Networking 2013 1569709409

Fast Wireless Protocol: A Network Stack Design for Wireless Transmission

Daniel Havey
Department of Computer Science
University of California
Santa Barbara, California 93106-5110
Email: dhavey@yahoo.com

Kevin Almeroth
Department of Computer Science
University of California
Santa Barbara, California 93106-5110
Email: almeroth@cs.ucsb.edu

***Abstract*—The robust and efficient streaming of video over wireless networks poses serious challenges. Inherent instabilities in the wireless medium lead to large, highly variable delay, throughput variations, and data loss. To cope with this problem the wireless MAC has incorporated many features such as reliability and aggregation. These features allow transport protocols such as TCP to work by hiding the frame losses from layers above the MAC. The problem is that in doing so they create overhead leading to loss of throughput. In this paper, we investigate a receiver side network stack design that allows these features to be switched off in the MAC. We instead provide reliability and aggregation features at the session layer. Our network stack design reduces the overhead cost significantly over MAC frame aggregation and increases throughput. We demonstrate the throughput advantages of our approach in testbed and emulation and analyze the effects of overhead created by our system.**

I. Introduction

There has been much work in attempting to deliver video over wireless and in particular, video over HTTP/TCP over wireless. What makes this an important, relevant, and (again) a timely topic is the evolution in wireless technology and the demand for multimedia at high resolution. The traditional approach to dealing with the lossy wireless medium is to push more features into the wireless MAC protocol. For example the wireless MAC provides reliability and frame aggregation services. The reason for reliability is obvious. The missing frames must be replaced in order to deliver complete data to the user. However, reliability is not free. It costs overhead leading to loss of throughput.

The source of the overhead is the need for the 802.11 MAC to acknowledge frames. Radio time spent transmitting an ACK is radio time that could have been used to transmit data. The frame aggregation system attempts to address this problem by transmitting many frames in an aggregate and then a single acknowledgment for the entire aggregate. However, the wireless MAC is inherently limited in its ability to provide reliability and efficiency because of its position in the stack. Since the MAC is below the transport layer, it must replace the missing frames before the TCP Retransmit Time Out (RTO). Otherwise, TCP will trigger a false congestion event. In fact, the wireless MAC must also beat the MAC reorder buffer timeout (typically 40 ms). If the MAC has not replaced the missing frames before this deadline, the frames will be released without the replacement frame. The missing segment number that was inside of the lost frame will cause a false TCP congestion event. The implication of this is that the data flow will be reduced.

Our approach to solving this problem is novel because, rather than solving the reliability problem at the MAC layer and incurring the associated overhead, we provide reliability at the session layer in *socket.c*. It is better to keep the data flowing through the wireless MAC, IP, and transport layers and retransmit in the session layer closer to the application. The intuition driving this approach is that, for most multimedia streams, there is a large jitter buffer that can absorb the reliability overhead much better than the wireless MAC is capable of. In fact, Video on Demand (VoD) normally has a jitter buffer of 30 seconds, and live video a buffer of 8 seconds.[1] This large difference in buffer size allows us to reduce the ACK count and the overhead dramatically.

In our Fast Wireless Protocol (FWP), the retransmission system is in the session layer above the transport layer. We reduce the acknowledgment overhead to near zero because we are not restricted by the transport or the MAC reorder buffer. The tradeoff with our FWP approach is that the reliability mechanism is implemented over HTTP. This places additional load on the network because some of the data has to be retransmitted from the server over the network. We weigh the benefits of our solution in terms of increased throughput from taking advantage of all transmission opportunities against the drawback of increased overhead in the network in our evaluation.

For the sake of deployability our FWP system is implemented in the receiver stack only. There are no changes to any device in the network and no special data encodings at the server. Since FWP is implemented without crossing administrative domains, it can be deployed without the need to secure the cooperation of other administrative entities.

In this paper our contributions are to analyze the overhead caused by wireless 802.11-style MAC protocols, and compare this to our FWP network stack. We build a prototype system in testbed and evaluate FWP in terms of throughput fairness and overhead. Additional contributions are TCP Centric, a transport system that distinguishes congestion loss from wireless bit

[1]We focus on video delivery, for example entertainment. Other kinds of video delivery, for example, video conferencing with its more stringent delay requirements is beyond the scope of this work. However, given that video delivery has grown to dominate Internet traffic [8], [19], [9], this is a reasonable tradeoff

errors, and an HTTP retransmission system that exists in the session layer.

The remainder of this paper is organized as follows. In Section II we review relevant work in this area. In Section III we provide a background of 802.11 frame aggregation systems and closely examine the problem with wireless transmission of data. In Section IV we describe the operating details of our FWP system. In Section V we describe the implementation details of our prototype used for experimentation. In Section VI we perform experiments and analyze our solution. Finally in Section VII we conclude and discuss our future work.

II. Related Work

Research solutions addressing the efficient utilization of bandwidth in wireless networks fall into a few general categories, the most popular of these being the new generation of DASH video streaming applications. DASH solutions adapt the quality of the video stream delivered in order to match the bit rate required for media playout to the available throughput provided by TCP [25]. Examples of this type of solution are Adobe HTTP Dynamic Streaming server[2], Adobe OSMF[3], Microsoft's IIS Smooth Streaming player, [4] and the proprietary protocols used by Netflix, Move Networks[5], and others.

Further work in this area includes an evaluation of Akamai HD Network for Dynamic Streaming of Flash over HTTP provided by Cicco et al. [6]. Of particular interest in this study are the experiments showing how the player reacts to sharing the bottleneck router with a greedy TCP flow. In addition, a study by Akhshabi et al. compare the behavior of Microsoft Smooth Streaming, Adobe OSMF, and the Netflix player [2]. These solutions prevent video playout from stopping due to a lack of throughput. However, they do this by reducing the quality of the video stream, not by increasing the efficiency of the network stack.

Before the adoption of HTTP/TCP by video streaming service providers, much research was done to investigate alternatives to the TCP protocol. Prime examples of this type of solution are the Datagram Congestion Control Protocol (DCCP) [16], [17], and RTP/UDP [23], [22]. A cross-layer example of a UDP based protocol is provided in Krishnamachari et al. [30]. Although these types of solutions provide demonstrable results, they were not adopted for general use. It is a common practice for Internet service providers to use firewalling to drop UDP packets making non-HTTP solutions unacceptable. However, some streaming applications such as Skype[6] will attempt to find an RTP/UDP connection before falling back to an HTTP/TCP stream.

Many adaptations to TCP's congestion control algorithms have been studied in [7], [10], [27], [15], [24], [20]. However, these solutions require changes to the TCP sender as well as the receiver. It has proven difficult to convince large content providers to change their network stacks. Our solution works within the receiver kernel and can be deployed without such large scale changes.

[2]http://www.adobe.com/products/
[3]http://www.osmf.org/
[4]http://www.iis.net/media/experiencesmoothstreaming
[5]http://www.movenetworks.com/
[6]www.skype.com

Equation based TCP friendly solutions were studied in, [4], [11]. However, these equations are only fair within a factor of 2. This is inadequate for TCP fairness. TCP fairness and benchmarks have been studied in [28], [29], [5], [1], [21]. We used these studies to help generate our fairness metric.

Multipath TCP has been studied in, [31], [12]. These types of solutions attempt to increase throughput by using multiple paths through the Internet. Split TCP solutions separate channel condition loss from congestion loss thus increasing TCP throughput in lossy networks [14]. However, these solutions do not address the overhead problems in the wireless network stack.

Erasure coding and it's use with TCP have been studied by Luby, and Mitzenmacher et al., [18], [26]. TCP/NC applies network coding in a shim layer between the TCP and IP layers. This approach is powerful, but it's positioning in the network stack requires a large overhead ($5n+7$ where n is the number of packets involved in a linear combination). This overhead is caused because of the variable size of TCP packets. Our solution works at the Session layer where fixed size blocks can be used thus avoiding almost all of this overhead. In addition, TCP/NC requires sender side kernel changes making it as difficult to deploy.

The ossification of the TCP protocol has been studied and it has been found that TCP modifications that relay on TCP options have difficulty passing through proxies such as those found in many wireless networks, [13], [3]. Our receiver side only architecture avoids this problem

All of these solutions, although effective in their own right, solve fundamentally different problems than our FWP system. None of these solutions effectively address the overhead problem encountered in 802.11 wireless MAC.

III. Background

The 802.11 retransmission scheme causes a tremendous amount of overhead at higher bit rates. This is because the radio header, the Short Interframe Spacing (SIFS) wait times, and the ACK transmission take up a significant amount of radio time ($2 * SIFS + ACK$). In order to reduce this overhead the 802.11n standard provides for two frame aggregation schemes. The Aggregate Mac Service Data Unit (A-MSDU) system, and the Aggregate Mac Protocol Data Unit (A-MPDU) system. in this section we describe each of these aggregation techniques and then summarize the overhead of 802.11 reliability.

A. A-MSDU frame aggregation

Frame aggregation systems combine MAC data units, either service or protocol, into an aggregate with a single header. This significantly reduces overhead. The A-MSDU aggregation system combines Mac Service Data Units (MSDU) into 7935 byte aggregates with one MAC header and the payload protected by a single CRC. The A-MSDU frame aggregation system is very efficient in relatively error free channels. However in error prone channels it suffers. The single CRC error protection does not allow the decoding of bits that were received correctly and the entire aggregate must be retransmitted.

Figure 1 demonstrates the A-MSDU aggregation system. In Time Series A, frame 2 is lost. There is no per packet Frame

Check Sequence (FCS) so the entire aggregate is undecodable and nothing is received. The entire transmission sequence is overhead. In Time Series B, all the frames are received correctly and the overhead to data ratio is good. The A-MSDU frame aggregation suffers greatly from error prone conditions.

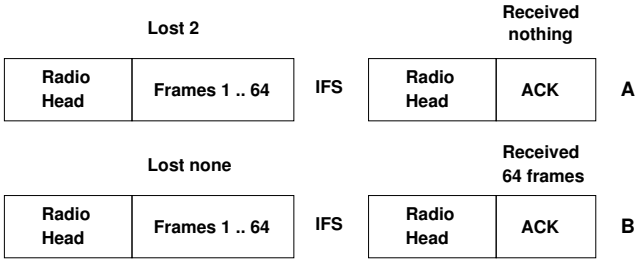

Fig. 1. A-MSDU Frame Aggregation

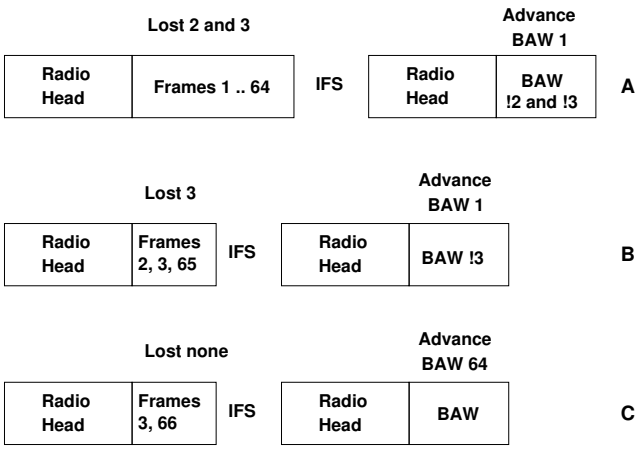

Fig. 2. A-MPDU Block ACK Window Advance

B. A-MPDU frame aggregation

The A-MPDU system has an error correction protocol to cope with lost frames individually rather than in aggregate. In the A-MPDU system up to 64 frames are aggregated into a single A-MPDU with a single radio header. A checksum of 4 bytes is provided for each frame. With these checksum bytes the receiver can determine which packets were received correctly then generate a bit map called a Block ACK (BA) requesting the missing frames. The Block ACK Window (BAW) sliding window system is implemented to retransmit the requested missing frames.

Figure 2 demonstrates the BAW advance mechanism. In Time Series A, the aggregate is full and 64 frames are transmitted. Frames 2 and 3 were not received correctly and the BA indicates that they should be retransmitted. The BAW sliding window cannot be advanced past the first missing frame until a BA indicates that frame has been successfully received. The BAW is advanced one frame and this queue has stalled.

In Time Series B, a smaller aggregate is transmitted. Only one new frame (number 65) can be added to the A-MPDU because the BAW was only advanced by one. The two lost frames can also be added for a total of 3 frames. The aggregate is now very small compared to the 64 frames that could have been transmitted. In this example frame 3 is again lost leading to another small aggregate transmission in Time Series C.

The A-MPDU system is reasonably efficient even in error prone conditions. However, its sliding window retransmission system can sometimes stall resulting in the transmission of a small aggregate. This becomes a problem when there are other stations contending for air time. The station has used its transmission opportunity to send the small aggregate consisting of only a few frames. Now it must wait until it wins the contention for air time to transmit again.

C. 802.11 Reliability

Reliability in the 802.11 MAC is provided by an acknowledgment system. This acknowledgment system creates a great deal of overhead especially at higher speeds. The frame aggregation systems are designed to reduce this overhead. However, even with modest frame losses, which are quite common, they lose efficiency and reduce throughput.

IV. Fast Wireless Protocol

Our approach to resolving issues with overhead problems created by wireless MAC reliability is to switch it off. With the MAC acknowledgment system turned off there is no ACK overhead, and since the sliding window system is turned off, it cannot cause a queue stall. We instead push reliability and in-order delivery systems to the session layer above the transport.

With the MAC acknowledgment system turned off and the MAC reorder buffer disabled the transport system will receive a stream of data with missing segments. In common forms of TCP such as Cubic, Compound, and NewReno this will cause false congestion events and reduce throughput. To resolve this problem we designed a receiver side modification to TCP called compatible TCP.

Compatible TCP will deliver the data stream to the Session Layer with missing chunks of data. To resolve this problem, we designed an HTTP retransmission scheme with a reorder buffer that will replace the missing chunks of data. Once the retransmission is complete, it will release the buffer and deliver the data to the application. The application remains oblivious that the stack has changed beneath it.

In order to make the needs of our FWP system more concrete we have developed a set of requirements that we believe an FWP stack must have.

1) Remove data protection and in-order delivery services from the MAC.
2) Remove data protection and in-order delivery services from the transport layer.
3) TCP congestion control must remain intact.
4) Implement data protection and in-order delivery services at the session layer.

Requirements 1 and 2 ensure that the flow of data is not inhibited by the lower layers. Requirement 3 ensures that the design is interoperable with other flavors of TCP. Requirement 4 ensures that the application is unaware of the new network stack.

Requirement 1 is accomplished by turning off the positive acknowledgment system in the 802.11 MAC and reducing the MAC reorder buffer timeout to zero. We used the 802.11n

standard's NoAck policy to turn acknowledgments off. With the acknowledgment system turned off the A-MPDU frame aggregation system's BAW sliding window is disabled. In this mode there is no possibility of a queue stall. These steps satisfy Requirement 1.

Our TCP compatible solution fulfills Requirements 2 and 3. It removes reliability and in order delivery services from TCP while leaving congestion control intact. To avoid the reliability and in order delivery TCP compatible detects sequence number holes and fills them in by injecting placeholder segments. Congestion control is maintained by means of a switch. When there is no congestion, TCP compatible injects placeholder segments, when congestion is detected TCP compatible is switched off. With TCP compatible switched off TCP will throw congestion events normally and maintain fairness with other flows.

We fulfilled Requirement 4 by implementing data protection and in order delivery systems in the session layer at *socket.c*. This consists of a reorder buffer and an HTTP retransmission scheme. This design decision produces overhead since it retransmits the lost data over the entire path rather than just the wireless hop.

V. FWP Implementation

We implemented a prototype in order to facilitate our evaluation and to make practical the conceptual framework described in Section IV.

Our FWP prototype design consists of four components.

1) An 802.11 FWP aggregation emulator.
2) An 802.11 A-MPDU aggregation emulator.
3) A compatible TCP.
4) An HTTP retransmission scheme.

We decided to use an emulator rather than 802.11n drivers and hardware for our experiments. We made this choice because emulation allowed us to study the effects of 802.11 frame aggregation without interference from other 802.11 systems. This is difficult to accomplish with wireless hardware because wireless standards have multiple systems interacting with each other. The rate adaptation system interacts with the driver to construct the transmit retry chain, and both of these systems interact with the packet aggregation system. These interactions between systems are driven by external effects that are difficult to control in an experiment.

In addition, the emulators let us compare the systems while they are in a constant state. This is critical because we need to be certain that any effects observed in our experiments are not caused by seemingly random external effects. For instance if there is a throughput change we need to be certain that it was not caused by a Modulation and Coding Scheme (MCS) change or other system interaction.

We implemented our emulators using packet schedulers running in kernel modules at the IP layer. We implemented 4 MIMO streams with channel error calculated independently on each stream. The channel error rate on each stream is the same in order to facilitate channel contention experiments. We set MTU size to 1500 bytes making a packet about equal to a frame. Packets are enqueued to each aggregation queue (1 per MIMO stream) round robin. We implemented two modes of operation, FWP, and A-MPDU. We also built an A-MSDU aggregation emulator. However, the performance of A-MSDU aggregation was so poor that we will not highlight any of the experiments in order to save space. Also we found that in the Atheros 802.11n drivers that A-MSDU aggregation is not implemented and we suspect that is the case for other drivers as well.

In the dequeue function of the emulators we implemented aggregation. When enough packets have been enqueued to an aggregation queue to fill an aggregate (or to fill it as much as possible in the case of an A-MPDU) the aggregate is delayed to account for the overhead of the radio header, IFS, and ACK time. Frame loss is calculated, then successful packets are sent across 1 Gbps Ethernet. Many simulators use a Bit Error Rate (BER) curve to calculate frame loss over wireless channels. However, since aggregations do not contain error correction bits (only per frame checksums) this is unnecessary. To streamline kernel calculation, we used the Frame Error Rate (FER) model instead. Our designs are based on the open source Atheros driver code and the IEEE 802.11n standard. Other drivers are proprietary. However, we believe that they behave in a similar fashion.

A. 802.11 FWP Aggregation Emulator

Requirement 1, Section IV calls for the wireless driver to operate in A-MPDU mode with no ACKs, this is called ADDBA noack mode. We emulate this by queuing packets into our emulated aggregation queues. The maximum A-MPDU aggregation size in the Atheros driver code is 32 rather than the 64 specified in the IEEE standard. We use 32 packets to an aggregate following the Atheros code. Frame loss is calculated using the FER. Packets representing lost frames are dropped. Packets representing successful frames are transmitted across 1 Gbps Ethernet after an appropriate delay for wireless transmission. This is demonstrated in Figure 3.

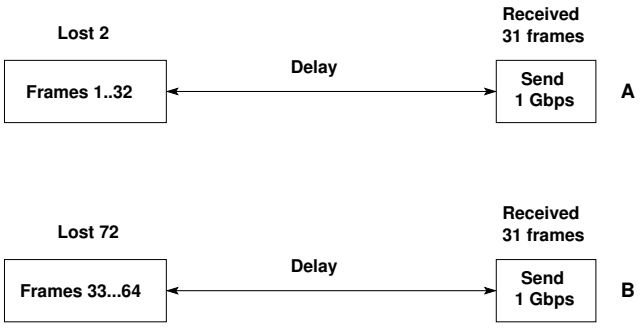

Fig. 3. FWP 802.11 emulator using ADDBA noack

In Time Series A, enough packets to represent 32 frames are queued. Frame 2 is lost so its packet is dropped. A total of 31 frames are received. In Time Series B, the aggregate is filled with 32 more packets and the process is repeated. Another frame is lost and 31 more frames are received. A total of 62 frames have arrived at the receiver. The aggregate is always filled and there is no sliding window. The throughput with FWP is always linear with frame loss because frame errors are not corrected.

B. 802.11 A-MPDU Aggregation Emulator

In our A-MPDU emulator frame loss is also calculated using FER. However this time, packets representing lost frames are not dropped. Instead they are held in the queue. Packets representing successfully transmitted frames are delayed for overhead then transmitted over 1 Gbps Ethernet. In the next aggregate the BAW is advanced up to the first lost frame. New packets are added to the aggregate queue to account for the BAW advance. The aggregation queue now holds these new packets as well as any packets representing lost frames from the last round.

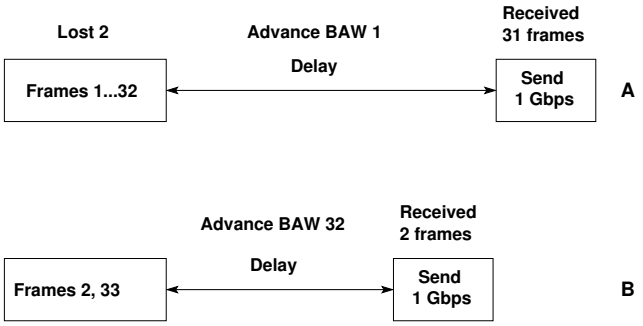

Fig. 4. A-MPDU 802.11 emulator

Figure 4 demonstrates this. In Time Series A, frame 2 is lost and the BAW is advanced 1. The packet representing frame 2 is not transmitted. The successful frames are delayed and transmitted over 1 Gbps Ethernet. In Time Series B, frame 2 is still in the queue from last time. The BAW has advanced one so the end of the window now points to 33. Packets representing frames 2 and 33 are transmitted. All of the other frames within the window have been successfully transmitted and the A-MPDU cannot be filled. In this example, both frames are successful in Time Series B, so the packets are delayed for overhead and then dequeued.

The A-MPDU in Time Series B, represents a lost transmission opportunity. If there is no contention then this will not matter much and the A-MPDU because the station will be able to transmit again without contention. The system will achieve reasonable performance. However, in the normal case when there is contention for the wireless medium the station will not be able to regain airtime until it wins contention again. In contention with other stations the A-MPDU system will lose throughput because of missed transmission opportunities.

C. TCP Compatible

A standard TCP such as Cubic, Compound, or New Reno will not work well with our FWP system because it delivers data to the transport layer with missing segments. In fact, these TCPs are quite sensitive to sequence number holes. When a sequence number hole occurs dupACKs are sent for the missing segment until it is received. It takes approximately one Round Trip Time (RTT) to retrieve a missing segment. In normal use with RTTs above 30 ms many dupACKs will be sent before the missing segment is received. This behavior erroneously triggers the congestion control mechanism when segments are lost due to wireless transmission through the 802.11 FWP Aggregation Emulator.

In order to cope with this problem we developed compatible TCP. In keeping with our receiver side only design philosophy the server side transport layer code remains untouched implementing whatever congestion control mechanism is selected in the server kernel. Compatible TCP will fulfill Requirements 2, and 3 from Section IV.

In order to fulfill Requirement 2 (remove in order delivery, and reliability) we monitor the TCP ACK stream. When a duplicate ACK is detected a placeholder segment is generated. The sequence number of the placeholder segment is fixed to the sequence number of the missing segment, and the DATA section is filled with marker data to indicate that the data in this segment was not received. The placeholder segment is then checksummed and injected into the stack filling in the sequence number hole. This prevents TCP from detecting missing segments removing data protection and in order delivery services from the transport layer as specified in Requirement 2 from Section IV.

Requirement 3 specifies that the TCP congestion control must remain intact. TCP congestion control not only prevents congestion collapse but also maintains fairness with other TCPs. In order to re-establish the congestion control mechanism we operate our compatible TCP in a bi-stable mode. In one state the compatible TCP injects placeholder packets, and in the other state normal TCP behavior is observed.

In order to discern congestion loss from wireless loss we use information from the 802.11n rate adaptation system. It provides us with the expected frame loss for our current Modulation and Coding Scheme (MCS). We measure our segment loss rate in TCP compatible. If the loss rate measured over a window of 1 RTT is less than the expected wireless frame loss rate then the loss is determined to be wireless related. We remain in placeholder injection mode. If the loss rate exceeds the expected wireless frame loss rate then the loss is determined to be congestion related and we operate in normal TCP mode.

An example of this is shown in Figure 5. Segment 3 is lost and filled in with a placeholder as soon as the sequence number hole is detected (when segment 4 arrives). At segment 9 the sequence number hole count has exceeded the frame error rate reported by the 802.11n rate adaptation system. In response we switch to normal TCP mode. No placeholders are injected and duplicate ACKs are sent triggering a congestion event. TCP fairness is maintained.

D. HTTP Retransmission Scheme

The final requirement from Section IV implements in order delivery and data protection services. This allows applications to remain unaware of the rearrangement in the stack below them. We use an HTTP retransmission scheme in order to accomplish this. The placeholder packets are easily recognized in the data stream by a character search. The beginning and the end of the placeholder data indicate the byte range that must be requested to replace the missing data. We use curl[7] to request the byte range and insert the missing data into the stream before releasing the buffer to the application.

[7] http://curl.haxx.se/

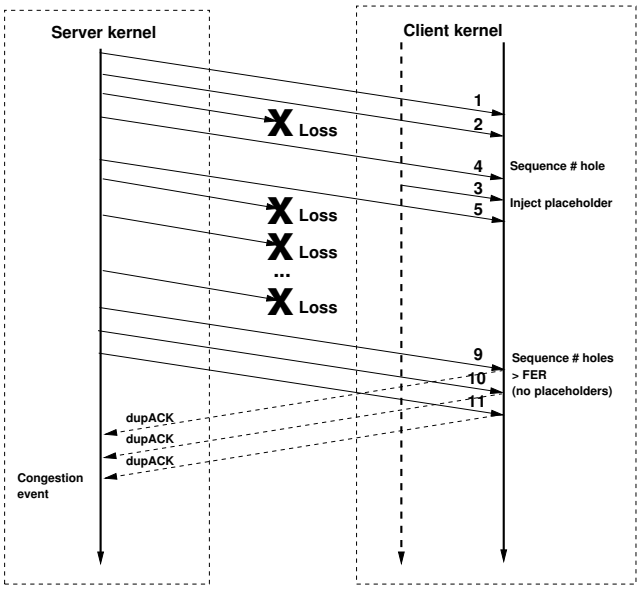

Fig. 5. TCP Compatible Transport

We had to implement a reorder buffer in order to wait for the data to be replaced. Data is released normally until the first missing chunk. In our prototype the system waits until the data is replaced. In a production system we believe a timeout should be implemented. The system introduces a variable end-to-end delay (jitter). We believe that the jitter is not important since our target applications have large jitter buffers, 30 seconds for Video on Demand, and 8 seconds for live video. This is enough to absorb the jitter.

There are two types of overhead that are caused by this retransmission scheme. The first is the network overhead caused by retransmitting the data from the server. This is discussed in detail in Section VI-D. The second type of overhead is that caused by the cpu having to identify the placeholders (character search) and replace the data once it has been retrieved. Because the cpu is many times faster than the network (in almost all cases) this overhead is negligible.

VI. Evaluation

We evaluated FWP by observing three key characteristics of our solution: throughput, TCP interoperability (fairness), and overhead. FWP is designed to take full advantage of every transmit opportunity and achieve better throughput than 802.11 frame aggregation during contention. We evaluate the throughput gains of our solution over varying channel conditions and number of competing stations.

In our design, we replaced the receiving side TCP with our compatible TCP. To determine interoperability with other TCPs we perform fairness testing. We introduce a variant of Jain's fairness metric to determine whether our TCP solution shares fairly with other TCPs. In addition we characterize the overhead introduced by our HTTP retransmission scheme.

Our wireless hardware is emulated. We understand that emulators have difficulty modeling the complex interactions of the wireless channel. Because of this, we do not rely on our emulator to provide absolute values, but instead use it to understand behaviors and trends that cannot be observed in isolation with real hardware. We first describe our testbed then the results.

A. Testbed

In order to evaluate our FWP solution and compare it against A-MPDU frame aggregation we built a testbed in Emulab facilities provided by the Flux Group, part of the School of Computing at the University of Utah[8].

Nodes 0 through m in Figure 6 are client nodes. Node m is equipped with an FWP stack. Because Emulab topologies are easily configurable, we can vary the number of clients and servers to fit the needs of the experiments. This allowed us to test contention with a variable number of client and server nodes in our testbed. The servers (nodes $m+1$ through n) each serve one client from $nodes0$ through m. Nodes $n+1$ and $n+2$ emulate the Internet path from each server to each client. All links are 1 Gbps Ethernet.

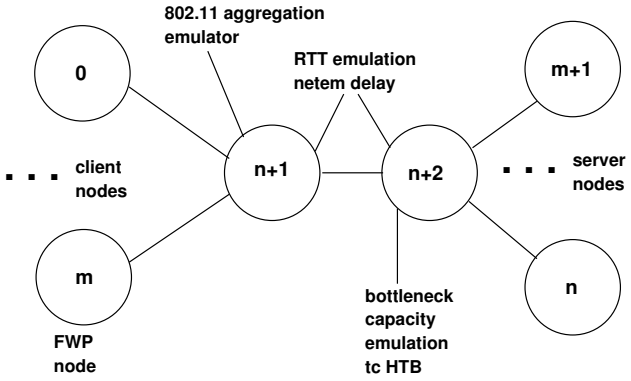

Fig. 6. Emulab testbed

Node $n+1$ contains the aggregation emulators and Node $n+2$ emulates Internet bottleneck router capacity. We use our aggregation emulators and tc[9] filters and an HTB packet scheduler[10] to accomplish this. The Internet path RTT is implemented using netem[11] to introduce one way delay on the egress interface of nodes $n+1$ and $n+2$. All streams flow through the same bottleneck and have the same RTT in order to compare FWP against 802.11 aggregation in the client nodes.

Experiment Parameter	Default Value
Round Trip Time	40 ms
RTT Variance	10 percent
Bottleneck Bandwidth	600 Mbps
Frame Loss	$10^{-3} - 10^{-2}$
Experiment Duration	120 seconds
Experiment Runs	10

TABLE I. Default Experimental Parameters

Table I shows a list of default experimental parameters. Unless otherwise specified in an experiment the parameter values will be set as shown in the table. We chose a default

[8]https://www.emulab.net/
[9]http://www.lartc.org/
[10]http://linux.die.net/man/8/tc-htb
[11]http://www.linuxfoundation.org/networking/netem

RTT of 40 ms. Although the average RTT of an Internet path is a nebulous and debatable point our observations have shown that RTTs between 30 and 50 ms are within reason.

Because of the interaction of multiple router queues along an Internet path RTTs in the Internet are not stable values. In order to emulate this we used netem to randomly varying the delay according to a distribution we chose to vary the RTTs +/-10% in a normal distribution about the mean.

We chose to use 600 Mbps throughput capacity at the bottleneck since this is the theoretical maximum of 802.11n. Our links are 1 Gbps so it was not possible for us to measure more than one 802.11n station operating at full theoretical maximum. Experiments were run for 2 minutes with 10 experimental runs.

B. Throughput

To understand the throughput gains achieved by our FWP system we first highlight an example from a series of experiments with 10 competing stations sharing 600 Mbps of bandwidth. Figure 7 shows a comparison of our FWP versus 802.11 frame aggregation measuring throughput achieved at the receiving station against probability of frame loss.

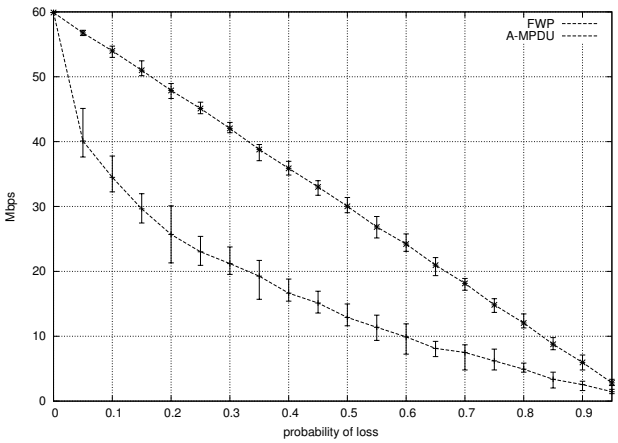

Fig. 7. Throughput gains from 1 FWP station competing with 9 A-MPDU aggregation stations

As expected the throughput achieved by our FWP system is very linear to loss. This is because we transmit a full (32 frame) aggregate every time we win contention and do not stop to retransmit. The A-MPDU system on the other hand is not linear to loss. The A-MPDU aggregation system suffers a drastic reduction of throughput at even very small error rates. It achieves about two thirds of the throughput of FWP at .05 FER, and about half at a probability of .20 FER.

The emulation shows that the trending throughput gains are significant especially at the FER range of 10% to 40%. This range of FER is critical because this is where a rate adaptation system might make a decision such as 40% FER at 600 Mbps is preferable to 0% FER at 300 Mbps.

Next we sought to determine the effect of the number of competing stations on our FWP system versus A-MPDU aggregation. We plot the throughput gains achieved by 1 FWP station competing with 1 to 9 A-MPDU stations.

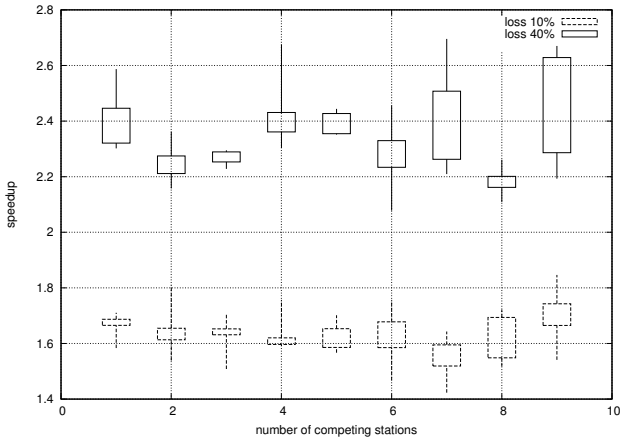

Fig. 8. Speedup of FWP against number of competing A-MPDU stations

We define a metric called speedup to measure the throughput gains. Speedup is the number of times faster that FWP is than A-MPDU aggregation. For instance a speedup of 1 would be the same throughput; a speedup of 2 would indicate twice as much throughput. The experiments were each two minutes in duration. The graph in Figure 8 shows a whisker plot of 10 experimental runs. The bars in the whisker plot show values from the 10th to the 75th percentile, and the whiskers show the minimum and maximum values.

We can see from Figure 8 that the throughput gains are more strongly effected by FER (loss) than by the number of competing stations. In fact the throughput gains are reasonably flat across the number of competing stations. They are averaging greater than 1.6 for 10% FER, and greater than 2 for 40% FER.

These experiments have demonstrated that the throughput gains of FWP over current wireless aggregation technology are significant and not dependent on the number of competing stations. This indicates that in real wireless hardware FWP would almost always (any non zero channel error condition) develop significant throughput gains and that these gains would increase with the amount of channel error.

C. Fairness

One of the key goals of our dirty slate design for FWP is deployability. Because of this we had to determine whether our compatible TCP operates fairly with other TCPs. First we wished to determine if the characteristic competitive behavior of TCP has been affected by our modifications. This involved many time series experiments to determine that the waveform generated by our compatible TCP is similar to the waveform generated by a popular TCP such as Cubic.

In Figure 9 we see an excerpt from one of this series of experiments. The bottleneck router capacity in this experiment was 600 Mbps and the RTT was 40 ms. The graph shows that the competitive behavior of TCP remains intact in our compatible TCP. One flow gains an advantage over the other for a time and then the roles reverse. Normally the variations in throughput are quite small but sometimes over larger periods of time the fluctuations are larger.

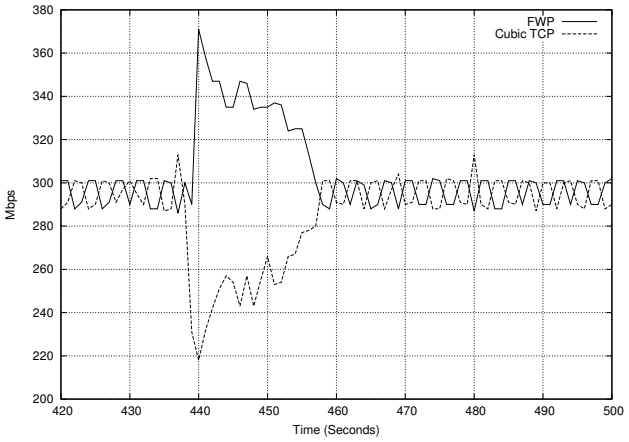

Fig. 9. Compatible TCP competing with Cubic over a time series

These experiments have shown that the competitive behavior of TCP is largely unaffected by our modification. In the next series of experiments we perform a more rigorous system testing of TCP fairness over a variety of RTTs. In order to clearly present these experiments we define a metric called fairness. This is loosely based on Jains fairness index.

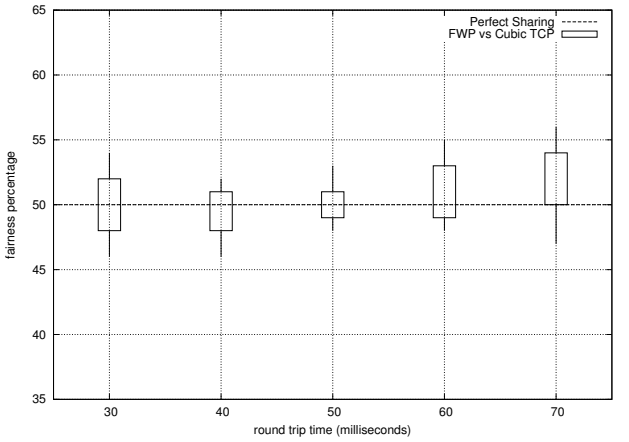

Fig. 10. Throughput variations of compatible TCP against TCP Cubic

We ran this series of experiments for 2 minute durations and determined the variations in throughput over 1 second intervals. The bottleneck router capacity in these experiments was 600 Mbps and we plot the percentage of throughput achieved by our TCP compatible against a competing Cubic TCP.

The design of our TCP compatible can take up to an RTT longer than a normal TCP to begin congestion control behavior. This could lead to more aggressive behavior over larger RTTs. We believe that it is okay for our TCP compatible to act in a more aggressive manner when the RTT is larger because TCP throughput gets smaller as RTTs increase. However, the amount of aggression should not be too great at normal RTTs.

The whisker plot in Figure 10 shows our fairness percentage results over 10 experimental runs. The line at 50% fairness indicates ”perfect sharing”. This would mean that both TCPs (compatible and Cubic) received 1 half of the throughput over each 1 second interval. A ”fair” TCP should not fluctuate much above or below this perfect sharing line. Plus or minus 5% would indicate very good sharing, and +/- 10 % would still be a very reasonable amount of fairness.

The graph shows that our compatible TCP operates in a reasonably fair manner when competing with a Cubic TCP across a wide range of RTTs from 30 to 70 ms. We see a small amount of additional aggression at the higher RTTs (60 - 70 ms), however, the sharing of bottleneck router resources is still very reasonable.

These experiments have shown that our compatible TCP behaves in a manner consistent with standard TCP behavior and that it is reasonably fair with other TCPs. We believe that our compatible TCP is interoperable with other TCPs fulfilling requirement 3 from Section IV.

D. Overhead

One of the tradeoffs with our FWP solution is that the HTTP retransmission reliability system generates network overhead. There are two types of overhead that we examined in our experimentation. The first is the additional data transfer across the wireless link. The overhead across the wireless portion of the link is small because the data portion would have had to have been retransmitted anyways. The only additional overhead added to the wireless hop is the HTTP header required to specify which portion of the data needs to be retransmitted.

The second type of overhead that we examine is the amount of additional data transfer across the Internet path. Because our HTTP retransmission scheme retrieves missing data from the server this data must be sent all the way across the path. For every other hop on the path the retransmitted data is pure overhead since the data would have been transmitted from the access router instead of the server. Figure 11 shows the comparison of both kinds of overhead. The overhead across the Internet path grows quickly reaching 100% at a 50% FER. This shows that our FWP system places a large burden on the Internet path rather than the wireless hop.

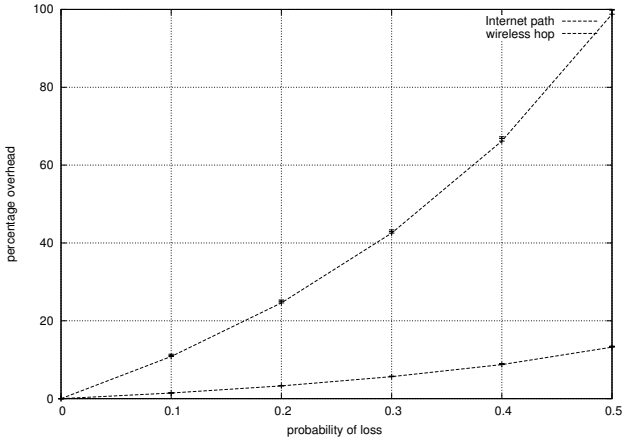

Fig. 11. Overhead for Internet path versus wireless hop

The overhead for the wireless hop however, is quite small. It only reaches 13% at 50% FER. These experiments have

shown that our FWP system provides a great deal of throughput gain versus overhead from the point of view of the wireless hop. However, the increasing overhead across the Internet path becomes a limiting factor. We plan to address this problem in future work.

VII. Conclusions and Future Work

In this work we have developed FWP, a very fast wireless protocol. We have done this using our novel receiver side only architecture. The system does not require any in-network or server side changes whatsoever. In addition, it does not require any special encodings or data formats. Our receiver side only architecture solves the deployment problems encountered by previous work in this area.

However, we found that even though the amount of overhead introduced into the wireless hop is minimal, the amount of overhead introduced into the Internet path is large. In our future work we plan to address this problem by expanding the definition of administrative domain to include not only the receiving station, but also the access router. This will allow us to use the driver at the access router to buffer frames. We will construct a modified version of our session layer retransmission scheme that does not use HTTP. It will instead use the Block ACK in the 802.11 frame aggregate to release or retransmit frames. This will eliminate the overhead across the Internet path. In addition, we intend to investigate the idea of injecting placeholder frames in the 802.11 driver rather than in the transport layer.

References

[1] A comprehensive tcp fairness analysis in high speed networks. *Computer Communications*, 32(13):1460–1484, 2009.

[2] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *ACM conference on Multimedia systems (MMSYS)*, pages 157–168, 2011.

[3] S. Bauer, R. Beverly, and A. Berger. Measuring the state of ecn readiness in servers, clients,and routers. In *Internet Measurement Conference*, IMC, pages 171–180, 2011.

[4] L. Chen. TFRC Modeling and Its Applications. Master's thesis, University of Hong Kong, China, 2003.

[5] D.-M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1):1–14, 1989.

[6] L. D. Cicco and S. Mascolo. An experimental investigation of the akamai adaptive video streaming. In *HCI in work and learning, life and leisure: workgroup human-computer interaction and usability engineering*, pages 447–464, 2010.

[7] N. Dukkipati, M. Mathis, Y. Cheng, and M. Ghobadi. Proportional rate reduction for tcp. In *Internet Measurement Conference*, IMC, pages 155–170, 2011.

[8] J. Erman, A. Gerber, K. Ramadrishnan, S. Sen, and S. O. Over the top video: the gorilla in cellular networks. In *ACM SIGCOMM (IMC)*, pages 127–136, 2011.

[9] A. Gember, A. Anand, and A. Akella. A comparative study of handheld and non-handheld traffic in campus wi-fi networks. In *Passive and Active Measurement*, volume 6579, pages 173–183. 2011.

[10] S. Ha, I. Rhee, and L. Xu. Cubic: a new tcp-friendly high-speed tcp variant. *SIGOPS Operating Systems Review*, 42(5):64–74, July 2008.

[11] M. Handley, S. Floyd, J. Padhye, and J. Widmer. Tcp friendly rate control (tfrc): Protocol specification, 2003.

[12] S. Hassayoun, J. Iyengar, and D. Ros. Dynamic window coupling for multipath congestion control. In *International Conference on Network Protocols (ICNP)*, pages 341–352, October 2011.

[13] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend tcp? In *Internet Measurement Conference*, IMC, pages 181–194, 2011.

[14] R. Jain. *Design and implementation of split tcp in the linux kernel*. PhD thesis, 2007.

[15] C. Jin, D. X. Wei, and S. Low. Fast tcp: motivation, architecture, algorithms, performance. In *INFOCOM*, volume 4, pages 2490–2501, March 2004.

[16] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340 (Proposed Standard), March 2006.

[17] E. Kohler, M. Handley, and S. Floyd. Designing dccp: congestion control without reliability. *SIGCOMM Computer Communication Review*, 36(4):27–38, August 2006.

[18] M. Luby, T. Stockhammer, and M. Watson. Iptv systems, standards and architectures: Part ii - application layer fec in iptv services. *IEEE Communications Magazine*, 46(5):94–101, May 2008.

[19] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband internet traffic. In *ACM SIGCOMM (ICM)*, pages 90–102, 2009.

[20] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *Mobile Computing and Networking*, MobiCom, pages 287–297, 2001.

[21] D. W. Pei. Time for a tcp benchmark suite? 2008.

[22] H. Schulzrinne, , S. Casner, R. Fredrick, and V. Jacobson. Rtp: A transport protocol for real-time applications. RFC 3550, July 2003.

[23] H. Schulzrinne, G. Fokus, and S. Casner. RTP: A transport protocol for real-time applications. RFC 1889, January 1996.

[24] R. Stewart and C. Metz. Sctp: new transport protocol for tcp/ip. *IEEE Internet Computing*, 5(6):64–69, Novenber 2001.

[25] T. Stockhammer, G. Liebl, and M. Walter. Optimized h.264/avc-based bit stream switching for mobile video streaming. *EURASIP Journal of Applied Signal Process.ing*, 2006:1–19, January 2006.

[26] J. Sundararajan, D. Shah, M. Me anddard, S. Jakubczak, M. Mitzenmacher, and J. Barros. Network coding meets tcp: Theory and implementation. *Proceedings of the IEEE*, 99(3):490–512, March 2011.

[27] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound tcp approach for high-speed and long distance networks. In *INFOCOM*, pages 1–12, April 2006.

[28] A. Tang, L. L. H. Andrew, M. Chiang, and S. H. Low. Transport layer. In *Encyclopedia of Computer Science and Engineering*, pages 2930–2938. January 2009.

[29] A. Tang, J. Wang, S. Hegde, and S. Low. Equilibrium and fairness of networks shared by tcp reno and vegas/fast. *Telecommunication Systems*, 30:417–439, 2005.

[30] M. van der Schaar, S. Krishnamachari, S. Choi, and X. Xu. Adaptive cross-layer protection strategies for robust scalable video transmission over 802.11 wlans. *IEEE Journal onCommunications*, 21(10):1752–1763, December 2003.

[31] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, implementation and evaluation of congestion control for multipath tcp. In *Networked Systems Design and Implementation (USENIX)*, NSDI, 2011.