ENERGY EFFICIENT TCP

Master's thesis: L. Donckers
CAES/002/01

May, 2001

University of Twente
Department of Computer Science
Division: Computer Architecture and Embedded Systems

Supervisors:
dr. ir. G.J.M. Smit
dr. ing. P.J.M. Havinga
ir. L.T. Smit

SAMENVATTING

Een nieuw type handcomputer is in ontwikkeling in het Moby Dick project. Door de gewenste functionaliteit en prestaties, zal het energie verbruik de limiterende factor zijn voor deze handcomputer. Draadloze communicatie zal ook een belangrijke factor zijn in het project, wat een energie-efficiënt transport protocol, compatible met TCP/IP, wenselijk maakt. Dit verslag beschrijft het ontwerp van zo'n energie-efficiënt transport protocol voor mobiele draadloze communicatie.

Er is echter nog niet veel onderzoek gedaan naar de energie efficiëntie van transport protocollen. Daarom zijn er eerst maten ontwikkeld om de energie efficiëntie van transport protocollen te kunnen meten. Deze maten zijn gebruikt om de prestaties van TCP/IP op draadloze verbindingen nauwkeurig te bestuderen. Vier probleemgebieden zijn gedefinieerd, die TCP/IP ervan weerhielden een hoog niveau van energie efficiëntie te behalen. Voor deze probleemgebieden zijn mogelijke oplossingen aangedragen waarna de haalbaarheid er van is onderzocht.

De resultaten van dit onderzoek zijn gebruikt om E^2TCP te ontwerpen. Een simulatie model van dit energie-efficiënte transport protocol is geïmplementeerd en onderworpen aan een grondige evaluatie. Uit de resultaten kan geconcludeerd worden dat E^2TCP niet alleen een hogere energie efficiëntie heeft dan TCP/IP, maar dat het ook in staat is beter te presteren op meer traditionele punten: throughput en latency.

ABSTRACT

A new generation handheld computer is under development in the Moby Dick project. Because of the desired functionality and performance, the energy consumption will be the limiting factor for this handheld. Wireless communication will also be an important factor in the project, which makes an energy-efficient transport protocol, compatible with TCP/IP, desirable. This thesis describes the design of such an energy-efficient transport protocol for mobile wireless communication.

However, not much research has yet been done on the energy efficiency of transport protocols. First metrics were developed to measure the energy efficiency of transport protocols. These metrics were used to study the performance of TCP/IP on wireless links carefully. Four problem areas were defined that prevented TCP/IP from reaching high levels of energy efficiency. For these problem areas, solutions were proposed and their feasibility was examined.

The results of this study were used to design E^2TCP. A simulation model of this proposed energy-efficient transport protocol has been implemented and was subject to a thorough evaluation. The results show that E^2TCP not only has a higher energy efficiency than TCP/IP, but it also manages to outperform TCP/IP on more traditional performance metrics: throughput and latency.

TABLE OF CONTENTS

PREFACE

I would like to take the opportunity to thank my supervisors: Gerard Smit, Paul Havinga and Lodewijk Smit for guiding me while I was working on this assignment. I would also like to thank my mother, brother and sister for being there for me. For the same reason I thank Ella, Jac and Jessie.
Last but *certainly* not least, I would like to thank my girlfriend: Lonneke. Without her support this would have been impossible.

Enschede, May 2001

Lewie Donckers

1 INTRODUCTION

This thesis is part of the Moby Dick project at the Computer Science department of the University of Twente. The Moby Dick project is a joint European project to develop and define the architecture of a new generation of mobile handheld computers. Due to the increasing demand for performance and functionality, the energy consumption will be the limiting factor for the capabilities of such a new generation handheld. Therefore reducing energy consumption plays a crucial role in the architecture. An important aspect of the Moby Dick project is wireless communication. Because of the importance of energy efficiency to the Moby Dick project, the wireless communication should also be optimized to minimize energy consumption.

Unfortunately, not a lot of research has been done on energy-efficient transport protocols. Even though it would be quite rewarding to do so, because in mobile systems, the radio (which is used for wireless communication) is one of the parts that consume the most energy [STE97]. Computer chips (like CPUs and memories) are becoming increasingly energy efficient because of advances in IC design. Radios however simply require a certain amount of energy to transmit and receive information. Furthermore, multimedia applications are using network services more extensively and continuously than before. The impact of minimizing the energy spent on wireless communication, will therefore only increase [HAV00b].

For these reasons, it would be beneficial to the Moby Dick project if an energy-efficient transport protocol would be designed. As a basis for such a protocol, TCP/IP (Transport Control Protocol/Internet Protocol) is a likely candidate. This would enable the handheld to communicate, via the Internet, with vast numbers of systems.

This led to the following problem statement:

> *The energy efficiency of transport protocols on wireless links should be researched. Based on that research, E²TCP –an energy efficient version of TCP/IP– should be designed and implemented, to test its energy efficiency and performance.*

The remainder of this chapter further delineates the problem area of E²TCP. Chapter 2 explains what energy efficiency is and how it can be calculated. Chapter 3 then describes TCP/IP, explains what keeps it from reaching high levels of energy efficiency and what can be done to remedy this. The solutions presented in Chapter 3 were used to design E²TCP, which is described in Chapter 4. In Chapter 5 an implementation of E²TCP will be introduced before a thorough energy efficiency and performance evaluation will be given. Finally Chapter 6 gives the conclusions of this thesis.

1.1 Problem area

E²TCP will be used on a wireless link between a mobile host and a base station. The mobile host should be able to connect to the Internet via the base station in such a way that it is transparent to the mobile host and the Internet host it is connected with. This means neither the mobile host nor the Internet host should be able to tell whether E²TCP is used between the mobile host and the base station or regular TCP/IP. (Because the behavior of TCP is highly dependant on IP, E²TCP will have the functionality of both TCP and IP. This maximizes the potential gain in energy efficiency.)

The design of this protocol is limited to the transport layer. The lower layers (the link layer and medium access layer (MAC) layer) should not be changed in any way. This is because the

protocol will be used as a drop-in replacement for TCP/IP on the mobile host and should be usable on all wireless links, just like TCP/IP is usable on all wired networks. The protocol should therefore make no assumptions about the link- and MAC layers.
Now it is clear what should not be changed, it is time to explain what is allowed to be changed. There are three parts of the systems that can be changed. They are additive, which means that the second proposed change also includes the first and that the third proposed change also includes the second and first.
The first proposed change is to replace TCP/IP at the base station with E^{2}TCP. This change requires no user intervention but it is expected that by only applying this change, the increase in energy efficiency is rather small.
The second proposed change is to also replace the transport protocol on the mobile host. This requires some user intervention because the user must install the protocol on its mobile host. Because of the user-friendliness of modern operating systems this should not be a big problem. When this proposal is executed a truly new transport protocol can be designed because it does not have to communicate directly with TCP/IP. A proxy application at the base station can then handle the translation of the energy efficient protocol on the wireless link to TCP/IP on the wired part of the path. The design of the proxy is not part of the problem statement: E^{2}TCP will be the sole point of focus. It is expected that this proposal will yield an increase in energy efficiency compared to only the first change. Such a setup is shown in Figure 1.1.

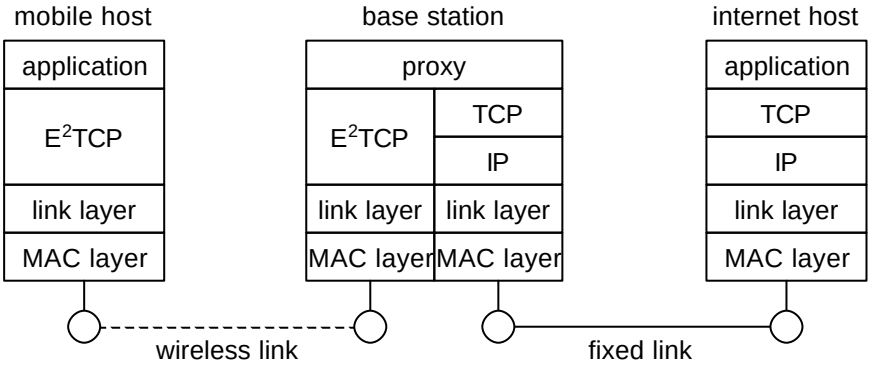

Figure 1.1: The intended setup of E^{2}TCP.

The third proposal is to change (some of) the applications on the mobile host as well. These applications will then be able to optimize the energy efficient connections for their intended use with Quality of Service-like (QoS-like) parameters. It is expected that this will increase the energy efficiency even further in certain cases, which would not have been possible with only the first two proposed changes.
The conclusion of this chapter is that the protocol should meet the following requirements:

- It should be a transport protocol and should be compatible with TCP/IP (through translation at the base station).
- It should be energy efficient.
- It should make no assumptions about the lower layers and leave them unchanged.
- Higher layers (applications) should be able to use E^{2}TCP just like TCP/IP. However, if they are E^{2}TCP-aware, they should be able to optimize E^{2}TCP connections for their intended use with QoS-like parameters.

2 MEASURING ENERGY EFFICIENCY

When researching the energy efficiency of protocols, it is of course important to know what exactly energy efficiency (of a protocol) is and how to calculate it. In this chapter both topics will be discussed.

2.1 Energy efficiency and energy overhead

Energy efficiency is a measure to indicate how much energy a protocol uses to transmit data (in a certain case) compared to an ideal protocol. It will not be measured in absolute values because different cases (with different amounts of payload) should be comparable. An energy efficiency near 0% means little of the spent energy was used efficiently, while an energy efficiency of 100% means that no energy was wasted, which can only be achieved by an ideal protocol.
It is important to know that for a given data transmission medium there is a minimum amount of energy that is required to send data from source to destination. No protocol can use less energy and still successfully complete the transmission. Let's call this minimum M. This is probably different from the actual spent energy, called S. The difference between those two values is called W; the amount of wasted energy. These values are shown in Figure 2.1.

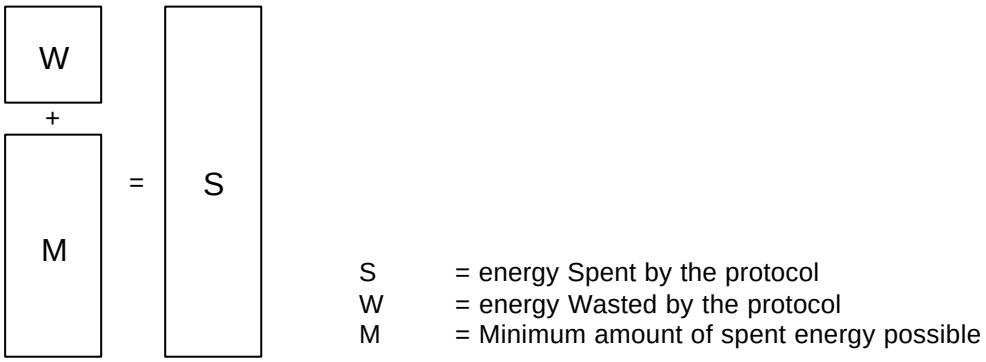

Figure 2.1: The relation between the spent, wasted and minimum amount of energy.

Energy efficiency then is:

$$EE = M / S$$

Equation 2.1: Energy efficiency.

or in words: the part of the spent energy that was used useful. If the protocol is ideal and it only uses the minimum amount of energy (S = M) the energy efficiency is 100%. Since M is fixed and S can only increase, the energy efficiency can only become lower.
Even though this is exactly what is needed to know about the protocols in this assignment, energy efficiency is not a good way to compare various protocols. This is because the differences in energy efficiency will be quite small even though the amount of wasted energy can differ quite much. Consider the following example.

> Example 2.1: The minimum amount of energy for a given data transfer is 100 (the measure used does not matter). Protocol A spends an amount of 125 to complete the transmission and protocol B 150. Clearly protocol B wastes twice as much

energy as protocol A (W is 50 and 25 respectively). But the energy efficiency of protocol A is 100 / 125 = 80% and that of B is 100 / 150 = 67%. When one only looks at the energy efficiency it is easy to see that protocol A is better than protocol B. When one tries to see *how much* protocol A is better, the energy efficiency numbers are not that convenient.

There is another measure that is closely related to energy efficiency: energy overhead. Energy overhead is the amount of wasted energy compared to the minimum amount of energy, or:

$$EO = W / M$$

Equation 2.2: Energy overhead.

This can be seen as the amount of energy that is spent more than the minimum the protocol requires. Because of its close relation with energy efficiency, energy overhead can be calculated when only energy efficiency is known, and vice versa. Unlike energy efficiency however, energy overhead is more suited to show the differences between two protocols. This is shown in the next example.

Example 2.2: Consider the previous example but now the energy overhead will be calculated instead of energy efficiency. The energy overhead of protocol A is 25 / 100 = 25% and the energy overhead of protocol B is 50 / 100 = 50%. This shows precisely that protocol B wastes twice as much energy as protocol A.

Energy overhead will be used to compare protocols from now on, while energy efficiency numbers will sometimes be stated to be complete.

2.2 Data overhead and time overhead

Now a definition of energy overhead has been given, it is time to show how it is calculated. Before this can be done it is important to understand what precisely influences the energy efficiency and overhead of a protocol. Basically there are two characteristics that influence them. 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. Therefore the protocol becomes less energy efficient. The second characteristic that influences the energy efficiency of a protocol is time overhead. In certain cases, the longer the protocol needs to transmit the same amount of data, the longer the radio has to be active. When the radio is active, it requires (extra) energy to operate. Thus, the more time a protocol requires to send the same amount of data, its energy efficiency decreases. These two characteristics are sometimes related.
The question remains how much these characteristics each influence energy efficiency. The answer really 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. For convenience the combination of transceiver, link layer and MAC layer will be called a radio from now on. To distinguish between different types of radios, a general power model of radios will be presented first.

2.3 Power model of radios

A radio has various states in which it operates. In each state the radio requires a certain amount of power to operate. In the following table as an example, the various states of a

WaveLAN modem will be listed, together with a description of the amount of power consumption [HAV00].

State	Power consumption	WaveLAN power consumption (mW)
Off	none	0
Sleep	low	35
Active	high	1325
Transmit	slightly higher than active	1380
Receive	slightly higher than active	1345

Table 2.1: Powerstates of radios.

Of course one could think that the ideal radio would be in the off state continuously, except when it has to receive or transmit data. However, real radios behave differently. When a radio switches between two states it takes a certain amount of time and possibly some amount of energy to complete the switch. Switching to and from the off mode takes so much time it is infeasible to use it to save energy between consecutive transmits and receives. The sleep state can be used for such a purpose. To effectively use the sleep state, however, takes extra coordination and increases the complexity of the lower level protocols. Furthermore, a lot of radios are not optimized for power consumption but for performance. So there are still a lot of radios that do not use the sleep state to save power to its full effect.
It is also important to understand the concept of a network session. A network session is a period in time in which there is a established connection between the mobile and the base station. During a network session it is possible to use the network. For instance by requesting email from a mail server or establishing a telnet session with a telnet server. During a network session it is often infeasible for the radio to enter the off state. This is because switching to and from the off state requires much time. So before and after a network session the radio can be put in the off state to save energy. However, doing this during a session is not a thing a lot of radios are able to do.
Now a general power model of radios has been given, some types of radios will be discussed. There are two extreme types of radios. Not all radios will fit in either categories. All radios however can be placed on a gliding scale between the those two types. These types are:

- Always active. Such a type of radio is always in the active state during a network session. 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 sooner 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 would always be 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 would only have a very small impact on energy efficiency. Data overhead, on the other hand, is much more important. E^2MaC is an example of this type of radio [HAV98].

So at one side of the gliding scale, time overhead is very important and data overhead is not, while at the other side of the scale, data overhead is very important and time overhead is not. Because our energy efficient protocol could be used on either extreme of the scale, it would be best to minimize both types of overhead.

2.4 Calculating data overhead and time overhead

The problem of measuring energy efficiency has now boiled down to two simpler problems. How to measure data overhead and how to measure time overhead. Data overhead will be calculated as follows:

$$DO = S / D - 1$$

Equation 2.3: Data overhead.

where:

DO is Data Overhead.

D is the amount of Data that should be transmitted by the protocol (measured in bytes). This is the payload of the protocol and is often referred to as user data.

S is the amount of data the protocol actually Sent, to transmit the payload D to the receiver (measured in bytes). This includes retransmitted packets, packet headers and acknowledgements.

Consider the following examples:

Example 2.3: when TCP/IP is used to send 1000 bytes in one packet, that would generate one 1040 byte packet (40 bytes header and 1000 bytes payload) and one 40 byte acknowledgement. That would result in:

D = 1000
S = 1040 + 40 = 1080

DO = 1080 / 1000 – 1 = 8%

Example 2.4: when TCP/IP is used to send 1000 bytes in one packet (just like in Example 2.3) but this packet is lost upon first transmission, it would have to be retransmitted. That would result in:

D = 1000
S = 1040 + 1040 + 40 = 2120

DO = 2120 / 1000 – 1 = 212%

Time overhead will be calculated as follows:

$$TO = T / (D / B) - 1$$

Equation 2.4: Time overhead.

where:

TO is Time Overhead.

T is the Time the protocol required to transmit the payload D to the receiver (measured in seconds). Time is measured until the destination has received all data and the sender is aware that this has happened.

B is the Bandwidth available on the link (measured in bytes per second).

Example 2.5: when a protocol requires 1 seconds to transmit 1000 bytes over a link with a bandwidth of 1500 bytes per second, that would result in:

D = 1000
T = 1
B = 1500

TO = 1 / (1000 / 1500) – 1 = 50%

2.5 Calculating energy overhead

Now it is clear how data- and time overhead are calculated, it is time to show how to calculate energy overhead. It has been shown, in this chapter, that energy overhead (and efficiency) depends on data- and time overhead. It has also been shown that how much each characteristic influences energy overhead depends on the type of radio used. Because of this energy overhead will be calculated as the weighed average of data overhead and time overhead. Three ratios will be used, which are all assumed to correspond closely to a certain type of radio. It should also be noted that the 'always active' and 'ideal' types of radio were assumed to be almost always active and almost ideal. So they are not as far on the extreme ends of the scale as mentioned in Paragraph 2.3. They are listed in the following table.

Type of Radio	Data Ratio	Time Ratio
Always active	0.1	0.9
Intermediate	0.5	0.5
Ideal	0.9	0.1

Table 2.2: Data- and time ratios for different types of radios.

Energy overhead can then be calculated like this:

$$EO = DR_R * DO + TR_R * TO$$

Equation 2.5: Energy overhead.

where:
EO is Energy Overhead.
DR_R is Data Ratio with radio R.
TR_R is Time Ratio with radio R.

2.6 Calculating energy efficiency

Like energy overhead is the weighed average of data- and time overhead, energy efficiency is the weighed average of data- and time efficiency. From the definition of energy efficiency it is easy to deduce the definitions of data- and time efficiency. Data efficiency is the part of the amount of data actually sent to complete the transmission, that was used for the payload. And time efficiency is the part of the time it took the protocol to complete the transmission, that the minimum time is. So they will be calculated as follows:

$$DE = D / S$$

Equation 2.6: Data efficiency.

$$TE = (D / B) / T$$

Equation 2.7: Time efficiency.

where:
DE is Data Efficiency.
TE is Time Efficiency.

Please note that both data- and time efficiency (just like energy efficiency) are percentages and are always larger than 0% and less than or equal to 100%. As can be seen data efficiency and data overhead are closely related. One can be used to calculate the other:

$$DE = 1 / (1 + DO)$$

Equation 2.8: Data efficiency as a function of data overhead.

because:
$DE = 1 / (1 + DO)$
$DE = 1 / (1 + (S / D – 1))$
$DE = 1 / (S / D)$
$DE = D / S$

and:

$$DO = (1 / DE) – 1$$

Equation 2.9: Data overhead as a function of data efficiency.

because:
$DO = (1 / DE) – 1$
$DO = (1 / (D / S)) – 1$
$DO = S / D – 1$

Time efficiency and time overhead are similarly related:

$$TE = 1 / (1 + TO)$$

Equation 2.10: Time efficiency as a function of time overhead.

because:
$TE = 1 / (1 + TO)$
$TE = 1 / (1 + T / (D / B) – 1)$
$TE = 1 / (T / (D / B))$
$TE = (D / B) / T$

and:

$$TO = 1 / TE - 1$$

Equation 2.11: Time overhead as a function of time efficiency.

because:
$TO = 1 / TE - 1$
$TO = 1 / ((D / B) / T) - 1$
$TO = T / (D / B) - 1$

With the same ratios listed in Table 2.2, it is now possible to calculate the energy efficiency of a transmission:

$$EE = DR_R * DE + TR_R * TE$$

Equation 2.12: Energy efficiency.

where:
EE is Energy Efficiency.

2.7 Summary

In this chapter, two measures were introduced that say something about the amount of spent energy of a protocol: energy efficiency and energy overhead. It was also shown that, even though the goal of this thesis was the design of an energy efficient protocol, energy efficiency is not the best measure to compare the performance of different protocols. Energy overhead is more suited for this.
To calculate energy overhead (and efficiency) it was stated that two characteristics of a protocol should be know: the data overhead and the time overhead. How much each characteristic influences the energy overhead depended on the type of radio used. Different types of radios were shown and explained, after which it was shown how to calculate energy overhead. Finally, the calculation of energy efficiency was discussed.

3 ASPECTS OF TCP

The protocol known as TCP has become the de facto standard high level protocol used in large (inter)networks. It became the best known transport protocol, through the enormous growth of the Internet in both size and popularity. In this chapter, it will be explained how TCP works, what keeps it from reaching high levels of energy efficiency on wireless links and what can be done to remedy this.

3.1 Transport Control Protocol

At the lowest level, computer communications networks provide unreliable packet delivery. Packets can be lost or destroyed when transmission errors interfere with data, when network hardware fails, or when networks become too heavily loaded to accommodate the load presented. Networks that route packets dynamically can deliver them out of order, deliver them after a substantial delay, or deliver duplicates. At the highest level however, applications programs often need to send large volumes of data from one computer to another. A general purpose (connection oriented) protocol that provides reliable in-order delivery of data over all these kinds of low level networks, is required to be able to efficiently code networked applications and to provide a means to knit networks together into one large (global) network. TCP provides just this.

3.1.1 Reliability

To be able to provide reliable delivery, even though TCP packets themselves may be lost or duplicated, TCP uses positive acknowledgements (with retransmissions). Such schemes are also known as ARQ (Automatic Repeat reQuests) schemes. It requires the recipient to communicate with the sender, by sending back an acknowledgement message for each packet it receives correctly. The sender keeps a record of each packet it sends and waits for an acknowledgement before sending the next packet. The sender also starts a timer when it sends a packet and retransmits the packet if the timer expires before the acknowledgement arrives. In this way packets that are lost will be retransmitted until the sender receives an acknowledgement indicating the recipient has correctly received the packet.
The second reliability problem arises when the underlying packet delivery system duplicates packets. Duplicates can also arise when networks experience high delays that cause premature retransmissions. To solve this problem each packet is assigned a sequence number and the receiver is required to remember which sequence numbers it has received. To avoid confusion caused by delayed or duplicate acknowledgements, each acknowledgement carries the same sequence number as the packet it is supposed to acknowledge.

3.1.2 Sliding Windows

The retransmission scheme mentioned above leaves a substantial amount of bandwidth unused because it must delay sending a new packet until it receives an acknowledgement for the previous packet. To operate more efficiently TCP uses a sliding window scheme. Such a scheme allows the sender to transmit multiple packets before waiting for an acknowledgement. The easiest way to envision the operation of a sliding windows scheme is to think of a sequence of packets to be transmitted. The protocol then places a small window on the sequence and transmits all packets that lie inside the window. Once the sender receives an acknowledgement for the first packet in the window, it slides the window along and sends the next packet.

A packet is called unacknowledged if it has been transmitted but no acknowledgement has been received. So the number of unacknowledged packets is constrained by the window size. With a window size of one packet, this sliding window scheme behaves exactly the same as the scheme mentioned above. By setting the window size to a large enough value, it is possible to eliminate network idle time completely. A sequence of packets with a sliding window is shown in the figure below.

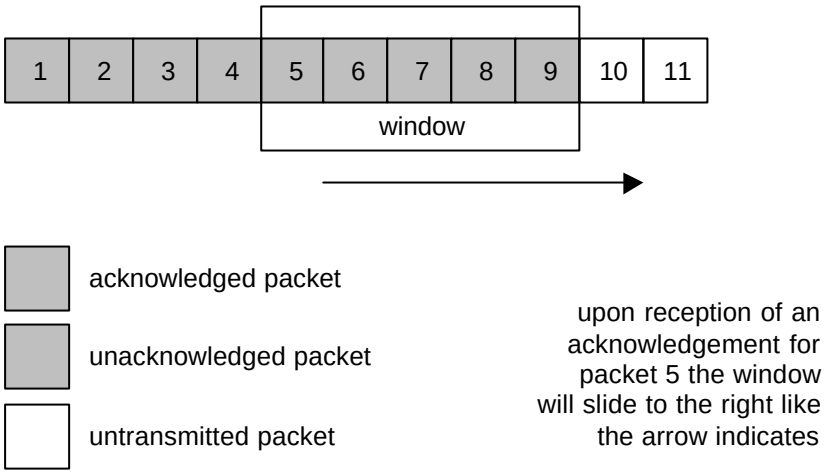

Figure 3.1: The sliding window mechanism.

3.1.3 Acknowledgements and Retransmission

Because TCP may send data in variable length packets, and retransmitted packets can include more (or less) data then the original, acknowledgements cannot easily refer to packets. Instead they refer to a position in the stream (the data that needs to be transmitted) using stream sequence numbers. At any time, the receiver will have reconstructed zero or more bytes contiguously from the beginning of the stream, but may have additional pieces of the stream from packets that arrived out of order. The receiver always acknowledges the longest contiguous prefix of the stream that has been received correctly.

This acknowledgement scheme is called *cumulative* because it reports how much of the stream has accumulated at the receiver. Such a scheme has both advantages and disadvantages. One advantage is that acknowledgements are both easy to generate and unambiguous. Another advantage is that lost acknowledgements do not necessarily force retransmission. A disadvantage however is that the sender does not receive information about all successful transmissions.

3.1.4 Timeout and Retransmission

Like other reliable protocols, TCP expects the destination to send acknowledgements whenever it successfully receives new octets from the data stream. Every time it sends a packet, TCP starts a timer and waits for an acknowledgement. If the timer expires before data in the packet was acknowledged, TCP assumes that the packet was lost or corrupted and retransmits it.

TCP however, is intended for use in an internet environment. In an internet, a packet traveling between a pair of machines may traverse a single, low-delay network, or it may travel across multiple intermediate high-delay links. Furthermore, the total delay between the origin and destination of data, depends on network traffic on intermediate links, and can therefore vary

over time. Thus it is impossible to choose a timeout value a priori that will suit each situation optimal. To solve this problem, TCP does not use a fixed timeout value but measures the round trip time of data and updates its timeout value accordingly.

3.1.5 Window Size and Flow Control

TCP allows the size of the sliding window to vary over time. Each acknowledgement contains a window advertisement that specifies how much data the recipient is prepared to accept. This can be seen as specifying the receiver's current buffer size. In response to an increased window advertisement the sender increases the size of its sliding window and in response to a decreased window advertisement it does the opposite.
The advantage of a variable window size is that it provides flow control. Through these window advertisements the receiver can control the rate at which the sender transmits data. Having a mechanism for flow control is essential in an internet environment, where machines of various speeds and sizes communicate through networks and routers of various speeds and capacities. There are really two independent flow problems. First, internet protocols need end-to-end flow control between the sender and the ultimate receiver. Window advertisements provide this kind of flow control. Second, internet protocols need intermediate flow control to handle congestion on intermediate networks.

3.1.6 Response to Congestion

Congestion is a condition of severe delay caused by an overload of packets at an intermediate switching point (e.g., a router). When congestion occurs, delays increase and the router starts to queue packets until it can route them. Of course each queued packet is stored in memory and a router has only finite memory. In the worst case, the total number of packets arriving at the congested router grows until the router reaches capacity and starts to drop packets.
Endpoints do not usually know if, where and how congestion occurred. Senders only experience timeouts for the packets that were dropped by the router. Under normal circumstances TCP would simply retransmit the packet, thereby increasing traffic. This aggravates congestion instead of alleviating it.
To avoid congestion, the TCP standard recommends using two techniques known as slow-start (with congestion avoidance) and multiplicative decrease. To control congestion TCP maintains a second limit to the window size (besides the advertised window). This limit is called the congestion window limit. The allowed window size of the sender is then at all times the minimum of both limits.
Multiplicative decrease reduces the congestion window limit by half, upon every loss of a packet. So multiplicative decrease can be seen as the mechanism that slows TCP down in case of congestion. When TCP no longer experiences congestion on its path, it uses slow-start (additive) recovery. Slow start begins with a congestion window limit of one and increases it for every acknowledgement it receives. Once the congestion window limit reaches one half of its original size before the congestion, congestion avoidance takes over. During congestion avoidance, it increases the congestion window only if all packets in the window have been acknowledged.

3.2 Problems of TCP

In this chapter some characteristics of TCP will be discussed that make it less suitable for wireless links.

3.2.1 Large headers

TCP was intended to be a highly deployable transport protocol. It has a lot of features and options, some of them rarely used, which make it suitable for operation on a wide range of (inter)networks. When TCP became popular, an increasing number of changes and additional options were proposed. Some of these options are widely used today. To accommodate the most basic features, TCP has a header size of 40 bytes. This is a fixed size, which means that even though not all header fields will be used, the header size will still be 40 bytes. When widely used options are activated the size can grow to 80 bytes.
This means that for every packet, there are 40 to 80 bytes overhead. An acknowledgement adds another 40 to 80 bytes to the overhead. This means that for packets with a 1000 byte payload, TCP has a data overhead of about 8% (without retransmissions). As can be seen, there is lots of room for optimization here.

3.2.2 Simple acknowledgement scheme

The acknowledgement scheme employed by TCP is fairly simple and does not allow an efficient retransmission scheme. Even though some optimizations have been proposed, TCP's standard scheme always remained unchanged, so no incompatibilities were introduced.
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. Based on this incomplete information the sender can not know what the most energy-efficient retransmission scheme will be. A more advanced acknowledgement scheme will be easy to implement and will undoubtedly increase the energy efficiency. An example of what the receiver acknowledges and how that differs from the actual situation is given next.

Example 3.1. Consider the following receiver state and the sender's view of it, both listed in Figure 3.2.

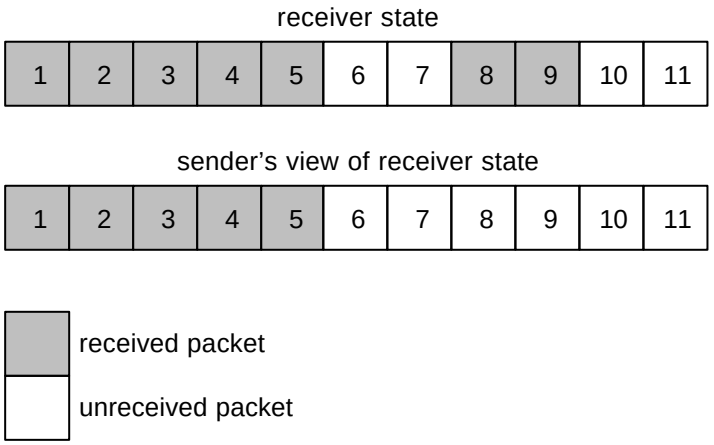

Figure 3.2: Example based on the acknowledgement scheme of standard TCP.

Assume the sender has sent a total of 9 packets. As can be seen the receiver has received all packets up to and including packet 5. Packet 6 and 7 were lost however, after which the receiver received packet 8 and 9. Upon reception of both packets, the receiver has to send an acknowledgement. Because of the positive

cumulative acknowledgement scheme, the receiver sends acknowledgements for packet 5.
Because the sender receives multiple acknowledgements for packet 5, it knows *something* went wrong. It can safely assume packet 6 was lost but nothing more. It now has two options, both of which are potentially inefficient. It can either send one packet (number 6) or all packets (numbers 6 up to and including 9).
If it would retransmit all packets, two packets would be sent too much. However, if the sender follows the standard and retransmits only packet 6, it must wait for the acknowledgement before it can decide what and how much to send next. Thus, it reverts to a simple protocol and may lose the advantage of having a large window.

3.2.3 Loss is considered congestion

TCP was designed with highly reliable links in mind. When it encounters packet loss it interprets this as congestion. In a (highly reliable) wired network this is a valid choice because in such setups congestion is the major source of packet loss. On wireless links however, the higher bit error rates cause much more packet loss (due to errors) than generally encountered on wired links. Interpreting all packet loss as congestion is not a realistic solution on wireless links, because from an energy efficiency point of view the ideal response to congestion differs from that to (burst) errors. Using an optimized window size management scheme, which also considers (burst) errors as the cause of lost packets will probably yield an increase in energy efficiency.

3.2.4 Complete reliability

Complete reliability may not seem a problem, but there are situations in which TCP's complete reliability is undesirable. When receiving streaming audio (or video) with TCP, the protocol will rerequest all lost data. These rerequests will make sure the application (e.g. a media player) will receive all data. The extra latency introduced, will probably make the playback stall for a time and then fast-forward to the part where it was supposed to be by then. So rerequesting lost data has little use in such situations because the data will arrive too late. Since streaming media can usually be enhanced to cope with reasonable amounts of data loss, it would be better not to send rerequests for lost data (up to a reasonable level) in that case.
Note that UDP (user datagram protocol) could be considered as a replacement for TCP in such cases. Just like TCP, UDP is a protocol that works on top of the IP protocol. Unlike TCP, it is connection-less and offers no reliability at all. Basically UDP offers too little features and too low reliability to be a real improvement over TCP. Using UDP shifts the problem to the application, because when using UDP, the application is responsible for connection setup/termination and the creation/handling of acknowledgements.

3.3 Possible solutions

In this chapter possible solutions to the energy efficiency problems of TCP will be discussed. Some of these solutions are mere concepts while others are extensions to existing protocols or even complete new protocols. All of these protocols try to optimize TCP (for wireless networks) in one way or another. Unfortunately most of them try to optimize the performance of TCP instead of the energy efficiency. Energy efficiency, however, depends only on byte- and time overhead. These two metrics are also often used to measure the performance of TCP, so there is a large overlap.

3.3.1 Header compression

To address the data overhead of TCP several proposals have been voiced to compress the headers of TCP. This was often done with low-bandwidth serial (wireless) links in mind. TCP/IP header compression was first standardized with RFC 1144 [JAC90] (and later with RFC 2507 [DEG99], RFC 2508 [CAS99] and RFC 2509 [ENG99]). The scheme only works on single hop links (i.e.: there are no intermediate hosts) and needs to be supported by both end points. Although it only works on single hop links, this only applies to the compression of the TCP connection. The actual connection can still travel a path with many intermediate hops. The compression is transparent to other hosts except for the two end points of the single hop link on which the compression takes place.

In [JAC90] the standard TCP/IP header fields were analyzed and for each field the way the values change during a TCP/IP connection were examined. Four different types of changes are defined and for each type a (new) representation method is chosen. For instance large integers, which only change slightly with each packet are represented by a small integer, which only represents the change from the last packet. This type of change is known as a *delta change* and the new representation is called a delta value accordingly. Almost all header fields are made optional and are only included if a certain flag in the compressed header is set.

One interesting optimization is the replacement of the IP addresses and port fields with a connection identifier. Each TCP/IP headers stores the IP address and the port of both the sender and the receiver. Combined, these fields require 12 bytes of the header. Since these fields do not change during a connection, a connection identifier gets assigned to the connection during the connection establishment. From then on the compressed headers in the connection only carry the 1-byte connection identifier.

A typical compressed header size is 3 bytes with the proposed scheme instead of 40 bytes. Of course this is a great improvement. In order to reach it however, the protocol has become less robust. Because the connection identifier is not always included and the two most used options are replaced by delta values, a lost packet can cause all subsequent packets to be misinterpreted. Naturally, checks are proposed to remedy this, but the necessary error recovery scheme can still cause normal valid packets to be discarded. This extra overhead will probably cause severe performance penalties on wireless links because of the high packet loss generally encountered. A less extreme compression method will almost certainly attain less data overhead than this scheme in case of high packet loss. This is shown in the next example.

> Example 3.2: Consider a transmission of 25000 bytes with packets that have a 1000 byte payload. Upon transmission one of the necessary 25 packets will be lost. Three versions of TCP will be compared. The first version is standard TCP with 40 byte headers and acknowledgements. The second is TCP with a robust form of header compression, which has 8 byte headers and acknowledgements. The final version is TCP with the described header compression. This version has 3 byte headers and acknowledgements but the loss of the packet will cause the next two packets to be misinterpreted and retransmitted. The amounts of transmitted bytes then are:
>
> 26 * 1040 + 25 * 40 = 28040 bytes for normal TCP
> 26 * 1008 + 25 * 8 = 26408 bytes for TCP with robust header compression
> 28 * 1003 + 25 * 3 = 28159 bytes for TCP with described header compression

As can be seen, the described header compression would have a higher data overhead than normal TCP in this situation. A more robust header compression method would perform best however.

3.3.2 Selective acknowledgements

The selective acknowledgement scheme is an extension to the TCP protocol that addresses some of TCP's problems by enhancing the acknowledgement scheme. It was standardized in RFC 2018 [MAT96] but a SACK (as selective acknowledgements are called) scheme was already mentioned in RFC 1072 [JAC88]. (An extension to RFC 2018 was published under RFC number 2883 [FLO00].) Both end points of the TCP connection need to support the SACK option in order to be effective.
The SACK scheme adds extra information to acknowledgements about the receiver's state each time TCP's standard positive cumulative acknowledgement scheme is lacking. This happens when there are 'gaps' in the data the destination host has received. Standard TCP would acknowledge all data up to the first gap but TCP with SACK can effectively bridge a gap by sending an extra SACK block. By sending more information in acknowledgements the sender is better able to react to the actual state of the link and the receiver. The difference in supplied information to the sender is shown in the next figure.

Example 3.3. Consider the same situation as in Example 3.1. However, this time there is also a SACK enabled receiver. The receiver state and both the senders' view of it are represented in the next figure.

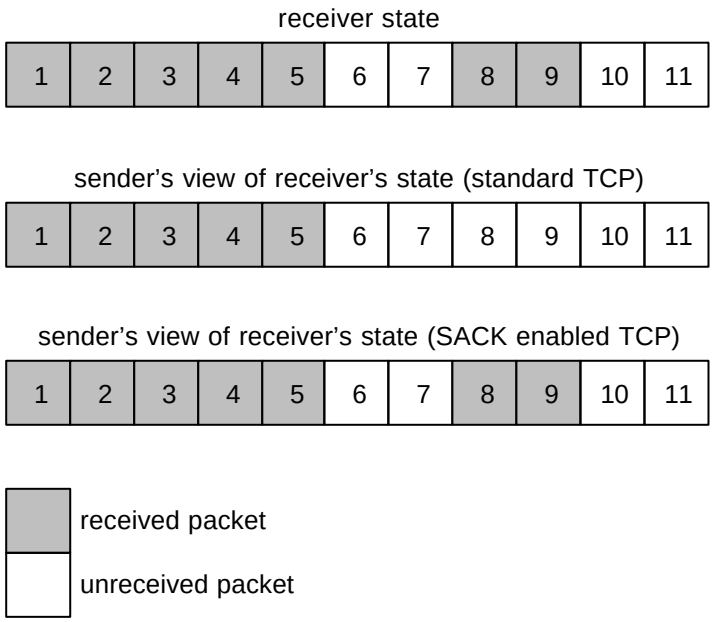

Figure 3.3: Example based on the acknowledgement scheme of standard TCP and SACK enabled TCP.

Remember that the standard TCP sender had to choose between two potentially inefficient courses of actions. The SACK enabled sender, however, knows what packets were lost and can simply retransmit those. This is always the most efficient method.

The SACK scheme can include any number of SACK blocks up to a maximum of four. This is because there is a limit on the size of TCP headers. All TCP options (including SACK blocks) are included in a special TCP header field called TCP options. The more options a TCP implementation uses the less space there is left for SACK blocks. In general TCP implementations that include SACK support, there is enough space left for three SACK blocks.
Even though its headers are larger, a TCP implementation with SACK support generally has less data- and time overhead than a comparable implementation without SACK support, because it can handle retransmits more efficiently. Because of the less data- and time overhead SACK also performs better (in terms of throughput) than other protocols as was shown in [FAL96].

3.3.3 Delayed acknowledgements

In principle a TCP receiver should acknowledge each packet that it receives. So each packet that reaches its destination immediately triggers a 40 byte acknowledgement (sometimes it can be piggybacked on normal packets bound for the other host however). This can of course be considered as a waste of bandwidth. To remedy this the TCP standard allows for a receiver to delay the sending of an acknowledgement for a period of time (with a maximum of 500 milliseconds) [COM95]. In this way multiple acknowledgements can be combined and/or the acknowledgement(s) can be piggybacked on a normal data packet.
This of course reduces data overhead. Unfortunately there are also a few drawbacks. The first disadvantage is that the importance of an acknowledgement increases. That is, if such an acknowledgement is lost more information on the state of the receiver is lost than would be the case with a undelayed acknowledgement. Because more information is lost, the consequences can be more severe, possibly increasing data- and time overhead. The second drawback is that time overhead will probably increase because the receiver will not immediately send an acknowledgement but will wait for a period of time before doing so. This will cause the sender's window to be built up more slowly.
Because TCP relies on acknowledgements to accurately estimate the round trip time, the sender is not allowed to combine too much acknowledgements. For every second data packet an acknowledgement should be sent, further reducing the decrease in data overhead.
Below are listed two graphs that show the energy overhead for Tahoe (a version of TCP) with and without delayed acknowledgements. For more information on the used test method and why only these graphs suffice to compare the performance, see Chapter 5 and especially Paragraphs 5.1, 5.2, 5.3 and 5.6.1. In these paragraphs, everything needed to understand these graphs will be explained. At this time, the graphs could use some explanation. While the left axis speaks for itself the lower axis might be unintelligible. It roughly resembles the quality of the channel. The left side of the axis resembles a high quality channel (little errors) while the right side of the axis resembles a low quality channel (lots of errors). Above the graphs a scenario is mentioned. The scenario indicates the error model used. Scenario A uses fixed length bad states (burst errors) while scenario B uses variable length bad states.

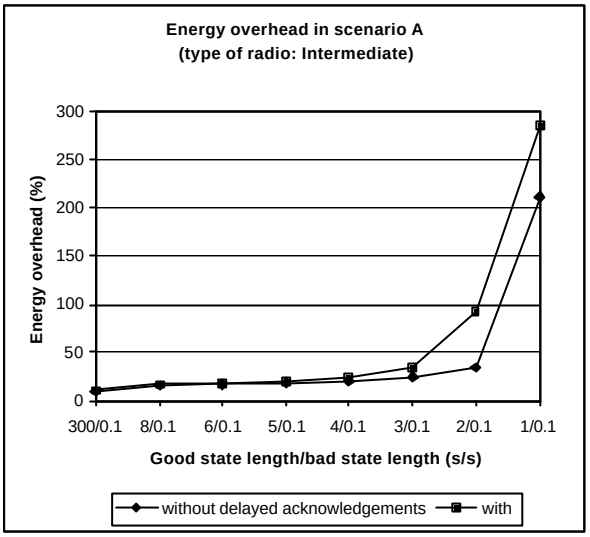

Figure 3.4: Energy overhead of delayed acknowledgements in scenario A.

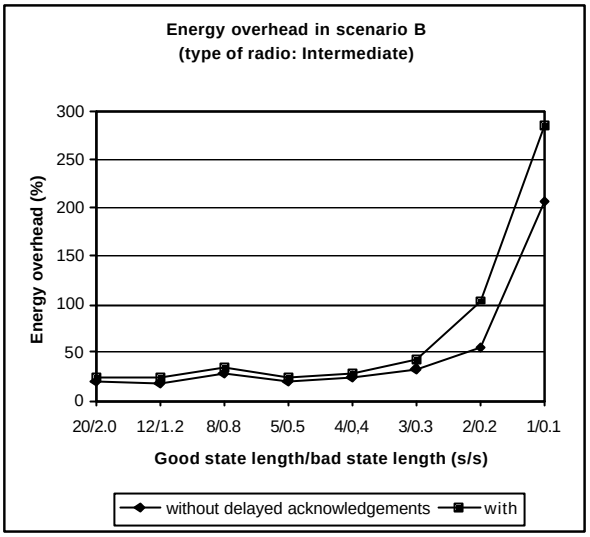

Figure 3.5: Energy overhead of delayed acknowledgements in scenario B.

In both graphs it is quite clear that the use of delayed acknowledgements increases the energy overhead of a protocol and thus decreases its energy efficiency. From an energy efficiency standpoint, delayed acknowledgements should be avoided.

3.3.4 Explicit congestion notification

TCP's flow control mechanisms rely on packet drops to detect congestion. When this happens TCP is already late in reacting because the congestion already occurred. It would be better if TCP could be notified when it is about to cause congestion so that it can react before packets are lost. Lost packets should be avoided because they will have to be retransmitted, increasing data (and time) overhead.

To improve TCP, an explicit congestion notification (ECN) system was proposed in RFC 2481 [RAM99]. In the proposed configuration both routers and end points should be ECN capable. When a router predicts that congestion will occur, it marks packets by setting a special ECN field in TCP packets. The receiver of the packet can then take appropriate measures to make sure the transmission rate of the connection will be reduced. In such a way congestion can often be avoided.

Although this scheme can decrease data- and time overhead (because less retransmits should be necessary) it is also possible it increases time overhead. When a TCP connection is incorrectly told to decrease its transmission rate for instance. Overall this scheme can decrease energy efficiency but is only suited for multi-hop paths. E^2TCP will be used on a single-hop path. Thus it will not experience congestion on intermediate nodes, which is the reason there is no need to make use of explicit congestion notification in E^2TCP.

3.3.5 Forward error correction

Forward error correction (FEC) is often mentioned or proposed when the performance of (multimedia) streams over (wireless) links is found to be lacking. FEC protects data by adding a little bit of redundancy to each data unit. When errors occur on the wireless link and the data unit has errors in it, the FEC scheme can correct those errors (up to a certain amount). When the amount of FEC is increased the protected data can withstand more errors. On the other

hand the processing power increases as well as the total size of the data that needs to be transmitted. This of course also results in increased transmission times.
ARQ schemes provide exactly as much error correction as is needed, because they only kick in when errors actually occur. The amount of error correction used in FEC schemes however, does not directly depend on the amount of errors that occurred. It in stead depends on the amount of expected errors and so the amount of FEC is decided upon in advance. Unfortunately predicting the future is still impossible, even for FEC schemes. So most of the time FEC schemes will either offer too much protection or too little. When too much protection is offered, too much data has been sent which means data- and time overhead could have been reduced. When too little protection was offered, the receiver could not correct the errors in the packet and the packet should be transmitted again. This is of course also the case when using an ARQ scheme but with the FEC scheme the packets are larger because of the added protection. So ARQ schemes can be said to be more energy efficient than FEC schemes [HAV99].
Furthermore using FEC in the transport layer is only possible when the lower layers also hand packets with errors to the transport layer. Because no assumptions were to be made about the lower layers FEC can not be used in E^2TCP.

3.3.6 I-TCP

Indirect TCP [BAK95] is a solution specifically designed to be used with wireless connections. It was one of the first proposals to use split connections. The connections are named so because connections between the mobile host and fixed hosts are split up in two separate connections at the base station: one regular TCP connection between the base station and the fixed host and another connection between the mobile host and the base station. This last connection is a single-hop connection over a wireless link and there is no need to use standard TCP. Rather a more optimized wireless-link protocol can be used which solves some of TCP's problems on wireless links. Another advantage of split connections is that it effectively separates the flow and congestion control at the base station. This way flow and congestion control on each sublink can be optimized separately from the other.
Indirect TCP largely refrains from changing the protocol on the wireless link and solely focuses on the split connection principle. Still this proposal is able to obtain impressive results [BAK97] and the split connection principle is very well suited for wireless access to a TCP/IP network.

3.3.7 Protocols inspired by I-TCP

The obvious advantages of the split connections approach inspired some other protocols. These protocols all use a (lightly) optimized version of TCP on the wireless links to further improve performance over I-TCP.
The Berkeley Snoop Module [BAL95] is another proposal to tackle the performance problems of TCP on wireless links. Just like I-TCP it proposes a split connection setup but the Snoop Module is more active than the I-TCP setup. The Snoop Module caches packets and performs local retransmissions as soon as packet loss is detected. This further increases performance over I-TCP.
The M-TCP protocol [BRO97] also performs instant local retransmissions, just like the Snoop Module. Furthermore it adds user data compression support to decrease the payload size and through special flow control mechanisms it allows the sender to resume sending after breaks (like handoffs) at full speed.

In [RAT98] another protocol for networks with wireless links is proposed: WTCP. It closely resembles I-TCP but stresses the importance of accurate round trip time sampling and is constructed accordingly.

3.3.8 Delayed duplicate acknowledgements

The delayed duplicate acknowledgement scheme [VAI99] tries to mimic the behavior of the Snoop protocol but does it TCP-unaware, unlike the Snoop Module, which is TCP-aware. A TCP-aware protocol needs to look in the TCP headers in order to take appropriate measures. It is possible however that the TCP headers are not readable by intermediate hosts (because of encryption). This scheme tries to behave in the same way as the Snoop Module without examining the TCP headers. Because this scheme has less information to base its decisions on, it performs slightly worse than the Snoop Module. On the other hand it can be used in more situations.

3.3.9 Mobile-TCP

The Mobile-TCP protocol as described in [HAA97] is one of the few protocols that try to optimize for energy efficiency. It also employs the split connection principle but drastically changes the protocol on the wireless link. An asymmetric protocol is proposed: the protocol stack running at the mobile host is kept as small and simple as possible and as much processing is offloaded to the base station.
In order to save energy the protocol uses very small custom headers and makes use of the connection ID principle, known from header compression. The implementation at the mobile host also features as few timers as possible and the protocol does not use the sliding windows principle, allowing for smaller buffers. Furthermore, the protocol for instance, does not support flow control or resequencing.
Overall this protocol sacrifices so much in order to save on processing power, it will undoubtedly spend more energy on retransmits than other (energy efficient) protocols. Because the relative power consumption of processors, compared to radios, keeps decreasing, it is not that interesting to focus on minimizing required processing power. Minimizing data- and time overhead seems a better way to increase energy efficiency.

3.3.10 PRTP

The partial reliable transport protocol (PRTP) [BRU00] was not specifically designed with wireless links in mind but with a type of traffic. The strict reliability guarantees of TCP make it less suited for many multimedia applications. Often when streaming media experiences small amounts of data loss, retransmission is actually undesirable. They cause the playback to stall and the retransmitted data will already have ‘expired’ upon arrival. Furthermore most streaming media can withstand small amounts of data loss without a noticeable loss in quality. For such connections the partial reliability transport protocol, which is compatible with TCP, offers a solution.
It allows the application to set a lower limit on the reliability. When the overall reliability does not drop below the limit, the receiver will not ask for a retransmission. If the overall reliability of the connection is in danger, the receiver will ask for retransmissions in the standard TCP manner. This enables a PRTP receiver to correctly operate with a TCP sender.
When the application can deal with data loss the reliability can be set to values as low as 90%. In [GAR00] it is shown that with an optimized JPEG coder, images can tolerate up to 10% data loss before the quality of the images becomes noticeably less.

In case of packet loss PRTP performs very well compared to other versions of TCP. Please note that this does mean that the PRTP receiver receives not all data. Because wireless links generally experience much packet drops, PRTP is extremely well suited for streaming media over wireless links.

3.3.11 Optimized window management

One might think that this concept does not deserve it's own paragraph. However, the way in which TCP reacts to (burst) errors on wireless links leaves a lot to be desired. Every packet loss is considered to be caused by congestion. For each packet loss TCP will drastically reduce its transmission speed so experienced congestion will quickly be cleared. The assumption that each packet loss is caused by congestion is valid in wired networks. Because of the high reliability of such links, the largest portion of packet loss is indeed caused by congestion. However, on wireless links, the assumption is not valid. Because of the relatively low channel quality of wireless links, a lot of packets will be corrupted while in transit. For each of those errors TCP will also reduce its transmission speed. A huge loss in time overhead can therefor be reached by optimizing the window management scheme of a protocol for wireless links.

3.3.12 Conclusions

Some of the concepts and protocols presented in this chapter are not applicable when optimizing for energy efficiency. They either focus on ways to improve performance that do not increase energy efficiency or they optimize the power consumption of the wrong part of the system. The other concepts presented here will be used in order to design a energy-efficient transport protocol and these include:

- split connections
- small headers
- selective acknowledgements
- partial reliability
- optimized window management

4 E²TCP

In this chapter, E²TCP will be described in detail. First an overview of the architecture of E²TCP will be given, where the reasons for and expectations of the changes to TCP will be discussed. After that, the header format will be explained in detail, followed by the selective acknowledgement scheme. Finally, the window management will be described, as well as the partial reliability mechanism.

4.1 Architecture overview

One of the primary goals of this project was to design a transport protocol that would be compatible with TCP. It was therefore only self-evident that TCP would serve as a basis for this new protocol. Because E²TCP is derived of TCP, its architecture and mechanisms are roughly the same. On four points, however, adjustments were made to increase the energy efficiency of the protocol. These points are the headers, the acknowledgements, the window management and the reliability requirements. All four changes will be introduced in the following paragraphs.

4.1.1 Headers

The large header size of TCP was first introduced as a problem in Paragraph 3.2.1. The unnecessarily large headers are the cause of equally-unnecessary data overhead. The custom headers of E²TCP are the result of a rather straightforward implementation of some of the ideas of header compression standards, presented in Paragraph 3.3.1. The main principle that was used was: if it is not necessary to transmit a certain header field, don't do it. This principle is so self-evident; one could wonder why such a system was not incorporated in the TCP standard.

All header fields of TCP/IP datagrams were analyzed whether they should be included in the headers of E²TCP at all, whether they should always be sent or whether they were to be made optional. Such an optional header field will then only be sent if it is necessary to do so. Care was taken to keep the headers robust because the problems of a non-robust compressed header system, explained in Example 3.2, have to be avoided.

The header size is reduced from 40 bytes to 8 bytes (in most situations). When using 1000 byte packets for instance, the data overhead introduced by the headers of E²TCP will be 1.6% as opposed to 8.0% for the headers of standard TCP. Because less data has to be transmitted, the time overhead will probably also decrease somewhat, although perhaps not as much as the data overhead. This will probably result in a decrease in energy overhead of about 5%. The details of the headers of E²TCP will be discussed in Paragraph 4.2.

4.1.2 Acknowledgements

The simple acknowledgement scheme of TCP, introduced as a problem in Paragraph 3.2.2, is another point of TCP that could be improved to increase energy efficiency. In case of missing packets the sender simply has not enough information about the state of the receiver. On those occasions, it is possible the sender not always decides on the optimal course of action. A solution to this problem is the use of selective acknowledgements, which were introduced in Paragraph 3.3.2.

An E²TCP receiver is able to construct selective acknowledgements. It does this by adding one or more SACK blocks to an acknowledgement. The headers (also used as acknowledgements) of E²TCP allow a maximum of two SACK blocks to be sent. This enables the receiver to fully acknowledge a received stream with two sets of subsequent missing

packets. When the sender receives such an acknowledgement it is always able to choose the most energy efficient course of action. Because of the diminishing returns of adding more SACK blocks and the fact that SACK blocks increase the size of the acknowledgements, a maximum of two SACK blocks is used.
Although SACK blocks increase data overhead because the acknowledgements increase in size when these blocks are used, the effect of selective acknowledgements on the energy efficiency will be positive. This is because the sender is able to react in an optimal way to lost packets, which slightly decreases data overhead (because of less retransmits) and reduces time overhead substantially (because of a better utilization of the available bandwidth). However, giving an exact estimate of the increase in energy efficiency is impossible. The implementation of selective acknowledgements in E^2TCP will be explained in detail in Paragraph 4.3.

4.1.3 Window management

The problems of TCP on wireless links with respect to its window management were introduced in Paragraph 3.2.3. 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. This is because the inherent unreliability of wireless networks, which causes a substantial amount of packets to be lost because of errors on the link itself. So, the window management of TCP should be altered to include (burst) errors as a possible source of packet loss, as was indicated in Paragraph 3.3.11.
The window management mechanism of E^2TCP differs on four points from that of TCP. First of all, E^2TCP 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^2TCP will be used on a single-hop link and no packet reordering can take place on such a link. Under the same conditions 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^2TCP reacts to (burst) errors in a different way. If few errors occur, E^2TCP considers this to be the result of normal static and barely reduces its transmission speed. When lots of errors occur, E^2TCP considers a burst error to be the cause and drastically reduces its transmission speed. This way, E^2TCP reacts to (burst) errors in a very energy efficient way, as will be shown in Paragraph 5.6. It should be noted that this new window management scheme relies on selective acknowledgements to detect the number of errors. Both data and time overhead will decrease because of this change.
E^2TCP also features a minimum window size, which is the third point on which the window management of TCP and E^2TCP differ. This minimum window size causes E^2TCP 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 an extra timer. The timers used in E^2TCP are similar to the transmission timer in TCP, only one is used for transmissions and one is used for retransmissions. An extra timer increases the responsiveness of the protocol to changes on the channel but also increases the complexity of the protocol. One extra timer is considered to be a good tradeoff. This change will also decrease time overhead.
The four changes will undoubtedly cause a decrease in energy overhead but it is impossible to give an estimation of the size of that decrease. The details of the implementation of the window management scheme can be found in Paragraph 4.4.

4.1.4 Reliability requirements

Because the strict reliability requirements of a TCP connection are not always desirable, as was shown in Paragraph 3.2.4, the concept of partial reliability was developed, which was introduced in Paragraph 3.3.10. When transmitting streaming media, energy can be saved if unwanted retransmits can be avoided. Partial reliability provides a way to do this, by enabling the application to set the minimum desired reliability of the channel.
The implementation of partial reliability in E^{2}TCP is rather straightforward. The receiver keeps track of how much data was successfully received and how much was lost. If it detects packet loss it will check if the actual reliability still exceeds the minimum desired reliability and if so, will simply acknowledge the lost packet. The sender will think it was received correctly and will refrain from retransmitting.
This simple mechanism will be able to decrease the energy overhead. How much is uncertain but its effect will increase when channel conditions deteriorate. This is because the effect of stopping retransmits increases when more packets are lost. The details of the implementation will be discussed in Paragraph 4.4.6.

4.2 Header format

The headers of E^{2}TCP packets will be explained in this paragraph. Because E^{2}TCP needs to be compatible with TCP/IP, the headers of IP and TCP will be examined first. Based on that information, a decision can be made on what header fields should be included in E^{2}TCP headers, which will be explained in Paragraph 4.2.3.

4.2.1 IP header

The Internet calls its basic transfer unit an (IP) datagram. Such a datagram is divided into a header (of 20 bytes) and a data area in the following way:

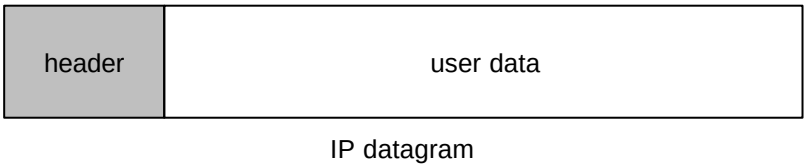

Figure 4.1: An IP datagram.

According to [DEG99], all fields in headers can be classified into one of the following four categories depending on how they are expected to change between consecutive headers in a packet stream. These four categories are:

- Inferred: The field contains a value that can be inferred from other values, and thus need not be transmitted.
- Nochange: The field is not expected to change during the packet stream. Such a value only has to be transmitted once.
- Delta: The field may change often but usually the difference from the field in the previous header is small, so that is more efficient to send the deviation from the previous value rather than the current value.
- Random: The field changes unpredictably and should therefore probably be sent in full.

Now the general layout of a datagram and the classes of headers has been described, the header can be described in more detail. The following figure presents the IP (version 4) header format:

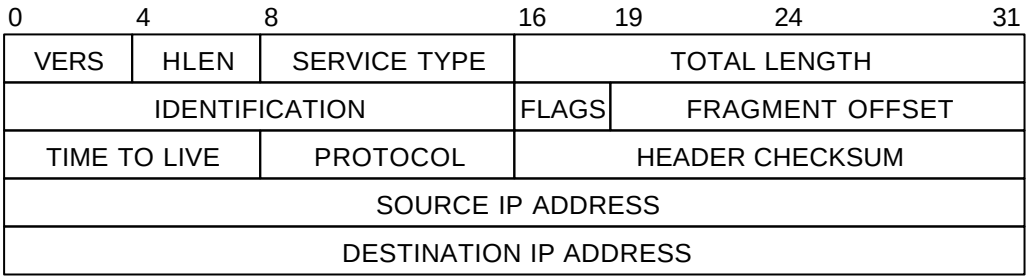

Figure 4.2: The IP header format.

The header fields of an IP datagram are:

Field	Type	Description
Protocol version (VERS)	nochange	This field contains the version of the IP protocol that was used to create the datagram and is of course not expected to change within a packet stream. On the wireless link, E^{2}TCP will operate and it does not need to know which version of TCP is used on the wired part of the connection. Therefore this field can be omitted from the E^{2}TCP header.
Header length (HLEN)	inferred	This field contains the length of the header but this can be determined by other means as well, so there is no need to include it in the header of an E^{2}TCP packet.
Service type	nochange	With this field the sender can specify the type of transport desired. It is, however, often ignored by hosts and routers and is not expected to change. E^{2}TCP does not support different types of services and it does not need to include this field in its headers.
Total length	inferred	This field contains the length of the complete datagram but that will also be specified by any reasonable link-level protocol. It is unnecessary to include it in E^{2}TCP headers.
Identification	random	For each datagram a unique number is stored in this field. It is used to refragment split up datagrams. On a point-to-point link (where E^{2}TCP will operate) no fragmentation will take place and each packet will be identified by its sequence number or acknowledgement number (TCP header fields).
Flags	random	These flags control the fragmentation of the datagram and can be left out of the header.
Fragment offset	random	This field is used in datagram refragmentation and does not need to be included in E^{2}TCP headers.

Field	Type	Description
Time to live	nochange	This field contains the maximum number of hops the datagram is allowed to take over the internet and is not expected to change. Because E^{2}TCP operates on a single-hop link this field would serve no purpose.
Protocol	nochange	This field indicates the next level protocol used in the data portion of the IP datagram. Because E^{2}TCP only has one mode of operation this field can also be omitted.
Header checksum	random	This is a checksum on the header only. In E^{2}TCP a checksum will be used but like the checksum field of a TCP header it will protect the entire datagram and not just the header.
Source IP address	nochange	This field stores the source address and will be included in E^{2}TCP headers.
Destination IP address	nochange	The destination address is contained in this field and will be included in an E^{2}TCP header just like the source IP address.

Table 4.1: The IP header fields.

IP also allows some optional extra information to be sent in its headers. Timestamps and source routes are among them. As said these fields are optional and need not be supported by E^{2}TCP. Furthermore the base station can still support most of them, so these options can be used on the wired path of the connection.
Of all these header fields only three will be included (in one way or another) in the E^{2}TCP header: the source- and destination IP address and the checksum.

4.2.2 TCP header

A TCP packet is encapsulated in an IP datagram and is divided into a header (of 20 bytes) and payload in a way similar to an IP datagram.

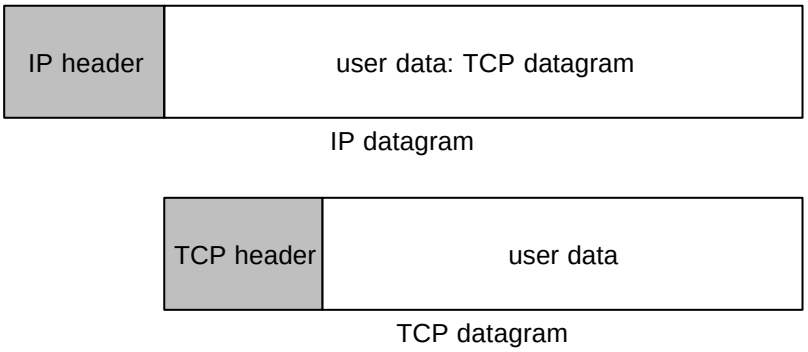

Figure 4.3: A TCP datagram within an IP datagram.

A TCP header can be presented as follows:

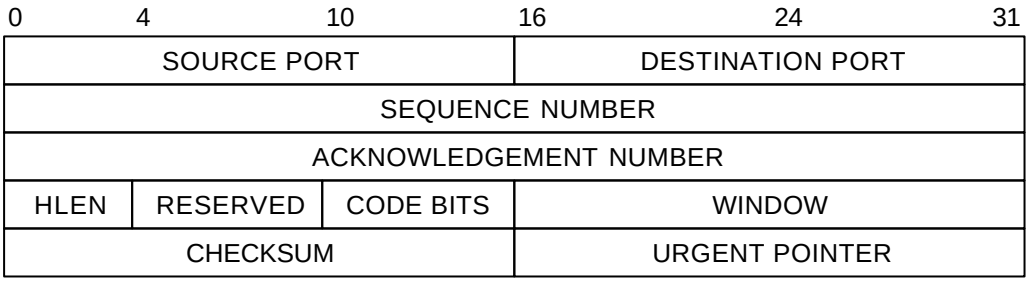

Figure 4.4: The TCP header format.

The fields of a TCP header are:

Field	Type	Description
Source port	nochange	The port number of the sender is stored in this field and should be included in E^2TCP headers so it is compatible with TCP.
Destination port	nochange	This field stores the port number of the receiving side and should also be included in an E^2TCP header.
Sequence number	delta	This field indicates what part of the data stream is included in this datagram. This field is required to interoperate with TCP.
Acknowledgement number	delta	This field stores a number, which indicates what part of the data stream has already been received by the destination and should not be omitted from the headers of E^2TCP.
Header length (HLEN)	random	In this field the length of the TCP header is stored. It is unnecessary to include it in E^2TCP headers.
Code bits	random	This field contains 6 code bits. These are: • URG, which indicates whether the urgent pointer field is valid or not. • ACK, which indicates whether or not the acknowledgement field is valid. • PSH, indicates if this packet requests a push. • RST, indicates if the connection should be reset. • SYN, indicates if the sequence numbers should be synchronized. • FIN, indicates if the sender has reached the end of its byte stream.
Window	random	The window field is used by the receiving side to exercise flow control over the sender. It will be included in E^2TCP so flow and congestion control is possible.

Field	Type	Description
Checksum	random	As indicated in the previous paragraph, a checksum will be included in the E^2TCP headers, which will protect the entire E^2TCP datagram.
Urgent pointer	random	This field indicates which data in the packet is of a special urgent type, which deserves special treatment from the receiver. In order to interoperate with TCP, this field will be included in the headers.

Table 4.2: The TCP header fields.

TCP also allows some optional extra information to be sent in TCP headers. Timestamps and SACK blocks are among them. As said these fields are optional and need not be supported by E^2TCP. Furthermore the base station can still support most of them, so these options can be used on the wired path of the connection.
Of all these header fields the following will be used in E^2TCP headers: source- and destination port numbers, sequence and acknowledgement numbers, window, urgent pointer and, as already said, the checksum.

4.2.3 E^2TCP header

To get an overview of the fields that were chosen to be included in the headers of E^2TCP, they will be listed again with their type and size.

Field	Type	Size (in bytes)
Source IP address	nochange	4
Destination IP address	nochange	4
Source port number	nochange	2
Destination port number	nochange	2
Sequence number	delta	4
Acknowledgement number	delta	4
Window	random	2
Urgent pointer	random	2
Checksum	random	2

Table 4.3: The E^2TCP header fields.

But that is not all information that should be included in an E^2TCP header. Some flags to use for connection startup and termination (like the SYN and FIN code bits in TCP headers) are also required. Furthermore E^2TCP will have selective acknowledgement support so some SACK blocks should be included as well.
Because the source- and destination IP addresses and ports require 12 bytes of storage and will not change during a connection, they will only be sent the first time. In the E^2TCP headers a connection identifier field will also be included. During connection startup a connection identifier will be chosen, which –from then on– will only be used for that combination of source- and destination IP addresses and ports until the connection is terminated. This type of header compression is also used in various standards as was seen in Paragraph 3.3.1.

In the code bits field in a TCP header two bits are included that indicate whether or not the urgent pointer and acknowledgement fields are valid. One could wonder why they are still included in the TCP headers when they are not valid: if the code bits indicate they are not valid there is no reason to include the fields in the headers at all. To optimize the E^{2}TCP headers, fields that will not always be included will have a bit in the header indicating whether or not they are included. When they are included the receiver should conclude they are also valid.
An actual E^{2}TCP header will then look like this:

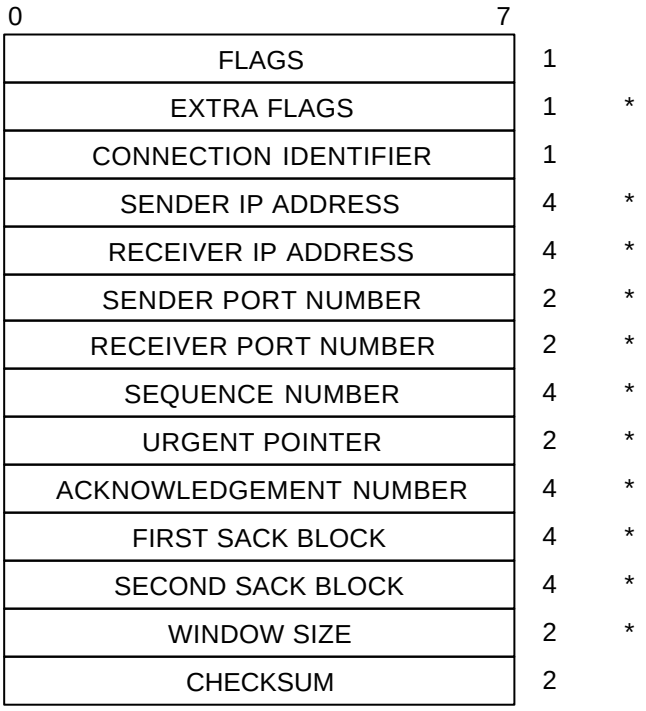

Figure 4.5: The E^{2}TCP header format.

The fields mentioned in the figure will now be explained in detail.

4.2.3.1 Flags

This field contains certain status bits that indicate how to interpret the rest of the header. Its size is one byte and it is not optional. This means every E^{2}TCP packet will feature this header field. The field looks like this:

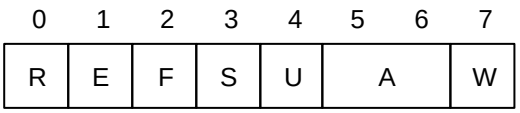

Figure 4.6: The flags field format.

The bit fields are:

Bit field	Size (in bits)	Description
Reserved	1	This bit field is reserved for future extensions/versions of E^{2}TCP. For now it should always be '0'.
Extra flags included	1	This bit field indicates whether or not the extra flag field is included in the header.
Full addresses included	1	If this field is set, the full addresses of both the sender and the receiver are included. This means the source- and destination IP address and port fields are included in the header.
Sequence number included	1	This bit field indicates whether the sequence number field is included in the header or not.
Urgent pointer included	1	This field is used to indicate if the urgent pointer field in included.
Acknowledgement type	2	This field indicates what kind of acknowledgement is included in the header. • '00' means there are no acknowledgement fields included. • '01' means only the acknowledgement number field is included. • '10' means that beside the acknowledgement number field, also the first SACK block is included • '11' means that all acknowledgement fields are included (the acknowledgement number and both SACK blocks).
Window included	1	This bit field indicates whether or not the window field is included.

Table 4.4: The bit fields of the flags field.

4.2.3.2 Extra flags

This field contains extra status bits that are needed on certain occasions. Its size is one byte and it is optional. The field looks like this:

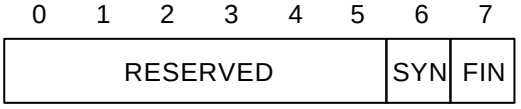

Figure 4.7: The extra flags field format.

The bit fields of the extra flag field are:

Bit field	Size (in bits)	Description
Reserved	6	This bit field is reserved for future extensions/versions of E^{2}TCP. For now it should always be '000000'.

Bit field	Size (in bits)	Description
SYN	1	This bit field is used to indicate that the sequence numbers should be synchronized. This is used on connection setup.
FIN	1	This bit field is used to indicate that the sender has reached the end of its data stream. It is used to terminate connections.

Table 4.5: The bit fields of the extra flags field.

4.2.3.3 Connection identifier

This field is used to store the connection identifier of the packet. Its size is one byte and it is not optional. The 1-byte size means that a mobile host running E^{2}TCP to connect to a base station (and the internet) has a maximum of 256 simultaneous connections. This should be more than enough, even for heavy use of the Internet.

4.2.3.4 Sender IP address

In this field the IP address of the sender is stored. Its size is four bytes and it is optional. This field should only be sent until a connection identifier has been agreed upon.

4.2.3.5 Receiver IP address

In this field the IP address of the receiver is stored. Its size is four bytes and it is optional. This field should only be sent until a connection identifier has been agreed upon.

4.2.3.6 Sender port number

This field is used to store the port number of the sender. Its size is two bytes and it is optional. This field should only be sent until a connection identifier has been agreed upon.

4.2.3.7 Receiver port number

This field is used to store the port number of the receiver. Its size is two bytes and it is optional. This field should only be sent until a connection identifier has been agreed upon.

4.2.3.8 Sequence number

In this field the sequence number of the last byte in the packet is stored. Its size is four bytes because complete sequence numbers are stored. Even though the type of the corresponding TCP header field was delta, E^{2}TCP will always transmit complete sequence numbers and not the difference with the last packet. Upon data loss a scheme which, only transmits the difference, can lose multiple packets because the correct decoding of each packet depends on the correct decoding of the previous packet. On wireless links with a high number of errors, such a scheme is unacceptable. The sequence number field is optional and is only used when the sender transmits data to the receiver. The following example shows which sequence number is stored:

> Example 4.1: Consider an E^{2}TCP packet, which payload consists of bytes 5, 6, 7 and 8 of the data stream. The sequence number field would then be used to store the number eight.

Please note that this differs slightly from the TCP sequence number field. Care must be taken that the base station converts the values.

4.2.3.9 Urgent pointer

This field is used to indicate urgent data is included in the packet. It is two bytes large and optional. When the urgent pointer included bit is set, a stream of urgent data is included in the payload of the packet. The urgent pointer indicates the end of the urgent data stream.

4.2.3.10 Acknowledgement number

This field is used to acknowledge data by the receiver. Its size is four bytes because sequence numbers are used to indicate what has been received and what not. For the same reason the full sequence number is stored in the sequence number field, it is done here as well. This field is also optional and will only be sent when the receiver needs to acknowledge data to the sender. For more information on how acknowledgement numbers are chosen, see Paragraph 4.3.

4.2.3.11 First SACK block

This field is used to store the first SACK block and is optional. It will only be used in certain cases where the receiver wants to acknowledge data to the sender. Its size is four bytes. For more information on SACK blocks, see Paragraph 4.3.

4.2.3.12 Second SACK block

This field is used to store the second SACK block is used in the same way as the first SACK block.

4.2.3.13 Window size

This field is used to store the limit on the window size the receiver sets for the sender. Its size is two bytes because TCP uses 16 bit unsigned integers to store the window size. The field is optional.

4.2.3.14 Checksum

In this field the checksum of the complete E^2TCP packet is stored. Its size is two bytes because the same checksum algorithm as in TCP is used. This field is not optional and should be transmitted with each packet to protect it.

4.2.4 E^2TCP header sizes

Because almost all fields in the E^2TCP headers are optional and only need to be transmitted when they are required, E^2TCP headers are usually quite small. A normal data packet will have a header of 8 bytes versus a 40 byte TCP header. Especially with small payloads the overhead will be reduced dramatically. Normal acknowledgements will have a size between 8 and 16 bytes depending on how many SACK blocks are used. TCP acknowledgements 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.

4.3 Selective acknowledgements

E^2TCP not only supports selective acknowledgements but also relies on them to effectively increase its energy efficiency. Because E^2TCP will work on a single-hop link and performs local retransmissions, it will know, when a packet is received out of order, that the intermediate packets were lost. It is able to do so, because no packet reordering can take place on a single-hop link. Upon noticing out of order packets, the receiver will indicate to the sender (with selective acknowledgements), that it has not received the intermediate packets.

Upon reception of an acknowledgement with SACK blocks the sender can immediately retransmit the lost packets and does not have to wait on timeouts or duplicate acknowledgements. This will reduce the time overhead of E^{2}TCP without increasing the data overhead.
When the destination host receives a packet it should always send an acknowledgement and acknowledge as much data as possible. Because E^{2}TCP depends on selective acknowledgements the receiver is always required to send as much SACK blocks as possible.
The acknowledgement number field should contain the number of the last byte in the contiguous received prefix of the stream. The following example shows this:

Example 4.2: Consider the following receiver state.

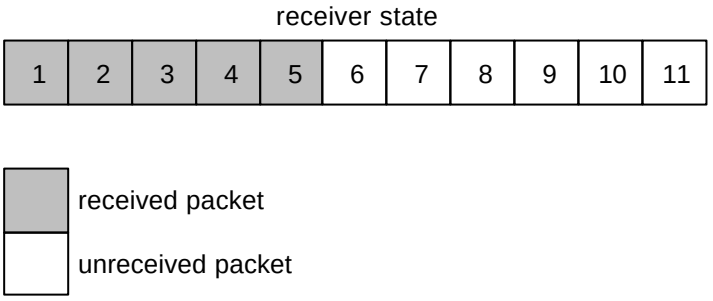

Figure 4.8: Example of a receiver state.

Because the receiver must acknowledge as much data as possible, it should acknowledge all packets up to and including packet 5. It is not allowed to only indicate it has received all packets up to and including packet 4, even though, strictly speaking, that would also be true.

This is slightly different from the acknowledgement number field in TCP and the base station should take care in converting the values.
The SACK blocks resemble their TCP counterparts even less. This is because the TCP variants are unnecessarily large. Their size is ten bytes for each SACK block. Two full sequence numbers of four bytes each that indicate the beginning and ending of the block and a two byte option field. E^{2}TCP does things differently. Because the SACK block will always fall within the maximum possible window size (because no more has been transmitted) the difference in sequence numbers between the acknowledgement numbers and the beginning and ending of the SACK blocks is always representable by a 16 bit number. So E^{2}TCP only requires two two byte numbers for each SACK block.
A SACK block is constructed in the following way:

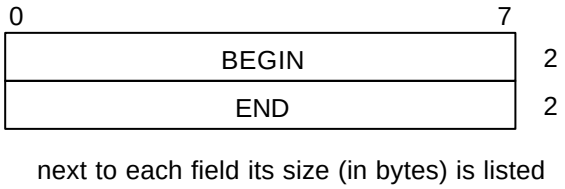

Figure 4.9: The SACK block field format.

The first number indicates the starting position of the SACK block. It is measured as the difference between the sequence number of the first postion of the SACK block and the second sequence number after the previously highest acknowledged sequence number in this packet (either by the acknowledgement number or the previous SACK block). The second number indicates the ending position of the SACK block and is measured as the difference between the sequence number of the end and the sequence number of the beginning of the block. The following example shows this:

Example 4.3: Consider the following receiver state.

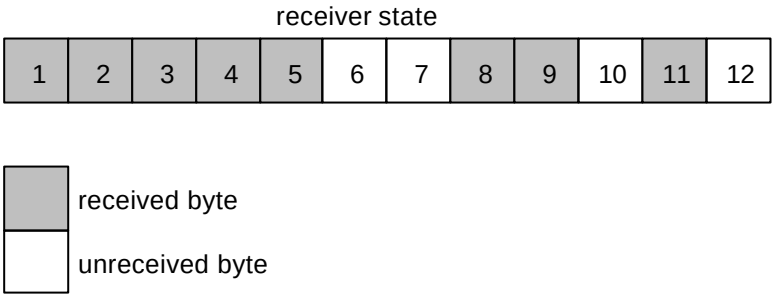

Figure 4.10: An example of a receiver state.

As has been shown, the acknowledgement number would be 5. The first SACK block should acknowledge bytes 8 and 9. The begin field of the first SACK block would be: 8 – 5 – 2 = 1 and the end field would be: 9 – 8 = 1. The second SACK block should acknowledge byte 11. The begin field of the second SACK block would then be: 11 – 9 – 2 = 0 and the end field would be: 11 – 11 = 0.

4.4 Window management

E^2TCP features a window management scheme that is optimized for energy efficiency on wireless single-hop links. First the congestion and flow control mechanisms will be explained, followed by how E^2TCP transmits and retransmits packets. After that will be shown how acknowledgements influence the window size and finally the round trip time estimation will be discussed.

4.4.1 Congestion and flow control

As told in the chapter on TCP, congestion can occur on the intermediate hosts (simply called congestion problems) and at the endpoints of the connection (called flow problems). Because E^2TCP operates on a single-hop link there is no real distinction between congestion and flow control.

Flow control is provided by the window field in the E^2TCP header. If the receiver includes this field in one of its acknowledgements the maximum window size of the sender will be set to the included value. The maximum size will remain so until the receiver specifies otherwise. This will help reduce data overhead because only when changes occur, the new value will be sent. When a connection is setup the maximum window size will be set to its default value.

4.4.2 Transmission

E^2TCP 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. When not all packets are acknowledged before the timer expires, all sent unacknowledged non-retransmitted data within the current window will be transmitted again. All sent unacknowledged non-retransmitted data out of the current window is marked to be transmitted again as soon as the window allows for it. After a transmission timeout the window size will be set to a fixed small value. Transmitted packets will be transmitted again after a transmission timeout or will be retransmitted after packet loss (there is a subtle difference). If all transmitted packets are acknowledged and the transmission timer is still active, it is canceled.

4.4.3 Retransmission

When E^2TCP detects packet loss it will immediately retransmit those lost packets. As said in Paragraph 4.1.3, E^2TCP also features a retransmission timer. With each retransmitted packet the retransmission timer will be set in the same way the transmission timer is set. This is different from TCP because normal TCP implementations have only one transmission timer. By adding one timer so regular transmissions and retransmissions each have their own timer, the retransmission scheme can be made more energy efficient. This is because with a total of two timers, the time it takes before one of them expires is bound to be lower than with only one timer. Thus burst errors will be noticed sooner and E^2TCP will be more responsive to variations in the quality of the channel, which reduces time overhead. The following example will show this.

> Example 4.4: Consider the following situation. Both a TCP and a E^2TCP 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 packet at time 5. So no sooner than time 11, it is able to detect both packet losses.
> An E^2TCP 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.

When not all retransmitted packets are acknowledged before the retransmission timer expires, all unacknowledged already retransmitted packets within the current window will be retransmitted again. All unacknowledged already retransmitted packet out of the current window are marked for future retransmission and will be sent as soon as the window allows for it. After a retransmission timeout the window size will be set to a small fixed value. If all retransmitted packets are acknowledged and the retransmission timer is still active, it is canceled.

4.4.4 Acknowledgements and window size

TCP always considers packet loss to be the result of congestion. This is one of the reasons TCP is not energy efficient on wireless links, as was shown in Paragraph 3.2.3. E^2TCP also considers single and burst errors on the wireless channel to be the cause of lost 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. If the amount of newly reported errors is still below a certain error limit, E²TCP considers the packet loss to be the result of normal static on the channel and will decrease the window size but not below the minimum window size. It is also possible the amount of errors exceeds the limit. E²TCP considers this to be the result of a burst error and the window size is set to its minimum value. In this way E²TCP discriminates between single errors and burst errors and is able to achieve a higher energy efficiency.

4.4.5 Round trip time estimation

Because E²TCP operates on a single-hop link the delay will not vary much, even though it's a wireless link. E²TCP can therefore refrain from using timestamps in its headers, which normally increase data overhead. Round trip time estimations are only done on transmitted packets. No more than one measurement can take place at the same time. Upon packet transmission, a new round trip time measurement is started if possible. If the round trip time measurement is not finished before the transmission timer expires, the measurement is canceled. If the sender receives the acknowledgement that was triggered by the packet that started the measurement, the round trip time is recorded. If an acknowledgement arrives that acknowledges data with higher sequence numbers than the packet that started the measurement, the measurement is canceled. This way only accurate measurements are recorded. E²TCP remembers the last five measurements and uses them to calculate estimations on the current round trip time, which are used to set the (re)transmission timers.

4.4.6 Burst error detection

Although E²TCP has an improved window management scheme to deal with (burst) errors more efficiently than standard TCP, another more sophisticated mechanism was originally intended to be used. Unfortunately it did not perform very well and was abandoned for a cleaner and simpler version that did perform as intended, as was described in Paragraph 4.4.

On a channel with burst errors it is very important that the transport protocol reacts in the right way to burst errors. When the burst error encountered is very small it is best to keep sending at the original pace. This is because the protocol has no time to react. Once it has noticed the burst error, it has already passed. 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. This also means that the length of the burst error is not known a priori. The second problem is that when the protocol stops sending in case of a long burst error, it has no way of telling the burst error is over. So it always has to keep sending some packets. Something that can be thought of as polling.

A scheme was developed that would be able to guess the length of the next burst error, based on the measured lengths of the last burst errors. This scheme kept track of the state of the channel and defined the states as: normal, possible burst error and burst error. When it was in the normal state, it would operate very much like the scheme that is now used. When it suddenly detected a timeout or a lot of errors it would switch to burst error mode. It would set the window size to a very small fixed value and would guess the time the burst error started. Once out of the burst error it would guess the time the burst error ended and would remember the calculated length of the burst error. If the protocol was in the normal state and would only notice a few errors it would guess the time of the start of the errors and switch to the possible burst error state. When the errors would continue it would then switch to the burst error state

and continue as stated above. When the errors would stop however, it would conclude it was no burst error after all and switch back to the normal state.
If enough burst error lengths were recorded the mechanism added an action. Upon noticing the start of a burst error it would guess its length. If it was below a certain limit it would still decrease its window size but only slightly. This way the protocol could still send at almost full speed. If the burst error was indeed as small as predicted it would react in an optimal way. If the burst error was longer than predicted however, it is possible the protocol would react in a very inefficient way. If the predicted length was above the limit it would set the window size to a very small value and set a timer to the predicted end of the burst error. When the timer expired, it would start sending again at near full speed.
As told, this mechanism did not perform very well. It was not stable enough because its measurements were unfortunately very inaccurate. There is no reliable way to accurately measure the start and end times of a burst error for instance. This caused the recorded burst error lengths to be quite inaccurate. When the mechanism then tried to guess the length of the next burst error, it would be based on the inaccurate information. Therefore it would not be very reliable itself. Furthermore it can be argued if the length of the next burst error corresponds in any way with the lengths of the previous burst errors. It was clear that in order to make the mechanism more robust the length measurements could not be used. This caused the scheme to become quite simple but a lot more energy efficient. The performance of this mechanism will be compared with the current mechanism in the following graphs.

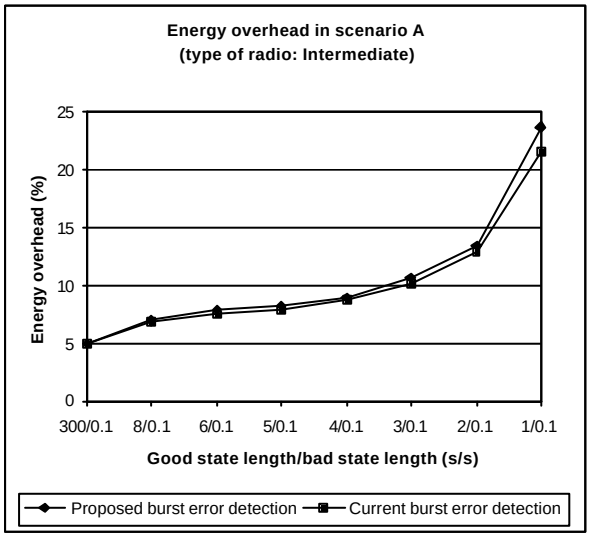

Figure 4.11: Energy overhead of burst error detection mechanisms in scenario A.

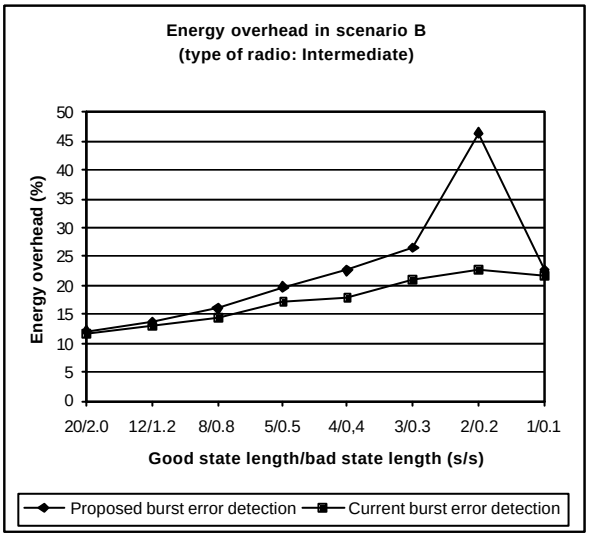

Figure 4.12: Energy overhead of burst error detection mechanisms in scenario B.

In scenario A, the difference between the two burst error detection mechanisms is quite small. Still it should be clear that the current (simple) mechanism has a slightly lower energy overhead in all situations. The superiority of the current scheme becomes especially clear when the graph of scenario B is examined. Clearly the current scheme is more energy efficient than the other.
It is also interesting to note the decrease in energy overhead on the right side of the graph in Figure 4.12. Both mechanisms experience this decrease allthough the decrease in energy overhead of the proposed mechanism is much more pronounced. The decrease is the result of the window management of E^2TCP. When the lengths of bad states drop below a certain

point, E^{2}TCP correctly decides to keep transmitting. This is an efficient solution because the bad state is so short, it will be over before E^{2}TCP has decreased its transmission speed. The graphs show that not decreasing the transmission speed under these conditions is indeed an energy efficient solution.

4.5 Partial reliability

Partial reliability is only relevant to the receiver. An application will be able to set a certain amount of reliability for each connection with a Quality of Service-like parameter. This parameter: the reliability level, can be set from 0% to 100% in one percent steps. Of course E^{2}TCP defaults to full reliability when an application does not set a new reliability (because it is unaware of the partial reliability option for example). When the receiver encounters lost packets, it checks its current reliability level. If it is still above the specified limit, the receiver will falsely acknowledge as much lost packets as possible without violating the reliability demand, so the sender will not retransmit them. After that the receiver will of course update its current reliability level. If the reliability demand is not met, the receiver will send a normal acknowledgement; so all lost data will be retransmitted.
Thus it is possible the application at the sender only receives parts of the stream. The application itself is responsible for handling the gaps in the stream.

5 TEST RESULTS

In this chapter a thorough performance evaluation of E^{2}TCP will be given. To measure the performance and energy efficiency of E^{2}TCP and compare the protocol with other versions of TCP, an implementation of E^{2}TCP was made in the Network Simulator 2 (NS2) [FAL00]. NS2 is an open source discrete event simulator targeted at network research and has substantial support for TCP over wired and wireless links. Because NS2 is free and features implementations of all kinds of simulated applications, versions of TCP, MAC layers, link layers and interconnects like duplex point-to-point links, LANs, wireless LANs, etc, it has become a very popular tool in network research to evaluate (new) protocols.
In this chapter an explanation on what is involved in the tests comes first, followed by an explanation of the home built error model used in the tests. The choice of default values for the parameters, briefly mentioned in the previous chapter, will be discussed after that, followed by a look at how much each method, adopted to make E^{2}TCP energy efficient, makes a difference. Finally the energy efficiency of E^{2}TCP will be compared to that of other TCP variants.

5.1 Simulation model

NS2 is a simulator and not a real environment. Therefore the model of the protocol has been simplified. Sometimes because of limitations in NS2 and sometimes because a part of the protocol was not required to measure the energy performance of the protocol. Differences between the specification and the implementation in NS2 are listed below. All other mechanisms are implemented according to the specification.

- There is no flow control. This is a limitation of NS2 and flow control is therefore also not used in other protocols in NS2. This is not a problem because E^{2}TCP will be tested on a single-hop wireless link to measure its energy efficiency. The absence of flow control does not hamper or improve its basic performance.
- The stream is not byte oriented but packet oriented. This means that all sequence numbers are measured in packets instead of bytes. Packets will have a fixed sized payload of 1000 bytes. Because E^{2}TCP will be tested on a single-hop link this is not a problem at all. This also means the size of the windows is measured in packets instead of bytes.
- There is no connection setup and termination phase. An E^{2}TCP state machine, as TCP has, is not implemented. This is done because it has very little influence on the overall energy efficiency and the TCP variants in NS2 also lack this part of the protocol.
- Sequence numbers can not overflow. No mechanism is in place to let the sequence numbers wrap around when its maximum value (2^{32}) has been reached. Because the implementation also lacks a connection startup phase, the sequence numbers will always start at 0. Since sequence numbers apply to packets instead of bytes this means almost 4 Terabytes can be sent in the simulation before a problem will arise. This is more than enough to measure the performance of E^{2}TCP.

5.2 Test setup

The test setup consists of two hosts connected by a wireless LAN. Because they are the only hosts on the LAN, it can also be seen as a full duplex point-to-point link. Concern may arise that this setup is not representative for wireless LANs with more hosts, but because modern MAC protocols use collision avoidance, the performance of such networks will strongly

resemble a LAN with two hosts and a lower bandwidth. It is not important which host will model the mobile host and which the base station because E^2TCP is a symmetrical protocol. Each host will be running E^2TCP and together they will create one E^2TCP connection that connects both hosts. The sender will start the transmission and during the test, all kinds of data will be collected so the energy efficiency can be calculated. Each test will be run 10 times, of which the average will be used.

There are a lot of parameters to each test, which will influence the outcome. The bandwidth and delay of the channel, the length of the burst errors and the periods between them, the version of TCP used and the kind of traffic.

The bandwidth and delay parameters apply to the wireless LAN itself. This includes the physical medium, the MAC layer and the link layer. E^2TCP should be evaluated with various values for both characteristics but a 'default' value should be chosen for tests in which these characteristics are not the main concern. The default bandwidth will be 1 Mbps. This resembles IEEE 802.11 [IEE99] and Bluetooth [BLU01]. A closer look to the effect of bandwidth will be taken by varying the bandwidth from 0.5 to 5 Mbps, resembling anything from lower speed serial links to the new high speed IEEE 802.11b standard [IEE99b]. For the delay the default value will be 50 ms. This 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 [CHE94]. The effects of the delay on the energy efficiency of various protocols will also be examined by varying the delay between 40 and 70 ms.

An error model is attached to the wireless LAN model in NS2. Such an error model can cause packets to be dropped because of random noise or burst errors. Because the hookup that is supposed to connect one of the standard error models to the wireless LAN model was broken in the version of NS2 that was used, a custom error model was written. This model will be explained in the next paragraph.

The protocols that will be compared to each other are three standard versions of TCP: Tahoe, Reno and NewReno, PRTP in the partial reliability tests, and of course E^2TCP. For PRTP a NS2 implementation was kindly supplied by the PRTP team from the Karlstad University of Sweden. Tahoe, Reno and NewReno were chosen because these are widely known versions of TCP and they are already implemented in NS2.

Various forms of traffic will be simulated to model different types of applications. A (mass) data transfer will be used as the default application. This resembles file transfer, browsing the Internet and sending and receiving emails. In the tests where a closer look will be taken at the effect of the type of traffic, an interactive traffic model will be used as well as a constant bit rate model. 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 (when the requested pages or emails are relatively small). The constant bit rate traffic resembles streaming media, like audio and video.

The (default) test setup then looks like this:

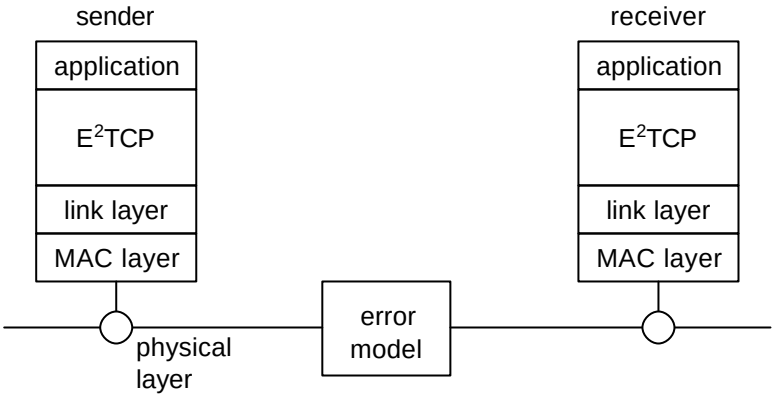

Figure 5.1: The default test setup.

5.3 Error model and setup

As said, a custom error model was created to be used in the tests. A simple two-state error model was chosen because with two states it is already possible to realistically model random noise and burst errors. Each state has three parameters: its minimum length, its maximum length and its error rate. When the minimum length does not equal the maximum length, a random length is chosen between the extreme values at each switch to that state. The more the value approaches one of the extremes the less likely it will be chosen. This way, the chosen value will be near the center of the range most of the time and will sometimes be a lot smaller or larger. The error model will then switch between these two states constantly. The error rate of the state applies to packets because the implementations of the various protocols in NS2 are packet oriented as well. All packets on the wireless LAN are transparently routed through the error model, which randomly corrupts the packets with a chance that corresponds to the error rate of the state it is currently in. This way the corrupted packet will still travel the physical medium and use bandwidth but will be dropped by the MAC layer, just like in real life.
Typically one state will be setup in such a way it resembles a high quality channel with some modest random noise and the other state will represent a burst error with a very high error rate. For the good state an error rate of 0.05% was chosen which corresponds to measurements done by [ECK96]. 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 tests and the lengths of the states were varied to model different channel conditions. These two states and the transitions between them are shown in Figure 5.2.

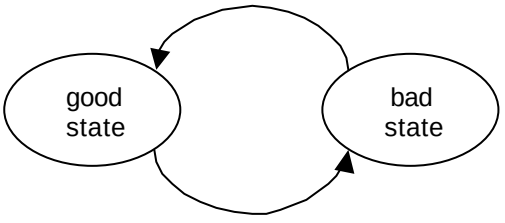

Figure 5.2: States and transitions of the error model.

The choice of state lengths is somewhat more difficult. It is not sufficient to examine the proportions of the good state and the bad state lengths, to see how well a protocol will perform. The length of the bad state itself can have a large impact on the energy efficiency. A

protocol can behave quite differently when the good state and bad state lengths are changed from 20 seconds and 2 seconds to 10 seconds and 1 second respectively, even though the proportions remain the same. So not one but two scenarios were chosen. 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 tests were constructed to be finished 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 proportions are fixed so the channel's quality remains the same. 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.

5.4 E^2TCP parameters

In the previous chapters a few parameters of E^2TCP were mentioned. These include the minimum window size, the maximum window size, the window size after a timeout and the error limit. In this paragraph will be explained what kind of effects each parameter has and how the default values were chosen.

5.4.1 Minimum window size

An E^2TCP sender initializes its window size to the minimum window size and unless a timeout occurs it will not set its window size below this value. The window size is an important parameter because it has a large effect on the energy efficiency of the protocol. When the minimum window size increases the time overhead will diminish. This is because the sender's minimum speed will be higher. An increased minimum window size also means that in case of (long) burst errors the data overhead will increase too, because the sender's transmission rate during burst errors will be quite high. Choosing a good value for this parameter is partly a tradeoff between a decreased time overhead and an increased data overhead. In the following graphs, the performance of minimum window sizes of 5, 10, 15 and 20 packets are given.

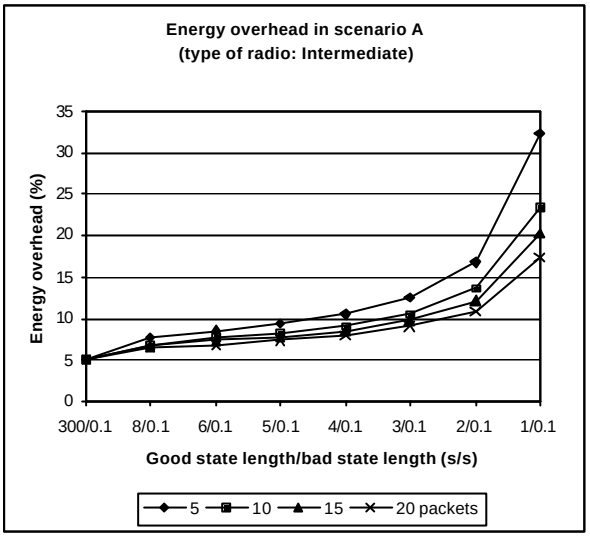

Figure 5.3: Energy overhead of E²TCP with various minimum window sizes in scenario A.

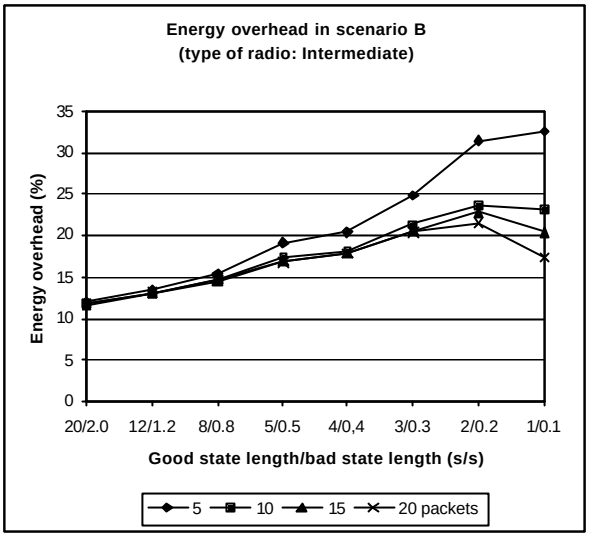

Figure 5.4: Energy overhead of E²TCP with various minimum window sizes in scenario B.

As can be seen from the graphs the higher the minimum window size is set, the lower the energy overhead becomes. One might assume that choosing the highest value possible would be best. The situation is a little bit more complicated however. First of all, the higher the minimum window size the higher the data overhead in certain situations (high bad state lengths in scenario B for example). This causes the energy overhead (with an ideal type of radio) of a minimum window size of 20 packets to be higher than the others in those situations. So a high minimum window size is not always the best solution. The mentioned data overhead and energy overhead figures are shown in the following blown up graphs.

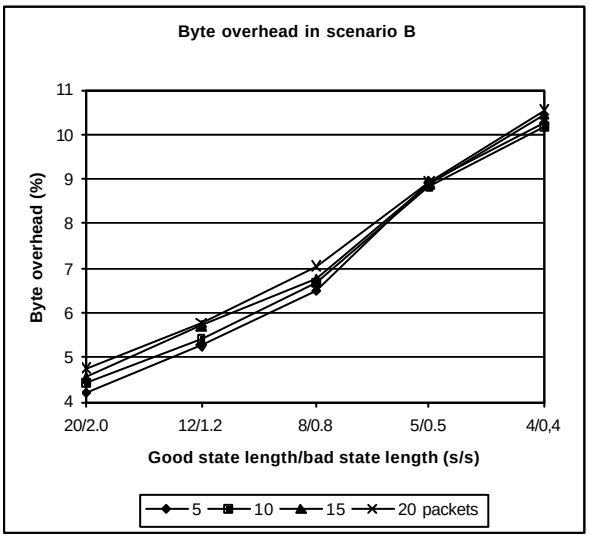

Figure 5.5: Byte overhead of E²TCP with various minimum window sizes in scenario A.

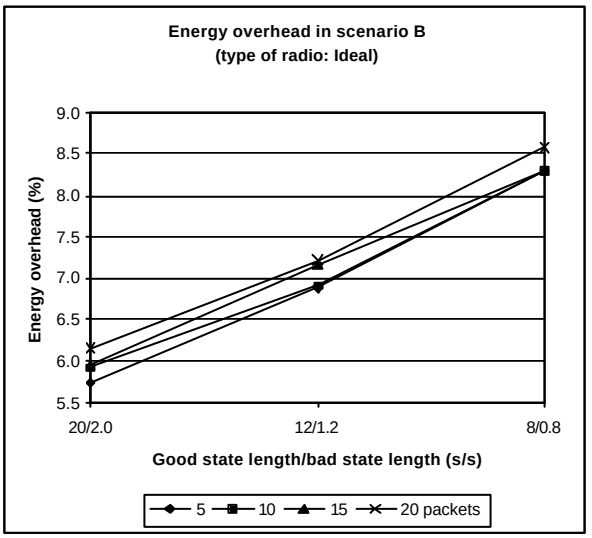

Figure 5.6: Energy overhead of E²TCP with various minimum window sizes in scenario B.

Furthermore, it is important that E²TCP remains adaptive. The higher the minimum window size, the smaller the difference will be between the minimum and maximum window size.

This reduces the adaptivity of E^{2}TCP and makes it less suitable for a wide variation of situations. For these two reasons, a default value of 12 packets was chosen.

5.4.2 Maximum window size

The maximum window size also has quite a large impact on energy efficiency. An E^{2}TCP sender will never set its window size higher than the maximum window size. The higher this value, the greater the bandwidth the protocol can fully utilize. So large values decrease the time overhead, especially on high bandwidth links. Unfortunately very large values can hamper performance on low bandwidth links. A large maximum window size also causes high data overhead in case of (long) burst errors because more traffic is 'in flight' and it takes longer for the protocol to reach an acceptable window size. The maximum size should not be too close to the minimum window size because the protocol can not adapt itself enough to the various channel situations. Choosing a good default value can only be done by making a tradeoff between performance on low and high speed links and time versus data overhead. In the following graphs, the performance of maximum window sizes of 15, 20, 25, 35 and 45 packets are given.

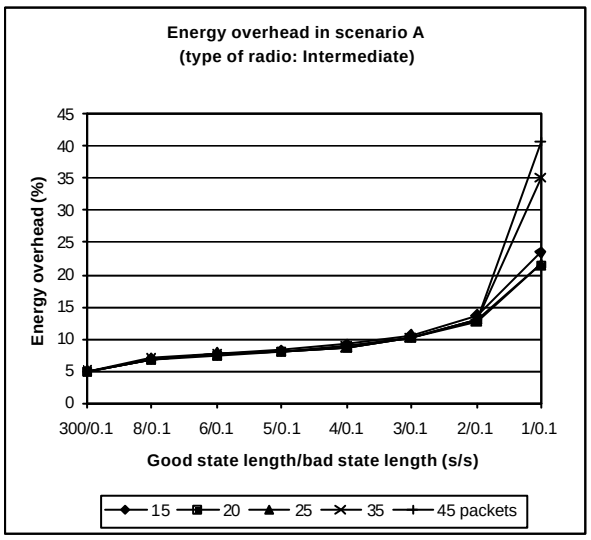

Figure 5.7: Energy overhead of E^{2}TCP with various maximum window sizes in scenario A.

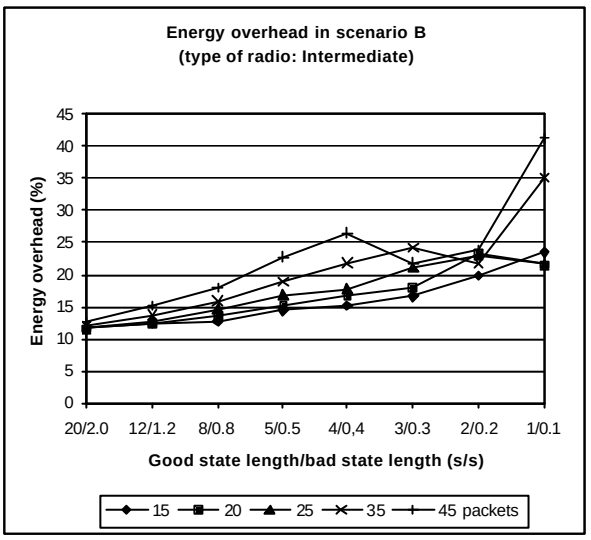

Figure 5.8: Energy overhead of E^{2}TCP with various maximum window sizes in scenario B.

Perhaps it is difficult to see but E^{2}TCP with a maximum window size of 20 or 25 packets scores best in scenario A. A lower or higher value causes the energy overhead to increase. Because of the long bad state lengths in scenario B, the lower the maximum window size the better E^{2}TCP performs (as was predicted). Care should also be taken to make the difference between the minimum and maximum large enough for E^{2}TCP to remain adaptive. This all makes the choice for this parameter quite difficult. A default value of 25 packets was chosen because this value satisfies all requirements best.

5.4.3 Window size after a timeout

The window size is set to this value when a (re)transmission timeout occurs. A large value causes the sender to quickly recover after a burst error but causes extra data overhead during the burst error itself. So again, choosing a default value boils down to making a tradeoff between data and time overhead. As you can see in the graphs however, this parameter does

not have a tremendous effect on the energy overhead. The following graphs show the energy overhead of E^{2}TCP with the following window sizes after a timeout: 1, 2, 5 and 10 packets.

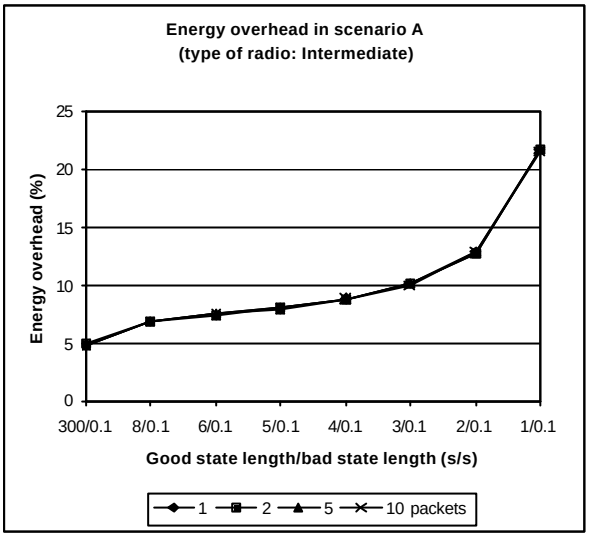

Figure 5.9: Energy overhead of E^{2}TCP with various window sizes after a timeout in scenario A.

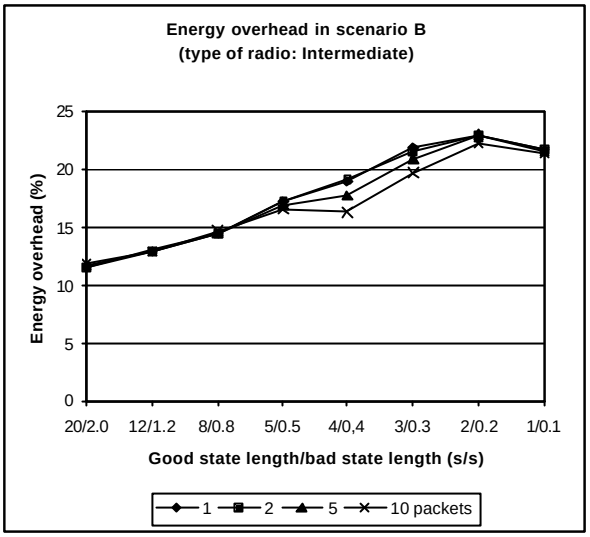

Figure 5.10: Energy overhead of E^{2}TCP with various window sizes after a timeout in scenario B.

The differences are minute but a value of 5 packets has the lowest energy overhead in scenario A. In the other scenario the differences are somewhat larger and a size of 10 packets performs best. The longer the bad state length however, the better the lower values perform. Because of the small differences this parameter does not warrant too much attention. A value of 5 packets seems to be the best overall performer.

5.4.4 Error limit

The error limit parameter decides when an E^{2}TCP sender thinks of the channel as being in a burst error state. So the higher this value the more errors should occur before E^{2}TCP drastically reduces its transmission speed. With a large value E^{2}TCP will have a smaller time overhead in case of (small burst) errors. On the other hand it will have a higher data overhead in case of long burst errors because E^{2}TCP is slower in reacting. And yet again a tradeoff must be made between data- and time overhead before a good default value can be chosen. The energy overhead of E^{2}TCP with error limits of 1, 2, 5 and 10 errors is presented in the following graphs.

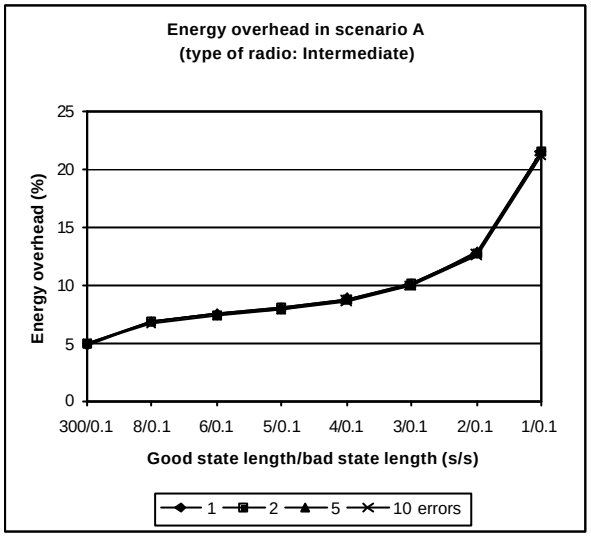

Figure 5.11: Energy overhead of E^2TCP with various error limits in scenario A.

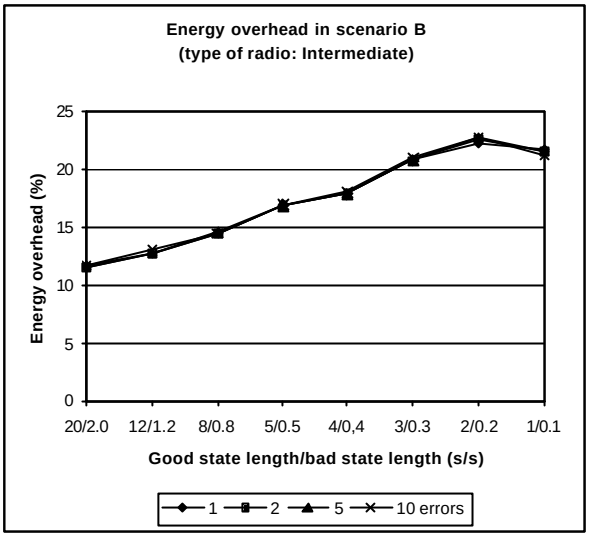

Figure 5.12: Energy overhead of E^2TCP with various error limits in scenario B.

As can be seen the error limit does not have a large impact on the energy overhead of E^2TCP. In scenario A, E^2TCP performs better as the error limit increases. This is because the bad state lengths are so short, decreasing the transmission speed serves no purpose. In scenario B however, the energy overhead of E^2TCP becomes lower as the error limit decreases. The bad states are long enough in this scenario to warrant slowdowns. A tradeoff between these two scenarios yields a default value of 5 errors.

5.4.5 Conclusions

By studying the performance of E^2TCP with different parameters, a set of optimal parameters was chosen as the default values. Most of the times the selection of values for these parameters was quite difficult and often it was necessary to make a tradeoff by increasing a performance metric for a certain situation and decreasing another performance metric (possibly for another situation). The minimum- and maximum window size both have quite a large impact on energy efficiency, especially compared to the window size after a timeout and the error limit, which both hardly influence the energy efficiency of E^2TCP. In *certain* situations, choosing another value for both the minimum- and the maximum window size could yield a maximum decrease in energy overhead of about 25%. Still the chosen default values are considered to be the best *overall* values. To summarize the selection of values, they will be listed in the following table.

Parameter	Default value
Minimum window size	12 packets
Maximum window size	25 packets
Window size after a timeout	5 packets
Error limit	5 errors

Table 5.1: The default values for the parameters of E^2TCP.

5.5 E²TCP dissected

In this paragraph a performance evaluation of the various methods to increase energy efficiency in E²TCP will be given. The methods used are optimized window management, selective acknowledgements, small headers and partial reliability. The performance evaluation will start by comparing Tahoe with E²TCP, which only has optimized window management enabled. The comparison will be done with Tahoe because it is the most energy efficient version of TCP, as will be shown in Paragraph 5.6.1. After that selective acknowledgements will be added to E²TCP, followed by its own headers and finally partial reliability. All tests will be done with the default setup.

5.5.1 Window management

In this test, E²TCP only has its own window management enabled. Unfortunately, the window management scheme of E²TCP relies on selective acknowledgements to operate properly. This version of E²TCP will therefore be severely crippled. The energy overhead of both protocols is shown in the following two graphs.

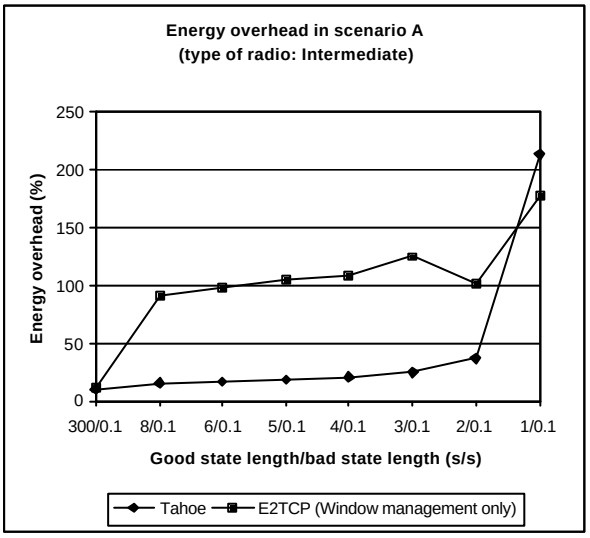

Figure 5.13: Energy overhead of various protocols in scenario A.

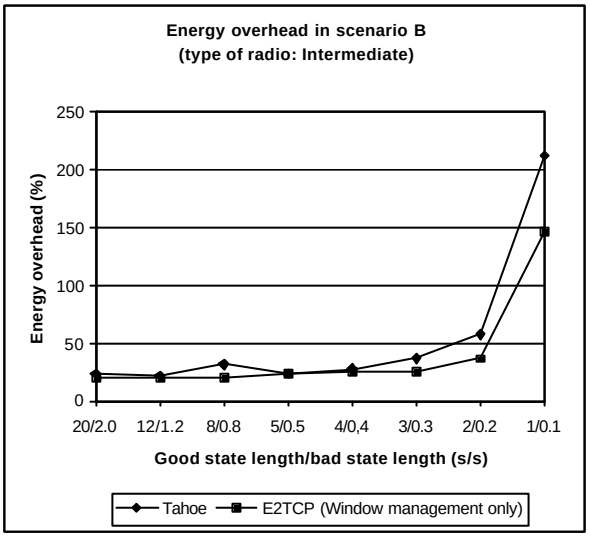

Figure 5.14: Energy overhead of various protocols in scenario B.

In scenario A, it is clear that E²TCP is too crippled to reach low levels of energy overhead. Tahoe clearly scores better. Certainly there is a lot of room for improvement. In scenario B, E²TCP already outperforms Tahoe and is therefore the more energy efficient protocol of the two.

5.5.2 Selective acknowledgements

Selective acknowledgements will also be enabled for E²TCP in this test. This should also make the window management scheme perform better because it directly depends on SACK. The energy overhead for both scenarios is shown in the following graphs.

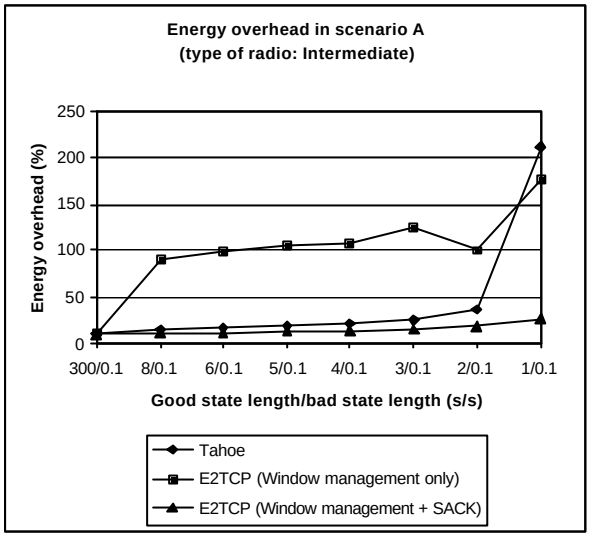

Figure 5.15: Energy overhead of various protocols in scenario A.

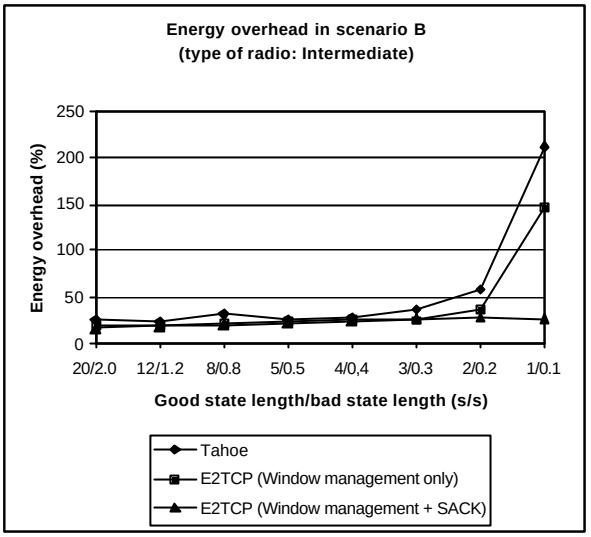

Figure 5.16: Energy overhead of various protocols in scenario B.

As can be seen in the graphs, E^{2}TCP has less energy overhead than Tahoe in scenario A this time. Clearly E^{2}TCP depends on selective acknowledgements in this scenario. In scenario B, the gain is less impressive but especially with small good and bad state lengths E^{2}TCP with selective acknowledgements is more energy efficient than E^{2}TCP with window management only. Clearly selective acknowledgements make E^{2}TCP more energy efficient in both scenarios.

5.5.3 E^{2}TCP headers

The custom headers of E^{2}TCP were also used in this test. Because they are much smaller than standard TCP/IP headers they should also contribute to less energy overhead. Enabling the custom headers gets E^{2}TCP up to full strength. If all versions would be listed in the graphs, they would become quite hard to study. Therefore, only E^{2}TCP with both its window management and selective acknowledgements enabled will be used to compare standard E^{2}TCP with. The energy overhead of both versions of E^{2}TCP will be shown in the following graphs.

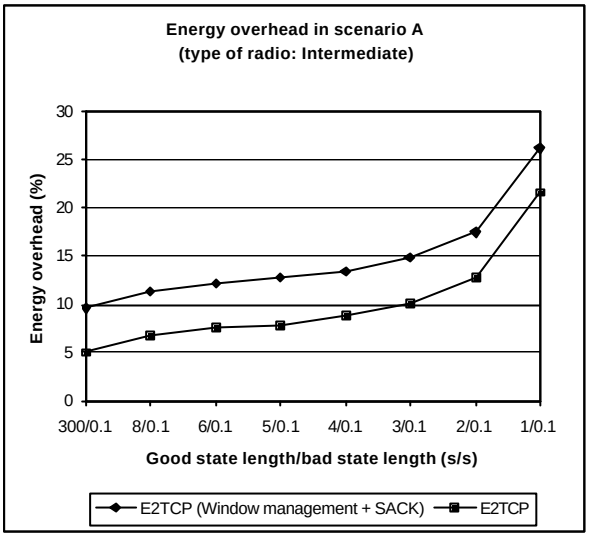

Figure 5.17: Energy overhead of various protocols in scenario A.

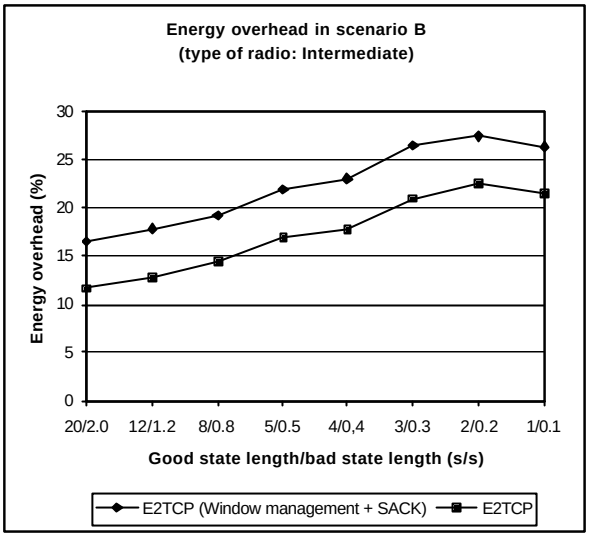

Figure 5.18: Energy overhead of various protocols in scenario B.

It should be clear from the graphs that enabling the custom headers, lowers the energy overhead of E^2TCP in both scenarios in all situations with about 5%.

5.5.4 Partial reliability

In the previous paragraph was shown what the energy overhead of standard E^2TCP was. When an application allows for it, E^2TCP can also enter a partial reliable mode of operation. This will further enhance its energy efficiency. In this test E^2TCP will be used at 100% and 90% reliability. The energy overhead of both reliabilities is shown in the following graphs.

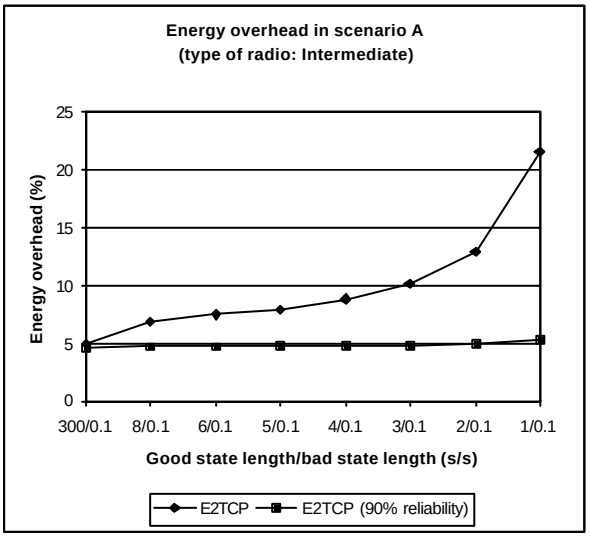

Figure 5.19: Energy overhead of various protocols in scenario A.

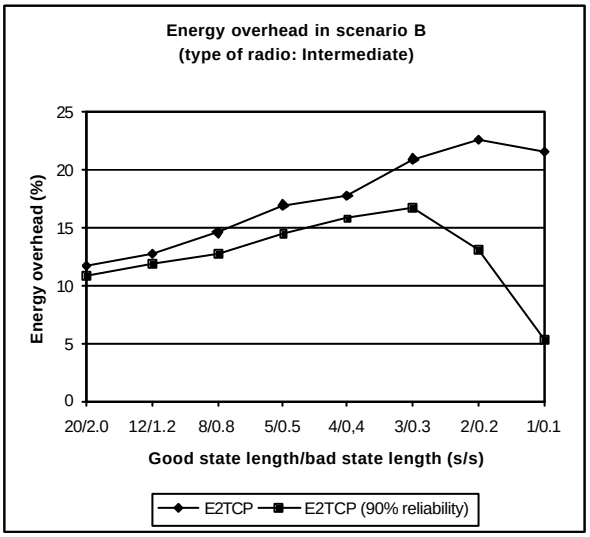

Figure 5.20: Energy overhead of various protocols in scenario B.

In both scenarios partial reliability manages to improve the energy efficiency of E^2TCP considerably. Partial reliability obviously has a larger effect when the quality of the channel

deteriorates. The reason for this is that when more packets are lost, the advantage of not retransmitting them increases.

5.5.5 Conclusions

By 'dissecting' E^{2}TCP, it became clear how much each energy efficient method present in E^{2}TCP, contributed to the overall energy efficiency of E^{2}TCP. Only enabling the window management scheme of E^{2}TCP, clearly crippled E^{2}TCP so much it was unable to perform better than Tahoe in scenario A. 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 just like enabling partial reliability. To give an indication of the energy overhead of all versions, the *average* energy overhead 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 (Window management only)	102.4	39.6
E^{2}TCP (Window management + SACK)	14.7	22.4
E^{2}TCP	10.1	17.4
E^{2}TCP (90% reliability)	4.9	12.6

Table 5.2: The average energy overhead of various protocol versions.

In Paragraph 1.1, three proposals were mentioned to deploy E^{2}TCP. The first proposal only allowed the transport protocol at the base station to be replaced with E^{2}TCP. The second proposal was to replace the transport protocol at both the base station and the mobile host, while the third proposal was to change the applications at the mobile (and possibly the internet) host as well. It was expected that the first proposal would yield the smallest gain in energy efficiency, while the third proposal would yield the largest gain in energy efficiency. Now E^{2}TCP has been 'dissected', it is possible to check if those expectations were correct.
When only the base station is running a version of E^{2}TCP, its possible to use the optimized window management method. There are two problems however. First E^{2}TCP with only its window management enabled can have a higher energy efficiency than Tahoe, but also a considerably higher one. It depends on the situation. The second problem is that only when the base station is the sender, energy can be saved. The first problem is solved when the mobile host is running a SACK enabled version of TCP, but still it is quite useless to consider the first proposal. As said, it is unclear if energy will be saved and if energy is saved it will be at the base station and not at the mobile host, where it is needed.
When the second proposal is executed things become more interesting. Both the base station and the mobile host will be running E^{2}TCP so there is no problem in using both the window management scheme and selective acknowledgements. It has been shown that when both are enabled, E^{2}TCP clearly is more energy efficient than other protocols and the mobile host will definitely save energy. This is because the energy overhead of other protocols is two to three times as high as that of E^{2}TCP. Furthermore, E^{2}TCP can use its own headers further increasing its energy efficiency. Then the energy overhead of other protocol is three to four times as high as that of E^{2}TCP. The energy efficiency (based on the average energy overhead) of Tahoe is 69% and 65% in scenario A and B respectively, while E^{2}TCP scores 91% and 85% respectively.

Partial reliability can be enabled when the third proposal is executed: changing the applications at the mobile- and internet hosts. This causes the energy overhead of E^{2}TCP to drop even further to levels that are anywhere from four to nine times as small as normal TCP. The energy efficiency (based on the average energy overhead) of E^{2}TCP with partial reliability is 95% and 89% in scenario A and B respectively. However, partial reliability can only be used with streaming media.

5.6 Evaluation of E^{2}TCP

The evaluation of E^{2}TCP will begin with a detailed look at the default setup. Next, a closer examination of its performance on wireless links with different bandwidths. After that a closer look will be taken at the influence of different delays, traffic and finally the effect of partial reliability will be studied.

5.6.1 Default setup

In this test all parameters will be set to their defaults. This means that the bandwidth will be 1 Mbps, the delay 50 ms and the protocols compared will be Tahoe, Reno, NewReno and E^{2}TCP. The partial reliability mechanism of E^{2}TCP will be disabled so E^{2}TCP is 100% reliable, just like the other protocols. The simulated traffic will be a mass data transfer of 20 MB in total. Both error scenarios will be used.
This first time a performance evaluation will be given, a closer look will be taken at the data overhead and the time overhead before examining energy overhead. In later evaluations these graphs will be omitted because finally only energy overhead counts. Should a later test yield interesting results in respect to data- or time overhead, the graphs will be included.

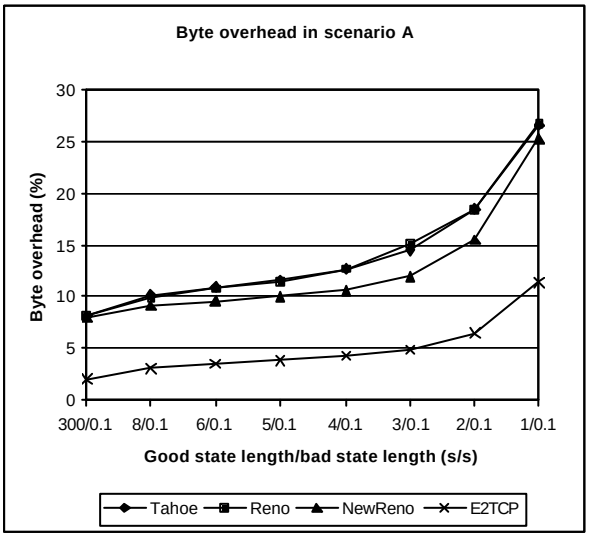

Figure 5.21: Byte overhead of various protocols in scenario A.

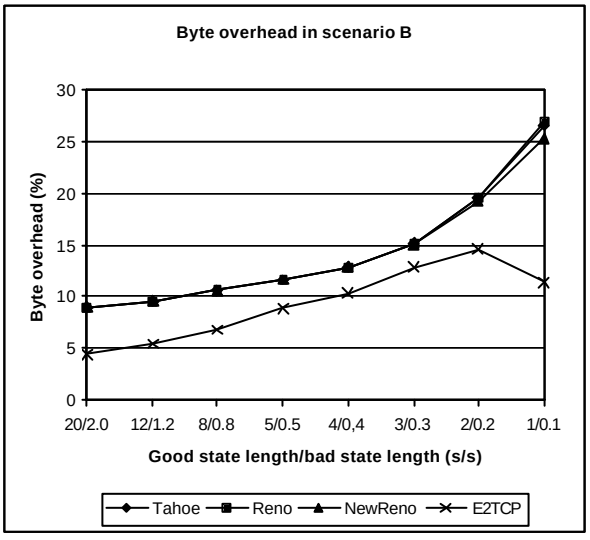

Figure 5.22: Byte overhead of various protocols in scenario B.

As can be seen, E^{2}TCP 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 E^{2}TCP at the right side of

the graph. Unlike standard TCP, E^{2}TCP does not decrease its transmission speed for very small burst errors, resulting in a very low data overhead when burst errors are very small.
Another characteristic of the graphs that should be noted is that all three standard versions of TCP behave the same. The absolute values in data overhead may differ somewhat but the tendency of each version closely resembles that of the others.

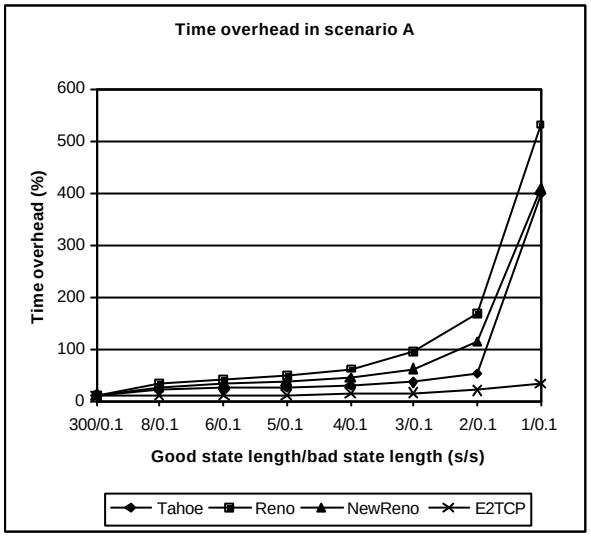

Figure 5.23: Time overhead of various protocols in scenario A.

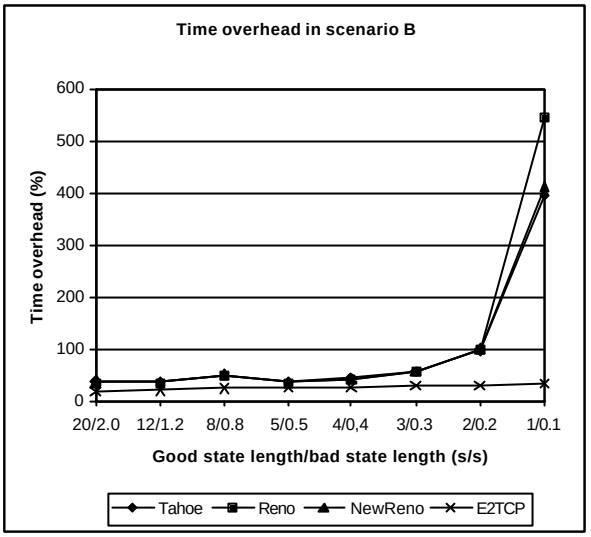

Figure 5.24: Time overhead of various protocols in scenario B.

Because of the enormous differences in time overhead, it is hard to see how much the time overhead of E^{2}TCP differs from that of the other protocols in the left sides of the graphs. When examining the source data for the graphs, it is clear that E^{2}TCP has a time overhead that is about twice as small as the other protocols in the worst cases. 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 worsens.
Just like with data overhead, the three versions of TCP tested behave in the same way. The absolute values differ somewhat again, but they all have the same tendency. It should also be clear that the other versions of TCP have much more time overhead than data overhead. Because of the way energy overhead is calculated, there will probably be large differences between the different types of radios.
What this means for the energy overhead will be shown in the following graphs. For each scenario three graphs will be shown, each graph corresponding to a certain type of radio.

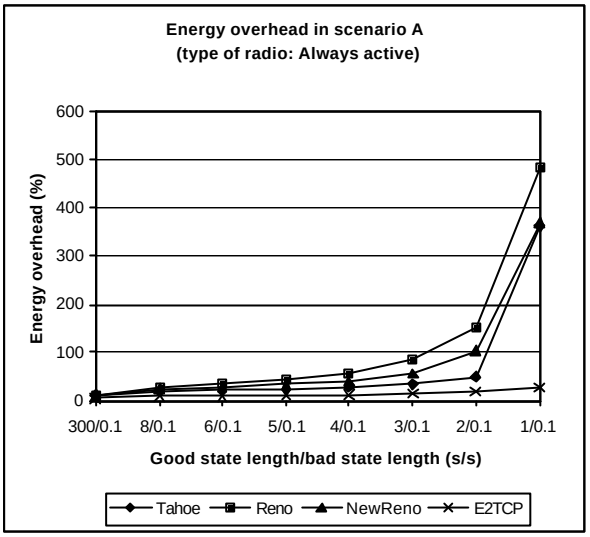

Figure 5.25: Energy overhead of various protocols in scenario A.

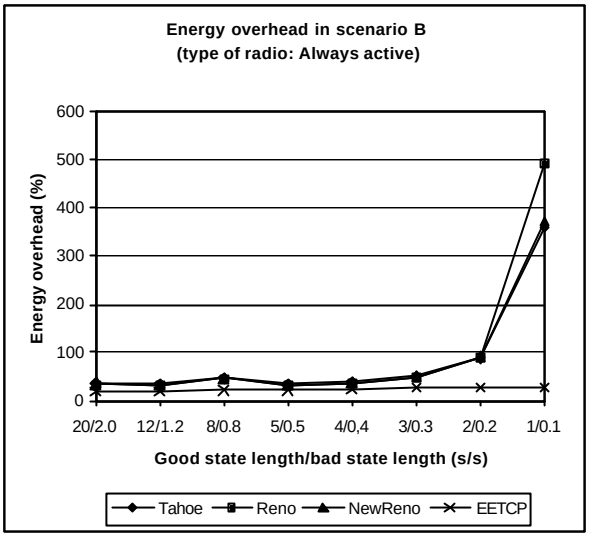

Figure 5.26: Energy overhead of various protocols in scenario B.

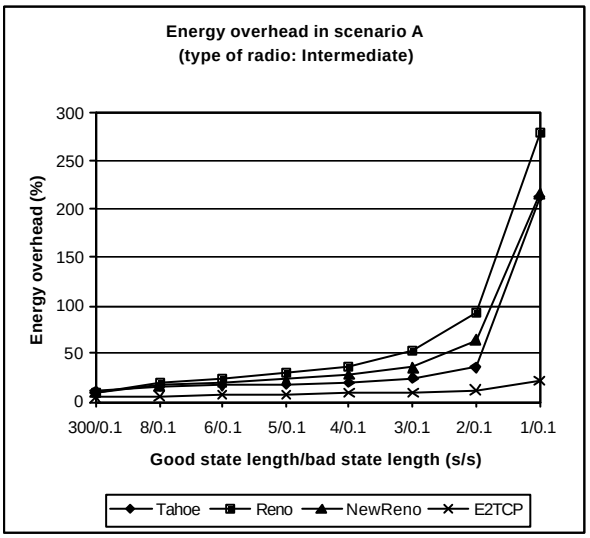

Figure 5.27: Energy overhead of various protocols in scenario A.

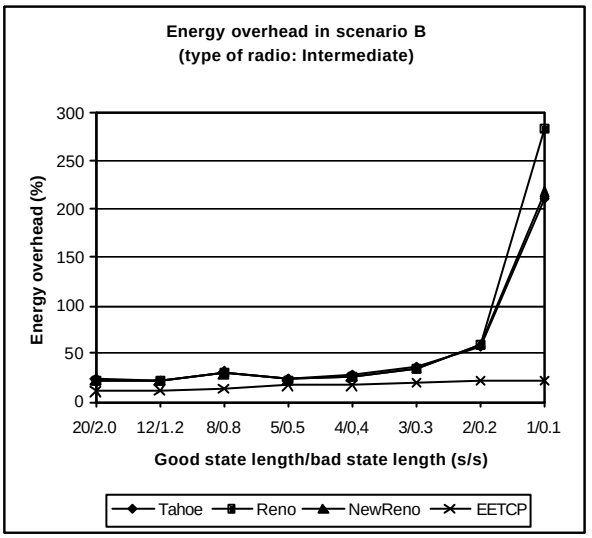

Figure 5.28: Energy overhead of various protocols in scenario B.

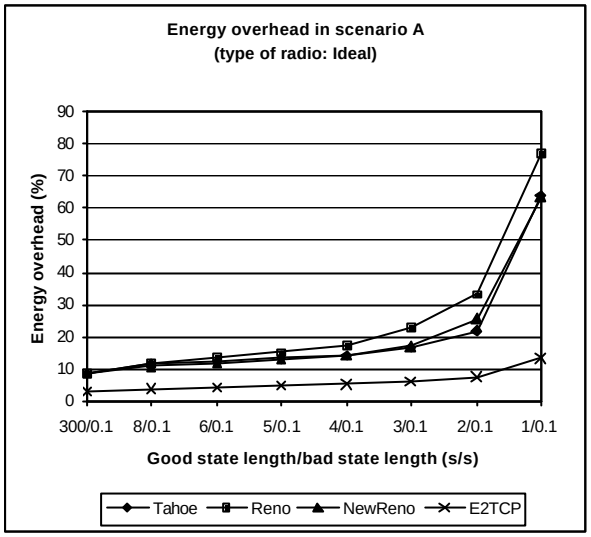

Figure 5.29: Energy overhead of various protocols in scenario A.

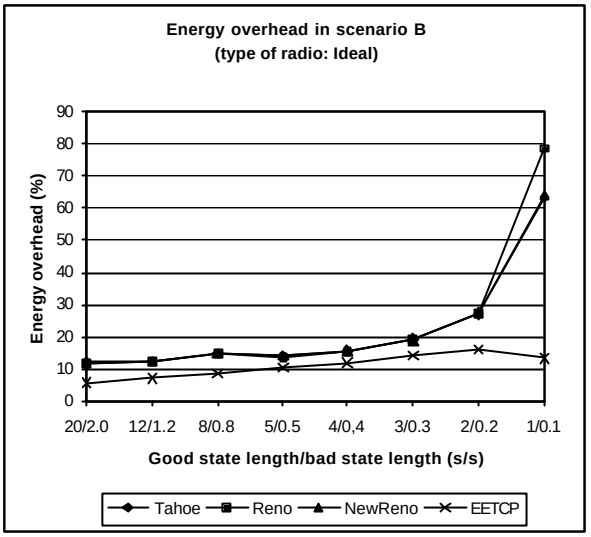

Figure 5.30: Energy overhead of various protocols in scenario B.

The first thing that can be concluded is that E^{2}TCP has a significantly lower energy overhead than the other three protocols, in both scenarios and with all types of radios. Thus E^{2}TCP has a higher energy efficiency than the other protocols in this test. Usually E^{2}TCP has an overhead that is at least twice as small as that of another protocol, but the difference can increase with shorter bad state lengths to 16 times as small.

Another important thing to note is that just as with data- and time overhead, the three versions of TCP behave in the same way. From now on only one other protocol will be used to compare the performance of E^{2}TCP to. If the graphs are studied closely, Tahoe can be said to be the most energy efficient protocol of the three and will therefore be used.

It should also be clear that the energy overhead obtained by the protocols on an ideal type of radio is much lower than that on the other types of radio. This is because for an ideal type of radio time overhead has not a big impact. As seen, (for the standard TCP versions) time overhead is much higher than data overhead, which causes the increased energy overhead for the first two types of radios.

The final remark that will be made, is that although the absolute values for energy overhead differ between the three types of radios, the tendencies of each protocol are the same regardless of what type of radio is used. Therefore, only one type of radio will be used in future tests. The intermediate type will be chosen because the other types are at the extremes of the scale. The intermediate type will therefore probably be better suited to be compared to real radios. To be complete, the energy efficiency graphs for the intermediate type of radio and both scenarios will be listed below.

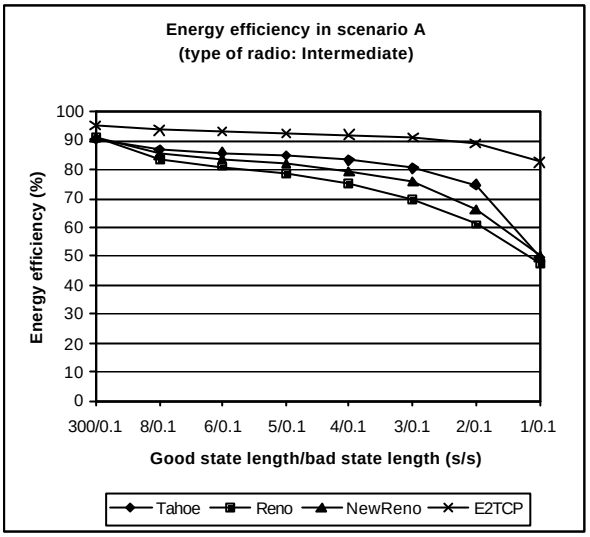

Figure 5.31: Energy efficiency of various protocols in scenario A.

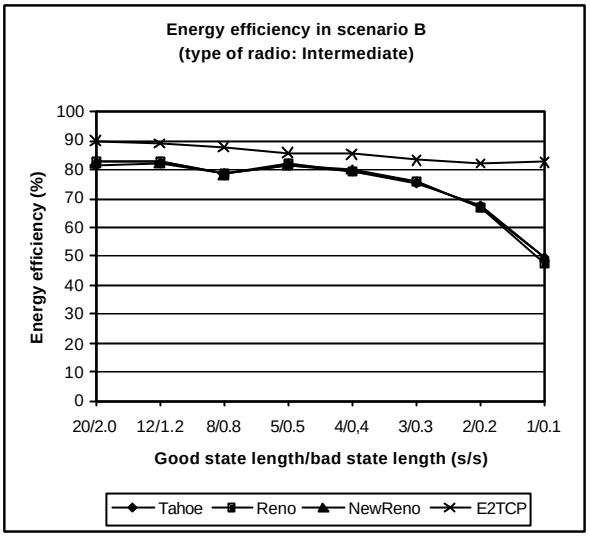

Figure 5.32: Energy efficiency of various protocols in scenario B.

As was already concluded from the energy overhead graphs, it is clear that E^2TCP has a higher energy efficiency than the other protocols.

5.6.2 Bandwidth

In this test the impact of the channel's bandwidth on the energy overhead will be examined. The test setup equals the default setup except for a few changes. The bandwidth will not be fixed at 1 Mbps but four different bandwidths will be used: 0.5, 1, 2 and 5 Mbps. As told in the previous paragraph, only Tahoe and E^2TCP will be used and only the energy overhead graphs of the intermediate type of radio will be shown. The results of the test can be seen in the following graphs:

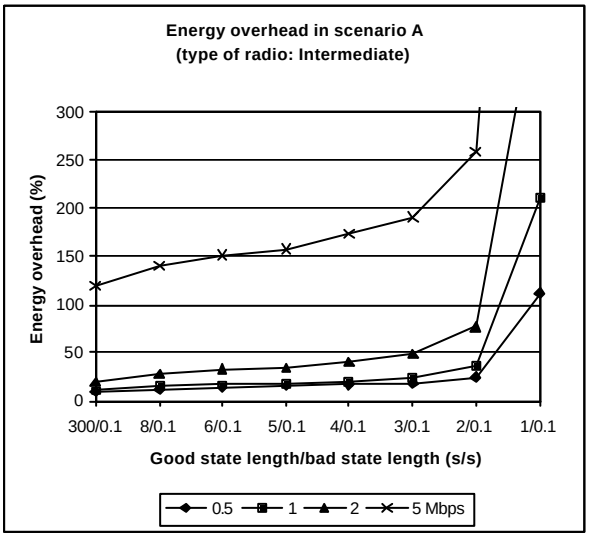

Figure 5.33: Energy overhead of Tahoe with various bandwidths in scenario A.

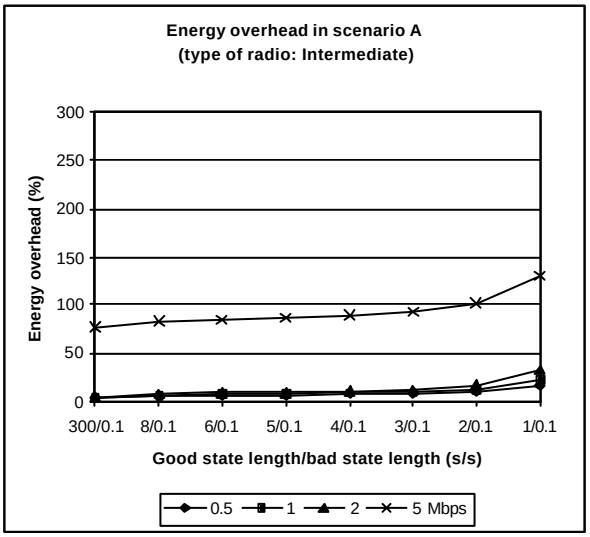

Figure 5.34: Energy overhead of E^2TCP with various bandwidths in scenario A.

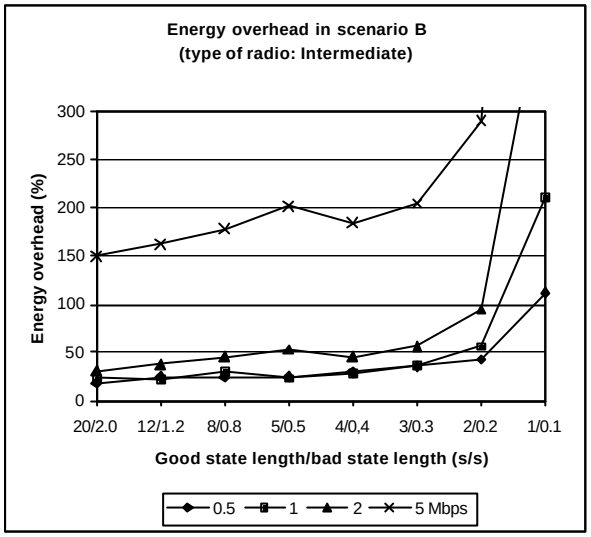

Figure 5.35: Energy overhead of Tahoe with various bandwidths in scenario B.

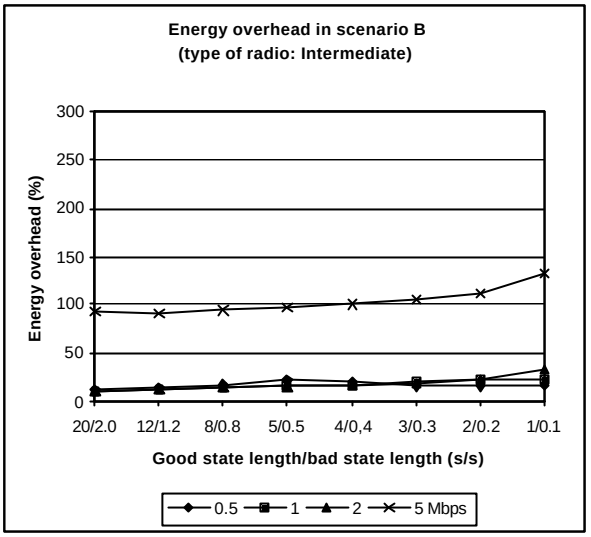

Figure 5.36: Energy overhead of E^2TCP with various bandwidths in scenario B.

Because of the terribly high overhead of Tahoe on high bandwidth low quality links, the graphs were truncated at 300% overhead. On links with the smallest good state length Tahoe had 441% and 1128% energy overhead on 2 and 5 Mbps links respectively for scenario A and 444% and 1125% for scenario B.

Upon close examination of the graphs, it can be seen that E^2TCP has less energy overhead than Tahoe for each bandwidth/quality of channel combination. In most cases Tahoe even has an energy overhead that is at least twice as large as that of E^2TCP. This means that E^2TCP is more energy efficient than Tahoe (in this test). The second conclusion is that independent of bandwidth, E^2TCP scales better than Tahoe when channel conditions deteriorate. It should also be clear that E^2TCP does not scale worse than Tahoe when bandwidth increases. This is an important characteristic of E^2TCP because the bandwidths on new wireless standards are rapidly increasing.

5.6.3 Delay

The impact of the delay of the channel on the energy overhead of Tahoe and E^2TCP was examined in this test. The test setup equals the default setup except for the following changes: the delay was not fixed at 50 ms but the test was run with delays of 40, 50, 60 and 70 ms. The energy overhead for both scenarios is shown in the following graphs.

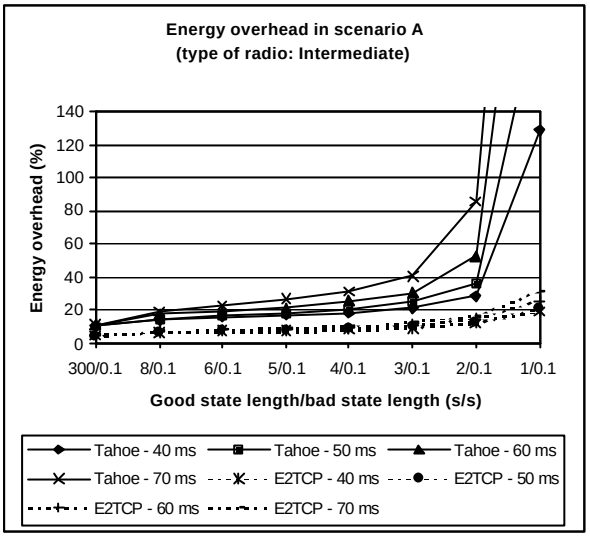

Figure 5.37: Energy overhead of Tahoe and E²TCP with various delays in scenario A.

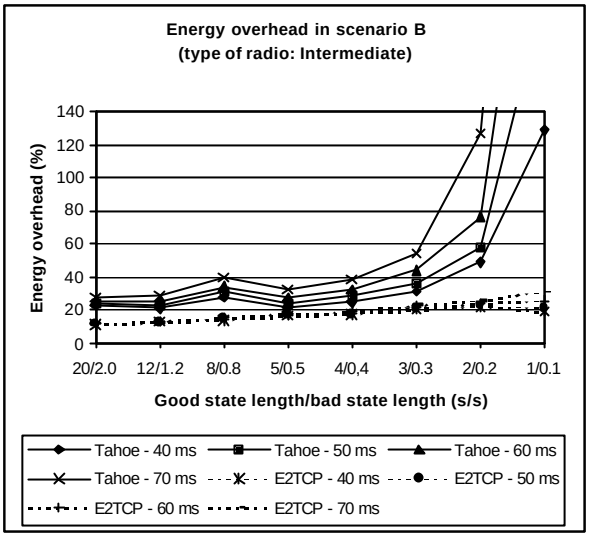

Figure 5.38: Energy overhead of Tahoe and E²TCP with various delays in scenario B.

Again the graphs were truncated because of the large differences in overhead. This time at 140%. On links with the smallest good state length Tahoe had 211%, 351% and 524% energy overhead on links with 50, 60 and 70 ms delay respectively for scenario A and 211%, 340% and 540% for scenario B.

Two conclusions can be drawn by studying the graphs. The first conclusion is that E²TCP again has a lower energy overhead than Tahoe on all delay/channel quality combinations and thus is more energy efficient. The second conclusion is that E²TCP scales much better when the channel delay increases.

5.6.4 Traffic

In this test the energy overhead of Tahoe and E²TCP will be examined for various types of traffic. Up to now, all tests were done with a simulation of a (mass) data transfer. In this test two other types of traffic will be used for the simulation. The first simulation will be done with an interactive application model that models interactive types of traffic like telnet sessions, chatting and instant messages. The second simulation will be done with a constant bit rate application, that models streaming audio and video.

The interactive traffic model works a bit different than the data transfer model. With an 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 test 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). The energy overhead for both scenarios is shown in the following graphs.

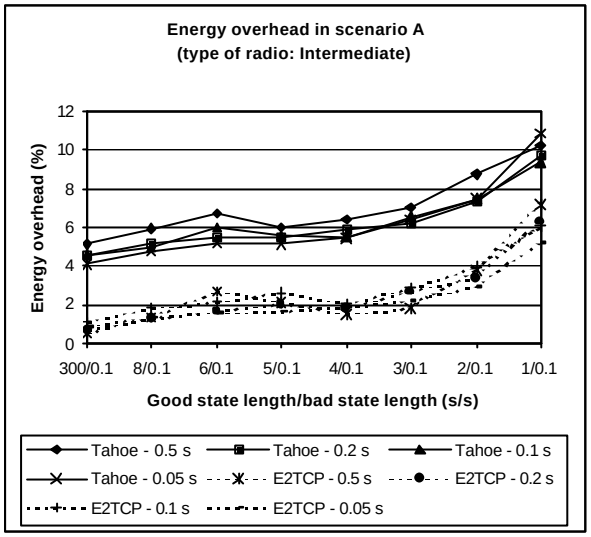

Figure 5.39: Energy overhead of Tahoe and E²TCP with various interdeparture times in scenario A.

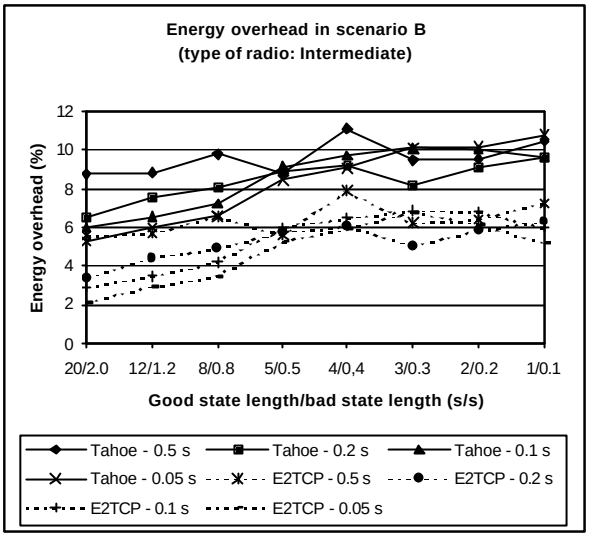

Figure 5.40: Energy overhead of Tahoe and E²TCP with various interdeparture times in scenario B.

As can be seen, 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, the differences are not as large but E²TCP still manages to score lower energy overhead scores.

The constant bit rate traffic model resembles the interactive model somewhat. It too sends data at a specified rate. To model streaming media, data rates of 0.25, 0.5 and 1 Mbps were used. The energy overhead for both scenarios is shown in the following graphs.

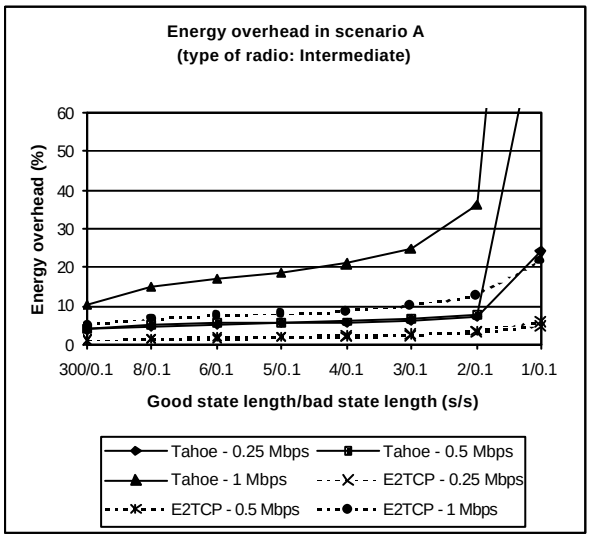

Figure 5.41: Energy overhead of Tahoe and E²TCP with various data rates in scenario A.

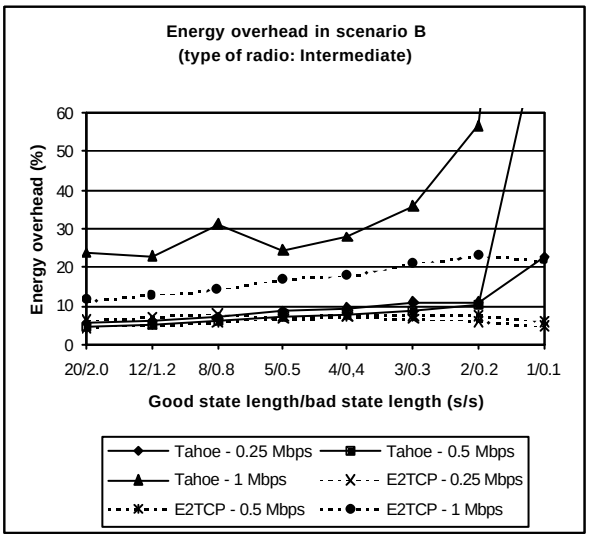

Figure 5.42: Energy overhead of Tahoe and E²TCP with various data rates in scenario B.

Because of the large differences in energy overhead both graphs were truncated at 60%. In scenario A with a good state length of 1 second, Tahoe scores 89% and 211% for data rates of 0.5 and 1 Mbps respectively in scenario A and 87% and 212% respectively in scenario B.

From the graphs can be concluded that in scenario A, E^{2}TCP is much more energy efficient than Tahoe. Furthermore E^{2}TCP scales better when data rates increase. In scenario B, Tahoe is able to equal E^{2}TCP's energy overhead when bad states are long and data rates low. E^{2}TCP however, is more energy efficient when data rates increase and/or bad state lengths shorten. So E^{2}TCP has a higher energy efficiency in this test.
Another conclusion that can be drawn from the graphs is that even though the absolute numbers differ when E^{2}TCP is used with different data rates, the tendencies do not. This means that for different data rates E^{2}TCP behaves the same.
The 1 Mbps constant bit rate traffic is able to completely saturate the link because it has a bandwidth of 1 Mbps itself. Because of this, that traffic behaves exactly the same as a mass data transfer. Closely examining the source data of the graphs, proved this.

5.6.5 Partial reliability

In this test the partial reliability of E^{2}TCP will be examined. The default setup will be used with the following protocols: Tahoe, PRTP and E^{2}TCP. Tahoe is of course fully reliable. E^{2}TCP will be set to 95% and 90% reliability while PRTP will be used at 90% reliability. It was the intention to test PRTP at 95% reliability as well, but PRTP does not allow for such fine-tuning of the reliability. Apparently with E^{2}TCP the application has a more fine-grained control over the reliability of the connection.
It is only useful to use partial reliability on certain types of traffic. Streaming media applications and sometimes mass data transfer (images, audio and video) applications are suited to adapt to partial reliability. Therefore only the constant bit rate and mass data transfer models should be used in this test. In the previous paragraph it was shown that for a data rate of 1 Mbps the constant bit rate application behaves exactly the same as the mass data transfer application. Furthermore it was shown that E^{2}TCP behaves the same when different data rates are used for the constant bit rate application. Therefore it is sufficient to use the mass data transfer (as in the default test setup) for this test. The energy overhead for both scenarios is shown in the following graphs.

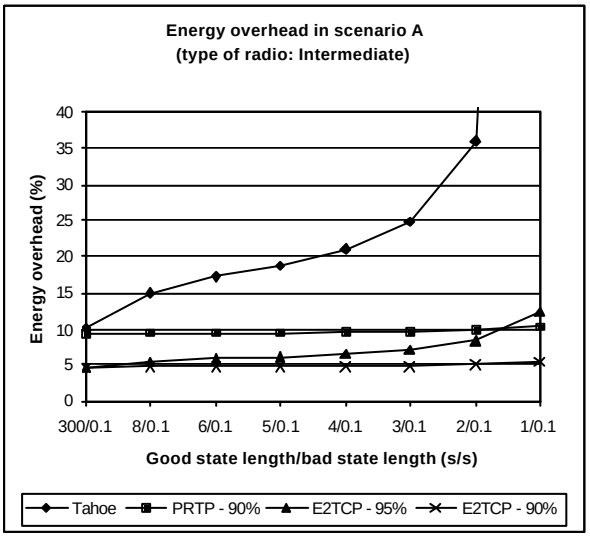

Figure 5.43: Energy overhead of various protocols in scenario A.

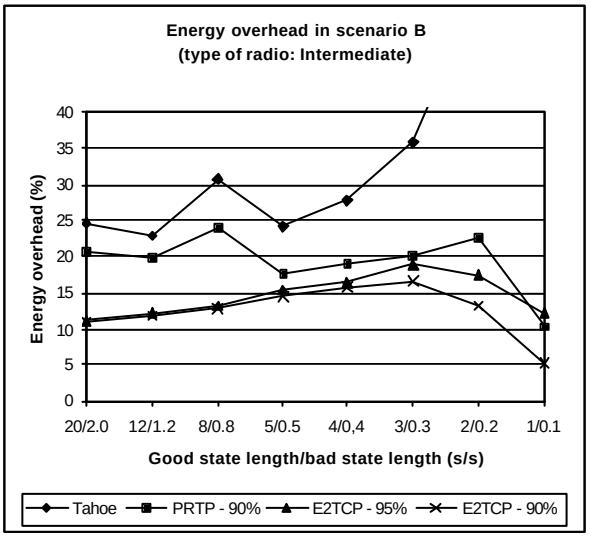

Figure 5.44: Energy overhead of various protocols in scenario B.

The graphs were truncated again because of the large differences in energy overhead. This time at 40%. In scenario A, Tahoe has an energy overhead of 211% when the good state length is 1 second while in scenario B the same protocol scored 57% and 212% with good state lengths of 2 and 1 second respectively.
From the graph of scenario A a few things can be deduced. First of all, PRTP (with a reliability of 90%) clearly has less energy overhead than Tahoe. Another interesting thing to note is that both PRTP and E²TCP at 90% reliability score (almost) the same independent of the quality of the channel. Because of the loose reliability constraints both protocols can deal very efficiently with errors. E²TCP with 95% reliability clearly has more trouble when the quality of the channel worsens because the reliability constraints are tighter. Still it manages to surpass PRTP in all but the worst channel conditions.
In scenario B, the last point is also valid. That is: PRTP is more efficient than Tahoe, while E²TCP with a reliability of 95% surpasses the performance of PRTP in all but the worst channel conditions. Just like in scenario A, E²TCP with a reliability of 90% is the most energy efficient protocol.

5.6.6 Performance

So far, only the energy efficiency of E²TCP has been examined. Because of the goal of this thesis, this 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. The comparison of the various protocols with respect to traditional performance will not be as extensive as the evaluation of the energy efficiency.
Throughput is a measure to indicate the utilization of the link. It is measured in bits per second and can of course never exceed the bandwidth of the link. Throughput can be calculated by dividing the payload of the data transmission with the total time it took to complete the data transmission. The faster a data transmission was finished, the higher the throughput will be. Because the time overhead also decreases when the time to completion decreases (and vice versa), it can be concluded that the lower the time overhead of a protocol is, the higher its throughput will be. Because the time overhead of E²TCP and other versions of TCP have already been examined, a prediction can be made about the throughput of those protocols. It is expected that E²TCP will have a higher throughput than other versions of TCP.
To test the throughput of E²TCP, the default test setup was used. Therefore the throughput can be no higher than 1 Mbps. In the next graphs, the throughput of E²TCP and other versions of TCP will be shown.

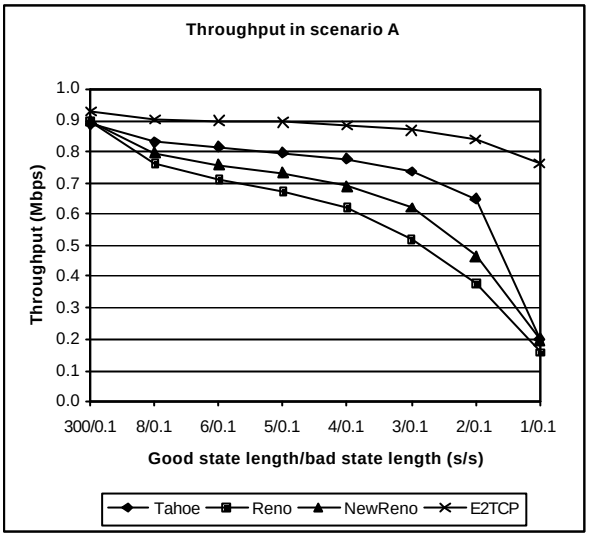

Figure 5.45: Throughput of various protocols in scenario A.

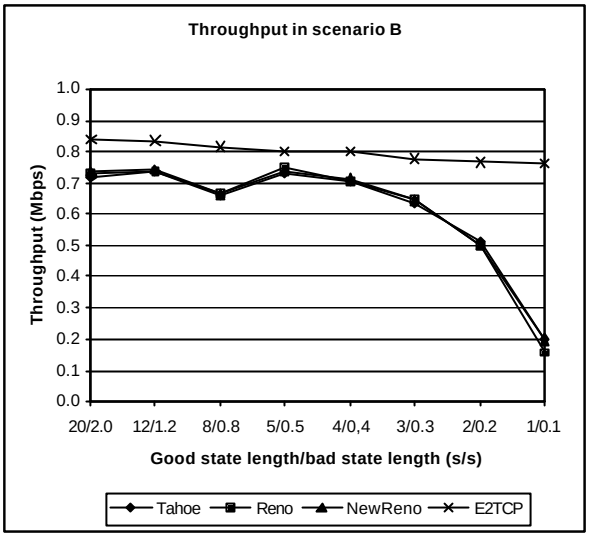

Figure 5.46: Throughput of various protocols in scenario B.

As can be seen in both graphs, the throughput of E^{2}TCP is clearly higher than that of the other versions of TCP. When channel conditions deteriorate, the difference in throughput becomes exceptionally large. Just as in the time overhead graphs in Paragraph 5.6.1; the other versions of TCP behave the same. Their absolute throughput scores may differ somewhat (especially in scenario A), but their graphs all show the same tendency. By optimizing E^{2}TCP for energy efficiency by lowering its time overhead, the throughput was unintentionally increased. Because of the direct relation between time overhead and throughput, it is unnecessary to examine other test setups. The time overhead of E^{2}TCP in all test setups was closely examined to calculate the energy overhead. In all test setups E^{2}TCP had less time overhead than the other protocols and therefore, its throughput will always be higher.

The latency of a protocol (measured in milliseconds), is another traditional performance metric. The latency of a packet is the time between the first transmission of the packet at the sending host and the successful reception of the packet by the destination host. Latency can be no lower than the delay of the wireless link, link layer and MAC layer combined. In case of retransmissions, the latency will surely increase. The average latency is the average of the latency of each packet. This metric will be used in this test. Unlike throughput, average latency has no direct relation to either data- or time overhead. Still, there is a weak relation between time overhead and average latency: the lower the time overhead the lower the average latency will *probably* be. Therefore, it is expected that E^{2}TCP will perform better than the other protocols. The average latency of E^{2}TCP and the other versions of TCP, were calculated from the results of the tests done with the default test setup and are shown in the two following graphs.

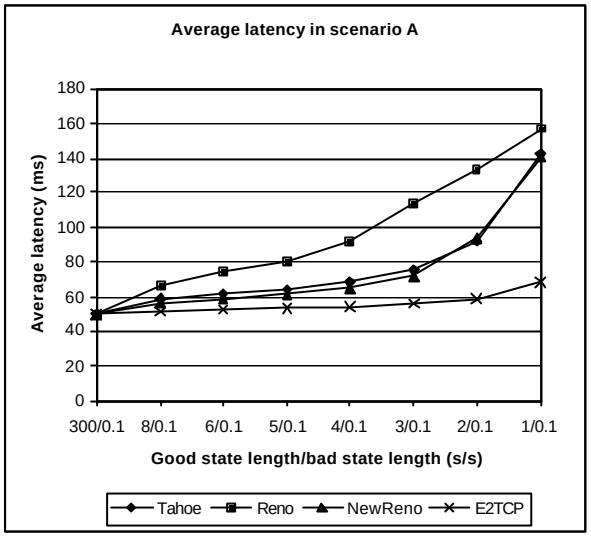

Figure 5.47: Average latency of various protocols in scenario A.

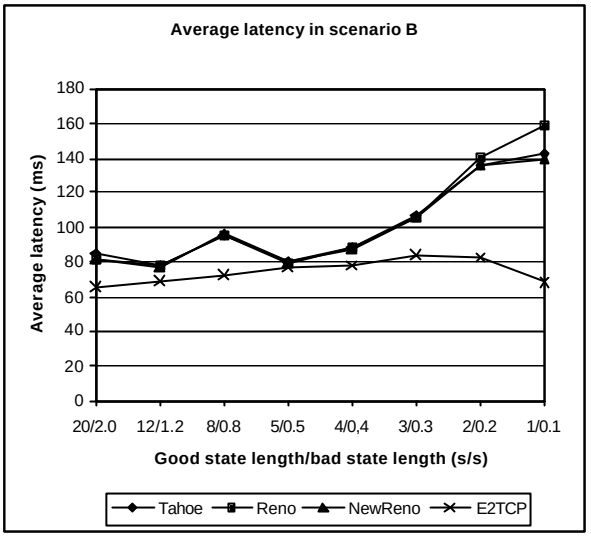

Figure 5.48: Average latency of various protocols in scenario B.

Clearly, the graphs show that the average latency of E^{2}TCP is lower than that of the other versions of TCP. As with throughput, the difference only increases when the channel conditions deteriorate. Because the default test 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 impressing. 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.

5.6.7 Conclusions

In Paragraph 5.6, a performance evaluation of E^{2}TCP was given. In the first test, Tahoe, Reno and NewReno were compared to E^{2}TCP in the default test setup. It can only be said that E^{2}TCP clearly has a higher energy efficiency than those other protocols. In the other tests Tahoe, which is the most energy efficient standard version of TCP, was compared to E^{2}TCP with different bandwidths, delays and types of traffic. E^{2}TCP consistently had a lower energy overhead than Tahoe. To give an indication of the efficiency of E^{2}TCP, the *average* energy overhead of the tested protocols will be listed in the following table for both scenarios, using the default test setup.

Protocol	Scenario A (%)	Scenario B (%)
Tahoe	44.5	54.5
Reno	68.6	63.4
NewReno	52.2	54.9
E^{2}TCP	10.1	17.4

Table 5.3: The average energy overhead of various protocols.

After that, some partial reliability tests were run, which compared PRTP with E^{2}TCP under various reliability constraints. Again, E^{2}TCP was the most energy efficient protocol. To give an indication of the efficiency of the protocols used in those tests, the *average* energy overhead of the tested protocols will be listed in the following table for both scenarios.

Protocol	Scenario A (%)	Scenario B (%)
PRTP (90% reliability)	9.7	19.3
E^2TCP (95% reliability)	7.1	14.6
E^2TCP (90% reliability)	4.9	12.6

Table 5.4: The average energy overhead of various protocols under various reliability constraints.

In Paragraph 5.6.6, a traditional performance evaluation of E^2TCP was conducted. From its results can be concluded that for both traditional performance metrics; throughput and latency, E^2TCP manages to outperform the other versions of TCP by a significant amount. To give an indication of the performance of E^2TCP, the *average* throughput and *average* latency of the tested protocols will be listed in the following two tables for both scenarios, using the default test setup.

Protocol	Scenario A (Mbps)	Scenario B (Mbps)
Tahoe	0.71	0.61
Reno	0.59	0.61
NewReno	0.64	0.62
E^2TCP	0.87	0.80

Table 5.5: The average throughput of various protocols.

Protocol	Scenario A (ms)	Scenario B (ms)
Tahoe	76.6	101.7
Reno	96.1	103.5
NewReno	74.7	100.8
E^2TCP	55.9	75.0

Table 5.6: The average latency of various protocols.

6 CONCLUSIONS AND RECOMMENDATIONS

From the results of the simulations presented in Chapter 5, the most important conclusion is that E^{2}TCP is indeed energy efficient. When comparing E^{2}TCP to standard versions of TCP, like Tahoe, Reno and NewReno, it is clear E^{2}TCP consistently has less energy overhead and therefore, a higher energy efficiency.

E^{2}TCP is optimized for energy efficiency on four points. Each optimization addresses one of the four problems of TCP, that keep it from reaching high levels of energy efficiency. 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 selective acknowledgements 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, which effect cannot really be determined separately, cause the greatest decrease in energy overhead: about 75% of the total decrease in energy overhead. The third optimization is the use of partial reliability to limit unwanted retransmits during the transmission of streaming media. This optimization is the cause of about 13% 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 TCP 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. The bandwidth was varied from 0.5 Mbps, representing lower speed serial links, to 5 Mbps, representing the new high speed IEEE 802.11b wireless network standard. 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 E^{2}TCP. The second conclusion is that independent of bandwidth, E^{2}TCP scales better than TCP when channel conditions deteriorate. It should also be clear that E^{2}TCP does not scale worse than TCP when bandwidth increases.

The delay of the wireless link, link layer and MAC layer combined was varied from 40 ms to 70 ms, representing all kinds of wireless links. Two conclusions can be drawn by studying the results. The first conclusion is that E^{2}TCP has a lower energy overhead than TCP on all delay/channel quality combinations and thus is more energy efficient. The second conclusion is that E^{2}TCP scales much better than TCP, when the channel delay increases.

Three types of traffic were used in the simulations of E^{2}TCP: interactive traffic (representing chatting and instant messaging for example), mass data transfers (representing file transfers, browsing and emailing for example) and constant bit rate traffic (representing streaming media). From the results can be concluded that E^{2}TCP continuously manages to outperform TCP in terms of energy efficiency, with every kind of traffic and all channel qualities.

A traditional performance evaluation of E^{2}TCP was also conducted. It consisted of throughput and latency comparisons with the standard versions of TCP. Because of the optimizations to reduce data and time overhead, the throughput of E^{2}TCP was increased. Therefore, no version of TCP was able to reach a higher throughput than E^{2}TCP. The delay of E^{2}TCP was also lower than that of other versions of TCP.

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.

As for future research, four areas are recommended to be examined. First of all, the assumptions made on energy efficiency and how it is calculated, should be subject to further examinations. As with all assumptions, it is unclear how accurately they portray reality. Therefore, a detailed study to how the actual energy consumption caused by a protocol can be measured and calculated from evident data, is desirable.

The second area of recommended research is the characteristics of burst errors. Little research can be found on what kind of error characteristics wireless links experience. Therefore, it is unclear how to accurately model errors in simulated environments. If more information would be available on the length of burst errors, the time between consecutive burst errors and the bit error rate of the wireless link under normal conditions and during burst errors, more accurate models could increase the reliability of simulations.

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 most important reason for implementing the base station is to examine the effect of a complete connection on the energy efficiency of E^2TCP (at the mobile host).

The final recommendation for future research is to compare E^2TCP to other protocols for wireless links. A lot of adaptions of TCP have been proposed 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 certainly possible these protocols are more energy efficient than standard TCP. Whether or not they are able to surpass E^2TCP, remains to be seen, but is certainly an interesting research area. For such a comparison implementations of these protocols in NS2 would have to be obtained.

BIBLIOGRAPHY

[BAK95] Bakre A.V., Badrinath B.R., *I-TCP: Indirect TCP for mobile hosts*, Proceedings of the 15th international conference on distributed computing systems, May 1995.

[BAK97] Bakre A.V., Badrinath B.R., *Implementation and performance evaluation of indirect TCP*, IEEE transactions on computers, V. 46 N. 3, March 1997.

[BAL95] Balakrishnan H., Srinivasan S., Katz R.H., *Improving reliable transport and handoff performance in cellular wireless networks*, ACM wireless networks, V. 1 N. 4, 1995.

[BLU01] Bluetooth Special Interest Group, *Specification of the Bluetooth system – volume 1: Core*, Version 1.1, http://www.bluetooth.com/, February 2001.

[BRO97] Brown K., Singh S., *M-TCP: TCP for mobile cellular networks*, Computer communications review, V. 27 N. 5, October 1997.

[BRU00] Brunstrom A., Asplund K., Garcia J., *Enhancing TCP performance by allowing controlled loss*, Proceedings of SSGRR 2000 computer & ebusiness conference, L'Aquila, Italy, August 2000.

[CAS99] Casner S., Jacobson V., *Compressing IP/UDP/RTP headers for low-speed serial links*, RFC 2508, February 1999.

[CHE94] Chen K., *Medium access control of wireless LANs for mobile computing*, IEEE network magazine, V. 8 N. 5, September 1994.

[COM95] Comer D.E., *Internetworking with TCP/IP – volume 1: Principles, protocols and architecture*, 3rd edition, Prentice-Hall, Upper Saddle River, The United States of America, 1995.

[DEG99] Degermark M., Nordgren B., Pink S., *IP header compression*, RFC 2507, February 1999.

[ECK96] 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.

[ENG99] Engan M., Casner S., Bormann C., *IP header compression over PPP*, RFC 2509, February 1999.

[FAL96] Fall K., Floyd S., *Simulation-based comparison of Tahoe, Reno and SACK TCP*, Computer Communication Review, V. 26 N. 3, July 1996.

[FAL00] Fall K., Varadhan K., *The NS manual*, The VINT project, http://www.isi.edu/nsnam/ns/, October 2000.

[FLO00] Floyd S., Mahdavi J., Mathis M., Podolsky M., *An extension to the selective acknowledgement (SACK) option for TCP*, RFC 2883, July 2000.

[GAR00] Garcia J., Brunstrom A., *A robust JPEG coder for a partially reliable transport service*, Proceedings of the 7th international workshop IDMS 2000, Enschede, The Netherlands, October 2000.

[HAA97] Haas Z.J., Agrawal P., *Mobile-TCP: an assymetric transport protocol design for mobile systems*, ICC '97, Montreal, Canada, June 1997.

[HAV98] Havinga P.J.M., Smit G.J.M., *E^2MaC: an energy efficient MAC protocol for multimedia traffic*, Moby Dick technical report, http://www.cs.utwente.nl/~havinga/, 1998.

[HAV99] Havinga P.J.M., *Energy efficiency of error correction on wireless systems*, IEEE wireless communications and networking conference, September 1999.

[HAV00] 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.

[HAV00b] Havinga P.J.M, Smit G.J.M., Bos M., *Energy efficient adaptive wireless network design*, The 5th symposium on computers and communications (ISCC'00), Antibes, France, July 2000.

[IEE99] IEEE, *Wireless LAN medium access control (MAC) and physical layer (PHY) specifications*, IEEE standard 802.11, 1999.

[IEE99b] IEEE, *Higher speed physical layer (PHY) extension in the 2.4 GHz band*, IEEE standard 802.11b, 1999.

[JAC88] Jacobson V., Braden R., *TCP extensions for long-delay paths*, RFC 1072, October 1988.

[JAC90] Jacobson V., *Compressing TCP/IP headers for low-speed serial links*, RFC 1144, February 1990.

[MAT96] Mathis M., Mahdavi J., Floyd S., Romanov S., *TCP selective acknowledgement options*, RFC 2018, October 1996.

[RAM99] Ramakrishnan K., Floyd S., *A proposal to add explicit congestion notification (ECN) to IP*, RFC 2481, January 1999.

[RAT98] Ratnam K., Matta I., *WTCP: an efficient mechanism for improving TCP performance over wireless links*, Proceedings of the 3rd IEEE symposium on computer and communications, June 1998.

[STE97] Stemm M., Katz R.H., *Measuring and reducing energy consumption of network interfaces in hand-held devices*, IEICE transactions on communications, V. E80-B N. 8, 1997.

[VAI99] Vaidya N.H., Mehta M., Perkins C., Montenegro G., *Delayed duplicate acknowledgements: a TCP-unaware approach to improve performance of TCP over wireless*, Technical report 99-003, Computer science department, Texas A&M University, February 1999.