TCP-Cognizant Adaptive Forward Error Correction in Wireless Networks

Benyuan Liu ¹, Dennis L. Goeckel ², Don Towsley ¹
¹ Department of Computer Science
University of Massachusetts
² Department of Electrical and Computer Engineering
University of Massachusetts

Abstract—Wireless links are characterized by high bit error rates and intermittent connectivity. These can significantly degrade the performance (goodput) of TCP over wireless networks since non-congestion related packet losses can be misinterpreted by TCP as indications of network congestion, resulting in unnecessary congestion controls, and thus a reduced goodput. In this paper, we propose a technique, TCP with adaptive forward error correction (TCP-AFEC), to improve TCP performance over wireless networks. TCP-AFEC combines the well-established performance characterization of TCP with an understanding of the link layer error control scheme to dynamically select the forward error correction (FEC) that maximizes TCP goodput according to the current channel condition. The benefit of coupling TCP performance characterization with link layer FEC to improve TCP goodput is demonstrated by comparing the performance of TCP-AFEC against those of TCP-SACK, Snoop protocol and a physical layer optimization scheme. Simulation results show TCP-AFEC outperforms TCP-SACK, Snoop and the physical layer optimization scheme for a wide range of wireless channel conditions.

Keywords—TCP performance, wireless networks, forward error correction, adaptive coding

I. INTRODUCTION

TCP is a reliable transport protocol originally designed for wired networks. Typical wired links exhibit very stable transmission characteristics and a very low bit error rate. As a consequence, packet losses in wired networks are mostly caused by buffer overflows, indicating that the network is congested. The TCP sender then responds to network congestion by triggering its congestion control mechanism. The congestion window is reduced either to half of the original size or to one packet, depending on whether the congestion indication is a timeout (TO) or triple duplicate acknowledgements (TD). This significantly reduces the amount of traffic injected into the network and has been shown to be very effective for alleviating congestion.

Recently there has been substantial activity in the area of mobile wireless data networks. Since TCP is the prevalent reliable transport protocol in today’s Internet and has been widely used in many application layer protocols, such as HTTP, SMTP, FTP, etc., it must be supported in the wireless regime in order to make the wireless networks an integral part of the Internet. However, wireless links exhibit very different characteristics from traditional wired links. Wireless transmissions are subject to attenuation with distance (“path loss”), shadowing, multipath fading, and other interference [1], resulting in high bit error rates that vary with time over the links. In a TCP connection, the packets corrupted over wireless links are dropped from the protocol stack at the receiver and, thus, are lost. The TCP sender cannot distinguish packet loss due to bit corruption from real congestion loss due to buffer overflow. Packet loss due to bit corruptions over wireless links can be misinterpreted by TCP as indications of network congestion, and will unnecessarily trigger the TCP congestion control mechanism, resulting in a degraded performance. Therefore, it is crucial to improve the performance of TCP over wireless links to support the fast adoption and deployment of wireless networks. In the past few years, there have been many proposals to improve TCP performance in wireless networks, most of which aim at shielding the original TCP sender from the non-congestion related packet errors over wireless links. These proposals will be summarized in Section II.

In this paper, we propose to couple TCP with an adaptive forward error correction protocol (AFEC) in order to improve TCP’s performance in a base station oriented wireless network. The approach integrates the TCP performance characterization and the link layer forward error correction (FEC) performance by employing a well-established TCP throughput formula [2] to compute the forward error correction (FEC) code that maximizes TCP goodput. In particular, we analyze the effect that adding FEC to TCP packets between the base station and mobile host has on TCP goodput and present an algorithm to select the code to be used for a given channel condition. The benefit of com-
bining TCP performance characterization with link layer FEC to improving TCP goodput over wireless networks is demonstrated through comparison with several other proposals. We show that TCP combined with adaptive FEC, TCP-AFEC, significantly improves TCP performance over TCP-SACK across a wide range of wireless channel conditions. We also compare TCP-AFEC with the Snoop protocol [3], a proposal that provides the biggest improvement over TCP Reno [4]. Our simulation results show that TCP-AFEC achieves a goodput comparable to that of Snoop at low bit errors rates and considerably higher goodput than Snoop when bit error rates are high, and thus is more robust than Snoop over a broad range of wireless channel conditions. Furthermore, we demonstrate the importance of knowledge of TCP characteristics in improving TCP performance by comparing the performance of TCP-AFEC to that of a physical layer optimization scheme, which maximizes the wireless link performance from the physical layer perspective. Without knowledge of TCP performance characterization, the physical layer optimization scheme achieves much worse performance than TCP-AFEC for medium to bad wireless channel conditions.

Note that a similar approach has been independently proposed in [5]. However, the development in this paper is more closely tied to the physical layer assumptions (as opposed to the packet error process model assumed in [5]) and is expected to yield better performance. Furthermore, we consider a number of important practical issues.

The rest of the paper is organized as follows. Section II summarizes the existing related works on TCP over wireless networks. Section III describes in detail the TCP-AFEC protocol and highlights the characteristics of the approach. In Section IV, we evaluate the performance of TCP-AFEC, and compare the goodput of TCP-AFEC with that of TCP-SACK and the Snoop protocol. In Section V we discuss some implementation and practical issues related to TCP-AFEC. Conclusions are presented in Section VI.

II. RELATED WORK

In this section, we review previous work on improving TCP performance over wireless networks. Previous efforts approximately fall into one or more of the following several categories. For each category, we describe the common characteristics and introduce some of the representative works that belong to the category.

1. Split-Connection Approach

There are many proposals that fall into this category, among which I-TCP [6][7] was one of the early efforts. In this approach, a TCP connection is split into two separate TCP connections at the base station. One connection is between the sender and the base station, while the other is between the base station and the mobile host. Both connections use TCP for the transport protocol and the two TCP connections are concatenated together to replace the original one. This approach tries to shield the TCP sender from packet loss over the wireless link by separating the wireless connection from the wired part. The drawbacks of this approach include the violation of end-to-end semantics of TCP and a large overhead for state maintenance and protocol stack operations at the splitting point. Experimental results [4] show that in I-TCP, the sender often stalls due to timeouts on the wireless connection, resulting in poor end-to-end throughput. Other variations include [8][9][10]. These protocols split the TCP connection at the base station but propose to replace the TCP protocol over the wireless link with more fine tuned wireless-specific schemes.

2. Proxy-Assisted Approach

A typical example of this approach is the Snoop protocol [3]. Snoop installs a TCP-aware proxy at the base station to monitor the packet loss over the wireless link, attempting to do local recovery for packet errors. The wireless loss indications (duplicate acks) are suppressed and shielded from the TCP sender. Snoop has been shown to yield a 10-30% higher throughput than normal TCP-RENO. One drawback of Snoop is that the Snoop proxy needs to cache all of the TCP packets and maintain the states (such as local retransmission timer, wireless hop RTT estimate, etc) at the base station for potential retransmissions. This introduces considerable management and buffer overhead at base stations.

3. Link-layer Error Control Approach

Forward error correction (FEC) and automatic repeat request (ARQ) are two widely used link layer error control techniques. These two techniques or their combination have been adopted [11] to improve TCP performance by providing error control at the lower layer, thus shielding the losses from the TCP sender. In another proposal, TULIP [12] uses a simple selective repeat retransmission scheme coupled with a packet interleaving strategy at the link layer to improve TCP performance over half-duplex links. One well-known disadvantage of link layer approaches is that the interactions between link layer and TCP retransmissions can have an adverse effect on the TCP performance [13][4]. Some of the link layer approaches [14][15][16] dynamically adjust the level of redundancy in FEC codes, retransmission limits, or packet sizes to adapt to the changing wireless environment. However, no analytical guidelines were provided regarding how to select the optimal FEC redundancy level to maximize the TCP throughput. In a recent work [5], the authors analyzed the tradeoff between the bandwidth consumed by forward erasure correction codes and the goodput gained by a TCP session. They then proposed an algorithm to compute the optimal code that maximizes TCP goodput.
The focus of this paper was on forward erasure correction codes. However, for wireless links, the major concern is not packet loss, but bit errors within the packets. In our work, we focus on the bit errors over wireless links and use forward error correction instead. We also provide a more comprehensive treatment of many related issues.

4. ELN-based Approach

This approach [17][18] uses explicit loss notification (ELN) signals to distinguish between congestion losses on the wired links and non-congestion related losses over the wireless link. A TCP-aware agent is located at the bases station, watching the packets passing through. When the loss is the result of corruption over the wireless link, the agent sets an ELN bit in acknowledgement packets to notify the TCP sender that the loss is not caused by congestion. Upon detection of the ELN bit, the TCP sender does not invoke normal congestion control. The sending window is not reduced in this case and degradation of the TCP goodput is thus avoided. Like the Snoop protocol, there is state maintenance overhead at the base station. Experiments [4] show that ELN effectively prevents rapid fluctuations in the congestion window and achieves a throughput improvement of more than a factor of two over TCP-Reno. Note that if the world used ECN [19], then all losses could be interpreted to be due to wireless noise.

5. End-to-End Approach

The end-to-end approach attempts to improve TCP performance at end hosts, without any assistance from intermediate nodes like base stations. Many protocols belonging to this approach proposed modifications or extensions to TCP-Reno to handle the problems that arise in a wireless environment. TCP-SACK uses selective acknowledgements from the receiver, thus allows the TCP sender to recover from multiple packet losses in a window without resorting to a coarse timeout. TCP-Westwood [20] computes the congestion control window and slow-start threshold using the measured effective bandwidth. This helps the TCP sender to discriminate the reason of packet loss, which is the major problem that causes performance degradation in TCP-Reno.

III. TCP-AFEC Protocol

In this section we first introduce a formula for the throughput of a TCP connection as a function of packet loss rate and session round trip time (RTT), the effect of forward error correction on TCP goodput, and the wireless channel model that will be employed. Following this, we then present the TCP-AFEC protocol.

A. TCP Throughput Formula

Despite the complex behavior of TCP due to its various mechanisms such as slow start, congestion control, timeout, etc., it has been shown in [2] that the throughput of a TCP connection is a simple expression of packet loss rate (\(p\)) and average round trip time (\(RTT\)). The TCP goodput, \(G_f\), can simply be obtained by scaling the throughput by a factor of \((1 - p)\):

\[
G_f = \min\left\{ \frac{W_{\text{max}}}{RTT}, \frac{1}{RTT \sqrt{\frac{W_{\text{max}}}{3} + T_0 \min\{1, 3\sqrt{\frac{W_{\text{max}}}{3}}\} p(1 + 32p^2)} \right\} (1 - p)
\]

where \(W_{\text{max}}\) is the maximum congestion window size of the TCP sender, \(b\) represents the effect of delayed ack, and \(T_0\) is the TCP transmission timeout value.

The above formula has been shown to accurately predict TCP goodput over a wide range of packet loss rates [2][21][22]. Using the formula, it is easy to obtain the achievable TCP goodput for a given packet error rate and round trip time (RTT), provided that the link bandwidth is sufficient for the goodput. Note that the packet error rate, \(p\), appears as a factor in the denominator of the formula. Hence, reducing the packet error rate helps to increase TCP goodput. This is exactly the underlying theoretical argument for many proposals that try to shield the TCP sender from packet loss over the wireless link.

B. Effect of Forward Error Correction

Forward error correction (FEC) has been widely used in wireless data communication systems to combat transmission errors at the link layer. In FEC, parity-check bits add redundancy are added to the data to form a codeword, and the codeword is transmitted. The parity bits are used by the receiver to attempt to recover from any errors that may have occurred on the wireless link.

Consider the effect of FEC on TCP goodput. On one hand, FEC can reduce the packet error rate using its error correction mechanism. Based on the TCP goodput formula, this leads to a larger achievable TCP goodput. On the other hand, part of the link bandwidth would have to be used to carry parity bits, resulting in a smaller effective channel bandwidth for the real payload.

If the effective channel bandwidth is larger than the achievable TCP goodput (obtained from the TCP goodput formula), the real TCP goodput should be well approximated by the formula. If the effective channel bandwidth is not large enough to meet the requirement of the achievable TCP goodput, the TCP sender can achieve at most the effective link bandwidth. In general, the real TCP goodput can be approximated by the minimum of the achievable TCP goodput and the effective channel bandwidth. Increasing the level of FEC redundancy increases the achievable TCP goodput but decreases the effective channel bandwidth. Since the achievable TCP goodput is a monotonically increasing function of the level of FEC redundancy while the...
effective channel bandwidth is the opposite, the TCP goodput is maximized when the effective channel bandwidth becomes equal to the achievable TCP goodput.

Due to the mobility of the mobile host and other objects in the propagation environment, the quality of a wireless link can vary significantly with time. An optimal code for a low bit error rate will likely not have sufficient correction power for high bit error rates. Similarly, the optimal code for a high bit error rate would introduce unnecessary overhead under good channel conditions. Both cases lead to a suboptimal TCP goodput. It is then beneficial to dynamically adjust the FEC redundancy level to adapt to the changing channel conditions.

In TCP-AFEC, we use the TCP goodput formula to analyze the tradeoff between the gain of the TCP goodput and the reduction of effective channel bandwidth through the application of FEC. An algorithm is provided to compute the optimum FEC code that maximizes TCP goodput.

C. Physical Layer Assumptions

For wireless links, the received signal strength is affected by three major factors: (1) path loss due to signal attenuation caused by propagation over the distance between the transmitter and receiver, (2) signal shadowing caused by the presence of large objects in the path between the transmitter and the receiver, and (3) multipath fading caused by the constructive/destructive interference effects of multiple received reflections of the transmitted signal. In this paper, we assume the presence of an average signal to interference plus noise ratio (SINR) measurement, where the averaging is over the multipath fading, at the base station. Thus, this average SINR indicates the path-loss and shadowing that the wireless link is undergoing, but does not assume knowledge of the multipath fading, which varies at a much more rapid rate. This average SINR measurement is generally assumed to be available in wireless communication systems for use in functions such as power control [23], handoff [24], and adaptive rate control [25]. For this paper, it will be assumed that this SINR is constant over a given TCP session, which would be appropriate in a wireless networking environment where the hosts are not mobile (e.g., wireless local area networks (WLANs)). Note that the mobility of objects in the environment, which leads to time-varying multipath fading, is still considered. While the applicability to systems with highly mobile hosts is conceptually similar, there would be significantly different considerations in such cases.

Given the average SINR (or, more generally, any measurement that is correlated to link quality) of the wireless link, it is critical for our work to establish the packet loss rate as a function of the wireless system parameters; in particular, it is important to be able to characterize the packet error rate as a function of the code rate and the average SINR measurement. In this paper, we will consider three cases that lend themselves to analytic formulation so that we can demonstrate the gains achievable through the proposed approach:

- **Case 1**: Independent fading from symbol to symbol within a packet, independent fading between packets.
- **Case 2**: Identical fading for symbols within a packet, independent fading between packets.
- **Case 3**: Identical fading for symbols within a packet, identical fading for packets within a TCP window.

The first case applies to systems with relatively quickly varying multipath fading and with interleaving of symbols within a packet. Symbol interleaving rather than bit interleaving is considered, since bit interleaving is generally not employed inside a code over a higher order symbol field as will be considered below. Note that the multipath fading must change at a rate of once per symbol to an independent realization for all of the assumptions of Case 1 to strictly hold, although it can be viewed as a reasonable approximation for a system with interleaving that is experiencing multipath with relatively high variability. The second case is a block fading model, which can be used for situations where the multipath fading varies at a moderate rate (approximately once per packet). The third case is a block fading model for relatively slow fading, where the fading varies over the round-trip time (RTT) of the system but can be assumed constant over a TCP window duration. Since the TCP window duration is generally very small relative to the RTT of the system, Case 3 is viewed as the most likely to occur in systems with relatively low mobility environments, such as WLANs.

In this work, we assume the use of block codes for FEC where the level of redundancy can be adjusted. In particular, we consider an \((N, K)\) Reed-Solomon code, where \((N - K)\) parity symbols are added to \(K\) data symbols to form a codeword of size \(N\). The number of information symbols per codeword, \(K\), is fixed and the code length \(N\) is varied to adjust the redundancy level of the code. Here a symbol is the basic information unit used in a Reed-Solomon code, and is composed of a certain number of bits. Assume a symbol carries \(m\) bits; then the length of the code will not exceed \(2^m\), i.e., \(N < 2^m\).

All three of the cases discussed above are simplifications of the physical layers employed in practice, as practical systems generally employ some form of trellis coding rather than simply a Reed Solomon code. However, these systems should yield representative gains, and the model can be extended to any system as long as a mapping from SINR to packet error rate can be obtained for the physical layer. This can generally be done through analysis or simulation.
Here we provide the mapping from SINR to packet error rate through analysis for the above three cases.

Case 1: When we assume independent Rayleigh fading from symbol to symbol within a packet, the SINR can be mapped to a symbol error rate through a simple analysis of the modulation; for example, if the $2^m$-ary symbols are transmitted with $2^m$-ary orthogonal frequency shift keying which is noncoherently decoded, the symbol error rate is given by [26, pg. 790]:

$$ SER = \frac{2^m-1}{T} \sum_{i=1}^{2^m-1} \frac{(-1)^{i+1} (2^m-1)}{1 + l + lSINR} $$

(2)

Analogous formula can be employed for $2^m$-QAM modulation or for the case when multiple channel symbols are combined to signal the $2^m$-ary symbol across the channel. In all cases, the frame error rate (FER) is then given by:

$$ FER = 1 - \frac{1}{N} \sum_{i=0}^{N-K/2} \binom{N}{i} SER^i (1 - SER)^{N-i} $$

(3)

Case 2: When we assume block fading from packet to packet, we must first condition on the fading value $\alpha$, calculate the PER as a function of $\alpha$, and then average over $\alpha$. For example, conditioned on the fading value $\alpha$, the symbol error rate for noncoherently decoded $2^m$-ary orthogonal frequency shift keying is given by [26, pg. 262]:

$$ SER(\alpha) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \left[ 1 - \left( \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{y} e^{-x^2} dx \right)^{2^m-1} \right] \exp \left[ -\frac{1}{2}(y - \sqrt{2\alpha ^2 SINR})^2 \right] dy $$

(4)

The frame error rate and packet error rate as a function of $\alpha$ then follow similarly:

$$ FER(\alpha) = 1 - \frac{1}{\binom{N-K}{2}} \sum_{i=0}^{N-K/2} \binom{N}{i} SER^{(\alpha)} (1 - SER)^{(N-i)} $$

(5)

$$ PER(\alpha) = 1 - \prod_{i=1}^{N_f} (1 - FER_0(\alpha)) $$

(6)

and then

$$ PER = \int_0^{\infty} PER(x) p_\alpha(x) dx $$

(7)

where $N_f$ in (6) is the number of frames of the packet and $p_\alpha(x)$ in (7) is the probability density function of the channel fading gain $\alpha$.

Case 3: When we assume constant fading over a TCP window, this introduces burstiness to the packet loss process. Analytically, the packet error rate is identical to Case 2, but now the errors are correlated. Note that the model designed in [2] is based on a similar (but different) assumption, since it assumes that all subsequent packets within a TCP window are lost if a single packet is lost. Thus, the PER from Case 2 will also be used to choose the optimal points for the proposed scheme in Case 3.

D. TCP-AFEC Protocol

In this subsection, we describe the TCP-AFEC protocol. Consider a TCP connection between a host in a wired network and a mobile host via a base station, where the wireless link is the bottleneck of the connection. In TCP-AFEC, a link layer agent is added to the base station and the mobile host respectively to enable the improvement of TCP goodput. At the wireless hop of the TCP connection, the link layer agent at the upstream node of the data flow estimates the channel BER and TCP session RTT. For each data packet passing by, the agent divides the packet into frames, computes and constructs the optimal FEC code for each frame that maximizes TCP throughput, adds appropriate fragment headers, and then transmits the frames over the wireless link. At the downstream node of the TCP data flow, the frames are assembled and delivered to the transport layer if the number errors in each frame is correctable by FEC. Otherwise, the whole packet is discarded. Note that our TCP-AFEC protocol does not attempt to retransmit the error frames here. However, the performance of a protocol that supports frame retransmission is an interesting avenue for future research. The size of TCP acknowledgement packets is very small (about 40 bytes), with FEC, these ACK packets are much less prone to bit errors than the large data packets. In this work, we assume these ack packets are not subject to errors in the network.

A typical TCP/IP header is about 40 bytes. For a packet of 1500 bytes, the typical MTU for wired LAN environment (IEEE 802.3), the header only constitutes a small portion (2.67%) of the original packet. However, if the normal fragment header operation is to be used in our approach, which fragments a data packet into small frames, the header overhead will not be negligible. For a frame size of 255 bytes, the TCP/IP header constitutes 15% of the total frame size. TCP/IP Header compression [27][28] can reduce the header size by an order of magnitude down to 3-6 bytes while yielding a performance very close to ideal case across a wide range of bit error rates. In this work, we employ the enhanced header compression technique in the fragmentation process and assume that it can achieve the ideal performance, where the correct TCP/IP header can always be
constructed for each packet.

Given the estimate of packet error rate (PER) and round trip time RTT $^1$, we can compute the achievable TCP goodput $G_f(N)$ using (1). Assume the raw link bandwidth of the wireless channel is $B_c$, the effective link bandwidth, $G_e$, is computed as the portion of bandwidth that is used to carry real payload scaled by the percentage of successful transmissions (1-PER). Since each TCP packet is fragmented into frames of size $N$, of which $K$ symbols are used for real data. The effective link bandwidth is

$$G_e(N) = B_c \frac{K}{N}(1 - \text{PER})$$

The real TCP goodput ($T_{\text{tcp}}$) is the minimum of the achievable TCP goodput ($G_f$) and effective link bandwidth ($G_e$), i.e.,

$$G_{\text{tcp}}(N) = \min(G_e(N), G_f(N))$$

The optimal Reed-Solomon code $(N_0, K)$ is the code that maximizes TCP goodput and is computed as follows.

$$N_0 = \arg \max_N (G_{\text{tcp}}(N))$$

As explained earlier, the TCP throughput is maximized when the achievable TCP throughput equals the effective channel bandwidth. Therefore, ideally $N_0$ is just the solution to the equation $G_e(N) = G_f(N)$. Note that $G_e$ is a decreasing function of $N$ while $G_f$ is an increasing function of $N$. Hence, there is a unique solution $N_0$ to $G_e(N) = G_f(N)$. However, for an $(N, K)$ Reed-Solomon code, $N$ can only take integer numbers within a certain range. For trellis-based codes, the code rate can only be chosen from an even smaller set of values, for example, rates 3/4, 2/3, 1/2, 1/3 and etc. The protocol goes through the set of available codes and find the code that yields the largest goodput.

Above we described the algorithm to compute the optimal code that maximizes TCP goodput, given packet error rate (PER) and session round trip time (RTT). In the presence of multiple TCP flows via a base station, the base station just estimates the BER and RTT for each TCP session, and then applies the algorithm to find the optimal code for each session.

Compared to other related work, one of the key features of TCP-AFEC is that the protocol takes a formula-based approach to analytically derive the optimal FEC that maximizes TCP goodput. The required modifications to implement TCP-AFEC include some link layer operations at the base station and mobile host. In this sense, it is also proxy-assisted since it requires the involvement of the basestation.

$^1$The estimation of RTT will be discussed in Section V.

Furthermore, since the modified link layer operations are transparent to the TCP at the end hosts, the end-to-end semantics of TCP is preserved.

IV. PERFORMANCE EVALUATION

In this section, we first describe the implementation of the TCP-AFEC protocol and the simulation model used in our study. We then measure the performance of TCP-AFEC through simulation and compare it to that of TCP-SACK, Snoop, and a physical layer optimization scheme. This is done for TCP-SACK and Snoop without any FEC or with a fixed amount of FEC.

A. Implementation

We implemented TCP-AFEC in ns-2 simulator[29] and used it for the following experiments. We also simulated Snoop in ns-2.

Upon receipt of TCP data packets from the transport layer, the link layer TCP-AFEC agent computes the optimal code that maximizes TCP goodput using the algorithm described in Section III. The original packet is then fragmented into frames and corresponding compressed TCP/IP header and parity bits are added to the original data. These frames are then transmitted over the wireless link in order. When the TCP-AFEC agent receives frames from the channel, it assembles the frames belonging to one packet and delivers the packet to the transport layer if the errors are correctable. In the case that any of the frames in a packet has errors beyond the correction power of the code, the whole packet is discarded.

B. Simulation Model

A typical wireless data application includes data flows between a fixed host in the wired network and a mobile host via a base station. Without loss of generality, we consider the case that the data source is a fixed host in the wired network. The path from the fixed host to the base station is modeled as a link with a bandwidth of 10 Mbps and one-way propagation delay of 40 ms. Since our focus is on the erroneous nature of wireless links, we assume there is no loss on the wired link and the buffer at the base station is large enough that there is no buffer overflow. The wireless link is modeled as an erroneous link with bandwidth of $B$ bps and delay of $d$ ms. We assume the bottleneck of the
TCP session is not in the wired part, but lies on the wireless link, i.e., $B < 10Mbps$. The above simulation model is shown in Figure 1. For case 1 of Section III-C, which will be considered in detail here, the symbol errors are assumed to form an i.i.d. Bernoulli process with an average rate of $SER$, when conditioned on SINR. Analysis of the other two cases can be done in a similar manner. Although the simple model does not contain all of the characteristics of the wireless networks, it captures the essential features of the network and we expect the performance results to be representative of more complicated systems.

In this work, we consider two different types of wireless links. The first one represents an indoor wireless local area network (WLAN), such as IEEE802.11 [30]. For this type, we choose a typical bandwidth of 2Mbps, a one-way propagation delay of 4ms, and a packet size of 1460 bytes. The second wireless link type tries to capture the characteristics of a wireless wide area network (WWAN), such as CDPD, GPSR, and etc. Compared to WLAN, the WWAN scenarios have a smaller bandwidth and larger round trip time. For example, CDPD offers a bandwidth of only 19.2 Kbps, while GPSR offers a higher bandwidth of 100 Kbps. The round trip time in these WWAN networks can vary from 10s of milliseconds to seconds. In the following simulations, we choose the bandwidth to be 19.2 Kbps and delay to be 200 ms. The packet size is set to be 512 bytes.

C. Comparison with TCP-SACK, Snoop and a physical layer optimization scheme

In the following we validate our TCP goodput analysis and then compare the goodput of TCP-AFEC to those of TCP-SACK, Snoop, and a physical layer optimization scheme through simulation. This is done for both types of the wireless links described above. In the experiments, a TCP source at the fixed host has an infinite amount of data to send. Each scenario was simulated for one hour and the average TCP goodput was then measured for the duration. TCP-SACK was used as the baseline TCP protocol for comparison since it is gaining increasing share in the current Internet. Also TCP-SACK has been shown to be quite effective in dealing with a high packet error rate in wireless networks [3], and substantially improve the TCP goodput over TCP-RENO. In the experiments, we set the symbol length to be 8 bits, i.e., a byte. The number of data bytes for each frame is chosen to be 175 bytes. We use a compressed TCP/IP header of 5 bytes for each frame. The total frame size $N$ can vary from 180 to 255 bytes. The experiments were conducted for a wide range of symbol error rates, $8 \times 10^{-6}$, $8 \times 10^{-5}$, $8 \times 10^{-4}$ and $8 \times 10^{-3}$. The range of symbol error rates is typical for wireless links, ranging from good to bad conditions.

![Fig. 2. Analytical and simulation results of TCP goodput for different frame sizes](image)

**C.1 WLAN Scenarios**

**(a) Analysis v.s. Simulation**

Figure 2 depicts the goodputs of TCP-AFEC obtained through ns simulation and analysis (equation (9)) for the symbol error rate of $8 \times 10^{-3}$. In the analysis, the real TCP goodput is derived as the minimum of the achievable TCP goodput and effective link bandwidth. The optimal code that maximizes TCP goodput predicted by (10) is highlighted by an arrow in the graph. From the figure, we observe that the analysis matches simulation results very well for all different frame lengths. Moreover, our algorithm yields very good predictions for the optimal codes that maximize TCP goodput. This is true for all the symbol error rates in the experiments.

The TCP goodput exhibits a two-phase behavior in the figure, which can be explained as follows. When the redundancy level is low, the packet error rates are not reduced enough and the TCP achieves a goodput predicted by (1), even though the effective link bandwidth allows more. As the level of redundancy increases, the packet error rate is further reduced, leading to a larger TCP goodput. After a certain point, the achievable TCP goodput will grow larger than what the effective link bandwidth can afford. In this case, the actual TCP goodput is determined by the effective link bandwidth. The goodput starts to decrease inversely to $N$, as the effective link bandwidth follows inversely to $N$. In each of the two phases, TCP goodput is dominated by a specific factor. Before the peak point, the TCP goodput is determined by the limitations in the TCP protocol, predicted by equation (1); while after the peak, it is determined by the effective link bandwidth. This two-phase behavior of TCP verifies our initial discussion of the tradeoff between
the achievable TCP goodput and effective link bandwidth in Section III.

(b) Comparison with TCP-SACK and Snoop without FEC

Figure 3 compares the goodput of TCP-AFEC, TCP-SACK and Snoop as a function of symbol error rate. Here no FEC is used in TCP-SACK and Snoop protocol. Comparison of TCP-AFEC with TCP-SACK and Snoop of fixed coding follows shortly after in this section. The symbol error rates, ranging from $10^{-5}$ to $10^{-2}$, produce packet error rates ranging from $1.2\%$ to $99\%$ TCP-SACK and Snoop.

For a symbol error rate of $10^{-6}$, TCP-SACK achieves an average goodput of 756.3 Kbps, while TCP-AFEC and Snoop yield a similar goodput around 1.8 Mbps, an improvement of $140\%$ over TCP-SACK. Note that in this case both TCP-AFEC and Snoop achieve a goodput close to the link bandwidth of 2 Mbps. As the symbol error rate increases to $10^{-5}$, the packet error rate for TCP-SACK and Snoop grows to $11.3\%$. In this case, the goodputs of both TCP-AFEC and Snoop drop drastically to 146 Kbps and 492 Kbps, respectively, while Snoop still outperforms TCP-SACK by a large margin. For the same symbol error rate, the goodput of TCP-AFEC only drops slightly from that for symbol error rate $10^{-6}$, providing an eleven-fold gain over TCP-SACK and nearly a three-fold gain over Snoop. When the symbol error rates become even higher ($8 \times 10^{-4}, 8 \times 10^{-3}$), the transmission of both TCP-SACK and Snoop stalls due to the TCP congestion control. The packet error rates become so large that Snoop is not able to provide improvement over TCP-SACK. Our preliminary results show that in these cases, even though Snoop is able to significantly reduce packet loss rate over the wireless link by doing local recovery, it substantially increases the TCP RTT at the same time. The prolonged RTT over-cancels the gain by reducing the packet loss rate, resulting in a poor performance.

(c) Comparison with TCP-SACK and Snoop with fixed FEC

In the above comparison, no FEC is employed by TCP-SACK and Snoop. Now we investigate the performance of TCP-SACK and Snoop with a fixed FEC. At the wireless link hop, the TCP data packets are fragmented and made into fixed FEC frames at the link layer of upstream node. The frames are then transmitted over the wireless link and assembled at the downstream node.

In [31], the authors proposed two different redundancy levels in their adaptive FEC policy to improve wireless
channel efficiency. One of the two codes has a 5% coding overhead while the other has about 30% coding overhead. We used the same amount of coding overheads in our study to illustrate the effect of fixed coding overhead on the performance of TCP-SACK and Snoop protocol. The corresponding Reed-Solomon codes of 5% and 30% coding overheads are (175, 190) and (177, 255), respectively. These two coding overheads represent light and medium levels of code redundancy.

The goodput of TCP-SACK and Snoop protocol under the two fixed coding overhead are compared with that of TCP-AFEC in Figure 4. From the Figure 4 (a), we observe that the 5% overhead FEC significantly improves the goodput of TCP-SACK for all four symbol error rates. It also improves the performance of Snoop except for the symbol error rate $8 \times 10^{-3}$, where the goodput of Snoop slightly decreases from the case without FEC. The goodputs of TCP-SACK and Snoop for symbol error rates $8 \times 10^{-6}$, $8 \times 10^{-5}$, $8 \times 10^{-4}$ are now only about 6.7% smaller than those of TCP-AFEC. However, for a symbol error rate of $8 \times 10^{-3}$, TCP-AFEC still outperforms Snoop and TCP-SACK by 13.7% and 78.5%, respectively. This is because the packet error rate of TCP-SACK and Snoop is still 3.5% after the application of 5% overhead FEC. The packet errors cause a substantial performance degradation for TCP-SACK. Snoop is able to provide considerable improvement, but still falls below TCP-AFEC. For symbol error rates from $8 \times 10^{-6}$ to $8 \times 10^{-4}$, the 5% overhead FEC reduces the packet errors rates to such small values that goodputs of TCP-SACK and Snoop have reached the limit of effective link bandwidth.

The goodputs of TCP-SACK and Snoop under 30% overhead FEC are shown in Figure 4 (b). In this case, the redundancy level is large enough to account for the high symbol error rate $8 \times 10^{-3}$. However, the large coding overhead is an overkill for the low symbol error rates, resulting in smaller goodputs than those under 5% overhead FEC.

This example well illustrates the detrimental effect of using fixed FEC to improve TCP performance over a time-varying wireless channel. A light overhead FEC good for small symbol error rates may not have sufficient error correction capability for large symbol error rates. Similarly, FEC with high redundancy level may end up paying too much for a low error environment. In TCP-AFEC, the protocol continuously monitors the channel condition and chooses the appropriate FEC code that maximizes TCP goodput. Therefore, the goodputs of TCP-AFEC is very stable across the wide range of symbol error rates.

\textit{(d) Comparison with a physical layer optimization scheme}

Without the knowledge of TCP performance characterization, a physical layer scheme can only try to optimize a quantity that is understood by the physical layer. Consider the coding overhead and bandwidth wasted by corrupted packets, a reasonable quantity that a physical layer optimization scheme may want to maximize is $B_{CN}(1 - PER)$, the effective link bandwidth that is used to successfully transmit real data. Recall TCP-AFEC maximizes $\min(G_c, G_f)$, where $G_f$ represents the achievable TCP goodput limited by the TCP congestion control mechanism, and $G_c$ represents the effective link bandwidth limited by the channel bandwidth and FEC overhead. Basically the physical layer optimization scheme tries to maximize only the first term of the quantity TCP-AFEC optimizes.

The comparison of TCP-AFEC and the physical layer optimization scheme is shown in Figure 5. Note for small symbol error rates ($< 8 \times 10^{-4}$), the physical layer optimization scheme achieves the same goodputs as TCP-AFEC. However, for large symbol error rates, ($8 \times 10^{-3}$, $8 \times 10^{-4}$), TCP-AFEC well outperforms the physical layer optimization scheme. In this large symbol rates region, the physical layer optimization scheme chooses smaller $N$ values than TCP-AFEC does. This is because the quantity it tries to maximize is now dominated by the term $K/N$, where $N$ in the denominator plays a critical role. In this case, the TCP cannot achieve the goodput of $G_c$, predicted by the physical layer optimization scheme; the actual TCP goodput is limited by $G_f$, what TCP congestion control mechanism allows. TCP-AFEC, on the other hand, considers both $G_c$ and $G_f$ and is able to choose a FEC that maximizes the actual TCP goodput.

This example further demonstrates the benefit of combining TCP performance characterization with link layer er-
ror control scheme to improving TCP goodput over wireless networks. Without TCP performance knowledge, the physical layer optimization scheme yields smaller goodputs than TCP-AFEC for wireless channels ranging from medium to bad conditions.

C.2 WWAN Scenarios

For the WWLAN scenarios, we compared the performance of TCP-AFEC to those of TCP-SACK, Snoop and the physical layer optimization scheme for the same range of channel condition as for WLAN scenarios. This was done for TCP-SACK and Snoop with 0 %, 5 % and 30 % coding overheads. For all these WWAN scenarios, we obtain similar results as for WLAN scenarios and draw the same conclusions regarding the relative performances of TCP-AFEC, TCP-SACK, Snoop, and the physical layer optimization scheme. Due to the page limit, the figures are not included in this paper; they can be found in the extended technical report of this work [32].

The above types of wireless links represent high-bandwidth low-delay and a low-bandwidth high-delay scenarios. In addition to these types of networks, we also simulated wireless networks with medium bandwidth and delay. We found that the same observations hold for those scenarios.

V. DISCUSSIONS

In this section we present some of the implementation and practical issues of the TCP-AFEC protocol and discuss their implications to the protocol.

A. Round Trip Time (RTT) Estimation:

In TCP-AFEC, the upstream link layer agent at the wireless hop requires an estimate of the TCP session round trip time (RTT) to carry out the algorithm. Basically the RTT value is needed by the TCP throughput formula to compute the achievable TCP throughput.

In TCP, the estimate of RTT is conducted at the sender. The sender records the time at which a packet is sent and the time the corresponding acknowledgement arrives. The elapsed time between the above two times is then computed as a RTT sample. The average RTT value and the variance are maintained as the weighted running average of the samples and these two quantities are used to help set appropriate timeout values for subsequent packets.

If TCP is initiated by the mobile host. The TCP sender at the mobile host would have the RTT estimates. This information can be sent to the link layer agent in the same host by providing some type of interaction between the transport and link layer in the protocol stack.

However, for TCP connections from wired network to mobile host, the upstream link layer agent at the basestation is not able to measure the session RTT samples based on the information of packets passing by. We propose two solutions. The first one has the TCP sender embed the current RTT estimate in the OPTIONS field of the TCP header. For each data packet, the TCP-AFEC agent simply retrieves the average RTT value from the TCP header and uses it for the computation. The other solution involves active probing using ICMP from the basestation. The RTT between the basestation and the server can be estimated based on the RTT of the probing packets. The RTT between the basestation and the mobile host can be estimated in the same manner as TCP sender measures the session RTT, since the TCP-AFEC agent at the basestation can record the time of that a packet arrives at the basestation and the time its corresponding acknowledgement returns. The session RTT can then be approximated as the sum of the RTT’s of the two segments.

B. TCP Packet Size Selection

In TCP-AFEC, original TCP data packets are fragmented into frames, each containing a fixed size of data (K data symbols). If the original data packet size is not an exact multiple of size K, the last frame will contain less than K data symbols. This leads to a smaller effective link bandwidth, hence a smaller TCP goodput. In extreme cases, this packet size effect can be very harmful to the TCP performance. In the extended technical report of this work [32], we provide an example to illustrate the packet size effect on TCP goodput. We also propose a solution that helps the TCP sender to select an appropriate packet size.

VI. CONCLUSIONS

In this paper, we presented a novel technique (TCP-AFEC) to improve TCP performance over wireless networks using adaptive forwarding error correction. The protocol integrates the TCP performance characterization at the transport layer with link layer error control schemes. A well-established TCP goodput formula is combined with the error correction power of FEC to select the optimal code that maximizes TCP goodput for different channel conditions. We analyzed the tradeoff between the gain of TCP goodput and the degradation of effective channel bandwidth by applying link layer forward error correction. An algorithm is provided to compute the optimal code for maximum TCP goodput.

The benefit of combining TCP performance characterization and link layer FEC is validated by comparing the performance of TCP-AFEC to those of TCP-SACK, Snoop protocol and a physical layer optimization scheme. Simulation results show that TCP-AFEC can effectively im-
prove TCP goodput over TCP-SACK, Snoop and the physical layer optimization scheme for a wide range of wireless channel conditions.

Compared to other related work, one of the key features of TCP-AFEC is that the protocol takes a formula-based approach to analytically derive the optimal FEC that maximizes TCP goodput. The required modifications to implement TCP-AFEC include some link layer operations at the base station and mobile host. In this sense, it is also proxy-assisted since it requires the involvement of the basestation. Furthermore, since the modified link layer operations are transparent to the TCP at the end hosts, the end-to-end semantics of TCP is preserved.

REFERENCES