2004:261 CIV

EXAMENSARBETE

DCCP-Thin in Symbian OS

MAGNUS ERIXZON

MASTER OF SCIENCE PROGRAMME

Luleå University of Technology
Department of Computer Science and Electrical Engineering
EISLAB • Embedded Internet Systems Laboratory

LULEÅ TEKNISKA UNIVERSITET

2004:261 CIV • ISSN: 1402 - 1617 • ISRN: LTU - EX - - 04/261 - - SE

DCCP-Thin for Symbian OS

Magnus Erixzon

September 15, 2004

Abstract

The Datagram Congestion Control Protocol (DCCP) is a new Internet transport protocol. It is meant to serve as an alternative protocol for applications that require low delays and where a certain degree of data loss is acceptable. TCP is not well-suited for these applications, since its reliable in-order delivery, combined with its congestion control, can cause arbitrarily long delays. UDP avoids long delays, but UDP does not provide congestion control. DCCP provides built-in congestion control for unreliable datagram flows.

DCCP supports several different congestion control profiles. One such profile is aimed towards small clients, i.e. devices that have limited memory and/or processing power. An example of such a device is the mobile phone. Modern mobile phones today use the Symbian OS, which is the standard operating system for mobile devices. The goal of this Master Thesis was to implement DCCP with the thin congestion control profile in the Symbian OS.

Experiments show that the DCCP implementation adapts well to changes in the network and in the mobile phone, but that the currently available communication technology, GPRS, is too limited to support real-time applications where DCCP could be applicable.

Contents

Chapter 1

Introduction

1.1 Background

The Internet connects computer networks world wide. Computers communicate with each other using standardized protocols. The protocols are divided into different layers based on the functionality they provide. Certain protocols handle the routing of network packets through the network, others govern the details regarding messaging.

Transport protocols handles the transfer of data between computer systems and exist only in the end nodes of a communication. The Internet currently has two widely deployed transport protocols, Transmission Control Protocol (TCP) [1] and User Datagram Protocol (UDP).

TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer networks. Very few assumptions are made about the reliability of the communication protocols below the TCP layer. TCP establishes the connection before sending data, and verifies that the data has been received. The data transmission rate is adjusted in accordance with the capacity of the network. TCP is used for data transmissions where reliability is essential. Examples include email messages and web page delivery.

UDP provides a connection-less, unreliable service. The connection is not established and delivery is not guaranteed. The transmission rate does not adapt to network capabilities. UDP can be used to transfer data without establishing the connection between two computers. UDP is for example used by the DNS [1] query protocol. UDP can also be used in live media transfers that require low delays, and where the reliability is not crucial.

[1]Domain Name System, used to translate between host names and Internet Protocol addresses

Network capabilities world wide are increasing. A growing number of computer users have access to high-speed Internet connections, which augments the demand for multimedia streaming, live audio and video distribution. The problem with using UDP for these applications is that large amounts of UDP traffic potentially can starve other flows in the network that back off in case of congestion. A mechanism, called congestion control, is needed to ensure that all data-flows get a fair amount of the available bandwidth.

Congestion control is about distributing bandwidth usage between nodes in the network. Compare this with a road intersection. Traffic lights make sure cars in all directions are granted time to pass through the intersection. Without any traffic lights, cars would collide when traffic increase.

Some applications already implement their own congestion control over UDP, but experience shows that congestion control is difficult to get right, and application programmers should not be forced to reinvent this particular wheel. A solution is introducing a new transport protocol. The Datagram Congestion Control Protocol (DCCP) [2] provides an unreliable datagram service, just like UDP, but with built-in congestion control.

DCCP offers several congestion control profiles. Each profile is assigned a specific ID, referred to as a Congestion Control IDentifier (CCID). The application programmer can choose the profile that is most appropriate for the specific application. The current DCCP drafts describe two such mechanisms. The first is called TCP-like and has CCID 2 [3]. As the name suggest, it provides congestion control similar to that of TCP. The second profile, CCID 3, is named TCP-Friendly Rate Control (TFRC) [4]. This mechanism aims to minimize abrupt changes in the sending rate, while still operating fairly with TCP. Its design would for example be preferred for applications streaming media.

A variant of TFRC has been proposed, named CCID 3-Thin [5]. It is more restricted than CCID 3; it limits allowable options, acceptable feature values, and so forth. It was designed for small clients where a full DCCP implementation would be too expensive. Mobile phones are perfect examples of such clients. This Master Thesis aims to implement the DCCP protocol with CCID 3-Thin congestion control, in the Symbian OS.

1.2 Symbian OS

Symbian [6] is a software licensing company and the trusted supplier of the Symbian Operating System. It was established as a private independent company in 1998, and is owned by Ericsson, Nokia, Panasonic, Samsung

Electronics, Siemens and Sony Ericsson.

Symbian OS is an operating system for data-enabled mobile phones. It is flexible and scalable enough to be used in a big variety of mobile phones. There are a set of standard application programming interfaces enabling software developers to build applications and services for a global mass-market of advanced mobile phones.

1.3 Goals

The primary goal of this Master Thesis is to present a working implementation of DCCP-Thin in Symbian. The code should be well documented and problems that have been encountered should be reported. Also, an analysis of the design of the thin client shall be carried out within the frames of the Master Thesis, i.e., is there any functionality that should have been kept/can more functions be removed?

Chapter 2

Communication Technologies

2.1 Introduction

A mobile phone uses wireless technology. Wireless communications is currently a very active area of development. The development is primarily being driven by the transformation of what has been mainly a medium for supporting voice telephony into a medium for supporting packet-data, such as the transmission of video, images, text, and data. There are now over one billion GSM users worldwide [7]. These numbers make cellular telephony a very important driver of wireless technology development. In recent years the push to develop new mobile data services, which go collectively under the name third-generation (3G) cellular, has played a key role in motivating research in new techniques for wireless communication.

2.2 3G

Third generation (3G) mobile telephony is the generic term used for the next generation of mobile communications systems. It provides data rates of at least 2 Mbps for stationary use, and at least 384 kbps while traveling less than 120 kilometers per hour [8]. 3G services will add an invaluable mobile dimension to services that are already becoming an integral part of modern business life: Internet and Intranet access, video-conferencing, and interactive application sharing.

Although 3G would be the preferred communication technology, it is currently not available to the author and will therefore not be used in this Master Thesis.

2.3 GPRS

In the absence of 3G, General Packet Radio Service (GPRS) is the most advanced technology for data services. GPRS provides 'always-on' networking for mobile devices. GPRS has a theoretical maximum speed of 171.2 kilobits per seconds (kbps) [9], but that is just theoretical. The Sony Ericsson P800 phone is limited to a sending rate of 8-12 kbps, and a receiving rate of 32-40 kbps [9]. The operator might also put limits on the bandwidth. Another drawback, besides the bandwidth limit, is that GPRS usage is charged on a byte basis [1].

This is the "real-world" technology used for testing in this Master Thesis.

2.4 Through computer

By connecting the mobile device to a computer, it is possible to get Internet access. While this technique has no "real-world" value, it is valuable for testing. The main reason being that it is free of charge.

[1] At least in Sweden

Chapter 3

Symbian

3.1 Introduction

The Symbian operating system is designed for the specific requirements of advanced, data-enabled mobile phones. Symbian OS is based on a micro kernel architecture and implements full multi-tasking and threading. System services such as telephony, networking middle-ware and application engines all run in their own processes. The operating system has been designed from the ground up with mobile devices in mind, using advanced object-oriented techniques, leading to a flexible component based architecture.

Symbian OS provides a rich core of application programming interfaces that are common to all Symbian OS phones. Key features of Symbian OS v7.0 are:

- Browsing - a WAP stack is provided with support for WAP 1.2.1 for mobile browsing.
- Messaging - multimedia messaging (MMS), enhanced messaging (EMS) and SMS; Internet mail using POP3, IMAP4, SMTP and MHTML; attachments; fax.
- Multimedia - audio and video support for recording, playback and streaming; image conversion.
- Graphics - direct access to screen and keyboard for high performance; graphics accelerator API.
- Communications protocols - wide-area networking stacks including TCP/IP (dual mode IPv4/v6) and WAP, personal area networking support include infrared (IrDA), Blue-tooth and USB; support is also provided

for multi-homing capabilities and link layer Quality-of-Service (QoS) on GPRS/UMTS networks.

- Mobile telephony - Symbian OS v7.0 is ready for the 3G market with support for GSM circuit switched voice and data as well as packet-based data.
- International support - conforms to the Unicode Standard version 3.0.
- Data synchronization - over-the-air synchronization support using SyncML; PC-based synchronization over serial, Blue-tooth, Infrared and USB; a PC Connectivity framework providing the ability to transfer files and synchronize PIM data.
- Security - full encryption and certificate management, secure protocols (HTTPS, WTLS and SSL and TLS), WIM framework and certificate-based application installation.

Programming in the Symbian OS can be made using 2 languages, C++ or Java. C++ was chosen in this Master Thesis, since a network protocol is a kernel component. The Symbian OS is written in C++.

3.2 Code design

Since the Symbian OS is used on several different platforms, one single common user interface (UI) is not appropriate. Instead, the Symbian OS provides several user interfaces targeted at specific mobile phones.

There are currently three available user interfaces:

- Series 60 - Designed for the phones with the smallest screens. These phones are for one-handed use and thus require an UI that is simple to navigate with a joystick, through soft-keys or a jog-dial. Examples of such phones are the Siemens SX1 and the Nokia 6660 models.
- UIQ - For phones with a larger screen, which support pen-based interaction. The Sony Ericsson P800 and the Motorola A925 models are two phones using the UIQ user interface. Since testing will be done using a Sony Ericsson P800 phone in this Master Thesis, UIQ is the interface used.
- Series 80 - Have the largest screen of all Symbian OS phones, and can use a full keyboard and/or a touch screen for user input. The Nokia 9500 is an example of a device using the Series 80 user interface.

Since Symbian does provide several different user interfaces, it is important to separate the application engine [1] from the application user interface. When porting the code to another phone, only the user interface has to be replaced.

There are rather strict standards [10] specified for produced code, covering everything from naming conventions to code indentation. An interesting aspect of the function names is that there is functionality built into the names themselves. If the name ends with a capital L, it means that function can *Leave*.

If a function can *Leave*, that means it can fail [2]. The program will then be closed and resources released. When developing for Symbian, it is important to consider what parts might fail, and if proper arrangements has been made to free all resources and the allocated memory.

3.3 Debugging

Debugging is an essential part of development. There exists both an emulator and tools for on-device debugging, both they both require the developer to purchase an IDE [3]. The IDE endorsed in the book "Symbian OS C++ for Mobile Phones" [11] is Code Warrior from Metrowerks.

Without these tools, it is quite tricky to properly debug the code. All testing have to be carried out on the mobile phone.

3.4 Kernel programming

A network protocol is different from a normal application in that it should be a part of the kernel, that can be used by application programmers. The interface between the DCCP network protocol and the application should not differ from the interface to other network protocols such as TCP. This is currently not possible. The API required to create a proper network protocol has not been publically released by the Symbian Ltd. corporation. Without this API, the DCCP 3-Thin implementation could not be included in the kernel as was originally intended. Instead, the DCCP protocol will be using the UDP protocol to communicate and the code will be created as an application library. It should however be easy to convert the code to a kernel implementation when Symbian Ltd. decides to release the required API.

[1] The part that does all computing
[2] Compare with exceptions in Java
[3] Integrated Development Environment

Chapter 4

DCCP Thin

4.1 Introduction

The Internet Engineering Task Force (IETF) [12] is the organization that handles the standardization of Internet protocols. While a protocol is being developed, its specification is published as an Internet draft. Such documents are work in progress that may be updated or replaced at any time. When a protocol has been adopted as a standard, its specification is published as part of the "Request For Comments" (RFC) document series. The documents specifying DCCP are at draft level and thus work in progress that may be updated or replaced at any time.

The general theory behind congestion control is to adjust the transmission speed according to the currently available bandwidth over the link. Several different mechanisms exist (see [13] for the algorithms used by TCP). This Master Thesis will focus on TCP-Friendly Rate Control (TFRC) [14] congestion control.

TFRC is a receiver-based congestion control mechanism that provides a TCP-friendly send rate, while minimizing abrupt rate changes. It is appropriate for applications that prefer a relatively smooth sending rate. Such applications include streaming media applications with small or moderate buffering possibilities at the receiver application before playback. TCP-like congestion control, which halves the sending rate in response to a congestion event, cannot satisfy this preference for a relatively smooth sending rate.

During a session, the TFRC sender transmits a stream of data packets to the receiver at some rate. The receiver sends a feedback packet to the sender roughly once every round-trip time [1]. Based on information contained in the feedback packets, the sender adjusts its sending rate in accordance with an

[1] The time it takes for a packet to travel from the sender to the receiver and back.

equation-based model of TCP. If no feedback packets are received in several consecutive round-trip times, the sender halves its sending rate. Before we delve into the specific details of the congestion control algorithms, we define the necessary variables.

- RTT - Round-trip time. The time it takes for a packet to travel from the sender to the receiver and back in seconds.
- X - Transmit rate. The number of bytes which the sender is allowed to send per second.
- S - Packet size in bytes.
- P - Loss event rate. The number of loss events as a fraction of the number of packets transmitted. Takes a value between 0.0 and 1.0.
- TO - Timeout. The sender will continue to send without receiving any feedback until the "no feedback timer" is triggered and then the send rate is decreased.

During connection setup, the sender estimates the initial RTT. The current drafts specify that RTT measurements should start after a connection has been established. Measuring the RTT during setup instead of initially using a fixed, pre-defined value has however been discussed on the DCCP mailing list and is the method used in our implementation. When the connection has been established, the sender starts by sending 4 packets per RTT. After the first data packet has been sent, the sender waits $RTT/4$ seconds before sending the next one, to get equally long intervals between packets. When sending the first packet, a "no feedback timer" is started which has an initial time limit, TO, of $RTT * 4$ seconds.

The transmit rate, X, is used to control how many bytes that can be sent per second. It is initially set to $4 * S/RTT$, which means 4 packets are allowed to be sent per round-trip time. It is then adjusted according to the feedback information received.

The packet size S should be an average of recent packet sizes [4]. The draft is not specific about how to estimate the average packet size. This implementation estimates S as the mean of the packet sizes of the 10 last sent packets. Since this congestion control algorithm is rate based, a user should not be able to force a higher throughput by increasing the packet sizes.

4.2 Receiving feedback

When the sender receives an acknowledgment packet, there are a number of actions to be performed.

- Calculate a new round-trip sample. This is done by subtracting the time when the packet acknowledged was sent, from the time the feedback was received. A delay value is also deducted, which is the time passed from when the data packet arrived at the receiver, until feedback was sent. By subtracting this delay, the sender gets a better estimate of the actual network component of the round-trip time.

- Update the round-trip estimate using equation 4.1.

$$RTT_{avg} = 0.8 * RTT_{old} + 0.2 * RTT_{sample} \tag{4.1}$$

  RTT_{old} is the moving average of the previous RTT samples. RTT_{sample} is the latest RTT value sampled. Through filtering, the RTT_{avg} is less sensitive to sudden dramatic changes in an individual sample.

- Update the no-feedback timeout interval.

$$TO = 4 * RTT_{avg} \tag{4.2}$$

- Update the sending rate. If the loss event rate is 0, the sender is in the so called slow-start phase, where it approximately doubles the sending rate each round-trip time until a loss occurs. When the sender has left the slow-start phase, the TCP throughput equation, 4.3, is used to compute the allowed sending rate.

$$X = \frac{S}{RTT_{avg} * \sqrt{2 * P/3} + (TO * (3 * \sqrt{3 * P/8}) * P * (1 + 32 * P^2))} \tag{4.3}$$

  In this implementation, the computation is split up in two steps,

$$f(P) = \sqrt{2 * P/3} + (12 * \sqrt{3 * P/8} * P * (1 + 32 * P^2)) \tag{4.4}$$

  and

$$X = \frac{S}{RTT_{avg} * f(P)}. \tag{4.5}$$

A lookup table will be used for the function $f(P)$.

4.3 No feedback timer

The no feedback timer is used to detect when no acknowledgment packets has been received within the time period given by equation 4.2. When the timer expires, the sending rate is cut in half. The no feedback timer is then restarted with a value computed using equation 4.6.

$$TO = max(4 * RTT_{avg}, 2 * \frac{S}{X}) \tag{4.6}$$

The timer is necessary to make the sender slow down if there are several packets lost in either direction. If a lot of packets are lost, the data sender might not receive any feedback from the data receiver. The sender would then continue to send at the same too high rate. The no-feedback timer makes sure the sender slows down in this situation.

4.4 Loss event rate

The loss event rate is an estimate of how many packets that have been lost of those transmitted, and is calculated by the receiver. The receiver keeps track of which packets were received, and when they were received. If a packet with sequence number $X + 3$ is received, when the packet with sequence number X was not yet received, packet X is considered lost [2].

If a packet is considered lost, the receiver first has to determine if the packet loss belongs to an old loss event, or if it starts a new one. Therefor the receiver compares the sequence numbers and timestamps of the packets that have arrived. The following variables are used:

- S_{loss} - The sequence number of the lost packet
- S_{before} - The sequence number of the last packet to arrive with sequence number lower than S_{loss}
- S_{after} - The sequence number of the first packet to arrive with sequence number higher than S_{loss}
- T_{loss} - Estimated reception time of S_{loss}

[2] TFRC allows a certain amount of reordering

- T_{before} - Reception time of S_{before}
- T_{after} - Reception time of S_{after}

The reception time of the lost packet is estimated through equation 4.7 .

$$T_{loss} = T_{before} + \frac{(T_{after} - T_{before}) * (S_{loss} - S_{before})}{S_{after} - S_{before}} \tag{4.7}$$

If $T_{before} + RTT_{avg} \geq T_{loss}$ then S_{loss} is part of the existing loss event. Otherwise it is the first packet of a new loss event.

Thereafter the average loss interval is compared using a filter that weighs the 8 most recent loss intervals. The weights W_0 to W_7 have the values $1.0, 1.0, 1.0, 1.0, 0.8, 0.6, 0.4, 0.2$. The average loss interval I_{mean} is then given by equation 4.8.

$$I_{mean} = \frac{\sum_{n=0}^{7} I_n * W_n}{\sum_{n=0}^{7} W_n} \tag{4.8}$$

and the loss event rate P is simply

$$P = \frac{1}{I\_mean} \tag{4.9}$$

Chapter 5

Experiments

5.1 Introduction

As stated earlier in this report, it was not possible to create a real transport protocol in this Master Thesis. Thus, all data was tunneled through UDP.

While testing the code we found that TCP is given a much higher priority than UDP in the mobile phone. This was noted when an application wanted to send two data streams, one over TCP and the other over UDP, simultaneously. When the TCP flow started, the UDP stream was basically halted. Therefore, it was not possible to carry out tests where TCP and DCCP were competing.

In the tests the mobile phone communicates with a user-space application running on a computer. The computer application was also written. If it had been possible to make a real network protocol, communication with the FreeBSD DCCP implementation created in a project course at Luleå University of Technology [15], would have been preferable.

Some notes about the tests:

- The GPRS provider is Telia [16].
- The computer end-point runs Linux kernel version 2.6.3 [17]. It is assumed that the Internet Protocol and the Transmission Control Protocol implementations in this kernel have valid functionality.
- For the DCCP (over UDP) tests, a Perl [18] program has been written.

5.2 TCP

We will first do tests with TCP. It is the best starting point for exploring the capabilities of the mobile phone and the network, since its implementation is independent of ours. Afterwards it is possible to compare the TCP and DCCP results. From the TCP results we will also be able to draw some general conclusions about the network abilities of the mobile phone in question.

5.2.1 Phone to computer over data-link

Initially, we let the mobile phone send data to a computer. The phone is connected to the Internet through the USB port of another computer. The network is a 2 hop link over a local Ethernet network.

```
Mobile phone <--( USB )--> Computer <--( network )--> Computer
```

In figure 5.1, we see the throughput of a TCP stream when an application running on the mobile phone is sending data at the maximum rate.

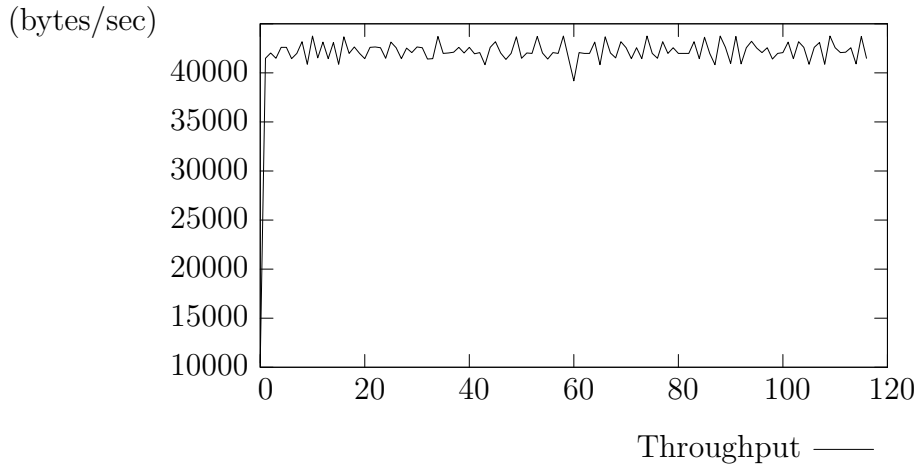

Figure 5.1: 1 TCP stream from phone to computer

The average throughput is about 42000 bytes/second, which seems to be the maximum data rate the Sony Ericsson P800 mobile phone can send at.

In the next test, the application will attempt to send 2 data streams at maximum rate. The purpose of this test is to see if both streams get a fair share of the available bandwidth. The throughput of the data streams can be seen in figure 5.2.

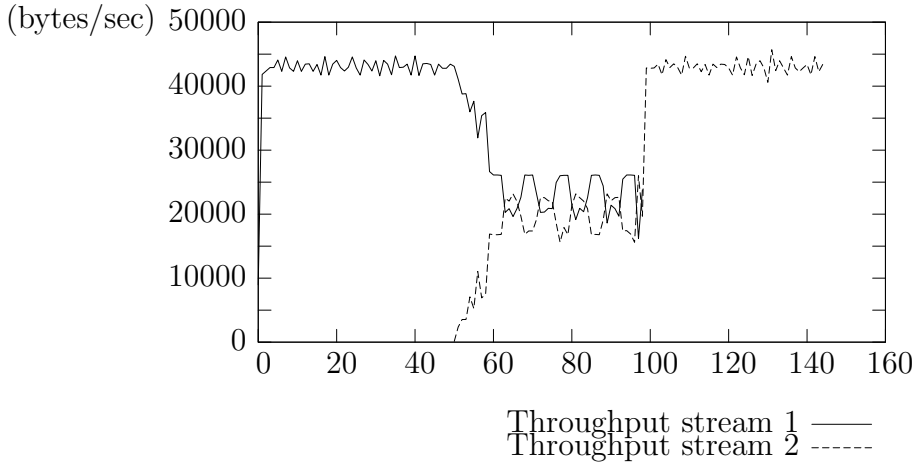

Figure 5.2: 2 TCP streams from phone to computer

The first TCP stream is started at second 0. After 50 seconds, another TCP stream is started. As we can see, the two streams share the available bandwidth. After yet another 50 seconds, the first stream is closed. Stream 2 then gets all the bandwidth until it finishes. This is the expected behavior.

5.2.2 Computer to phone over data-link

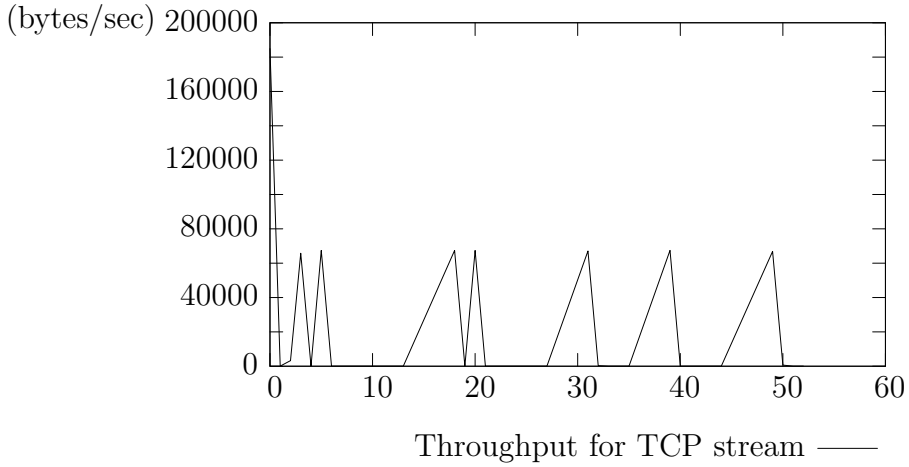

Figure 5.3: TCP stream from computer to phone, throughput

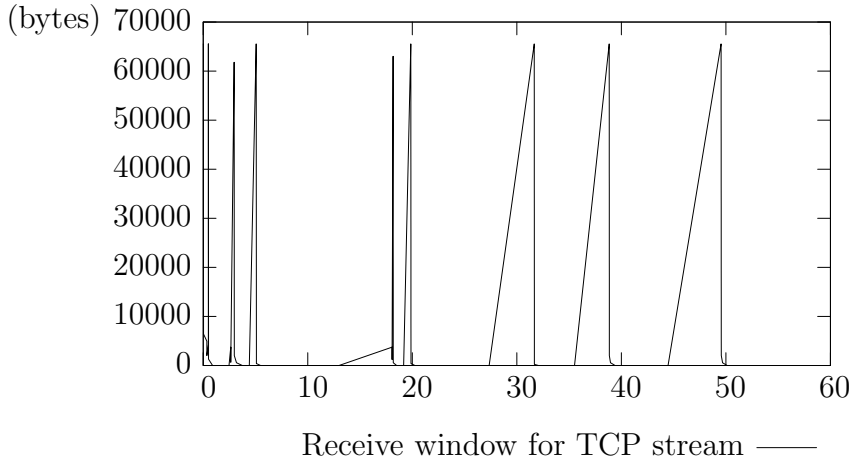

Figure 5.4: TCP stream from computer to phone, receive window

As seen in the previous test, the mobile phone seems to have an upper limit of about 42000 bytes/second when sending data. When the application is sending data at the maximum speed - from the computer to the mobile phone - we get the throughput shown in figure 5.3 .

It appears that the data is sent in bursts, with long intervals of silence. This strange behavior is explained by the receive window values reported to the sender, which can be seen in figure 5.4 .

The receive window equals zero several times and for long periods of time. The sender may then only probe the receiver by sending small packets to find out when the receive window is greater than zero. The computer sends data at a too high rate for the phone to handle. Its receive buffer is filled up and it sets the receive window to zero, which causes the computer to cease sending data. After the receiver has processed some of the data in its receive buffer, the phone increases the receive window, allowing the computer to send at a higher rate. This behavior repeats itself over and over again.

We conclude that the Sony Ericsson P800 is not made to handle such high data rates. This is however neither strange nor a problem, since the only real-world [1] connection method available is via the relatively slow GPRS. Knowing the limitations of the mobile phone is however important when selecting test cases. We now know the maximum rate at which the phone can send and receive data and we are aware that sending data to the phone at higher rates can cause strange behaviors.

5.2.3 Computer to phone over GPRS

Now we change technology to GPRS. The communication path is visualized below:

```
Mobile phone <--( GPRS .. Internet )--> Computer
```

Besides the change of the network topology, the setup is the same. We are interested in finding out which sending rate that can be achieved over GPRS, and how smooth the sending rate is.

[1] In situations where one would be interested in connecting to the Internet using a mobile phone, one most probably would not be situated at a networked computer

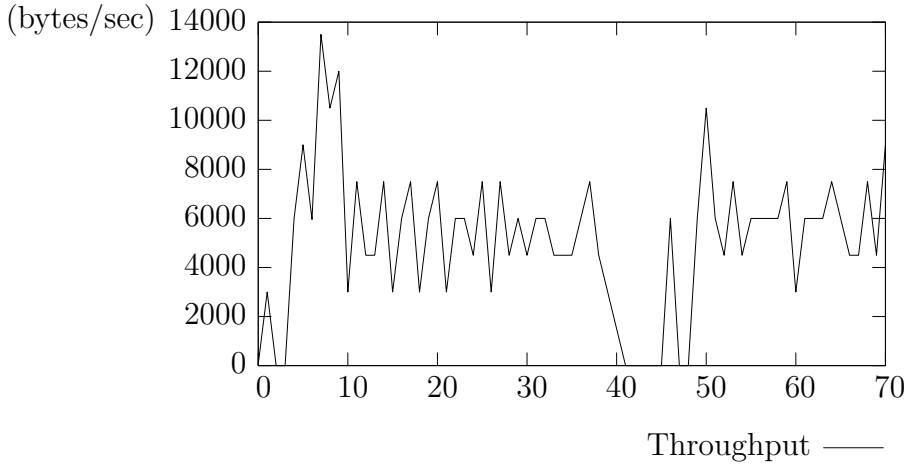

Figure 5.5: TCP stream from computer to phone over GPRS, throughput

In figure 5.5 we see the throughput for the data stream. It settles at about 6000 bytes/second. The flow is fairly smooth, except for a glitch between seconds 38 and 48. For several seconds, no data packets are being sent. Through several tests, a similar glitch appeared in each transmission, seemingly random in location and length.

The receive window is zero at one point, but that happens at second 44, see figure 5.6. At second 44, the sender has already been silent for several seconds. Hence, the receive window does not cause the interruption of the transfer. The receive window figure does however reveal that a large amount of data is received right before second 44. It is possible that several packets of data were held up in the network and reached the receiver close after one another.

To see if this assumption is correct, we examine figure 5.7 which shows the reception times of the individual data packets. Most of the time, the reception times increase steadily. Between seconds 38 and 43, no packets are received. At second 43, a lot of packets are received at virtually the same time.

Figure 5.7 verifies the assumption that the glitch is caused by a sudden delay spike caused by the network.

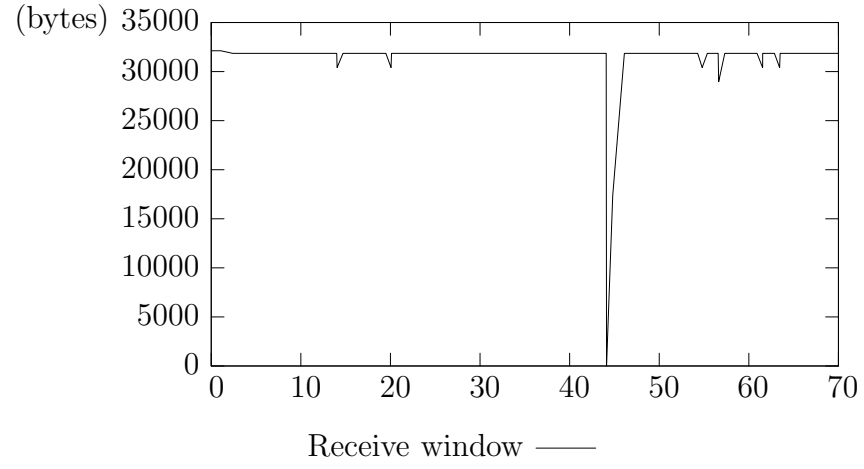

Figure 5.6: TCP stream from computer to phone over GPRS, receive window

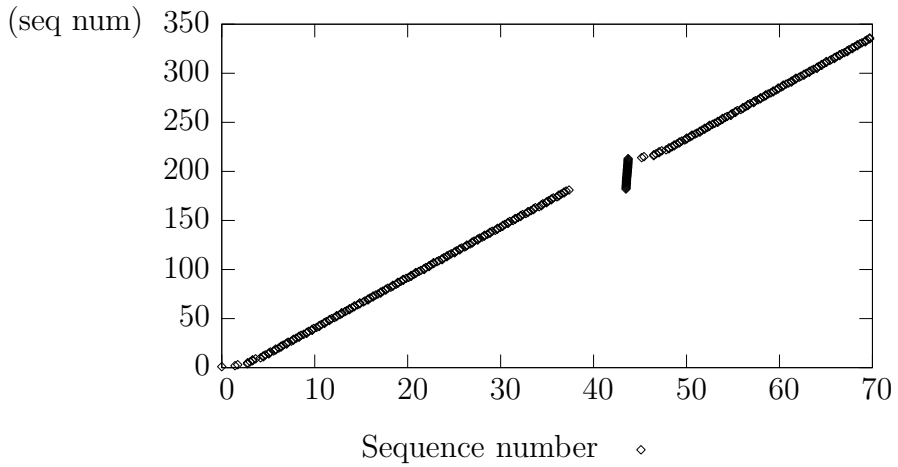

Figure 5.7: TCP stream from computer to phone over GPRS, reception times

5.3 DCCP

The purpose of the tests performed with DCCP is to find out how well it utilizes the available bandwidth and whether two DCCP streams can share the available bandwidth fairly.

5.3.1 Phone to computer over data-link

The network topology is the same as in the TCP tests;

```
Mobile phone <--( USB )--> Computer <--( network )--> Computer
```

In the first test, we send a single DCCP stream from the phone to the computer. The application is sending at maximum rate. In figure 5.8, the throughput is shown. The throughput averages at about 42000 bytes/second, which is the same average throughput we observed for TCP in this case. Thus we conclude that DCCP manages to use available bandwidth.

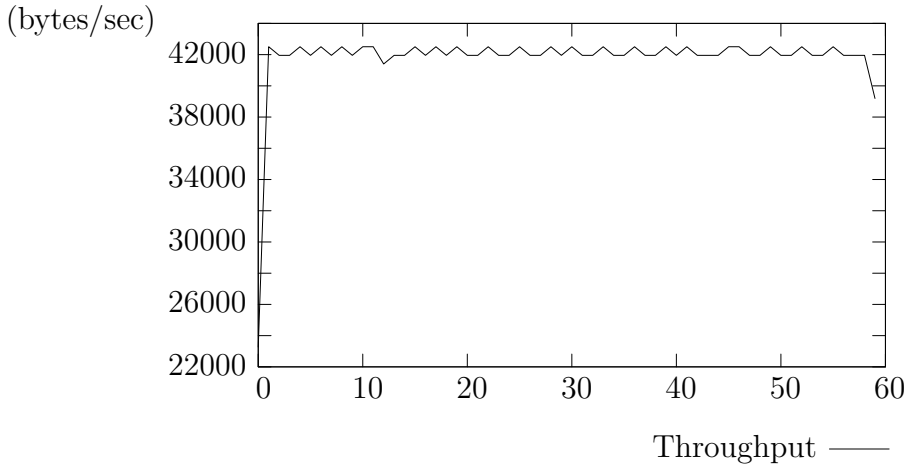

Figure 5.8: DCCP stream from phone to computer, throughput

Figure 5.9 shows the throughput of 2 DCCP streams, both being sent at maximum rate from the phone to the computer. Stream 2 is started 23 seconds after stream 1. Stream 1 is closed after 45 seconds. The figure clearly shows that two simultaneous DCCP streams share the available bandwidth fairly.

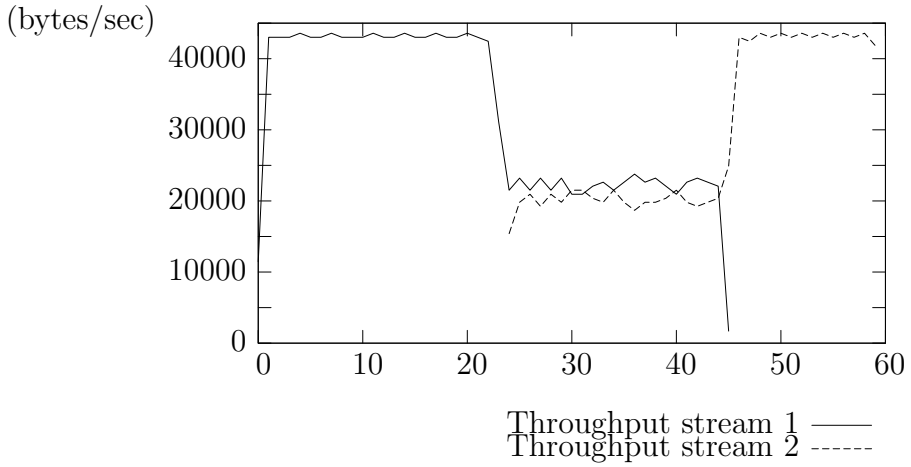

Figure 5.9: 2 DCCP streams from phone to computer

5.3.2 Computer to phone over data-link

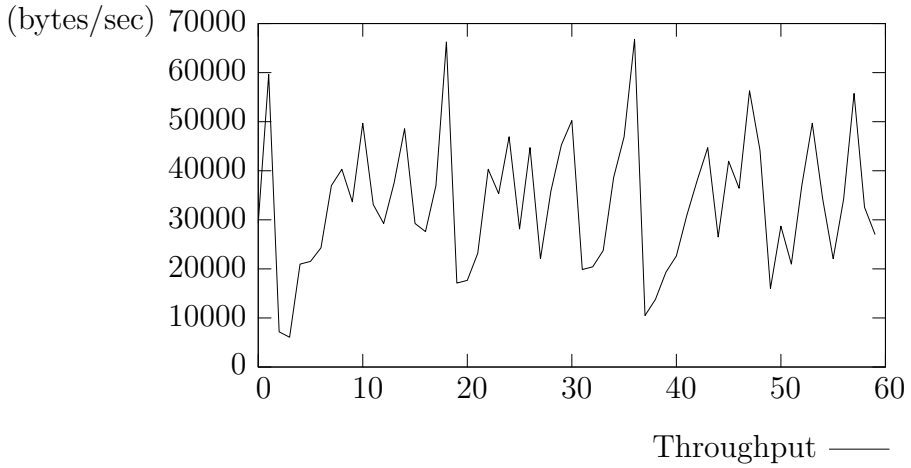

Figure 5.10: DCCP stream from computer to phone, throughput

Figure 5.10 shows the throughput for the DCCP stream when the application is sending at maximum rate from the computer to the phone. The send rate is not smooth. As we saw in the TCP test, the phone can not handle high data rates well.

In the TCP tests, we looked at the receive window information in the packets. DCCP-Thin does not have anything that corresponds to a receive window. Instead, we look at the receive rate information, which indicates at

which rate the receiver is accepting data. The receive rate has been visualized in figure 5.11. Since there is no packet loss during this transmission, the receive rate is a large factor when calculating the send rate. The sender is in the so called slow-start phase where the send rate is doubled each round-trip time, but the send rate can never more than twice the last reported receive rate.

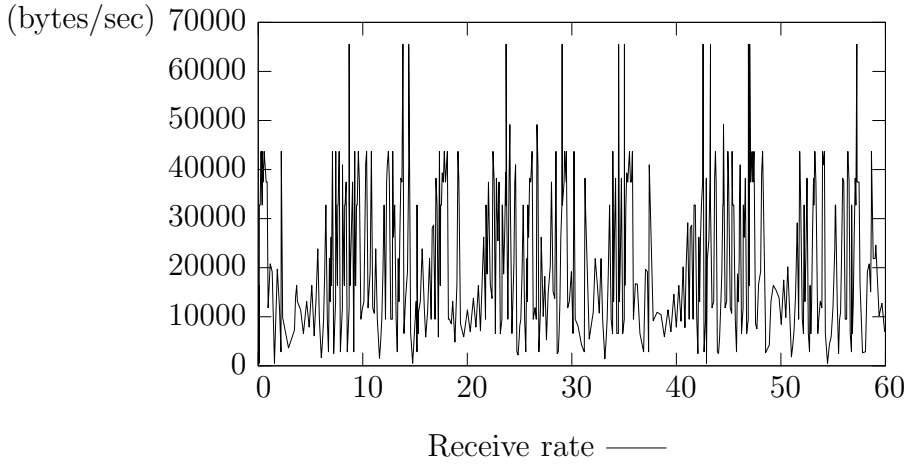

Figure 5.11: DCCP stream from computer to phone, receive rate

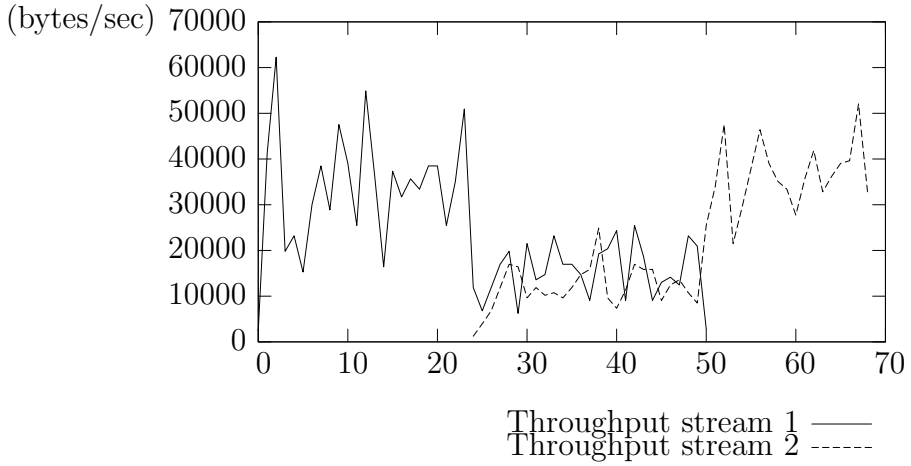

Figure 5.12: 2 DCCP streams from computer to phone

Figure 5.12 shows the throughput for two simultaneous DCCP streams. Stream 2 is started about 23 seconds after stream 1, and stream 1 is closed

after about 50 seconds. The figure shows that the two streams share the available bandwidth fairly.

A difference we can note here, compared to the TCP graph, is that the streams are much steadier. This is due to the nature of TFRC, which aims at keeping a constant sending rate. But there are still large fluctuations, due to the phones inability to handle high data rates.

5.3.3 Bandwidth-limited data-link

In this section we examine two scenarios where we limit the data flow in different ways. The purpose is to observe how DCCP adjusts to network limitations.

5.3.4 Bandwidth constraining

In the first case, the network link only allows 20 kbytes/second. Exceeding data is delayed, not dropped. The application attempts to send data from the computer, to the mobile phone, at maximum rate.

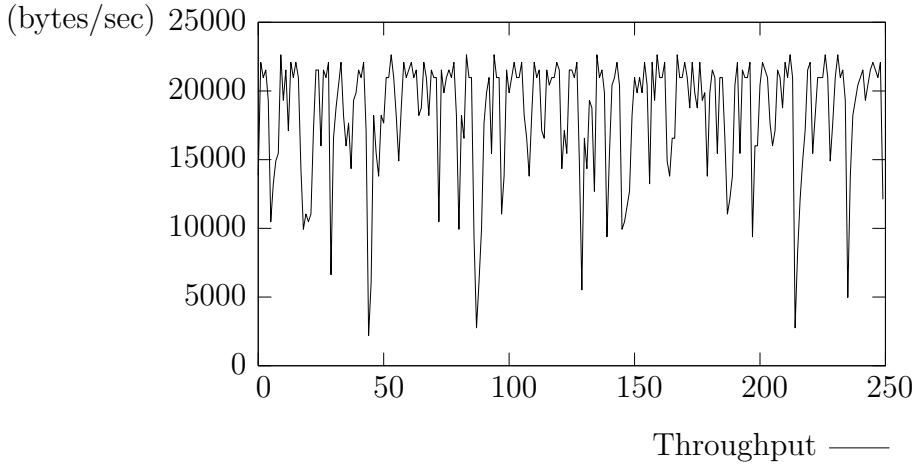

Figure 5.13: DCCP stream from computer to phone, with delay

The throughput for the DCCP stream is shown in figure 5.13. The stream is constantly pushing towards a higher rate, but can not get above 20 kbytes. The throughput is occasionally rather low.

This behavior is believed to be triggered by the fact that there is no packet loss in the transmission. When this is the case, the transfer rate depends on the receive rate reported from the receiver. Figure 5.14 shows

the throughput again, now together with the receive rate information taken from the feedback packets.

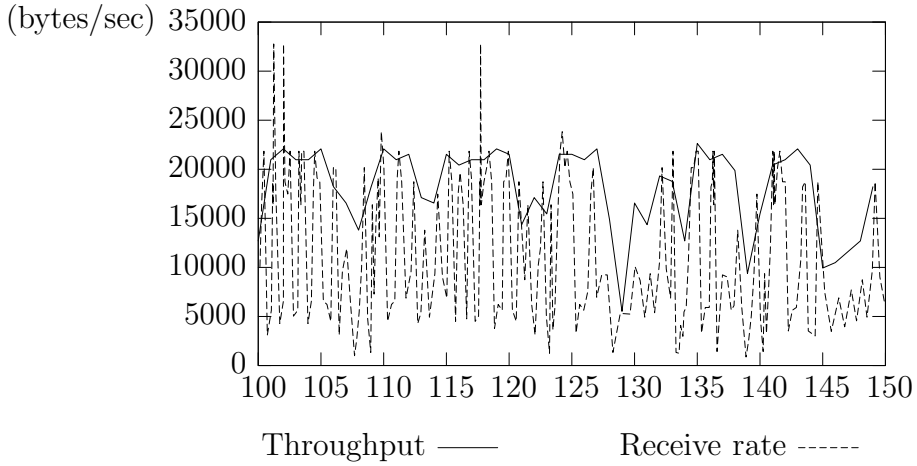

Figure 5.14: DCCP stream from computer to phone, with delay

What we can see in figure 5.14 is that the reported receive rate constantly changes between a wide range of values. The receive rate is calculated as the total number of bytes received in the latest t seconds, where t is the larger of the following; the time since the last acknowledgment packet was sent, and the estimated round-trip time. Since the receive rate is calculated on a rather short time-span there is a risk for big fluctuations. This is especially true on the mobile, where it seems that the timer occasionally dont update properly when being polled at high rate. This was noted when frequently polling the timer, two adjacent time stamps could then get the exact same time when in fact a short amount of time had passed.

While calculating the receive rate over short intervals is good for triggering fast reactions to changes in the transmission rate, calculating it over a larger time-frame could help to provide a more stable transfer rate.

5.3.5 Dropping packets

The next two scenarios illustrates the behavior of TFRC when packets are being dropped instead of being delayed. We look at two situations where packets are dropped in different ways.

The data packets in both examples consist of 512 bytes of data. The application is set to send 25 packets per second, which would make a total send rate of 12800 bytes/second.

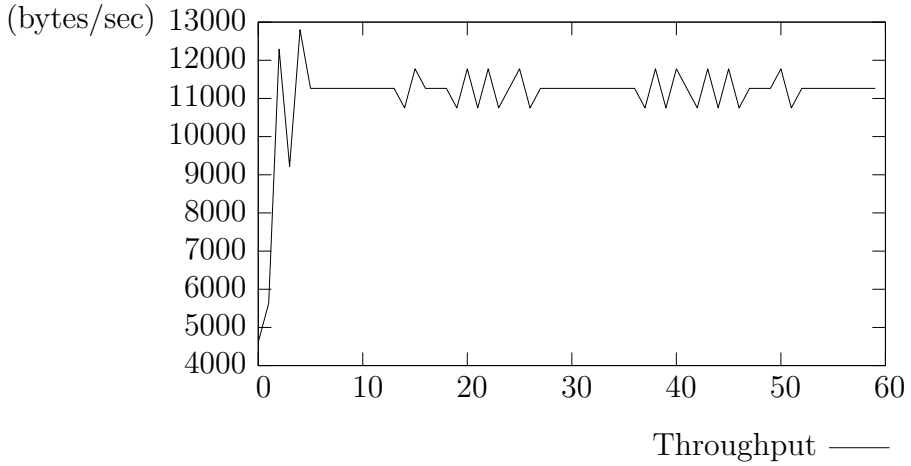

Figure 5.15: 10% symmetrical packet loss, throughput

In the first example, every 10th packet is dropped. The loss intervals will be constant, which should make the loss rate stabilize after a short time. In figure 5.15 we see that the throughput is smooth. Since there is packet loss, the sender quickly leaves the slow-start phase and equation 5.1 is then used to calculate the send rate.

$$X = \frac{S}{RTT_{avg} * f(P)} \tag{5.1}$$

X is the send rate, S is the average packet size, RTT_{avg} is the round-trip time, and $f(P)$ is looked up from a table based on P, the loss rate reported by the receiver. The round-trip time is measured to about 0.022 seconds. The low round-trip time make every loss start a new loss event. In figure 5.16 we see that the loss rate is 0.10. The loss rate 0.10 is correct, since every loss interval consist of one lost packet and nine received packets. The lookup table translates 0.10 to 0.565. The packet size is 512 bytes. Placing these numbers into the send rate equation, we get

$$X = \frac{512}{0.022 * 0.565} \tag{5.2}$$

With these numbers, the send rate X is over 40000 bytes/second. The actual sending rate will never be this high, for two reasons;

- The send rate will never be more than twice the receive rate.
- The application only wants to send 12800 bytes/second.

Examining figure 5.17, we see that the receive rate mainly is between 10000 and 15000. Thus, the throughput is controlled by the sending rate requested by the application. The reason for the different receive rates being reported is the short time-span over which it is calculated.

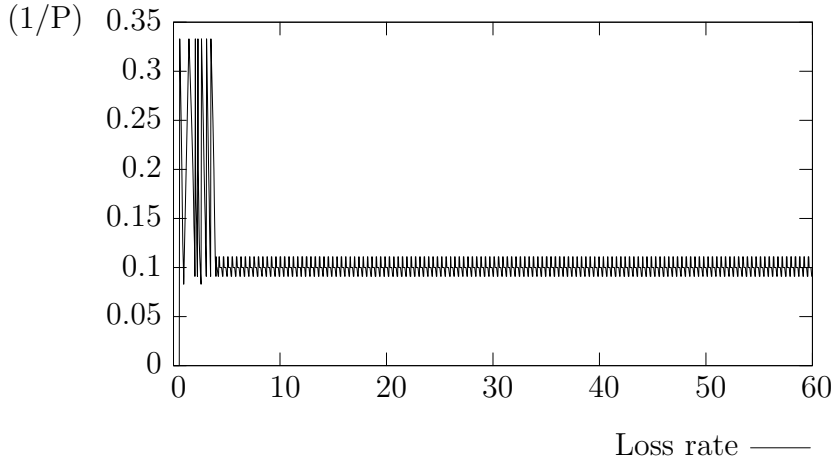

Figure 5.16: 10% symmetrical packet loss, loss rate

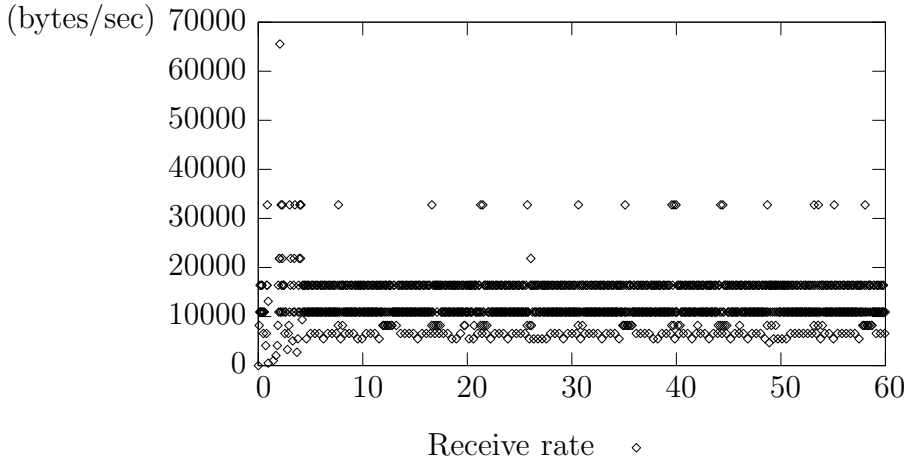

Figure 5.17: 10% symmetrical packet loss, receive rate

In the next example we introduce random packet loss. For every packet there is a random number between 0 and 99 generated, using the Linux urandom device. If this number is below the *loss* value chosen, the packet is not sent to the receiver. The *loss* value is set to 10, thus on average 10% of all data packets will be lost. The network setup is the same as in the previous experiment.

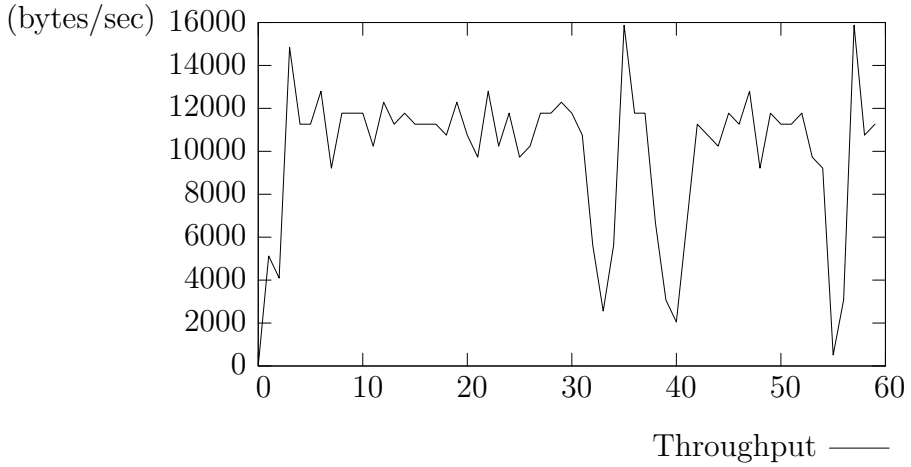

Figure 5.18: 10% random packet loss, throughput

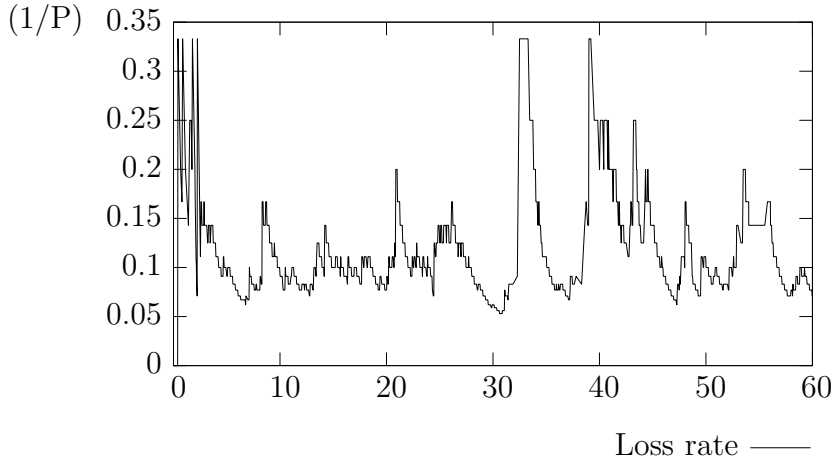

Figure 5.19: 10% random loss, loss rate

Figure 5.18 shows the observed throughput. The throughput is not as smooth as it was in 5.15 where the packet losses were evenly spaced. The loss rate is frequently changing values as seen in figure 5.19. When the loss rate peaks, at seconds 34 and 40, the throughput in 5.18 becomes very low. This is the result of the randomization.

5.3.6 Computer to phone over GPRS

Changing communication technology to GPRS, the network topology looks like this:

```
Mobile phone <--( GPRS .. Internet )--> Computer
```

The application running on the computer is sending data at maximum rate to the phone. We are interested in seeing if the send rate settles and, in that case, at which rate the throughput is smooth. We are also interested in how the receive rate changes. The receive rate value changed a lot when the data was sent over the computer network. Now when it is sent via GPRS, the RTT becomes higher. A higher RTT means that acknowledgment packets become less frequent and the time-frame used for measuring the receive rate longer. Longer time-frames can possibly make the measurements more accurate.

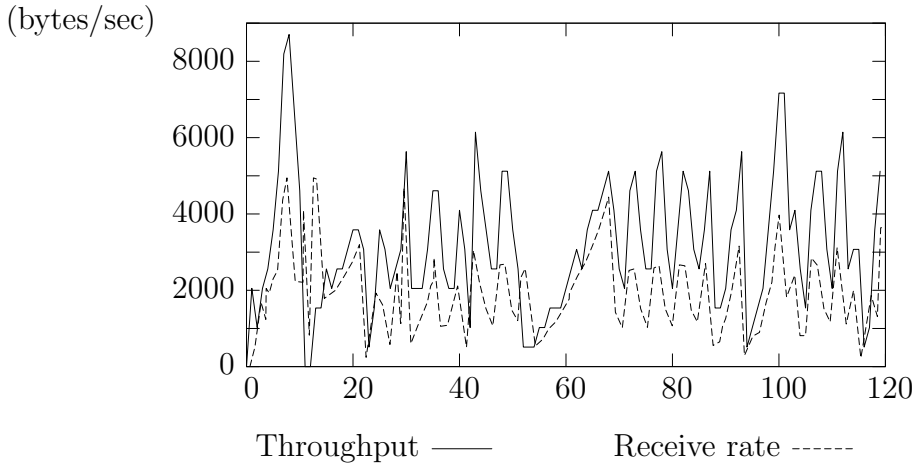

Figure 5.20: DCCP stream over GPRS

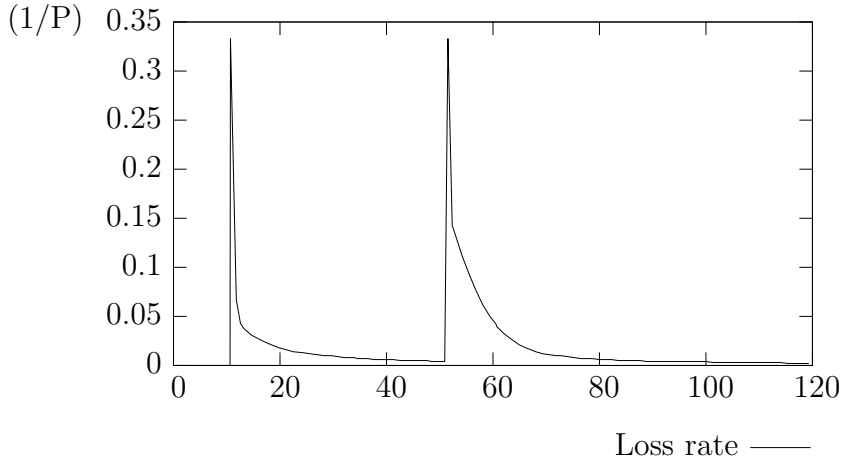

Figure 5.21: DCCP stream over GPRS, loss rate

Figure 5.20 shows the throughput and the receive rate for the DCCP stream over GPRS. The receive rate no longer fluctuates like it did during the previous tests. The measured round-trip time in this stream is in average about 0.94 seconds. Looking at figure 5.21, we see that there is a packet loss event at second 10 and at second 50. After 10 seconds, the sender leaves the slow-start phase and the TCP throughput equation comes into play. The send rate is then mainly controlled by two values;

- Loss rate - An increase in the loss rate, for example at second 50, decreases the output of the TCP throughput equation.

- Receive rate - The send rate may never be more than twice the receive rate.

Looking at figure 5.20 again, we see that the throughput is between 2000 and 5000 bytes/second, which most of the time is about twice the value of the receive rate. When packet loss occurs, the send rate is decreased in order to be friendly with other flows in the network.

5.4 VoIP

One area of application where DCCP could be useful is VoIP [2]. By using VoIP in the mobile phone, one would change the current voice service to become an integrated part of Internet. While this would have certain benefits, issues regarding the quality of the service are introduced. A phone user has come to expect good sound quality with low delays. As we have seen in earlier sections, GPRS is rather limited. In this section we explore the limitations of GPRS.

In these tests, we simulate the GSM full-rate codec traffic [19]. It operates at 13 kbit/second and provides good quality for speech. This bit-rate translates into one packet with a data-size of 33 bytes being generated every 20 ms.

5.4.1 Excessive overhead

Each data packet has to contain an IP header (20 bytes), an UDP header (8 bytes) and a DCCP header (12 bytes). This is a total of 40 bytes of data header. For proper voice traffic, we would also use RTP [20] which adds an additional 12 bytes header. Since the data size is 33 bytes, the overhead is massive. Even if we were able to implement DCCP as a proper network protocol (and get rid of the UDP part), there would still be a total header size of 44 bytes (IP + DCCP + RTP). The solution to this problem is header compression, which could reduce the total header to a few bytes [21]. This is, however, not within the scope of this Master Thesis.

5.4.2 Test setup

The application running on the computer is sending 33 bytes of data per packet, and the interval between packets is 20 ms. With the headers discussed above, the total packet size is 73 bytes. The desired throughput is thus 3650 bytes per second.

This test was designed to examine the delays one would encounter when using a VoIP application over GPRS, using the DCCP protocol. A step-by-step description of the test follows:

- The data sender stores the send time, *sent*, of the data packets sent
- The receiver stores the reception time of the data packets, $reception\_time$

[2] Voice over Internet Protocol

- When the receiver sends an acknowledgment packet, it includes the sequence number and an elapsed time for each data packet received. The elapsed time is calculated by equation 5.3.

$$elapsed = sending\_ack - reception\_time \tag{5.3}$$

  where *sending_ack* is the time the acknowledgment packet is sent.

- Upon receiving this information, the data sender uses the time when the acknowledgment came, *received*, and calculates the delay for each data packet using equation 5.4.

$$delay = \frac{received - sent - elapsed}{2} \tag{5.4}$$

  This might not give a totally accurate delay for the link in the data sending direction, as it calculates the total delay for a transmission in each direction and then divides it by 2 [3] , but it should provide an estimate of the one-way delay from the sender to the receiver.

It is generally agreed that the end-to-end delay should be less than 150 milliseconds (ms) for toll quality phone calls [22].

5.4.3 Test results

Before testing over GPRS, a few runs were made over data links. A first test between two computers connected to each other gave an end-to-end delay of about 1-1.5 milliseconds. When transmitting from a computer to the phone where the phone is connected to the Internet via the USB port of another computer, the measured delays were around 5-6 milliseconds. Such delays would not be an issue at all. More interesting is what kind of delays would be found when communicating via GPRS.

As stated above, the application wanted to send 3650 bytes per second, including headers. In figure 5.22, we can see the actual throughput on two separate occasions.

The throughput on the "bad day" is not getting close to the desired level. As a matter of fact, such results were far more common than the "good day" throughput.

It should be stated that, even on a bad day, higher throughputs were possible by sending fewer packets with much more data per packet. That

[3] It is possible that the path for the acknowledgments is slower than the path for the data packets

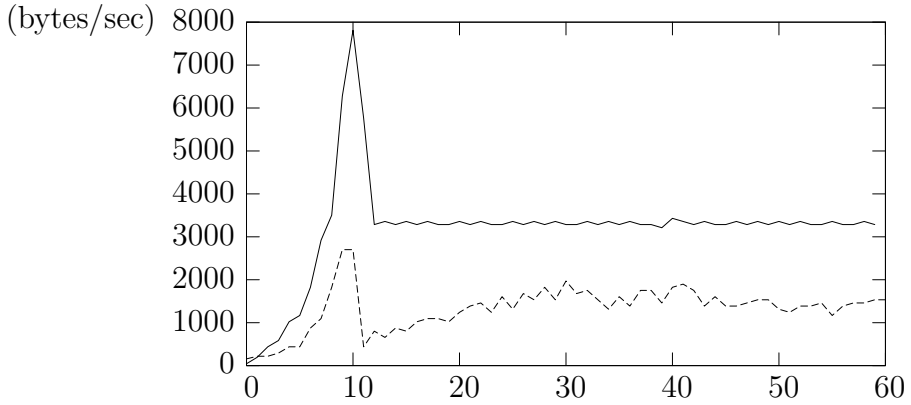

Figure 5.22: Throughput for VoIP test over GPRS

is however not an option in real-time communication. The data in a VoIP-communication is read from the microphone in real-time. If we changed the data packets to contain 10 times as much data, 330 bytes instead of 33 bytes, the packet intervals would be 10 times as long. When these 330 bytes of voice data is ready to be sent, the first bytes of data have been waiting in the send buffer for 200 ms. Such delays are not acceptable.

On the good day, the delays measured are, when the data rate has settled, at about 400 ms - 600 ms. On the bad days, it was averaging a bit lower, around 350 ms - 450 ms. These are very large numbers, way above what would be tolerable in a 2-way voice communication. The difference could possibly be explained by the difference in the amount of data transferred. To get a point of reference, a series of ICMP PING (generally known as "ping") was sent to a number of high-speed Internet hosts.

5.4.4 ICMP PING

The "ping" application, found on basically any computer system, measures the round-trip time to a specific host. For Symbian, the application newTELnet [23] is available, which includes the possibility to ping. Here are average results from pinging `www.google.com` (located in CA, USA), `www.yahoo.se` (located in England), and a host in Luleå using Bredbandsbolaget as its Internet Service Provider.

The data size is in bytes, the interval in milliseconds, and the RTT values are in seconds.

At the lowest rate, with the smallest possible data size, the average round-

Data size (b)	Interval (ms)	No. packets	Avg RTT	Max RTT	Min RTT
60	1000	20	0.877	1.078	0.671
576	1000	20	0.825	1.015	0.687
60	20	100	1.256	2.281	0.671

Table 5.1: Average ICMP PING results

trip time is almost 0.9 seconds. When approaching the rate which VoIP would work at, a packet interval of 20 milliseconds, the average round-trip time is above 1.2 seconds. These numbers correlate rather well with the delays measured in the VoIP test [4]. It is apparent that GPRS is too slow to consider VoIP, or any real-time application. An end-to-end delay of half a second results in a very inconvenient conversation.

[4]The delay measured is about the half round-trip time

Chapter 6

Discussion and further work

Since the Symbian operating system is closed source [1] , the only source of information to the inner workings of the system is the documentation provided by the Symbian corporation. When documentation is lacking, the developer can only use trial and error methods of testing. Another consequence of Symbian being closed source is that this Master Thesis could not be carried out as it initially was intended. The original idea was to implement a transport layer protocol, but the API [2] needed for that is not available to third party developers. The natural next step, if Symbian Ltd. releases the necessary API, is to make an actual protocol out of our DCCP Thin implementation.

The Datagram Congestion Control Protocol has been introduced to solve the problem with the increasing amount of Internet traffic that has no congestion control (UDP). Other suggestions include creating an unreliable variant of TCP, and adding congestion control to the Real-time Transfer Protocol (RTP). RTP is often used by media applications on top of UDP and is an application layer protocol. It would be possible to extend RTP to include end-to-end congestion control. An issue with RTP is however that some applications only want congestion control, whereas RTP has many other functions as well. DCCP provides a clean transport protocol focused on end-to-end congestion control, where the application can choose the congestion control profile that best suit its needs. An unreliable variant of TCP would work well for applications that can tolerate the send rate variations of TCP, which are considered to be rapid. However, TCP-like congestion control might not be suitable for all applications, therefore a protocol that offers several congestion control mechanisms for the application designers to choose from, has a large chance of being deployed.

We have found that the receive window was bottomed out on several

[1] The source code is not available

[2] Application Programming Interface

occasions when sending data to the phone at high speed using TCP. Although this particular phone is only aimed at GPRS traffic, situations might arise when the receiver could provide information to help improve the quality of a session. Presently the receiver can slow down the sender either by decreasing the value of the receive rate reported, or by reporting packet loss for packets that actually were not lost. Both methods have flaws. If the receiver has a high CPU load and is unable to cope with the high sending rate, lowering the receive rate will decrease the sending rate but not necessarily the work load. The sending application might decide to change the encoding of the data being sent, to a codec that requires more computing power at the receiver but allowing fewer packets to carry the same information. A solution to this problem already exist in the DCCP main draft in the form of two options, Slow Receiver and Data Dropped, but these options are not included in the DCCP-Thin profile. We suggest adding the Slow Receiver and Data Dropped DCCP options to the CCID 3-Thin specification, which would enable the sender to make an informed decision about the encoding and send rate to be used.

Another possible improvement is to allow the calculation of the receive rate over a longer period of time. As was shown in 5.3.3 on page 25, the receive rate was not smooth when the round-trip time was low. A more stable receive rate should be received if it was calculated over an extended interval. This is not a problem when communicating over GPRS, since it has very high delays, but could be an issue when faster communications techniques are used. A drawback with increasing the time interval over which the receive rate is computed, is a slower reaction to sudden network changes. This should however not be a problem, as long as the time range is not excessively long, but further studies are needed to determine a suitable duration.

Chapter 7

Conclusion

As available data rates for mobile phones become higher, the interest for VoIP and similar application areas increase. Neither of the two widely deployed transport layer protocols in the Internet today is suitable for this kind of traffic. TCP prioritizes reliable delivery before speed. UDP does not have any congestion control. What is needed is a transport protocol that focuses on speed of delivery but also has congestion control. Whether DCCP is the best new protocol to fill this void has been and will continue to be discussed.

Results presented in this Master Thesis shows that a DCCP-Thin implementation utilizes the available bandwidth as well as TCP. Two DCCP flows share the bandwidth fairly. Upon packet loss, the send rate was reduced of concern to the congestion level in the network. Different packet loss distributions creates different throughputs, as expected. When losses are randomly distributed, the loss rate changes over time, while symmetrical packet loss causes the loss rate to stabilize and thus a smoother send rate.

The experiments also pointed out some of the limitations of the GPRS technology. GPRS provides a high-delay service where sudden delay spikes are not uncommon. The DCCP option Slow Receiver is currently not part of of the CCID 3- Thin profile. Test results suggest that including the Data Dropped and Slow Receiver options could be useful, since they would enable the data sender to distinguish between network congestion and receiver problems, such as low buffer space or an overloaded CPU.

DCCP is a new transport protocol proposal with focus on end-to-end congestion control. Since it provides several congestion control profiles, it can be useful to a wider range of applications. DCCP is one of the solutions suggested to solve the problem of an increased amount of unregulated traffic traversing the Internet. In comparison to other proposals, DCCP seems the most flexible. Whether DCCP will succeed in becoming a standardized and accepted network protocol depends on its ability to satisfy the need for con-

gestion control sought-after by the network, while at the same time providing a reasonably good service to its target applications. Determining if DCCP lives up to the expectations will require extensive testing and evaluation. The results presented here are a step on the way.

Bibliography

[1] J. Postel editor. Transmission control protocol (tcp). RFC793, Sep 1981.

[2] E. Kohler, M. Handley, and S. Floyd. Datagram congestion control protocol (dccp). draft-ietf-dccp-spec-06.txt, Feb 2004. Work in progress.

[3] S. Floyd and E. Kohler. Profile for dccp congestion control id 2: Tcp-like congestion control. draft-ietf-dccp-ccid2-05.txt, Feb 2004. Work in progress.

[4] S. Floyd, E. Kohler, and J. Padhye. Profile for dccp congestion control id 3: Tfrc congestion control. draft-ietf-dccp-ccid3-05.txt, Feb 2004. Work in progress.

[5] E. Kohler. Dccp ccid 3-thin. draft-ietf-dccp-ccid3-thin-00.txt, Oct 2003. Work in progress.

[6] Symbian Ltd. Symbian. `http://www.symbian.com`.

[7] `http://www.cellular.co.za/stats/stats-main.htm`, 2004.

[8] GSM World. Intro to 3g. `http://www.gsmworld.com/technology/3g/intro.shtml`, 2004.

[9] GSM World. Gprs class types. `http://www.gsmworld.com/technology/gprs/class.shtml`, 2004.

[10] Symbian. Symbian os c++ coding standards. `http://www.symbian.com/developer/techlib/papers/coding_stds/2003-01_Sy0%SCodStn.pdf`, 2004.

[11] Richard Harrison, editor. *Symbian OS C++ for Mobile Phones*. Wiley, 2003. ISBN 0-470-85611-4.

[12] `http://www.ietf.org`, 2004.

[13] M. Allman and V. Paxson. Tcp congestion control. RFC2581, Apr 1999.

[14] M. Handley, S. Floyd, J. Padhye, and J. Widmer. Tcp friendly rate control (tfrc): Protocol specification. RFC3448, Jan 2003.

[15] Dccp implementation to freebsd. `http://www.freebsd.dccp.org/`, 2004.

[16] Telia ab. `http://www.telia.se/`, 2004.

[17] The linux kernel archives. `http://www.kernel.org/`, 2004.

[18] Perl, programming language. `http://www.perl.com/`, 2004.

[19] `http://www-mobile.ecs.soton.ac.uk/speech_codecs/standards/gsm.html`, 2004.

[20] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Real-time transport protocol. RFC3550, Jul 2003.

[21] IETF. Robust header compression. RFC3095, Jul 2001.

[22] H. Zheng, G. Rittenhouse, and M. Recchione. The performance of voice over ip over 3g downlink shared packer channels under different delay budgets. In *Vehicular Technology Conference*, volume 4, pages 2501 – 2505. IEEE, Oct 2003.

[23] `http://www.newnet-marketing.de`, 2004.