

ENERGY EFFICIENT TCP

L. Donckers, P.J.M. Havinga, G.J.M. Smit, L.T. Smit

University of Twente
department of Computer Science
Enschede, the Netherlands
lewie.donckers@philips.com, {havinga, smit, smitl}@cs.utwente.nl

Abstract. This paper describes the design of an energy-efficient transport protocol for mobile wireless communication. First we describe the metrics used to measure the energy efficiency of transport protocols. We identify several problem areas that prevent TCP/IP from reaching high levels of energy efficiency. In the design of Energy Efficient TCP (E²TCP) we provide solutions for these problems, and concentrate on the wireless link only. A simulation model of this proposed energy-efficient transport protocol has been implemented. The simulation results show that E²TCP not only has higher energy efficiency than TCP/IP, but it also manages to outperform TCP/IP on more traditional performance metrics: throughput and latency.

1 Introduction

Energy efficiency and performance of the communication system are two prominent issues in current and future mobile systems. The *Transport Control Protocol* (TCP) is a reliable, end-to-end, transport protocol that is widely used to support various applications. TCP was designed for wired networks and has been highly tuned over the years. Although TCP is very efficient on wired networks, it has been shown to perform poorly on wireless networks in both performance and energy efficiency. In the presence of a high packet error rate and periods of intermittent connectivity of wireless links, TCP may overreact to packet losses, mistaking them for congestion. TCP responds to all losses by invoking congestion control and avoidance algorithms. Until recently, TCP was studied solely (and extensively) in the context of performance, and significant improvements have been achieved. To alleviate the effects of non-congestion-related losses on TCP performance over high-loss networks (like wireless networks), several schemes have been proposed. In [1] and [11] several schemes have been examined and compared. These schemes are classified into three categories: *end-to-end protocols*, where the sender is aware of the wireless link; *link-layer protocols*, that provide local reliability and shields the sender from wireless losses; and *split-connection protocols*, that break the end-to-end connection into two parts at the base station.

Studies on the *energy efficiency* of TCP have been very limited so far. E.g. [15] analyses the energy consumption performance of various versions of TCP. They argue that energy efficiency can be improved by avoiding periods of bad channel conditions. In fact, this is exactly what the window adaptation algorithm of TCP does. However, since their model is based on many simplifications, it is unclear how accurate their results are. Moreover, in their analysis they only considered the energy spent in transmitting data, and did not consider energy consumption while the interface is idle. Tsaoussidis et al. [13] compared the energy performance of several TCP variants. They make fewer simplifications, and their results are based on simulations. Their results indicate that the energy performance of the various TCP variants is fairly similar.

In this paper we present our energy efficient TCP variant (E²TCP), which is based on the split connection principle. We present simulation results and compare other TCP variants with E²TCP, taking into account both energy spent in communication as well as spent in idling.

1.1 E²TCP objectives and assumptions

The objective of this work is to design an energy-efficient transport protocol E²TCP that is suitable for the wireless link between base station and mobile host in a split-connection scheme. Addressing link errors near the site of their occurrence seems intuitively attractive because they understand their particular characteristics and are likely to respond more quickly to changes in environment. We will show that because of the totally different characteristics of the wireless channel and the wired network, better energy-efficiency can be achieved when separating them. To realize this, the transport protocols on the mobile nodes and the base stations need to be adapted. The protocol makes no assumptions about the link- and media access control (MAC) layers, or requires any interaction between them. Therefore, the protocol can be used as a drop-in replacement for TCP/IP on the mobile host and should be usable on all wireless links.

1.2 Energy consumption

The wireless network interface of a mobile computer consumes a significant fraction of the total power [12]. Typically, the transceiver can be in five power modes; in order of increasing energy consumption, these are off, sleep, idle, receive, and transmit. The difference in the amount of energy consumed in these modes is significant. For example, the power consumption of a WaveLAN modem when transmitting is typical 1675 mW, 1425 mW when receiving, and 80 mW when in sleep mode [14]. Also important to notice is that the transition times between the operating modes can be quite high. In WaveLAN power-up takes 200 ms, a transition time from sleep to idle takes 250 μ s [9]. Then, before the payload will be transmitted another 254 μ s is required.

Basically there are two characteristics that influence the energy efficiency and overhead of a protocol. The first characteristic is the *data overhead of a protocol*. When a protocol uses more bytes to transmit the same amount of data, more bytes are wasted, and thus the protocol becomes less energy efficient. The second characteristic that influences the energy efficiency of a protocol is *time overhead*. In general, the longer the protocol needs to transmit the same amount of data, the longer the radio has to be active, and consequently the more energy is wasted.

How much each characteristic influence energy consumption depends on the type of transceiver (transmitter and receiver) and what kind of link and MAC layer are used to transmit and receive the packets. We can identify two extreme types of wireless interfaces (also referred to as radio):

- *Always active*. Such a type of radio is always in the active state during a network session. A *network session* is a period in which there is an established connection between the mobile and the base station, for instance by requesting email from a mail server or establishing a telnet session with a telnet server. During a network session it is possible to use the network but it is often infeasible for the radio to enter the off/sleep state. Because of the small difference between the energy consumption levels of the active state and the transmit and receive states, data overhead does not have a large impact on energy efficiency. Time overhead is much more important because the quicker the data has been transmitted and the network session can be ended, the sooner the radio can put in the off state. WaveLAN is an example of such a type of radio.
- *Ideal*. An ideal radio is always in the sleep- (or even off-) state during a network session, except when it has data to transmit or receive. For such a type of radio, time overhead only has a very small impact on energy efficiency. Data overhead is

much more important. E²MaC is an example of this type of radio [9].

We will calculate energy overhead (see the next paragraph for a definition) as the weighed average of data overhead and time overhead. In the simulations we will assume an *intermediate type* of radio in which the time and data overhead have an equal share (both 50%).

1.3 Metrics

Energy efficiency is a measure to indicate how much energy a protocol uses to transmit data compared to an ideal protocol. For a given data transmission medium there is a minimum amount of energy M that is required to send data from source to destination. The actual spent energy is called S . The difference between those two values is called W : the amount of wasted energy. *Energy efficiency* \mathcal{E} then is: $\mathcal{E} = M / S$. However, energy efficiency is not always a good metric to compare various protocols since it is related to the amount of data transmitted. Therefore we introduce another metric: *energy overhead* O . Energy overhead is the amount of wasted energy compared to the minimum amount of energy, or $O = W / M$. Energy overhead is more suited to show the differences between two protocols.

2 E²TCP

In this section we introduce the basic mechanisms of E²TCP. Because E²TCP is derived of TCP, its architecture and mechanisms are roughly the same. On several points, however, adjustments were made to increase the energy efficiency of the protocol. These points are the acknowledgements, the window management, and the headers. All these changes will be introduced in the following paragraphs. More details on the design of E²TCP can be found in [5].

2.1 Selective acknowledgements

Standard TCP can only generate positive cumulative acknowledgements. This means that when the end station receives an out-of-order packet (due to packet reordering or packet loss) it is unable to send this information to the sender. Selective acknowledgements (SACK) can convey information to the sender about multiple non-contiguous blocks of successfully transmitted segments [8]. E²TCP not only supports SACK but also relies on them to effectively increase its energy efficiency. Because E²TCP will work on a *single-hop* link and performs *local* retransmissions, it knows when a packet is received out of order, that the intermediate packets were lost. Upon noticing out of order packets, the receiver will indicate to the sender (with SACK), that it has not received the intermediate packets. The sender can immediately retransmit the lost packets and does not have to wait for timeouts or duplicate acknowledgements. This will

reduce the time overhead of E^2 TCP without increasing the data overhead.

Although SACK blocks increase data overhead, we will show that the effect of selective acknowledgements on energy efficiency is positive because the sender is able to react in a better way to lost packets. The effect is that it slightly decreases data overhead (because of less retransmits) and reduces time overhead substantially (because of a better utilization of the available bandwidth).

An E^2 TCP receiver can add up to two SACK blocks to an acknowledgement. This enables the receiver to fully acknowledge a received stream with two sets of subsequent missing packets. Standard SACK implementations require ten bytes for each SACK block. We have reduced this to just 2 bytes per block, since the SACK block will always fall within the maximum possible window size (because no more has been transmitted). A 16-bit number can always represent the difference in sequence numbers between the acknowledgement numbers and the beginning and ending of the SACK blocks. When the destination host receives a packet it always send an acknowledgement, and acknowledge as much data as possible. The acknowledgement number field contains the number of the last packet in the contiguous received prefix of the stream.

2.2 Window management

E^2 TCP features a window management scheme that is optimized for energy efficiency on wireless single-hop links. The assumption of TCP that each packet loss is an indication of congestion is valid on wired networks, but has little value on wireless links where packets are lost due to errors. The window management mechanism of E^2 TCP differs on four points from TCP.

- First of all, E^2 TCP features *immediate retransmits*. When the receiver indicates it has received an out-of-order packet, the sender can immediately retransmit the missing packets, because E^2 TCP will be used on a single-hop link and no packet reordering can take place on such a link. Under the same conditions standard TCP would wait on a timeout before it would retransmit the lost packet, causing substantial delays. This change will therefore primarily decrease the time overhead.
- The second change is that E^2 TCP takes into account the *error characteristics of the wireless channel*. If few errors occur, E^2 TCP considers this to be the result of normal random errors on the wireless link. When lots of errors occur, E^2 TCP considers this to be caused by a burst error and *drastically* reduces its transmission speed since the next packets are likely to be lost anyway. This way, E^2 TCP reacts to (burst) errors in a very energy-efficient way, as will be shown in Paragraph 3.3. Both data and time overhead will decrease because of this change. It should be noted that this new

window management scheme relies on selective acknowledgements to detect the number of errors.

- E^2 TCP also features a *minimum window size*, which is the third point on which the window management of TCP and E^2 TCP differ. This minimum window size causes E^2 TCP to quickly recuperate after a burst error, which will decrease time overhead.
- The final change to the window management of TCP is the use of a *retransmission timer*. The timers used in E^2 TCP are similar to the transmission timer in TCP, only one is used for transmissions and one is used for retransmissions. The extra timer increases the responsiveness of the protocol to changes on the channel and thus decrease time overhead.

Operation

TCP always considers packet loss to be the result of *congestion*, and reduces its transmission rate even when the packet loss was caused by errors. This in contrast to E^2 TCP which considers *errors* on the wireless channel to be the cause of lost packets. Because E^2 TCP discriminates between single errors and burst errors it is able to achieve higher energy efficiency. When the burst error encountered is relatively small it is best to continue sending at the original pace since the protocol has no time to react. When the burst error is long however, it would be best to stop sending until the burst error has passed.

There are a few problems that have to be overcome before a scheme like this can be implemented. First, it is unknown a priori when a burst error will start and end. Therefore, the protocol has to detect it by itself, which takes at least as long as the delay on the channel. The second problem is that when the protocol stops sending in case of a long burst error, it has no way of telling when the burst error is over. So, the protocol has to probe (poll) the channel regularly by sending some packets.

Upon reception of an acknowledgement a scoreboard, which keeps track of acknowledged data, is updated to reflect the changes. Each acknowledgement is analyzed to see if it informs the sender of new lost packets. If the amount of newly reported errors is zero, the window size is enlarged up to the *maximum window size*. If the amount of newly reported errors is still below a certain *error limit*, E^2 TCP considers the packet loss to be the result of normal single errors on the channel and will decrease the window size but not below the *minimum window size*. When the amount of errors exceeds the limit E^2 TCP considers this to be the result of a burst error and the window size is set to the *burst error window size*. E^2 TCP will transmit as much as the current window size allows. With each transmission it will set the transmission timer to a value slightly higher than the round trip estimate to compensate for small variations in the actual round trip

time. After a transmission or retransmission timeout E^2 TCP assumes this to be caused by a burst error, and reduces the window size to its minimum size. When E^2 TCP detects packet loss it will *immediately* retransmit those lost packets.

Ideally, each datagram transmission should have its own timeout timer, but this is rather costly when the window size is large. In standard TCP there is only one timer: the transmission timer. E^2 TCP is able to detect errors on the wireless link much sooner than TCP because it applies *two* timers: the transmission timer and a retransmission timer. The retransmission timer will be set in the same way the transmission timer is set. With two timers, E^2 TCP is able to detect burst errors sooner. Therefore, E^2 TCP is more responsive to variations in the quality of the channel, which reduces time overhead. Consider for example the following situation. Both a TCP and an E^2 TCP sender will retransmit a packet at time 1 and transmit a new packet at time 5. The timers will be set to expire in 6 time units. At time 1, a TCP sender will set its timer to 7 and reset it to 11 when it transmits the new packet at time 5. So no sooner than time 11, it is able to detect both packet losses. An E^2 TCP sender, however, will set its retransmission timer to 7 at time 1 and its transmission timer to 11 at time 5. At time 7 it is already able to detect the loss of a packet.

2.3 Header format

TCP has a minimum header size of 40 bytes. When options are activated the size can grow to 80 bytes. E^2 TCP headers are general much smaller and are still robust for errors. Reducing the size of the header not only implies that less data has to be transmitted; it also makes the packet less susceptible to errors. Both effects increase energy efficiency.

The E^2 TCP header may contain some of the standard TCP fields like source- and destination IP and port numbers, sequence and acknowledgement numbers, window, urgent pointer, and checksum. Because the source- and destination IP addresses and ports are large, and will not change during a connection, they will only be sent during connection startup. Until the connection is terminated, a one-byte connection identifier will be used instead. This is comparable to other header compression mechanisms (e.g. [3]). The header also includes some flags to use for connection startup and termination (like the SYN and FIN code bits in TCP headers). Furthermore E^2 TCP has fields to indicate SACK blocks to support selective acknowledgement (SACK). Another optimization is realized by having a special field that indicates whether a certain option is included in the header or not. Optional fields have a bit in the header indicating whether or not they are included. Because almost all fields in the E^2 TCP headers are optional and only need to be transmitted when they are required, E^2 TCP headers are usually quite small, just 8 bytes. Normal

acknowledgements will have a size between 8 and 16 bytes depending on how many SACK blocks are used. This is considerably less than TCP acknowledgements that have a size of 40, 50 or 60 bytes (with none, one and two SACK blocks respectively) up to a maximum of 80 bytes if more options are used.

3 Simulation

3.1 Simulation setup

To measure the performance and energy efficiency of E^2 TCP and compare the protocol with other versions of TCP (Tahoe, Reno and NewReno), an implementation of E^2 TCP was made in Network Simulator 2 [7]. The simulation setup consists of two hosts connected by a wireless LAN. Each host is running TCP and together they create one TCP connection that connects both hosts.

The *default simulation setup* uses the intermediate radio, and has a bandwidth of 1 Mbps. The default delay will be 50 ms, which is an estimation of the delays introduced by a typical IEEE 802.11 physical layer, link layer and MAC layer combined, based on measurements by [4]. Various forms of traffic will be simulated to model different types of applications. The default simulated traffic will be a mass data transfer of 20 MB in total. In addition to this traffic type, we simulated the impact of an interactive traffic model and a constant bit rate model as well. The interactive traffic models applications that feature more or less randomly interspersed small amounts of data. This resembles interactive applications like telnet sessions, instant messaging services, chatting and possibly browsing and sending and receiving emails. The constant bit rate traffic resembles streaming media, like audio and video.

3.2 Error model and setup

All packets on the wireless LAN are transparently routed through the channel that is based on an error model. The error model randomly corrupts the packets with a chance that corresponds to the error rate of the state it is currently in. We use a two-state error model with a *good state* that resembles a high quality channel with some modest random noise and a *bad state* that resembles a burst error with a very high error rate. The state will switch between the states after a certain time. Each state has three parameters: its minimum time, its maximum time and its error rate. The error rate of the state applies to packets (datagrams). For the good state an error rate of 0.05% was chosen which corresponds to measurements done by [6]. For the bad state an error rate of 80% was chosen, causing an average of 4 out of 5 packets to be corrupted. These values were fixed during the simulations and the lengths of the states were varied to model different channel conditions.

We have performed all our simulations using two Scenarios, which represent two quite different error

characteristics. The first Scenario (*Scenario A*) has a fixed bad state length of 0.1 second and the good state length varies from 300 seconds to 1 second. This corresponds to a nearly perfect channel (the simulations finish within 300 seconds of simulated time) to a very bad channel. In this Scenario the proportions between the good state and bad state length are gradually worsened. In the other Scenario (*Scenario B*) the good state lengths vary from 20 to 1 second, with the bad state length always being one tenth of the good state length. This allows the protocols energy efficiency to be examined with varying bad state lengths while the proportions between the good state and bad state length remain the same.

3.3 Evaluation of E²TCP

All simulations in this section are performed with the default setup, unless specified otherwise. We first determine the time and data overhead separately. Then, simulations will be presented with diverse bandwidths, various channel delays, and traffic types. Finally, we look at the throughput and latency performance of E²TCP.

Data and time overhead

As discussed before, energy efficiency is influenced by both time overhead and data overhead. In this simulation setup, we evaluate these issues separately.

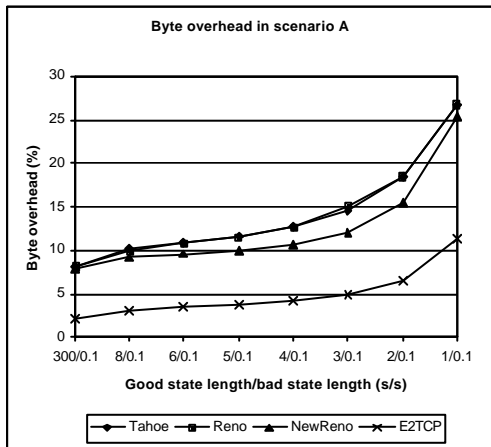


Figure 1: Data overhead of various protocols in Scenario A.

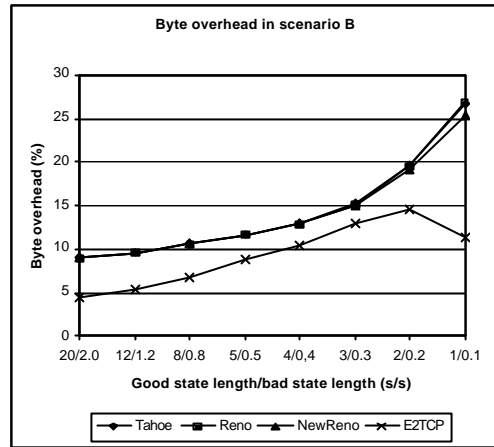


Figure 2: Data overhead of various protocols in Scenario B.

As can be seen in Figure 1 and Figure 2, E²TCP has less *data overhead* than the other TCP versions, in both Scenarios, at all points. This can be attributed to the small headers and its optimized window management in combination with selective acknowledgements. Especially in Scenario A it is clear that when the quality of the channel deteriorates, the data overhead increases. It is interesting to note the decrease in data overhead in Scenario B for E²TCP at the right side of the graph. Unlike standard TCP, E²TCP does not decrease its transmission speed for small burst errors, resulting in a very low data overhead.

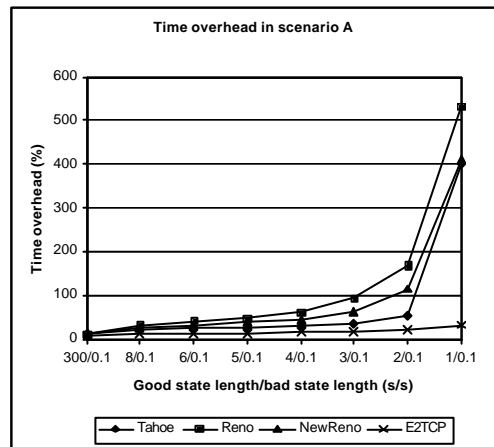


Figure 3: Time overhead of various protocols in Scenario A.

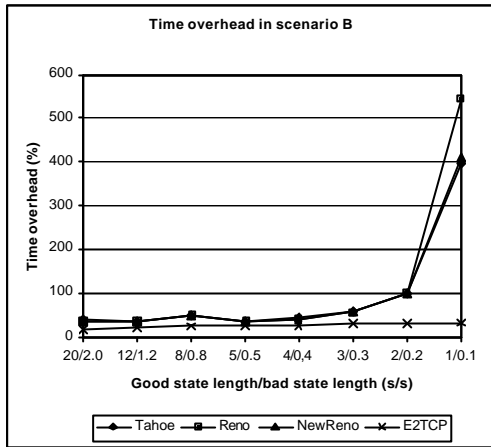


Figure 4: Time overhead of various protocols in Scenario B.

The *time overhead* of E^2 TCP is far less than other protocols in all cases (Figure 3, Figure 4). Especially when the quality of the channel deteriorates (the right side of the graphs), the difference in time overhead between E^2 TCP and the other protocols increases. This means that (considering time overhead) E^2 TCP scales much better than the other protocols when the quality of the channel gets worse.

It should also be clear that the other versions of TCP have much more time overhead than data overhead. Note that all three standard TCP versions behave the same, which is in line with the results presented in [13]. Since Tahoe tends to perform slightly better, in the following simulations we will compare E^2 TCP with Tahoe only.

Bandwidth

In this simulation we evaluate the impact of the channel's bandwidth on the energy overhead and determine whether the protocol scales well. It can be expected that using a faster channel, the overhead will increase.

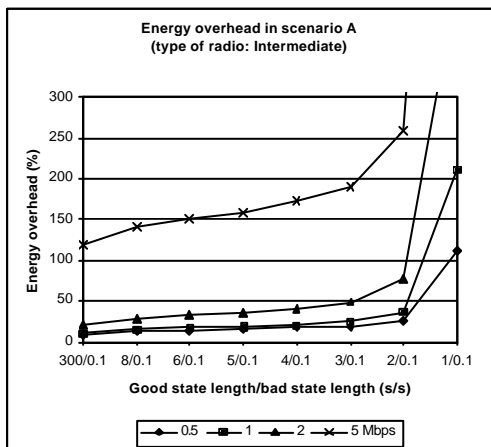


Figure 5: Energy overhead Tahoe various bandwidths Sc. A.

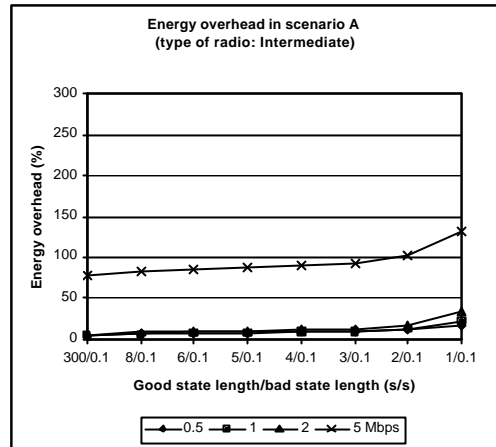


Figure 6: Energy overhead E^2 TCP various bandwidths Sc. A.

E^2 TCP clearly has less energy overhead than Tahoe for each bandwidth/quality of channel combination (see Figure 5 - Figure 8). In most cases Tahoe even has an energy overhead that is at least twice as large as that of E^2 TCP. Independent of bandwidth, E^2 TCP scales better than Tahoe when channel conditions deteriorate.

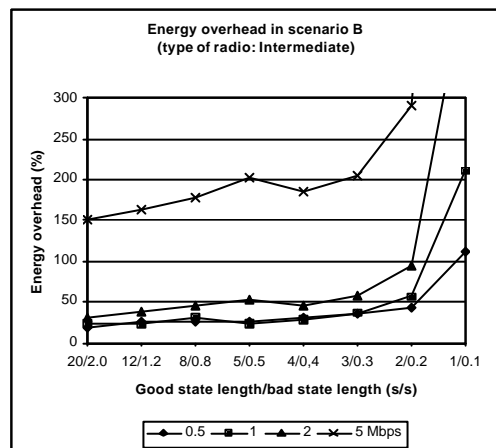


Figure 7: Energy overhead Tahoe various bandwidths Sc. B.

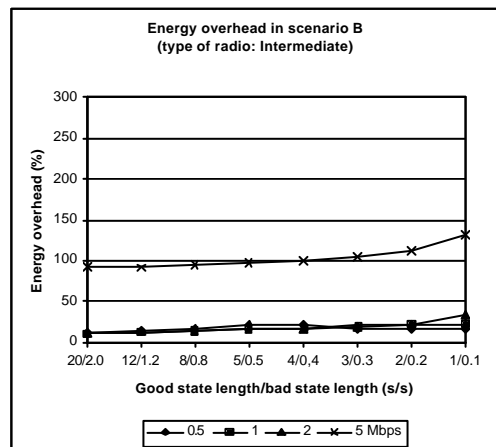


Figure 8: Energy overhead E^2 TCP various bandwidths Sc. B.

Channel delay

The impact of the delay of the channel on the energy overhead of Tahoe and E²TCP was examined in this simulation. In this simulation the delay was varied between 40, 50, 60 and 70 ms.

As expected, the channel delay has a high influence on the efficiency (see Figure 9 and Figure 10). The graphs further show that E²TCP again has a lower energy overhead than Tahoe on all delay/channel quality combinations and thus is more energy efficient. Moreover, E²TCP scales much better when the channel delay increases.

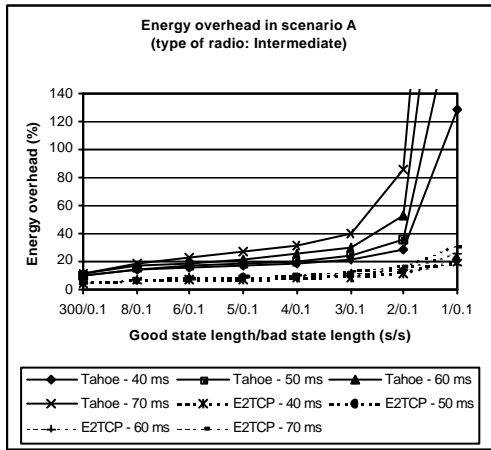


Figure 9: Energy overhead of Tahoe and E²TCP with various delays in Scenario A.

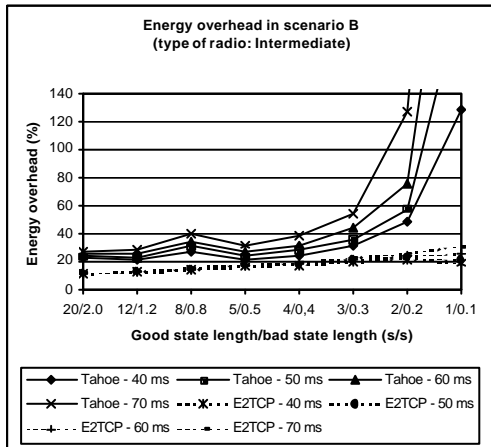


Figure 10: Energy overhead of Tahoe and E²TCP with various delays in Scenario B

Traffic type

Up to now, all simulations was done with a simulation of a (mass) data transfer. In this simulation two other types of traffic will be used for the simulation: an *interactive application model* that models interactive types of traffic like telnet sessions, chatting and instant messages, and a *constant bit rate model*, that models streaming audio and video. With the interactive traffic model the delay between consecutive packets can be set. The application will then randomly create packets in such a way that the average delay

between packets equals the set value. In this simulation interdeparture times of 0.5, 0.2, 0.1 and 0.05 seconds were used, which resemble data rates of 16 to 160 Kbps (2 to 20 KBps).

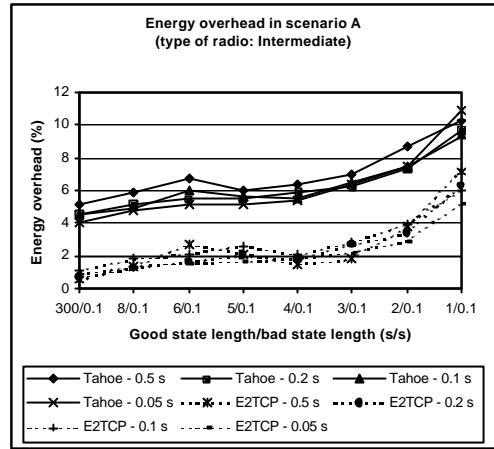


Figure 11: Energy overhead of Tahoe and E²TCP for interactive traffic in Scenario A.

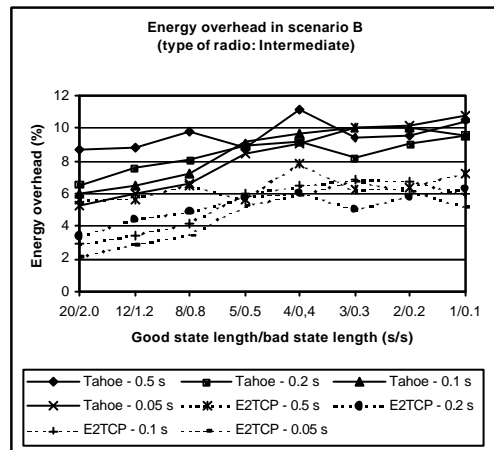


Figure 12: Energy overhead of Tahoe and E²TCP for interactive traffic in Scenario B.

As can be seen in Figure 11, the performance of both protocols in Scenario A barely changes when the interdeparture times are altered. In that Scenario Tahoe consistently has about twice as much energy overhead as E²TCP. In Scenario B (Figure 12), the differences are not as large but E²TCP still manages to score lower energy overhead.

For the *constant bit rate traffic model* we used data rates of 0.25, 0.5 and 1 Mbps.

From the graphs (Figure 13 and Figure 14) can be concluded that in Scenario A, E²TCP is much more energy efficient than Tahoe. Furthermore E²TCP scales better when data rates increase. In Scenario B, Tahoe is able to equal E²TCP's energy overhead when bad states are long and data rates low. E²TCP however, is more energy efficient when data rates increase and/or bad state lengths shorten. Note that even though the absolute numbers differ when E²TCP is used with

different data rates, the tendencies do not. This means that for different data rates E²TCP behaves the same.

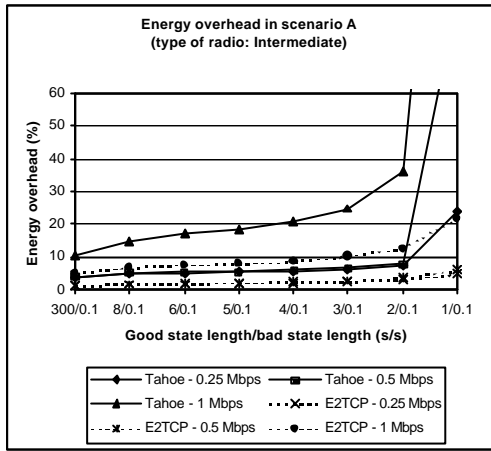


Figure 13: Energy overhead of Tahoe and E²TCP CBR traffic in Scenario A.

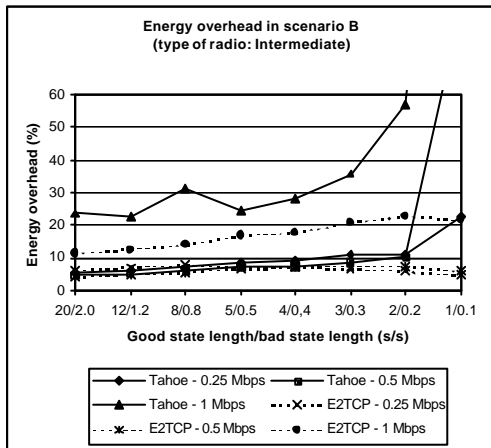


Figure 14: Energy overhead of Tahoe and E²TCP CBR traffic in Scenario B.

Throughput and latency

So far, only the energy efficiency of E²TCP has been examined, which is of course a very important metric. However, it is also important to take a look at some traditional performance metrics, like throughput and latency. In the next graphs (Figure 15 and Figure 16), the *throughput* of E²TCP and other versions of TCP will be shown. Because the default simulation setup was used, the throughput can be no higher than 1 Mbps.

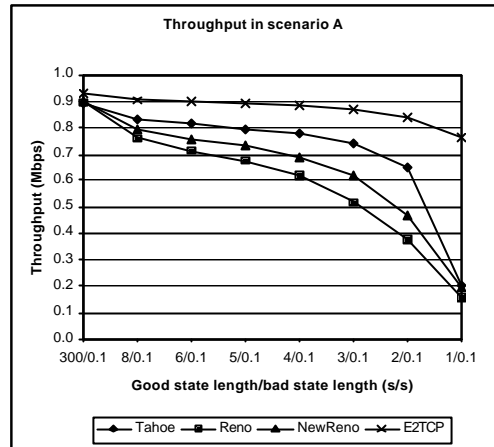


Figure 15: Throughput of various protocols in Scenario A.

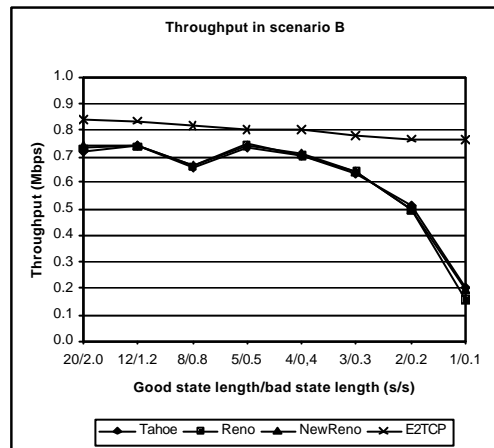


Figure 16: Throughput of various protocols in Scenario B

As can be seen in both graphs, the throughput of E²TCP is clearly higher than that of the other versions of TCP. When channel conditions deteriorate, the difference in throughput becomes exceptionally large. By optimizing E²TCP for energy efficiency by lowering its time overhead, the throughput was increased as well.

The *latency* of a protocol is another traditional performance metric. Unlike throughput, average latency has no direct relation to either data- or time overhead.

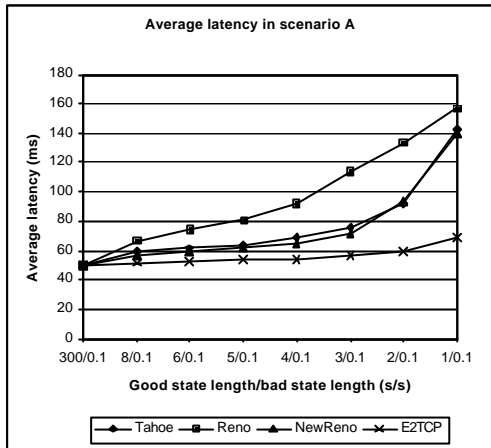


Figure 17: Average latency of various protocols in Scenario A.

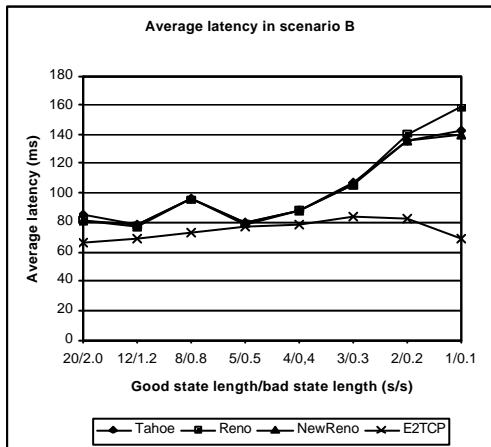


Figure 18: Average latency of various protocols in Scenario B.

Clearly, the graphs shown in Figure 17 and Figure 18 show that the average latency of E^2 TCP is lower than that of the other versions of TCP. As with throughput, the difference increases when the channel conditions deteriorate. Because the default simulation setup was used the average latency could not drop below 50 ms. Taking this into account, the performance of E^2 TCP becomes even more impressive. When the channel conditions are worst, E^2 TCP adds about 20 ms to the minimum latency while the other protocols add about 100 ms.

This could raise the question whether or not optimizing for energy efficiency is the same as optimizing for throughput and/or latency. This is not so. An example that shows that a protocol with a high throughput does not automatically have a high energy efficiency, is a TCP sender that transmits at the highest possible speed. Such a sender would have a very high throughput. However, it would also waste a substantial amount of energy because it would also transmit at the highest possible speed during burst errors. Therefore,

optimizations for energy efficiency are distinct from optimizations for throughput and/or latency.

E^2 TCP dissected

In these simulations we have evaluated the various methods used in E^2 TCP to increase energy efficiency. The methods used are optimized window management, selective acknowledgements, and small headers. When selective acknowledgements were added, E^2 TCP already became quite energy efficient and had less energy overhead than Tahoe in both Scenarios. Using E^2 TCP's custom headers further increased its energy efficiency. To give an indication of the energy overhead of all versions, the *average* energy overhead over all simulated error conditions will be listed in the following table for each version and both Scenarios.

Protocol version	Scenario A (%)	Scenario B (%)
Tahoe	44.5	54.5
E^2 TCP (without header compression)	14.7	22.4
E^2 TCP (with header compression)	10.1	17.4

Table 1: Average energy overhead of various protocol versions.

4 Conclusions and recommendations

E^2 TCP is optimized for energy efficiency on several points. The first point is the acknowledgement scheme of TCP, which is unable to provide the sending host with enough information about the state of the destination host. E^2 TCP uses an *efficient* selective acknowledgement mechanism to overcome this problem. These selective acknowledgements are also required for the second optimization: the window management. This optimization is the result of efforts to make TCP aware of burst errors. Because burst errors are a major cause of packet loss on wireless links and TCP considers all packet loss to be the result of congestion, TCP was unable to react to burst errors in an energy efficient way. These two optimizations cause the greatest decrease in energy overhead: about 75% of the total decrease in energy overhead. The final optimization is the use of custom headers, which rely on techniques from header compression standards to minimize wasted energy. This optimization is the cause of the last 12% of the total decrease in energy overhead.

E^2 TCP has been compared to standard versions of TCP, like Tahoe, Reno and NewReno under various conditions. The bandwidth, delay, type of traffic and channel conditions were widely varied to get a complete overview of the energy efficiency characteristics of E^2 TCP. From the results can be concluded that E^2 TCP has less energy overhead than TCP for each bandwidth/quality of channel

combination. In most cases TCP even has an energy overhead that is at least twice as large as that of E²TCP. Also, E²TCP scales better than TCP when channel conditions deteriorate. Similarly, when the channel delay increases E²TCP scales much better than TCP. Also on traditional metrics like throughput and latency E²TCP outperformed the others easily.

As for *future research*, four areas are recommended to be examined. First of all, energy consumption is device-, operation-, protocol-, and application-specific. Accurate quantitative numbers of the energy consumption of the system, at various stages of communication, and for various configurations and applications would be useful to verify our assumptions. Similarly, we would like to have accurate error characteristics of the wireless channel for various state-of-the-art radios. We have tried to alleviate this by using two quite different Scenarios, and performed all our simulations using both Scenarios. Furthermore, simulations for the base station should be designed and implemented. This would allow for simulations of the entire setup, instead of just the wireless part. Information on the performance of a complete connection (from mobile to internet host and vice versa) would be valuable.

The final recommendation for future research is to compare E²TCP to other protocols that are designed specifically for wireless links. Such protocols generally focus on optimizing performance of the connection with respect to throughput and/or delay. Because of the overlap of optimizing for traditional performance and for energy efficiency, it is expected that these protocols are more energy efficient than standard TCP.

References

- [1] Balakrishnan H., et al.: "A comparison of mechanisms for improving TCP performance over wireless links", *Proceedings ACM SIGCOMM'96*, Stanford, CA, USA, August 1996.
- [2] Brunstrom A., Asplund K., Garcia J., Enhancing TCP performance by allowing controlled loss, *Proceedings of SSGRR 2000 computer & e-business conference*, L'Aquila, Italy, August 2000.
- [3] Casner S., Jacobson V., Compressing IP/UDP/RTP headers for low-speed serial links, *RFC 2508*, February 1999
- [4] Chen K., Medium access control of wireless LANs for mobile computing, *IEEE Network magazine*, V. 8 N. 5, September 1994.
- [5] Donckers L.: "Energy Efficient TCP", MSc. thesis University of Twente, department of Computer Science, http://www.cs.utwente.nl/~havinga/papers/e2tcp_thesis.pdf, 2001.
- [6] Eckhardt D., Steenkiste P., Measurement and analysis of the error characteristics of an in-building wireless network, *Proceedings of the ACM SIGCOMM '96 conference*, October 1996.
- [7] Fall K., Varadhan K., The NS manual, *The VINT project*, <http://www.isi.edu/nsnam/ns/>, October 2000
- [8] Floyd S., Mahdavi J., Mathis M., Podolsky M., An extension to the selective acknowledgement (SACK) option for TCP, *RFC 2883*, July 2000.
- [9] Havinga P.J.M., Smit G.J.M., Energy-efficient TDMA medium access control protocol scheduling, *Proceedings of the Asian International Mobile Computing Conference (AMOC 2000)*, November 2000.
- [10] Mathis M., Mahdavi J., Floyd S., Romanov S., TCP selective acknowledgement options, *RFC 2018*, October 1996.
- [11] Pentikousis K.: "TCP in wired-cum-wireless environments", *IEEE Communications Surveys*, Fourth Quarter 2000, pp 2-14.
- [12] Stemm, M, et al.: "Reducing power consumption of network interfaces in hand-held devices", *Proceedings mobile multimedia computing MoMuc-3*, Princeton, Sept 1996.
- [13] Tsaoussidis V., Badr H., "Energy / Throughput Tradeoffs of TCP Error Control Strategy", *IEEE Symposium on Computers and Communications, IEEE ISCC 2000*, France, 2000.
- [14] WaveMODEM 2.4 GHz Data Manual, Release 2, AT&T 1995.
- [15] Zorzi M., Rao R.R.: "Is TCP energy efficient? ", *Proceedings IEEE MoMuC*, November 1999.