THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART

ALBERT NERKEN SCHOOL OF ENGINEERING

Fair TCP Channel Access for IEEE 802.11 WLANs

by

Christopher Sang

A thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Engineering

February 26, 2014

Dr. Sam Keene

Advisor

THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART

ALBERT NERKEN SCHOOL OF ENGINEERING

This thesis was prepared under the direction of the Candidate's Thesis Advisor and has received approval. It was submitted to the Dean of the School of Engineering and the full Faculty, and was approved as partial fulfillment of the requirements for the degree of Master of Engineering.

Dr. Teresa Dahlberg

Dean, School of Engineering

Dr. Sam Keene

Candidate's Thesis Advisor

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Keene, for all of his hard work teaching and guiding me through this project. I also want to thank Max Kang for helping me with my research, particularly on the topic of packet capture. Lastly, I would like to thank my family. This project would not be possible without all of your support.

ABSTRACT

TCP exhibits unfairness in IEEE 802.11 WLANs due to the high rate of channel errors, frame collisions, and delays associated with wireless transmission. While much existing work has focused on improving TCP throughput in the presence of random data loss and delays, it has not completely addressed the problems that occur when TCP segments are involved in collisions. For example, when TCP acknowledgments (ACKs) collide with data segments, the data almost always survives due to the packet capture effect, while the ACKs cannot be decoded. Excess ACK errors can create unintended behavior in TCP congestion control algorithms, leading to slow data rates and unequal throughput distribution across parallel connections. Fair TCP Channel Access (FTCA) is a comprehensive MAC-layer solution developed to fix these issues. FTCA prioritizes TCP control packets at the MAC layer using quality-of-service mechanisms in the 802.11 standard. It reduces TCP ACK delays and prevents collisions between TCP control packets and data segments that could cause improper protocol operation. FTCA is designed to be easily deployable in existing networks. Using NS-3 network simulations, we perform a thorough comparison of FTCA against other algorithms in capture-enabled WLANs. We show that FTCA significantly improves TCP fairness with minimal throughput overhead.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The widespread use of Internet-connected electronic devices is placing an increasing demand on wireless networks. Consumers are accessing the Web through portable devices including laptops, smartphones, tablets, and e-readers. At the same time, manufacturers are incorporating wireless connectivity into non-portable systems such as televisions and game consoles. The use of wireless networking is only expected to continue as the electronics industry experiments with the next-generation of wearable gadgets and home automation technology. As the number of devices continues to increase, so does the level of Internet traffic generated by them. Today, the typical user is not only a consumer of data, but a producer as well [1]–[3].

As a result, the end-to-end characteristics of an Internet connection have changed. While the Internet backbone is still composed of high-speed wired links, homes and businesses are deploying wireless local area networks (WLANs). Unlike wired connections, these WLANs have lower data rates. They are subject to external interference, which can lead to high bit error rates and packet losses. Since the wireless medium is shared between nodes, packet collisions and transmission delays also occur [4], [5].

The Transmission Control Protocol (TCP), a transport-layer mechanism for ensuring the reliable delivery of data over a network, is not optimized for use in wireless links. TCP was introduced before the advent of 802.11 WLANs, and although it has undergone numerous modifications since then, components of TCP still fail to address the unique challenges of wireless transmission. These shortcomings are resulting in slow, unreliable connections and unfair sharing of network resources.

2. BACKGROUND

2.1. The Transmission Control Protocol

TCP [6] is a connection-oriented transport layer protocol. A TCP sender receives a stream of data from the application layer, separates the data into segments, and appends a header to each segment before passing it down to the network layer for further processing. The format of the TCP header is depicted in Figure 2.1. The header includes a 32-bit sequence number, a 32-bit acknowledgment number, a checksum, and a set of flags used to signal additional information to the TCP receiver. The options field in the TCP header is usually left blank, so the typical size of the header is 20 bytes [5].

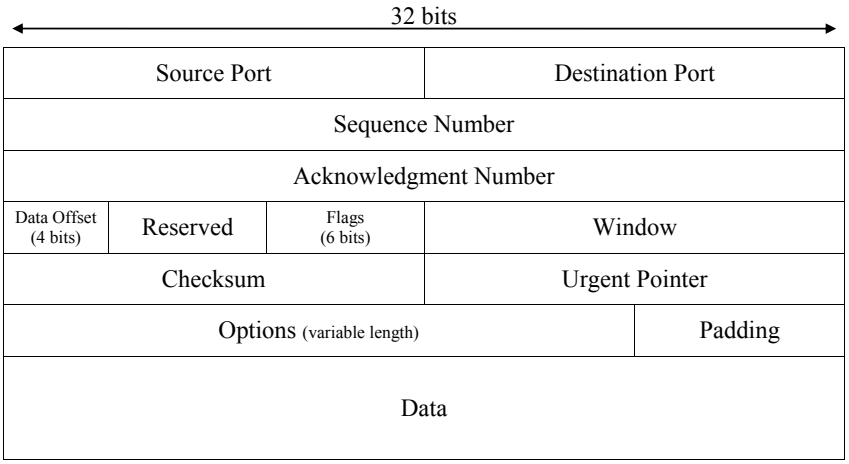


Figure 2.1: TCP Header Format

Additionally, a TCP sender may transmit a segment consisting of only the header, and no data. These segments, known as *TCP control packets*, are used to report state information such as the opening and closing of a TCP connection, as well as the successful delivery of

a segment. Together, the header information in data segments and control packets is used to guarantee several critical services including reliable data transfer and congestion control.

2.1.1. Reliable Data Transfer

A TCP sender maintains a 32-bit sequence number for the lifetime of the connection. Whenever the sender transmits a data segment, it sets the sequence number of the segment to the current sequence number and then increments the current sequence number by the length of the data. The sequence numbers are used by the sender and receiver to identify lost, corrupted, or reordered segments.

When a receiver obtains a data segment (passed up from the network layer), it verifies the checksum and inspects the sequence number and data length. At this point, one of four different possibilities may occur: In the first case, the checksum may be invalid. This event indicates that one or more bits in the segment have been corrupted due to channel errors. The receiver must drop any bad segments that it encounters. A second possibility is that the checksum is valid but the sequence number of the segment is less than the sequence number that the receiver is expecting. This means that the segment is a retransmission of data already seen by the receiver. The receiver must drop this redundant data. In the third case, the sequence number of the segment is equal to the expected number. The receiver accepts the segment and delivers it up to the application layer. The receiver's next expected sequence number is now equal to the current expected sequence number plus the length of the accepted data. Finally, in the fourth case, the sequence number can be greater than the expected value. This indicates that a valid portion of data has been received but that a previous portion of data in the sequence has not been seen yet. A simple TCP implementation might drop the segment, treating it as invalid. More complicated TCP implementations also exist that will buffer the out-of-sequence packets and wait to fill in the gaps in the data stream [7].

The TCP receiver periodically transmits acknowledgments (ACKs) to notify the sender about which packets have been received successfully. An ACK is a control packet in which

the acknowledgment number is set to the receiver's next expected sequence number. A TCP implementation might transmit an ACK for every segment received. However, for performance reasons, a common strategy is to send an ACK for every two packets, or if the connection has been idle for a set period of time. If the TCP connection is bidirectional (the receiver has data to send back to the sender), ACK information can be included in the header for the data segments and no additional control packets are necessary [5].

Under ideal conditions, the sender receives an ACK segment with an ACK number that is equal to the sequence number of the next data segment to be transmitted. However, if a segment has been lost, the ACK number will be lower, indicating that the sender must retransmit the data segment with that number. Multiple duplicate acknowledgments indicate that the same segment has been lost several times in a row and must continue to be transmitted until successful.

Note that it is possible for an ACK to be lost or corrupted in the same way that data segments can be lost. Also, duplicate ACKs cannot be generated if every data segment in the stream never arrives at the receiver. Therefore, the TCP sender must also maintain a retransmit timer as part of its state [8]. If a retransmission timeout (RTO) occurs, the sender retransmits the data segment corresponding to the last seen ACK number.

2.1.2. Connection Setup and Teardown

A TCP connection is established when the sender and receiver exchange a series of three control packets, known as the *three-way handshake* [5]. First, the sender selects an initial sequence number (usually at random) and then transmits the sequence number within a SYN segment, a control packet with the SYN flag set. When the receiver obtains the SYN segment, it allocates resources for the connection and then replies with a SYNACK segment. Note that the ACK number of the SYNACK segment is set to the client's next expected sequence number. Finally, the sender replies to the SYNACK by transmitting an ACK.

If the sender has transmitted a SYN but does not receive the SYNACK before the RTO,

it retransmits the SYN. Similarly, the receiver will retransmit SYNACKs until it receives a valid ACK. Once the three-way handshake is complete, the sender is aware of which sequence number to use for its next data segment and the receiver knows which sequence number to expect, so data losses can easily be detected using the reliable data transfer procedures previously discussed.

When the data transfer is complete, the sender transmits a FIN control packet to the receiver. The FIN segment notifies the receiver to deallocate resources for the connection. The receiver will reply to the FIN by sending an ACK. In addition, the receiver will transmit its own FIN to notify the sender to deallocate its connection resources. As usual, the FIN exchanges are protected by timeouts to safeguard against lost ACKs.

2.1.3. Congestion Control

A typical end-to-end connection across the Internet is comprised of several different links connected by a series of packet switches and routers. These links can have drastically different capacities based on the type of connection and the current network load, leading to network bottlenecks. For example, a desktop computer may be connected to a router using 802.3 wired Ethernet [9], with a typical link rate of 1 Gbps. However, the router's outbound connection to the Internet service provider may be limited to only 10 Mbps due to hardware limitations and/or traffic shaping. When a sender on the LAN transmits outgoing data at a rate greater than 10 Mbps, the router will attempt to buffer the extra packets until it can catch up. However, if the sender continues to transmit at a high rate, the router will run out of memory and will be forced to drop packets. Continued transmission would waste resources and place unnecessary load on network devices [5].

TCP senders attempt to detect network congestion and limit their send rate when it occurs. A TCP sender maintains a state variable called the congestion window (*cwnd*) for the lifetime of the connection. The sender transmits data until the total amount of unacknowledged data reaches the window size. When the sender obtains ACKs indicating that the data has been

received successfully, it is free to transmit more data, but only up to the congestion window limit.

When the TCP connection is first created, the congestion window is initialized to a low value. As data is transmitted and ACKs are returned successfully, the window size increases. If duplicate ACKs occur, indicating that data has been lost, the sender assumes that congestion-induced buffer overflow is occurring somewhere in the network and the window size decreases.

The exact values by which the congestion window changes is implementation-specific. However, several guidelines have been established in [10]. Each TCP sender has a maximum segment size (MSS) which limits the size of individual data transmissions. TCP begins in a slow start phase where the congestion window increases by at most 1 MSS for every new ACK that is received. This rule causes the window to increase exponentially with respect to the transmission round. At some point, the window reaches the slow start threshold (*ssthresh*). This causes the TCP sender to exit slow start mode and enter congestion avoidance mode. In this phase, the window is increased by approximately 1 MSS per round-trip time (RTT), resulting in a linear characteristic. In practice, a standard implementation is to apply the following update after each ACK:

$$cwnd' = cwnd + \frac{MSS \times MSS}{cwnd} \tag{2.1}$$

The slow start threshold represents the TCP sender's estimate of the current network capacity. Slow start mode is designed so that the transmission rate begins at a low value, but then ramps up to the predicted network capacity as quickly as possible. Once the capacity is reached, congestion avoidance mode allows the TCP sender to gradually probe for a higher rate without placing too much instantaneous load on the network.

If packet loss events occur, the slow start threshold is set to half of the current *cwnd*. The *cwnd* drops based on the type of loss event. Retransmission timeouts (ACK losses or complete data losses) are the most severe, resetting *cwnd* to 1 MSS and causing the sender to

return to the slow start phase. Potential data losses, indicated by 3 duplicate ACKs, cause *cwnd* to decrease based on the following equation:

$$cwnd' = \frac{cwnd}{2} + 3\,\text{MSS} \tag{2.2}$$

After updating *ssthresh* and *cwnd* in response to the duplicate ACKs, the sender also has the option to enter a fast retransmit/fast recovery phase. The lost data segment is immediately retransmitted before the retransmission timer expires. If more duplicate ACKs occur for the segment, *cwnd* increases by 1 MSS for each ACK. Once the segment is successfully received, the sender returns to congestion avoidance mode. Fast recovery is included in TCP implementations such as Reno and NewReno [11]. It is not present in TCP Tahoe [5]. Figure 2.2 provides an example of congestion window behavior in these algorithms.

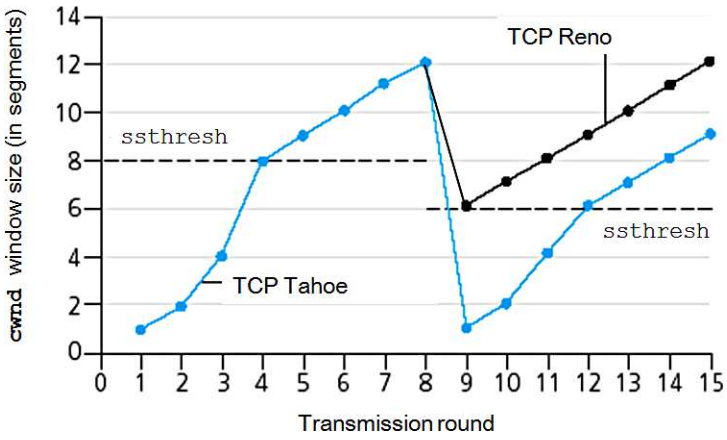


Figure 2.2: An Example of Congestion Window Behavior in TCP Tahoe and Reno [5]

2.1.4. TCP Design Considerations

All TCP congestion control algorithms are designed to ensure fairness. If multiple TCP connections are active at the same time, all senders should receive an approximately equal share of the network capacity. An example of a property that helps create fairness is the halving of the congestion window in response to 3 duplicate ACKs [5]. Suppose that a set of TCP senders is in congestion avoidance mode. As each sender increases its congestion window past the

limits of the network, loss events begin to occur. Senders with larger congestion windows will drop their window by a higher amount compared to senders with lower congestion windows. This allows senders with smaller windows to regain a larger share of the network. Over time, the average congestion window for each node will converge approximately to the same point, causing each node to transmit data at the same average rate.

TCP algorithms are also designed to maximize system throughput. The total data rate for all TCP senders should be as high as possible, while still remaining fair. For example, the use of fast retransmit and fast recovery helps prevent TCP senders from overreacting to sparse data losses. It allows them to maintain larger, but not unsafe, *cwnd* values. Other strategies such as generating a cumulative ACK for every other successful data segment instead of every single data segment reduce control packet overhead. TCP improvements are constantly being developed to meet changing network requirements. However, wireless transmission poses a particularly difficult challenge with its complex channel characteristics and MAC layer behavior [12].

2.2. IEEE 802.11 Infrastructure WLANs

Wireless LANs are governed by the IEEE 802.11 family of standards [4]. Infrastructure WLANs consist of a set of client nodes, known as stations (STAs), communicating via a central access point (AP). A STA is defined specifically as the origin and/or destination of a message. The AP controls which STAs have access to the WLAN, and usually serves as a gateway to outside resources on wired networks. In common 802.11 infrastructure WLANs, the AP is placed at a fixed position, and STAs may be stationary or mobile. The range of the AP is relatively short, and if there are mobile nodes, they move at low velocities. Infrastructure WLANs are commonly seen in homes, offices, and public locations such as coffee shops. Other types of WLANs exist, such as ad-hoc networks where STAs communicate directly, mesh networks where there are multi-hop links, and vehicular networks where STAs are moving at high speeds. These types of WLANs are outside the scope of this work.

2.2.1. The 802.11 MAC Protocol

As with wired LANs, each 802.11 wireless node is assigned a unique MAC address. In addition, the AP is assigned a human-readable name known as the service set identifier (SSID). The AP broadcasts beacon frames at regular intervals advertising its SSID, MAC address, and other link information. STAs looking for information about available APs can perform passive scanning for these beacon frames, or send out probe requests for immediate feedback. Once a STA is aware of the AP, it can join the network by exchanging an association request and response with the AP.

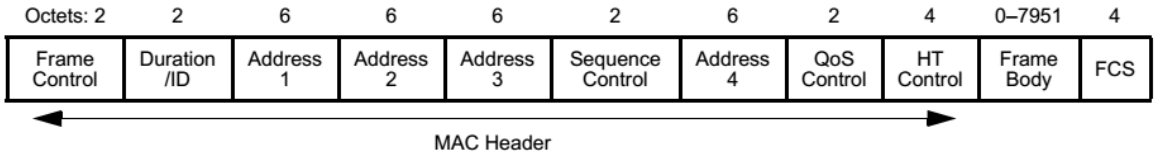


Figure 2.3: 802.11 MAC Frame Format [4]

Frames exchanged between nodes follow the format depicted in Figure 2.3. The frame control field contains flags that identify the type of frame and any special options. Address fields identify the source, destination, transmitting STA, and receiving STA, where applicable. The FCS field contains a 32-bit checksum to detect bit errors. The 802.11 MAC ensures reliable data transfer using an acknowledgment mechanism similar to TCP. Therefore, sequence numbers are included as part of the sequence control field.

Note that the reliable data transfer services provided by TCP and the 802.11 MAC are not redundant. TCP ensures segment delivery at an end-to-end level, whereas the MAC ensures reliability of the link. TCP is required because data can be lost due to router congestion or bit errors on wired links that do not implement error recovery. At the same time, 802.11 WLANs must include a special error recovery layer because of the high error rates associated with wireless transmission. Link-layer acknowledgments allow for faster error detection and retransmission. They can conceal excess errors from TCP which would otherwise cause long

timeouts. However, MAC-layer error recovery is not perfect, as limits are placed on retransmissions for performance reasons, and some transport layer timeouts can trigger before the retransmissions occur.

2.2.2. Collision Avoidance

Frame transmission between associated STAs and the AP is governed by the 802.11 Distributed Coordination Function (DCF). The DCF helps prevent nodes from transmitting frames simultaneously and causing collisions. To determine whether the medium is idle, each STA performs carrier sensing, analyzing incoming signal energy to detect the activity of other STAs in the network. Since constant carrier sensing is inefficient, each STA also maintains a timer called the Network Allocation Vector (NAV), which predicts when the channel is likely to be busy. The NAV is updated whenever a STA receives a frame and is set based on the duration specified in the frame header. Use of the NAV can be thought of as "virtual carrier sensing."

The 802.11 standard defines specific timing relationships between consecutive frames including the Short Interframe Space (SIFS) and DCF Interframe Space (DIFS), as shown in Figure 2.4. The STA can only transmit data after the medium has been idle for at least a DIFS. Once the receiver decodes a frame successfully, it waits for a SIFS and then transmits an ACK. Since the value of a SIFS plus the propagation delay is less than a DIFS, STAs that need to transmit ACK frames are given near-immediate channel access, while other STAs waiting to transmit data will detect the busy channel. This system prevents collisions between data frames and ACKs.

If a STA senses that the medium is busy, it enters a random backoff period. The STA selects a random integer n uniformly from the interval $[0, CW]$. The contention window $CW = 2^x - 1$, where x is initialized to some low positive integer. Once the medium becomes idle, instead of transmitting immediately after a DIFS, the STA will wait for an additional n slot times. If the medium is still idle, the STA is allowed to transmit. However, if it is busy, the

value x is incremented and the STA selects a new backoff time n.

If two STAs are contending for medium access, it is likely that they will choose different backoff times. One STA will begin transmitting before the other one, and the other one will be able to sense its transmission and avoid a collision. However, if the STAs select the same backoff time, then a collision is likely to occur. The minimum and maximum values of *CW* must be chosen to provide a large enough window to minimize the probability of collisions, but avoid unnecessary delays in transmission.

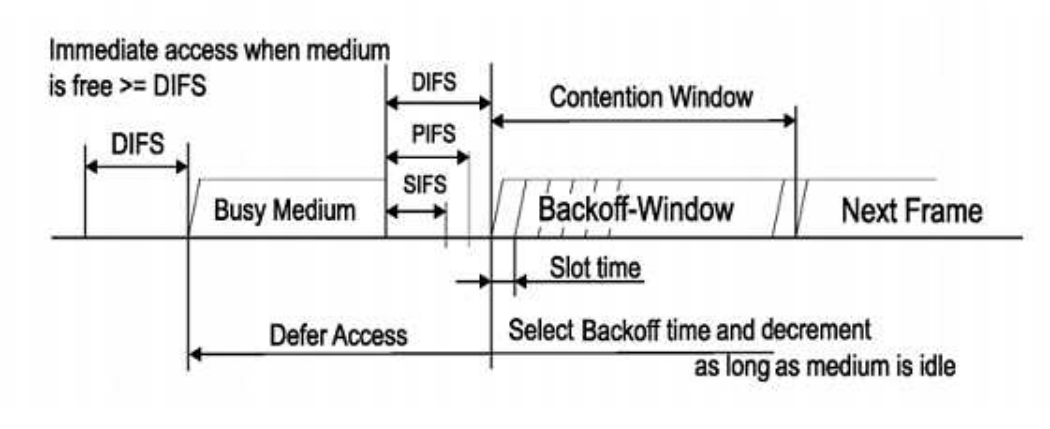


Figure 2.4: 802.11 MAC Interframe Spacing [4]

2.2.3. Hidden Terminals and RTS/CTS

Even with the DCF, collisions can occur due to the hidden terminal problem. Consider the network setup depicted in Figure 2.5. Node B (the access point) is in range of Nodes A and C. However, Nodes A and C are not in range of each other. They will not be able to sense each other's transmissions, so there is a possibility that they will both transmit data at the same time to Node B.

To mitigate this problem, the 802.11 standard defines an optional system known as Request-to-Send/Clear-to-Send (RTS/CTS). If a STA wishes to transmit a data frame to a particular node, it first sends an RTS frame to that node. The duration field of the RTS frame specifies the amount of time that the data transmission is expected to take. The recipient of

the RTS frame broadcasts a CTS frame, which repeats the duration information from the RTS. Any STA that hears any part of the RTS/CTS exchange must update its NAV with the specified duration, as illustrated in Figure 2.6. Thus, in Figure 2.5, if Node A has data to send to B, it will transmit an RTS, causing B to send out a CTS that blocks C from interfering.

With RTS/CTS in place, collisions will still happen if STAs transmit RTS frames at the same time. However, since RTS frames are very small compared to data frames, the cost of an RTS-RTS collision is low. The disadvantage of using RTS/CTS is the delay it introduces to data transmission. The RTS/CTS exchange introduces an overhead of two control packet lengths plus 1 RTT to every data packet. For packets such as TCP ACKs, this overhead can be problematic. As a result, a standard practice is to enable RTS/CTS only when transmitting frames above a specified length [4]. When used for large packets, RTS/CTS significantly improves saturation throughput and network efficiency, especially when there is a high number of contending nodes [13].

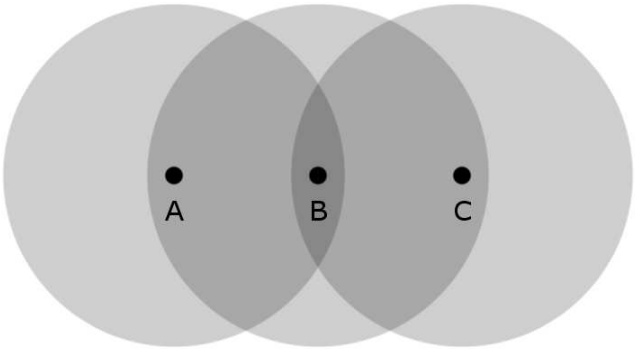


Figure 2.5: An example of the hidden terminal problem. The shaded regions depict the transmission range of each node.

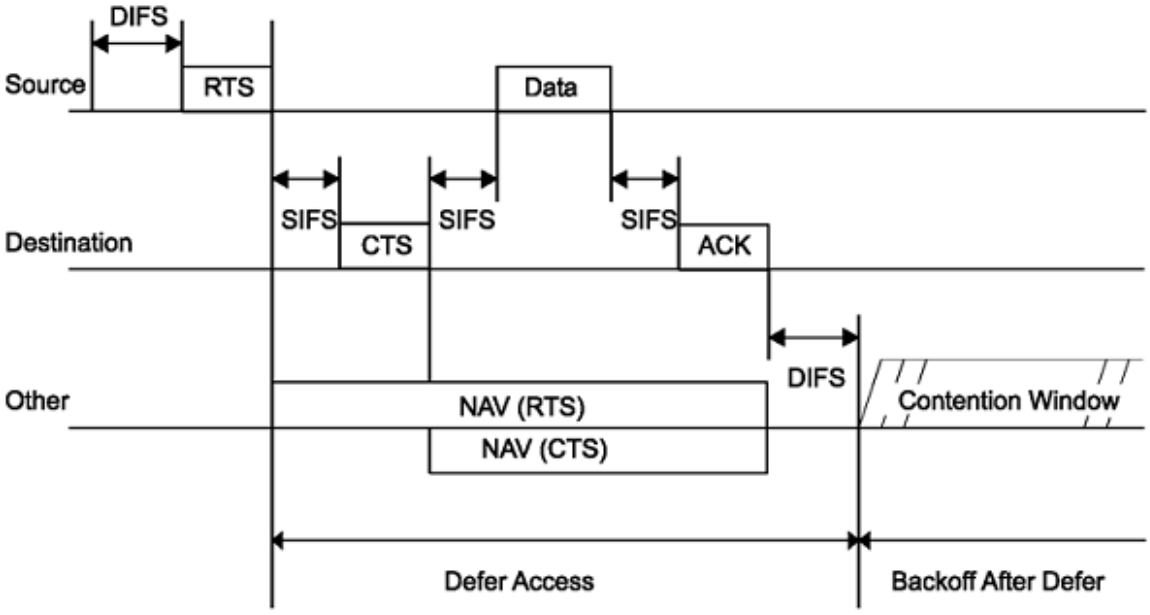


Figure 2.6: NAV Setting due to RTS/CTS Exchange [4]

3. TCP IN WIRELESS NETWORKS

While TCP provides a dependable mechanism for reliable data transport and congestion control in wired networks, it has been shown to perform poorly in wireless links. For example, Figure 3.1 shows the average throughput attained by 8 parallel TCP connections in a simulation of a standard 802.11 WLAN. All STAs are in range of the AP and establish TCP upload connections. The connections achieve wildly different data rates. One TCP sender transmits at 3.4 Mbps while another is limited to only 0.54 Mbps. TCP senders are experiencing low throughput and unfair network access. To understand why these problems occur, we need to examine the core assumptions that TCP implementations make, and how wireless networks violate these assumptions.

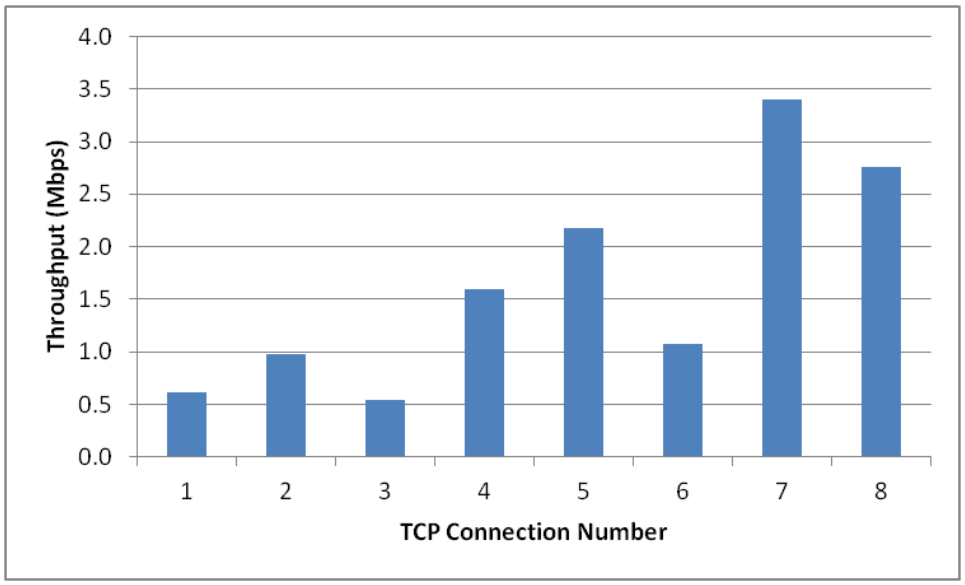


Figure 3.1: Throughput for 8 Parallel TCP Connections in an 802.11 WLAN

3.1. Random Packet Loss Events

Wireless networks are subject to higher bit error rates than wired networks due to factors such as random noise, path loss, and external interference. The loss of TCP segments to these errors disrupts congestion control algorithms. When TCP data segments are randomly lost, the receiver will detect gaps in the sequence numbers and return duplicate ACKs to the sender. The duplicate ACKs cause the sender to decrease its congestion window and slow start threshold. The sender decreases its rate because it assumes that packet losses are due to network congestion. However, in wireless networks, random packet losses and network congestion are two independent problems.

Decreasing the congestion window causes an instantaneous drop in the TCP throughput, while decreasing the slow start threshold prevents the TCP sender from recovering from random loss events. If the slow start threshold is too low, the TCP sender will spend too much time in congestion avoidance mode. The window will increase too slowly, and the throughput will take much longer to return to the optimal value, just below the actual capacity of the network. Since the time spent in congestion avoidance is longer, the probability that the sender experiences a second loss event before reaching the optimal throughput also increases. As a result, one random loss event can trigger improper congestion control behavior which persists for the lifetime of the TCP connection. Unfairness can occur if some TCP senders experience their first random loss event sooner than others.

Figure 3.2 provides an example of this problem. Within the first 5 seconds, Node 1 experiences a loss event that causes its throughput to drop. This frees up the channel for Node 2, which begins to dominate the network. Meanwhile, Node 1 spends over 25 seconds attempting to recover from the loss, but cannot do so effectively because it is in congestion avoidance mode. Finally, its recovery is stopped by a second loss event. Over the entire life of the connection, Node 1 has an average throughput of 1.93 Mbps, but Node 2 has an average throughput of 2.65 Mbps.

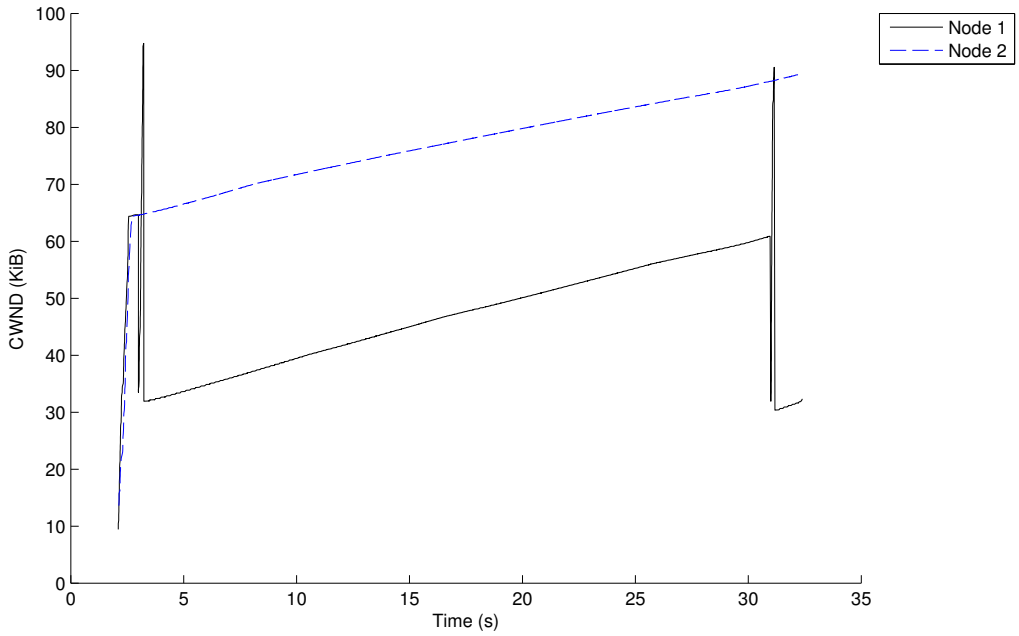


Figure 3.2: Unfair Congestion Window Allocation Among 2 TCP NewReno Senders

These problems are even more severe if TCP control packets are randomly lost. TCP senders maintain an internal retransmission timer to detect lost ACKs. After transmitting a data segment, it takes at least 1 RTT until the sender receives the corresponding ACK. The actual time is influenced by random network delays. Therefore, the retransmission timeout must be significantly greater than the RTT. For each valid data-ACK exchange, the TCP sender records the RTT and then combines the RTT measurements to create a smoothed estimate of the average RTT (SRTT) and the variation in the RTT (RTTVAR) [8]:

$$RTT = t_{ACK} - t_{DATA} \tag{3.1}$$

$$SRTT' = (1-\alpha)\,SRTT + \alpha\,RTT \tag{3.2}$$

$$RTTVAR' = (1-\beta)\,RTTVAR + \beta\,|SRTT - RTT| \tag{3.3}$$

Then, the retransmission timeout is set using:

$$RTO' = \max\{1\,\text{s},\, SRTT' + 4\,RTTVAR'\} \tag{3.4}$$

The minimum RTO is 1 second and can be much greater, especially if the network experiences a high degree of latency or jitter. Therefore, random ACK loss will force the sender to block for long periods of time. When the timeout finally occurs, the sender will reduce its congestion window and slow start threshold, causing the same problems that occur when data segments are lost. Furthermore, the sender may be retransmitting valid data, since ACK loss does not necessarily signal the presence of data segment errors.

The loss of SYN or SYNACK segments is the most costly, since it causes delays in connection setup. The TCP sender will not be able to send any data until it can complete the three-way handshake. This can pose problems for applications that send out many requests using TCP or rely on real-time traffic.

3.2. Packet Collisions

Channel conditions are not the only cause of loss events in wireless networks. The wireless medium is shared between nodes, and errors will occur when transmissions from multiple senders overlap or collide. There are two possible locations where frame collisions may occur. A transmitter-side collision takes place when a frame arrives at a STA at the same time that the STA is transmitting a frame. These collisions can occur frequently with TCP connections since they generate two-way traffic. In a transmitter-side collision, the incoming frame can never be detected or received. Wireless links are half-duplex, meaning that STAs cannot transmit and receive at the same time. Designing hardware that allows for simultaneous transmission and reception is prohibitive due to the differences in power between the outgoing and incoming signals and they way that they interfere with each other [5].

In a receiver-side collision, two frames arrive at a receiver within the same time interval, such that the signals overlap and sum. The probability of these collisions is extremely high in saturated WLANs, as shown in Figure 3.3. Simple models of wireless networks assume that when receiver-side collisions occur, both frames are completely lost. However, recent research shows that it is possible for an 802.11 receiver to recover one of the frames due to a process

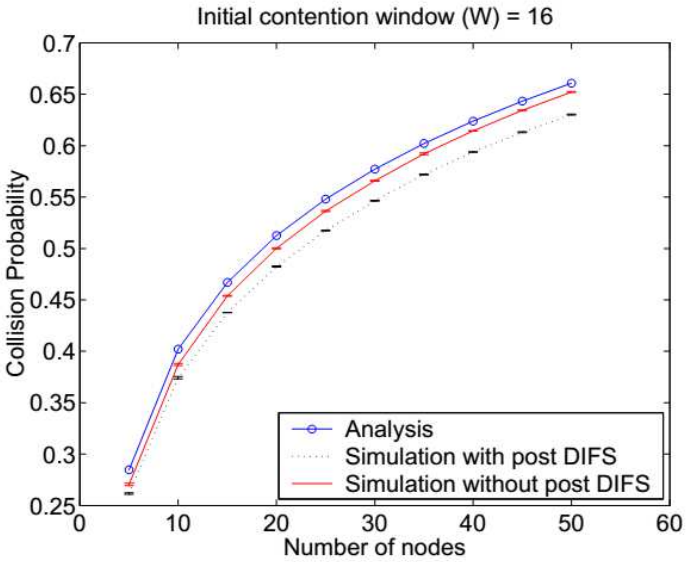


Figure 3.3: Collision Probability Models for Saturated 802.11 WLANs [14]

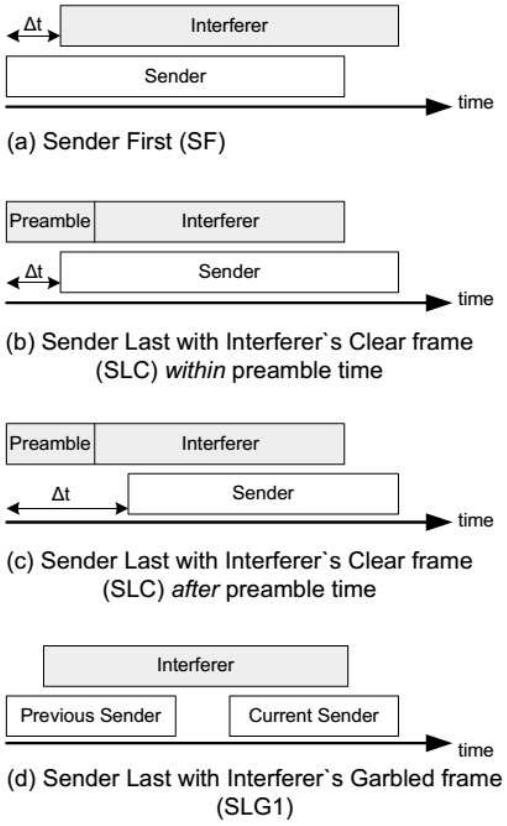


Figure 3.4: Four Packet Capture Cases [15]

known as *packet capture*. This process tends to favor certain nodes over others, leading to excess unfairness in TCP, as will be shown.

3.3. Packet Capture

Packet capture occurs in 802.11 radio modems supporting message in message (MIM) mode * [15], [16]. When an 802.11 frame arrives at a receiver, the receiver performs detection and synchronization based on training symbols in the preamble [4]. If the preamble is decoded successfully, the receiver begins to decode the remainder of the frame, including the MAC layer header and the payload. The receiver verifies the checksum and passes the frame to the next layer. However, the arrival of a new frame during the reception process will interrupt the decoding of the current frame. The MIM-enabled receiver measures the incoming signal energy due to the new frame and compares it to the energy of the current frame. If the difference is greater than a specified message retraining threshold, γ_{MR}, the receiver discards the current frame and begins synchronizing to the new frame [16]. Now, the new frame must be decoded, treating the previous frame as interference.

Packet capture can be divided into 4 cases based on the timing of the incoming frames [15]. These scenarios are illustrated in Figure 3.4. In sender first (SF) capture, the desired frame arrives before the interferer. According to [15], the sender's frame is almost always decoded correctly as long as the sender's signal is stronger than the interference (SIR > 0 dB). This occurs because the interferer does not overlap the sender's preamble enough to cause synchronization errors. The sender's payload and checksum do suffer from interference, but these errors are usually concealed by mechanisms such as forward error correction [4].

In sender last capture with interferer's clear frame (SLC), the desired frame arrives after the interferer. The modem initially synchronizes to the interferer, and then must resynchronize to the desired frame in MIM mode. The required threshold for reliable decoding of the desired

*Modern chipsets such as Atheros are believed to implement MIM, based on experimental evidence. However, the exact details of the implementation are unknown due to their proprietary nature.

frame is approximately 10 dB [15]. The probability of successful packet capture is highest when the desired frame arrives after the preamble of the interferer.

In sender last capture with interferer's garbled frame (SLG), the interfering frame cannot be synchronized with or decoded. This situation occurs when the SINR of the interferer is too low, or when the preamble of the frame is corrupted due to the presence of a third frame. SLG occurs frequently and makes capture of the desired frame more difficult. The receiver will continuously attempt to synchronize to the interferer, even though every attempt will fail. These synchronization failures prevent the receiver from detecting the preamble of the desired frame, unless the difference in signal energy is very high—at least 15 to 25 dB [15].

In all cases, the probability of packet capture is an increasing function of SIR. Small differences in signal energy will cause SF capture to occur, while larger differences result in SLC and SLG. This characteristic introduces unfairness in wireless networks because of the way signal strength varies within a WLAN service area. Path loss causes signal strength to decrease exponentially with distance [17], so STAs that are farther from the AP will capture fewer packets compared to STAs that are closer. Walls and windows also introduce constant partition losses in the range of 1 to 30 dB [17], covering all possible capture thresholds. Therefore, some STAs will experience higher rates of TCP segment loss simply due to their location. They will scale back their congestion window too often, allowing other senders to gain excessive control of the channel.

Packet capture experiments also suggest that TCP ACKs are more vulnerable to capture failures due to their small size. When a TCP ACK and a TCP data segment collide, the ACK is usually overlapped completely. Meanwhile, the preamble of the TCP data frame is likely to remain intact, increasing its probability of capture. As with random bit error, the disproportionate loss of TCP ACKs to capture causes excess delays and retransmissions for all nodes.

3.4. ACK Compression

Even if TCP clients do not experience loss events, transmission delays at the MAC layer will interfere with TCP congestion control and packet timing. Wireless links experience burst errors due to time-varying channel conditions such as multipath fading [17]. When burst errors occur, the MAC layer will attempt to retransmit the lost packets to conceal the errors (up to a limit), while queueing any new packets sent down from the network layer. Once channel conditions improve, multiple packets will be transmitted in quick succession. For TCP ACKs, this is known as the *ACK compression effect* [12]. When several ACKs are received within a short period of time, the amount of unacknowledged data in flight decreases sharply, and the sender must transmit new data to fill the gap. At the same time, there is a sudden increase in the sender's congestion window. As a result, the sender tends to transmit too many data segments at once, causing network congestion.

3.5. Forward-Reverse Path Asymmetry

Another related issue is the difference in capacity between the forward path that carries data segments to the TCP receiver and reverse path that returns ACKs to the sender. In general, path asymmetry results from differences in hardware capabilities, traffic shaping, and current network utilization [12]. Since STAs are typically portable, battery-operated devices, they may operate at a lower transmission power compared to APs, reducing the reliability of upstream channel. Internet service providers also limit the capacity of upstream links since consumers tend to download content more often. For the same reason, downstream links might experience more congestion than upstream links.

TCP uploads are also subject to a specific type of MAC layer path asymmetry. Consider a WLAN with n STAs and 1 AP. The 802.11 DCF is designed with fairness in mind, so each node in the WLAN will have a $1/(n+1)$ share of all the opportunities to transmit. If all STAs perform TCP uploads, the total fraction of the transmission opportunities used up by TCP data segments is $n/(n+1)$. However, the fraction of the transmission opportunities available for

TCP ACKs is the AP's share, $1/(n+1)$ [18], [19]. This difference causes MAC layer packet queueing at the AP and excess delays for TCP ACKs. TCP senders will mistakenly interpret reverse path congestion for forward path congestion, causing them to reduce their congestion window and underutilize the channel [12].

4. EXISTING SOLUTIONS FOR WIRELESS TCP

Numerous attempts have been made to improve TCP performance for connections where the first and/or last hop is wireless. These approaches can be classified based on their scope and modification requirements. TCP-only approaches focus on developing better algorithms for acknowledgments, retransmissions, and congestion control. Areas of interest include selective acknowledgments, fast recovery, and smoother congestion window functions. Although the complexity of these algorithms can vary significantly, deployment of these systems is straightforward, involving software patches at each TCP sender.

MAC layer approaches can address the problems of hidden terminals, DCF unfairness, and packet capture. Important issues include when to use RTS/CTS, and how to influence capture probabilities through power and rate adaptation. MAC layer modifications are often developed using open systems such as MadWifi [20] or custom wireless testbeds that allow many MAC layer parameters to be adjusted in software. In real systems, however, MAC layer algorithms are difficult to deploy since they require changes to access point hardware, drivers, and firmware.

In cross-layer systems, the wireless MAC layer performs *deep packet inspection* to determine whether a data stream consists of any TCP segments. The MAC layer can treat these TCP segments differently from other types of data. For example, it can place TCP ACKs into a separate queue, increase retransmission limits, and employ different error correction strategies. The MAC layer can also inform the TCP sender of wireless link conditions by setting special flags in the TCP header. Cross-layer approaches offer the most flexibility but are more difficult to validate and deploy. They require changes at multiple layers in the network stack and break the principle of isolation between the layers.

4.1. Sender and Receiver-Side TCP Modifications

A standard for TCP selective acknowledgments (TCP SACK) is defined in [7]. TCP SACK addresses a common problem with cumulative ACKs and retransmissions in ordinary TCP: Suppose a sender transmits a group of TCP data segments, and that the first segment in the group is lost due to unfavorable channel conditions. Upon receiving the next segments, the receiver will generate duplicate ACKs and drop the out-of-sequence data. The sender will need to restart transmission of the entire group of data, causing it to send redundant information.

TCP SACK allows the receiver to specify the byte offsets of each correctly received out-of-sequence block using the options field in the TCP ACK header. When the sender processes the special ACK, it identifies and retransmits only the failed segments. This system reduces the cost of a lost ACK, but does have some limitations. It requires additional buffering and processing at the receiver. The options field in the TCP header has a maximum length of 40 bytes which might be too small to specify complex patterns of losses. Most importantly, SACK is a reactive strategy and does nothing to prevent congestion in the first place [12], [21]. However, it can be used in conjunction with other congestion control algorithms.

TCP Vegas [22] is a proactive congestion control algorithm that estimates and limits the amount of "extra" segments transmitted which are above the link capacity. If there are too many extra segments, router buffers will overflow and network congestion will ensue. However, it is beneficial to maintain a small amount of extra segments so that the network can respond to short-term fluctuations in bandwidth. To accomplish this goal, TCP Vegas computes the expected and actual throughput of the connection based on data size and recorded RTTs. It linearly adjusts the sending rate so that the total amount of extra data falls within an empirically determined range of 1–3 buffer lengths. TCP Vegas also defines a more conservative slow start period, which performs exponential increase in the sending rate once every other RTT. Through these mechanisms, TCP Vegas maintains a more accurate and stable congestion window compared to TCP Tahoe and Reno.

TCP Veno [23] estimates router backlogs using the same strategies derived in TCP

Vegas. However, instead of interpreting the queue backlog as an way to set the optimal rate, it uses it as an indicator of packet loss. While the backlog is small, TCP Veno assumes that all losses are due to random bit error and avoids drastic reductions in *ssthresh* and *cwnd*. If the backlog is large, then TCP Veno assumes that congestion is occurring, and follows the normal behavior of TCP Reno.

TCP Westwood [24] estimates the available capacity of a connection through statistical analysis on ACKs. The bandwidth estimate is used to improve settings of the congestion window and slow start threshold. For each received ACK, TCP Westwood obtains a sample of the current bandwidth:

$$b_k = \frac{d_k}{t_k - t_{k-1}} \tag{4.1}$$

where d_k is the size of the data segment and $t_k - t_{k-1}$ represents the elapsed time since the previous ACK. Since there can be significant fluctuations in consecutive samples, TCP Westwood applies a low-pass filter to obtain a estimate of the long term average bandwidth, BWE. Special care is taken to ensure that duplicate ACKs and delayed ACKs do not skew the estimate. When congestion occurs, as indicated by a set number of duplicate ACKs or a timeout, the receiver resets the slow start threshold to $BWE \times RTT_{min}$. Duplicate ACKs will cause the congestion window to reset to *ssthresh*, while timeouts reset the congestion window to 1. TCP Westwood recovers from loss events more quickly than TCP Tahoe and Reno, which reset *ssthresh* and *cwnd* based on fixed ratios.

TCP CUBIC [25], a refinement of TCP BIC [26], is in widespread use on modern Internet servers. It is the default TCP implementation in Linux kernels since version 2.6.13. CUBIC is designed for high speed, high RTT networks. Although not specifically targeted at wireless applications, CUBIC incorporates features that could improve wireless performance compared to previous standards.

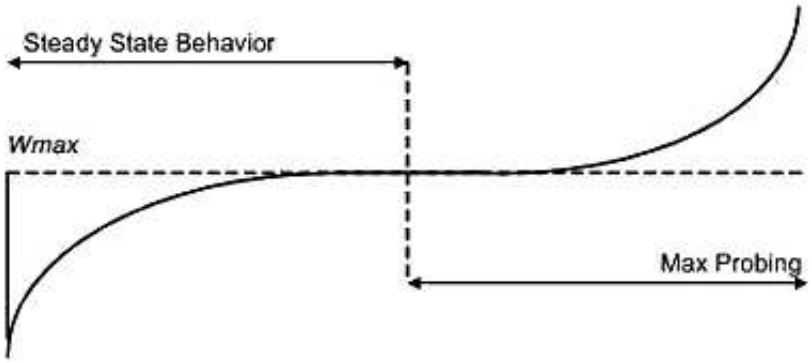


Figure 4.1: TCP CUBIC Congestion Window Function [25]

As the name suggests, TCP CUBIC employs the following cubic polynomial congestion window function, which is also plotted in Figure 4.1:

$$W(t) = C(t-K)^3 + W_{max} \tag{4.2}$$

This function is dependent on the elapsed time t since the last congestion event, and is independent of RTT. This reduces some of the issues associated with ACK compression and path asymmetry. W_{max} represents the window size at the previous loss event, and can be viewed as the sender's share of the network under steady state. When a loss occurs, the congestion window recovers quickly, minimizing the cost of that loss event. As the window size approaches W_{max}, it changes more slowly to maximize network stability. The time for the congestion window to reach steady state is given by

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \tag{4.3}$$

where β and C are tunable parameters that represent a tradeoff between convergence rate and congestion avoidance. After reaching W_{max}, the sender will probe for a greater share of the channel, but will start off slowly to maintain network stability. Then, if the sender succeeds at probing for additional bandwidth, it will begin to increase its congestion window more rapidly, to minimize the time spent in congestion avoidance mode. Thus, TCP CUBIC can recover more

quickly from incorrect values of W_{max} than TCP Tahoe or Reno can recover from incorrect values of $ssthresh$. Overall, TCP CUBIC achieves high RTT-fairness and TCP-friendliness compared to many other algorithms.

4.2. MAC Layer Modifications

The authors of [27] develop a system that exploits packet capture to provide MAC layer quality of service (QoS) differentiation. In this system, high priority STAs transmit at a higher power level than low priority STAs. Since packet capture is a function of SIR, frames originating from the high priority STAs will have a greater probability of capture. The overall result is fewer retransmissions and higher throughput.

The authors show that the power levels required to achieve adequate QoS are fairly low. In one typical setup, a power increase of 10 mW changed throughput by over a factor of 3. These findings are supported by various packet capture experiments including those in [15], which indicate that even a small increase in SIR will cause sender-first capture. This is a practical result, as it means that implementing capture-based QoS will not place a large strain on wireless hardware.

From the TCP perspective, QoS differentiation provides a way to avoid forward-reverse path asymmetry and reduce the ACK error rate. If there are only TCP uploads, prioritizing the AP will improve system throughput. However, this scheme might not scale well to larger networks where there are multiple protocols and connections active simultaneously. There are only two levels of QoS, and adding more power levels could cause the system to exceed the range supported by hardware. It might be possible to set transmit power on a per-packet basis, but this approach would require cross-layer design and place additional strain on wireless transmitters.

DCF+ [28] modifies the 802.11 Distributed Coordination Function to handle two-way traffic more reliably. It uses CTS frames as well as duration/NAV settings on outgoing data frames to reserve the channel for any incoming responses. Figure 4.2 illustrates the typical

operation of this system. The initial data frame (DATA1) sets the NAV of all STAs within range and protects the receiver's MAC ACK from collisions. If the receiver has its own data to return back to the sender (DATA2), it sets another NAV as part of this ACK. The ACK behaves like an RTS frame and causes the sender to reply with a CTS. At this point, the receiver can transmit its data without contention from other STAs.

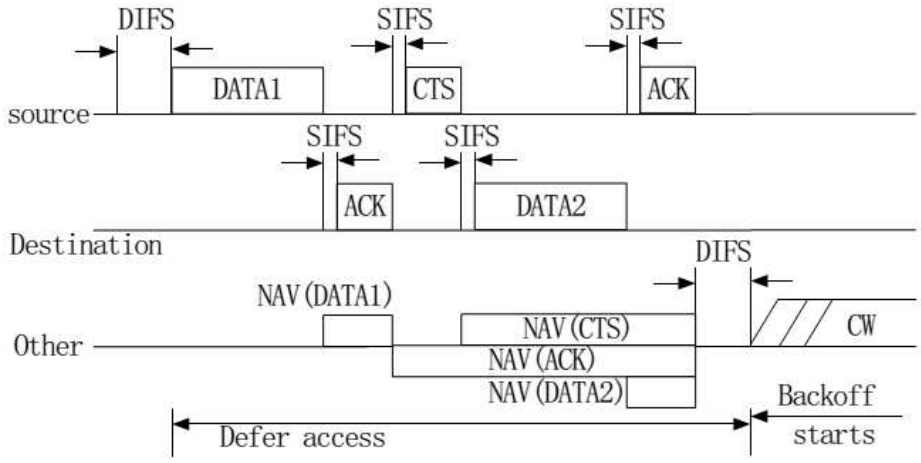


Figure 4.2: Frame Exchange in DCF+ [28]

In the case of TCP traffic, DCF+ allows TCP ACKs to be returned quickly and with fewer errors. DCF+ requires modification of the MAC layer at all wireless nodes. The extra CTS frame introduces a small overhead, but this overhead may be worthwhile, given the cost of a lost or delayed TCP ACK.

4.3. Multi-Layer and Cross-Layer Approaches

Explicit Congestion Notification (ECN) [29] is a standard modification to TCP and IP that allows routers to communicate congestion information to TCP senders and receivers. At the network layer, the IP sender announces ECN support by setting a 2-bit ECN-Capable Transport (ECT) codepoint in the IP header of outgoing packets. ECN-compliant routers implement Active Queue Management (AQM), which involves the use of Random Early Detection (RED)

instead of a tail-drop queue. As a router queue nears full capacity, random packets in the queue are marked with a 2-bit Congestion Experienced (CE) codepoint. The CE marking provides an early warning of network congestion to the IP receiver before buffer overflow occurs.

The receiver echoes the congestion warning back to the sender so that the sender can reduce its rate. When the receiver obtains a CE-marked packet, it passes the warning up to the transport layer. The TCP receiver prepares an ACK with the ECN-Echo (ECE) flag set in the header. When the TCP sender obtains the ACK, it reduces its congestion window accordingly, and reports the window reduction by setting the Congestion Window Reduced (CWR) flag in the following data segment.

ECN is a proactive approach that allows the sender to reduce its rate before any packet loss takes place. It can also be used to differentiate between the different types of losses experienced in a wireless link. For example, if a TCP sender receives duplicate ACKs without ECE markings, it should assume that the losses are due to bit errors and not reduce its congestion window.

ECN requires changes to the transport and network layers at the sender and receiver side, as well as network layer changes to all intermediate routers. Despite these requirements, ECN is not a full cross-layer system, as the intermediate routers are not concerned with the TCP header and do not need to perform intrusive techniques such as deep packet inspection. Instead, ECN can be viewed as a multi-layer approach for improving wireless TCP.

TCP Jersey [30] implements proactive congestion control by combining ECN with bandwidth estimation. It uses the same header formats as the ECN standard, but makes an important change to the IP queue management policy. When the queue length reaches a congestion threshold, instead of marking random packets in the queue with the CE codepoint, all outgoing packets are marked. According to the developers of TCP Jersey, RED queues are highly sensitive to parameter settings, and improper settings for the packet marking probability will have a negative effect on TCP performance. Probabilistic packet marking also reduces the reliability of ECN information for loss differentiation. Therefore, TCP Jersey simplifies the

decisions that must be made by the intermediate routers and makes the TCP sender responsible for deciding which congestion warnings are relevant.

Bandwidth estimation in TCP Jersey is based on the approach in TCP Westwood. However, instead of applying a low-pass filter to the instantaneous bandwidth measurements, TCP Jersey uses a time-sliding window estimator which has fewer parameters and is simpler to implement:

$$R_n = \frac{RTT \times R_{n-1} + L_n}{(t_n - t_{n-1}) + RTT} \tag{4.4}$$

where R_n is the estimated bandwidth for the nth ACK, L_n is the length of the data segment, and t_n is the ACK arrival time. When congestion warnings are detected, TCP Jersey will adjust its congestion window based on the optimum value, $RTT \times R_n$. TCP Jersey achieves high throughput in wireless links having moderate error rates, but requires proper implementation of ECN at all intermediate routers.

Two-phase loss differentiation algorithm (TP-LDA) is a comprehensive technique that discriminates between three different causes of packet loss: random bit error, packet collision, and congestion-related buffer overflow. [31] applies this algorithm to TCP-Friendly Rate Control [32], [33], but the same strategies can work in TCP. In the first phase of TP-LDA, congestion losses are separated from link losses. The sender measures the relative one-way trip time (ROTT) of the connection, which is proven to be highly correlated with the buffer overflow rate. Specifically, there is a large spike in overflow rate as the ROTT approaches a certain threshold. Data losses experienced during a high ROTT period are interpreted as congestion and cause the sender to reduce its congestion window.

In the second phase, the remaining link errors are classified as either packet collisions or losses due to random bit error. TP-LDA uses the error rate of IEEE 802.11 beacon frames as an indicator of the packet collision rate. Beacon frames are sent at regular intervals by the WLAN AP. Since they have short durations and are transmitted at the lowest bit rate, they

have very low error probabilities. However, beacon frames are subject to the same delays and collisions as data frames. If beacon loss and data loss occur within a short time, the network is experiencing congestion and the sender must reduce its rate. Link error differentiation gives TP-LDA a significant advantage over previous loss differentiation algorithms, which treat all link errors as random bit errors. However, it requires many cross-layer modifications to the wireless nodes.

[34] defines a system where TCP ACKs are not forwarded over the wireless part of the connection. Instead, 802.11 MAC layer ACKs are enough to guarantee reliable data transport. A TCP ACK agent is installed at the AP and the STA. Figure 4.3 shows the setup for TCP downloads.

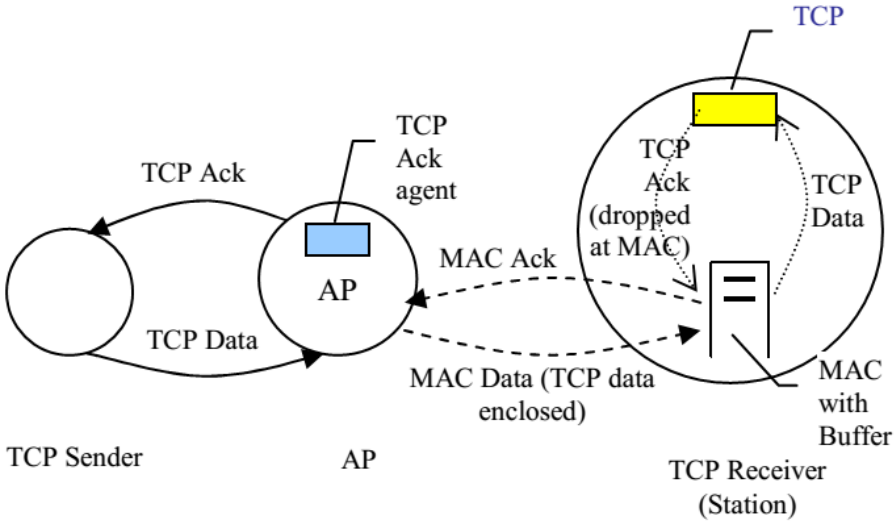


Figure 4.3: TCP ACK Agent [34]

The agent at the AP monitors all TCP data segments being forwarded out the wireless link. As MAC layer ACKs are received for each data packet, the agent generates a TCP ACK to return to the TCP data sender. The ACK information is set so that it appears to come from the TCP data receiver, thus making the system invisible to the end nodes. Meanwhile, the agent at the STA silently drops the real TCP ACKs from the TCP data receiver. The removal

of TCP ACKs from the wireless link reduces network load and lowers the possibility of frame collisions. It mitigates forward-reverse path asymmetry. One possible limitation of this system is that it does not address congestion on the data path. The use of the TCP ACK agent also requires significant modification to the WLAN interfaces and places more computational load on them.

Finally, a more extreme cross-layer solution is to split the TCP connection at the wireless AP into two separate connections [12], [35]. Regular TCP systems such as Tahoe or Reno are used over the wired part, while special TCP systems are deployed for the wireless link. Split TCP is useful for cellular networks, where end-to-end TCP is not enough to handle challenges such as mobile handoff and long disconnections. The base station conceals the presence of the wireless hop from senders on the wired backbone.

Split TCP introduces major design challenges because it extends the role of base stations/access points and completely violates the end-to-end semantics of the TCP connection. For each TCP connection, the base station must now maintain two transport layer sockets and implement all of the buffers and congestion control parameters associated with these sockets. Furthermore, the receipt of a TCP ACK no longer guarantees the successful delivery of the corresponding data segment to the intended receiver. It only indicates that the base station has intercepted the data, buffered it, and will forward it at a later time. To address this concern, the base station can delay the ACK until it receives the ACK on the second interface [35]. However, this method will introduce additional overhead. The demand exists for simpler TCP and MAC layer algorithms that improve wireless performance without imposing additional complexity on network devices.

5. FAIR TCP CHANNEL ACCESS

We propose a new MAC layer solution, *Fair TCP Channel Access* (FTCA), to more completely address the challenges of TCP over 802.11 WLANs without incurring the overhead and implementation difficulties of current systems. FTCA works by prioritizing TCP control packets at the MAC layer, giving them faster access to the wireless channel and reducing collisions between ACKs and data segments. By reducing the TCP ACK collision rate, FTCA prevents unfair packet capture, ACK delays, and ACK compression. It avoids the retransmission timeouts and congestion window backoffs that occur when ACKs are lost. Control packet prioritization also compensates for forward-reverse path asymmetry in TCP uploads. The fast return of ACKs ensures that more TCP senders are actively transmitting data instead of becoming blocked by MAC layer delays. As a result, FTCA improves TCP fairness without a significant loss in saturation throughput.

We develop FTCA using the MAC layer QoS capabilities that were first introduced in 802.11e [36] and are now incorporated into the 802.11 standard [4]. The standard defines four access categories (ACs), listed in Table 5.1, which are used to classify frames based on priority. Each AC maintains a separate transmit queue and gains access to the wireless channel using the Enhanced Distributed Channel Access (EDCA) mechanism, as depicted in Figure 5.1. Instead of transmitting after a DIFS, frames are transmitted after the Arbitration Interframe Space (AIFS) corresponding to their priority. High priority ACs have a shorter AIFS than low priority ACs. Therefore, when multiple STAs contend for the channel, STAs with high priority frames tend to gain access first, while STAs with low priority frames will defer transmission. If contention between ACs occurs within a STA, the high priority frames always obtain access, while low priority frames enter a backoff period. The AIFS is set for each AC as a multiple of

the slot time:

$$AIFS(AC) = AIFSN(AC) \times SlotTime + SIFS \tag{5.1}$$

The minimum AIFSN is 2, so that the shortest possible AIFS is equal to the DIFS. Figure 5.2 shows the relationship between the AIFS and the other types of interframe spaces.

ACs are also prioritized by setting different ranges for the contention window. High priority ACs have smaller allowable values for the contention window compared to low priority ACs. Therefore, if collisions occur, high priority frames experience shorter backoff times.

FTCA assigns all TCP control packets to the highest AC and assumes that all other packets fall into the lower ACs. For example, we may give all control packets the AC_VO designation and allow data segments to have the AC_BK, AC_BE, and AC_VI designations. Alternatively, if data packets need four levels of QoS differentiation, TCP control packets will need to be separated into a fifth AC. We study the former case as it is closer to the 802.11 standard, but note that our results are equally valid in cases where there are additional data ACs. The only requirement of FTCA is that control packets and data packets are kept in separate queues to prevent them from contending and colliding.

The use of AIFS for prioritizing traffic and enhancing network fairness has already been the subject of a few studies. [37] evaluates per-station AIFS as a solution for capture unfairness in 802.11b networks. The authors compare AIFS to other techniques including transmission power control and MAC retransmissions. Like [27], they show that changes to the transmit power have a significant effect on throughput. However, their experiments also reveal situations where power control is not enough to restore fairness. If there is sufficient network asymmetry, differences in received signal strength (and therefore capture probability) can exceed the range of transmit powers permitted by hardware. Furthermore, existing hardware does not have a high enough resolution of power levels and does not support per-packet power adaptation.

Increasing the MAC retransmission limit can be used to protect important packets such

Table 5.1: Default 802.11 Access Categories

Access Category	Designation	AIFSN	CWmin	CWmax	Sample Applications
AC_BK	Background	7	15	1023	FTP
AC_BE	Best Effort	3	15	1023	HTTP, normal data
AC_VI	Video	2	7	15	Video streaming
AC_VO	Voice	2	3	7	VOIP, control packets

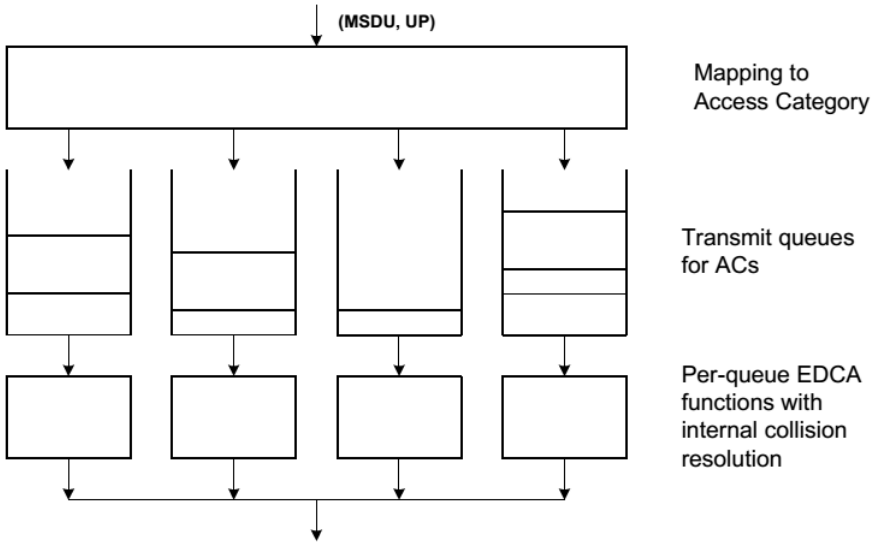


Figure 5.1: Packet Queueing in 802.11e [4]

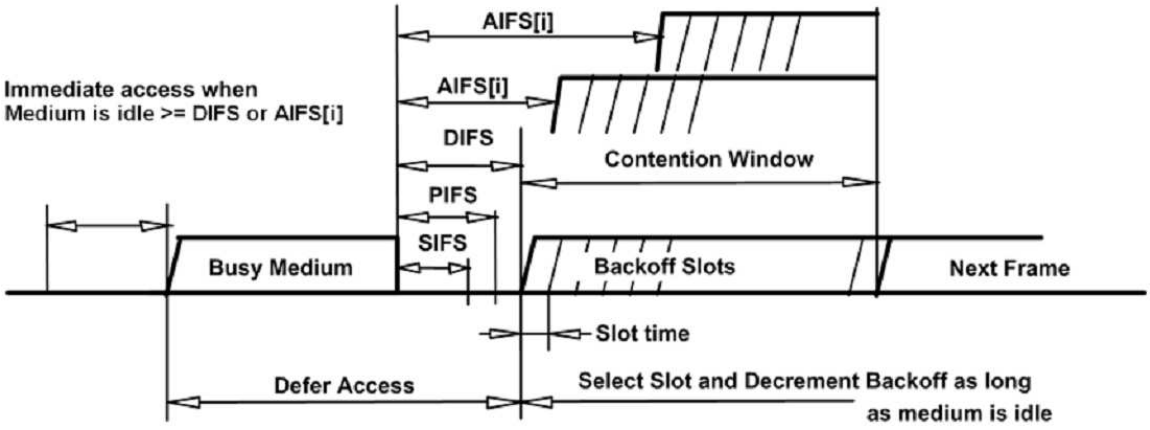


Figure 5.2: AIFS and other Interframe Spaces [4]

as TCP ACKs from loss. The MAC layer automatically recovers from bit errors or collisions and conceals these losses from the transport layer. Unfortunately, this modification could worsen MAC layer delays and queue length. If the MAC retransmission limit was decreased, weaker STAs will waste less time resending corrupted frames, causing UDP throughput to increase. However, the higher error rate will cause TCP performance to suffer. Overall, changing the MAC retransmission limit from its default setting is not advised.

AIFS provides sufficient adjustment granularity and range to compensate for capture unfairness. Decreasing the AIFS value (for all ACs) for weaker senders prioritizes their traffic and prevents collisions with stronger senders that result in unfair capture. Reducing the number of collisions also improves total throughput and reduces delay by a small amount. Our system is based on these concepts, but sets AIFS on a per packet basis instead of a per station basis to focus specifically on protecting the TCP ACKs.

The use of 802.11e for TCP control packet prioritization has also been investigated in [18]. This study attempts to solve the problem of forward-reverse path asymmetry. Recall that for TCP uploads, the AP has a smaller share of transmission opportunities compared to the STAs due to the station-level fairness enforced by the DCF. This results in excess queueing of TCP ACKs at the MAC layer. Prioritizing the ACKs using AIFS allows them to regain a higher share of the transmission opportunities from data segments. This restores path symmetry and increases system throughput, especially when there are a large number of contending nodes in the network.

The authors of this study propose a specific prioritization scheme where the TCP ACKs are assigned to the same AC as their corresponding data segments. Their argument is based on the assumption that the data traffic is spread out across all available queues, which operate at saturation, and that these queues are also shared with the TCP ACKs. Under this assumption, if all TCP ACKs were given the highest priority, they would still contend and collide with the high priority data segments. The low priority ACs would experience less contention, causing them to achieve higher throughput than the high priority ACs, which violates the QoS requirements

of the network. Spreading the ACK traffic across the queues maintains QoS for each data class while preventing excess contention within a particular queue.

However, we argue that [18] is still not an good solution because it maximizes the total throughput across all ACs by compromising fairness within each AC. Consider one of the four queues, AC_BE. All frames in this queue contend and collide with high priority frames from AC_VI and AC_VO. This contention applies equally to all AC_BE frames, and is a normal part of 802.11e. However, within AC_BE, TCP ACK and data segments also contend and collide with each other because they have equal priority. This contention is not normal, and it violates the TCP assumption [12] of low ACK error rates. It is equivalent to the contention seen in ordinary networks without QoS, and results in the same unfairness problems. TCP senders experiencing ACK loss will retransmit and decrease their congestion window excessively. Over time, the high rate of ACK loss causes continued retransmission failures that result in these senders backing off more often than others. The total throughput of AC_BE will still be relatively high compared to AC_BK, but it will be dominated by a small subset of nodes. We resolve this issue by requiring TCP ACKs to have their own exclusive queue, removing the possibility of any ACK collisions that could introduce unfairness.

In [19], the authors conduct a study that demonstrates the extent of TCP upload unfairness in networks without proper ACK prioritization. Figure 5.3 shows the result of one particular simulation. The authors set up ten TCP uploads over an 802.11b WLAN. The TCP SACK option is enabled. Two TCP senders achieve disproportionately high throughput while four of the senders cannot even access the channel. Note the similarity between these results and our simulations.

The authors address this problem by prioritizing TCP ACKs based on AIFS and minimum contention window size. Their approach is quite similar to ours, but focuses on transmission opportunities and EDCA timing relationships rather than the effect of collisions. By giving TCP ACKs nearly unrestricted access to the channel, the ACK rate is regulated by the transport layer instead of the MAC layer, which is the intended behavior of TCP. The ratio of

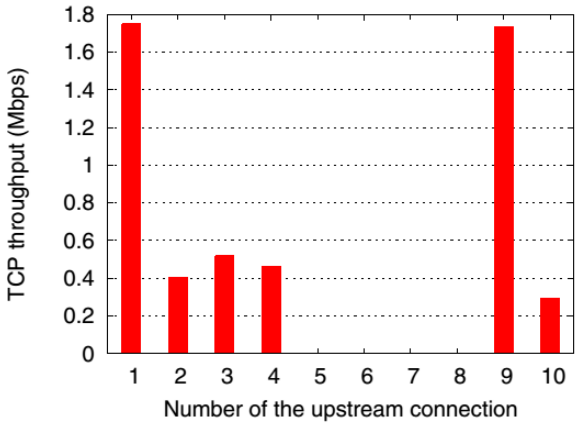


Figure 5.3: TCP Uploads without ACK Priority [19]

transmission opportunities obtained by ACKs compared to data segments is proportional to the ratio of CW_{min} between their ACs. The authors derive an analytic model to predict the throughput of each sender based on AIFS and CW_{min}, assuming that ACK prioritization eliminates all ACK-data collisions. We take a more complete approach and treat ACK-data collision avoidance as one of the main goals of FTCA. In doing so, we validate the authors' assumptions and obtain some new insights into TCP behavior.

The TCP ACK Priority (TAP) system [38] offers a different way of eliminating forward-reverse path asymmetry and ACK delays for TCP uploads. TAP allows the AP to transmit up to n ACKs in one transmission opportunity using multi-destination frame aggregation, another feature in 802.11e [4], [36]. In this system, the first ACK is transmitted immediately after a DIFS. If the AP has additional ACKs in the queue, up to $n-1$ ACKs are sent out in quick succession, separated only by a SIFS or a RIFS (Reduced Interframe Space). Appropriate values of n, determined through simulation, are 4 for basic access and 8 for RTS/CTS. TAP restores path symmetry and ensures that all TCP senders are actively transmitting data instead of waiting for ACKs. However, it is a system that is restricted to TCP uploads. TCP downloads may still encounter collision problems, because TAP cannot prioritize ACKs sent from the STAs to the AP.

Most of these existing solutions also suffer from another limitation—the failure to con-

sider TCP SYN and SYNACK segments. These control packets, used in TCP connection setup, are just as vital to fair protocol operation as ACKs are. Suppose that a STA is attempting to create a new TCP connection in a heavily congested WLAN. There is a high probability that the initial SYN or SYNACK segment will be lost due to collisions or delays, causing the sender to enter a backoff period. The instantaneous fairness impact of SYN or SYNACK loss is more severe than that caused by ACK loss. SYN or SYNACK loss completely prevents the TCP sender from transmitting data, while ACK loss causes a TCP rate reduction. Therefore, FTCA prioritizes all TCP control packets, not only ACKs.

6. DESIGNING NETWORK SIMULATIONS FOR FTCA

6.1. Overview of NS-3

We implement and evaluate FTCA in NS-3 [39], a discrete-event network simulator. NS-3 contains a set of C++ modules that implement the layers of a typical network stack, including TCP, IP, and the 802.11 MAC. Nodes can be set up according to various layouts and mobility models. Then, traffic-generating applications are installed and run on them. During simulations, all packets passing through the network are logged in Wireshark-style [40] packet capture (pcap) files. Trace sources throughout the code output statistics such as packet error counts, and record important events such as changes to the TCP congestion window.

NS-3 contains a basic implementation of a wireless channel in the `YansWifiPhy` module. This component provides a coarse approximation of frame transmission and reception events based on SINR. Transmitters send out frames at a constant power level. When these frames arrive at a receiver, the average SINR for the entire frame is computed based on simple path loss, interference, and noise functions. If the average SINR is above a threshold, the entire frame is received successfully. Otherwise, the entire frame is dropped.

The `YansWifiPhy` model averages out signal-level effects on frames, emphasizing computational efficiency at the cost of simulation accuracy. This is an acceptable tradeoff for simulations that focus on higher processes of the network stack, such as IP routing or queueing. However, `YansWifiPhy` is not detailed enough to analyze packet collisions and capture. Therefore, we have decided to replace `YansWifiPhy` with `PhySimWifiPhy` [41], a complete model of the 802.11 physical layer (PHY).

`PhySimWifiPhy` mimics the behavior of actual 802.11 radio modems. It performs orthogonal frequency division multiplexing (OFDM) modulation/demodulation, interleaving, and forward error correction. Received frames are processed in proper order: First the preamble must be detected in the presence of random noise vectors and interfering signals. Then the signal header is decoded, followed by the payload. `PhySimWifiPhy` supports MIM mode/packet capture. Unfortunately, this system comes with an extremely high computational cost. Simulations involving `PhySimWifiPhy` are up to 1000 or 10000 times slower than those using `YansWifiPhy`.

6.2. Physical Layer Model

In `PhySimWifiPhy`, we set up a standard 802.11a 5 GHz indoor wireless channel model. All nodes transmit at a constant power P_t. A simplified log-distance path loss model [17] is implemented, where the received power P_r is given by

$$P_r = P_t K \left[\frac{d_0}{d}\right]^{\gamma} \tag{6.1}$$

or, in decibels,

$$P_{r,dB} = P_{t,dB} + K_{dB} - 10\gamma \log_{10}\left[\frac{d}{d_0}\right] \tag{6.2}$$

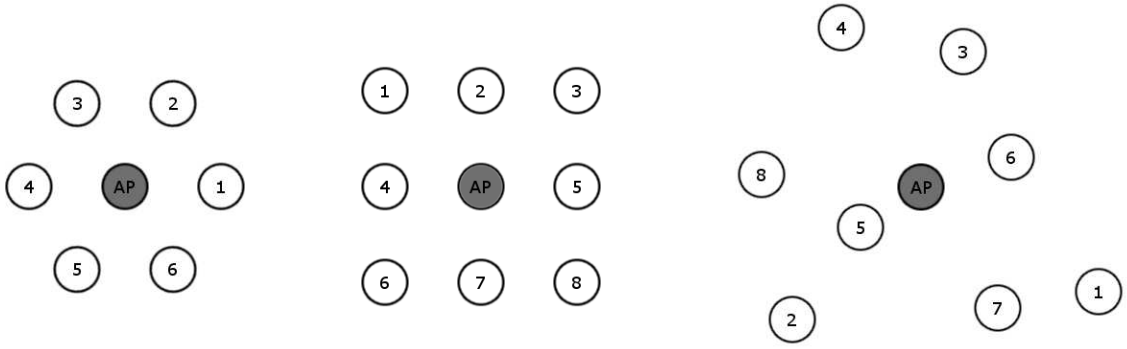
This model is valid for all transmission distances $d > d_0$ and is shown to conform well with real-world channel measurements. For indoor environments, a typical value for the reference distance is $d_0 = 1\,\text{m}$. The corresponding reference loss at this distance is set to the free space path gain

$$K_{dB} = 20\log_{10}\frac{\lambda}{4\pi d_0} \tag{6.3}$$

which results in a value of $K = -46.67$ dB for a center frequency of 5.15 GHz. For the path loss exponent, we use $\gamma = 3.5$, corresponding to a typical home or office environment. Table 6.1 provides a complete listing of these, and other physical layer simulation parameters.

Table 6.1: NS-3 Physical Layer Simulation Parameters

Parameter	**Description**	**Value**
P_t	Transmit Power	18 dBm
d_0	Reference Distance	1 m
K	Reference Loss	-46.67 dB
γ	Path Loss Exponent	3.5
P_n	Noise Floor	-99 dBm
γ_{MR}	MIM/Capture Threshold	8 dB



(a) 6-Node Star (b) 3×3 Grid (c) 8-Node Random Disc

Figure 6.1: Node Layouts for NS-3 Simulation

6.3. Node Layouts

For our simulations, we define three different node layouts: *star*, *grid*, and *random disc*. Examples of these layouts are shown in Figure 6.1. Each layout consists of an AP and a set of STAs. All nodes remain at fixed positions throughout the simulations; cases involving mobility are not considered.

In the star layout, n STAs are evenly spaced on a circle of radius r with the AP at the center. This layout is a highly symmetric configuration that does not reflect real-world situations. However, it is useful for obtaining an upper bound on network performance under fair conditions.

The grid layout consists of a set of STAs evenly spaced d units apart on an $m \times p$ grid. The AP is located at the center of the grid. This layout is common in office environments, where desks/workstations are spaced at even intervals. Grid layouts are also common in hardware testbeds, allowing for easier comparison of results.

In the random disc layout, n STAs are allocated within a circle of radius r, according to a uniform random distribution. The AP is placed at the center. This layout corresponds most accurately with real-world conditions.

Specific parameters for these layouts are chosen so that all nodes are in range of each other: For the star layout we have 6 nodes located at a radius of 5 m from the AP. The grid is 3×3, with a spacing of 5 m. The random disc layout has 8 nodes within a 10 m radius of the AP. Packet error rates for single connections at these distances are small but non-negligible—on the order of 1%.

6.4. Configuration of the Network Stack

Each node contains an instance of the default NS-3 network stack, which includes an implementation of TCP NewReno, IPv4, and an 802.11a OFDM MAC/PHY layer with added QoS support. A `BulkSendApplication` is installed on each STA, which establishes a TCP upload connection with the AP. Data segments are given AC_BE priority and TCP control packets are assigned to AC_VO. The TCP maximum segment size is 536 bytes. All TCP connections operate at saturation. There are no high-level constraints placed on data rate or amount, other than what is already provided by the TCP congestion control algorithm.

Rate adaptation at the MAC layer is disabled. Since all nodes are in range of each other, data frames are transmitted at 54 Mbps, the highest 802.11a supported rate. RTS/CTS

and ACK frames are transmitted at 6 Mbps, the lowest possible rate. When RTS/CTS is used, it is enabled for data packets only.

6.5. Evaluation Metrics

For each scenario, 5 to 8 simulation runs are conducted for 30 seconds each, and the results of the simulation runs are averaged. For each TCP connection i, we measure the overall throughput x_i for TCP data packets arriving at the receiver. The system throughput is defined simply as the sum of the individual throughput values:

$$x = \sum_{i=1}^{n} x_i \tag{6.4}$$

Fairness is measured using Jain's Fairness Index (JFI) [42], defined as:

$$JFI = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} \tag{6.5}$$

JFI values close to 1 indicate network fairness, while values lower than 1 indicate unfairness.

Collisions involving TCP segments are tracked. A data-ACK collision is defined as the loss of a TCP data segment due to interference by an overlapping TCP ACK (the ACK may or may not have been captured). An ACK-data collision is defined as the loss of an ACK due to interference by a data segment. Data-data and ACK-ACK collisions are defined similarly.

7. Simulation Results and Analysis

7.1. TCP Connections

7.1.1. Star Layout

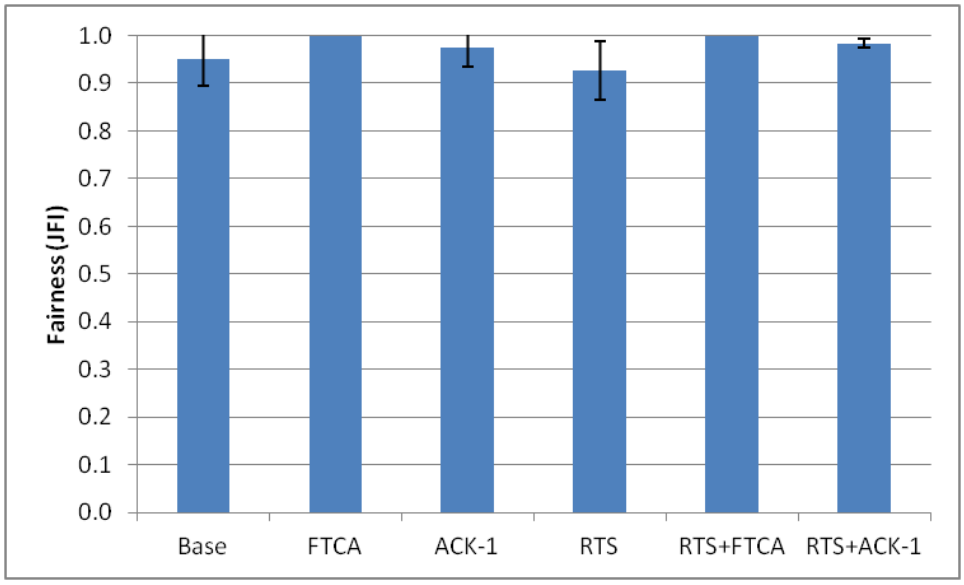


Figure 7.1: Average Fairness for Each System (Star Layout)

Figure 7.1 shows the average fairness observed for simulations of a 6-node star layout. (Error bars in the graph depict the 95% confidence interval.) Due to the symmetry of the layout, the base system without TCP ACK prioritization already achieves a high degree of overall fairness. The average JFI is approximately 0.95. However, a closer inspection of some of the individual simulation runs shows that the fairness is still less than ideal. Figure 7.2 shows the per-node throughput for one particular simulation run. Node 5 attains a throughput of 3.0 Mbps, while Node 6 attains a throughput of 1.8 Mbps. The throughput values for the other four nodes fall in between these extremes. In another case, depicted in Figure 7.3, there is

a good degree of fairness among Nodes 2–5, which all have a throughput of around 2.8 Mbps. However, Node 1 has a 1.5 Mbps throughput and Node 6 has a 0.65 Mbps throughput. These differences are quite significant, even though they still translate to relatively high JFI values.

Figures 7.4 and 7.5 show the evolution of the TCP congestion window for each node in these simulation runs. For $0 \leq t < 2$, there are no active TCP connections. (This part of the simulation was a setup phase.) At $t = 2$ seconds, the TCP senders are enabled and the slow start phase begins. All senders increase their congestion window rapidly and transition to congestion avoidance mode at around $t = 2.5$. However, some nodes experience loss events early on and momentarily end up with lower slow start thresholds. By $t = 5$, it becomes apparent that these nodes are obtaining a much lower share of the network throughput.

The unfair distribution of the congestion window size and the throughput continues for the life of the connection. Nodes with small slow start thresholds cannot quickly recover from loss events brought on by wireless link conditions. The first loss event also triggers subsequent loss events for these nodes. Meanwhile, the dominant nodes detect little or no network problems, so they continue to increase their congestion window without bound. As a result, the TCP congestion control process does not properly converge.

Enabling FTCA leads to a near-perfect average JFI of 0.9996, and analysis of the individual simulation runs confirms its improvement over the base system. Figure 7.6 depicts a typical situation where all nodes have a throughput of around 2.3 Mbps. The TCP congestion window for each node is plotted in Figure 7.7. Some unfairness is present for the first 5–10 seconds of the connection. This time, however, the dominant nodes experience loss events and the congestion windows converge to a relatively stable point.

Next, we examine the performance of FTCA when it is used in conjunction with RTS/CTS instead of the 802.11 basic access mechanism. Enabling RTS/CTS alone leads to an average fairness of 0.93, comparable with the base system. Using FTCA with RTS/CTS improves the fairness to 0.9995. The per-node throughput and congestion window relationships are similar to those for basic access.

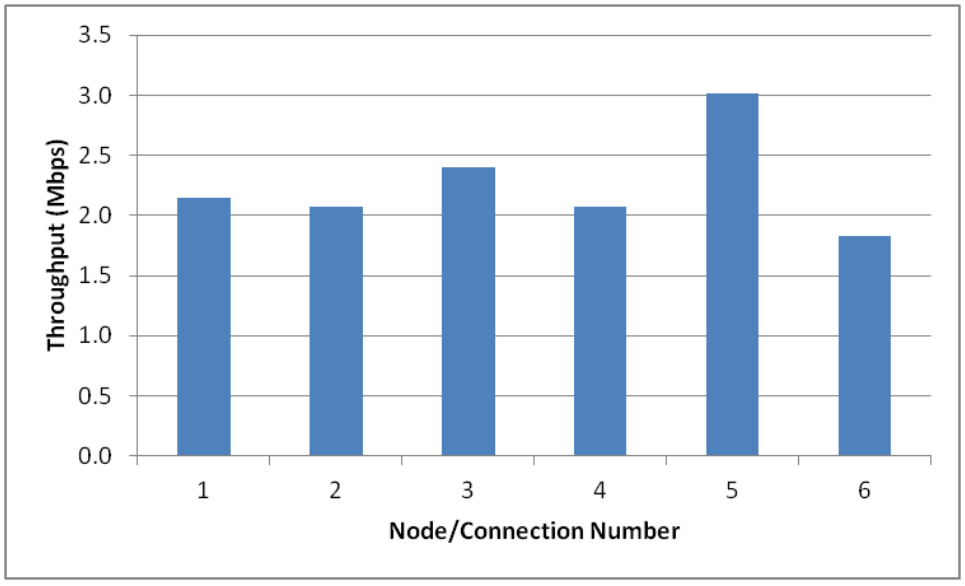


Figure 7.2: Per Node TCP Throughput (Base System, Star, Run A)

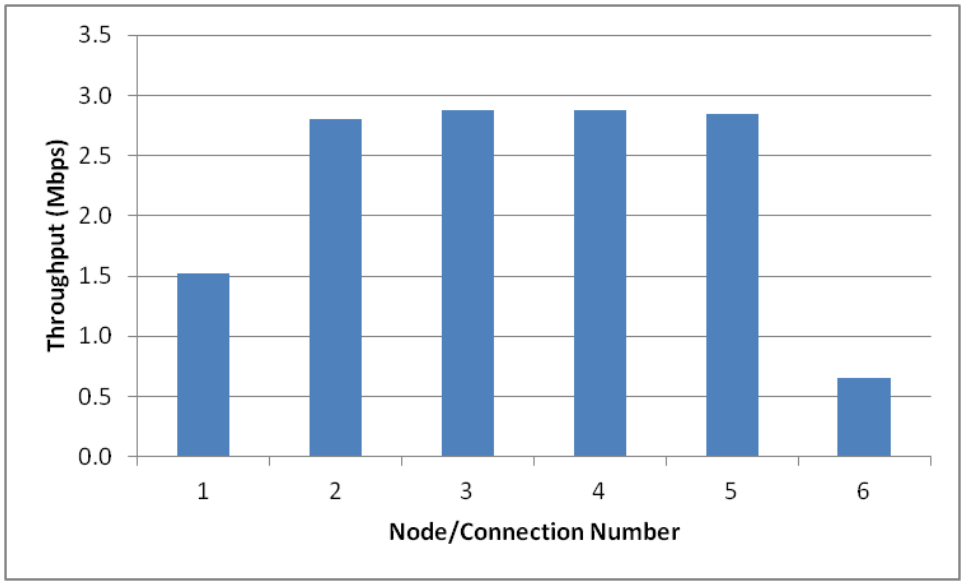


Figure 7.3: Per Node TCP Throughput (Base System, Star, Run B)

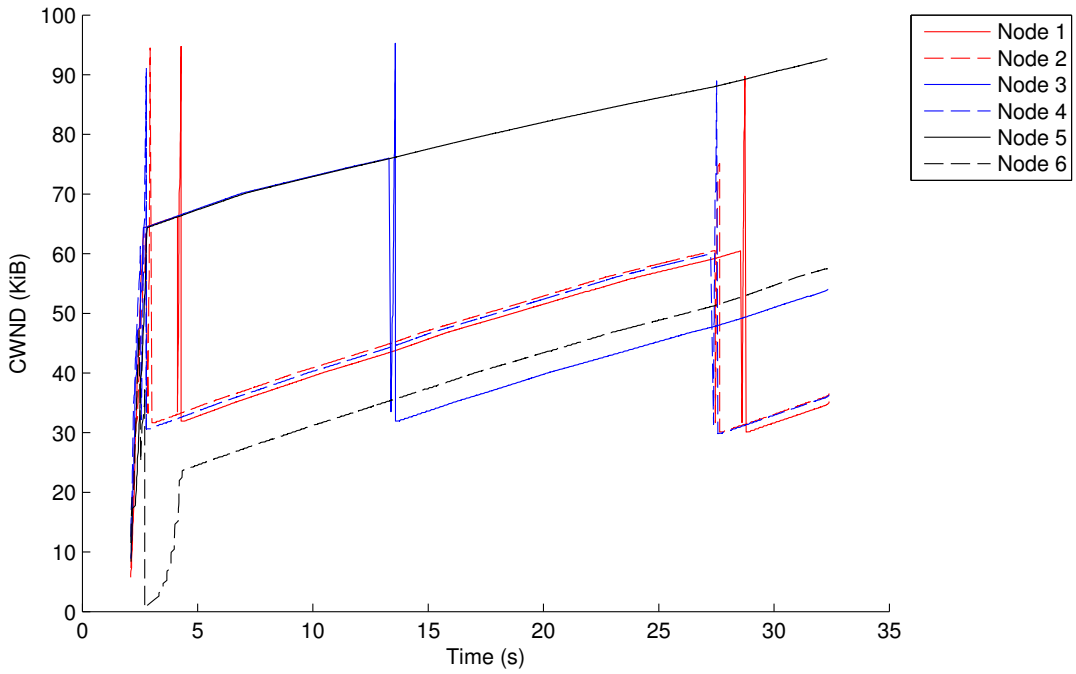


Figure 7.4: TCP Congestion Window (Base System, Star, Run A)

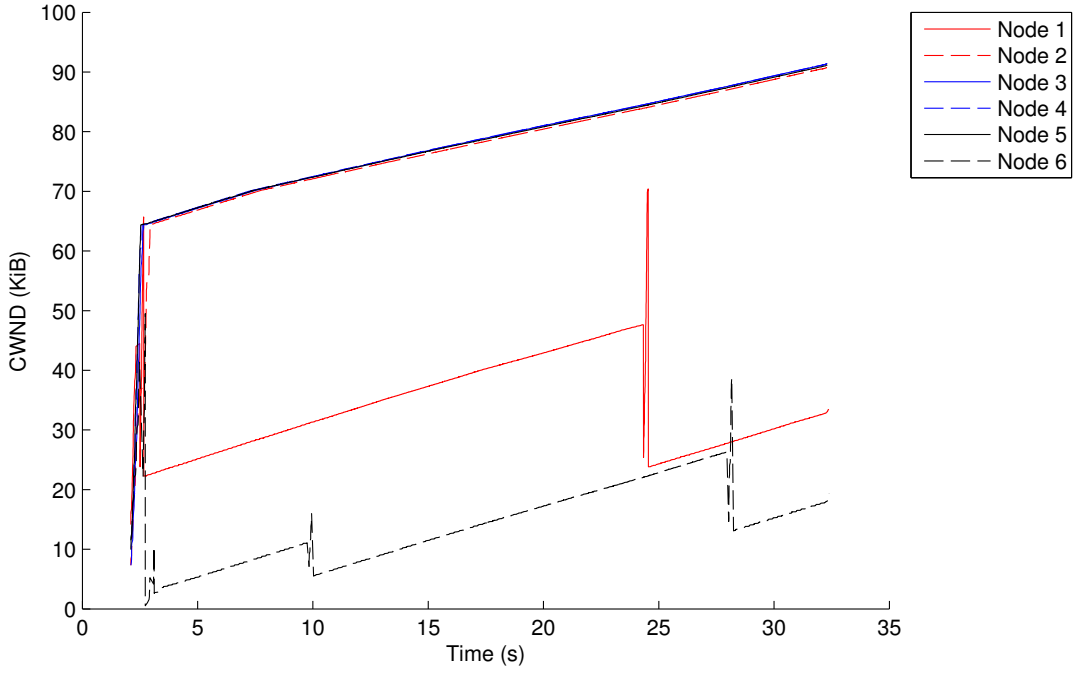


Figure 7.5: TCP Congestion Window (Base System, Star, Run B)

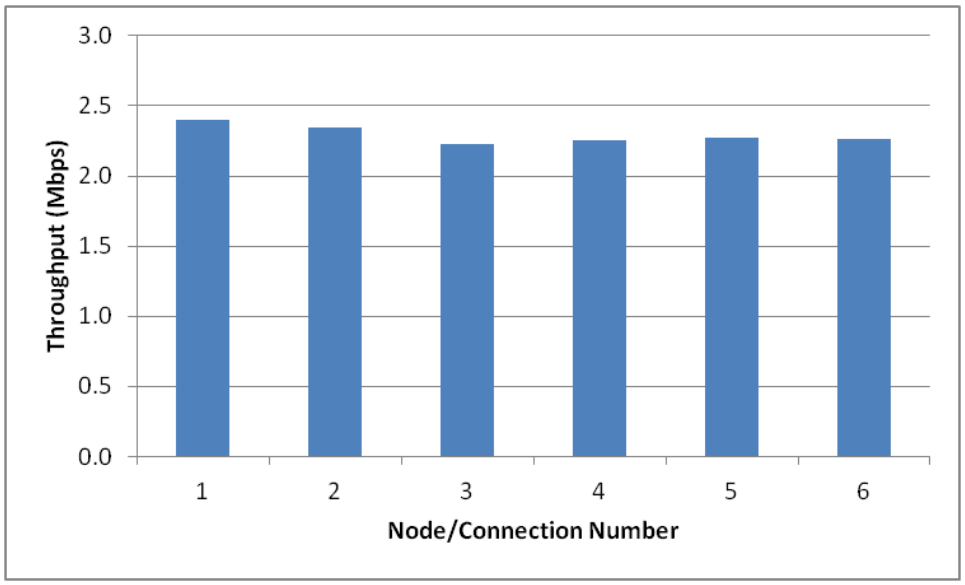


Figure 7.6: Per Node TCP Throughput (FTCA, Star Layout)

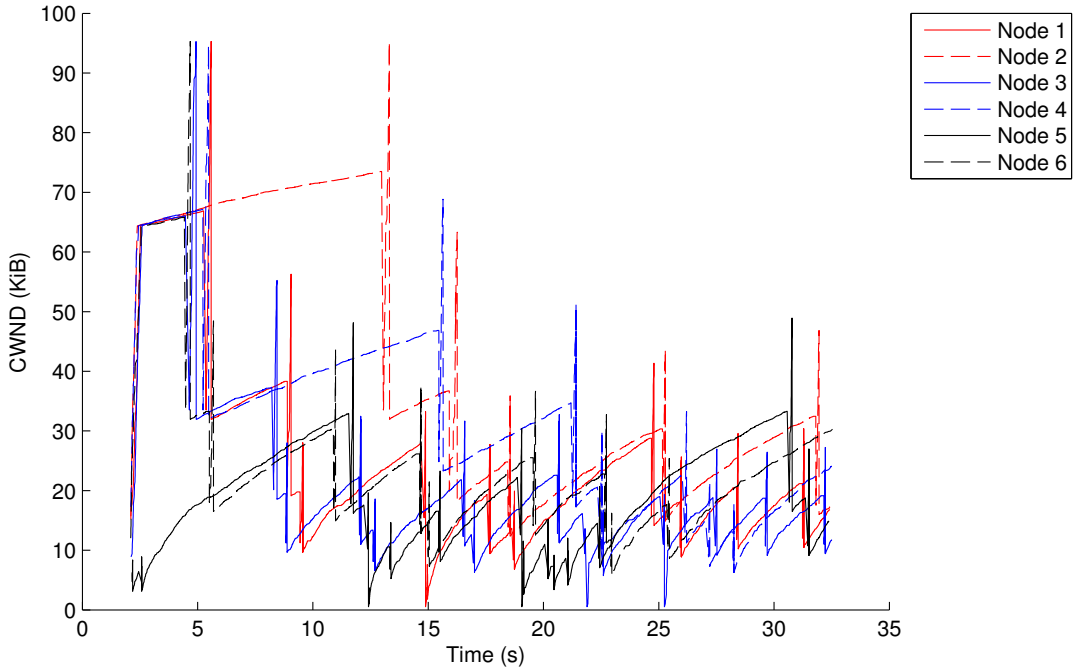


Figure 7.7: TCP Congestion Window (FTCA, Star Layout)

It is also useful to compare what performance gains are achievable through limited changes to the TCP parameters. *ACK-1* refers to a simple TCP modification where receivers send out ACKs immediately for every single data segment instead of for every other segment (*ACK-2*). This strategy reduces the cost of lost TCP ACKs and addresses some of the problems that arise from cumulative acknowledging and delayed acknowledging. From Figure 7.1, it can be seen that ACK-1 improves slightly on the base system, increasing fairness to 0.97. With RTS/CTS enabled, the fairness is 0.98. These performance gains are still less than those for FTCA, however.

To evaluate the reliability of all systems and obtain a lower bound on their performance, we record the highest throughput difference between nodes for each system across all simulation runs. These results are presented in Figure 7.8. Here, the base system has a maximum throughput difference of 2.22 Mbps, compared to the 0.19 Mbps for FTCA. Even though the base system performs adequately overall, it sometimes exhibits high unfairness. FTCA not only outperforms the base system on average, but it is also more consistent. Note that ACK-1 with RTS/CTS performs well too, with a worst-case throughput difference of 0.65 Mbps.

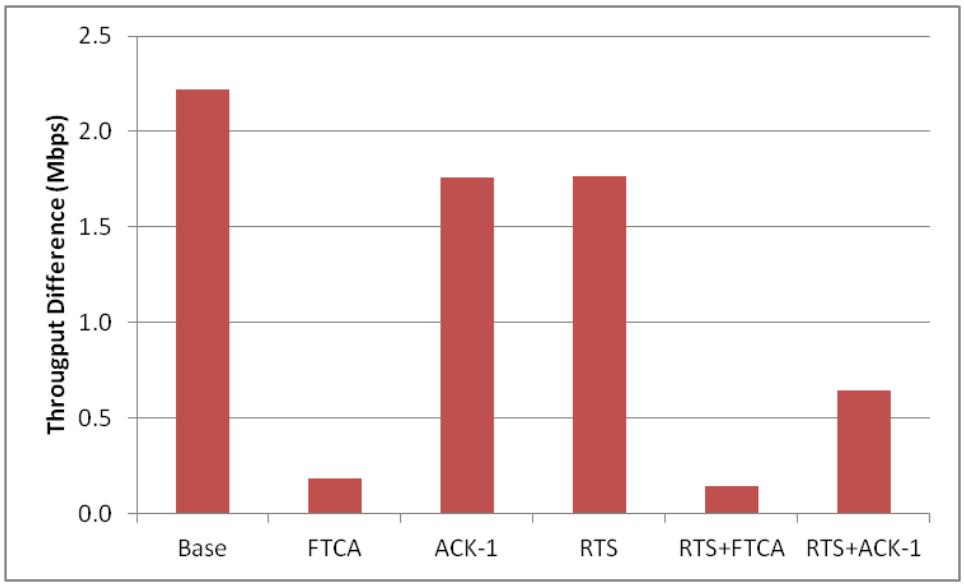


Figure 7.8: Maximum Observed Throughput Difference (Star Layout)

Figure 7.9 shows the average system throughput for each approach. For the base system, the total throughput is 13.57 Mbps. FTCA actually improves the throughput slightly to 13.76 Mbps. This result is important because it means that FTCA increases overall fairness without introducing significant throughput overhead. In contrast, other collision avoidance strategies such as RTS/CTS incur a heavy throughput penalty. The system throughput under RTS/CTS is 10.17 Mbps, which increases to 10.24 Mbps with FTCA. The ACK-1 strategy also has some overhead, since the extra ACKs take up some transmission opportunities. ACK-1 has a system throughput of 12.9 Mbps with basic access and 9.7 Mbps with RTS/CTS.

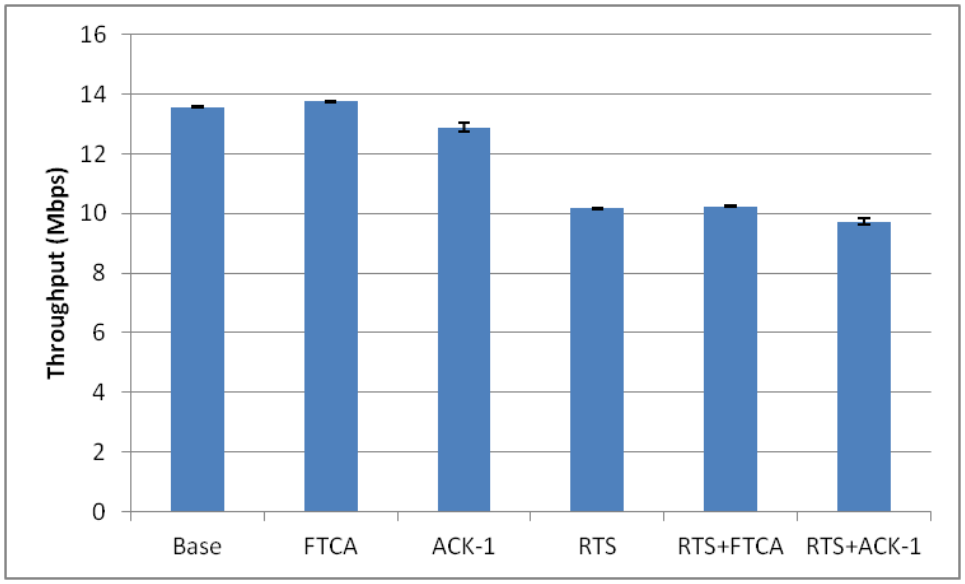


Figure 7.9: Average TCP System Throughput (Star Layout)

One of the main reasons why FTCA restores TCP fairness is that it reduces the probability that TCP ACKs will collide with data segments. Figure 7.10 shows the average error rate for TCP ACKs under each system. Without FTCA, 15.8% of TCP ACKs are lost. The primary cause of ACK loss is ACK-data collisions. Approximately 12.4% of the ACKs are corrupted due to collisions with data segments. Other ACKs have been lost due to noise and unavoidable issues such as transmitter-side collisions. The rate of ACK-ACK collisions is negligible (less than 0.01%). In this case, ACK-ACK collisions are rare because for TCP uploads, nearly all

ACKs originate from the AP, which cannot transmit multiple frames at once. ACKs are also very small compared to data segments, reducing the probability and extent of any overlap.

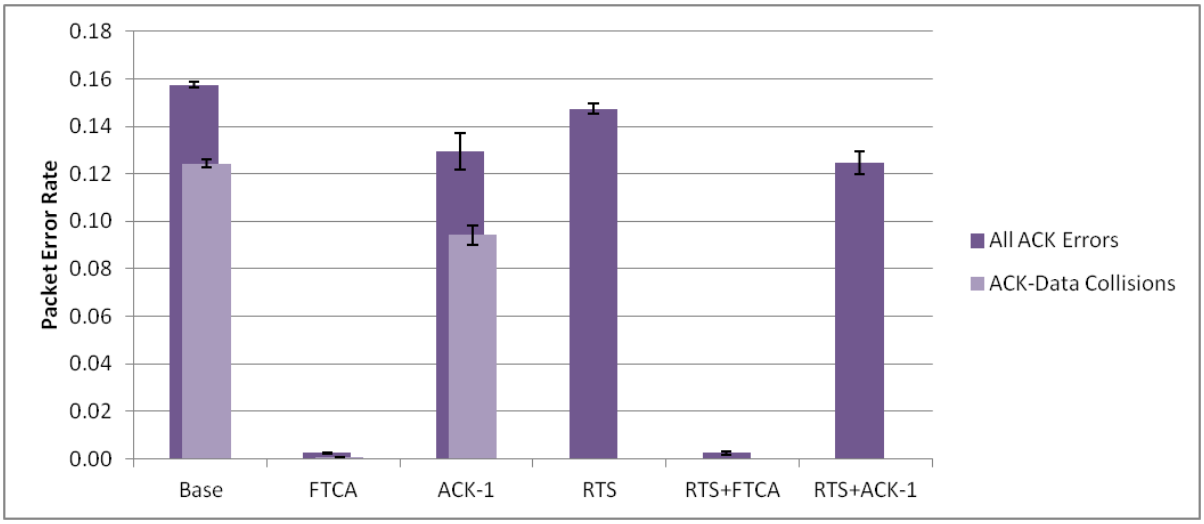


Figure 7.10: Average TCP ACK Error Rate (Star Layout)

With FTCA, the ACK error rate is reduced to 0.24% and the ACK-data collision rate is lowered to 0.06%. These packet error rates are much closer to what the TCP sender can tolerate from the channel.

RTS/CTS does not improve TCP fairness because it does not significantly reduce the ACK error rate. Although ACK-data collisions have been eliminated, they have been replaced with ACK-RTS collisions. Since the TCP ACKs are not protected with RTS/CTS due to their small size, they are contending with the RTS frames from STAs preparing to transmit TCP data segments. In theory, enabling RTS/CTS for TCP ACKs would prevent these collisions. However, the reduction in ACK error rate will be outweighed by the extra delay required for the RTS-CTS exchanges. Using FTCA with RTS/CTS prioritizes the TCP ACKs over the RTS frames. The ACK error rate is reduced from 14.7% to 0.3%.

Interestingly, the ACK-1 method does lead to a slight reduction in the ACK error rate compared to the base system, which may account for its small improvement in TCP fairness.

With basic access the error rate is 13%, and with RTS/CTS the error rate is 12.5%. Unfortunately, these error rates still break the core assumptions required for normal TCP operation.

A side effect of ACK error reduction is an increase in the data error rate. Figure 7.11 shows the average error rate for TCP data segments under each system with basic access. Collisions between two data segments account for most of these errors. The remainder of the errors come from random noise and transmitter-side collisions. Data-ACK collisions are rare, and even when they occur, the data packets are almost always captured successfully. Under the base system, the data error rate is 21.2% and the data-data collision rate is 13.4%. FTCA increases these values to 31.9% and 31.8%, respectively.

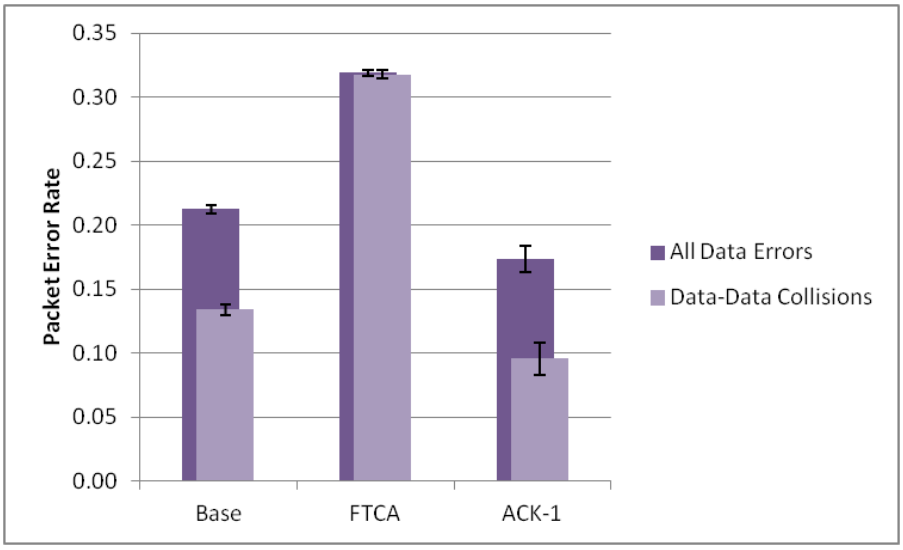


Figure 7.11: Average TCP Data Segment Error Rate (Star Layout)

The high data error rate does not have a negative impact on TCP fairness or throughput. Instead, it coincides with normal protocol operation. Since ACK prioritization reduces the time that it takes for TCP senders to receive an ACK, more senders are ready to transmit their next data segment at any given time. Therefore, more STAs are actively contending for the channel. The data error rates for FTCA are consistent with analytical models of collision probability in saturated networks [14]. TCP senders can tolerate this error because it occurs on the forward path of the connection.

7.1.2. *Grid Layout*

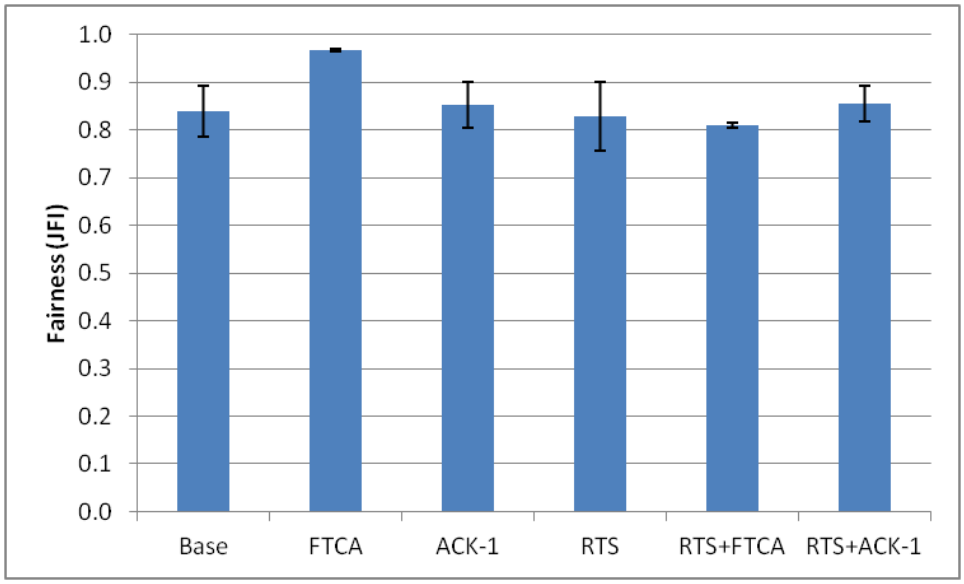


Figure 7.12: Average Fairness for Each System (Grid Layout)

In the grid layout, all systems experience a lower average fairness than in the star layout, as shown in Figure 7.12. For the base system, the average JFI is 0.84. Analysis of the individual simulation runs shows that the throughput is typically dominated by the STAs that are located closest to the AP. Some of this behavior is expected because transmissions from the closer STAs have a higher power at the receiver and thus a higher packet capture probability. However, there are cases where nodes that are equidistant from the AP still have extremely different throughput values due to improper TCP operation. For example, in Figure 7.13, STAs 2, 4, 5, and 7 are all located 5 meters from the AP. Node 7 has a throughput of 3.14 Mbps but Node 4 has a throughput of 0.97 Mbps.

The corresponding congestion window plot in Figure 7.14 confirms that the TCP congestion control algorithm is misbehaving. Node 7 experiences only one loss event near the beginning of the simulation. Afterwards, its congestion window increases without bound. The other nodes experience varying degrees of loss and cannot properly recover from them.

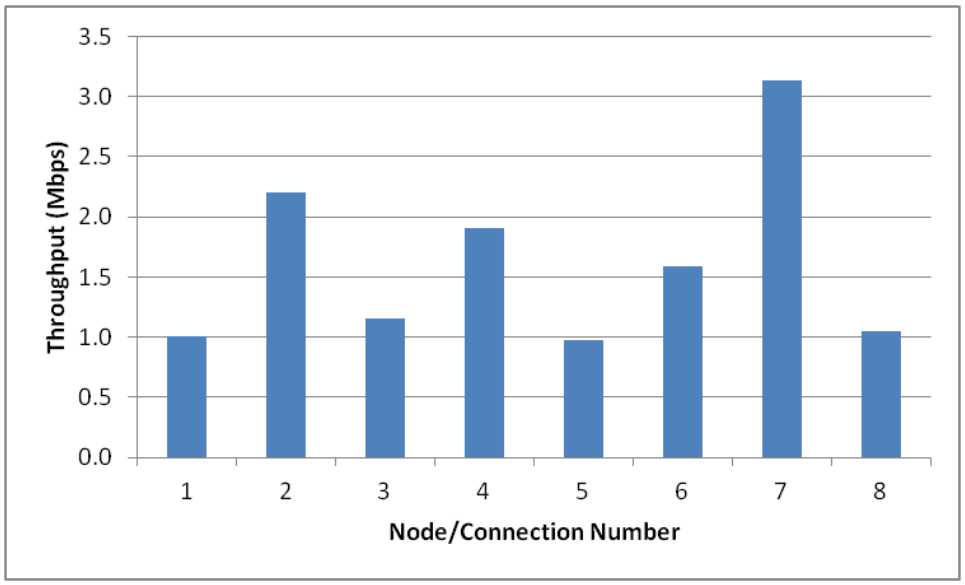


Figure 7.13: Per Node TCP Throughput (Base System, Grid Layout)

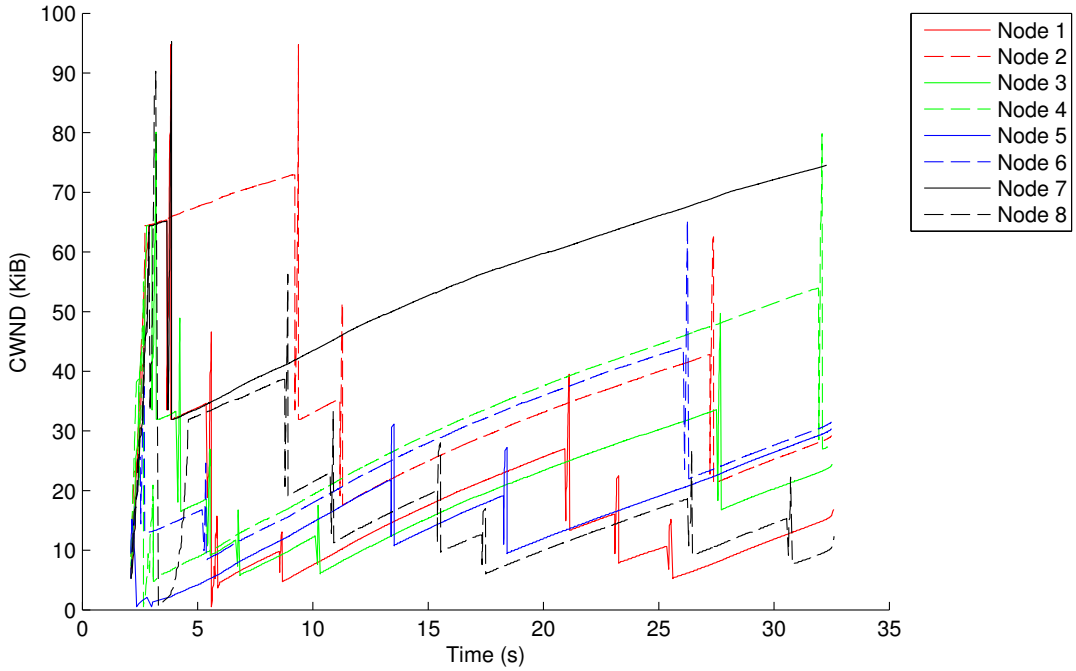


Figure 7.14: TCP Congestion Window (Base System, Grid Layout)

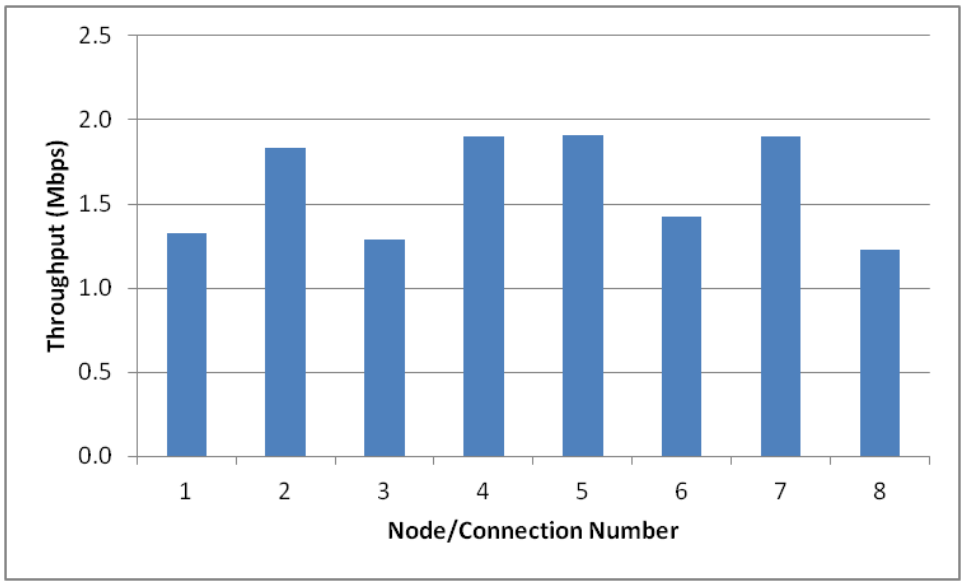


Figure 7.15: Per Node TCP Throughput (FTCA, Grid Layout)

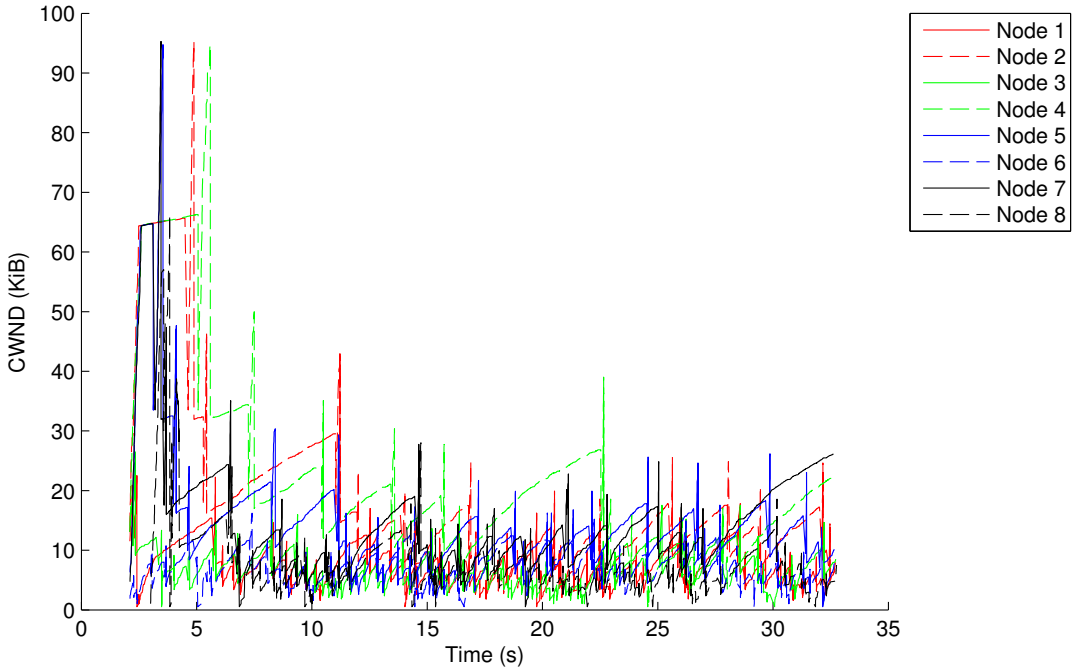


Figure 7.16: TCP Congestion Window (FTCA, Grid Layout)

With FTCA, there are still per-node throughput differences, but they are much less severe. Figure 7.15 depicts a typical simulation run where the strongest node has a throughput of 1.91 Mbps and the weakest node has a throughput of 1.23 Mbps. Groups of STAs that are the same distance from the AP have almost equal throughput values. The congestion window for each node, shown in Figure 7.16, converges properly. Overall, FTCA increases the average fairness to 0.97.

The ACK-1 strategy does not lead to a statistically significant improvement in fairness over the base system. The average fairness for ACK-1 is 0.85 but the JFI for individual simulation runs has varied wildly from 0.76 up to 0.92.

With RTS/CTS enabled, the average fairness is also not very different from that of the base system. It is interesting to note that unlike with the star layout, using FTCA alongside RTS/CTS in the grid layout does not improve fairness significantly. Due to the overhead of RTS/CTS and its limited impact on fairness, it is probably best to stay with the basic access mechanism unless there is a known issue with hidden nodes in the network.

Figure 7.17 shows the worst-case throughput difference observed for each system. The base system exhibits extremely inconsistent performance, with a maximum throughput difference of 3.0 Mbps. Similar problems occur for ACK-1 and RTS/CTS. The worst observed throughput difference for FTCA is 0.80 Mbps, a much more acceptable value.

Figure 7.18 shows the average system throughput and gives an indicator of protocol overhead. In the ideal star layout, FTCA has led to a slight increase in throughput, but in the grid layout, it causes the throughput to decline slightly. The base system has an average system throughput of 13.06 Mbps, while FTCA has an average system throughput of 12.80 Mbps. Therefore, it is apparent that FTCA will introduce some overhead in regular WLANs. Nevertheless, this throughput overhead is still small compared to the overhead associated with RTS/CTS, which has a system throughput of around 10.22 Mbps.

There is no significant difference between the star layout and the grid layout in terms of packet error and collision characteristics. The only notable trend is an increase in the overall

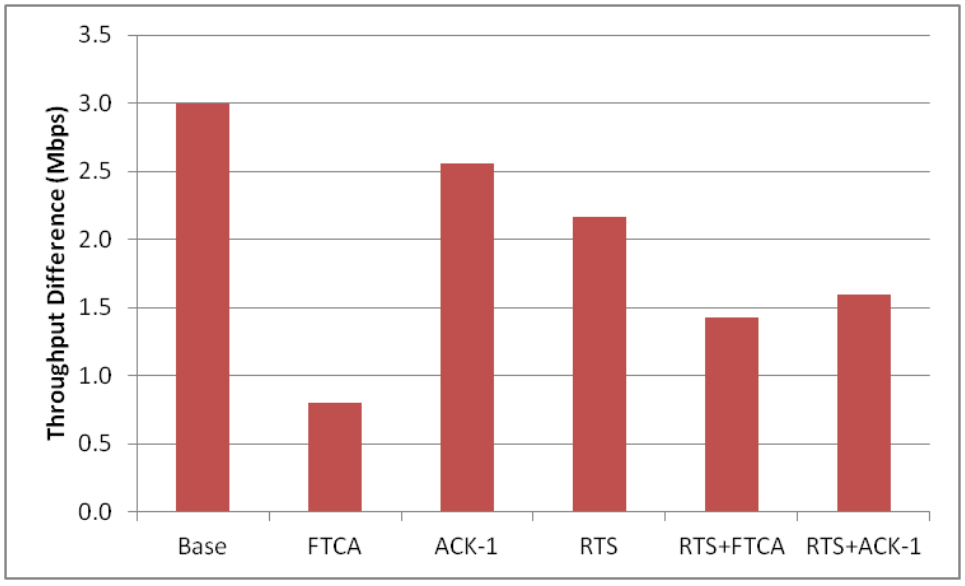


Figure 7.17: Maximum Observed Throughput Difference (Grid Layout)

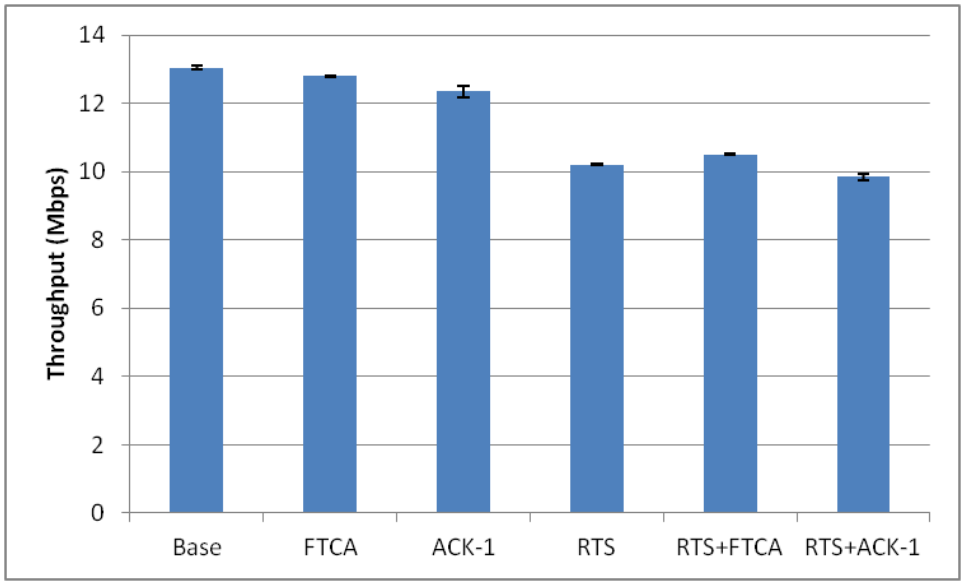


Figure 7.18: Average TCP System Throughput (Grid Layout)

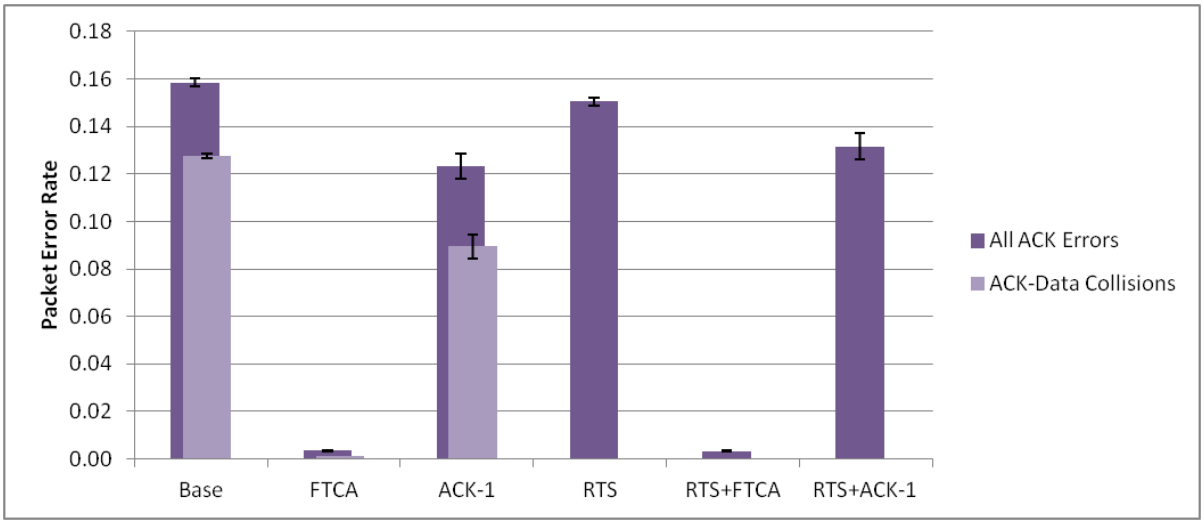


Figure 7.19: Average TCP ACK Error Rate (Grid Layout)

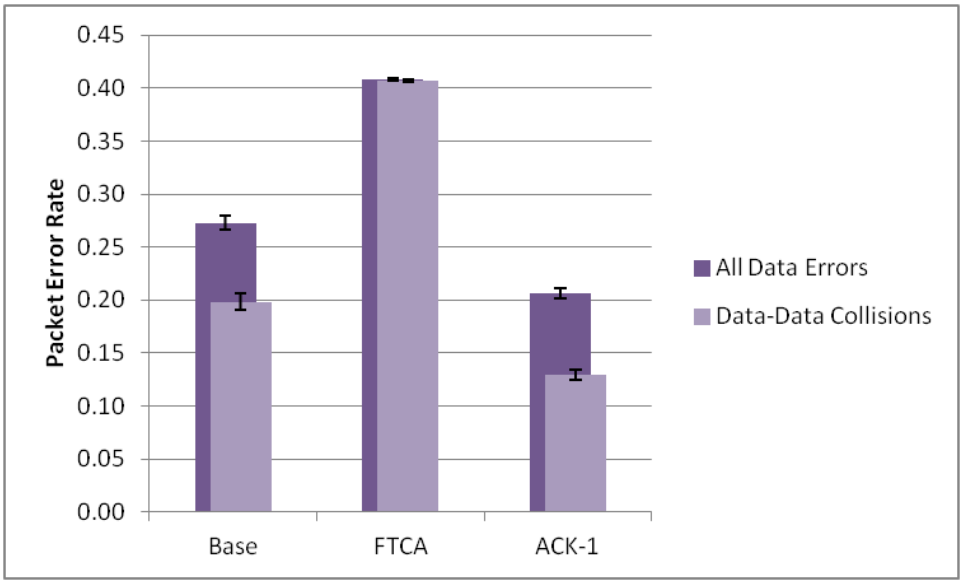


Figure 7.20: Average TCP Data Segment Error Rate (Grid Layout)

collision rate because the grid layout has 2 additional TCP senders compared to the star layout. For TCP ACKs, Figure 7.19 shows that the base system has a high error rate which is dominated by ACK-data collisions. ACK-1 reduces the error rate by a small amount, but FTCA almost entirely solves the collision problem. With FTCA, the ACK-data collision rate is 0.13% and the error rate is 0.35%. RTS/CTS also eliminates ACK-data collisions, but the ACK error rate is still high due to ACK-RTS collisions.

FTCA still increases the collision rate for TCP data segments, as indicated in Figure 7.20. The base system has a data error rate of 27.3% and a data-data collision rate of 19.9%. FTCA has a data error rate of 40.8% and a data-data collision rate of 40.7%. Fortunately, the collision probability for FTCA still follows analytical models for 8-node WLANs [14], and the high collision rate does not have a negative impact on TCP performance.

7.1.3. Random Disc Layout

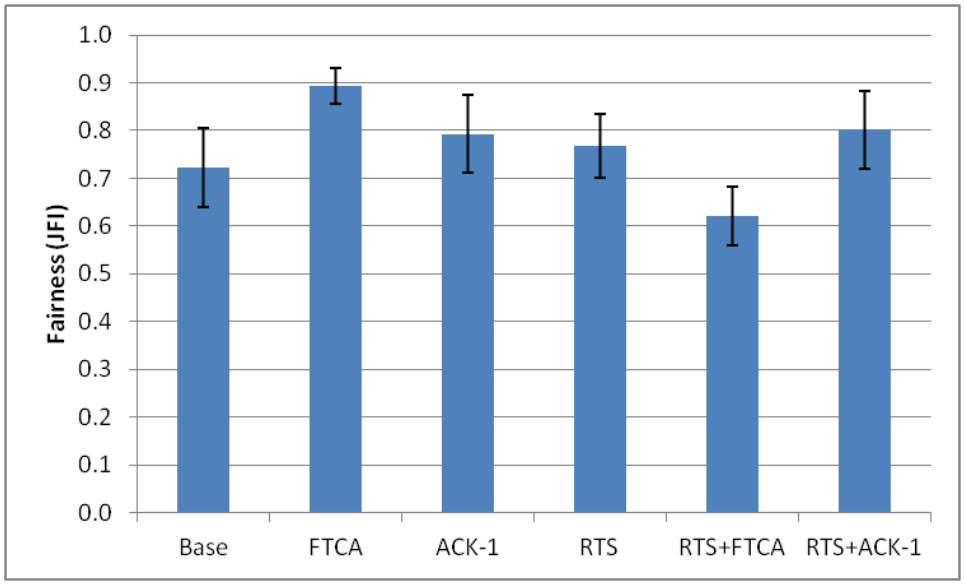


Figure 7.21: Average Fairness for Each System (Random Disc Layout)

The random disc layout exhibits the lowest overall fairness compared to the highly symmetric star and grid layouts. Distance-based packet capture unfairness is a major problem

which FTCA cannot completely address. Still, for the base system, there are cases where nodes that are close to the AP have unexpectedly low throughput due to improper TCP behavior. FTCA cannot restore perfect fairness, but it does correct these problems and lead to a significant improvement over the base system.

Figure 7.21 shows the average fairness for each system. The data are less precise than those for the previous simulations due to the randomness involved here. The base system has an average JFI of 0.72. FTCA has an average JFI of 0.89, and its improvement over the base system is statistically significant. The other systems are harder to compare. ACK-1 and RTS/CTS do not significantly change the fairness. Using FTCA together with RTS/CTS might cause a slight decrease in fairness, but more data is required to confirm this.

Figure 7.22 shows the maximum recorded throughput difference for each system. The worst-case behavior of FTCA is less than that of the base system, as expected. ACK-1 and RTS/CTS also have satisfactory worst-case behavior. However, FTCA is subject to occasional fairness problems when used alongside RTS/CTS. Based on these simulation results, as well as the results for the grid layout, it is apparent that RTS/CTS does not have a positive effect on TCP fairness.

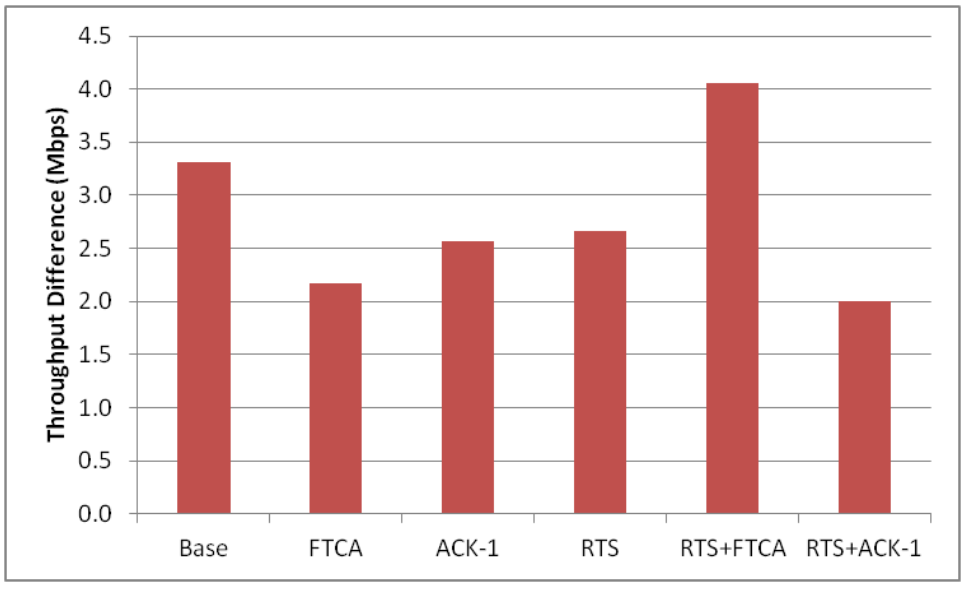


Figure 7.22: Maximum Observed Throughput Difference (Random Disc Layout)

The average system throughput is shown in Figure 7.23. The results are mostly similar to those for the star and grid layouts. FTCA introduces a small amount of throughput overhead, while RTS/CTS and ACK-1 introduce larger degrees of overhead.

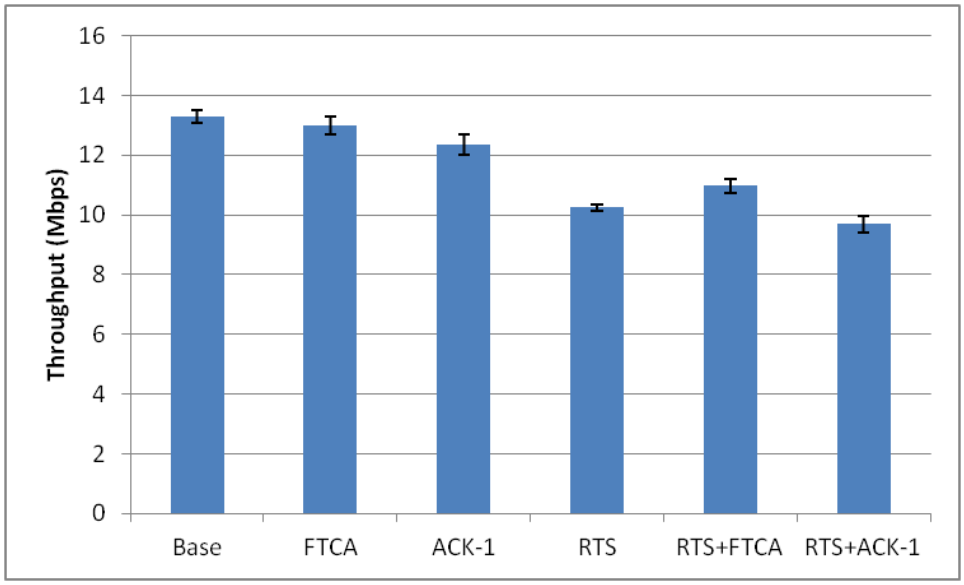


Figure 7.23: Average TCP System Throughput (Random Disc Layout)

Figure 7.24 shows the TCP ACK error rates and frame collision rates. Unlike in the star and grid layouts, FTCA does not perfectly eliminate ACK-data collisions. Under the base system, the ACK-data collision rate is 14.5% and the total ACK error rate is 18%. With FTCA, the collision rate is 0.9% and the total error rate is 2%. These error rates are still high enough to introduce some TCP unfairness, though at a much lower degree than in the base system.

Error statistics for data segments are summarized in Figure 7.25. The results here are very similar to those for the grid layout. Data error rates in the base system are around 24%. FTCA increases the error rate to around 40%, but the frame collision rate is not abnormal.

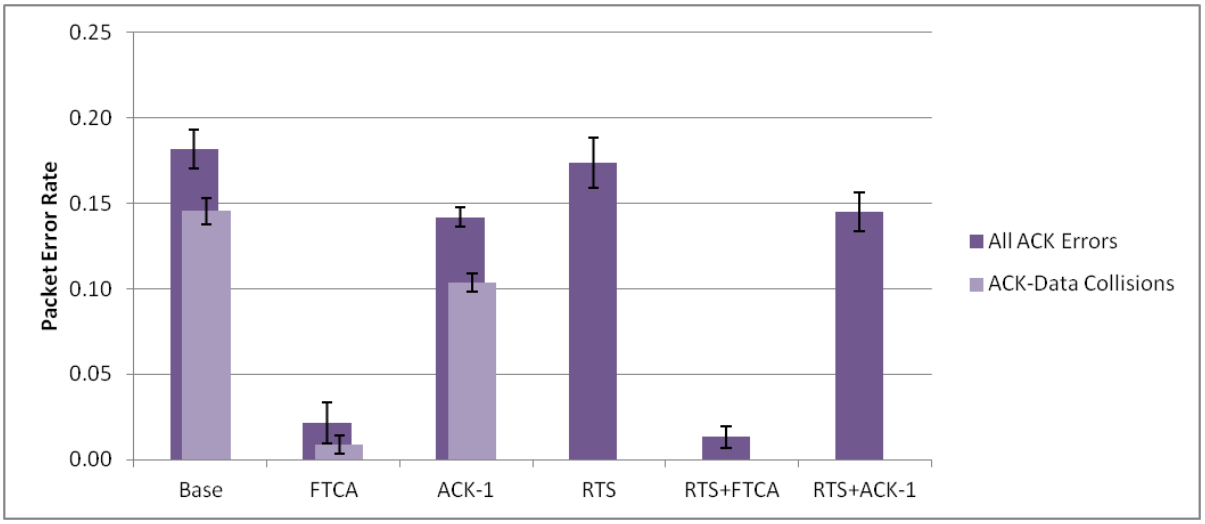


Figure 7.24: Average TCP ACK Error Rate (Random Disc Layout)

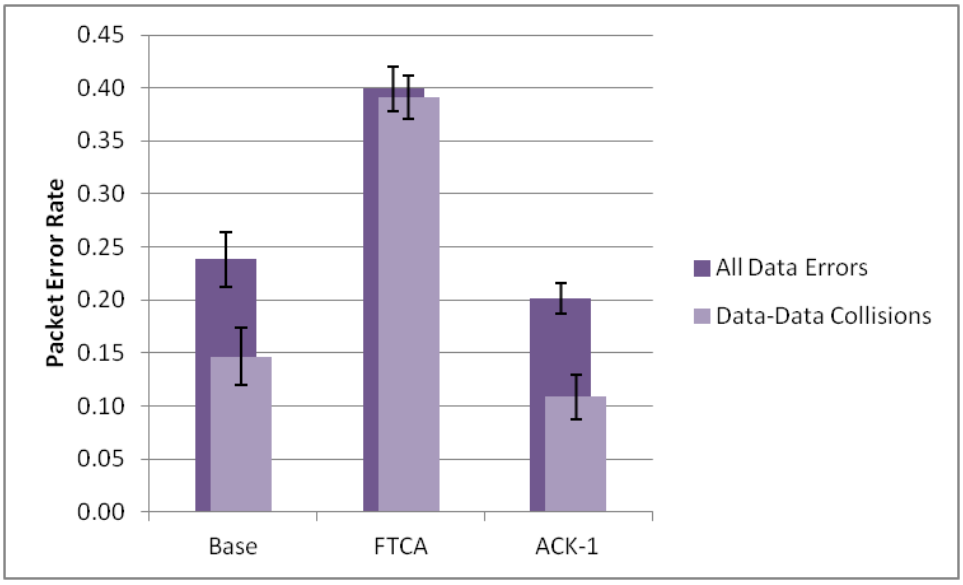


Figure 7.25: Average TCP Data Segment Error Rate (Random Disc Layout)

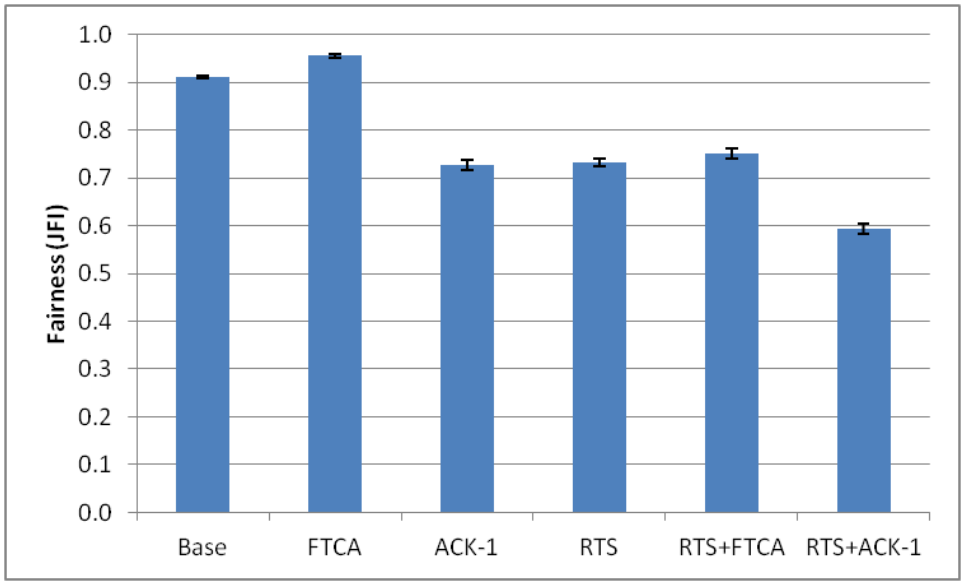


Figure 7.26: Average Fairness for Each System (TCP and UDP, Grid)

FTCA also improves the performance of TCP connections when there are other UDP senders in the network. In the base system, TCP data segments, TCP ACKs, and UDP packets all contend for transmission opportunities. Since UDP has no rate control mechanisms, it can crowd out the TCP flows. FTCA does not stop UDP packets from interfering with TCP data segments, but it will prevent them from colliding with TCP ACKs and causing excess retransmission timeouts.

We set up the same grid layout used in the previous simulations, but with STAs 5–8 establishing UDP upload connections instead of TCP connections. Each UDP sender transmits a continuous stream of 1-KiB packets at 1 ms intervals. The UDP packets have the same QoS priority as the TCP data segments (AC_BE).

Figure 7.26 compares the average fairness for each system. The base system has an average JFI of 0.911. FTCA improves the fairness to 0.956. Surprisingly, the ACK-1 method has an average JFI of only 0.73. ACK-1 has performed well in the previous grid setup where there are only TCP connections. In this scenario, it seems that the extra ACKs are placing too

much strain on the network and are worsening the already high levels of congestion created by introducing the UDP flows. As a result, ACK-1 may not be suitable for use in real-world networks where there are many different types of traffic that may or may not have flow control.

RTS/CTS also leads to poor levels of fairness. The base system with RTS/CTS has an average JFI of 0.73. FTCA slightly improves the average JFI to 0.75. The overall JFI drops to 0.59 when ACK-1 is enabled. RTS/CTS should perform better in networks with hidden nodes, but here it is not a good solution.

Figure 7.27 shows the worst-case throughput difference for each system. The maximum throughput differences across all systems are already somewhat large since the UDP senders achieve higher overall rates compared to the TCP senders. Still, FTCA has better worst-case behavior compared to the other approaches.

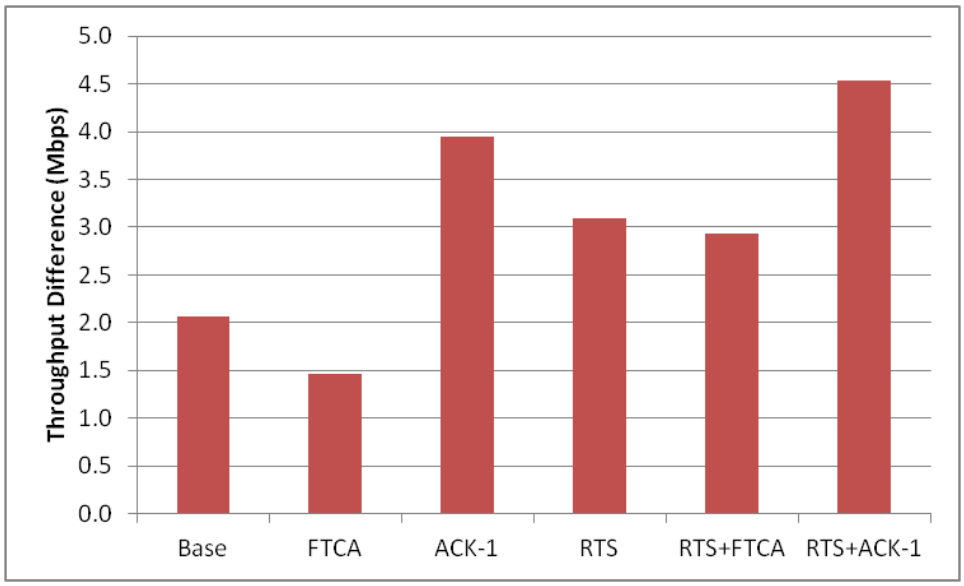


Figure 7.27: Maximum Observed Throughput Difference (TCP and UDP, Grid)

The average system throughput for each approach is shown in Figure 7.28. UDP traffic accounts for a large portion of the throughput. The base system has a throughput of 16.7 Mbps. FTCA introduces a minimal amount of overhead, causing the throughput to drop to 16.5 Mbps. ACK-1 causes the throughput to increase to 17.7 Mbps, but the network is dominated by a small

subset of nodes. RTS/CTS introduces a significant amount of throughput overhead compared to basic access.

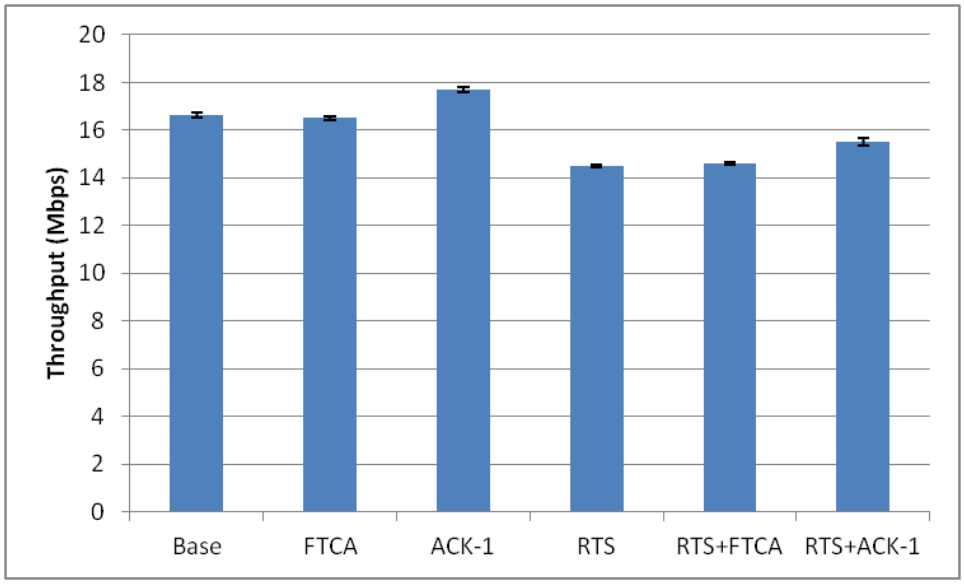


Figure 7.28: Average System Throughput (TCP and UDP, Grid)

The packet error statistics are summarized in Figures 7.29 and 7.30. FTCA significantly reduces the number of collisions between TCP ACKs and TCP or UDP data packets. ACK-1 places too much load on the network and actually increases the ACK-data collision rate. Once again, RTS/CTS eliminates ACK-data collisions but replaces them with ACK-RTS collisions that are still harmful to the TCP senders. Data-data collision rates are high in all cases when basic access is used. However, the collision rates still conform to analytical models [14], and do not have a significant effect on fairness or throughput.

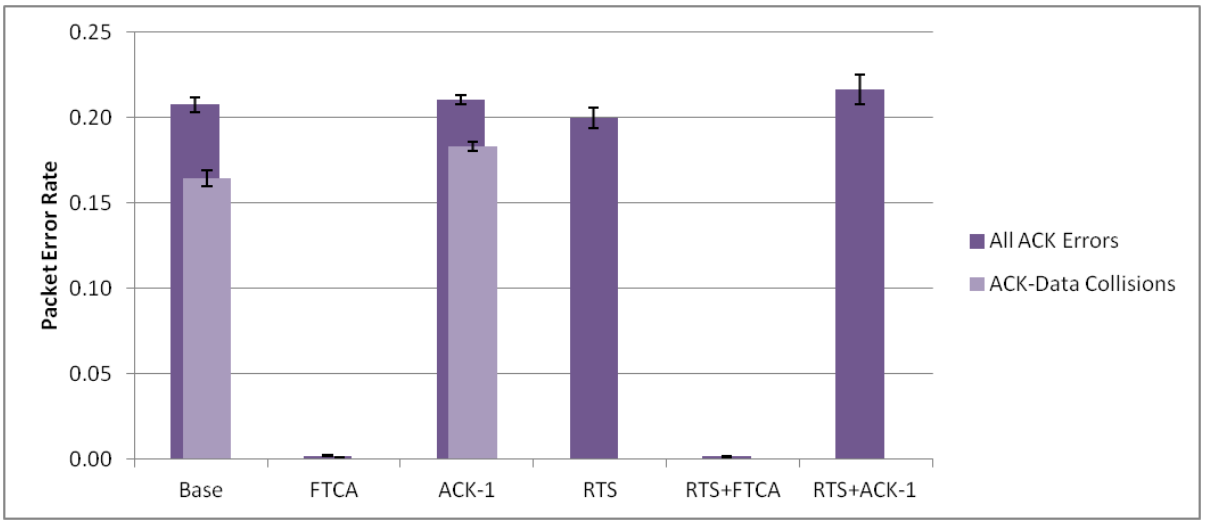


Figure 7.29: Average TCP ACK Error Rate (TCP and UDP, Grid)

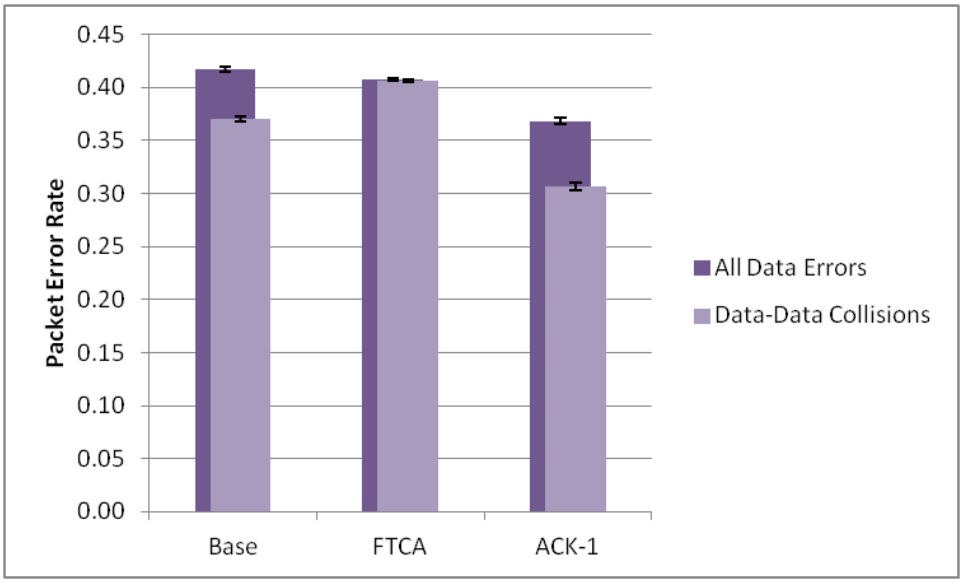


Figure 7.30: Average TCP/UDP Data Error Rate (TCP and UDP, Grid)

8. Conclusions and Recommendations

TCP exhibits poor performance in wireless networks due to the high probability of random channel errors, frame collisions, packet capture, and delays. Packet loss events in wireless networks can trigger improper behavior in TCP congestion control algorithms, which leads to unfair throughput allocation across parallel connections. We have devised a new MAC-layer solution, Fair TCP Channel Access, to address these problems. FTCA prioritizes TCP ACKs at the MAC layer using standard QoS mechanisms in 802.11. Unlike typical MAC-layer and cross-layer approaches for wireless TCP, FTCA requires no hardware modifications and is easy to deploy in existing WLANs. Since FTCA works independently from the transport layer, it is compatible with almost any TCP implementation, including NewReno and CUBIC.

Simulations in NS-3 show that FTCA consistently improves TCP fairness in a variety of network layouts. It reduces collisions between TCP ACKs and TCP data segments, and restores proper congestion control operation. It also improves TCP performance in situations where there are contending UDP flows. The throughput overhead of FTCA is minimal. FTCA may exhibit some performance issues when used alongside RTS/CTS, so it should be used with basic access unless there are specific problems with hidden nodes in the network.

Overall, FTCA demonstrates an effective application of QoS systems in the 802.11 standard. Prioritizing TCP control packets at the wireless MAC layer protects them from extraneous losses and improves network reliability. Further research should be conducted to test whether the concepts developed here may be applied to other network protocols that also rely on control packets to establish and regulate connections. QoS will become an increasingly important component of modern wireless networks as they continue to expand.

REFERENCES

[1] A. Hepburn. (2013, Oct.) Infographic: 2013 mobile growth statistics. [Online]. Available: http://www.digitalbuzzblog.com/infographic-2013-mobile-growth-statistics/

[2] M. Fidelman. (2012, May) The latest infographics: Mobile business statistics for 2012. Forbes. [Online]. Available: http://www.forbes.com/sites/markfidelman/2012/05/02/the-latest-infographics-mobile-business-statistics-for-2012/

[3] Pew Research Center. (2013, May) Device ownership. [Online]. Available: http://www.pewinternet.org/Static-Pages/Trend-Data-%28Adults%29/Device-Ownership.aspx

[4] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std. 802.11, 2012.

[5] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 5th ed. New York: Addison-Wesley, 2010.

[6] "Transmission control protocol," RFC 793, Sep. 1981. [Online]. Available: http://tools.ietf.org/html/rfc793

[7] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," RFC 2018, Oct. 1996. [Online]. Available: http://tools.ietf.org/html/rfc2018

[8] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's retransmission timer," RFC 6298, Jun. 2011. [Online]. Available: http://tools.ietf.org/html/rfc6298

[9] *IEEE Standard for Ethernet*, IEEE Std. 802.3, 2012.

[10] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," RFC 5681, Sep. 2009. [Online]. Available: http://tools.ietf.org/html/rfc5681

[11] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The NewReno modification to TCP's fast recovery algorithm," RFC 6582, Apr. 2012. [Online]. Available: http://tools.ietf.org/html/rfc6582

[12] Y. Tian, K. Xu, and N. Ansari, "TCP in wireless environments: Problems and solutions," *IEEE Commun. Mag.*, vol. 43, no. 3, pp. S27–S32, Mar. 2005.

[13] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 3, pp. 535–547, Mar. 2000.

[14] H. L. Vu and T. Sakurai, "Collision probability in saturated IEEE 802.11 networks," in *Proc. ATNAC '06*, Dec. 2006.

[15] J. Lee, W. Kim, S.-J. Lee, D. Jo, J. Ryu, T. Kwon, and Y. Choi, "An experimental study on the capture effect in 802.11a networks," in *Proc. ACM WiNTECH '07*, Sep. 2007, pp. 19–26.

[16] C. Ware, J. Chicharo, and T. Wysocki, "Simulation of capture behavior in IEEE 802.11 radio modems," in *Proc. IEEE VTC '01*, vol. 3, Oct. 2001, pp. 1393–1397.

[17] A. Goldsmith, *Wireless Communications*. New York: Cambridge Univ. Press, 2005.

[18] A. Thangaraj, Q.-A. Zeng, and X. Li, "Performance analysis of the IEEE 802.11e wireless networks with TCP ACK prioritization," in *Proc. IEEE ICCCN '08*, Aug. 2008, pp. 1–6.

[19] D. Leith and P. Clifford, "Using the 802.11e EDCF to achieve TCP upload fairness over WLAN links," in *Proc. IEEE WIOPT '05*, Apr. 2005, pp. 109–118.

[20] The MadWifi project. [Online]. Available: http://madwifi-project.org/

[21] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 756–769, Dec. 1997.

[22] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[23] C. P. Fu and S. C. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 2, pp. 216–228, Feb. 2003.

[24] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. ACM SIGMOBILE '01*, Jul. 2001, pp. 287–297.

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

[26] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *Proc. IEEE INFOCOM '04*, vol. 4, Mar. 2004, pp. 2514–2524.

[27] A. Nyandoro, L. Libman, , and M. Hassan, "Service differentiation using the capture effect in 802.11 wireless LANs," *IEEE Trans. Wireless Commun.*, vol. 6, no. 8, pp. 2961–2971, Aug. 2007.

[28] H. Wu, Y. Peng, K. Long, S. Cheng, and J. Ma, "Performance of reliable transport protocol over IEEE 802.11 wireless LAN: Analysis and enhancement," in *Proc. IEEE INFOCOM '02*, vol. 2, 2002, pp. 599–607.

[29] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP," RFC 3168, Sep. 2001. [Online]. Available: http://tools.ietf.org/html/rfc3168

[30] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for wireless IP communications," *IEEE J. Sel. Areas Commun.*, vol. 22, no. 4, pp. 747–756, May 2004.

[31] S. Park, X. Shen, J. W. Mark, and L. Cai, "A two-phase loss differentiation algorithm for improving TFRC performance in IEEE 802.11 WLANs," *IEEE Trans. Wireless Commun.*, vol. 6, no. 11, pp. 4164–4175, Nov. 2007.

[32] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. ACM SIGCOMM '00*, 2000, pp. 43–56.

[33] ——, "TCP friendly rate control (TFRC): Protocol specification," RFC 5348, Sep. 2008. [Online]. Available: http://tools.ietf.org/html/rfc5348

[34] Q. Pang, S. C. Liew, and V. C. M. Leung, "Performance improvement of 802.11 wireless access network with TCP ACK agent and auto-zoom backoff algorithm," in *Proc. IEEE VTC '05*, vol. 3, May 2005, pp. 2046–2050.

[35] G. Xylomenos, G. C. Polyzos, P. Mähönen, and M. Saaranen, "TCP perfomance issues over wireless links," *IEEE Commun. Mag.*, vol. 39, no. 4, pp. 52–58, Apr. 2001.

[36] *Medium Access Control (MAC) Quality of Service Enhancements*, IEEE Std. 802.11e, 2005.

[37] S. Ganu, K. Ramachandran, M. Gruteser, I. Seskar, and J. Deng, "Methods for restoring MAC layer fairness in IEEE 802.11 networks with physical layer capture," in *Proc. ACM REALMAN '06*, May 2006, pp. 7–14.

[38] N. Choi, J. Ryu, Y. Seok, T. Kwon, and Y. Choi, "Optimizing aggregate throughput of upstream TCP flows over IEEE 802.11 wireless LANs," in *Proc. IEEE PIMRC '07*, Sep. 2007, pp. 1–5.

[39] ns-3. [Online]. Available: http://www.nsnam.org/

[40] Wireshark. [Online]. Available: http://www.wireshark.org/

[41] J. Mittag and S. Papanastasiou. PhySimWiFi for NS-3. DSN Research Group, Karlsruhe Inst. of Tech. [Online]. Available: http://dsn.tm.kit.edu/english/ns3-physim.php

[42] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," Digital Equipment Corp., Tech. Rep. DEC-TR-301, Sep. 1984.