Bit-Error Recovery with Adaptive Packet Sizes for Wireless Network Environments

CS261 Final Project
Final Paper

Shawn Hsiao and Adon Hwang
{shawn, adon}@eecs.harvard.edu

January 15, 2000

Abstract

TCP suffers from poor performance under error-prone wireless environments. The network research community has explored a wide range of solutions for this situation, including link-layer solutions, end-to-end solutions, and other hybrid solutions.

We propose a simple but novel scheme of adaptive packet sizing at the transport layer. We argue that adaptation is necessary in a changing environment where static configuration of error control does not yield the best performance for many situations. We propose a multiplicative-decrease/additive-increase packet sizing protocol in TCP, given error feedback by the network core and peers. We present simulation results for our scheme.

1 Introduction

The future of computing is mobile and wireless. A parallel trend in networking is the convergence of network protocols toward IP. We can envision the proliferation of compact mobile computers equipped with wireless network interfaces conversing with each other using IP. However, existing TCP/IP protocols do not perform well under mobile wireless environments, due to the inherently high bit error rates.

This paper examines a novel mechanism to increase performance in error-prone environments. To deal with high bit-error rates, we find that reducing the packet error rate by dynamic packet sizing is a fundamentally effective solution. We combine this and prior work of explicit error notification by the intermediate routers. This ultimately reduces the overhead of retransmitted bytes from the sending host.

We have implemented, in the `ns` network simulator [MF00], an adaptive algorithm similar to TCP's congestion control and avoidance algorithms of multiplicative decrease additive increase. We investigate performance across a range of bit-errors through several metrics such as the end-receiver's goodput, data overhead caused by smaller packets and retransmissions, and the actual packet loss rate. We also attempt to find well-suited parameters for adaptive packet sizing. Our results show that a simple end-to-end scheme can be an effective mechanism to boost TCP performance, with some minor tradeoffs. TCP with adaptive packet sizing can survive under harsh bit errors (10^{-3}), where traditional TCP performs rather poorly due to high packet error rates. We argue that our proposal is a viable alternative for improving TCP performance.

2 Background

2.1 TCP and Bit Errors

This section discusses the basic features of TCP [Pos81] and [Ste94]. TCP is a windowed flow control protocol. This means that the sender maintains a window of unacknowledged packets in flight to the receiver. This allows the sender to utilize the capacity of the links, maximizing throughput. Otherwise, for each packet, the sender would have to wait for an acknowledgment from the receiver before sending a new data packet (this is called stop-and-go). The size of the window is governed by end-to-end flow control feedback. Only the receiver participates in the acknowledgment of sender's packets. The acknowledgments are cumulative, which means that the receiver tells the sender the highest in-sequence packet it has received. A packet dropped in an intermediate router is a signal of congestion– the router ran out of buffer space that would hold the incoming packet. The sender detects this signal by a lack of acknowledgments for a period of time – a time-out – or by a certain number of duplicate acknowledgments triggered by a hole in the packet sequence. The obvious thing for the sender to do at this point is to shrink the window size. The higher the congestion probability, the smaller the window size. TCP incorporates a fast retransmit capability when it deems that a packet retransmission is necessary and not redundant for the receiver. This mechanism requires a sufficient window size to work effectively.

The intermediate routers detect bit errors in a packet by the use of checksums. Corrupted packets are useless for the end hosts because TCP/IP has no mechanism to recover from individual bit errors. Thus, routers or hosts sensibly drop a packet silently and let the sender retransmit the undelivered packet. Since the sender attributes all losses to congestion, it applies the congestion control algorithm accordingly to the transmission.

TCP performs miserably under environments that have high packet error rates. The higher the packet error rate, the more "holes" in the stream of data. To recover from the holes, TCP either needs to wait for timeouts, which reduce throughput dramatically, or wait for duplicate ACKs. The high packet error rate also reduces the congestion window size– there are fewer packets in flight – and decreases the chances to get duplicated ACKs because there are few packets in the window in the first place. So, if the packet error rate is higher than a certain level, it is possible that the throughput is almost zero, regardless of congestion.

Under high bit error environments, the congestion control algorithm may unnecessarily hurt TCP's performance. The only way to solve this problem is to decouple the logic of congestion control from the error recovery logic.

2.2 Related Work

The bit error rate of a typical wireless network may be as high as 1 bit error in 10^{-5} bits, which is about 100 times greater than a wired network (10^{-7}) [CWKS97], [Gil60]. There has been much work to address the poor performance of TCP in this environment.

One fundamental way is to decouple the control logic of congestion from the error recovery logic.

TCP decoupling[Wan99] builds a trunk (an aggregation of many TCP connections into one), treats user TCP packets as payload of the trunk and inserts virtual headers which are additional small packets, just before the user packets. TCP decoupling requires that intermediate routers drop those virtual headers instead of data packets when they have to drop packets for lack of buffer space or other reasons. The trunk performs congestion control when the virtual headers get dropped but not for dropped user packets. Also, due to the small size of the virtual headers, virtual headers are less likely to have bit errors. This decoupling approach can increase TCP performance in high bit error environments. The trunk idea may not be appropriate for normal

wireless applications, but the decoupling approach can effectively increase the performance of TCP.

In Snoop, the authors modify wireless bridges to insert logic to help TCP performance [BPSK97] and [BK97]. When a bridge detects bit errors in a packet, it notifies the sender who retransmits it directly. The protocol enables the network core to perform packet loss signaling, which they call Explicit Loss Notification (ELN). The protocol provides a way to recover lost packets as early as possible, which reduces the number of timeouts and improves performance.

Another way to improve TCP is to speedup retransmission of lost packets by generating hints from receiver's side. TCP header option Selective Acknowledgments [MMFR96] takes this approach by letting the receiver explicitly mark in the returning ACKs the holes of unreceived packets. This enables the sender to recover from multiple packets lost, without waiting for the sender to timeout and retransmit.

Another approach is to reduce the packet loss rate instead. Some have proposed schemes to do dynamic packet sizing at the link layer [Mod99], [ES98] by predicting the bit error rate of the link. Smaller packet size results in lower packet error rate under uniform bit error rate model.

Our scheme is not similar to any one mentioned above, however, it coincidentally contains part of them.

3 Design Issues

We focus our scheme on two parts. We believe that the combined solution can effectively improve TCP performance under wireless environments.

The first part is Explicit Error Notification (EEN). Like Snoop and SACK, this scheme tries to recover from packet loss as soon as possible. Also like Snoop, it requires cooperation from intermediate routers. Instead of notifying sender directly like Snoop, the routers mark the corrupted packets and forward them to the receiver. Some packets may have corruption in the header and may lack enough information for forwarding. These get silently discarded.

The second part is to do Multiplicative Decrease and Additive Increase on the packet size. Over time, the error model of a wireless network is close to constant bit error rate [CWKS97], so the smaller the packet, the smaller the probability of packet corruption. But making the packet size small also decreases the efficiency since the headers occupy a larger portion of the packet. An adaptive scheme according to the error rate must be employed to improve efficiency.

We will look into details of some design issues in the following subsections.

3.1 Multiplicative Decrease and Additive Increase

TCP does Multiplicative Decrease and Additive Increase (MDAI) control over the window size to reflect the change in bottleneck bandwidth. Much prior work has shown that MDAI provides fairness and robustness for competing network flows, by probing the bottleneck bandwidth.

With Multiplicative Decrease and Additive Increase of window size, when the sender detects congestion, it reduces the window size multiplicatively (one half in TCP), and otherwise it increases the window size gradually (one packet per round-trip time). One important property of the algorithm is aggressive retreat and conservative advance. This is an important characteristic for fair and robust probing when the uncertainty regarding the network conditions is high.

The bit error rate (BER) experienced in wireless networks may also change over time, like the available bandwidth above. Because of the high packet loss rate, TCP may suffer under the change of bit error rate over time. TCP cannot efficiently recover from high packet loss rates. Proposals such as selective acknowledgments [MMFR96] address this problem.

Working from this prior experience with congestion control in TCP, we propose the use of MDAI to change packet sizes dynamically and to probe for bit error rates. This approach can effectively control the packet error rate (PER), which for TCP, results in better performance in wireless networks.

3.2 Error Model

In wireless environments, packets transmitted over the air may have bit errors introduced. If we assume that the bit error rate is constant over time or over a bit stream, then the packet error rate grows exponentially when the sender increases the packet size:

$$\text{PER} = 1 - (1 - \text{BER})^{(\text{length of packet in bits})}$$

Of course, an invariant bit error model over time or bit stream may not necessarily reflect the real world. However, this model can reveal the relationship between PER and BER.

For this study, we assume that bit error rate is constant over time, so the distance between two consecutive bit errors is exponentially distributed over the bit stream. This model is the prevalent one in the literature [CWKS97], [Gil60].

3.3 Bit Errors in Header and Data

We assume that upon detection of bit errors, the intermediate routers will mark the packet as corrupted and will continue to forward packets to the end receiver. In the event of header corruption, the router cannot rely on the information carried in the packet header to forward the packet correctly and must discard the packet. However, an error in the data does not impact the routing decision. Such a corrupted packet can provide both the sender and receiver with useful information that they use to adapt to the environment.

According to the error model, the larger the packet size the higher the chance of its corruption. The relative probability of header corruption compared to data corruption, is low. This is because, for TCP, the typical header is 40 bytes while the data may range from a couple hundred of bytes up to the maximum segment size (MSS), usually 1500 bytes. The data payload of a packet is 3-400 times longer than the packet header; by the relationship of PER and BER, given a fixed bit error rate, the probability of header corruption is rare.

3.4 Forward EEN and Backward EEN

In our scheme, the EEN (Explicit Error Notification) we used is a forward EEN approach. An alternative Backward EEN method deserves some discussion here.

First, under Forward EEN, the routers mark packets with bit errors along the forwarding path to the receiver. The receiver will mark the corresponding acknowledgment (ACK) to notify the sender that it should retransmit the corrupted packet (instead of interpreting the absence of an ACK as a signal for congestion).

Routers under Backward EEN, on the other hand, send a notification packet to the sender when the corrupted packet is detected. This signal does not propagate to the corresponding receiver. Snoop protocol is a typical backward EEN example.

We make some analogies to Sally Floyd's work in comparing forward and backward congestion notification [Flo94]:

- Forward marking will not trigger sending extra packets that subsequently are corrupted as well. If a packet is corrupted, chances are subsequent packets are also corrupted, due the high bit error rate of the link at that given moment.

- Backward marking is more responsive than forward marking. (The sender does not have to wait for the receiver to reflect this information back.)
- Since backward marking uses an extra, special packet, it can be detected by the other routers along the path from sender. These routers may take additional action accordingly.

We prefer to use forward marking in our scheme, because the end-to-end control semantics state its flexibility and clear design, especially in minimal involvement by intermediate routers. The receiver may also want to collect information about the links, because it will most likely use the link to send back data and acknowledgments.

3.5 Dynamic Sizing on Link Layer or Transport Layer

We choose to perform dynamic packet sizing at the transport layer instead of at the link layer as previously proposed [Mod99], cite steenkist. We invoke end-to-end control arguments to support our transport later implementation.

End-to-End arguments state that functionalities of reliable transmission should be put as close to applications as possible, because, they may have some applications that do not need the particular functionality. This is the central motif in IP networking. Also, to gain scalability and flexibility, higher layer protocols should not rely on the lower layers for such functionality transparently.

More precisely, even if the packets sent over the link layer are further fragmented into smaller chunks, chances are that some of those fragments may get corrupted in transit. The transport layer will not be able to fully reconstruct the original packet, so the retransmission responsibility ultimately lies with the higher layer.

Moreover, the link-layer approach introduces more latency and complexity to implement in the link hardware. However, an end-to-end adaptive scheme suffers from delayed feedback.

4 Adaptive Packet Sizing Algorithm

Our algorithm performs adaptation at the transport layer. It is end-to-end and adaptive as discussed above:

- Marking: Intermediate routers discover an incoming packet's corruption by the failure of the link-level checksum. If the IP packet header is not corrupted, the router simply forwards the packet toward the destination (after optionally marking the EEN flag in the packet).
- EEN Signaling: The receiver host discovers the router's EEN notification by the failure of the TCP checksum or by an optional flag in the packet header. The receiver then sets the EEN echo flag in the corresponding acknowledgment packet. This separate and explicit flag is necessary since the ACK packet itself may experience corruption during its return.
- Sender Reaction: Upon receiving the marked ACK packet, the sender host immediately retransmits the missing packet. The sender then reduces the TCP MSS (maximum segment size) by an adaptive factor (multiplicative decrease, but additive decrease may also apply).
- Sender Action: When it receives ACK packets for new data, the sender opportunistically increases the TCP MSS by a second adaptive factor (additive increase, but multiplicative increase may also apply).

Note that the marking of EEN flags, which is proposed by prior work such as ELN [BPSK97] and the retransmission decision of EEN-marked packets is orthogonal to the dynamic packet sizing.

5 Simulation Results

5.1 Setup

We have implemented our adaptive scheme and EEN in the UCB/LBNL/VINT network simulator [MF00]. All results are for 10 simulated minutes averaged for 3 runs with different random number seeds. The variance between runs is statistically insignificant. There is a single TCP sender on a drop-tail access link of 10 Mbps entering a error-prone drop-tail bottleneck link of 1 Mbps. The router queues are sufficiently large that drops due to buffer overflow do not occur unless the error rate is zero. The end-to-end propagation delay is uniformly randomized between 45 ms and 55 ms. The congested router corrupts incoming packets in both directions, with an exponential byte probability distribution. If the corruption is within the TCP/IP header, the router silently discards the packet. If the corruption is beyond the TCP/IP header, the router marks the EEN flag within the TCP header and forwards the packet toward the receiver. The receiver, then, echoes the EEN flag to the sender in the corresponding ACK. Upon receipt of the EEN-flagged ACK, the sender retransmits and takes adaptive action (multiplicative decrease) on the size of subsequent packets. When the sender receives non-EEN-flagged ACK packets from the receiver, it takes opportunistic adaptive action (additive increase) on the size of subsequent packets.

We compare adaptive packet sizing TCP with unmodified TCP and with TCP-EEN which only includes EEN notification and not packet sizing, to determine the exact effects of packet sizing.

We note that determining the exact adaptive parameters is somewhat of a black art, as with many other such configuration systems. The figures presented in this section present one of the better-behaving parameters. There can be poor configurations, but any adaptive scheme performs better than non-adaptive TCP.

5.2 Packet Size

Figure 2 shows samples of the average packet size over time. We subsequently refer to MDAI parameters by i/j, where i is the multiplicative decrease factor, and j is the additive increase factor. In this subsection, we present results for MDAI parameters 2/1 and 8/128, at a $1 * 10^{-5}$ bit error rate, for a 30-second portion of a 100 second simulation.

When the packet size shrinks, it is due to detection of a corrupted packet. The packet size increases (gradually or suddenly depending on the parameter), for ACKs of non-corrupted packets. We bound the maximum and minimum packet sizes to 1500 and 128 bytes respectively, which means that the packet size cannot grow larger or shrink smaller than the limits. These maximum and minimum packet sizes may not be optimal– 1500 bytes is due to the maximum Ethernet frame size and 128 bytes is set to retain sufficient efficiency in terms of header overhead.

We compute average packet size by dividing the total number of bytes sent by the total number of packet sent, in intervals of 100 ms. Timeouts cause discontinuous interruptions, since the sender sends no packets during that time.

Note that the choice of parameters influences the sender's behavior. A conservative parameter such as 2/1 results in a stable average packet size over time, while an aggressive parameter as 8/128 results in quick changes. From our experience, conservative parameters perform better in terms of packet loss rate and data transmission overhead. Aggressive parameters maintain a large average packet size that approaches the behavior of non-adaptive fixed-size packet cases. Aggressive decrease parameters result in packet sizes too small; aggressive increase parameters result in packet sizes too large, compared to what packet size is appropriate for the error rates.

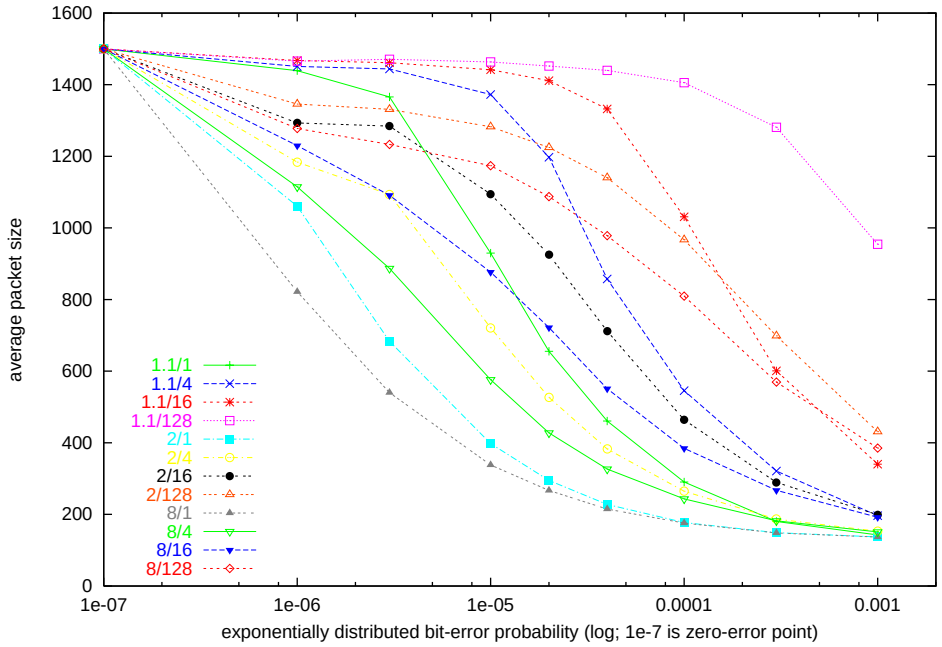


Figure 1: Average packet size for adaptive packet sizing: i/j where i is the multiplicative decrease factor; j is the additive increase factor.

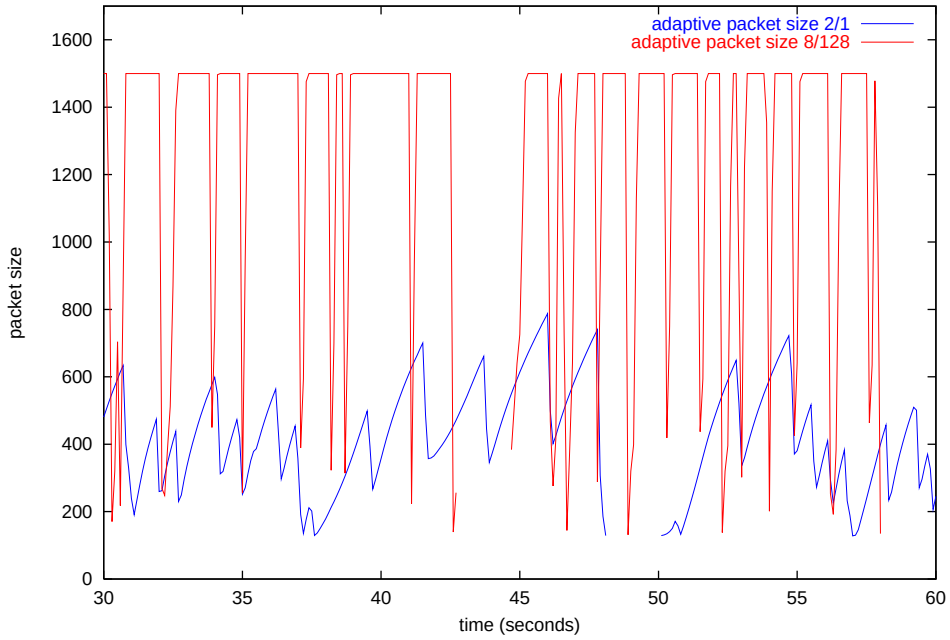


Figure 2: Average packet size change over time.

5.3 Packet Loss Rate

Our adaptive scheme can control the packet loss rate within a range acceptable for TCP. The PER is as low as 12 percent even under high bit error rates as 10^{-3} (on average 1 bit error in every 1000 bits!).

It is known for TCP that the packet loss rate influences the throughput negatively:

$$W \leq \sqrt{\frac{8}{3p}}$$

where W is the window size and p is the packet loss rate [FF99].

Note that this relation does not involve the bit error rate of the communication channel. By controlling the packet loss rate, we decrease TCP's chances of falling into timeout.

We compare our scheme to Snoop [BPSK97] that performs backward error notification. We adapt the Snoop approach (backward notification) to a forward one where the receiver reflects the error signal. However, instead of adjusting the packet sizes, the sender blindly retransmits corrupted packets.

It is clear from figure 3(b) that our scheme improves the packet loss rate compared to a Snoop-like mechanism (TCP-EEN). We choose to compare one of the better MDAI parameters with the various fixed-size schemes of unmodified TCP and TCP-EEN. Adaptive packet sizing with MDAI parameter 2/1 yields 12% packet loss; EEN with 1500 byte MSS yields 54% packet loss under bit error rate as high as 10−3. Note that the packet error rates do not differ between EEN-enabled TCP and unmodified TCP. EEN's advantage is in increasing TCP goodput as we show later, not in reducing packet error rates.

5.4 Data Transmission Overhead

In the course of retransmitting a corrupted packet there is overhead incurred by the sender in this duplicate transmission. By adapting to the bit error, we lower the probablilty of packet corruption and hence, lower the probability of retransmission. We measure the overhead of retransmission:

As shown in figure 4, overhead of either TCP or TCP with only EEN increased dramatically when the BER increased. This occurs because the two implementations continue to send packets while they disregard the link condition. Our scheme adopts to different BERs well and performs better than either of the two implementations.

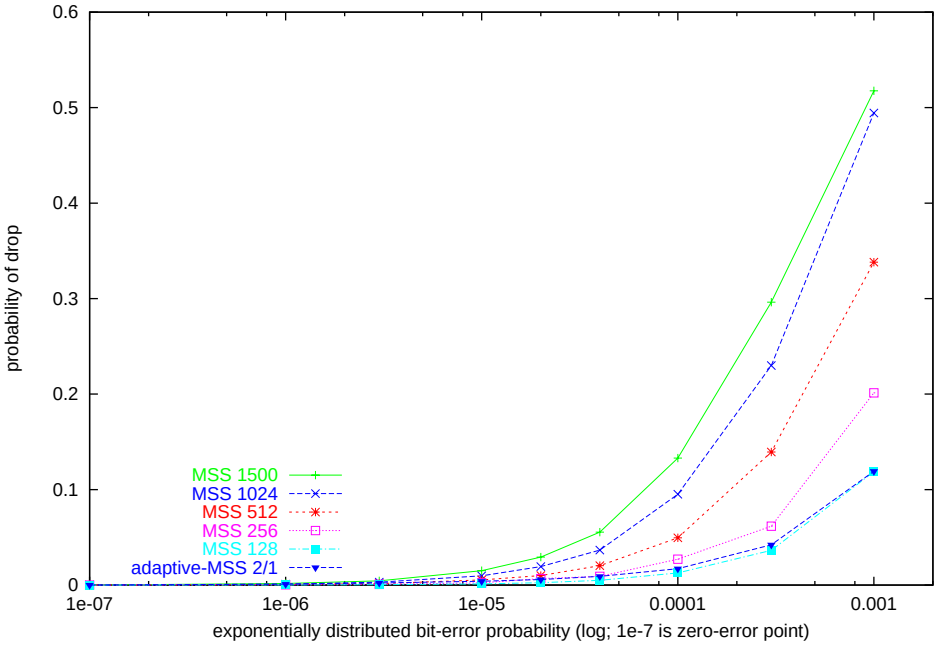
Adaptive MSS with MDAI parameter 2/1 consistently performs with the least overhead (higher the number the smaller the overhead). Again, overhead with unmodified TCP and TCP-EEN do not differ dramatically.

We note that lowering the data transmission overhead is valuable for mobile hosts that have strict power consumption constraints. By reducing the total amount of data transmitted, such hosts may also reduce the power consuption by the associated machine resources.

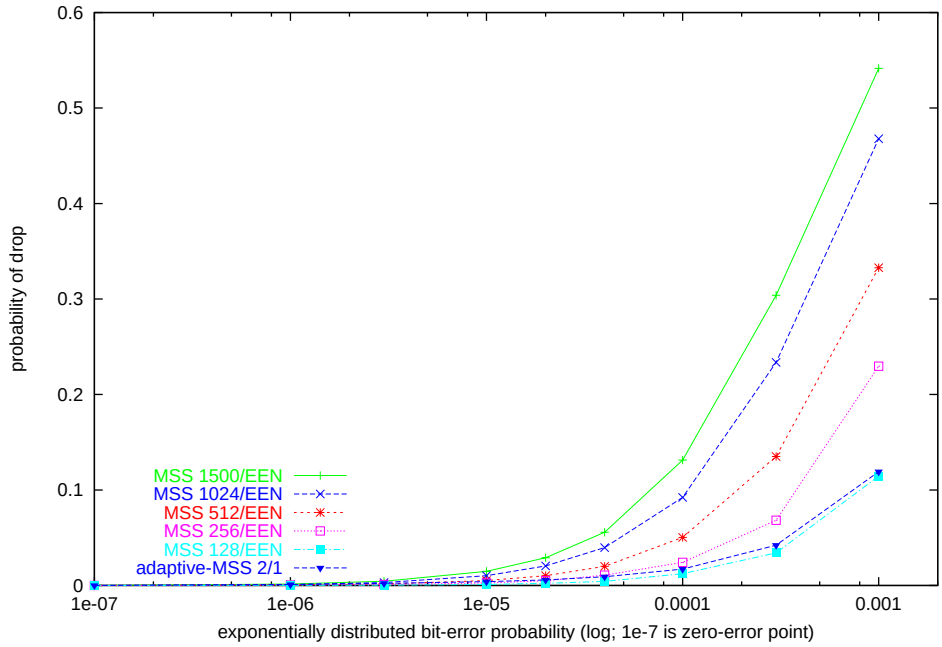
5.5 TCP Goodput

Goodput, defined as the number of TCP bytes received excluding redundant data received, is a common metric to measure end performance. It is clear that our scheme does not, in of itself, provide much improvement beyond what EEN-induced retransmissions already provide.

This is caused by fact that we merely shrink the packet size, without updating the congestion window size to maintain a certain level of data throughput. We are working toward a proper way of updating congestion window size without violating any TCP properties.

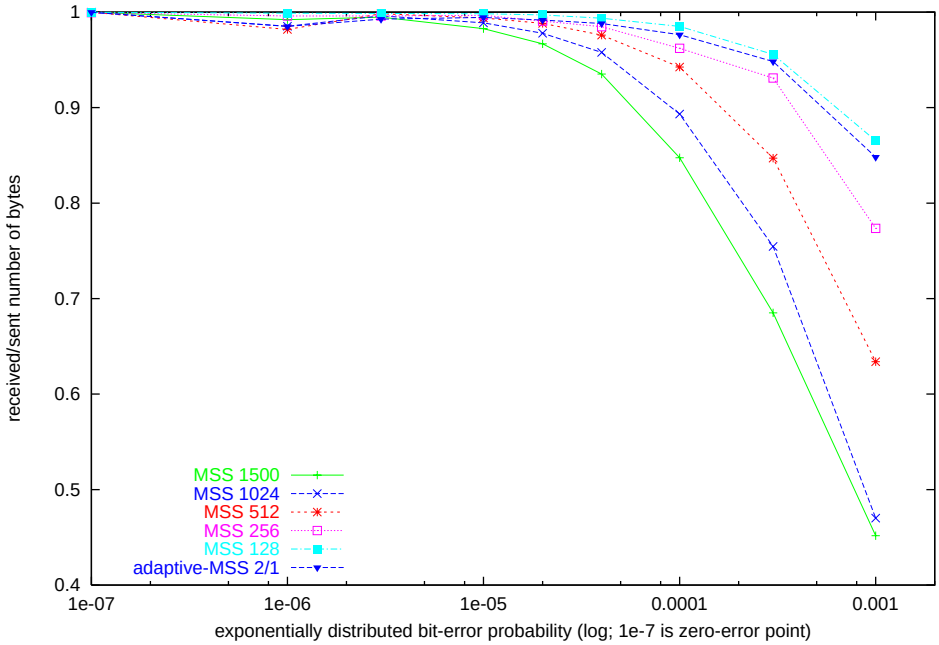


(a) TCP

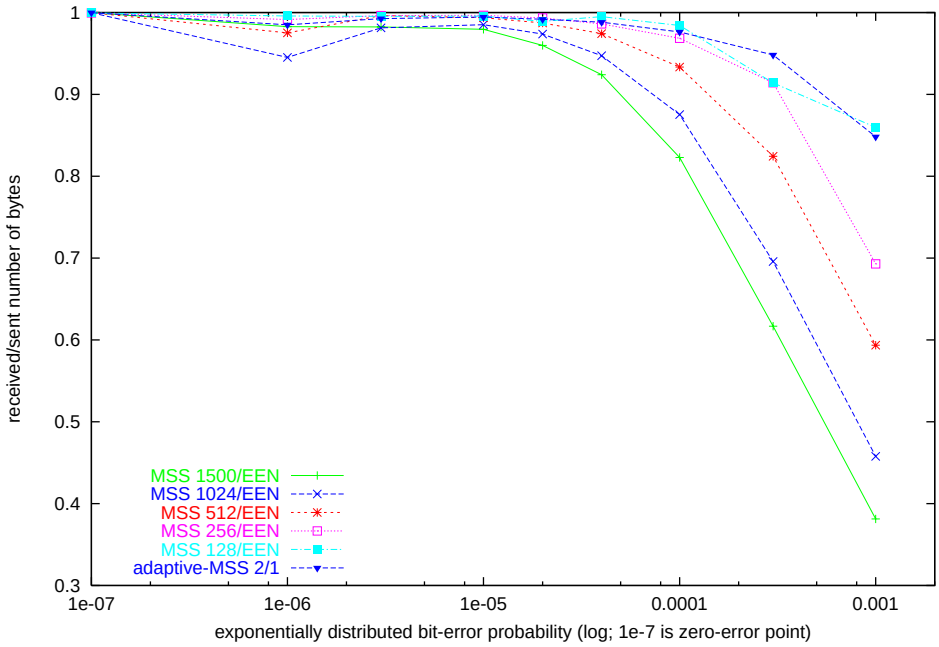


(b) TCP-EEN

Figure 3: Packet loss rate.

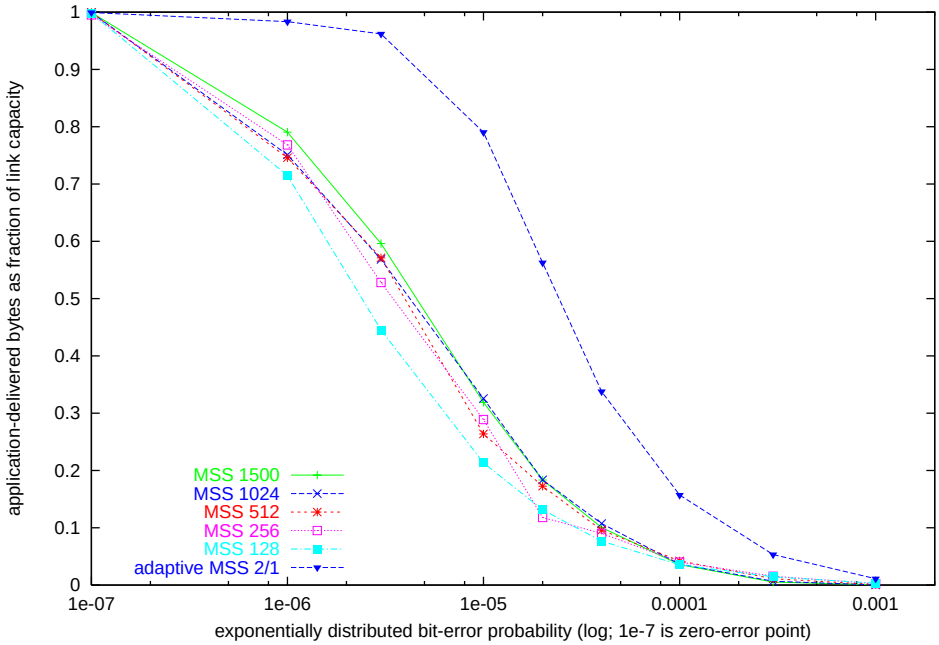


(a) TCP

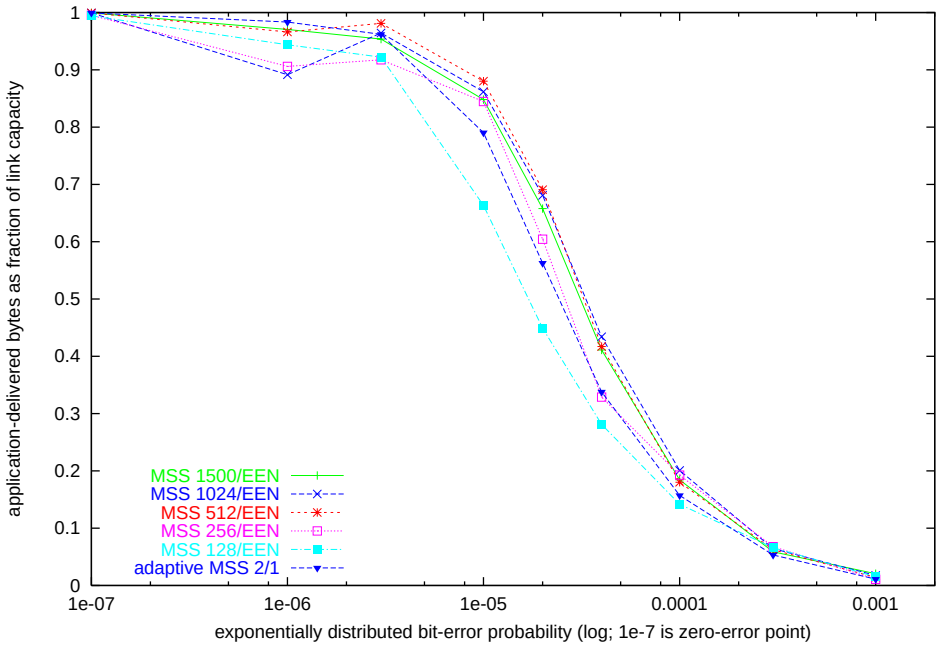


(b) TCP-EEN

Figure 4: Data transmission overhead.



(a) TCP



(b) TCP-EEN

Figure 5: Application goodput.

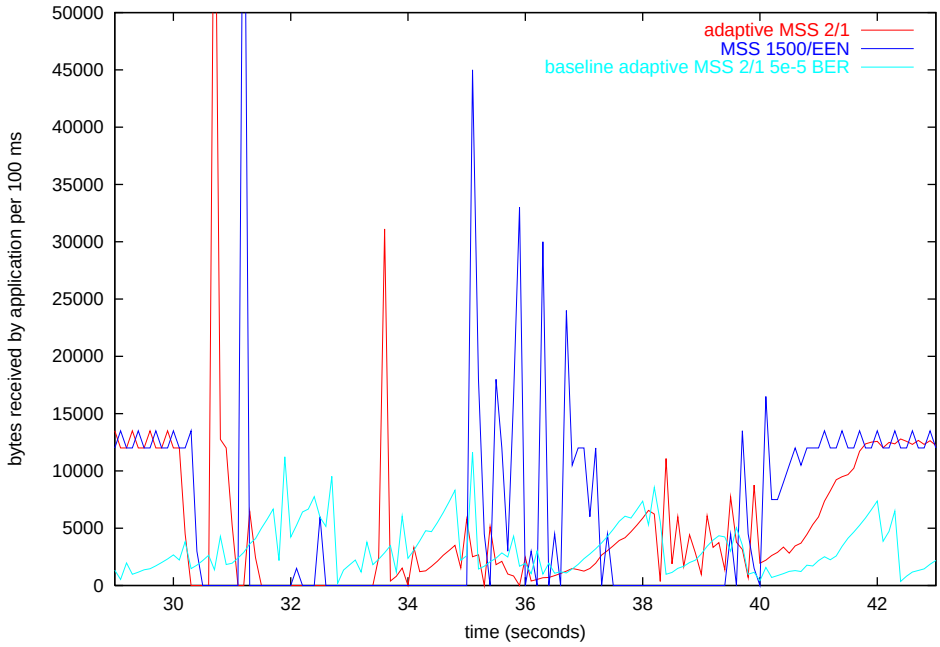


Figure 6: Performance under BER transition.

5.6 Error Transition

It is important to evaluate the stability of our algorithm under changing error environments. It is clear from our results above that fixed packet size methods cannot accomodate a wide range of bit error rates.

Figure 6 shows the number of bytes received by the end application over time, sampled every 100 ms. The bit error rate starts at 10^{-6} in the beginning of the simulation and changes to $5 * 10^{-5}$ after 30 seconds. The BER drops back to 10^{-6} after 10 seconds (40 second point). The link bandwidth is 1 Mb/s, which is 12500 bytes per 100 ms in this figure. There are some spikes in the figure, greater than the link bandwidth. This is caused by the TCP protocol. With TCP, the end application receives data in-sequence. If there is a lost packet, data delivery to the application is delayed until the sender fills the hole.

The intervals with zero received bytes are TCP timeouts if longer than one second. During the timeout, the sender sends no data, after which the sender starts from slow start.

Though omitted from the figure, unmodified TCP has poor performance during the high error duration and recovers slowly after it. It suffers from a couple timeouts during the 10 seconds, each accounting for more than 1 second. It also takes about 2.2 seconds to recover after the high error duration ends. TCP-EEN performs the best during the transition and recovers the earliest. However, the goodput varies so dramatically because there are lots of packet losts. The sender repeatedly retransmits corrupted packets and retransmits corrupted retransmissions, because of the high packet loss rate.

Our adaptive scheme has smooth change in goodput though it recovers from the high error duration somewhat gradually. Though we do not show it, the sender (regardless of what type it is) suffers a timeout as soon as the high error duration begins. This is an unavoidable artifact of large windows during the low error duration.

6 Conclusion

We have shown that performing adaptive packet sizing at the end hosts provides a good framework for increasing TCP performance in error-prone wireless environments. Simulations show that adaptive packet sizing can dramatically improve the packet loss rate and the data transmission overhead. Doing so is advantageous for the mobile hosts that need to conserve power by transmitting the least amount of data necessary. Adaptive packet sizing, however, does not show significant improvements in TCP goodput seen by the receiver. We shall investigate the validity of this result further. We also note that the space of explorable adaptive parameters is quite large and yields quite varied results We hope to produce a usable heuristic in choosing the parameters for MDAI and even perhaps dynamically adapt to the parameters themselves during the course of a TCP transfer.

Deployment of any modification to TCP is quite challenging due to the large installation base of TCP/IP. However, any compelling feature added to TCP can propagate to the end hosts in a reasonable manner due to the limited number of TCP/IP implementations in operating systems. Existing core routers already have the capability to mark packets under congestion. It is not difficult to modify routers to mark corrupted packets.

7 Future Work

The question of high bit error rate can be solved by at least two different ways: one is to retransmit a corrupted packet as soon as possible, another is reducing packet sizes and thus increasing the probability for packets to get through safely without corruption.

We investigate the combined solution of both approaches, and observe some improvements. However, for the first approach, it's far more complicated than what we present in this paper and other papers that mention the idea. This is because of the difficulty in decoupling the corrupted packet processing from the congestion control mechanism. For example, consider a case that has a packet corrupted and the immediately following packet lost by lack of router buffer, all happening in the same TCP congestion window. The receiver emits two different signals for each loss– an EEN signal for the first, and duplicate ACKs for the second. This prompts the sender to fill the first hole (due to EEN), the packet not received correctly due to bit error. The duplicate ACKs, however, also indicate that the first hole exists, because ACKs are cumulative and not selective. The sender can only retransmit the second lost packet after a timeout. This problem may degrade performance because of this unnecessary timeout. We do not have a complete solution for this yet.

For the second approach, we use the MDAI parameters 2/1 for our scheme in most of the results presented, because it outperforms the other parameters in most cases. This is not necessary the best parameter under all network configurations, and we are interested in examining how to choose parameters appropriately.

Another question about correctness is in regard to the TCP congestion window, which is measured in bytes. By shrinking the packet sizes, we can either shrink the window to preserve the number of packets in the window, or we can not change the window to preserve the number of bytes. Though the definition of a congestion window is in terms of bytes, the intent of TCP may dictate preserving the number of packets. This is because the routers in the network core perform buffer accounting in packets, not bytes, which is the original assumption for TCP. Our goodput measurements may suffer from this dillema. We currently choose to preserve the number of packets. Shrinking packets, then, will decrease the number of bytes to send.

References

[BK97] H. Balakrishnan and R. H. Katz. Explicit Loss Notification and Wireless Web Performance. In *Proc. IEEE Globecom Internet Mini-conference*, December 1997.

[BPSK97] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking*, December 1997.

[CWKS97] B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai. IEEE 802.11 Wireless Local Area Networks. *IEEE Communications Magazine*, September 1997.

[ES98] David A. Eckhardt and Peter Steenkiste. Improving Wireless LAN Performance via Adaptive Local Error Control. *Sixth IEEE International Conference on Network Protocols (ICNP'98), Austin*, October 1998.

[FF99] Sally Floyd and Keven Fall. Promoting the Use of End-To-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, August 1999.

[Flo94] Sally Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5), October 1994.

[Gil60] E.N. Gilbert. Capacity of a burst-noise channel. *Bell System Tech. Journal*, 39, September 1960.

[MF00] S. McCanne and S. Floyd. UCB/LBNL/VINT Network Simulator - ns (version 2). *http://www-mash.cs.berkeley.edu/ns/ns.html*, 2000.

[MMFR96] Matthew Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP Selective Acknowledgement Option. *Internet Request for Comments (RFC 2018)*, October 1996.

[Mod99] Eytan Modiano. An adaptive algorithm for optimizing the packet size used in wireless ARQ protocols. *Wireless Networks*, 1999.

[Pos81] J. B. Postel. Transmission Control Protocol. *Internet Request for Comments (RFC 793)*, September 1981.

[Ste94] W. R. Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, 1994.

[Wan99] S.Y. Wang. Decoupling Control from Data for TCP Congestion Control. *Ph.D. Thesis, Harvard University*, September 1999.