id's as losses – this is after accounting for link layer retransmissions, so the losses are due to queue overflow. We use these values for the last n frames as input features. That is, the input feature vector associated with the i frame is:

$$X^{(i)} = [N_{i-n+1} \quad N_{i-n} \quad \dots \quad N_i \quad L_i^p]^T \tag{4.6}$$

where N_j is aggregation of the jth frame and L_i^p is the fraction of observed packets lost out of the last p packets received. We define target variable

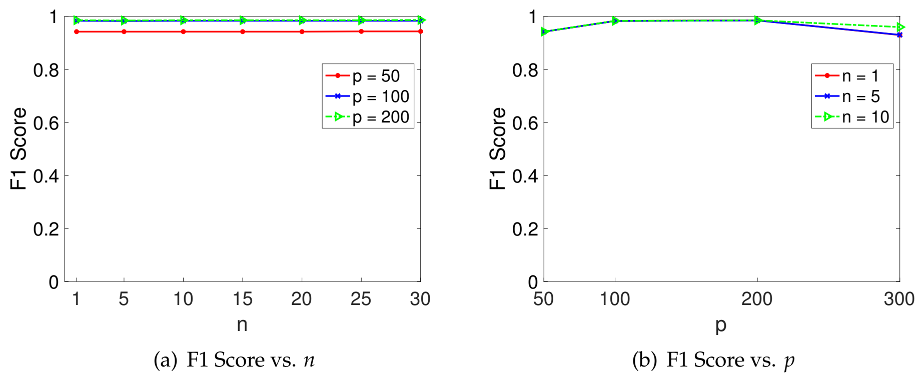


FIGURE 4.17: Performance of the logistic regression classifier vs the number n of input features and packets p used. MAC timestamps, experimental data, Samsung Galaxy handset, setup in Section 4.5.1, AMDSU aggregation ($N_{max} = 128$). Ethernet rate limiting (100/1000Mbps).

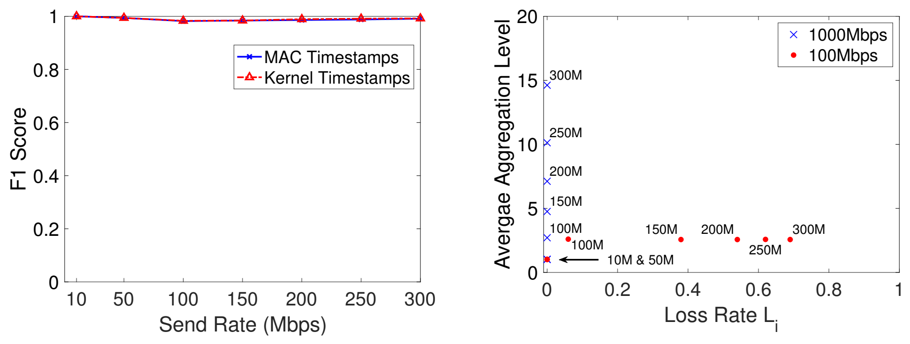
$Y^{(i)}$ as taking value 1 when the bottleneck is in the backhaul link and 0 otherwise.

Once again we try to use logistic regression to perform the classification. Performance is measured as the F1 score, with 20-fold cross-validation used. Figure 4.17 plots the measured performance vs the number n of input features used and the number of packets p. Note that for each value of n and p we hold the classifier parameters fixed i.e we use the same classifier for the full range of send rates and network configurations. Data is shown for when MAC timestamps are used to measure the aggregation level N_j but the performance is similar when kernel timestamps are used. The F1 score is above 90% for all values of n and p but is slightly higher for p in the range 100-200 packets. Note that the case of $n = 1$ corresponds to simply thresholding on the aggregation level and loss rate of the current frame. We use $n = 5$ and $p = 100$ in the following, but it can be seen that the results are not sensitive to these choices.

4.5.2.2 Classifier Performance

Figure 4.18(a) plots the measured classification accuracy vs the send rate when $n = 5$ and $p = 100$. Each point includes data collected for 100Mbps and 1000Mbps link rates. When the link rate is 100Mbps the backhaul link acts as the bottleneck for send rates of 100Mbps and above, when the link rate is 1000Mbps the WLAN acts as the bottleneck for send rates of around 500Mbps and above. It can be seen the classification accuracy is very high, close to 100% (we do not show data for send rates above 300Mbps in Figure 4.18(a) but for higher send rates the accuracy is also close to 100%).

We can gain some insight into this good performance from Figure 4.18(b), which plots the aggregation level vs the loss rate L_i for a range of send rates. The send rate is indicated beside each point and the points marked by an $\times$ are when the backhaul is 1000Mbps while those marked by $\bullet$ are when the backhaul is 100Mbps. When the backhaul is 1000Mbps it can be seen that the loss rate L_i stays close to zero while



(a) F1 Score vs. Send Rate (b)

FIGURE 4.18: Performance of the logistic regression classifier, $n = 5$, $p = 100$. Experimental data, Samsung Galaxy handset, setup in Section 4.5.1, AMDSU aggregation ($N_{max} = 128$). Ethernet rate limiting (100/1000Mbps).

the aggregation level increases with send rate. When the backhaul is 100Mbps it can be seen that the loss rate increases for send rates above 100Mbps while the aggregation level remains low at all rates. Hence, we can easily separate the points at the top left and bottom right corners of this plot, corresponding to higher send rates. At send rates around 100Mbps the points for 100Mbps and 1000Mbps backhaul are quite close but the classifier is still accurate.

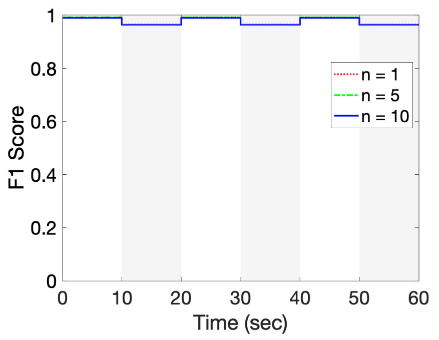
Similar performance is observed when the transmitting to two WLAN clients as might be expected (as already noted, there is per station queueing at the AP).

4.5.3 Bottleneck Classification: Cross-Traffic

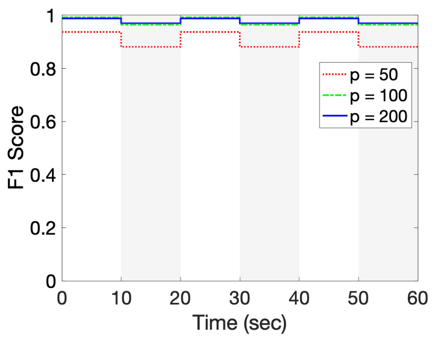
We now consider situations where there is cross-traffic sharing the backhaul link to the AP. When this cross-traffic is sufficiently high then the bottleneck for the WLAN traffic shifts from the WLAN to the backhaul, and vice versa when the level of cross-traffic falls.

We collect data for a setup where the backhaul link to the AP is gigabit ethernet. When the WLAN is the bottleneck the send rate with a single client station is around 500Mbps, e.g. see Figure 4.15. We use iperf to generate UDP cross-traffic of 600Mbps to move the bottleneck from the WLAN to the backhaul link. Figures 4.19(a)-(b) shows time histories of the measured F1 score of the logistic regression bottleneck classifier as the cross-traffic switches on and off, so moving the bottleneck back and forth between backhaul and WLAN. The gray shaded areas indicate when the bottleneck lies in the backhaul link i.e. when the cross-traffic is active. Results are shown as the number n of features used and the number of packets p used to estimate the loss rate are both varied. It can be seen that the performance is insensitive to the choice of n and close to 100% when $p \geq 100$.

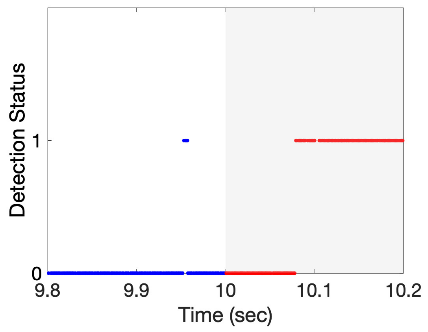
Using $n = 5$ and $p = 100$ (the same as used in the previous section), Figures 4.19(c)-(d) show more detail of the performance time histories. Figure 4.19(c) shows the classifier output following a transition of the bottleneck from the WLAN to the backhaul, and Figure 4.19(d) shows



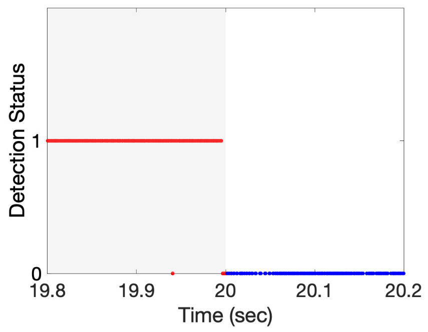
(a) $p = 100$



(b) $n = 5$



(c) $n = 5, p = 100$



(d) $n = 5, p = 100$

FIGURE 4.19: Performance of logistic regression classifier as the bottleneck moves between WLAN and backhaul due to cross-traffic (gray shaded areas indicate when the bottleneck lies in the backhaul link i.e. when the cross-traffic is active). Experimental data, setup in Section 4.5.1, AMDSU aggregation ($N_{max} = 128$), gigabit ethernet backhaul.

the output for a transition of the bottleneck from backhaul to WLAN. It can be seen that the estimator detects the transitions within 100ms or less. We observe similar performance under a range of network conditions, including for data from a production eduroam network, but omit it here since it adds little.

4.6 Hardware & Software Setup

4.6.1 Experimental Testbed

Our experimental testbed uses an Asus RT-AC86U Access Point (which uses a Broadcom 4366E chipset and supports 802.11ac MIMO with up to four spatial streams [49]). It is configured to use the 5GHz frequency band with 80MHz channel bandwidth. This setup allows high spatial usage (we observe that almost always three spatial streams are used) and high data rates (up to MCS 9). Note that we also carried out experiments with different chipsets at the AP (e.g., QCA chipsets) and did not observe any major differences. By default aggregation supports AMSDU's and allows up to 128 packets to be aggregated in a frame (namely 64 AMSDUs each containing two packets). In our tests in Section 4.3 we disabled AMSDU's to force AMPDU aggregation since this facilitates monitoring, in which case up to 64 packets can be aggregated in a frame.

A Linux server connected to this AP via a gigabit switch uses iperf 2.0.5 to generate UDP downlink traffic to the WLAN clients. Iperf inserts a sender-side timestamp into the packet payload and since the various machines are tightly synchronized over a LAN this can be used to estimate the one-way packet delay (the time between when a packet is passed into the socket in the sender and when it is received). Note, however, that in production networks accurate measurement of one-way delay is typically not straightforward as it is difficult to maintain accurate synchronization between server and client clocks (NTP typically only synchronizes clocks to within a few tens of milliseconds).

In Section 4.3, where the clients use MAC timestamps to measure the aggregation level of received frames, the WLAN clients are Linux boxes running Debian Stretch and with Broadcom BCM4360 802.11ac NICs.

In Section 4.4 a non-rooted Samsung Galaxy Tab S3 running Android Oreo is used as the client (we also carried out experiments using a non-rooted Google Pixel 2 running Android Pie and did not observe any significant differences). Due to the lack of root privilege this client is restricted to using kernel timestamps to estimate the aggregation level of received frames. A separate machine running Debian Stretch and equipped with a Broadcom BCM4360 802.11ac NIC is used in monitor mode to sniff network traffic and so provide "ground truth" since it can log MAC timestamps. Iperf inserts a unique id number into the packet payload and this is used to synchronize measurements taken by the sniffer and by the mobile handset and in this way we can measure both the handset kernel timestamp and the packet MAC timestamp. The antennas of the sniffer are placed in the path between AP and handset so as to improve reception with MIMO operation, and it is verified that all frames are captured.

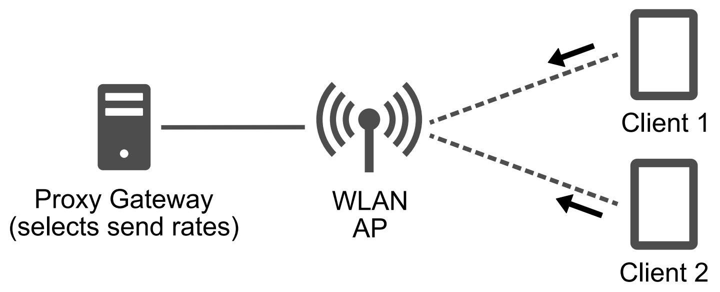


FIGURE 4.20: Schematic of scheduler architecture. Clients send reports of observed aggregation level and MCS rate to proxy which then uses this information to adjust the downlink send rate to each station.

4.6.2 Prototype Rate Allocation Implementation

We implemented feedback of measured aggregation level from the clients to the sender using the software architecture illustrated in Figure 4.20. Clients measure/estimate the aggregation level of received frames and periodically report this data back to the sender at intervals of Δ seconds as the payload of a UDP packet. The sender uses a modified version of iperf 2.0.5 where we implement the feedback collector and our rate control algorithm (see later for details). Recall that we are considering next generation edge transports and so the sender would typically be located in the cloud close to the network edge. While it may be located on the wireless access point this is not essential, and indeed we demonstrate this feature in all of our experiments by making use of a proprietary closed access point.

4.6.3 NS3 Simulator Implementation

While we mainly use experimental measurements, to allow performance evaluation with larger numbers of client stations and with controlled changes to channel conditions we also implemented our approach in the NS3 simulator. Based on the received feedbacks it periodically configures the sending rate of `udp-client` applications colocated at a single node connected to an Access Point. Each wireless station receives a UDP traffic flow at a `udp-server` application that we modified to collect frame aggregation statistics and periodically transmit these to the controller at intervals of Δ ms. We configured 802.11ac to use a physical layer operating over an 80MHz channel, VHT rates for data frames and legacy rates for control frames, PHY MCS=9 and with the number of spatial streams NSS = 2 i.e. similarly to the experimental setup. As validation we reproduced a number of the simulation measurements in our experimental testbed and found them to be in good agreement.

4.7 Summary & Conclusions

In this chapter we consider transport layer approaches for achieving high rate, low delay communication over edge paths where the bottleneck is an 802.11ac WLAN which can aggregate multiple packets into each frame. We first show that regulating send rate so as to maintain a target aggregation level can be used to realize high rate, low latency communication over 802.11ac WLANs. We then address two important practical issues arising in production networks, namely that (i) many client devices are non-rooted mobile handsets/tablets and (ii) the bottleneck may lie in the backhaul rather than the WLAN, or indeed vary between the two over time. We show that both these issues can be resolved by use of simple and robust machine learning techniques. We present a prototype transport layer implementation of our low latency rate allocation approach and use this to evaluate performance under real radio channel conditions.

Chapter 5

Regulating Queueing Delay in 802.11ac WLANs: Nonlinear Controller Analysis & Design

5.1 Introduction

In the previous chapter, we proposed a new transport layer approach for achieving high rate and low delay communications over 802.11ac WLANs using aggregation. In this chapter, we study and develop a simple and useful analytic model of the dependence on send rate of the mean aggregation level and delay and use this to refine the rate allocation algorithm.

5.2 Modeling Aggregation Level & Delay

Figure 5.1 shows example time histories of the frame aggregation level and delay as the send rate is increased from around 50 to 300Mbps. These illustrate a number of features that will be of interest to us. Firstly, observe in Figure 5.1(a) that the send rate is updated every 0.5s and held constant in between updates, and this is the update interval that we will use in our control design. While the send rate varies relatively

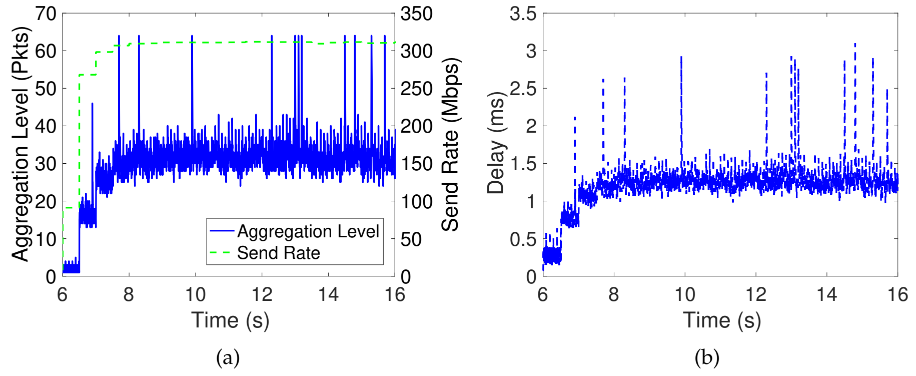


FIGURE 5.1: Example time histories of frame aggregation level and packet delay as send rate is varied. MCS 9, NSS 1, NS3 (see Section 5.5 for details).

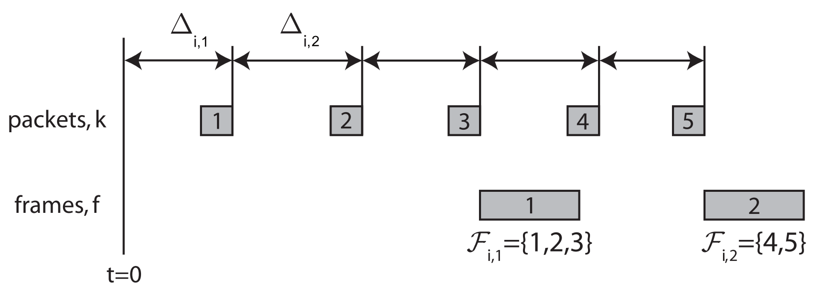


FIGURE 5.2: Illustrating notation used. Packets arriving at the AP for transmission to station i are indexed $k = 1, 2, \ldots$ with the time between packet $k-1$ and packet k being $\Delta_{i,k}$. Frames transmitted by the AP to station i are indexed $f = 1, 2, \ldots$ and the set of packets sent in frame f is $\mathcal{F}_{i,f}$.

slowly it can be seen that there are rapid short-term fluctuations in the aggregation level about a value that roughly tracks the send rate. Observe that the magnitude of the fluctuations varies with the send rate e.g. they are significantly lower in the early part of the time history around 6-6.5s, where the send rate is lower, than from 10s onwards. Hence, for control design we are interested in modeling the dependence on send rate of both the mean aggregation level (where the mean is taken over the short-term fluctuations) and the magnitude of the fluctuations in aggregation level.

With regard to delay, it can be seen from Figure 5.1(b) that the delay exhibits strikingly similar behavior to the aggregation level. This is not by accident since, as already noted, both are intimately related.

In this section we develop simple analytic models of the dependence on send rate of the mean aggregation level and delay suitable for control design. We also characterize the dependence of the fluctuations in aggregation level with send rate and network configuration.

5.2.1 Basic Setup

We consider downlink transmissions in a WLAN (so no collisions) with n client stations indexed by $i = 1, 2, \ldots, n$. Index the packets arriving at the AP for transmission to station i by $k = 1, 2, \ldots$ and let $\Delta_{i,k}$ denote the inter-arrival time between packet $k-1$ and packet k. Recall that we control the packet sender and so for simplicity we assume that this uses packet pacing. That is, the sender aims to transmit packets with fixed spacing, although end host constraints typically mean that this aim is only approximately achieved and the packet spacing has some jitter. We can therefore assume that the $\Delta_{i,k}$ are i.i.d. with $E[\Delta_{i,k}] = \Delta_i = 1/x_i$ where x_i is the send rate to station i in packets/sec.

Packets are transmitted to station i by the AP within 802.11 frames. Index these frames by $f = 1, 2, \ldots$ ($f = 1$ is the first frame sent, and so on) and let $\mathcal{F}_{i,f} \subset \{1, 2, \ldots\}$ denote the set of packets aggregated within frame f transmitted to station i. Then $N_{i,f} = |\mathcal{F}_{i,f}|$ is the number of packets aggregated. Since a minimum of one packet must be contained within a frame and a maximum of N_{max} (typically 32 or 64 packets) then $1 \leq N_{i,f} \leq N_{max}$. The setup is illustrated in Figure 5.2.

5.2.2 Frame Transmission Timing

Suppose, for simplicity, that all packets are of length l bits (this can be easily relaxed). The airtime used by frame f transmitted to station i is then given by

$$T_{air,i,f} := T_{oh,i,f} + \frac{l + l_{oh}}{R_{i,f}} N_{i,f} \tag{5.1}$$

where $R_{i,f}$ is the PHY rate used to transmit the payload of the frame, $T_{oh,i,f}$ is the time used for transmission overheads which do not depend on the aggregation level (namely, CSMA/CA channel access, PHY and MAC headers plus transmission of the ACK by the receiver) and l_{oh} is the MAC framing overhead (in bits) for each packet in the frame.

Assume that the AP transmits frames to stations in a round-robin fashion. We will also assume that the packet arrival rate is high enough that $N_{i,f} \geq 1, i = 1, \ldots, n$ i.e. the AP transmits at least one packet to each station in every round. This is reasonable since our primary interest here is in the high rate operation needed for next generation edge-assisted applications and $N_{i,f} \geq 1$ is ensured for sufficiently high rates (we give a lower bound on the rate needed at the end of the next section). Then the duration Ω_f of round f is given by,

$$\Omega_f = \sum_{j=1}^{n} T_{air,j,f} = C_f + \sum_{j=1}^{n} \frac{(l + l_{oh})}{R_{j,f}} N_{j,f} \tag{5.2}$$

where $C_f = \sum_{j=1}^{n} T_{oh,j,f}$. Index stations by the order in which they are serviced by the AP scheduler, i.e. within a round the ith frame transmitted is to station i. In general, the interval $\Omega_{i,f}$ between transmission of frames f and $f+1$ to station i is not equal to Ω_f, but under reasonable assumptions $\Omega_{i,f}$ has the same distribution as Ω_f i.e. $\Omega_{i,f} \sim \Omega_f$.

In more detail, we have that

$$\begin{aligned} \Omega_{i,f} =& \sum_{j=i}^{n} (T_{oh,j,f} + \frac{l + l_{oh}}{R_{j,f}} N_{j,f}) \\ &+ \sum_{j=1}^{i-1} (T_{oh,j,f+1} + \frac{l + l_{oh}}{R_{j,f+1}} N_{j,f+1}) \end{aligned} \tag{5.3}$$

The fixed CSMA/CA overhead $T_{oh,j,f}$ associated with channel access etc is i.i.d across stations j and frames f by virtue of the 802.11 MAC operation (fluctuations in $T_{oh,j,f}$ are due to the CSMA/CA channel access which is uniformly distributed between 0 and $CW - 1$ MAC slots, where CW is the 802.11 contention window). We therefore have that $\sum_{j=i}^{n} T_{oh,j,f} + \sum_{j=1}^{i-1} T_{oh,j,f+1} \sim C_f$. Assume the channel is stationary so that the MCS rate $R_{i,f}$ is identically distributed across frames f (but of course may vary amongst stations). Assume also that changes in the distribution of the aggregation level $N_{i,f}$ occur at a much slower time-scale than an AP scheduler round so that $N_{i,f}$ and $N_{j,f+1}$ can be approximated as being identically distributed. This is the key modeling approximation that we make but, as will see later, the control actions we consider which change the distribution of $N_{i,f}$ occur on a time-scale of 0.5-1s, whereas a scheduler round takes no more than 2-3ms, so this assumption is reasonable. It then follows that $\Omega_{i,f} \sim \Omega_f$.

Assuming $N_{i,f}$ and $R_{i,f}$ are independent and vary in an i.i.d. manner across frames[1] we now have that

$$\begin{aligned} E[\Omega_{i,f}] = E[\Omega_f] &= c + \sum_{j=1}^{n} \frac{l + l_{oh}}{\mu_{R_j}} E[N_{j,f}] \\ &= c + \boldsymbol{w}^T E[\boldsymbol{N}_f] \end{aligned} \tag{5.4}$$

where $c := E[C_f] = nE[T_{oh,j,f}]$, $\mu_{R_j} := 1/E[\frac{1}{R_{j,f}}]$ (note that in general[2] $E[\frac{1}{R_{j,f}}] \neq 1/E[R_{j,f}]$), $\boldsymbol{w} = (\frac{l+l_{oh}}{\mu_{R_1}}, \dots, \frac{l+l_{oh}}{\mu_{R_n}})^T$, $\boldsymbol{N}_f = (N_{1,f}, \dots, N_{n,f})^T$.

5.2.3 Mean Aggregation Level

To model the aggregation behavior we proceed as follows. Recall $\Omega_{i,f}$ is the interval between transmission of frame f and frame $f+1$ to station i. Let $P_{i,f}$ denote the number of packets arriving at the AP during this interval. When the time between packets is constant with $\Delta_{i,k} = \Delta_i$ then $P_{i,f} = \Omega_{i,f}/\Delta_i$ (ignoring quantization effects) but, as already noted, in general we expect some jitter between packet arrivals even when the sender paces its transmissions.

These packets are buffered in a queue at the AP until they can be transmitted. Letting $q_{i,f}$ denote the queue occupancy at the time when frame f is transmitted then $q_{i,f} = [q_{i,f-1} + P_{i,f} - N_{i,f}]^+$. It is reasonable to suppose that the AP aggregates as many as possible of these packets queued for transmission into the next frame f, in which case $N_{i,f} = \min\{q_{i,f-1} + P_{i,f}, N_{max}\}$ and

$$q_{i,f} = [q_{i,f-1} + P_{i,f} - \min\{q_{i,f-1} + P_{i,f}, N_{max}\}]^+ \tag{5.5}$$
$$= [q_{i,f-1} + P_{i,f} - N_{max}]^+ \tag{5.6}$$

There are three operating regimes to consider:

1. Firstly, when $E[P_{i,f}] > N_{max}$ then the queue is unstable. The queue occupancy grows and so $N_{i,f} = N_{max}$ eventually for all frames f. This regime is not of interest in the present work where our focus is on low delay operation.

2. Secondly, our main interest is in the regime where the queue backlog remains low i.e. $P_{i,f} < N_{max}$. The queue is cleared by each frame transmission so $q_{i,f} = 0$ and $N_{i,f} = P_{i,f}$.

[1] Variations in $N_{i,f}$ across frames arise due to fluctuations in the time $\Omega_{i,f}$ between transmission opportunities as a result of the stochastic nature of the 802.11 MAC (when the time $\Omega_{i,f}$ is longer then it is likely that more packets have arrived and $N_{i,f}$ is larger, conversely $N_{i,f}$ tends to be smaller when $\Omega_{i,f}$ is shorter). Variations in PHY rate $R_{i,f}$ primarily arise due to channel fluctuations. Hence, assuming independence of $N_{i,f}$ and $R_{i,f}$ seems reasonable and is also consistent with our experimental measurements. Assuming that the $N_{i,f}$, $f = 1, \dots$ are i.i.d. is reasonable since the contention times $T_{oh,j,f}$ are i.i.d. Similarly, provided channel variations are i.i.d then it is reasonable to assume the $R_{i,f}$, $f = 1, \dots$ are i.i.d.

[2] Indeed, to first-order $E[\frac{1}{R_{j,f}}] \approx \frac{1}{E[R_{j,f}]} + \frac{Var(R_{j,f})}{E[R_{j,f}]^3}$.

3. Thirdly, there is the transition regime between regimes one and two where $E[P_{i,f}] < N_{max}$ but $P_{i,f}$ may sometimes be greater than N_{max} and $q_{i,f}$ may be non-zero.

In regime two, $N_{i,f} = P_{i,f}$. Taking expectations $E[N_{i,f}] = E[P_{i,f}] = E[\Omega_{i,f}]/E[\Delta_{i,k}]$ (by renewal-reward theory since the $\Delta_{i,k}$ are i.i.d and independent of $\Omega_{i,f}$). Let $\mu_{\boldsymbol{N}} = (\mu_{N_1}, \dots, \mu_{N_n})^T$ with $\mu_{N_i} := E[N_{i,f}]$, and recall that $x_i := 1/E[\Delta_{i,k}] = 1/\Delta_i$ is the send rate of station i. Then substituting from (5.4) it follows that $\mu_{\boldsymbol{N}} = (c + \boldsymbol{w}^T \mu_{\boldsymbol{N}})\boldsymbol{x}$. Rearranging yields

$$\mu_{\boldsymbol{N}} = \frac{c\boldsymbol{x}}{1 - \boldsymbol{w}^T\boldsymbol{x}} \tag{5.7}$$

where $\boldsymbol{x} = (x_1, \dots, x_n)^T$ is the vector of station send rates.

To simplify the analysis we assume that the third regime can be lumped with regime two[3]. In regime three $E[P_{i,f}] > N_{max}$ and $E[N_{i,f}] = N_{max}$. Incorporating the N_{max} constraint into (5.7) gives

$$\mu_{\boldsymbol{N}} = \Pi \circ \frac{c\boldsymbol{x}}{1 - \boldsymbol{w}^T\boldsymbol{x}} = \Pi \circ \boldsymbol{F}(\boldsymbol{x}) \tag{5.8}$$

where Π denotes projection onto interval $[1, N_{max}]$ and $\boldsymbol{F}(\boldsymbol{x}) := \frac{c\boldsymbol{x}}{1-\boldsymbol{w}^T\boldsymbol{x}}$. Note that $x_i \geq 0$, $i = 1 \dots, n$ and $\boldsymbol{w}^T\boldsymbol{x} < 1$ are required for rate vector $\boldsymbol{x}$ to be feasible and so $\boldsymbol{F}(\boldsymbol{x}) \geq \boldsymbol{0}$. Also that $x_i \geq 1/(c + w_i) \geq (1 - \sum_{i \neq j} w_i x_i)/(c + w_i)$, $i = 1 \dots, n$ is sufficient to ensure that $\boldsymbol{F}(\boldsymbol{x}) \geq \boldsymbol{1}$.

5.2.4 Mean Delay

Recall $\mathcal{F}_{i,f} \subset \{1, 2, \dots\}$ is the set of packets in frame f sent to station i and that these packets arrive with inter-arrival times $\Delta_{i,k}$, $k \in \mathcal{F}_{i,f}$. In operating regime two (see above), the queue is cleared after each transmission. Hence, the first packet in frame f arrives to an empty queue and must wait $\sum_{k \in \mathcal{F}_{i,f}} \Delta_{i,k}$ seconds before the last packet in the frame arrives at the AP and so becomes available for aggregation. The delay experienced by the first packet in frame f (and so by all other packets sharing this frame) is at most $\sum_{k \in \mathcal{F}_{i,f}} \Delta_{i,k}$. This upper bound is attained if the frame is transmitted right before arrival of the first packet sent in the next frame $f + 1$ since if frame f was transmitted later then this packet would have been added to frame f. That is, mean packet delay at the AP is upper bounded by,

$$\mu_{T_i} = E[\sum_{k \in \mathcal{F}_{i,f}} \Delta_{i,k}] = E[N_{i,f}]\Delta_i = \frac{\mu_{N_i}}{x_i} \tag{5.9}$$

$$= \max\{\min\{\frac{c}{1 - \boldsymbol{w}^T\boldsymbol{x}}, \frac{N_{max}}{x_i}\}, \frac{1}{x_i}\} \tag{5.10}$$

[3]Our measurements in Section 5.2.6 support the validity of this simplifying assumption. In practice it amounts to assuming that the system transitions quickly between operating regimes one and two, i.e. regime three is only transient.

where recall $N_{i,f} = |\mathcal{F}_{i,f}|$ is the number of packets in the frame and $x_i = 1/\Delta_i$.

5.2.5 Invertibility of Map From Rate To Aggregation Level

Observe that $\boldsymbol{F}(\boldsymbol{x})$ is monotonically increasing for feasible rate vectors $\boldsymbol{x}$ since $\frac{\partial F_i'(\boldsymbol{x})}{\partial x_i} = \frac{c}{(1-\boldsymbol{w}^T\boldsymbol{x})^2}(1-\boldsymbol{w}^T\boldsymbol{x}+w_i x_i) > 0$ and $\frac{\partial F_i'(\boldsymbol{x})}{\partial x_j} = \frac{c w_i x_i}{(1-\boldsymbol{w}^T\boldsymbol{x})^2} > 0$ when $x_i \geq 0$ and $\boldsymbol{w}^T\boldsymbol{x} < 1$. Hence, $\boldsymbol{F}(\boldsymbol{x})$ is one-to-one and so invertible. In particular,

$$\boldsymbol{F}^{-1}(\mu_{\boldsymbol{N}}) = \frac{\mu_{\boldsymbol{N}}}{c + \boldsymbol{w}^T \mu_{\boldsymbol{N}}} \tag{5.11}$$

and it can be verified that $\boldsymbol{F}(\boldsymbol{F}^{-1}(\mu_{\boldsymbol{N}})) = \mu_{\boldsymbol{N}}$.

Given rate vector $\boldsymbol{x}$ we can therefore obtain the corresponding aggregation level from $F(\boldsymbol{x})$ and, conversely, given aggregation level vector $\mu_{\boldsymbol{N}}$ we can obtain the corresponding rate vector from $\boldsymbol{F}^{-1}(\mu_{\boldsymbol{N}})$. This will prove convenient in the analysis below since it means we can freely change variables between $\boldsymbol{x}$ and $\mu_{\boldsymbol{N}}$. For example, substituting $\boldsymbol{x} = \boldsymbol{F}^{-1}(\mu_{\boldsymbol{N}})$ the term $\frac{c}{1-\boldsymbol{w}^T\boldsymbol{x}}$ in the mean delay (5.10) can be expressed equivalently in terms of $\mu_{\boldsymbol{N}}$ as,

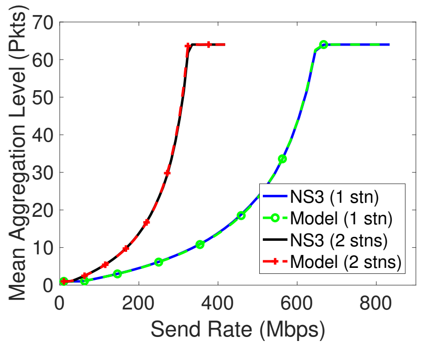
$$\frac{c}{1-\boldsymbol{w}^T\boldsymbol{x}} = c + \boldsymbol{w}^T\mu_{\boldsymbol{N}} \tag{5.12}$$

5.2.6 Validation Of Mean Model

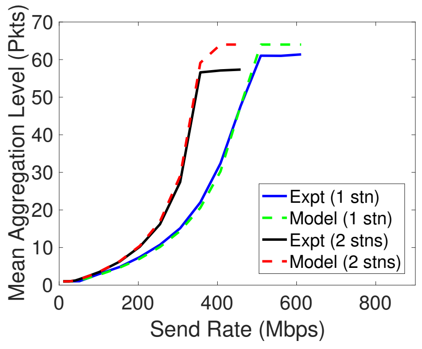
As partial validation of the mean aggregation level model (5.8) in Figure 5.3(a) we compare its predictions against measurements from the NS3 detailed packet level simulator as the send rate is varied. Data is shown for the case of a single client station and when there are two client stations both with the same send rate. The values of the model parameters c and w are derived from the 802.11ac MAC/PHY settings (80MHz channel, MCS 9, NSS 2). It can be seen that the model is in remarkably good agreement with the simulation data. We also collected measurements of aggregation level vs send rate in an experimental testbed, see Section 5.5 for details. Figure 5.3(b) compares these experimental measurements against the model predictions[4] and again it can be seen that there is excellent agreement between the model and the measurements.

The model (5.8) predicts that the aggregation level scales as the reciprocal of $1 - \boldsymbol{w}^T\boldsymbol{x} = 1 - \sum_{i=1}^{n} L/\mu_{R_i}$. Figure 5.3(c) compares the model predictions as the MCS rate μ_{R_i} is varied (for the 802.11ac setting used $\mu_{R_i} =$ 87.8Mbps at MCS index 0 increasing to 1170Mbps at MCS index 9). The model also predicts that for the ratio of the aggregation level of two stations is proportional to the ratio of their send rates and this behavior is evident in Figure 5.3(d) which plots the aggregation level for two stations when the send rate to the first station is increased from

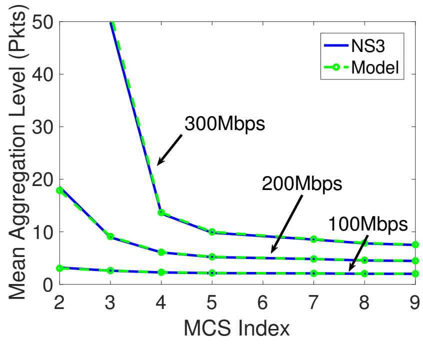
[4] 802.11ac settings: NSS=3, 80Mhz channel, the MCS used fluctuates over time due to the action of the 802.11ac rate controller and so an average value is used. The model c and w parameter values used in Figure 5.3(b) are $c = 270\mu s$, $\mu_R =$ 585Mbps for the one station data and $c = 320\mu s$, $\mu_R =$ 850Mbps for the two station data.



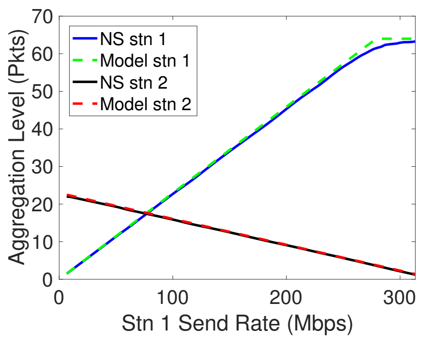
(a) One and two stations (same send rate), MCS=9, NSS=2, NS3.



(b) One and two stations (same send rate), testbed data.



(c) One station, MCS and send rate varied, NSS=3, NS3.



(d) Two stations with different send rates and MCSs. MCS 9 for station 1, MCS 3 for station 2, NSS=1, NS3.

FIGURE 5.3: Comparison of model (5.8) with measurements. Data is shown for sending UDP packets to one and two client stations, the same send rate being used to all stations and indicated on the x-axes of the plots. Plots (a),(c),(d) compare the mean aggregation level measured from NS3 simulations with the model, plot (b) compares measurements from an experimental testbed, see Section 5.5 for details. In (a)-(c) when there are two stations they have the same send rate, in (d) the stations have different send rates: the send rate to station 1 increases from 5 to 310Mbps while the send rate to station 2 decreases from 100 to 5Mbps.

5 to 310Mbps while that to the second station is decreased from 100 to 5Mbps.

In summary, the model (5.8) is in good agreement with measurements with regard to the dependence of aggregation level on overall send rate, MCS and ratio of station send rates.

5.2.7 Fluctuations Around Mean

5.2.7.1 Approximate Model

Equation (5.7) models the relationship between the arrival rate $\boldsymbol{x}$ and the mean aggregation level $\mu_{\boldsymbol{N}}$ when operating in regime two. We can also get an approximate model, useful for control design, of the fluctuations $\eta_{N_{i,f}} = N_{i,f} - \mu_{N_i}$ about the mean as follows. Neglecting the jitter in the packet inter-arrival times then the number of packets $P_{i,f}$ arriving at the AP during round f is approximately $\Omega_{i,f} x_i$. That is, the fluctuations in $P_{i,f}$ (and so $N_{i,f}$) are induced by fluctuations in the duration $\Omega_{i,f}$ of the scheduling round for station i. Neglecting the impact of the position of each station within a round then $\Omega_{i,f} \approx \Omega_f$ (this is exact in the case of a single station). Combining these we obtain the model

$$\boldsymbol{N}_{f+1} = (C_f + \boldsymbol{w}^T \boldsymbol{N}_f)\boldsymbol{x} \tag{5.13}$$

Since $\mu_{\boldsymbol{N}} = (c + \boldsymbol{w}^T \mu_{\boldsymbol{N}})\boldsymbol{x}$ it follows that

$$\eta_{\boldsymbol{N}_{f+1}} = \boldsymbol{x}\boldsymbol{w}^T \eta_{\boldsymbol{N}_f} + (C_f - c)\boldsymbol{x} \tag{5.14}$$

where $\eta_{\boldsymbol{N}_f} = [\eta_{N_{1,f}}, \ldots, \eta_{N_{n,f}}]^T$. Observe that $\eta_{\boldsymbol{N}_f}$ evolves according to first-order dynamics driven by i.i.d stochastic input $(C_f - c)\boldsymbol{x}$. In 802.11ac $C_f - c$ is a random variable uniformly distributed between 0 and 135μs (CW_{min} is 16 and a MAC slot is 9μs).

Figure 5.4(a) compares the predictions of the standard deviation of $\eta_{\boldsymbol{N}_f}$ calculated using the model (5.14) with measurements of the standard deviation of the aggregation level from NS3. Data is shown as the send rate and MCS rate are varied. It can be seen that the model predictions are in good agreement with the measurements except when the aggregation level hits its maximum value N_{max}, at which point the standard deviation of the measured data falls sharply to zero. That is, the model (5.14) is accurate within operating regime two but not in operating regime three, as expected.

Observe that the standard deviation of $\eta_{\boldsymbol{N}_f}$ increases with the send rate, which is intuitive. The main source of the fluctuations in $\boldsymbol{N}_f$ is the randomness in the channel access time associated with the operation of the CSMA/CA MAC. During a round where the channel access randomness leads to round being of longer than average duration then more packets arrive than on average, with the number arriving increasing with the send rate. At the next round these packets form the next frame, which is therefore larger than average. The magnitude of the fluctuations $\eta_{\boldsymbol{N}_f}$ in the frame size therefore tends to increase with the send rate.

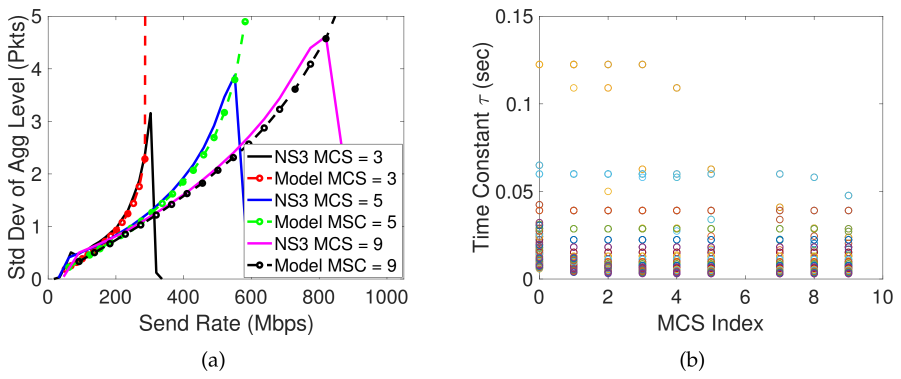


FIGURE 5.4: (a) Comparison of the standard deviation of η_{N_f} calculated using the model (5.14) with measurements from NS3, NSS=3. (b) time constant of dynamics (5.14) as the number n of stations is varied between 1 and 20, NSS is varied from 1 to 3 and the MCS index is varied fro 0 to 9 i.e. covering the full 802.11ac NSS/MCS range.

Note that larger frames also tends to make the next round longer than average since they take longer than average to transmit. This creates feedback whereby a random fluctuation in the duration of a round tends to create changes that persist for several rounds. It is this feedback that is reflected in the dynamics (5.14).

The measurement data in Figure 5.4(a) includes packet inter-arrival jitter of $\pm 6\mu$s. We also collected measurements for other values of jitter and found the standard deviation of η_{N_f} to be largely insensitive to the level of pacing jitter.

5.2.7.2 Time-Scale of Dynamics

The matrix $\boldsymbol{x}\boldsymbol{w}^T$ is rank one and has one zero eigenvalue $\boldsymbol{w}^T\boldsymbol{x} = \sum_{i=1}^{n} w_i x_i = \sum_{i=1}^{n}(l + l_{oh})x_i/\mu_{R_i}$ and an eigenvalue of zero with multiplicity $n-1$. The time constant of the dynamics is therefore $\tau : - = E[\Omega_f]/\log(\boldsymbol{w}^T\boldsymbol{x})$. Substituting for $E[\Omega_f]$ and from (5.11) for the send rate $\boldsymbol{x}$ then gives $\tau = -(c + \boldsymbol{w}^T\boldsymbol{N})/\log(\boldsymbol{w}^T\boldsymbol{N}/(c + \boldsymbol{w}^T\boldsymbol{N}))$. Figure 5.4(b) plots the value of this time constant as the number of stations is varied from 1 to 20, NSS is varied from 1 to 3 and the MCS index from 0 to 9. For each configuration the aggregation level N is the minimum of N_{max} and the level for which the mean delay μ_T is 5ms. It can be seen that the time constant is never more than about 0.12s, and tends to fall with increasing MCS rate. As we will see later, the online rate allocation algorithm we consider updates the packet send rate every Δ seconds, with Δ typically 0.5s or 1s and so operates at significantly longer time scales than the dynamics (5.14).

5.2.8 Measurement Noise & Main Source Of Model Uncertainty

5.2.8.1 Measurement Noise

The aggregation level $N_{i,f}$ can be observed at receiving station i via radiotap/prism libpcap packet headers via MAC timestamps, see Chapter 4. As already noted, our online rate allocation algorithm updates the packet send rate every Δ seconds. We can therefore estimate the mean aggregation level μ_{N_i} via the empirical average $\tilde{\mu}_{N_i}(k) = \sum_f N_{i,f}$ of frames sent to station i over interval $[0, \Delta k)$, $k = 1, 2, \dots$. As discussed in Section 5.2.7, $N_{i,f}$ fluctuates due to the MAC channel access randomness and this means that estimate $\tilde{\mu}_{N_i}$ is subject to significant measurement noise.

The model expressions (5.8) and (5.10) for the mean aggregation level and delay involve parameter $\boldsymbol{w} = (\frac{l+l_{oh}}{\mu_{R_1}}, \dots, \frac{l+l_{oh}}{\mu_{R_n}})^T$. The packet size l and framing overhead l_{oh} are known and the MCS rate $R_{i,f}$ used to send frame f to station i can be observed by receiving station i (again via radiotap/prism libpcap packet headers) and we can therefore estimate μ_{R_i} via the empirical average $1/\tilde{\mu}_{R_i}$ of the $1/R_{i,f}$ over interval Δ. This estimate suffers from measurement noise induced by fluctuations in the empirical mean of $R_{i,f}$ over interval Δ. However, typically the channel is fairly stable over short intervals and these fluctuations are small, thus the level of this measurement noise is low.

5.2.8.2 Model Uncertainty

The model expressions (5.8) and (5.10) also involve parameter $c = n\mu_{T_{oh}}$. The number n of stations to which downlink transmissions are ongoing is known but the mean channel access time $\mu_{T_{oh}}$ is harder to determine accurately since it cannot be measured directly (since we consider the transport layer we assume we do not have access to the MAC on the AP) and it depends on the channel state and so may be strongly affected by neighboring WLANs, interference etc. Hence, only a fairly rough estimate of parameter c is generally available and this is the main source of model uncertainty.

5.3 Proportional Fair Low Delay Rate Allocation

5.3.1 Utility Fair Optimization

Our interest is in achieving high rates while maintaining low delay at the AP. Formally, we consider the proportional fair low delay rate allocation that is the solution to the following optimization P:

$$\max_{\boldsymbol{x} \in \mathbb{R}^n_+} \sum_{i=1}^{n} \log x_i \tag{5.15}$$

$$s.t.\ \mu_{T_i}(\boldsymbol{x}) \le \bar{T}, i = 1, \dots, n \tag{5.16}$$

$$\mu_{N_i}(\boldsymbol{x}) \le \bar{N}, i = 1, \dots, n \tag{5.17}$$

Constraint (5.16) ensures that the mean delay at the AP is no more than upper limit $\bar{T}$, where $\bar{T}$ is a QoS parameter. Constraint (5.17) ensures that we operate at an aggregation level no more than $\bar{N} < N_{max}$ and so the AP can clear the queue at each transmission opportunity i.e. there is no sustained queueing and we are operating in regime 2. Maximizing objective (5.15) ensures utility fairness.

Substituting from (5.10) the constraints (5.16) can be written[5] as $\frac{c}{1-\boldsymbol{w}^T\boldsymbol{x}} \leq \bar{T}$. Rearranging gives $c \leq \bar{T}(1-\boldsymbol{w}^T\boldsymbol{x})$ i.e. $\boldsymbol{w}^T\boldsymbol{x} \leq 1 - c/\bar{T}$. In this form it can be seen that the constraint is linear, and so convex. Similarly, substituting from (5.8) the constraints (5.17) can be written equivalently as $\frac{cx_i}{1-\boldsymbol{w}^T\boldsymbol{x}} \leq \bar{N}, i = 1, \ldots, n$. Rearranging gives $cx_i \leq \bar{N}(1-\boldsymbol{w}^T\boldsymbol{x})$ i.e. $cx_i + \bar{N}\boldsymbol{w}^T\boldsymbol{x} \leq \bar{N}$, which again is linear. Hence, optimization P can be equivalently rewritten as optimization P':

$$\max_{x \in \mathbb{R}^n_+} \sum_{i=1}^{n} \log x_i \tag{5.18}$$

$$s.t.\ \boldsymbol{w}^T\boldsymbol{x} \leq 1 - c/\bar{T} \tag{5.19}$$

$$cx_i + \bar{N}\boldsymbol{w}^T\boldsymbol{x} \leq \bar{N}, i = 1, \ldots, n \tag{5.20}$$

which is convex.

5.3.2 Characterizing The Proportional Fair Solution

The Lagrangian of optimization P' is $-\sum_{i=1}^{n} \log x_i + \theta(\boldsymbol{w}^T\boldsymbol{x} - (1 - c/\bar{T})) + \sum_{i=1}^{n} \lambda_i(cx_i + \bar{N}\boldsymbol{w}^T\boldsymbol{x} - \bar{N})$ where θ and λ_i, $i = 1, \ldots, n$ are multipliers associated with, respectively, (5.19) and (5.20).

Since the optimization is convex the KKT conditions are necessary and sufficient for optimality. Namely, an optimal rate vector $\boldsymbol{x}^*$ satisfies

$$-\frac{1}{x_i^*} + \lambda_i c + \sum_{j=1}^{n} \lambda_j \bar{N} w_i + \theta w_i = 0 \tag{5.21}$$

i.e.

$$x_i^* = \frac{1}{\lambda_i c + D w_i} \tag{5.22}$$

where $D := (\bar{N}\sum_{j=1}^{n} \lambda_j + \theta)$.

Let $U = \{i : \mu_{N_i}(\boldsymbol{x}^*) < \bar{N}\}$ denote the set of stations for which the aggregation level is less than $\bar{N}$ at the optimal rate allocation. By complementary slackness $\lambda_i = 0$ for $i \in U$ and so $x_i^* = 1/(Dw_i)$. That is, $\mu_{N_i} = \frac{cx_i^*}{(1-\boldsymbol{w}^T\boldsymbol{x}^*)} = \frac{c}{D(1-\boldsymbol{w}^T\boldsymbol{x}^*)}\frac{1}{w_i}$. Observe that the first term is invariant with i and so the aggregation level of station $i \in U$ is proportional to $1/w_i = \mu_{R_i}/L$ i.e. to the mean MCS of the station. For stations $j \notin U$ the aggregation level $\mu_{N_j}(\boldsymbol{x}^*) = \bar{N}$.

Putting these observations together, it follows that

$$\mu_{N_i}(\boldsymbol{x}^*) = \min\{\frac{c}{D(1-\boldsymbol{w}^T\boldsymbol{x}^*)}\frac{1}{w_i}, \bar{N}\},\ i = 1, \ldots, n \tag{5.23}$$

[5]Note that constraint (5.17) ensures $\mu_{N_i}(\boldsymbol{x}) \leq \bar{N} < N_{max}$ and so $\mu_{T_i}(\boldsymbol{x}) < N_{max}/x_i$. Since our interest is primarily in applications requiring high rates we assume for simplicity that $\frac{c}{1-\boldsymbol{w}^T\boldsymbol{x}} \geq \frac{1}{x_i}$ although this could be added as the additional linear constraint $cx_i + \boldsymbol{w}^T\boldsymbol{x} \geq 1$ if desired.

Assume without loss that the station indices are sorted such that $w_1 \geq w_2 \geq \cdots \geq w_n$. Then

$$\mu_{N_i}(\boldsymbol{x}^*) = \min\{\mu_{N_1}(\boldsymbol{x}^*)\frac{w_1}{w_i}, \bar{N}\},\ i = 2,\ldots,n \tag{5.24}$$

Hence, once the optimal $\mu_{N_1}(\boldsymbol{x}^*)$ is determined we can find the optimal aggregation levels for the rest of the stations. With these we can then use inverse mapping (5.11) to recover the proportional fair rate allocation, namely $x_i^* = \mu_{N_i}/(c + \boldsymbol{w}^T\mu_{\boldsymbol{N}})$.

It remains to determine μ_{N_1}. We proceed as follows.

Lemma 1. *At an optimum $\boldsymbol{x}^*$ of P' then either (i) $\mu_{N_i}(\boldsymbol{x}^*) = \bar{N}$ for all $i = 1,\ldots,n$ or (ii) $\mu_{T_i}(\boldsymbol{x}^*) = \bar{T}$ for all $i = 1,\ldots,n$.*

Proof. We proceed by contradiction. Suppose at an optimum $\mu_{N_i}(\boldsymbol{x}^*) = \frac{cx_i^*}{1-\boldsymbol{w}^T\boldsymbol{x}^*} < \bar{N}$ for some i and $\mu_{T_i}(\boldsymbol{x}^*) = \frac{c}{1-\boldsymbol{w}^T\boldsymbol{x}^*} < \bar{T}$. Then we can increase x_i^* without violating the constraints (with this change $\frac{c}{1-\boldsymbol{w}^T\boldsymbol{x}^*}$ and $\frac{cx_i^*}{1-\boldsymbol{w}^T\boldsymbol{x}^*}$ will both increase, but since the corresponding constraints are slack if the increase in x_i^* is sufficiently small then they will not be violated). Hence, we can improve the objective which yields the desired contradiction since we assumed optimality of $\boldsymbol{x}^*$. Hence when $\mu_{N_i}(\boldsymbol{x}^*) < \bar{N}$ for at least one station then $\mu_{T_i}(\boldsymbol{x}^*) = \bar{T}$. Alternatively, $\mu_{N_i}(\boldsymbol{x}^*) = \bar{N}$ for all stations. □

It follows from Lemma 1 that $\mu_{N_1} = \min\{\bar{T}x_1^*, \bar{N}\}$. Substituting into (5.24) and combining with inverse mapping (5.11) it follows that

$$\mu_{N_1} = \min\{\bar{T}x_1^*, \bar{N}\} \tag{5.25}$$

$$\mu_{N_i} = \min\{\mu_{N_1}\frac{w_1}{w_i}, \bar{N}\},\ i = 2,\ldots,n \tag{5.26}$$

$$\boldsymbol{x}^* = F^{-1}(\mu_{\boldsymbol{N}}) \tag{5.27}$$

The complete vector $\mu_{\boldsymbol{N}}$ can now be found by solving equations (5.25)-(5.27).

5.3.3 Examples

We illustrate the nature of the proportional fair solution (5.25)-(5.27) via some brief examples.

Example 1: $\bar{N} = +\infty$

In this case there is no limit to the allowed aggregation level. It follows from (5.25)-(5.27) that $\mu_{N_i} = \mu_{N_1}\frac{\mu_{R_i}}{\mu_{R_1}} = T\frac{\mu_{R_i}}{\mu_{R_1}}x_1^*$ since $\bar{N}$ does not act to constrain the aggregation level. We know from Lemma 1 that the delay constraint is tight, $\mu_{T_i}(\boldsymbol{x}^*) = \bar{T}$. That is, by (5.12), that $c + \boldsymbol{w}^T\mu_{\boldsymbol{N}} = T$. Substituting for $\mu_{\boldsymbol{N}}$ this yields $c + \sum_{i=1}^{n}\frac{l+l_{oh}}{\mu_{R_i}}T\frac{\mu_{R_i}}{\mu_{R_1}}x_1^* = c + \frac{n(l+l_{oh})T}{\mu_{R_1}}x_1^* =$

T i.e $x_1^* = \frac{(T-c)\mu_{R_1}}{nLT}$ and so

$$\mu_{N_i} = \frac{(T-c)}{n(l+l_{oh})}\mu_{R_i} \tag{5.28}$$

i.e. the aggregation levels scale proportionally to the station MCS. The corresponding rates are

$$x_i = \frac{\mu_{N_i}}{c+\boldsymbol{w}^T\mu_{\mathbf{N}}} = \frac{(T-c)}{n(l+l_{oh})T}\mu_{R_i} \tag{5.29}$$

Recall that the mean airtime taken to send a frame to station i is $\frac{c}{n} + w_i x_i = \frac{c}{n} + \frac{(T-c)}{nT}$ which is the same for all stations i.e. the proportional fair rate allocation is an equal airtime one. The overall delay is $\mu_{T_i} = c + \boldsymbol{w}^T\mu_{\mathbf{N}} = c + \sum_{i=1}^{n}\frac{l+l_{oh}}{\mu_{R_i}}\frac{(T-c)}{n(l+l_{oh})}\mu_{R_i} = T$ i.e. equal to the target delay, as already noted.

Example 2: $T = +\infty$

Suppose now that the target delay $T = +\infty$ i.e. we seek the rate allocation that maintains the aggregation level at target value $\bar{N}$ similar to the feedback controller proposed in Chapter 4. From (5.25)-(5.27) we have that the proportional fair rate allocation yields aggregation levels that satisfy $\mu_{N_i} = \mu_{N_1}\frac{\mu_{R_i}}{\mu_{R_1}}$. Recall we assume the stations are ordered such that $w_1 \geq w_2 \geq \cdots \geq w_n$, and since $w_i = (l+l_{oh})/\mu_{R_i}$ it follows that $\mu_{R_1} \leq \mu_{R_2} \leq \cdots \leq \mu_{R_n}$ i.e. station 1 has the lowest MCS rate and n the highest. Hence, the aggregation level μ_{N_n} of station n is largest. We know that the aggregation levels of all stations are no more than $\bar{N}$, and in fact this limit will be attained since this maximises throughput. Hence, $\mu_{N_n} = \bar{N}$ and so

$$\mu_{N_i} = \bar{N}\frac{\mu_{R_i}}{\mu_{R_n}} \tag{5.30}$$

i.e. once again the aggregation levels scale proportionally to the station MCS. The corresponding rates also scale proportionally to the station MCS,

$$x_i = \frac{\mu_{N_i}}{c+\boldsymbol{w}^T\mu_{\mathbf{N}}} = x_n\frac{\mu_{R_i}}{\mu_{R_n}} \tag{5.31}$$

The mean airtime taken to send a frame to station i is therefore $\frac{c}{n} + w_i x_i = \frac{c}{n} + x_n(l+l_{oh})/\mu_{R_n}$ which is the same for all stations i.e. the proportional fair rate allocation when $T = +\infty$ is again an equal airtime one.

5.4 Inner-Outer Feedback Control

While we can solve convex optimization P' using any standard online algorithm, it turns out that we can use the extra insight into the structure of the proportional fair solution gained in Section 5.3.2 to construct efficient and robust feedback-based approaches for solving online solution of P'.

In particular, from the solution structure (5.25)-(5.27) we have that the proportional fair rate allocation is also the solution to the following nested optimization,

$$\boldsymbol{N}^* \in \arg \min_{\boldsymbol{N}\in\nu\boldsymbol{W},\nu\in[1,\infty)} (\nu - \min\{\bar{T}x_1^*(\boldsymbol{N}), \bar{N}\})^2 \tag{5.32}$$

$$s.t.\ \boldsymbol{x}^* \in \arg \min_{\boldsymbol{x}\in\mathbb{R}_+^n} \sum_{i=1}^{n} (\mu_{N_i}(\boldsymbol{x}) - \min\{N_i^*, \bar{N}\})^2 \tag{5.33}$$

where $\boldsymbol{W} = [1, \frac{w_1}{w_2}, \ldots, \frac{w_1}{w_n}]^T$. As we will shortly see, it turns out that this reformulation lends itself to an elegant feedback control implementation.

5.4.1 Inner Loop Controller

We begin by considering inner optimization $\min_{\boldsymbol{x}\in\mathbb{R}_+^n} \sum_{i=1}^{n} (\mu_{N_i}(\boldsymbol{x}) - \min\{N_i, \bar{N}\})^2$. While the solution is trivial our interest is using the optimization to derive a feedback update that is robust to model uncertainty. With this in mind therefore we change variables to $\boldsymbol{z} = \mu_{N_i}(\boldsymbol{x})$. Then the optimization becomes $\min_{\boldsymbol{z}\in\mathbb{R}_+^n} \sum_{i=1}^{n} (z_i - \min\{N_i, \bar{N}\})^2$. Gradient descent now yields the following iterative update,

$$\boldsymbol{z}(k+1) = \boldsymbol{z}(k) + K_1(\boldsymbol{N}_{target} - \boldsymbol{z}(k)) \tag{5.34}$$

where the ith element of vector $\boldsymbol{N}_{target}$ equals $\min\{N_i, \bar{N}\}$, K_1 is the step size and time is slotted with $\boldsymbol{z}(k)$ denoting the value in slot k. Using (5.11) we recover the rate from $\boldsymbol{x}(k) = \boldsymbol{F}^{-1}(\boldsymbol{z}(k)) = \boldsymbol{z}(k)/(c + \boldsymbol{w}^T\boldsymbol{z}(k))$.

We convert this update to a feedback control loop by substituting the measured aggregation level for $\boldsymbol{z}(k)$ over time interval $[0, \Delta k)$, $k = 1, 2, \ldots$, to obtain

$$\boldsymbol{z}(k+1) = \boldsymbol{z}(k) + K_1(\boldsymbol{N}_{target} - \tilde{\mu}_{\boldsymbol{N}}(k)) \tag{5.35}$$

$$\boldsymbol{x}(k) = \boldsymbol{F}^{-1}(\boldsymbol{z}(k)) = \boldsymbol{z}(k)/(c + \boldsymbol{w}^T\boldsymbol{z}(k)) \tag{5.36}$$

where $\tilde{\mu}_{\boldsymbol{N}}(k)$ is the measured mean aggregation level over time slot k when the send rate is held constant at $\boldsymbol{x}(k)$ over the slot.

It can be seen that update (5.35) is an integral controller that adjusts $\boldsymbol{z}$ to try to regulate $\boldsymbol{e} = \boldsymbol{N}_{target} - \tilde{\mu}_{\boldsymbol{N}}$ about zero. Namely, when $\boldsymbol{e} > 0$ then $\boldsymbol{z}$ is increased, which in turn tends to increase $\tilde{\mu}_{\boldsymbol{N}}$ and so decrease $\boldsymbol{e}$. Conversely, when $\boldsymbol{e} < 0$ then $\boldsymbol{z}$ is decreased which tends to increase $\boldsymbol{e}$. Since $\boldsymbol{z}$ etc are vectors (5.35)-(5.36) is a multiple-input multiple-output (MIMO) feedback loop. Since $\tilde{\mu}_{\boldsymbol{N}}$ is a nonlinear function of the send rate $\boldsymbol{x}(k)$ and $\boldsymbol{x}(k)$ is a nonlinear function of $\boldsymbol{z}(k)$ the feedback loop is also nonlinear.

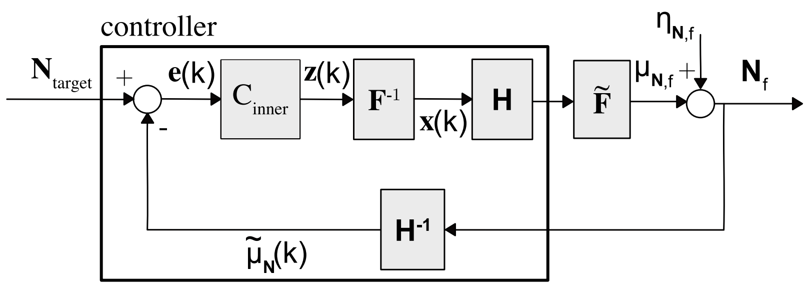


FIGURE 5.5: Schematic of inner feedback loop. Controller updates occur at the start of time slots $[0, \Delta k)$, $k = 1, 2, \ldots$. Controller update $z(k+1) = z(k) + K_1 e(k)$, nonlinear function F^{-1} is given by (5.11), H is a zero-order hold (i.e. holds the packet send rate constant at $x(k)$ for packets sent during slot $[0, \Delta k)$), μ_{N_f} is the mean frame aggregation level and η_{N_f} is the disturbance to this mean induced by MAC channel access randomness (see Section 5.2.7). H^{-1} maps from the sequence $N_f = \mu_{N_f} + \eta_{N_f}$, $f = 1, 2\ldots$ of individual frame aggregation levels to the empirical mean $\tilde{\mu}_N(k)$ of the frame aggregation level over slots $[0, \Delta k)$, $k = 1, 2, \ldots$.

5.4.1.1 Converting Between Slots and Frames

Update (5.35)-(5.36) is in term of time slots $[0, \Delta k)$, $k = 1, 2, \ldots$. To embed it within the real system we given rate $x(k)$ over slot k we fix the sender inter-packet time between packets sent during interval $[0, \Delta k)$ to be $1/x(k)$.

Conversely, given the sequence of observed individual frame aggregation levels $N_f = [N_{1,f}, \ldots, N_{n,f}]$, $f = 1, 2, \ldots$ we calculate $\tilde{\mu}_N(k)$ as the empirical mean of the frames received during interval $[0, \Delta k)$. That is,

$$\tilde{\mu}_{N_i}(k) = \frac{1}{|\Phi_i(k)|} \sum_{f \in \Phi_i(k)} N_{i,f} \tag{5.37}$$

where $\Phi_i(k)$ is the set of frames received at station i during interval $[0, \Delta k)$.

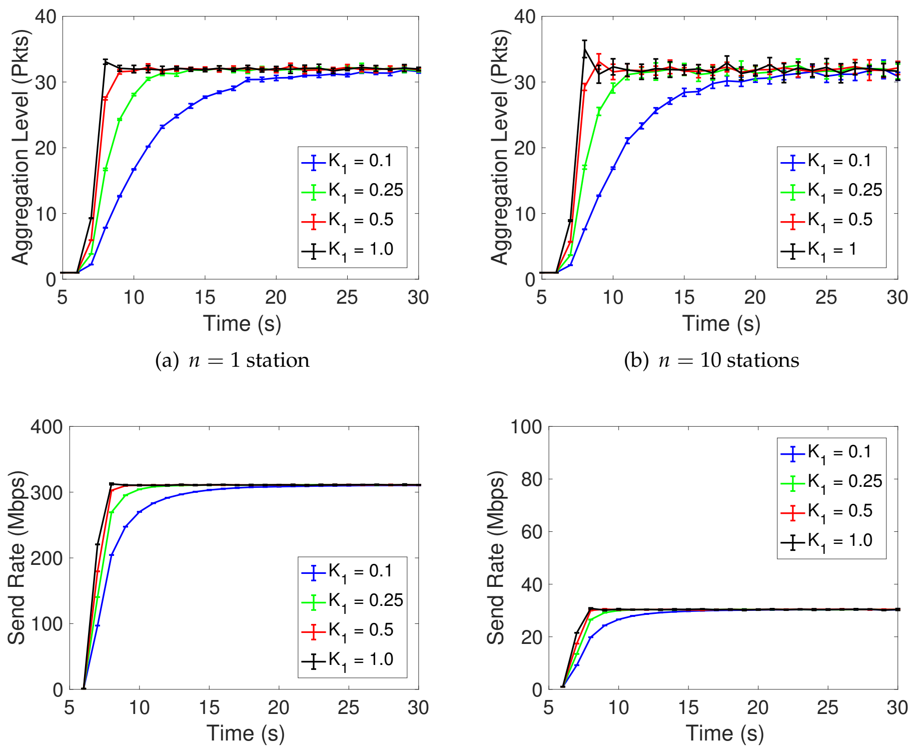
Figure 5.5 shows schematically the resulting feedback loop corresponding to (5.35)-(5.36). H holds the sender inter-packet time equal to $1/x(k)$ during controller update slot $[0, \Delta k)$. H^{-1} maps from the sequence of individual frame aggregation levels N_f to the empirical average aggregation level $\tilde{\mu}_N$ over slots $[0, \Delta k)$, $k = 1, 2, \ldots$.

Since $N_f = \mu_{N_f} + \eta_{N_f}$ the empirical mean$\tilde{\mu}_N(k)$ over slot k is $\tilde{F}_k + \eta_{\tilde{\mu}_N}(k)$ where $\tilde{F}_k = \frac{1}{|\Phi_i(k)|}\sum_{f\in\Phi_i(k)} \mu_{N_f}$ and $\eta_{\tilde{\mu}_N}(k) = \frac{1}{|\Phi_i(k)|}\sum_{f\in\Phi_i(k)} \eta_{N_f}$. That is, $\tilde{F}_k$ is the true mapping from rate to mean aggregation level at send rate $x(k)$ and $\eta_{\tilde{\mu}_N}(k)$ is the measurement noise. Due to mismatches between the model and the real system, in general $\tilde{F}_k \neq F$.

5.4.1.2 Linearizing Action of Controller

It can be seen from Figure 5.5 that to compensate for the nonlinearity $\tilde{F}_k$ we insert its (approximate) inverse F^{-1} so that $\tilde{\mu}_N(k) = \tilde{F}_k(F^{-1}(z(k)))$ and the system dynamics become

$$z(k+1) = z(k) + K_1(N_{target} - \tilde{F}_k(F^{-1}(z(k)))) \tag{5.38}$$



(a) $n = 1$ station (b) $n = 10$ stations

(c) $n = 1$ station (d) $n = 10$ stations

FIGURE 5.6: Impact of control gain K_1 on transient dynamics of aggregation level and send rate. Plots show average and standard deviation over 10 runs for each value of gain. NS3 simulation, setup as in Section 5.5: $N_{target} = 32$, NSS=1, MCS=9.

(neglecting the additive measurement noise $\eta_{\tilde{\mu}_N}(k)$ for now). When $\tilde{\boldsymbol{F}}_k(\boldsymbol{F}^{-1}(\boldsymbol{z}(k))) \approx \boldsymbol{z}_k$ then the resulting linearized loop dynamics are $\boldsymbol{z}(k+1) \approx \boldsymbol{z}(k) + K_1(\boldsymbol{N}_{target} - \boldsymbol{z}(k))$. That is, the controller transforms the nonlinear system to have first-order linear dynamics.

5.4.1.3 Robust Stability

Recall that the main source of model uncertainty is parameter c. That is, $\boldsymbol{F}(\boldsymbol{x}) = \frac{c\boldsymbol{x}}{1-\boldsymbol{w}^T\boldsymbol{x}}$ whereas to a good approximation $\tilde{\boldsymbol{F}}_k(\boldsymbol{x}) = \Pi \circ \frac{\tilde{c}(k)\boldsymbol{x}}{1-\boldsymbol{w}^T\boldsymbol{x}}$ with $\tilde{c}_k \neq c$ (recall projection Π captures the saturation constraint that $\tilde{\mu}_{\boldsymbol{N}}(k) \in [1, N_{max}]$). Hence, $\tilde{\boldsymbol{F}}_k(\boldsymbol{F}^{-1}(\boldsymbol{z}(k))) = \Pi \circ (\frac{\tilde{c}(k)}{c}\boldsymbol{z}(k))$ and dynamics (5.38) are

$$\boldsymbol{z}(k+1) = \boldsymbol{z}(k) + K_1(\boldsymbol{N}_{target} - \Gamma(k)\boldsymbol{z}(k)) \tag{5.39}$$

where $\Gamma(k) = diag\{\gamma_1(k), \ldots, \gamma_n(k)\}$ and $\gamma_i(k) = \frac{\Pi \circ (\tilde{c}(k) z_i(k)/c)}{z_i(k)}$.

Neglecting the input $\boldsymbol{N}_{target}$ for the moment, it is easy to see[6] that the dynamics $\boldsymbol{z}(k+1) = (I - \Gamma(k))\boldsymbol{z}(k)$ are exponentially stable provided $0 < \gamma_i(k) < 2$ for all $i = 1, \ldots, n$. Note that this stability holds for arbitrary time-variations in the $\gamma_i(k)$. Projection Π satisfies $0 \leq \frac{\Pi \circ z}{z} \leq 1$ and $\tilde{c}(k), c$ are both non-negative, so for stability it is sufficient that

[6]Try candidate Lyapunov function $V(k) = z^T(k)z(k)$. Then $V(k+1) = (I - \Gamma(k))^T(I - \Gamma(k))V(k)$ (since $\Gamma(k)$ is diagonal) and so is strictly decreasing when $0 < \gamma_i(k) < 2$.

$\tilde{c}(k)/c < 2$. This condition is also necessary since for constant $\tilde{c}(k)$ the system will be unstable if this condition is violated.

In summary, time-variations in the $\gamma_i(k)$ affect stability in a benign fashion and control parameter c can safely be larger than the (uncertain) plant gain $\tilde{c}(k)$ (as this reduces the loop gain) but should not be too much smaller (since this increases the loop gain).

Time-variations in the gains $\gamma_i(k)$, $i = 1, \ldots, n$ also affect regulation of the aggregation level at N_{target}. It can be seen that when $\gamma_i(k)$ is constant the equilibrium of dynamics (5.39) is $z_i(k) = N_{target}/\gamma_i(k)$. When variations in $\gamma_i(k)$ are sufficiently slow relative to the loop dynamics then $z_i(k)$ will still roughly track this equilibrium [50] although faster changes may lead to $z_i(k)$ only staying in a ball around it. Hence, when $\gamma_i(k) < 1$ the aggregation level tends to be larger than the desired value N_{target}, and vice versa when $\gamma_i(k) < 1$. Hence, adaptation of control parameter c to maintain $\gamma_i(k)$ close to 1 is desirable, and we will discuss this in more detail shortly.

5.4.1.4 Selecting Controller Gain K_1

Figure 5.6 plots the measured step response of the system aggregation level and send rate x as the gain K_1 and number of stations n are varied. This data is for a detailed packet-level simulation, see Section 5.5 for details. It can be seen from Figures 5.6(a)-(b) that, as expected, the aggregation level convergence time falls as K_1 is increased although the response starts to become oscillatory for larger values of K_1. It can also be seen from these figures that that step response is effectively invariant with the number of stations due to the linearizing action of the controller. Figures 5.6(c)-(d) show the send rate time histories corresponding to Figures 5.6(a)-(b) and the impact of the nonlinearity $\tilde{F}_k$ relating aggregation level and send rate is evident with the send rate being an order of magnitude smaller for the same aggregation level with $n = 10$ stations compared to with $n = 1$ station. Similar results are obtained when the MCS is varied.

Figure 5.7(a) plots the standard deviation of the frame aggregation level when the system is in steady-state vs the gain K_1. It can be seen that the controller starts to amplify the fluctuations in frame aggregation level as K_1 gets closer to the stability boundary at $K_1 = 2$ (indicated by the dashed line on the figure) but otherwise the standard deviation is insensitive to the choice of K_1. Recall that the fluctuations in the aggregation level are mainly induced by the randomness of the CSMA/CS channel access and occur on time-scales which are too short to be regulated by the controller, see Section 5.2.7.

In the remainder of this chapter we select $K_1 = 0.5$ unless otherwise stated since this strikes a reasonable balance between response time and robustness to uncertainty in c (with $K_1 = 0.5$ the value of c can be out by a factor of 4, corresponding to a gain margin of 12 dB, and the system dynamics will remain stable).

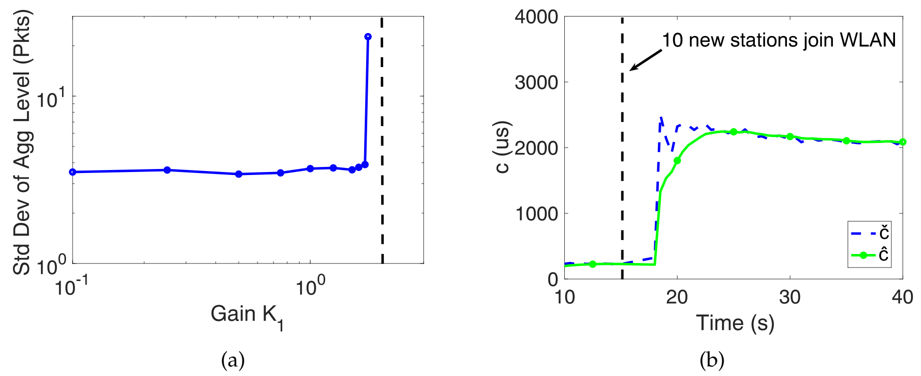


FIGURE 5.7: (a) Impact of control gain K_1 on standard deviation of fluctuations in aggregation level. (b) Illustrating c estimator (5.40) tracking a sharp change in the number of stations from $n = 1$ to $n = 11$ at time 15s. NS3 simulation, setup as in Section 5.5: one client station, $N_{target} = 32$, NSS=1, MCS=9

5.4.1.5 Adapting c

The controller depends on parameter $c = n\mu_{T_{oh}}$. The average channel access time for each frame transmission is $CW/2 \times S$ where CW is the MAC contention window, typically 16 in 802.11ac, and S is the MAC slot duration in seconds. The PHY slot length is typically $9\mu s$, but the MAC slot duration can be significantly longer when other transmitters share the channel since the AP will defer access upon detecting the channel to be busy and it is this which makes it challenging to estimate $\mu_{T_{oh}}$.

Note that an exact value for c is not necessary since the feedback loop can compensate for uncertainty in c, i.e. an estimator that roughly tracks any large changes in c is sufficient. Recall that $\mu_{N_i} = \frac{cx_i}{1-\boldsymbol{w}^T\boldsymbol{x}}$, i.e. $c = \frac{\mu_{N_i}}{x_i}(1 - \boldsymbol{w}^T\boldsymbol{x})$. Motivated by this observation we use the following as an estimator of c,

$$\hat{c}(k+1) = (1-\beta)\hat{c}(k) + \beta\check{c}(k) \tag{5.40}$$

with $\check{c}(k) := \frac{\tilde{\mu}_{N_1}(k)}{x_1(k)}(1 - \boldsymbol{w}^T\boldsymbol{x}(k))$, where β is a design parameter which controls the window over which the moving average is calculated (a typical value is $\beta = 0.05$).

Figure 5.7(b) illustrates the ability of this estimator to track a fairly significant change in the network conditions, namely 10 new stations joining the WLAN at time 15s and starting downlink transmissions. These new stations cause a change in c from a value of around $200\mu s$ to around $2200\mu s$ i.e. a change of more than an order of magnitude. It can be seen that estimator (5.40) tracks this large change without difficulty. We observe similar tracking behavior for changes in MCS and also when the channel is shared with other legacy WLANs.

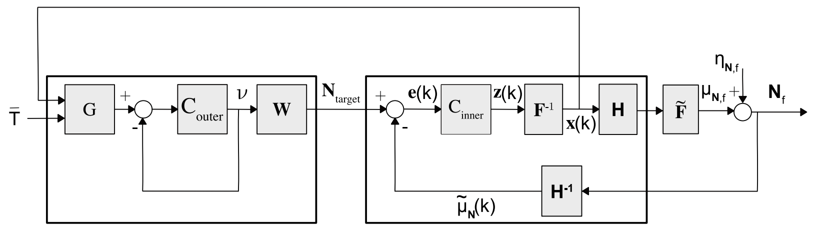


FIGURE 5.8: Schematic of coupled feedback loops. $\bar{T}$ is the target delay, $G(\boldsymbol{x}(k)) = \min\{\bar{T}x_1(k), \bar{N}\}$, C_{outer} denotes the outer control update (5.43) and $\boldsymbol{W}$ update (5.44). Other quantities are as in Figure 5.5.

5.4.2 Outer Loop Controller

We turn now to the outer optimization $\min_{\boldsymbol{N}\in\{\nu\boldsymbol{W},\nu\in[1,\infty)\}}(\nu - \min\{\bar{T}x_1^*(\boldsymbol{N}), \bar{N}\})^2$ in (5.32)-(5.33). The corresponding gradient descent update is

$$\nu(k+1) = \max\{\nu(k) - K_2(\nu(k) - G(\boldsymbol{x}^*(\boldsymbol{N}(k)))), 1\} \quad (5.41)$$
$$\boldsymbol{N}(k+1) = \nu(k+1)\boldsymbol{W} \quad (5.42)$$

where $G(\boldsymbol{x}) = \min\{\bar{T}x_1, \bar{N}\}$ and $\boldsymbol{W} = [1, \frac{w_1}{w_2}, \ldots, \frac{w_1}{w_n}]^T$. Step size K_2 and delay target T are design parameters and $x_1^*(\boldsymbol{N}(k))$ is the solution to optimization (5.33) with $\boldsymbol{N}^* = \boldsymbol{N}(k)$.

Replacing $\boldsymbol{x}^*(\boldsymbol{N}(k))$ by $\boldsymbol{x}(k) = F^{-1}(\boldsymbol{z}(k))$ from the inner loop and projecting $\nu(k+1)\boldsymbol{W}$ onto interval $[0, \bar{N}]$ so that the input to the inner loop is well-behaved, then we obtain the following coupled feedback loops,

$$\nu(k+1) = \max\{\nu(k) + K_2(G(\boldsymbol{x}(k)) - \nu(k)), 1\} \quad (5.43)$$
$$\boldsymbol{N}_{target}(k+1) = \min\{\nu(k+1)\boldsymbol{W}, \bar{N}\} \quad (5.44)$$
$$\boldsymbol{z}(k+1) = \boldsymbol{z}(k) + K_1(\boldsymbol{N}_{target}(k) - \tilde{\mu}_{\boldsymbol{N}}(k)) \quad (5.45)$$
$$\boldsymbol{x}(k+1) = \boldsymbol{F}^{-1}(\boldsymbol{z}(k+1)) \quad (5.46)$$

This setup is shown schematically in Figure 5.8. It can be seen that we "bootstrap" from the inner loop and use $G(\boldsymbol{x}(k))$ as the set point for outer loop control variable $\nu(k)$. We then map from $\nu(k)$ to the target aggregation level N_{target} using $\nu(k)\boldsymbol{W}$. Since the first element W_1 of vector $\boldsymbol{W}$ equals 1 we can identify $\nu(k)$ with the target aggregation level for station 1 i.e. the station with lowest MCS rate, and the target aggregation levels of the other stations are proportional to $\nu(k)$.

5.4.2.1 Sufficient Conditions For Stability

Substituting from (5.39) the system dynamics (5.43)-(5.46) can be rewritten equivalently as,

$$\nu(k+1) = \max\{\nu(k) - K_2(\nu(k) - G(\boldsymbol{F}^{-1}(\boldsymbol{z}(k)))), 1\} \quad (5.47)$$
$$\boldsymbol{z}(k+1) = \boldsymbol{z}(k) + K_1(\min\{\nu(k)\boldsymbol{W}, \bar{N}\} - \boldsymbol{\Gamma}_k\boldsymbol{z}(k)) \quad (5.48)$$

Assume the dynamics of the inner z loop are much faster than those of the outer ν loop (e.g. by selecting $K_2 \ll K_1$) so that $\boldsymbol{z}(k) = \nu(k)\boldsymbol{W}$. Then the system dynamics simplify to

$$\begin{aligned}
&\nu(k+1) \\
&= \max\{\nu(k) - K_2(\nu(k) - \min\{\bar{T}\frac{\nu(k)}{c+\nu(k)nw_1}, \bar{N}\}), 1\} && (5.49)\\
&= \max\{\nu(k) - K_2(\nu(k) - \gamma_0(k)\bar{T}\frac{\nu(k)}{c+\nu(k)nw_1}), 1\} && (5.50)\\
&= \max\{(1 - K_2\frac{c+\nu(k)nw_1 - \gamma_0(k)\bar{T}}{c+\nu(k)nw_1})\nu(k), 1\} && (5.51)
\end{aligned}$$

where $0 < \gamma_0(k) \leq 1$ captures the impact of the $\bar{N}$ constraint i.e $\gamma_0(k)$ equals 1 when $\bar{T}\frac{\nu(k)}{c+\nu(k)nw_1} \leq \bar{N}$ and decreases as $\bar{T}\frac{\nu(k)}{c+\nu(k)nw_1}$ increases above $\bar{N}$. We have also used the fact that $\boldsymbol{w}^T\boldsymbol{W} = nw_1$.

We can gain useful insight into the behaviour of the system dynamics from inspection of (5.51). Namely, ignoring the constraints for the moment (i.e. $\gamma_0(k) = 1$ and $\nu(k) \geq 1$) and assuming that $0 < K_2 < 1$ then it can be seen that when $c + \nu(k)nw_1 - \bar{T} < 0$ then $1 - K_2\frac{c+\nu(k)nw_1-\bar{T}}{c+\nu(k)nw_1}) > 1$ and so $\nu(k+1)$ increases (since $\nu(k) \geq 1$). Hence $c + \nu(k)nw_1 - \bar{T}$ increases until it equals 0 or becomes positive. Conversely, when $c + \nu(k)nw_1 - \bar{T} > 0$ then $1 - K_2\frac{c+\nu(k)nw_1-\bar{T}}{c+\nu(k)nw_1}) < 1$ and $\nu(k+1)$ decreases. Hence, $c + \nu(k)nw_1 - \bar{T}$ decreases until it equals 0 to becomes negative. That is, the dynamics (5.51) force $c + \nu(k)nw_1 - \bar{T}$ to either converge to 0 or oscillate about 0.

With the above in mind the impact of the constraints is now easy to see. When $\bar{T} > c + \bar{N}nw_1$ then the delay target is hit at an aggregation level above $\bar{N}$. It can be seen that $c + \nu(k)nw_1 - \bar{T} < 0$ for all admissible $\nu(k)$ and so $\nu(k)$ increases until it equals $\bar{N}$. When $\bar{T} < c + nw_1$ then the target delay is violated even when the aggregation level is the minimum possible $\nu(k) = 1$. It can be seen that $c + \nu(k)nw_1 - \bar{T} > 0$ for all admissible $\nu(k)$ and so $\nu(k)$ decreases until it equals 1.

To establish stability we need to show that persistent oscillations about $c + \nu(k)nw_1 - \bar{T} = 0$ cannot happen. We have the following lemma:

Lemma 2. *Suppose gain* $0 < K_2 < 1$ *and initial condition* $1 \leq \nu(1) \leq \bar{N}$. *Then for the dynamics (5.51) we have: (i) when* $c + nw_1 < \bar{T} < c + \bar{N}nw_1$ *then* $\nu(k)$ *converges to* $(\bar{T} - c)/(nw_1)$, *(ii) when* $\bar{T} \geq c + \bar{N}nw_1$ *then* $\nu(k)$ *converges to upper limit* $\bar{N}$ *and (iii) when* $\bar{T} < c + nw_1$ *then* $\nu(k)$ *converges to lower limit* 1.

Proof. Case(i): $c + nw_1 < \bar{T} < c + \bar{N}nw_1$. Try candidate Lyapunov function $V(k) = (c + \nu(k)nw_1 - \bar{T})^2/(nw_1)^2$. Letting $\nu^* = (\bar{T} - c)/nw_1$ then this can be rewritten as $V(k) = (\nu(k) - \nu^*)^2$ and since $c + nw_1 < \bar{T} < c + \bar{N}nw_1$ then $1 < \nu^* < \bar{N}$. In addition, to take care of gain $\gamma_0(k)$ we will show by induction that $\gamma_0(k) = 1$. By assumption $1 <$

$\bar{T}\nu(1)/(c+\nu(1)nw_1) \le \bar{N}$ and so $\gamma_0(1) = 1$. Suppose $\gamma_0(k) = 1$. Substituting from (5.51) it follows that

$$\begin{aligned}
V(k+1) &= (\nu(k+1) - \nu^*)^2 \\
&= (\max\{(1 - K_2\frac{c+\nu(k)nw_1 - \bar{T}}{c+\nu(k)nw_1})\nu(k), 1\} - \nu^*)^2 \\
&\overset{(a)}{\le} ((1 - K_2\frac{c+\nu(k)nw_1 - \bar{T}}{c+\nu(k)nw_1})\nu(k) - \nu^*)^2 \\
&= ((1 - K_2\frac{(c-\bar{T})/nw_1 + \nu(k)}{c+\nu(k)nw_1}nw_1)\nu(k) - \nu^*)^2 \\
&= (\nu(k) - K_2\frac{\nu(k) - \nu^*}{c+\nu(k)nw_1}nw_1\nu(k) - \nu^*)^2 \\
&\overset{(b)}{=} (1 - K_2\frac{\nu(k)nw_1}{c+\nu(k)nw_1})^2 V(k)
\end{aligned}$$

where (a) follows because $\nu^* > 1$. Since $0 < K_2 < 2$ and $0 < \frac{\nu(k)nw_1}{c+\nu(k)nw_1} < 1$ it follows from (b) that $0 < (1 - K_2\frac{\nu(k)nw_1}{c+\nu(k)nw_1})^2 < 1$ and so $V(k+1)$ is strictly decreasing unless $V(k) = 0$. Further, since $K_2 < 1$ then $\nu(k+1)$ has the same sign as $\nu(k)$ i.e. $\nu(k+1) > 0$. Putting these observations together, we have that $\nu(k+1)$ is closer than $\nu(k)$ to $\nu^* < \bar{N}$. Since $\nu(1) \le \bar{N}$ then $\nu(2) < \bar{N}$, while when $\nu(k) \le \bar{N}$ for $k > 1$ then $\nu(k+1) < \bar{N}$. So by induction $\nu(k) \le \bar{N}$ for all $k \ge 1$ and thus $\gamma_0(k) = 1$ for all $k \ge 1$. Since $V(k+1) < V(k)$ when $V(k) > 0$ then $V(k)$ decreases monotonically to 0 i.e. the system converges to the point $c + \nu(k)nw_1 - \bar{T} = 0$ as claimed.

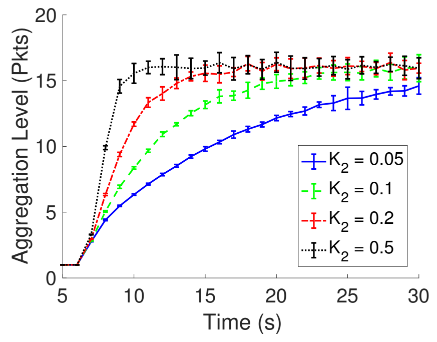
Cases (ii) and (iii). When $\bar{T} \ge c + \bar{N}nw_1$, respectively $\bar{T} < c + nw_1$, then $c + \nu(k)nw_1 - \bar{T} < 0$, respectively $c + \nu(k)nw_1 - \bar{T} > 0$ for all $1 \le \nu(k) \le \bar{N}$. The stated result now follows. □

Note that while the above analysis makes use of time-scale separation between z and ν so that $z(k) = \nu(k)\boldsymbol{W}$, in practice we observe that the system is well behaved even when this assumption is violated and conjecture that Lemma 2 also applies in such cases.

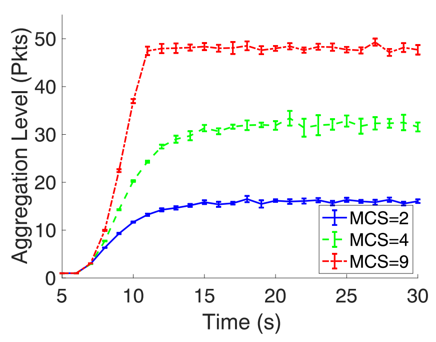
5.4.2.2 Selecting Control Gain K_2

Figure 5.9(a) plots the measured step response of the system aggregation level as the outer control gain K_2 is varied. It can be seen that the rise time falls with increasing gain, as expected. Although not shown on the plot to reduce clutter, we observe that for $K_2 \ge 1$ the response becomes increasing oscillatory suggesting that the sufficient condition for stability $K_2 < 1$ is in fact the stability boundary. In the rest of the chapter we select $K_2 = 0.2$ as striking a reasonable compromise between responsiveness and robustness to uncertainty.

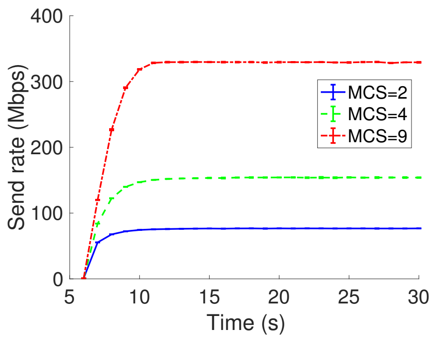
Figures 5.9(b)-(d) illustrate the adaptation by the outer feedback loop of N_{target} so as to regulate the delay about the target value $\bar{T}$. Figure 5.9(b) plots the aggregation level vs time, Figure 5.9(c) the send rate and Figure 5.9(d) the delay. Measurements are shown for three MCS



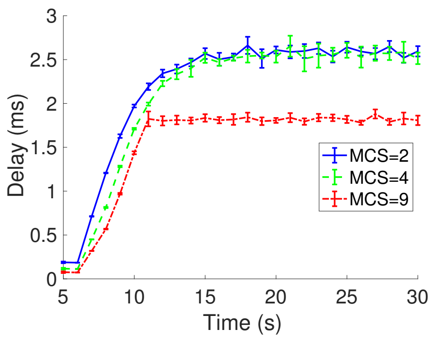
(a) MCS 2, $\bar{T} = 2.5ms$



(b) $K_2 = 0.2$, $\bar{T} = 2.5ms$



(c) $K_2 = 0.2$, $\bar{T} = 2.5ms$



(d) $K_2 = 0.2$, $\bar{T} = 2.5ms$

FIGURE 5.9: (a) Impact of outer loop gain K_2 on convergence time, (b) adapting N_{target} to regulate delay to below $\bar{T}$ as MCS is varied, (c), (d) send rate and delay measurements corresponding to (b). Plots show average and standard deviation over 10 runs for each value of gain. NS3, one client station, NSS=1, $\bar{T} = 2.5ms$, $\bar{N} = 48$.

values. It can be seen that as the MCS rate increases both the aggregation level and send rate increase while the delay is maintained close to the target value $\bar{T} = 2.5ms$. This is as expected since the mean delay is just the mean duration of a scheduling round $c + \boldsymbol{w}^T\mu_{\boldsymbol{N}}$. As the MCS rate increases $\boldsymbol{w}$ decreases and so the aggregation level $\mu_{\boldsymbol{N}}$ can increase while keeping product $\boldsymbol{w}^T\mu_{\boldsymbol{N}}$ (which is the overall time to transmit the frame payloads) unchanged.

We can quickly verify the measurements as follows. For the network configuration in Figures 5.9(b)-(d) fixed overhead c is around 200μs. MCS index 2 with NSS=1 corresponds data rate 87.7Mbps, the packet size $l = 1500$B, overhead $l_{oh} = 48$B and from Figure 5.9(b) the aggregation level is approximately 16 packets, so $\boldsymbol{w}^T\mu_{\boldsymbol{N}} = (1500 + 48) \times 8 \times 16/87.7 \times 10^6 = 2.3$ms and adding c to this gives $\bar{T} = 2.5$ms. Similarly, MCS index 4 with NSS=1 corresponds to a data rate of 175.5Mbps and plugging this value into the previous expression along with aggregation level 23 packets again gives $\boldsymbol{w}^T\mu_{\boldsymbol{N}} = 2.3$ms. MCS index 9 with NSS=1 corresponds to data rate 390Mbps. At this data rate we hit the limit $\bar{N} = 48$ packets before delay target $\bar{T}$ is reached ($\boldsymbol{w}^T\mu_{\boldsymbol{N}} = 1.5$ms when the rate is 390Mbps and the aggregation level is 48 packets, adding $c = 200\mu$s to this gives a delay of 1.7ms as can be seen in Figure 5.9(d)).

5.5 Experimental Measurements

In this section, we implement the inner-outer feedback controller described in Algorithm 2 on Linux and Android and assess performance in our testbed in two scenarios: single client and multiple clients.

5.5.1 Prototype Implementation

5.5.1.1 Implementation on Linux

To implement the inner-outer controller on linux, we wrote the server and client codes in C and then compiled it using `gcc` Linux utility. Here, we need root privilege to run the client code and capture MAC timestamps.

5.5.1.2 Implementation on Android

To extend the code to run on the Android OS, we use Android Studio [51] developed by Google to cross-compile the client code (written in C language) and build an application to install on Android devices. On Android we cannot directly read MCS values in C but there is a method in Java, i.e. the `getLinkSpeed`, that has access to this information without needing root privilege. To access the `getLinkSpeed` method from native code written in C, we use the Java Native Interface (JNI) which is an interface programming framework that enables Java code running in a Java Virtual Machine, to interact with the native code [52]. Applying this procedure, we read MCS values every 100ms to reduce the CPU load. In addition, we need to add the `ACCESS_WIFI_STATE` permission in the application. Note that in [53], some privacy issues associated with this

Algorithm 2 Nonlinear feedback loop adjusting transmit rate to regulate aggregation level.

```
$k = 1$
while 1 do
  We assume that station 1 has the lowest MCS rate and n the highest.
  for $i \leftarrow 1$ to $n$ do
    inner loop:
    $z_i \leftarrow z_i + K_1(N_{target,i} - \tilde{\mu}_{N_i})$, check $z_i \in [1, \bar{N}]$
    $\hat{c} \leftarrow (1-\beta)\hat{c} + \beta\frac{\tilde{\mu}_{N_1}}{x_1}(1 - \boldsymbol{w}^T\boldsymbol{x})$
    $x_i \leftarrow F^{-1}(z_i) = \frac{z_i}{\hat{c}+\boldsymbol{w}^T\boldsymbol{z}}$, check $x_i \in [\frac{1}{\hat{c}+\boldsymbol{w}^T\boldsymbol{z}}, \frac{\bar{N}}{\hat{c}+\boldsymbol{w}^T\boldsymbol{z}}]$
    outer loop:
    $\nu \leftarrow \max\{\nu + K_2(\min\{\bar{T}x_1, \bar{N}\} - \nu), 1\}$
    $N_{target,i} \leftarrow \nu \times \frac{w_1}{w_i}$, check $N_{target,i} \in [1, \bar{N}]$
  end for
  $k \leftarrow k + 1$
end while
```

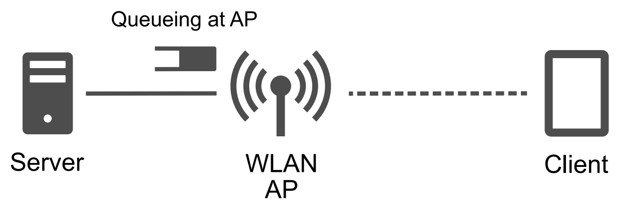


FIGURE 5.10: Schematic of a cloudlet-based edge transport architecture used for experimental setup with single client.

permission have been discussed, such as the possibility of detecting the location of an end user by applications that do not have permission to access to the GPS information.

5.5.2 Evaluation with Single Client

In Chapter 4, we show that the simple linear feedback loop outperforms Cubic and BBR in our testbed with an Android client, e.g. the aggregation-based approach with $N_\epsilon = 60$ decreases the delay more than 20 times compared to Cubic and BBR while its send rate is between those algorithms. Here, we evaluate performance of the nonlinear feedback controller, Algorithm 2, and compare it with the linear feedback controller, Algorithm 1, in our testbed depicted in Figure 5.10.

As we can see in Figure 5.11(a), the nonlinear controller has slightly higher throughput, is more stable and converges faster than the linear controller. The MCS is constant (866.7Mbps) in both cases during the 200sec of data transmission, see Figure 5.11(b). In Figure 5.11(c), the packet one-way delay for both controllers is depicted. It can be seen that the delay is extremely low ($\approx$ 1ms) while the receive rate is around 400Mbps. As shown in Figure 5.11(d), the packet loss of the nonlinear controller is almost half of the packet loss of the linear controller as the nonlinear controller can efficiently adjust the send rate based on the queue backlog inside the AP, channel conditions etc. However, the packet loss of the linear controller is also very low around 0.15%. In Figure 5.11(e), the performance of the feedback loop controllers to tune

TABLE 5.1: Summary of results with multiple clients, ($\mu \pm \sigma$)

Case	Rx Rate (Mbps)	Delay (ms)	MCS (Mbps)	Aggregation
PC_1	172 ± 25	4.3 ± 4.9	960 ± 35	29 ± 9
PC_2	193 ± 30	4.4 ± 6.2	1247 ± 93	18 ± 5
Tablet	168 ± 26	4.0 ± 3.9	866 ± 0	30 ± 9

the aggregation level is shown. We can see that both controllers have low fluctuations around the target aggregation level, but the nonlinear controller converges faster compared to the linear controller. The estimation of the average channel access time in a round-robin scheduler, $\bar{c}$, is illustrated in Figure 5.11(f).

5.5.3 Evaluation with Multiple Clients

We evaluate the performance of the inner-outer controller in an edge architecture shown in Figure 5.12. This setup includes two desktop PCs using MAC timestamps to capture the aggregation level and one Android tablet using kernel timestamps to infer the aggregation level using the logistic regression model described in Chapter 4. The setup is described in more details in Section 5.5.4.

As we can see in Figure 5.13(a), PC_2 has slightly higher throughput compared to PC_1 and the tablet. Also according to Figure 5.13(b), PC_2 has higher MCS and because of that the nonlinear controller (precisely the outer loop) selects a lower target aggregation level on average for this user ($\approx$ 20) while the average aggregation level for PC_1 and the tablet is around $\bar{N} = 30$, see Figure 5.13(e). In Figure 5.13(c), the one-way delay for the clients is depicted which is usually below $\bar{T} = 10$ms. As illustrated in Figure 5.13(d), the packet loss is quite low for all users but PC_1 has relatively higher losses due to more queue build-up at the AP, see spikes in one-way delay or in aggregation level. The estimation of the average channel access time in a round-robin scheduler, $\bar{c}$, is illustrated in Figure 5.13(f). The average results are also summarized in Table 5.1.

5.5.4 Experimental Testbed

Our setup uses an Asus RT-AC86U Access Point (which uses a Broadcom 4366E chipset and supports 802.11ac MIMO with up to four spatial streams [49]). It is configured to use the 5GHz frequency band with 80MHz channel bandwidth. Also, the firmware version of the Asus Access Point is 3.0.0.4.384_32799. A Linux server running Ubuntu Bionic 18.04.2 LTS with Linux kernel 4.18.0-25 is connected to this AP via a gigabit Ethernet link. The client in Figure 5.10 is a non-rooted Samsung Galaxy Tab S3 running Android Oreo 8. Due to the lack of root privilege, this client is restricted to using kernel timestamps to estimate the aggregation level of the received frames using the logistic regression model described in Chapter 4. In Figure 5.12, PC_1 and PC_2 are Linux boxes running Ubuntu Bionic 18.04.3 LTS with Linux kernel 4.15.0-55 and equipped with Broadcom BCM4360 802.11ac NICs to capture MAC timestamps.

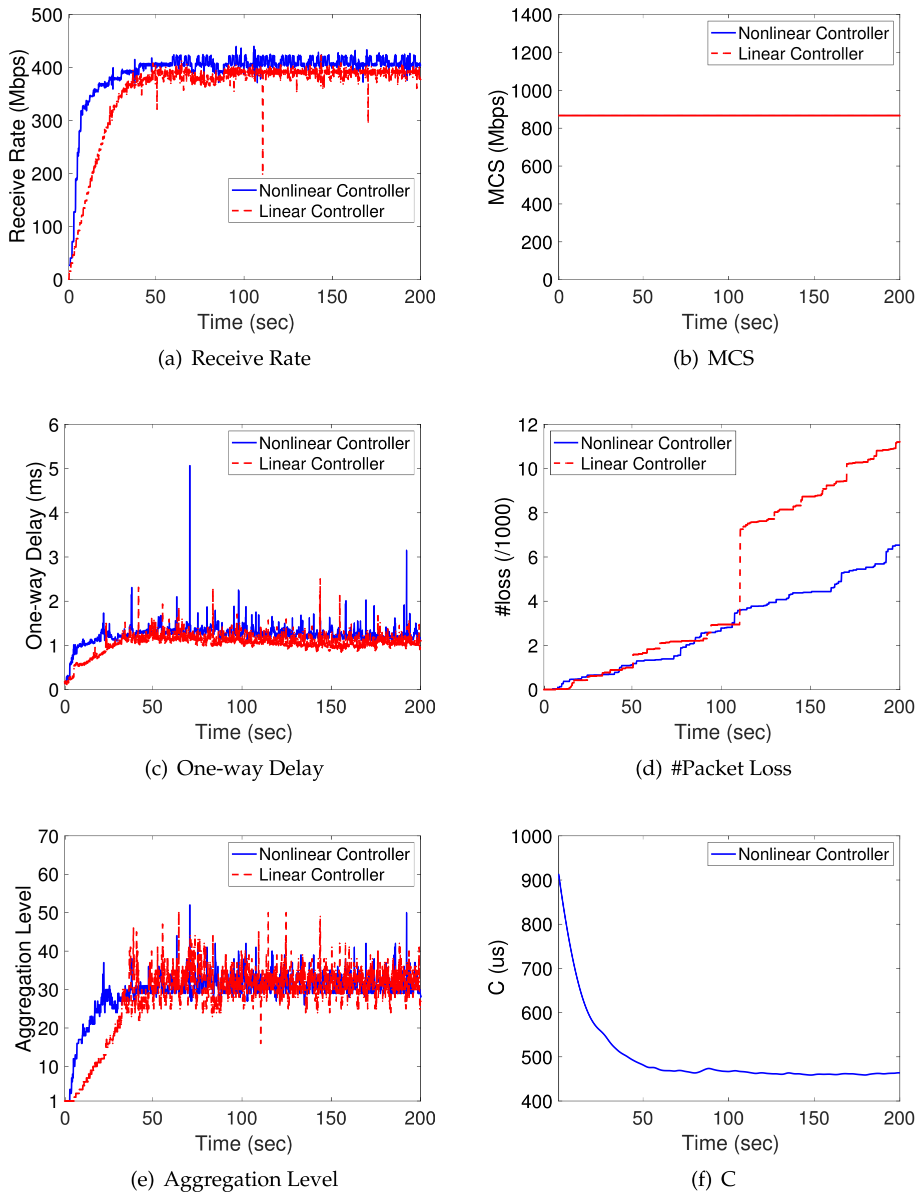


FIGURE 5.11: Compare the performance of nonlinear controller with simple linear controller. The one-way delay in (c) is averaged over 100ms intervals. For Linear Controller: $K = 0.5, N_\epsilon = 32, \Delta = 1000ms$ and for nonlinear controller: $K_1 = 0.5, K_2 = 0.2, \beta = 0.05, \bar{T} = 5.0ms, \bar{N} = 32, \hat{c}_{int} = 200\mu s, \Delta = 1000ms$. Experimental data, Samsung Galaxy handset, setup in 5.5.4, ($N_{max} = 64$).

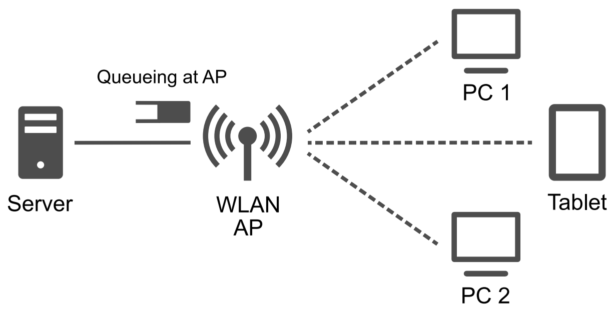
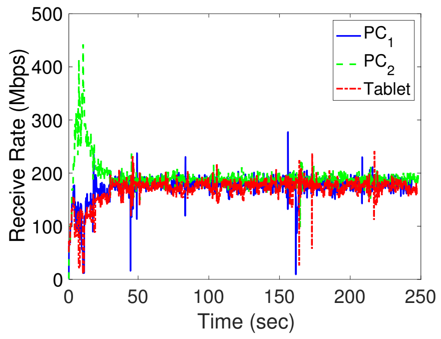
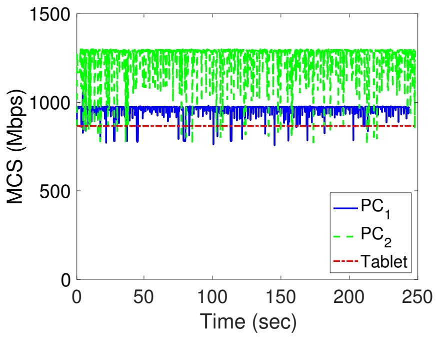


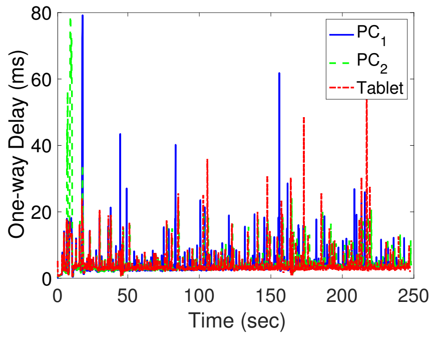
FIGURE 5.12: Schematic of a cloudlet-based edge transport architecture used for experimental setup with multiple clients.



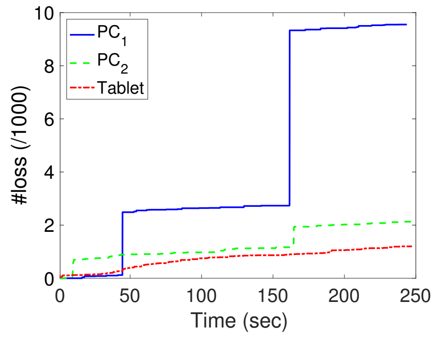
(a) Receive Rate



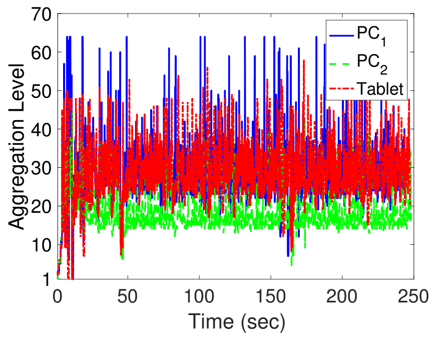
(b) MCS



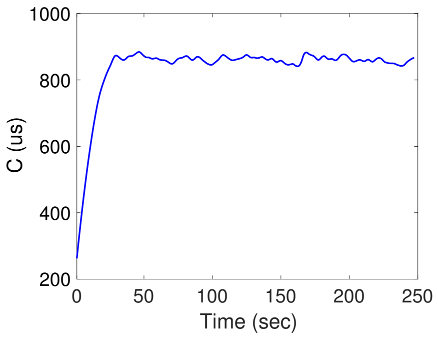
(c) One-way Delay



(d) #Packet Loss



(e) Aggregation Level



(f) C

FIGURE 5.13: Managing an edge network using the nonlinear feedback controller. The one-way delay in (c) and MCS values of tablet in (b) are averaged over 100ms intervals. The nonlinear controller is configured as follows: $K_1 = 0.5, K_2 = 0.2, \beta = 0.05, \bar{T} = 10.0ms, \bar{N} = 30, \hat{c}_{int} = 200\mu s, \Delta = 1000ms$. Experimental data, PC$_1$ and PC$_2$ use MAC timestamps and Samsung tablet uses kernel timestamps, setup in 5.5.4, ($N_{max} = 64$).

5.6 Summary & Conclusions

In this chapter we consider transport layer approach for achieving low delay and high rate in 802.11ac edge networks where the bottleneck lies in the last wireless hop. We first derive and validate models of the aggregation level and delay behavior in the downlink at the AP and show the mean channel access time is the main source of uncertainty of the models. We then propose a proportional fair low delay rate allocation algorithm as the online solution of an optimization problem. Using nested optimization techniques, we construct a nonlinear feedback controller which includes inner and outer closed loop controllers in a cascade structure. We then analyze the stability conditions of the proposed controllers. The inner feedback loop is responsible to tune the send rates for given target aggregation levels, specific for each client, and outer loop adjusts the target aggregation levels to achieve the target delay. We present a prototype transport layer implementation of our low latency rate allocation approach and use this to evaluate performance under real radio channel conditions.

Chapter 6

Conclusion

6.1 Summary & Conclusions

In this thesis, we design a transport layer approach for achieving low delay, high rate communications over edge networks where an 802.11ac wireless hop is the bottleneck. We begin by analyzing latency in 802.11ac edge networks and demonstrate that aggregation is coupled to delay in modern 802.11ac networks such that we can control the delay through regulating aggregation level. Then we implement a simple closed-loop controller on Linux (using MAC timestamps) to successfully adjust the transmit rate to tune the aggregation level. To implement this algorithm on non-rooted Android handsets, we apply logistic regression and support vector machine models to precisely infer aggregation level from kernel timestamps over a wide range of send rates. Later we implement the feedback controller on an Android tablet and compare its performance to TCP Cubic and Google BBR. We also propose a passive technique based on a logistic regression model to detect the location of the path bottleneck using aggregation level then implement it in our testbed and evaluate its performance in various scenarios.

In addition, we build models of the aggregation and delay behavior in 802.11ac access points with a round-robin network scheduler. We also propose a proportional fair rate allocation algorithm to achieve low delay, high rate in 802.11ac edge networks using aggregation where the bottleneck lies in an 802.11ac WLAN. Then we construct a nonlinear inner-outer feedback controller to solve the online solution of this approach. We implement the nonlinear controller on Linux and Android and show that that the closed-loop controller converges faster than the simple feedback loop and show a reliable performance in the presence of channel fluctuations.

6.2 Future Work

The work in this thesis highlights a number of areas for future work as follows:

- The logistic regression model introduced in Chapter 4 is trained using the data collected from a Samsung Galaxy Tab S3 tablet with Android Oreo 8. Assessing the performance of the model on a Google Pixel 2 handset with both Android 8 and 9 and also on a Samsung Galaxy S9 mobile phone with Android 9, reveals that the accuracy of the model depends on the Android version and the hardware configuration of the handset. Hence, the parameters of the model require to be tuned based on the handset.

- The work described throughout this thesis is based on use of pacing at the sender because sending packets in a burst can cause queue build-up at the AP. However, controlling the burstiness could be a solution to overcome the aforementioned problem with the proposed machine learning approach. For example, when we send multiple packets, i.e. equal to the desired aggregation level, back-to-back from the server to the AP in one go, it is more probable that the AP aggregates all of these packets together and sends one large frame instead. To evaluate this idea, we used the `sendmmsg` Linux utility to transfer a burst of UDP packets, e.g. $N_\epsilon = 32$, to the AP via a gigabit Ethernet link and measured the aggregation level at the client side using MAC timestamps. We surprisingly find that most of the time the AP sends a few frames with low aggregation levels, e.g. one or two packets, followed by frames with high aggregation levels (< 32). Therefore, sending in a burst without low aggregation levels may require designing a new scheduler at the AP.
- In Chapter 5, we assume the server (or cloudlet) is connected to just one 802.11ac WLAN. However, in enterprise settings or the like, it is common that multiple APs cover an area with multiple clients. Then we can consider a setup where users are under the control of a central node or proxy located at the edge of network. Hence, we can design a novel user/rate allocation algorithm on the proxy to increase the network QoS, e.g. increase the throughput and decrease the delay. In this scenario, we can also use the historical traffic data to predict the network and user behavior, the geographical positions of clients in the future etc to smartly map users to APs in order to improve the network performance. We can also optimize the main parameters of the nonlinear feedback controller such as the target aggregation level ($\bar{N}$) and the target delay ($\bar{T}$) using the information obtained from the prediction.
- In this thesis, we conduct a study on the benefits of aggregation to achieve low delay and high rate communications over 802.11ac edge networks. We also briefly investigated the presence of aggregation in cellular networks, i.e. LTE network, by taking measurements using iperf 3.6 but we did not observe a strong look between aggregation and queueing in LTE. However, it is probable that this will be added to 5G networks to reduce the overheads and improve the bandwidth efficiency. However, we did not have access to any 5G equipment during this study.

Bibliography

[1] *Next Generation Protocols – Market Drivers and Key Scenarios*. European Telecommunications Standards Institute (ETSI), 2016.

[2] J. Iyengar and I. Swett. "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2". In: *IETF Internet Draft* (2015).

[3] MinJi Kim et al. "Congestion control for coded transport layers". In: *Proc. IEEE International Conference on Communications (ICC)*. 2014.

[4] *Open Fast Path*. 2016. URL: http://www.openfastpath.org/.

[5] *5G White Paper*. Next Generation Mobile Networks (NGMN) Alliance, 2015. URL: https://www.ngmn.org/uploads/media/NGMN_5G_White_Paper_V1_0.pdf.

[6] Tobias Flach et al. "Reducing web latency: the virtue of gentle aggression". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 159–170.

[7] Wenxuan Zhou et al. "ASAP: A low-latency transport layer". In: *Proc. of the Seventh Conference on emerging Networking Experiments and Technologies*. ACM. 2011, p. 20.

[8] S. Souders. "Velocity and the bottom line". In: *Velocity (Web Performance and Operations Conference)*. 2009.

[9] *Qualcomm Snapdragon 820*. URL: https://www.qualcomm.com/products/snapdragon-820-mobile-platform.

[10] Y. Mao et al. "A Survey on Mobile Edge Computing: The Communication Perspective". In: *IEEE Communications Surveys Tutorials* 19.4 (2017), pp. 2322–2358. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2745201.

[11] Mahadev Satyanarayanan. "The Emergence of Edge Computing". In: *Computer* 50.1 (Jan. 2017), pp. 30–39. ISSN: 0018-9162. DOI: 10.1109/MC.2017.9. URL: https://doi.org/10.1109/MC.2017.9.

[12] Yunhong Gu and Robert L. Grossman. "UDT: UDP-based Data Transfer for High-speed Wide Area Networks". In: *Computer Networks* 51.7 (2007), pp. 1777–1799. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2006.11.009.

[13] Mohammad Karzand et al. "Design of FEC for Low Delay in 5G". In: *IEEE Journal Selected Areas in Communications (JSAC)* 35.8 (2016), pp. 1783–1793.

[14] Andres Garcia-Saavedra, Mohammad Karzand, and Douglas J. Leith. "Low Delay Random Linear Coding and Scheduling Over Multiple Interfaces". In: *IEEE Transactions on Mobile Computing* 16.11 (2017), pp. 3100–3114.

[15] J. Border et al. *Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations*. RFC 3135. RFC Editor, 2001.

[16] R. Karmakar, S. Chattopadhyay, and S. Chakraborty. "Impact of IEEE 802.11n/ac PHY/MAC High Throughput Enhancements on Transport and Application Protocols: A Survey". In: *IEEE Communications Surveys Tutorials* 19.4 (2017), pp. 2050–2091. DOI: 10.1109/COMST.2017.2745052.

[17] T. Li et al. "Aggregation with Fragment Retransmission for Very High-Speed WLANs". In: *IEEE/ACM Transactions on Networking* 17.2 (2009), pp. 591–604.

[18] S. Kuppa and G.R. Dattatreya. "Modeling and Analysis of Frame Aggregation in Unsaturated WLANs with Finite Buffer Stations". In: *IEEE International Conference on Communications (ICC)*. 2006, pp. 967–972.

[19] Boris Bellalta and Miquel Oliver. "A Space-time Batch-service Queueing Model for Multi-user MIMO Communication Systems". In: *Proc. 12th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. MSWiM 09. Tenerife, Canary Islands, Spain: ACM, 2009, pp. 357–364. ISBN: 978-1-60558-616-8. DOI: 10.1145/1641804.1641866. URL: http://doi.acm.org/10.1145/1641804.1641866.

[20] B.S. Kim, H.Y. Hwang, and D.K. Sung. "Effect of Frame Aggregation on the Throughput Performance of IEEE 802.11n". In: *IEEE Wireless Communications and Networking Conference (WCNC)*. 2008, pp. 1740–1744.

[21] David Malone, Ken Duffy, and Doug Leith. "Modeling the 802.11 Distributed Coordination Function in Nonsaturated Heterogeneous Conditions". In: *IEEE/ACM Transactions on Networking* 15.1 (2007), pp. 159–172. ISSN: 1063-6692. DOI: 10.1109/TNET.2006.890136.

[22] Abhinav Pathak et al. "A Measurement Study of Internet Delay Asymmetry". In: *Proc. 9th International Conference on Passive and Active Network Measurement*. PAM'08. Cleveland, OH, USA: Springer-Verlag, 2008, pp. 182–191. ISBN: 3-540-79231-7, 978-3-540-79231-4.

[23] David Malone, Douglas J. Leith, and Ian Dangerfield. "Inferring Queue State by Measuring Delay in a WiFi Network". In: *Traffic Monitoring and Analysis*. Ed. by Maria Papadopouli, Philippe Owezarski, and Aiko Pras. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 8–16. ISBN: 978-3-642-01645-5.

[24] Rajdeep Das et al. "Informed Bandwidth Adaptation in Wi-Fi Networks Using Ping-Pair". In: *Proc. 17th ACM Conference on Emerging Networking Experiments and Technologies (CONEXT)*. 2017, pp. 376–388.

[25] *Network Congestion Control: Managing Internet Traffic*. John Wiley & Sons, 2005.

[26] Van Jacobson. "Congestion Avoidance and Control". In: *Symposium Proceedings on Communications Architectures and Protocols*. SIGCOMM '88. Stanford, California, USA: ACM, 1988, pp. 314–329. ISBN: 0-89791-279-9. DOI: 10.1145/52324.52356. URL: http://doi.acm.org/10.1145/52324.52356.

[27] T. Henderson et al. *The NewReno Modification to TCP's Fast Recovery Algorithm*. RFC 6582. RFC Editor, 2012.

[28] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. 5th. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010. ISBN: 0132126958, 9780132126953.

[29] Sangtae Ha, Injong Rhee, and Lisong Xu. "CUBIC: A New TCP-friendly High-speed TCP Variant". In: *SIGOPS Operating Systems Review* 42.5 (2008), pp. 64–74. ISSN: 0163-5980. DOI: 10.1145/1400097.1400105.

[30] Douglas Leith and Robert Shorten. "H-TCP Protocol for High-Speed Long-Distance Networks". In: *Proc. 2nd Workshop on Protocols for Fast Long Distance Networks*. Argonne, USA, 2004.

[31] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. "TCP Vegas: New Techniques for Congestion Detection and Avoidance". In: *Proc. Conference on Communications Architectures, Protocols and Applications*. SIGCOMM '94. London, United Kingdom: ACM, 1994, pp. 24–35. ISBN: 0-89791-682-4. DOI: 10.1145/190314.190317.

[32] Cheng Peng Fu and S. C. Liew. "TCP Veno: TCP enhancement for transmission over wireless access networks". In: *IEEE Journal on Selected Areas in Communications (JSAC)* 21.2 (2003), pp. 216–228. ISSN: 0733-8716. DOI: 10.1109/JSAC.2002.807336.

[33] Sanjay Hegde et al. "FAST TCP in high-speed networks: An experimental study". In: *In Proc. First International Workshop on Networks for Grid Applications*. 2004.

[34] Neal Cardwell et al. "BBR: Congestion-based Congestion Control". In: *Communications of the ACM* 60.2 (2017), pp. 58–66. ISSN: 0001-0782. DOI: 10.1145/3009824.

[35] *BBR Development Google Group*. URL: https://groups.google.com/forum/\#!forum/bbr-dev.

[36] Van Jacobson. *pathchar*. URL: ftp://ftp.ee.lbl.gov/pathchar.

[37] B. A. Mah. *pchar: A tool for measuring internet path characteristics*. URL: http://www.ca.sandia.gov/bmah/Software/pchar.

[38] Arjun Balasingam. "Detecting if LTE is the Bottleneck with BurstTracker". In: to be published. 2019.

[39] Kevin I-Sen. Lai and Mary Baker. "Measuring the bandwidth of packet switched networks". English. PhD thesis. 2002.

[40] R. S. Prasad, C. Dovrolis, and B. A. Mah. "The effect of layer-2 store-and-forward devices on per-hop capacity estimation". In: *IEEE INFOCOM 2003*. Vol. 3. 2003, 2090–2100 vol.3. DOI: 10.1109/INFCOM.2003.1209230.

[41] S. Shioda, Y. Yagi, and K. Mase. "A new approach to the bottleneck bandwidth measurement for an end-to-end network path". In: *Proc. International Conference on Communications (ICC)*. Vol. 1. 2005, 59–64 Vol. 1. DOI: 10.1109/ICC.2005.1494321.

[42] K. Lai and M. Baker. "Nettimer: A tool for measuring bottleneck link bandwidth". In: *Proc. USENIX Symposium on Internet Technologies and Systems*. 2001.

[43] C. Dovrolis, P. Ramanathan, and D. Moore. "What do packet dispersion techniques measure?" In: *Proc. IEEE INFOCOM 2001*. Vol. 2. 2001, 905–914 vol.2. DOI: 10.1109/INFCOM.2001.916282.

[44] Nimantha Baranasuriya et al. "QProbe: Locating the Bottleneck in Cellular Communication". In: *Proc. 11th ACM Conference on Emerging Networking Experiments and Technologies (CONEXT)*. CoNEXT '15. Heidelberg, Germany: ACM, 2015, 33:1–33:7. ISBN: 978-1-4503-3412-9. DOI: 10.1145/2716281.2836118.

[45] Srikanth Sundaresan, Nick Feamster, and Renata Teixeira. "Home Network or Access Link? Locating Last-Mile Downstream Throughput Bottlenecks". In: *Passive and Active Measurement*. Ed. by Thomas Karagiannis and Xenofontas Dimitropoulos. Cham: Springer International Publishing, 2016, pp. 111–123. ISBN: 978-3-319-30505-9.

[46] *Linux Kernel*. URL: https://www.kernel.org/doc/Documentation/networking/timestamping.txt.

[47] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *J. Machine Learning Research* 12 (2011), pp. 2825–2830.

[48] Youngbin Im et al. "I Sent It: Where Does Slow Data Go to Wait?" In: *Proc. 14th EuroSys Conference*. EuroSys '19. Dresden, Germany: ACM, 2019, 22:1–22:15. ISBN: 978-1-4503-6281-8. DOI: 10.1145/3302424.3303961.

[49] *Broadcom 4366E*. URL: https://www.broadcom.com/products/broadband/xdsl/bcm4366e.

[50] D. J. Leith and W. E. Leithead. "Survey of Gain-Scheduling Analysis and Design". In: *International Journal of Control* 73.11 (2000), pp. 1001–1025.

[51] *Android Studio*. URL: https://developer.android.com/studio.

[52] Wikipedia contributors. *Java Native Interface — Wikipedia, The Free Encyclopedia*. [Online; accessed 6-August-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Java_Native_Interface&oldid=902520375.

[53] Jagdish Prasad Achara et al. "WifiLeaks: Underestimated privacy implications of the ACCESS-WIFI-STATE android permission". In: *Proc. 7th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)* (July 2014). DOI: 10.1145/2627393.2627399.