Department of Computer Science

Torbjörn Andersson

Single SCTP Association with Multiple Streams vs. Multiple TCP Connections: A Performance Evaluation

Master's Thesis

2005:08

Single SCTP Association with Multiple Streams vs. Multiple TCP Connections: A Performance Evaluation

Torbjörn Andersson

This thesis is submitted in partial fulfillment of the requirements for the Masters degree in Computer Science. All material in this thesis which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Torbjörn Andersson

Approved, 2005-06-07

Opponent: Eivind Nordby

Advisor: Johan Garcia

Examiner: Donald Ross

Abstract

The *Stream Control Transfer Protocol* (SCTP) is a new transport protocol designed to solve some of the shortcomings of the *Transmission Control Protocol* (TCP). SCTP has a new internal structure, which allows a connection, called association in SCTP, to contain multiple streams. These streams give users the opportunity to separate unrelated data into different streams and avoid *Head of Line Blocking* (HLB) situations, which can occur when unrelated data items are transferred in a single stream.

The goals of this study were to gain practical experience of SCTP and reveal information about the performance of the protocol. The means for reaching these goals were therefore naturally practical experiments on a real SCTP implementation with the necessary performance measurements.

Initialy, an evaluation of SCTP implementations was performed and the most appropriate for a performance evaluation was chosen. The experiment was designed to compare the performance of multiple streams against multiple TCP connections. The downloading of web-pages serving as an inspiration and the composition of the web-pages varied from 1 to 300 different files, each file transferred separately over a SCTP stream or a TCP connection. The result showed that SCTP can be 10% faster when 300KB are transferred with 50 streams/connections and under some circumstances up to 70% faster. The overhead generated by initializing multiple TCP connections was found partially responsible for TCP's lack of performance. Further, the way the TCP connections were initialized was found to have a big effect on TCP's results in the tests.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet community has been conservative when it comes to introducing new general transport protocols. Why it has been like this probably have many reasons, but one reason may be that new protocols have to interact with existing protocols in a friendly manner and not damage the functionality of Internet. The spreading of a new general transport protocol might also be hindered by the operating systems. General transport protocols work on top on the *Internet Protocol* (IP) and the access to the IP layer is often restricted by the operating system. General transport protocols are therefore often required to be implemented in the kernel of the operating system, which can be troublesome in some operating systems. Specialized transport protocols have in general avoided the IP layer and instead have they been working on top of *User Datagram Protocol* (UDP), which is a general transport protocol that offers a minimal transport service. UDP acts as a thin layer on top on IP and the access to the UDP layer is normally not restricted and the spreading of a specialized protocol is therefore not hindered by the operating system.

The *Transmission Control Protocol* (TCP), unlike UDP, offers a much more complex transport service, which in many cases is adequate for most applications. Now when the applications of the Internet evolve new transport services are needed, due to new demands from the applications. The transport service of TCP is in some cases too complex, expensive

or in some cases inadequate. SCTP was developed to meet some of these demands.

SCTP, which was in the beginning a specialized signalling transport protocol, was also at first designed to work on top on UDP, but SCTP evolved further. As SCTP evolved *Internet Engineering Task Force* (IETF) changed their mind and moved SCTP from working over UDP to instead work directly on top the IP layer, because that they saw SCTP as a new general transport protocol. The move was made and now SCTP has to prove itself as an important general transport protocol and persuade all operating system developers to add SCTP to their kernel.

The performance of SCTP is naturally a key factor to SCTP's future as a general transport protocol. This master thesis documents the performance study on SCTP done by the author during the year 2001. They study was performed at Karlstad University in close relation with the *Distributed System Communication* (DISCO) research group.

1.1 Aim of study

SCTP was at the start of this study a fairly new protocol and the DISCO group in Karlstad (Sweden) had no pratical experience of it at the time. The overall goal for this study was therefore to gain practical experience of SCTP and do a transmission performance comparison between SCTP and TCP. The natural input to the comparison was therefore results from experiments performed in a non simulated environment. These goals can be divided into a couple of specific goals. All goals are enumerated and listed below, where each goal is identified with the prefix "Goal" followed by a number.

Goal 1:

This goal involved finding the most suitable feature for the performance study. SCTP has several new transport features that a user can apply. The stream functionality of SCTP had been identified, in an undocumented pre-study, as interesting in a performance

perspective. However, other features might also have a positive effect on the performance and those were also to be regarded for this study.

Goal 2:

Design an experiment that tests the chosen SCTP feature from goal 1. The outputs from the experiment should reveal information about what kind of impact the chosen feature has on the performance in several different situations. Both different protocol usages and network characteristics should be tested so that broader conclusions can be made, where the network dependent and independent effects can be identified.

This goal demands that a test application that uses the chosen SCTP feature from goal 1 needs to be implemented. A logically identical application that uses TCP instead of SCTP must also be implemented. The TCP application must use TCP in a way which enables the application to offer the same transport service as the SCTP version. Further, the execution of the experiment requires a test environment, which contains a network that allows the network characteristics to be changed. The test environment must also support both the SCTP- and the TCP protocol.

1.2 Scope

Both SCTP and TCP are big protocols and there are many differences between the two. All of them can be evaluated but not within a single master thesis and therefore some limitations have been defined for this project. The limitations are listed below.

- The flow control of SCTP and TCP are not studied in this thesis.
- How SCTP interacts with other protocols on the same network, i.e. the fairness of SCTP is not studied in this thesis.

- The added features for redundancy/reliability implemented in SCTP are not studied in this thesis.

1.3 Document disposition

Chapter 1 describes the background, aim and limitations for this work.

Chapter 2 begins by describing the basic concepts of Internet communication. With the basic concepts as a foundation the chapter continues with the details of TCP and SCTP. The amount of details is balanced so that the reader can understand the differences between TCP and SCTP and later also understand the analysis of the results from the experiment.

Chapter 3 describes the different implementations of SCTP that were available when this study was performed. The chapter then continues with the motivations for why a particular implementation was chosen for the experiment.

Chapter 4 describes the different experiment possibilities that were identified for this thesis. It then follows with the motivation for why one of the possibilities was chosen and how that particular experiment was designed. The chapter describes all the scenarios that are to be executed in the experiment, defines network configurations for the experiment, describes details on how the test applications work and finally defines the test environment.

Chapter 5 describes the results from the experiment, how the results were checked for correctness, how the results were analyzed and the summary of the analysis. The chapter ends with the conclusions that can be drawn from the experiment results.

Chapter 6 is the summary of this thesis. The different goals of this study are compared with the results. The chapter finishes with the conclusions and thoughts from the author.

Finally Appendix A contains two tables containing descriptions for abbreviations and concepts and the reader should remember to look in Appendix A when the meaning of a certain abbreviation or concept is forgotten.

Chapter 2

A TCP and SCTP summary

This chapter is the result of the theoretical study on TCP and SCTP. This chapter first describes the basic concepts of Internet communications, focusing on TCP. The chapter then moves from TCP to SCTP and its description. The SCTP section describes SCTP and at the same time compares SCTP with TCP so that the reader more easily can identify the differences that were found during the study. The description of basic concepts, TCP and SCTP should give the reader the necessary knowledge to understand the following chapters.

2.1 Basic concepts

A *protocol* defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event [11]. A protocol can be designed in many ways and different protocols often address different problems. Some protocols address the problems of several interconnected computers and some only work between two computers. The service of one protocol can be a real-time signalling service and another protocol may want to give a very reliable, but slow, data transport service. The service SCTP offers is giving the user

a communication channel between two endpoints on Internet with some properties, very much similar to TCP. The endpoints are referred as peers to each other.

A major complexity of the Internet is that when two peers communicate the communication often involves additional computers or network devices that only forward the messages between the communicating peers. The computers on Internet are also interconnected in a more or less dense net and therefore exists it a large number of possible chains of computers that a message can follow to reach its peer. Many things can go wrong and we need good protocols that handle this complexity.

Often a single protocol is not enough to deliver data over Internet and therefore several protocols cooperate to be able to give a certain transport service. The protocols create a stack where the protocols are put on top of each other, creating a layered structure. Each protocol gives at least one service that the protocol above makes use of. The protocol stack used together with the Internet network is often referred to as TCP/IP, where TCP is the transport protocol that works on top of IP. Figure 2.1 visualizes a possible TCP/IP stack. The dotted lines in the figure show the conceptual flow of data and the solid lines show how the data actually flow through the stacks. The bottom layer is not named here but it can be seen as the hardware associated layers that handle a physical medium such as radio, Ethernet and other. The application layer located above TCP in the figure is also referred as the *Upper Layer Process* (ULP) for TCP.

The IP layer responsibility is to make sure that the packets end up at the correct computer on Internet and its service is classified as "best effort", meaning that packets can be lost and delivered out of order. TCP offers an application to application transport service and it does not address the problems arising from Internet-communication involving several computers. Instead it focuses on making sure that the data are delivered to the user in the same order they were sent. When the IP layer fails to deliver a packet TCP will automatically stand in and retransmit the packet by instructing the IP layer to transmit a new packet with the same contents that was lost. Except handling retransmission and

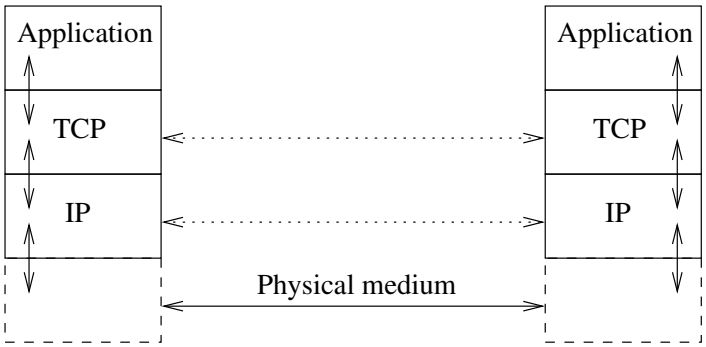


Figure 2.1: A TCP/IP stack

handling packets out of order TCP also has to ensure that the communication honours the flow, software related, and congestion, network related, constraints that ensure that the network is in a stable state without too much congestion.

The IP layer is a simple protocol and its service is well defined and is not an issue for optimizing. It was designed to address the problem of making sure that the packets are able to propagate to the right computer and not so much more and therefore it does not limit the upper layers. If the IP layer had offered a more complex and costly service then some ULPs may find this service unwanted and hindering in aspects of performance. Instead the transport protocols that work on top of IP are much more complex and they are under active development and SCTP is a result of this.

2.2 TCP

TCP has existed and been used from the very beginning of the Internet and is also today a very commonly used transport layer protocol. Many of the big Internet applications such as surfing the Web, communication with email and more use TCP's transport service. This part will describe the functionality and behaviour of TCP. The understanding of TCP is needed when comparing TCP with SCTP and when analyzing the outcome of the experiment, as described in Chapter 4.

2.2.1 General properties

This section describes the general properties of TCP that are relevant in this study. Such properties are connection sequence, data abstraction and congestion control. The bullet list below describes these properties shortly and then the next three sections discuss each separately.

- TCP is a connection oriented protocol meaning that a connection must be established before any data can be sent.
- A TCP connection is bidirectional, data can be sent by both peers using a single connection. Further, TCP has a byte stream abstraction, which means that TCP does not help the ULP to identify the start and the end of each ULP message. Finally the delivery of the data at the receiving side of the connection is always done in the same order they were sent.
- TCP has a congestion control functionality which monitors the transmission and tries to adapt to the networks capacity. The retransmission of lost packets is tightly coupled with the congestion control.

2.2.2 TCP connection sequence

The connection sequence is an interesting performance aspect, especially when small amount of data is to be sent over the new connection. This section therefore describes TCP's connection sequence. TCP utilizes a three-way-handshake to establish a connection between two peers and Figure 2.2 illustrates this handshake. First host A sends a SYN packet to its peer host B, which receives the SYN packet and answers host A with a SYN-ACK packet. Host B also allocate the necessary resources for the connection. When the SYN-ACK packet arrive at host A believes that the connection is established and after it has sent its ACK packet to the peer it can immediately start sending data. The connection is completely established when the ACK packet from host A arrives.

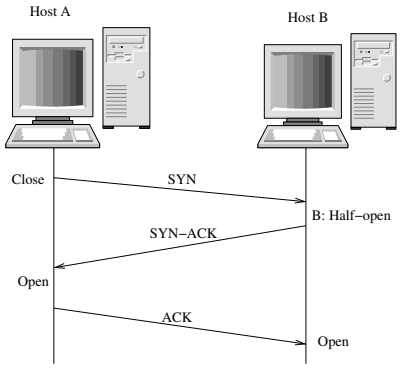


Figure 2.2: TCP Connection sequence

This three-way handshake is fairly fast but has at least one drawback. It is vulnerable for hostile attacks because when host B receives a SYN from a peer it must allocate resources for the connection, without verifying the existence of host A. Therefore an attacker may forge SYN packets with a fake source address and send SYN packets at a rate that is faster than the time-out time for the ACK response packet. At the timeout the allocated resources are freed for new connections, but eventually can an intensive stream of fake SYN packets empty the resources of host B, which no longer can handle new connections. Already existing connections may also be affected. This attack is called SYN flooding[5].

2.2.3 Data abstraction

TCP transfers data as a stream of bytes and it is limited to one single stream. If an application wants to transfer multiple data items with a single TCP connection, then the application has to merge them into the single data stream and separated the individual items at the receiver. TCPs single stream is not necessary always bad for the application, but for these kinds of applications there is one drawback. The *Head of Line Blocking* (HLB) phenomenon occurs when individual TCP packets experience transfer difficulties and are lost or delayed. The data that was lost or delayed can naturally not be delivered to its application. Further, the data that arrive after the time the lost or delayed data

arrives are indirectly affected by the loss and cannot be delivered to the application. This happens because TCP always delivers its data in the order it was sent. The data that are indirectly affected are experiencing head of line blocking.

TCPs byte stream orientation has also some implications to the behaviur of TCP. TCP does, in order to send data with as little over-head as possible, apply some rules before it sends out new data. What TCP does is that it waits for some more data from the application if it cannot create a TCP packet of the maximum size. The maximum time TCP waits for more data from the application is controlled by a few rules. This is not a problem for ordinary file or "bulk" transfers because these applications has no problem filling a TCP-packet when TCP is ready to send out a new packet. However, telephony signalling applications are an example of applications that dislike this kind of delay. These applications tend to send many small data items and are not always able fill up a TCP packet.

2.2.4 Congestion control

Congestion occurs when the buffers in a router somewhere in the network is full and has to throw packets due to traffic overload. The congestion control in TCP has a number of algorithms that help to avoid congestion. The following descriptions are short and will only give a brief summary of the algorithms. Knowledge of the algorithms is necessary to understand why TCP performs as it does. The descriptions are supported by Figure 2.3.

- The *Slow-start* algorithm is used as the initial algorithm, which probes the network path between two peers to find the currently available bandwidth. This algorithm starts with assuming that the network can handle one or two *Maximum Transfer Unit* (MTU) and the variable *Congestion WiNDow* (cwnd) is set to this value. In Figure 2.3 the cwnd starts with the value of 2*MTU. The value of the cwnd limits the amount of out-standing data the connection is allowed to have. Each segment acknowledgement received during slow-start increases cwnd by one MTU, which results

in that each acknowledged burst of segments will double the cwnd. The cwnd will exponentially increase until it reaches the *Slow Start threshold* (ssthresh), predefined at the start by the system. From ssthresh and up cwnd is increased only by one MTU per *Round Trip Time* (RTT), according to the *Congestion avoidance* algorithm[11].

- The *Congestion control* algorithm is used when the retransmission timer expires. This algorithm will decrease ssthresh to half of the current cwnd and the cwnd is set to two MTU. The slow-start is then restarted with the new values on ssthresh and cwnd[15]. In Figure 2.3 the retransmission timer expires early in the illustrated transfer and the values on ssthresh and cwnd can be observed after and before the timeout.

- The Fast retransmit algorithm is applied when the receiver has sent 3 consecutive duplicate ACKs. This algorithm will immediately retransmit the lost segment, indicated by ACKs, and this will shorten the time before the retransmission is done, which otherwise would have been done when the retransmission timer expires[15]. Figure 2.3 shows and points out two occurrences of Fast retransmit.

- The Fast recovery algorithm is applied after that a Fast retransmit has been performed. The sstresh is reset to half its value and cwnd is set to sstresh. Then the sender will then with new values on the cwnd and sstresh continue transmit data following the rules of the congestion avoidance rules[15]. Figure 2.3 show two occurrences of Fast retransmit and has therefore also two of Fast recovery.

- The *Selective ACKnowledgment* (SACK) algorithm is used to help the sender to get knowledge about lost packets. Without SACK enabled the sender will only be notified about one lost packet per RTT. This is because the receiver only sends ACKs with the sequence number of the first missing segment in the buffer. This limits TCP to only report one gap per RTT. Now SACK allows the receiver to specify for the sender exactly which segments it has received. The SACK information sent from the receiver

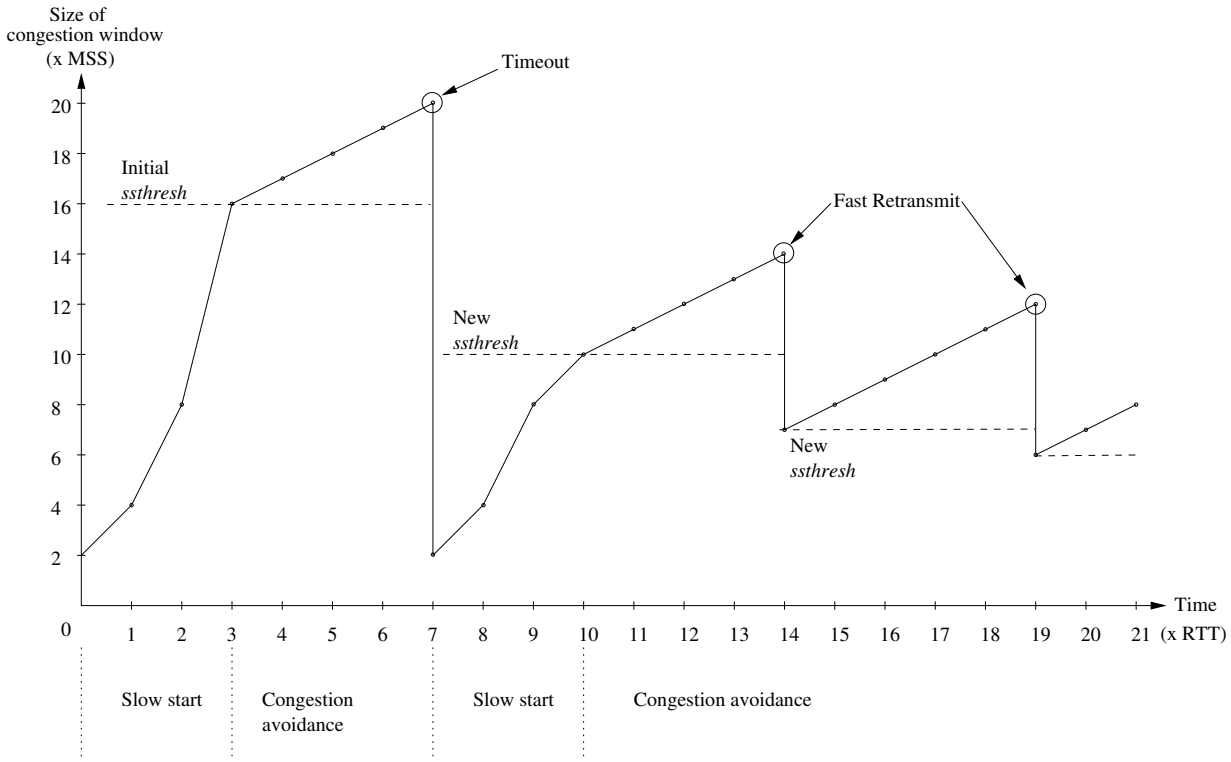


Figure 2.3: Congestion control[8]

to the sender is appended as a TCP option. However, TCP allows only 40 bytes for TCP options. This limits the SACK algorithm to SACK 4 blocks of received data, because the SACK option allocates 8*n+2 bytes, where n is the amount of SACK blocks. In the reality TCP often has other TCP option that it wants to send and therefore is the limit often less than 4 SACKs[12].

2.3 SCTP

SCTP is a connection oriented protocol that offers transport services similar to TCP. But SCTP has extended the concept of a connection between two computers to include a stream concept. SCTPs connection is called an association and can contain from 1 up to 65535

streams. All user data that is delivered over the association must be assigned to a stream.

So far things may seem similar to TCP but the interaction between the streams in SCTP makes on big difference. The streams deliver data independently of each other. A transmission error only affects the streams that lost data and the unaffected streams do not need to wait for resynchronization. The data that is received on the unaffected streams can therefore be deliver data to the ULP. By using several streams one can avoid the HLB issue of TCP.

This section has the same structure as the TCP section but it is extended with a description of the socket API that has been suggested for SCTP. Further SCTP has additional features that TCP does not have and these are described in the last subsection. To point out differences and similarities the descriptions referrers back to TCP when possible.

2.3.1 General properties

This section presents some properties of SCTP that have been found relevant when comparing with TCP. Item 1, 4 and 5 in the bullet list are each followed up with a separate section where they are described further. Item 2 is not described in detail since it is outside the scope for this study. The streams of SCTP are described in item 3 and are not described further.

- SCTP is as TCP a connection oriented protocol meaning that an association must be established before any data can be sent.

- SCTP is multi-homed meaning that an association can involve several IP addresses at each end; in contrast to TCP that only has one IP address at each end. The multiple IP addresses at each end make it possible for the packets to be transferred in several different paths. If several paths exist, then one path, the primary path, will be used for transfer and the others are redundant and are available for retransmissions and path failure situations.

- Within the association logical streams exist and a stream is uni-directional, meaning that data can only be transported in one direction. Streams are therefore specified in both directions and a SCTP endpoint can specify the amount of streams that it want to receive on, which implies that the endpoints can have different amount of streams ready to receive on. A SCTP association must have at least one stream in each direction.

- SCTP's data transfer is message oriented, as opposed to the byte stream oriented TCP. This means that the ULP can send messages with SCTP without having to neither merge them nor separate them. This is also known as conservation of message boundaries[23]. In general all ULP messages are assigned to a stream, which ensures ordered delivery within the stream. However SCTP allows ULP messages to be transmitted without being assigned to a stream. These messages are allowed to delivered to the ULP as soon as they arrive, which may result in an unordered delivery.

- SCTP has as TCP a congestion control functionality which monitors the transmission and tries to adapt to the networks capacity. SCTP's congestion control is made as similar as possible to TCP but there are differences.

2.3.2 The SCTP connection sequence

SCTP uses a 4-way handshake to initiate a connection with a peer. The motivation for the longer handshake, which includes a cookie mechanism, is to protect a server from SYN flooding, see Section 2.2.2. Figure 2.4 shows the sequence of packets passed between the two peers during the SCTP connection phase. First host A sends an INIT packet to host B. Host A answers with an INIT-ACK packet that includes a cookie, which is generated for each received INIT packet. This cookie contains all information needed by host B to setup this connection, but by sending this information over the network host B does not need to

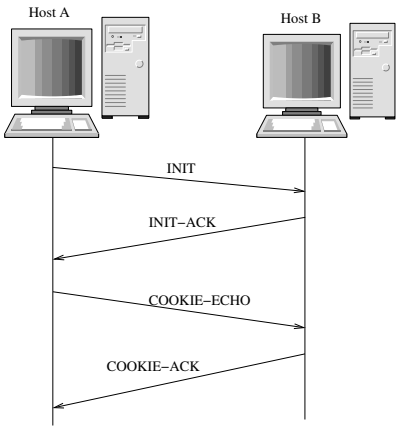


Figure 2.4: SCTP Connection sequence

store the information locally. Instead host A is supposed to send back this cookie in the COOKIE-ECHO packet. This ensures the existence of host A, which might not exist, and also allows host B to avoid allocating resources before host A has been verified. To ensure that the cookie is not modified by host A the cookie contains a *Message Authentication Code* (MAC). This MAC is generated with a secret key and the same key is used again upon the reception of the Cookie-ECHO packet. If host B receives a valid cookie from host A the association is seen as complete and a COOKIE-ACK is sent to host A, which regards the association complete when the COOKIE-ACK packet arrives.

This handshake procedure is longer than TCP's and might be too long for some applications. SCTP therefore allows data to be appended to the COOKIE-ECHO and COOKE-ACK packets, which is called piggybacking. This makes the number of packets to be sent before the data transfer can begin the same as in TCP. When comparing TCP's and SCTP's connection establishment in terms of performance, the only difference is the few extra bytes that are sent over the network.

2.3.3 Data abstraction

SCTP utilizes the concept of a message, which has a user defined size. These messages are built by the application and sent down to the SCTP layer, and they do not need to contain any data for explicitly defining the message boundaries. SCTP inserts the messages into one or more *chunks*, which is a unit used to transfer data. A chunk has its own header that describes its contents. There are several different chunk types in SCTP but only one chunk is used for user messages. Its header contains information about the classification of the data inside, the message length, a stream number, stream sequence number and flags. The flags are used to specify the chunk's relationship to other chunks, if the message should be delivered unordered and more.

SCTP can, to improve efficiency, bundle as many chunks as it wants into a SCTP packet, as long as the packet size does not exceed the MTU of the used path. Further can SCTP, as TCP, delay the transmission of a SCTP packet and wait for more chunks. In this way it can increase the amount of bundling.

2.3.4 Congestion control

The streams of an association share flow control, which operates at the association level. The congestion control of SCTP operates on a per path basis, because each path can be a different network with a different bandwidth congestion state. The design of SCTP's congestion control originates from TCP and the two have much in common. A SCTP association is designed to be equally aggressive when competing for bandwidth as a TCP-connection and this makes it possible to have TCP and SCTP connections on the same network without unfair competition for the shared network resources. Still there are some differences between the two protocols. The following items below are retrieved from [24] if no other reference is given.

- The initial cwnd is suggested to be 2*MTU in SCTP, which is usually one in TCP.

- In SCTP, the increase of the cwnd is controlled by the number of acknowledged bytes; while in TCP, it is done on a per packet basis where each received new acknowledgment increase cwnd with one MTU.

- The cwnd is a limit that SCTP is allowed to exceed with up to MTU-1 bytes, if the outstanding amount of data is less than cwnd before the transfer of a segment[17]. One says that SCTP "slops over" the cwnd limit.

- SCTP is required to be in slow start phase when the ssthresh is equal to the cwnd. For TCP is it optional to be either in the slow start phase or in the congestion avoidance phase when the ssthresh is equal to the cwnd.

- In SCTP; the SACK feature is not limited. This allows SCTP to send large amount of SACK information to the peer[20]. While in TCP, the SACK feature is limited by the small space that the SACK header has in the option header section in a TCP packet.

- In SCTP, Fast retransmit is triggered by the fourth SACK, reporting a missing chunk. While in TCP, Fast retransmit is triggered by the third missing report of a packet[17].

- SCTP has no explicit fast recovery algorithm like the one used in TCP. In SCTP, the parameter Max.Burst is used after the fast retransmit to avoid flooding the network. Max.Burst limits the number of SCTP packets that may be sent after processing the SACK, which acknowledges the data chunk that has been fast retransmitted.

2.3.5 SCTP socket API

Together with SCTP a socket API extension has been specified for the existing socket API used with TCP and UDP. This section gives a short summery of the draft[22], available at IETF home page[1], that define this extension.

[1] http://www.ietf.org

This extension allows users to easily interact with the SCTP stack and make use of all the new features. There are two API styles available, a TCP-style and a UDP-style design. The TCP-style has the purpose to allow applications to quickly change from using TCP to instead use SCTP. This style has a complete set of functions that one can use to access all features of SCTP, and one is also able to use the old primitives from the TCP API. The only mayor restriction of the TCP-style is that one can only have one association per socket, a default association.

The UDP-style gives the user the possibility to have multiple associations per socket. This socket style only allows calls to the sendmsg() and sendto() primitives but not to the send() primitive, which depend on the existence of a default association in the socket and such a thing does not exist in the UDP style socket. The sendmsg() and sendto() primitives will open a new association, when needed, and also reuse an already opened association. A user may branch off one association and create a new socket containing the selected association. The socket will be a TCP style socket which one can use the send() primitive on.

Common for both styles is that only sendmsg() allows the user to bundle ancillary data to the calls. These data tells the SCTP layer how the message is to be transferred, for instance which stream is to be used[23]. The send() and sendto() primitives use the socket default settings when sending data and therefore will data be sent by default on stream '0', depending on socket implementation.

The primitives for reception works in the same way, where recvmsg() is the only primitive that can give the user information about the way the message was transported, such as stream information and more. Messages sent with sendmsg() can be received with recvfrom() and recv() on the receiving side but with the possibility that data may seem to arrive out of order etc.

2.3.6 Additional features of SCTP

Here follows a list with additional interesting features that SCTP have, but they are regarded as out side the focus of this study. Some are also only drafts and do not exist in the current IETF proposed standard.

- Unordered message delivery: This allows the user to send an urgent message that will be delivered as soon as it arrives at the receiver. The receiver can identify urgent messages and separate them from the main flow of messages.
- Partial reliable transport[21]: Each stream of an association can be configured differently and it is possible to configure one or more streams for partial reliable transport. This feature basically allows SCTP to abort the transfer of a certain message if it has been delayed for too long, due to congestion, network losses etc.
- Add and remove paths[18]: During the initialization of an association several different paths can be specified, where a path is defined by an IP address at the receiver. This draft specifies how new paths can be added to or how existing ones can be removed from the association, after the initialization phase.
- The heartbeat functionality probes the different paths of an association during times of inactivity to check if they are valid for data transfer.

Chapter 3

An evaluation of SCTP implementations

Goal 2 of this study, defined in Section 1.1, includes building an experiment environment where both TCP and SCTP are supported. TCP has support in most operating systems but SCTP is not yet included in any operating system known by the author. But there are "stand alone" implementations that are operating system independent. This chapter compares two different implementations of SCTP that where found on Internet. The two are compared focusing on the support for SCTP features, usability and documentation. A SCTP implementation that supports a big set of SCTP features and is easy to use is preferred. The result from the comparison is used to choose one for the experiment of this study.

3.1 The competitors

The candidates were selected at the start of this work and the initial requirement was that the candidates' project was active. The first candidate is distributed and developed by Randall R. Stewart, who is one of the main actors in the SCTP field. His implementation is referred to as a reference implementation of SCTP. The source code of his implementation can be downloaded from his website[19]. The second candidate is a prototype of SCTP that

is developed by Siemens in cooperation with the Computer Networking Technology Group of the University of Essen. The source code of the implementation can be downloaded from their website[2].

3.2 Implementation basics

This section explains how the two candidates have implemented the SCTP layer. It does not compare the two candidates; instead this section gives an insight on the implementation techniques and concepts that the two candidates applied. The first subsection introduces important implementation techniques from both candidates. They were identified during the study of the implementations and are regarded as important when electing one implementation for the upcoming experiment. The second subsection describes why only one SCTP layer can be active on a host. This restriction has also affected how the SCTP layer been assigned an operating system process. In all three different process designs have been adopted by the two candidates and the final subsection describes these three process designs.

3.2.1 Implementation techniques and concepts

The SCTP protocol is new to the data communication world but it has already proven itself as being worthy to exist as a stand alone transport protocol, in opposite to many other protocols that have to work on top of UDP. As a stand alone transport protocol SCTP is designed to work directly on top of the IP-layer in the same way that TCP and UDP do today. Both TCP and UDP have been around for a long time now and they are therefore implemented as part of many operating systems. As a part of the operating system TCP and UDP are granted access to the IP-layer. SCTP is not implemented in any operating system and a non operating system implementations are often denied access the IP-layer and it makes it almost impossible to implement SCTP as a user process.

Luckily the RAW-IP socket, which some operating systems offers, allows privileged users to insert handcrafted IP packets into a network and the RAW-IP socket can also be used for reception of packets. [16]. The RAW-IP socket makes it possible to implement a SCTP layer as a user process with the restriction that the SCTP layer has to be executed by a user with privileged rights.

Together with the user space implementations of SCTP the need of a daemon arise, see section 3.2.2. A daemon is a separate process that provides a service point that other processes can connect to and use. This service could for example be a mail sending service or in this case a SCTP service point. One reason of making a single process responsible of a certain service is that there can in some cases only exist one such service point at any given time. For example there can only be one web server that provides a *Hyper Text Transfer Protocol* (HTTP) transfer service at TCP-port 80. If two processes are listening on this port would the outcome be undefined because they would both compete for the same messages arriving on a TCP-port. This situation is in most cases prohibited by the operating system but is seen as a good example of the need for a single service point.

Threads are an alternative for having multiple processes. Threads are sometimes called lightweight processes and they exist within a parent process. Threads share the process memory but have their own runtime stacks. The separate runtime stacks allows the threads to be executed in parallel. Threads are useful when one does not want several processes and still want to have multiple tasks to be performed in parallel. They also do not need a separate communication channel between each other because they share memory of the process[14].

3.2.2 One SCTP layer per host

At the time of writing SCTP was not yet implemented in any operating system kernel know by the author. Instead the studied implementations open an RAW-IP socket and listens for SCTP packets. The possibility to create multiple sockets that listens for the same packets

makes it possible to start multiple SCTP layers on a system, at least in LINUX which is the operating system used in the test environment.

But having multiple instances of SCTP active will not work because SCTP has been designed to handle packets that are not expected, called *Out Of The Blue* (OOTB) packets. In the normal case these packets may be received due to a crash followed by a quick restart of the computer, which had an SCTP association open to a peer before the crash. The peer will likely not be notified about the crash and therefore it could keep on sending packets to the crashed system. To notify the peer about the broken association SCTP's behaviour is to send an ABORT message to the peer that sends OOTB packets. When the peer's SCTP layer receives the ABORT message it can remove the invalid association and notify its ULP. Now if two SCTP layer instances receive the same packets, at least one instance would regard these packets as OOTB packets and send an ABORT message. This would sabotage for the other instance and make communication impossible.

3.2.3 User-space process assignments

A user-space implementation of the SCTP layer has to be executed, but how and where is an open subject. Three different approaches on where to put the SCTP layer have been seen in the two SCTP distributions from the competitors. This section describes them and explains their negative and positive sides that have been identified during this study.

1. An alternative is to run the layer inside the application, meaning that the layer will share a process with the main program. This has a negative effect; imagine that the application performs extensive calculation that require a long computation time. Then the SCTP layer would be left without any CPU time for a long time. The SCTP layer needs CPU time in a regular fashion so that it can obey its timing constraints. The performance of SCTP and its ability to act correctly is diminished if SCTP is not treated right. This approach makes it also impossible to have several user programs concurrently running a SCTP layer, because it would mean that two SCTP layers

would be active concurrently on a single host. For more details about why only one SCTP layer can be active on one host read section 3.2.2.

2. The second approach uses a daemon. The SCTP layer is assigned to a separate process, which makes it able to execute freely. The user program will also have its own process and can do as heavy calculations it want without risking to affect the SCTP layer. The side effect is that if one wants to use the daemon one has to use the separate communication channel. The positive side of this approach is that this design makes it possible to have several user programs concurrently using a single SCTP layer, which allows testing SCTP software on a single host.

3. The third approach is located somewhere between the first and second approach. By utilizing threads one can allow a process to perform more than one activity in parallel[14]. At least one thread can be assigned to the SCTP layer and one to the application. The threads will execute in parallel and SCTP will not be affected by the application behaviour. The need for an explicit communication channel between the application and the SCTP layer, which was needed in approach 2, is avoided due to that the threads exist in the same memory space. As in the first approach this approach has a side effect, which is that it is impossible to have two or more programs with a SCTP layer active on the same host.

3.3 Comparison

The two candidates will be compared focusing on four properties, its documentation, size of source code and usability. The documentation is important for the usability of implementation and the size of the source code for the implementation is regarded important for bug tracking and if any modification needs to be done to the SCTP layer. Further the usability of the implementation is affected by how the SCTP layer is implemented, read section 3.2.3 for details on different implementation approaches. The usability is also very

much depending on if the socket API is implemented, read section 2.3.5 for more details on this API.

The section describes each implementation, specified in Section 3.1, in detail focusing on the four properties mentioned above. The last subsection finally describes the motivation for why one was regarded as most appropriate for this study.

3.3.1 Randall R. Stewart

The purpose of this implementation is to allow tests on SCTP and also to show how one can implement a SCTP layer[23]. This study used version 4.05 of the implementation.

Documentation

The distribution available on Internet does not include a description of the use of this implementation of SCTP. There are header files with short descriptions and the function names often tell what they perform. There are also example programs bundled with the distribution that exemplifies how to use the distribution.

However, one can read chapter 14 in the book, "SCTP A Reference Guide", written by Randall Stewart and Xie Qiabing[23], to get a more solid description. It also explains the internal structure of the SCTP implementation and how the parts interact.

Size of source code

Approximately 25617 lines of code are written to realize the SCTP implementation, spread over 17 files (only counting the .c files).

Usability

Here there only exists one implementation variant and Randall focuses on the daemon approach, which allows the user to disregard SCTPs need of CPU time. To communicate with the daemon one must send UDP packets over the localhost interface, which are created

by a specific daemon API. The daemon approach has a positive affect on the usabililty because one can test SCTP software, both server and client, on a single host.

Socket API

The socket API is not implemented.

3.3.2 Siemens-Essen

This study has used the nineteenth pre-release of version 1, which is seen as a prototype implementation.

Documentation

Several manuals exist in their distribution and they explain how this distribution is compiled and used. There are also several test programs bundled and they serve as a complement for the manuals. These test programs are more up to date than the manuals and therefore they are seen as the most important information source. This distribution has three different APIs, described in the usability section below, and they are all well documented.

Size of source code

Approximately 19404 lines of code are written to realize the SCTP implementation, spread over 17 files (only counting the .c files).

The implementation is made general to allow it to be compiled and executed in several environments. As a consequence it is necessary to run a configure script that builds all the make-files and a configuration file (config.h), which is needed during compilation of the SCTP implementation and user applications. The generated configuration file contains macros that resolve conflicts with the local system libraries, which are supplied by the operating system.

Usability

This distribution contains three possibilities, Figure 3.1 gives an overview of the distribution. One can for starters create a single process application, containing both the application and the SCTP layer, and use the internal API. Through the mailing list for the Siemens project, it was told that this API is not regarded as final and it might be changed in later releases. This design also requires that the user application assign CPU time to the SCTP layer.

The second option is a multi-threaded single process design. The process contains the user application, the socket implementation and the SCTP layer. One thread is assigned to each part and they will receive CPU time separately from the operating system. The application uses the socket API to interact with SCTP. This API is under construction, but it is likely to exist in all future SCTP distributions.

The third and last option is the daemon. The API to the daemon is created nearly identical to the internal API of this implementation. This allows users to change at smallest possible cost between the internal API and daemon API. The daemon is implemented as a wrapper to SCTP and therefore it inherits all updates of the SCTP implementation. The daemon is unfortunately no longer maintained by the Siemens team and it will most likely be removed from newer versions of the distribution. Therefore, if one wants to use the daemon one must be aware that there might be bugs in the daemon and no support is available. Further, the daemon may have to be updated due to changes in the internal API of SCTP.

Socket API

A socket API implementation is available in this distribution. This API wraps the internal API of SCTP and not the daemon API, which is unfortunate due to the limitations implied with this API, which is that only one program with SCTP can execute at any given time on a host, see section 3.2. Both the TCP and UDP styles, mentioned in section 2.3.5, are

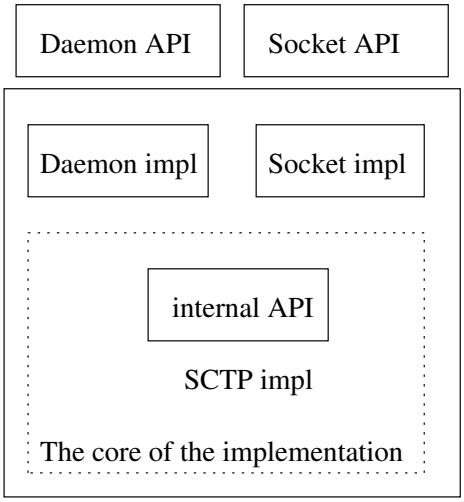


Figure 3.1: Implementation overview

available.

3.4 Conclusion

Both distributions help the user with assigning the SCTP layer enough CPU time, which is an essential service, but there are also differences between the two implementations. The most important difference is the socket API for SCTP, which is available in the Siemens distribution and not in Randal Stewart's. During the time of implementing small test applications for both implementations, Michael Tuxen, a member of the group that develops the Siemens implementation, recommended to use the socket API for SCTP. The other APIs are likely to be changed and will not be available any more when the kernel implementation of SCTP arrive. To keep the door open for code reuse in future experiments the Siemens implementation is preferable because it has the socket API.

Further, the documentation included in the Siemens distribution is much better. The usability of the Siemens distribution is also regarded as better due to socket API. The difference in source code size is seen as small and the two are regarded as identical. Hence

the Siemens distribution is regarded as most suitable mostly due to the availability of a socket API.

Chapter 4

Experiment design

This chapter defines the experiments that were conducted in this study in order to find out more about the transmission performance of SCTP. As mentioned in goal 1, from Section 1.1, other experiment possibilities than testing the streams of SCTP should be regarded before the final decision on which SCTP feature the experiments should test. This chapter starts with describing how the chosen feature of SCTP was elected and then it describes the experiments that tested the chosen feature. The last section describes the software components and the test network that made it possible to perform this experiments.

4.1 Choosing a suitable feature of SCTP

Not all of the features of SCTP are appropriate to address in a first study of SCTP. This section briefly discusses different features of SCTP that can be tested in a performance study and finally motivates why the stream feature of SCTP was chosen for this study.

4.1.1 Features of SCTP with experiment possibilities

There are several features in SCTP that are interesting because they allow data to be delivered in a way TCP cannot match in aspects of transport service and/or performance.

Here follows a list of features that were identified during this study as interesting and how each one of them can be applied.

- A possible application of the stream feature in SCTP is to adapt HTTP1.0 protocol for SCTP[23]. This application protocol is normally used on top of TCP and it is used for transporting web pages, with their HTML files, pictures and other parts. In HTTP1.0 each file requested by the client is normally transferred in a separate TCP connection[4]. If HTTP1.0 is used over SCTP each file can be sent over a separate stream, which eliminates the need for multiple connections. The throughput that underlying protocol achieves is the most interesting performance parameter for HTTP1.0. The use of HTTP1.0 over TCP and SCTP could be compared from the aspect of performance by measuring the throughput. One reason why SCTP could be faster, reaching higher a throughput, than TCP here is that the SCTP association will send more data than the individual TCP connections. A busy connection/association may in some cases discover packet loss faster than a connection/association with little data to transmit. Also the need to set up more than one connection is avoided with SCTP in contrast to TCP. Multiple TCP connections are, compared to a single SCTP association, on the other hand expected to more quickly make use of all available bandwidth on the network.

- A partial reliable transport is not a new feature for data communication, but by including it in SCTP some interesting possibilities appear. A partial reliable transport means that the user can specify how long SCTP should try to transmit a message. SCTP can then abandon the message when it has tried long enough. Further a SCTP association can be semi-reliable, meaning that a subset of streams can be unreliable and others reliable. This allows separation of traffic and one could then send application data that is not allowed to be lost over the reliable streams and application data with a high temporal importance can be sent over the unreliable streams. An application which can use this feature is a robust version of JPEG, where the control

information in the header of JPEG file and the picture data can be separated and merged by the application at the receiving end. This means that the picture can be transferred to the user with some degree of controlled data loss and the number of retransmission can be minimized, which shortens the total transmission time.

- Unordered message delivery can for example be used for a progressive delivery of a JPEG picture, which is coded for progressive display. A progressive coding means that each delivered part of a picture improves the picture quality of the entire picture, instead of just a small part, and finally when all the pieces have arrived the picture has the best possible quality. The good thing here is that a message might be lost without disrupting the stream of messages arriving to the user, which results in a more or less constant improvement of picture quality during the transfer.

- Multi homing is a feature that mainly improves the reliability of SCTP and is not designed to improve the transmission performance of SCTP. There are though some performance aspects that arise when retransmissions are made over the redundant paths but this is regarded as outside the scope of this study.

4.1.2 Motivation

One goal of this study was to perform an initial performance experiment on SCTP and therefore it was suitable to start with the basic features of SCTP. Streams are fundamental for SCTP and it is vital to understand this basic feature before more experiments on SCTP are done. It was also believed that the amount of time needed to prepare the test software for this feature was lower than the other.

The use of many streams was believed to be comparable to the usage of multiple TCP connection, which can give the same transport service. Both offer a transport service that allows sending multiple data objects simultaneously. An experiment where the two are compared can allow an interesting performance comparison. Table 4.1 lists possible

Streams	Connections	Description	Factors
1	1	One on One	Protocol overhead and differences in congestion control
x>1	x>1	HTTP1.0/SCTP vs HTTP1.0/TCP	Costs of multiple streams and connections

Table 4.1: Comparison possibilities

Scenario Name	Basic 1	Basic 2	Basic 3
Data (byte)	1K	3K	300K

Table 4.2: Basic scenarios

scenarios that were believed to be interesting to investigate together with factors that possibly have an effect on the result.

4.2 Experiment specification

As described in Section 4.1, the focus of this performance comparison was set on HTTP1.0 over SCTP and the usage of multiple streams. This section specifies the experiments of this study. Goal 2 for this study includes studying network dependent and independent effects and five different networks were therefore defined.

4.2.1 Experiment scenarios

In Table 4.1, two different scenarios are defined but they are loosely specified. This section specifies a limited subset of test scenarios that was tested in this study.

A small set of scenarios were defined to verify the basic behaviour of SCTP and TCP. These are referred as the basic scenarios. All data were in these scenarios transferred with a single TCP connection or a single stream in a SCTP association. The result of these scenarios was used as a reference when analyzing other scenarios, where many streams and connections were used. Table 4.2 specifis the amount of data sent in each scenario.

Scenario Name	MS 1	MS 2	MS 3	MS 4	MS 5	MS 6	MS 7	MS 8
Streams / Connections	1	2	5	10	30	50	100	300

Table 4.3: Multiple Stream (MS) scenarios

Before the scenarios for the HTTP1.0/SCTP vs. HTTP1.0/TCP comparison were specified, a small survey was performed to find a good example for modern web-page that could serve as a reference page. The front page of the Swedish news paper Aftonbladet[1] was found suitable. Microsoft Internet Explorer version 5 was used to download it and by monitoring the transmission it was found that around 50 TCP connections were used and the total amount of data was around 300KB. 300KB of data was regarded as valid for a many web-pages with quite a large set of pictures and text; 100 KB for the index HTML file and 200 KB for the pictures. With this information, it was decided that all the scenarios for the comparison should transmit a total of 300 KB of data. The only parameter varied in the scenarios was therefore the number objects that were transferred, where each object was assigned to a separate stream or TCP connection.

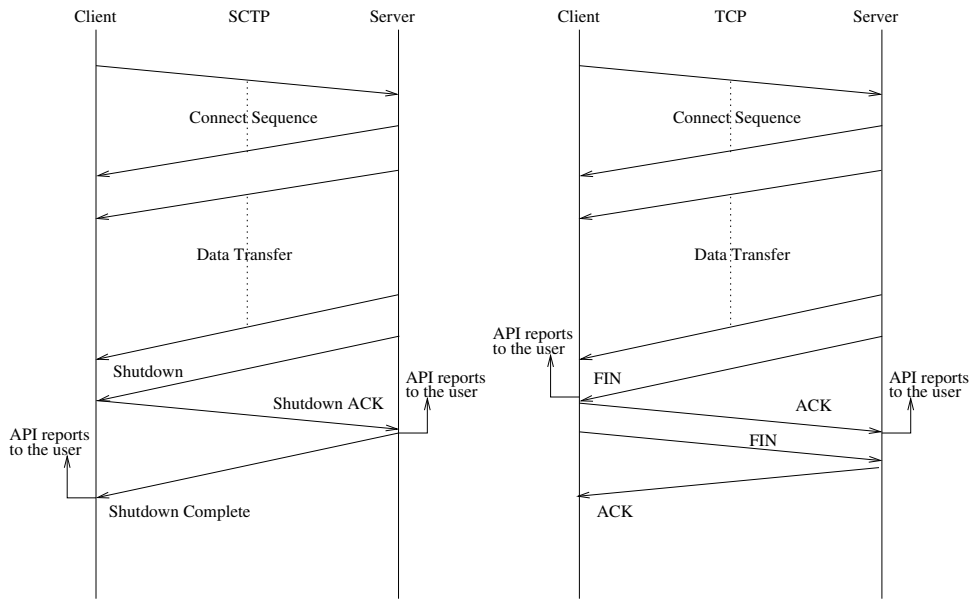
Table 4.3 specify the eight scenarios that were chosen for the HTTP1.0/SCTP vs. HTTP1.0/TCP comparison. The number of objects varied from 1 to 300 and the intermediate values were chosen to limit the object size to 1KB.

4.2.2 Test measurements

The measurements for comparing the the performance of SCTP and TCP were done by client, which initiates the connections/association to the server and receives the data from the server. The output from the client contained two different measurements. The client started its clocks right before it connected its connections or association and stopped its clocks at two different points, described below.

- R1: The time it took until the first connection/stream received its object.

[1] http://www.aftonbladet.se



(a) SCTP (b) TCP

Figure 4.1: Shutdown of SCTP and TCP

- R2: The time it took to receive all objects on all connections/streams, excluding the shutdown of the connection/association.

The R2 result was used to calculate the average throughput for both TCP and SCTP and the R1 result was used to calculate the specific throughput for the fastest connection and stream.

The shutdown of the connections/association was excluded due to socket-API differences between SCTP and TCP. SCTP's socket-API reports to the user when the association is completely closed and TCP's socket-API reports when no more data will be received on the connection and not that the connection is completely closed. Figure 4.1 shows a diagram for the shutdown of both TCP and SCTP. The diagrams illustrate the messages generated by the test applications specified in Section 4.3.2 and they also illustrate when the user is notified about changes in the status of the connection/association.

Name	Bandwidth	Loss Rate	Propagation delay (ms)	Router Buffer (packets)
GSM	7680 (bit/s)	0	310	50
Modem	48 down / 33.6 up (Kbit/s)	0	35	50
B1	60 (Kbit/s)	0	180	50
B1	60 (Kbit/s)	0	180	50

Table 4.4: Network configurations

4.2.3 Network configurations

Several different test networks were defined in order to get a wide perspective on the performance of HTTP1.0/SCTP and HTTP1.0/TCP. Today we have several different Internet connections available and all have different characteristics, such as bandwidth and round trip times. This experiment used the connection types shown in Table 4.4. This specification was a copied from a previous study on PRTP [9]. The effects on the performance from all these properties were not analyzed but they were regarded as valid as a starting point. The letter B in the network names stands for a broadband.

The network buffer were in all the networks configured to store up to 50 packets, which equals MSS*50 bytes of data, where MSS is 1400 bytes. This buffer size was greater than the highest possible *Receiver WiNDow*(rwnd) of TCP, which have a maximum of 63 KB, and SCTP, which have a maximum of 33 KB. The maximum flight size for both TCP and SCTP is the minimum of *Congestion WiNDow*(cwnd) and rwnd. A single connection/association will therefore eventually be limited by the rwnd and not cwnd and it prevents buffer overflow in the router. Buffer overflows can therefore only occur in tests with more than one concurrent connection/association. All the links in the test networks had a zero loss rate, which decrease the numbers of factors that affect the data transfer.

The internal performance of the TCP and SCTP layer may vary. Therefore, might a certain layer end up faster when the network allows very high transfer rates due to the internal performance. However, we did not expect to be affected by this because the test

networks all had a more or less moderate bandwidth.

4.2.4 Summary

In all, 22 scenarios were specified, counting both TCP and SCTP, and they were to be evaluated over five different network configurations. This made a total of 110 different experiments and to be able to verify the consistency of the results were they repeated ten times. Ten repetitions can be seen as a too small number for statistical analysis but it was regarded as enough to ensure that there were no unwanted parameters that affected the tests. The result from each run is a valid result as long as it has not been affected by any unexpected activity.

4.3 Experiment components and environment

This section describes the different components, including the test software that realized the experiment.

Goal 2 of this study states that a test application for HTTP1.0/SCTP and HTTP1.0/TCP must be created. This section first describes why the HTTP1.0 protocol was simulated and not fully implemented. Then an overview of the functionality of the test software is given, mapping the SCTP version with the TCP version. Then a detailed technical description follows for both the SCTP and TCP version, where the problems that occurred during development are described with their solutions. Finally, the network environment is described together with all its components.

4.3.1 Simulated HTTP transfer

To test HTTP1.0 over SCTP one has to both port existing an HTTP1.0 client and server to work with SCTP or write your own software from scratch. Porting existing software might take longer than one can believe and writing HTTP1.0 test software from scratch

is expected to be too time consuming. The third alternative is performing bulk-transfers that mimic the HTTP1.0 behaviour. This minimizes the time spent on developing software for both the SCTP and the corresponding TCP software. The request-reply protocol is therefore not implemented and the server application does not wait for the "GET" request after that a connection is established. Instead, it starts to deliver a specified amount of data over the new connection. This was regarded as a valid simulation, where the "GET" request is the only thing that is left out.

4.3.2 Software overview

Both the SCTP and the TCP versions of the client software retrieved a SCTP or TCP end point from the user and connect to it. In the case of TCP; a varying amount of connections were initialized and these connections were initialized by the client application in a sequence. The motivation for the sequential connection initializations can be read in section 4.3.4. In the case of SCTP; a single association was initiated with a varying amount of streams. Quickly summarized, multiple TCP connections are mapped to a single SCTP association with multiple streams.

The server side was notified about the new connections or the new association and started sending data on each stream/connection. Each stream was fed with the same amount of data, specified by the user at the start-up. The TCP connections were closed individually as soon as they have transferred all their data, but the SCTP streams are left unused when they are done and when finally all streams were done the application closed the entire association.

4.3.3 SCTP application details

Both the client and server application were single process construction that receive or send data on a varying number of streams. The socket implementation for SCTP has no bandwidth sharing between the streams in the association. Instead, it handles send requests

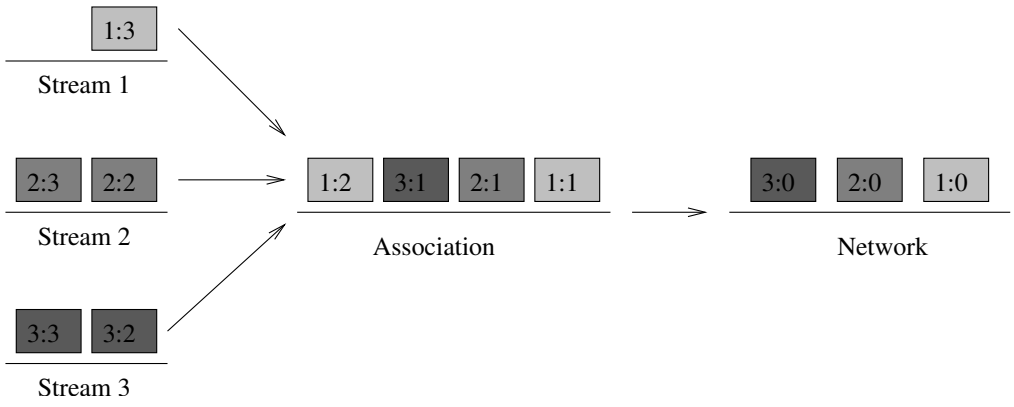


Figure 4.2: SCTP - Bandwidth sharing

in a first come first served fashion. A single send buffer serves all outgoing streams and if one wants the streams to transfer in parallel one must not have entire buffer completely occupied by one stream. Otherwise one will not be able to avoid *Head-of-Line Blocking* (HLB) situations. The application therefore splits the data into parts with a size equal to the MTU. It then iterates over all the streams in a round robin fashion, which forces SCTP to spread the bandwidth of the association evenly over all the streams.

A situation where the association is evenly utilized by three streams is illustrated in Figure 4.2. The squares in the figure symbolize the parts of the data and the shading and the number before the colon defines which stream the part belong. The number after the colon enumerates the packets per stream basis. In the figure, each stream has its own queue and the streams share the association. The association queue contains the packets that are queued for delivery and the network contains the packets that are being delivered.

4.3.4 TCP application details

The TCP version the client software is also a single process design, which handles multiple TCP connections. The client ensures that each connection is read often enough to avoid saturated connections, which happen when a connection's receive buffer becomes full.

Setting up a single SCTP association can only be done in one way but setting up multiple TCP connections can be done in several ways. Connections can be initiated in parallel, in sequence or in a way that combines the both previous alternatives. A pure parallel approach where the software tries to initialize all connections at once is expected to be bad. Sending a large number of SYN packets in parallel could fill the buffer of a packet-limited router in the network and the router could drop SYN packets. Further, too many parallel connections can together reach a far too high amount of outstanding data, causing high drop rates if the routers limitations are exceeded. In situations with high drop rates can it be difficult to initiate new connections that are not yet established.

A pure sequential algorithm does not risk filling the buffer in the same way. On the other hand, this algorithm is expected to be less aggressive and also slower in high capacity networks. A combined algorithm, located between the sequential and the parallel algorithm, is there third alternative where the connections are initiated in sequential bursts. This algorithm demands that a strategy is defined, defining the size of the bursts and the delay between the bursts.

The client was designed to initialize connections in a sequence, meaning that once a connection has been initialized the client initialize the next connection. This ensures the simplicity of the application and that all connections are established in low capacity networks. Figure 4.3 visualize the sequential connection sequence with three connections.

The server is a multi-process design. When a new connection is initiated from the client, a new process is created and assigned to serve this connection. This process will immediately send data on the new connection and close it when all is sent. By assigning one process to each connection will they be handled in parallel and compete for the bandwidth of the network.

The SCTP application avoids HLB with its multiple streams and TCP avoids it with its multiple connections. One down side of multiple connections is that it there is no way to ensure that the bandwidth of the network is evenly shared between the connections in

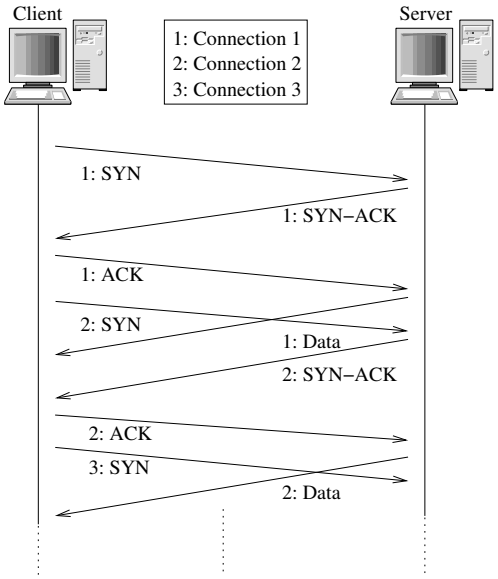


Figure 4.3: Sequential initiation of multiple TCP connections

a short time perspective. However, the TCP connections will in the long run eventually reach a steady state were the connections have the same throughput.

The data payload of a SCTP packet is limited to 1400 bytes in the Siemens implementation of SCTP. All TCP connections are configured to have the same limit, which allows the two protocols to be compared with as small parameter variation as possible. This was done by setting the *Maximum Segment Size* (MSS) for TCP to 1412 bytes, with 12 bytes for the Time-stamp option header. The SACK feature also transport it self as an option but SACKs don't need to be include in the calculations due to that data is only sent in one direction and SACKs are sent in the opposite direction and does compete with the data for space in the segments.

4.3.5 Implementation experience

The TCP client was somewhat difficult to program because the client had to measure the performance for all connections. The system call "poll" was the tool that fitted best for

this task, because it took all the connections as a parameter and reported back immediate when one or more connections have received data or when a connection have been closed by the server. The software simply uses the "poll" function until all connections are closed.

For the TCP server one issue came up because it was unknown what the maximum amount of new pending TCP connections a socket in Linux can handle. This limit is defined by the user, at the call to the listen() primitive, but it has also an upper limit that the system defines. The test environment uses the Redhat 7.2 Linux distribution and by investigating this limit, it was found that it was set to 256. Up to 300 connections were in the experiments, but the sequential initiation of the connections makes it impossible to reach this limit.

For the SCTP version of the applications, the socket-API was used and the socket implementation itself had some bugs in the beginning. These bugs had to do with the closing of associations and the reception of notifications, which made it hard to accurate measurements. The socket-API was luckily in contrast to the daemon API maintained and the bugs were solved in later releases of the Siemens distributions.

The socket API for SCTP was besides the initial bugs somewhat difficult to use. Several new data structures had to be used and new function calls had to be done, compared to the socket API for TCP. The send and receive primitives of SCTP, sendmsg() and recvmsg(), were the only ones that can make full use of the SCTP protocol and they were somewhat difficult to learn due to the new structures that had to be used. A conclusion that can be drawn is that if one wants to use all the features of SCTP one has to be an experienced programmer. The positive side with the socket-API of SCTP is that it gives a nice feeling of control over the association.

4.3.6 Network environment

The *Communications Applied Research Laboratory* (CARL) at Karlstad University have been used in several other communication experiments, for example has various perfor-

mance studies been made on the *Partial Reliable Transport Protocol* (PRTP)[3]. CARL is therefore prepared with several utilities and components one might need.

Physical components

The test environment is a controlled network with three computers; a client, a server and a network emulator. All three computers were Dell Optiplex GX1 with a 100Mbit/s network card. The client and server ran Linux Redhat 7.2 and the third computer ran FreeBSD and used *Dummynet* to emulate a drop-tail router with a limited buffer size. A drop-tail router discards arriving packets when its buffer becomes full. Dummynet also allows emulation a network links and they can be specified based on bandwidth, loss rate and delay. The emulated router together with the network link together creates a pipe, which the data will pass through.

By creating traffic filters can specific network traffic be selected, depending on several parameters such as transport protocol, source IP-address and more. By sending this traffic through a specific pipe can it be forced to experience a wanted network environment.

Both the client and the server hosts are setup with the UNIX tool *tcpdump* in order to log the traffic on network. *tcpdump* logs all traffic on the network by writing down the time when each packet was sent/received and it also stores the packet so that the conversation can be studied later.

Network topology and configuration

The three computers were connected so that they can create a link between the server and client that pass through the network emulator. Figure 4.4 shows the network topology and all the *Network Interface Cards* (NIC) together with their network addresses. The arrows between the NICs and the switching hub show how they are physically connected. Notice that 192.168.0.103 and 192.168.0.100 were connected directly with a crossover cable.

In order to make sure that all experiment traffic passes through the network emula-

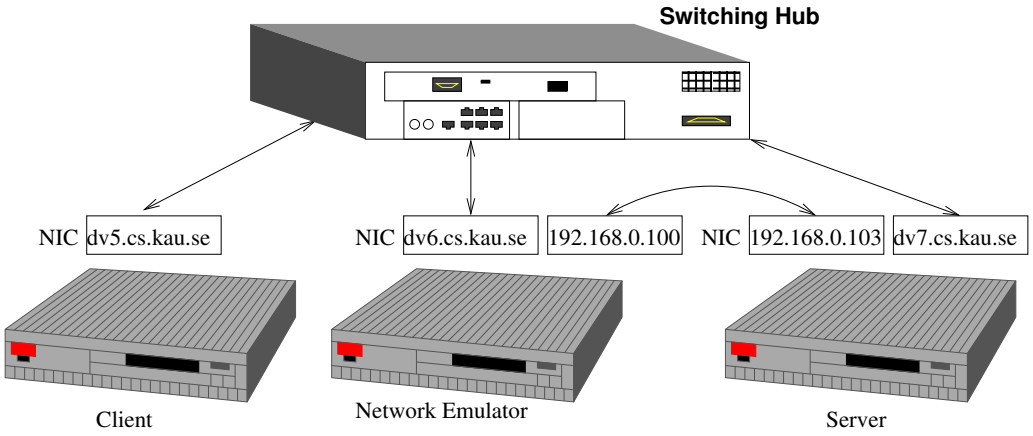


Figure 4.4: The components creating the experiment network

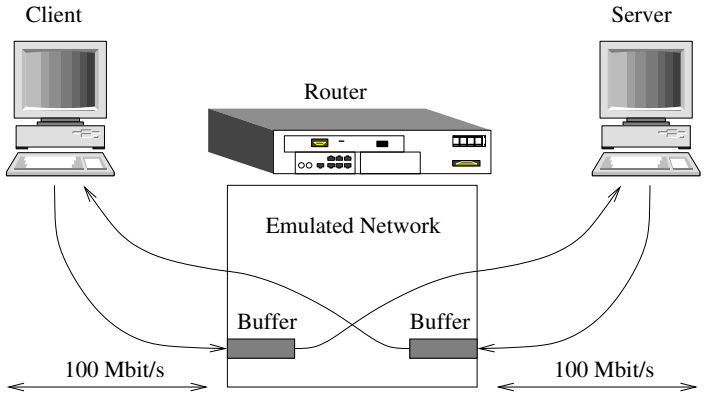


Figure 4.5: A logical view of the experiment network

tor the test software that executes on the server must use the interface with IP address 192.168.0.103. This forces all communication to go through the network emulator. The client does not have any direct access to the 192.168.0.* network and it therefore routed all its traffic to 192.168.0.103 through dv6.cs.kau.se. The network emulator forwarded the messages to 192.168.0.103 and at the same time applied the network emulation. The server always sent its traffic to dv5.cs.kau.se through 192.168.0.100 in order to make sure that the traffic in the opposite direction goes the same way and pass through the network emulator. Figure 4.5 gives a logical view over the components in the network.

Chapter 5

Experiment Results and Analysis

The execution of the scenarios generates a lot of data that has to be checked for correctness and analyzed. This chapter starts with the verification of correctness in the results and then continues with a presentation of the results of the throughput measurements. The final section contains the analysis, which tries to identify the details behind the results.

5.1 Verifying the results for correctness

The results from the execution of all scenarios have been verified and this section describes the method of verification and how the encountered issues were handled.

5.1.1 Method

The log files generated by tcpdump were analyzed with Ethereal[1] and tcptrace[13] to ensure that the measured results reported from the test applications are accurate. Ethereal is able to analyze both TCP and SCTP and quickly summarize communication and calculate the total transfer time. It can also display all the contents of the packets sent during the test by disassembling the stored information in the tcpdump log files. The output from ethereal is a readable list of packets with their contents.

The second tool used, tcptrace, can sadly only analyze TCP transfers but it has still been used. The output from tcptrace is both text and graphs summaries. The graphs from tcptrace are many and the most useful one is the Time-Sequence diagram. This diagram contains information about when packets are sent, when retransmissions occurred, when acknowledgments are retrieved and more. Another limitation that tcptrace have is that the generated graph only visualize one connection. Therefore, one has to look at several graphs to get the entire picture of what happened when several connections were active.

5.1.2 Variance in the results

The test results showed that there was only a small variance between the repetitions and it can be assumed that there are no unknown parameters, such as other data flows on the network, affecting the tests. An average value of all repetitions is used in this document. An average is seen as a good approximate for the value because there are no extreme variations in the results.

5.1.3 Application result versus the log

The R1 and R2 results, defined in Section 4.2.2, for both SCTP and TCP were checked with Ethereal to verify that the applications reported accurate values. The difference was one thousand of a second for both SCTP and TCP.

During the verification of the R1 results, the logs identified a problem with the client test application for TCP, which was designed as single process software. The problem was found in the handling of with sockets, which was configured as blocking, meaning that each connect operation on a socket blocks until it is done. This design did not allow the application to retrieve data on initiated connections until all connections have been initiated. The reported R1 results were therefore faulty in the scenarios where a connection had transferred all its data before all connections were initiated.

Further, this design hindered the client application from reading the arrived data from the already initiated TCP connections when it still was initiating new connections. This resulted in that some connections managed to transmit all its data, which temporary is stored in the receive buffer, before the last connection was established. In MS 6 for example, the effect was that 49 connections were closed at once when the last connection was established, due to that they had already transmitted all its data. These 49 connections then tried to close them self and all these closures generated 49 close message that caused congestion at the router, which had a negative effect on the remaining connection.

This was regarded as unfair for TCP, which was expected to perform better if the connections were independently connected, read and closed. The solution was a more complex version of the client application, where the single process design and sequential connection sequence was kept, but all sockets were instead configured as non-blocking, meaning that all operations on the socket return before the wanted result is acquired. This forces the program to check all sockets afterward to ensure that the wanted operation has been performed.

5.1.4 Auto-tuning in TCP

Scenarios Basic 3 and MS 1, defined in Section 4.2.1, are identical and the results from the two should also therefore be identical, but they were found different. However, a difference in the behaviour of TCP was found after a study of the logs. In Basic 3, TCP acted as anticipated and increased the amount of outstanding data until it reached the limit of rwnd. When MS 1 was executed, TCP acted less aggressive and did not reach the rwnd limit as TCP did in Basic 1. The result for TCP was not affected by this in any negative way but this less aggressive behaviour of TCP could be positive or negative in all the other MS scenarios, where multiple connections are used. For example, a less aggressive TCP might help avoid congestion and dropped packets and on the other hand may the less aggressive behaviour hinder the connections from using all the available bandwidth.

The reason for TCP's behaviour was found in the kernel 2.4 for Linux, which caches connection parameters from previous connections and this data is used by the new connections[6]. This makes TCP aware of the network even before it has sent any data. This however created a relationship between different experiments and repetitions and this is unwanted. Luckily, this cache could be cleared by the user, which was done before each repetition of a experiment.

5.2 Experiment results

The figures showing the result from the experiments are first described in this section and then this section continues with the results. The analysis and discussion of the results are left for the next section.

5.2.1 Data representation and figure explanations

Each scenario was repeated ten times for each network and the results from these were an average calculated. The averages from each scenario, network and protocol combination is presented in the graphs. A confidence interval is regarded as appropriate to use together with an average representation, but the interval would hardly be visible in graphs because it was found very small.

Two different types of graphs are used. For the BASIC scenarios, SCTP's and TCP's R2 results from all the different scenario and network combinations are compared in one single graph. The dots connected with a solid line represent SCTPs results as a percentage of TCP results. A value of 100% means that TCP and SCTP is identical, a value less than 100% means that SCTP is faster and a value greater than 100% means that SCTP is slower. The dots connected with a dashed line represent the difference between TCP and SCTP in seconds, which can be read in the right y-axes. A positive value means that SCTP is slower TCPs.

The second type of graphs is used to represent the results from the MS scenarios. This graph shows the R1 and R2 results of either TCP or SCTP in all the different scenario and network combinations. The R1 and R2 results from the same scenario and network create pairs that are connected with a line, which creates a interval. All streams/connections have finished within this interval. The figures also contain a comparison between the R2 result of MS1 and R2 result of MS 2-8. The round circles denote how much longer time in percent MS 2-8 required compared to MS1. The value was calculated with this formula $(MSx - MS1)/MS1$.

5.2.2 Basic scenarios

Figure 5.1 shows the results from the basic scenarios and shows that there was never more than 1 second between TCP and SCTP. Nevertheless, there was a difference between the two. One can see that SCTP was from 4 to 13 percent behind TCP in Basic 1 throughout all networks. SCTP was closer to TCP in Basic 2 than in Basic 1 scenario, where the SCTP results were from 3% to 6% slower than TCP. In Basic 3, where 300 KB of data was transferred, the figure shows that SCTP has caught up with TCP in all networks.

5.2.3 MS scenarios

The results of the MS scenarios are summarized in text and figures in this section. Figure 5.2 shows how both TCP and SCTP have reacted to changes in the amount of connections/streams and to changes in the network properties.

- R1 (Time until first stream/connection has finished): SCTP did in MS 1 finish its first stream as quickly as TCP. SCTP then finished its first stream further and further behind TCP the more connections/streams were used. The trend stops with MS 8, where SCTP again was on track with TCP and the two has identical results. One can see that the less data were sent on each connection the faster the first TCP

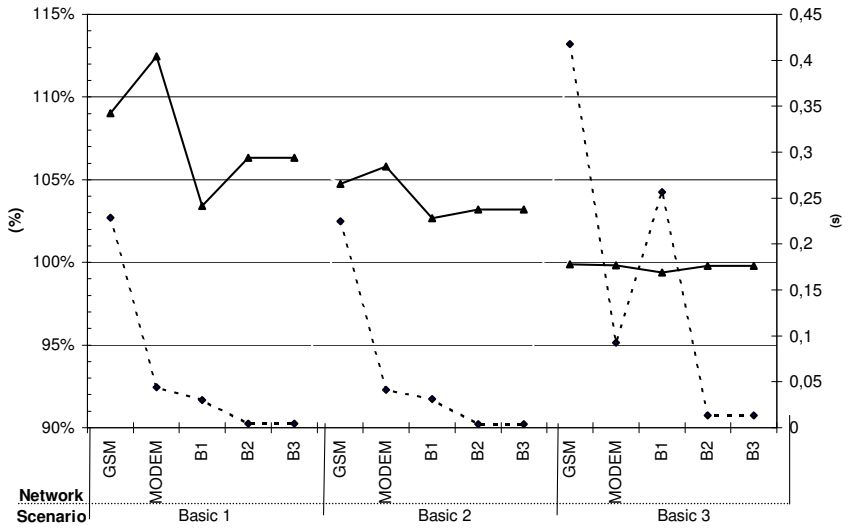


Figure 5.1: BASIC scenario results

connection was finished. This is true both for TCP and SCTP, but TCP finished its first connection far much earlier than SCTP's finished its first stream, except in MS 1 and MS 8. It was also been observed in the logs that the first initiated TCP connection was also the one that finishes first. The same was observed for SCTP's streams, which finish in the same order as they were used.

- R2 (Total transmission time): SCTP's result was in all networks almost identical no matter how many streams were used. The difference between MS 1 and 8 was less than two percent. In contrast to the SCTP application, the TCP application experienced longer transmission times the more connections were used. TCP's results for MS 2, 3 and 4 were less than 1% slower than MS 1 and MS 5 was only slightly slower than MS 1 in the GSM network. MS 6 was the first scenario for TCP where it struggled in all networks, it required from 2% to 21% more time than MS 1 and the transmission time increased in exponential fashion in MS7 and MS 8. One can also see that the difference between TCP's MS 1 result and the other scenarios vary between the different networks, which also indicate that the networks properties have

been a factor that have effected the TCP application's results.

5.3 Analysis

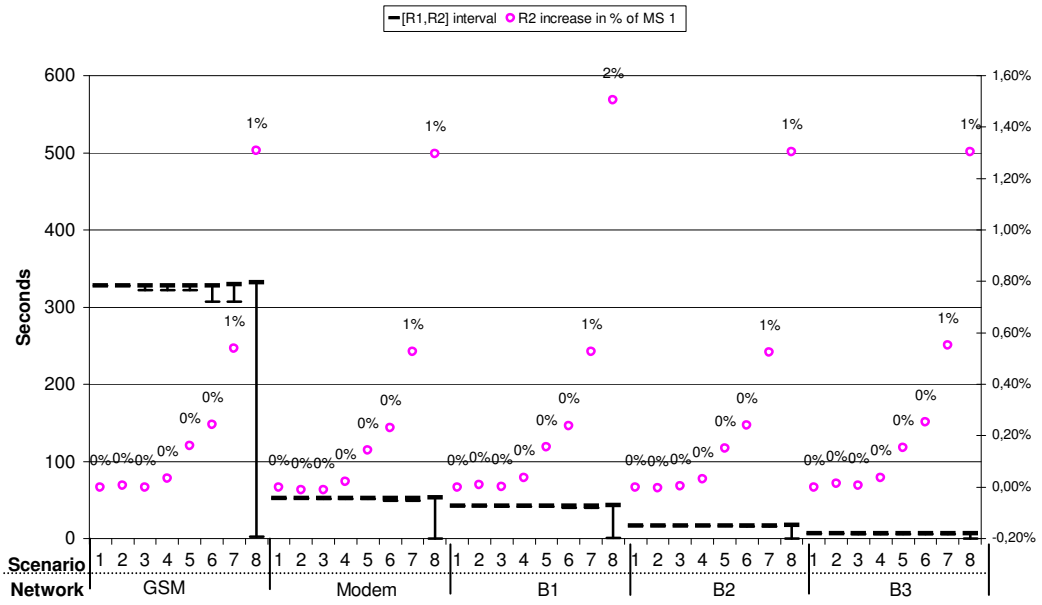
In addition to tcptrace, and ethereal one more tool was used during the analysis of the logs from the tests. The first section describes this tool and what it can do. The next section lists important issues found on why SCTP and TCP perform as they did. These issues are then used when explaining the results from the MS scenarios, which is done in the two final sections.

5.3.1 Excel as an analysis tool

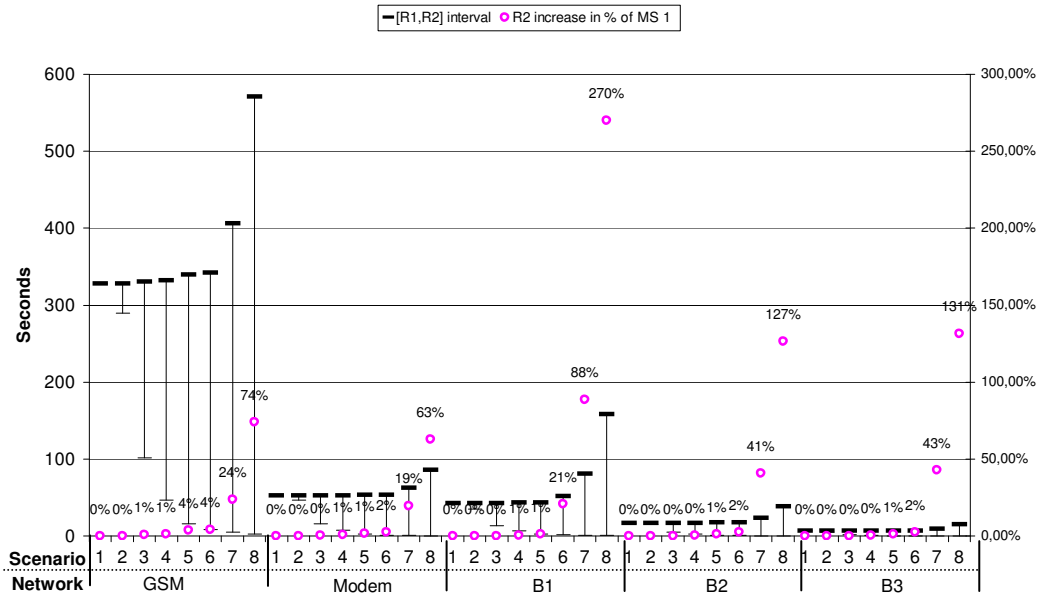
It is interesting to examine the utilization of the network during the tests, but this was not done during the execution of the tests. A method for examining this by utilizing the logs, generated by tcpdump, was created for this study. By filtering the log-files for the communication in one direction, in this case; Server to Client, one can focus on direction at the time. The filtered log-files were converted to text-files and then imported to Microsoft Excel, where the data was used as an input for calculations. By knowing the bandwidth of the network one can calculate how much transmission time each packet require before a new packet can be transmitted.

A snapshot from the resulting Excel tables can be seen in Table 5.1. The data in the table is from a run of MS 6 with TCP over the B1 network and the direction of the data is from the server to the client. The *Time* column shows the time when the packet was put into the network queue and the *Start time* column specifies when the packet starts using transmission resources. The *End time* column is the time when the packet is expected to have received all the necessary transmission resources and left the server.

The network is buffering when the *Start time* is later than *Time*, but if the two times are identical the network may have been idle for some time. The *Idle time* column in the



(a) SCTP



(b) TCP

Figure 5.2: MS scenario results

No.	Time	Pkt Size	Src. port		Dest. port	Info	Start time	End time	Idle time
2	0,000	76	5577	>	2886	[SYN,	0,000	0,010	0,000
4	0,383	1468	5577	>	2886	[ACK]	0,383	0,579	0,373
5	0,383	1468	5577	>	2886	[ACK]	0,579	0,774	0,000
7	0,390	76	5577	>	2887	[SYN,	0,774	0,784	0,000
9	0,944	1468	5577	>	2886	[PSH,	0,944	1,140	0,160
10	0,944	1468	5577	>	2886	[ACK]	1,140	1,335	0,000
11	0,944	468	5577	>	2886	[FIN,	1,335	1,398	0,000
14	1,146	1468	5577	>	2887	[ACK]	1,398	1,594	0,000
15	1,147	1468	5577	>	2887	[ACK]	1,594	1,789	0,000
17	1,154	76	5577	>	2888	[SYN,	1,789	1,799	0,000
22	1,959	1468	5577	>	2887	[PSH,	1,959	2,155	0,159
23	1,959	1468	5577	>	2887	[ACK]	2,155	2,350	0,000
24	1,959	468	5577	>	2887	[FIN,	2,350	2,413	0,000
27	2,160	1468	5577	>	2888	[ACK]	2,413	2,608	0,000
28	2,160	1468	5577	>	2888	[ACK]	2,608	2,804	0,000
30	2,168	76	5577	>	2889	[SYN,	2,804	2,814	0,000
35	2,974	1468	5577	>	2888	[PSH,	2,974	3,169	0,159
36	2,974	1468	5577	>	2888	[ACK]	3,169	3,365	0,000

Table 5.1: Idle calculations

table lists the difference between when the previous packet had received all transmission resources and when the next packet started using transmission resources. There are often packets in the buffer so this time is zero, but in the example in Table 5.1 one can see that the network was idle for 160ms three times, meaning that the transmission resources of the network was unused between no 2 and 4.

The data in the Excel sheets were also used as an input to a small Visual Basic program, which was developed to calculate how many packets the buffer was occupied with at the time when a packet is sent. This amount was compared with how many packets the router can queue and if the calculated value is greater then must a packet have been dropped.

5.3.2 Important parameters and behaviours that can affect the result

This section discusses a number of issues that have been identified during the analysis and seen as important when discussing SCTP's and TCP's results. The impact on the performance for each issue is not discussed in detail.

How TCP and SCTP sends ACK

It is especially interesting to know how TCP acknowledges the received data at the start of a connection. The aggressiveness of the TCP protocol is affected by how fast data is acknowledged. By studying the logs it was discovered that the Linux kernel 2.4 implementation of TCP does not delay ACKs in the beginning of the Slow-start algorithm. Delayed ACKs that acknowledges two or more segments hinder TCP from increasing its cwnd as much as if it had received two or more ACKs[10].

For SCTP the logs also show that the Siemens implementation of SCTP does not delay the ACK on the first packet but then it ACKs at least every second packet. If SCTP was as TCP ACK-counting then it would not increase its cwnd as fast as TCP due to the delayed

	GSM	Modem	B1	B2	B3
Data packet size (Bytes)	1468	1468	1468	1468	1468
ACK packet size (Bytes)	68	68	68	68	68
Bandwidth down (bit/s)	7680	48000	60000	150000	400000
Bandwidth up (bit/s)	7680	33000	60000	150000	400000
Propagation delay (s)	0.310	0.035	0.180	0.030	0.012
Bandwith delay product (Bytes)	595.2	354.4	2700	1125	1200
Transmission delay: 1 * Data pkt. (s)	1,529	0,245	0,196	0,078	0,029
Transmission delay: 2 * Data pkt. (s)	3,058	0,489	0,391	0,157	0,059
Transmission. delay: 3 * Data pkt. (s)	4,588	1,068	0,587	0,235	0,088
Time for DATA + ACK (s)	2,220	0,331	0,565	0,142	0,055

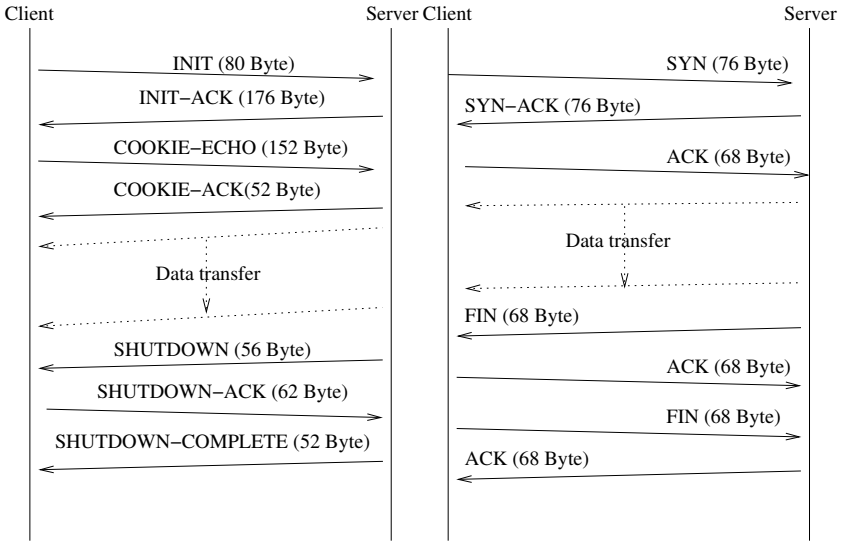
Table 5.2: Buffer calculations

ACKS. However, SCTP's congestion control algorithms are byte-counting so SCTP will not loose speed of the delayed ACKs.

Network parameters

Another key needed for understanding why SCTP and TCP perform as they do was retrieved by studying the network parameters that were chosen for these experiments, defined in Table 4.4. Table 5.2 contain various calculations for all networks and the results give us some interesting information. It turns out that networks are regardless the differences in bandwidth and delay very similar.

The B1 network was the only network where it requires more time to receive the ACK on the first data packet than it takes to transmit two data packets, each containing 1468 bytes of data. Remember that TCP sends two packets at the start of the initial slow-start. During the time after that the second packet was sent and when the ACK returns is the network idle, meaning that the network is idle. This idle period gives an opportunity for other connections to transfer data. This idle time is though very short and if a connection sends three instead of two packets at the start will the idle time be zero. In the other networks, the opposite was true meaning that there was no idle time in the network.



(a) SCTP (b) TCP

Figure 5.3: Connection comparison

The cost of the cookie mechanism

SCTP sends bigger messages than TCP during the initiation of an association due to its cookie mechanism. The extra amount of data, 240 bytes more than TCP, extends the transmission delay for the initiation. The size of the cookie sent back and forth over the network is implementation dependent and this value is only valid for the Siemens implementation. The significance of this becomes naturally less when the amount of user data is increased. Figure 5.3 visualize the packets sent by SCTP and TCP during the initialization with their sizes, including headers of IP layer and Ethernet.

Added transmissions delays from the multiple connection

Initiating multiple TCP connections increases the traffic load in both direction, but Figure 5.3 shows that TCP only needs to send one packet, SYN-ACK, in the same direction as the experiments user-data for each new connection. The transmission delay for this packet

in a GSM network is 80ms. A connection actively transmitting data in this direction will see its packets delayed with 80ms for each connection that initiates.

The closure of the TCP connections also adds on the amount of data sent during the test, illustrated in Figure 5.3. Two packets are sent from the server for each connection, a total of 136 bytes, during the shutdown, which generates a transmission delay 142 ms in a GSM network. In total, counting the connection phase and closure, each connection generates at least 222ms extra transmission delay in the direction from the server to the client.

TCP timeouts during the initialization

TCP has time limits on the response time for the packets sent during initialization of a connection and on the packets in the shutdown sequence. With several TCP connections active on the same network these limits can be exceeded due to long buffer delays in the network buffer or congestion.

TCP sends its SYN packet at max 5 times and the delay between the retransmission is {3,6,12,24} seconds, where 3s is the first delay and 6s is the second and so on. The peer, who receives the SYN packet answers with a SYN-ACK packet, which also has time constraints on how fast its response should reach back. A maximum of 4 retransmissions are done from each side but the logs show that TCP does not count the answers, SYN-ACKs, that are sent directly after receiving a SYN, as a retransmission. The passive side of the two TCP endpoints can therefore in a worse case scenario send up to nine SYN-ACK packets, 5 answers and 4 retransmits. The retransmission of SYN-ACK packets are the most relevant in this experiment because they are sent in the same direction as the user data and will add some extra delay.

These retransmissions can potentially increase the traffic load on the network, but it is hard to specify in advance exactly how much. The number of retransmissions can easily be investigated but to be sure about the increased load on the network one has to use the

logs from both sides of the connection. The two logs can only together verify whether a packet went through the drop-tail router or if it was dropped. A packet that is dropped by the router will never use any of the transmission resources of the bottleneck link.

Two packets are normally sent during the closure of a TCP connection from the server to the client, see Figure 5.3. The FIN packet may be retransmitted if the peer does not acknowledged the FIN packet fast enough. This is normally not needed because the time-out time is more accurate than the time-out times on the SYN and SYN-ACK packets. During the data transfer have TCP acquired information of the network and it can based on the information set the time-out time to a more accurate value. A FIN packet from the server that makes it through the network is very likely to be promptly acknowledged by the client, because the link from the client to the server has no congestion in the experiments. The FIN packet from the client have no problem reaching the server but the acknowledgment from the server may be lost due to congestion or delayed too much. The client may therefore retransmit its FIN packet but this does not delay the transmission of the user data in the other direction.

The header overhead for data packets

SCTP has a smaller header overhead than TCP. The header of a SCTP data packet, with one chunk, is 4 bytes less than TCP. Transferring 300 KB of data over one connection or SCTP stream will require 215 data packets for both TCP and SCTP. In total, not counting the connection phase, TCP will send 860(4*215) Bytes more than SCTP.

At the end of the transfer it often exist a small amount of data that does not fill an entire packet. SCTP can bundle several chunks in one SCTP packet if there are several streams with small chunks waiting to be delivered. Each chunk has its own header but they share the common header of the packet so the header overhead decreases. For TCP a similar situation can occur but several TCP connections cannot cooperate and share a single TCP packet, as the streams in SCTP share a SCTP packet. The overhead then increases even

Number of chunks	TCP header	SCTP header	Difference(TCP/SCTP)
1	32 B	28 B	+14%
2	64 B	44 B	+45%
3	96 B	60 B	+60%
4	128 B	76 B	+68%
5	160 B	92 B	+74%

(B = Bytes)

Table 5.3: Header overhead

more for TCP in comparison to SCTP. Table 5.3 contain a comparison between TCP and SCTP regarding how the header overhead change when SCTP bundle chunks in one packet and TCP sends them in separate packets. The TCP and SCTP header column lists the total amount of header data needed for the specified amount of chunks. The difference column list how much more header data TCP need than SCTP.

The slop-over effect

A SCTP association starts more aggressive than TCP connection because it may temporary send more data than the initial cwnd allows, read Section 2.3.4 for more details. In the B1 network, where TCP cannot fully utilize the available bandwidth at once, can SCTP slop over its cwnd and send three full packets (4200 Bytes). This is enough to keep the network working until the first ACK returns.

Data packets dropped by the router

When parallel TCP connections probe the network for the available bandwidth, they are likely to fill the buffer of the router. The router used in the experiment, which is a drop-tail router, handles the problem with a full buffer by dropping the packets that arrive when the buffer is full. The dropped packets have to be resent and packets can arrive unordered. The cost of dropping packets is not that expensive as one can think because they will not have used any of the constrained network resources.

The connection that loose a packet will on the other hand slow down and reduce the amount of outstanding data. The load on the network will become less, the network may be under utilized for some time and the transmission time will increase.

Bandwidth sharing between the data flows

The R1 result is very much dependent on how the data flows share the available bandwidth of the network. The test software for the SCTP version force SCTP to spread the bandwidth over all streams. The test software for the TCP version does not spread the bandwidth evenly between the flows because it has no possibility to do that. Multiple TCP connection does not have a shared association as the streams of SCTP have. Instead the connections compete for the bandwidth, where TCP's congestion avoidance rules define how the competitors act.

If the connections also are initiated in a sequence will the first initiated connection have a head-start on the others, which allows it to increased its cwnd before the other connection even have started. The subsequent connections are therefore hindered by the first connections transmission delay and their messages will be delayed. It can take some time before the other connections have forced the first connection to decrease its cwnd and evenly share the available bandwidth with the other connections. Figure 5.4 illustrates three connections that have two packets each to send. Connection 1 is the first connection to be initiated and the figure illustrate that connection 1 will be able to send all its packets at once and finish long before the other two.

5.3.3 Basic scenario analysis

This section analyzes the result for each Basic scenario, showed in Figure 5.1, and summarizes them by referring to some of the topics mentioned in the previous section. Remember that the R2 result is only valid in the basic scenarios because no more than one stream or TCP connection was used in Basic scenarios.

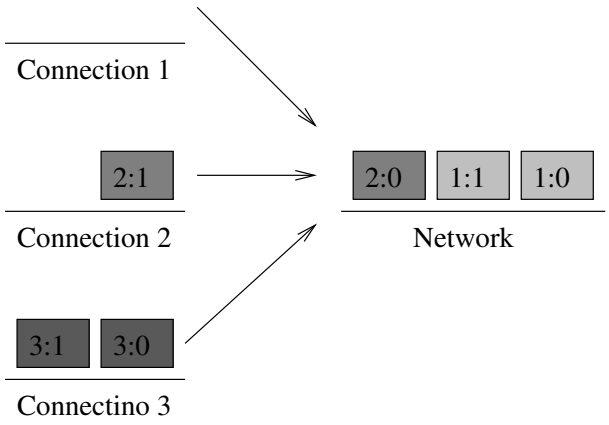


Figure 5.4: TCP - No bandwidth sharing

Basic 1

SCTP turned out to be slower than TCP because SCTP suffered from its connection phase, where the cookie mechanism required that 240 bytes more to be sent compared to TCP. 240 bytes generate a 0.25s transmission delay in a GSM network and it covers for the entire difference between TCP and SCTP in this network. SCTP dropped further behind in the Modem network, where the bandwidth was higher downstream than upstream and SCTP send more data upstream than TCP during initialization. In the B1 network, which had a high bandwidth in both direction but a long propagation delay, the transmission delay was tiny in comparison of the total delay. This hid the loss in time for SCTP during initiation. The situation in the B2 and the B3 networks, which had a less dominant propagation delay than B1, were basically the same as in the GSM network and SCTP's need of extra transmission time during initiation was again visible.

Basic 2

This scenario transfers three times more data than Basic 1 between the server and client and the transmission delay of the connection phase is therefore less significant. The cookie mechanism was however noticeable and it put SCTP behind TCP.

Basic 3

The amount of data transferred between the server and client was here 300 KB and we saw that SCTP caught up with TCP in all networks. The 240 bytes more that SCTP have to transmit during the initiation was eaten up by the fact that SCTP's data packet is four bytes smaller than TCP. SCTP did in total send less data than TCP. TCP sent 620 bytes (860-240) more, but it was only a little part of the entire transfer.

Another reason why SCTP has caught up with TCP was that it was favoured by its slop-over rule, which allows SCTP to temporary exceed its cwnd. SCTP was able to keep the B1 network, which can transfer more than 2*MTU bytes before the ACK returns on the first packet, busy long enough to allow the ACK to arrive. The amount of idle time for TCP during the first RTT was however only two tenth of a second and it was tiny in comparison to the total transmission time.

5.3.4 MS scenario analysis

This section analyzes the execution of the MS scenarios, which in opposite to the Basic scenarios use multiple streams and TCP connections. These multiple concurrent TCP connections make the analysis more complex, which sometimes makes it very hard to specify the exact effect of each issue. The complexity originates from the fact that each TCP connections have its own congestion control. For SCTP no such complexity arises since only one association, shared by all streams, was used. The reader should have read the topics mentioned in Section 5.3.2 before continuing further.

All the R1 result was quickly analyzed and the reason why SCTP was behind TCP in most MS scenarios was due to the implemented bandwidth sharing between the streams in SCTP, enforced by the test application. TCP, which have no such explicit bandwidth sharing, relies on that congestion occurs which will force the connection that loose packets to slow down and give room for other connections. The first initiated TCP connection

did initially use a majority of the available bandwidth and the following connections did struggle to acquire the bandwidth, due to the transmission delay generated by the first connection. The TCP connections did never evenly share the available network resources because the amount of data in the tests was too small. SCTP was in MS 8 on phase with TCP and this occurred because the data amount sent on each connection/stream was less than one MTU. Only one data packet was transferred on each connection/stream.

The R2 result turned out to be the most interesting and TCPs results have been hardest to analyze. The result for SCTP, showed in Figure 5.2, clearly says that SCTP was not affected by the amount of streams. Its speed remains almost constant and the simple reason is that there is no inherent overhead in using multiple streams. The TCP results are less obvious and the analysis below focuses on TCP and how it struggled to perform as SCTP.

MS 1

In this scenario, identical to Basic 3, TCP perform as SCTP. See the Basic 3 analysis in Section 5.3.3 for details.

MS 2

This scenario utilized the stream feature of SCTP and multiple TCP connection but no noticeable difference was seen between the two alternatives. The extra transmission delay of the second TCP connection was tiny in comparison to the total transmission time. It was not be regarded as a big issue, even when the second TCP connection experienced time-outs for its SYN and SYN-ACK packets in the GSM network. Read more about the transmission delay related to connection setup in Section 5.3.2.

Further, the extra aggressiveness of two TCP connections had no advantage over the single SCTP association in any of the network. The networks had a too low *Bandwidth*Delay*Product* (BDP) and the ACK packets arrived quickly enough to help the

single SCTP association to make sure that the network was fully utilized. The B1 network was the only network that gave two TCP connections an opportunity to use all of the bandwidth of the network more quickly than one SCTP association could. However, the opportunity was not big enough and the first TCP connection was able fully utilize the entire bandwidth when the second connection got ready and started sending data.

A negative effect of using two parallel TCP connections was observed in all networks. The two connections, which have their own congestion window, did together fill the buffer of the router. This resulted in that packets were dropped and retransmissions were performed. The test logs showed that packets were dropped twice in the experiments and each connection lost one packet. In both cases, TCP recovered with a fast retransmit and resumed with a reduced cwnd. However, the two connections did together make sure that the network was fully occupied and the networks were never in an idle state, even when one connection slowed down.

MS 3

The result from MS 2 showed that the extra aggressiveness of two TCP connections, compared to one SCTP association, had no positive impact on the throughput. In this experiment where five TCP connections used, which was prior to the experiment expected to be even more aggressive than two. The five connection had however no positive impact on the throughput and instead where more network resources used for transmitting non-user data, related to the initiation and the closure of the connections.

In the GSM network, the connections were forced to retransmit SYN-ACK packets during the initiation. Except the first connection, which had no problems, the connections sent their SYN-ACK packets in average 5,25 times. In total, the initiation together with the closure of the connections generated 2.4 KB of non user-data. This was however less than 1% of the total user data (300KB) so it did not have any noticeable impact on the R2 result. In the other networks where there no retransmission of SYN-ACK packets.

Significant for this scenario was that the load on the network occasionally was far higher than the capacity of network. The five TCP connections did together generate a high amount of outstanding data, caused full buffers in the router and consequently dropped packets. As in MS 2, the connections that lost packets slowed down but the TCP connections did together send at a high enough rate even during congestion and the network was never in a idle state.

MS 4

With ten TCP connections in the GSM network the amount of retransmissions of SYN-ACK packets increased and they began to make a noticeable difference. The total amount of none-user data sent from the server during initiation was 6,8 KB, which is 20 times more than what SCTP needs for one association. In the other networks were 2,1 KB of none user-data transferred.

The router buffer was in contrast to MS 2 and 3 never full in this experiment, which actually indicates that the ten TCP connections are less aggressive than two and five connections in MS 2 and 3. One factor for the aggressiveness turned out to be the initiation of the ten connections. The sequential initiation of the connections, which was used due to its robustness, and the length of each connection made less connections active in parallel. The length of each connection, defined by the amount of user-data sent with the connection, was in this scenario only 30 KB. This mades each connection finish earlier and therefore was less connection active in parallel.

The limited amount of data was found as another factor, which hindered each connection to reach a high amount of outstanding data. The amount of outstanding data of a connection is increased as the data are delivered, see slow-start and congestion avoidance algorithms, and with less data the connection reached a lower amount.

MS 5

In MS 5 each TCP connection only transmitted 10 KB of data, 300KB spread over 30 TCP connections. 10 KB of data only requires eight data packets and the connection was now finished after four burst of two data packets. It was observed that the network buffer contained fewer packets than MS 4 throughout the entire transmission.

Further when the TCP sessions are short will less connections transfer in parallel. A result of this was seen when studying the buffer of the network router, which have not been buffering packets to the same extent as in MS 1 ,2 ,3 and 4. This did shorten the delay that the SYN and SYN-ACK packets experienced and the amount of retransmissions of SYN-ACK packets decreased. In the GSM network was SYN-ACK packets still on average sent 5 times and the total amount of non-user data sent reached 15 KB. 15 KB is 5 % of the user-data and the cost of multiple TCP connections are now getting substantial in the GSM network. 6,4 KB of none user-data was transferred in the other networks.

Another consequence of fewer packets buffered in the network was that no packets were dropped due to congestion. This indicates that the network might be under utilized, but the logs shows that the network was fully utilized during the entire test. One can also see that MS 5 was not much slower than MS 1 in any network.

MS 6

With fifty TCP connections, the overhead from the initiations and closures became visible in the graph for all networks. The overhead was 18 KB in the GSM network, where retransmissions of SYN-ACK occurred two times per connection, and in the other networks 11KB. This is 6 % respectively 3,5 % of the 300 KB of user data.

The MS 6 result from the B1 network did however worsen more than the rest of the networks so there was some other parameter that affected the result. This was investigated and the network was found being regularly idle during the transmission. The network was delivering data packets faster than the TCP connections were putting data packets on the

Scenario name	GSM	Modem	B1	B2	B3
MS 6	0%	0%	16%	0%	0%
MS 7	13%	12%	45%	26%	29%
MS 8	35%	27%	69%	48%	51%

Table 5.4: Idle time percentage for TCP in MS 6, 7 and 8

network. The sequential initiation of the connection now separated the connection too much to allow the short TCP sessions to overlap enough so that network could be fully utilized. The idle time percentage of the total time was measured and can be seen in Table 5.4. Read to Section 5.3.1 for more details on how this is calculated.

MS 7

The overhead from multiple TCP connections continued to grow when 100 connections were used. The overhead was now 32 KB in the GSM network and 21 KB in the other network. This overhead did not alone explain why TCP needed more time to transfer the data, compared to MS 6.

The analysis discovered that TCP had the same efficiency problem that was seen with MS 6 in the B1 network, but now in all networks. The reason was the same but it was now worse due to that the each TCP connection was now even shorter, the amount of data was now only 3 KB.

MS 8

With 300 TCP connections the overhead created by the initiations and closures grew further compared to MS 7. No retransmission of SYN-ACK packets occurred in the GSM network and all networks experienced the same header overhead, 63,6 KB of non-user data. The overhead increase could not alone explain why the R2 result has increased with 74%, 63%, 270%, 127% and 131% in respective networks compared to MS1 in the same network.

The idle periods did compared to MS 7 occur with shorter intervals and even with

longer periods and it can be concluded that the efficiency problem has worsened from MS 7. The idle time is now responsible for a large part of the R2 result, from 27% to 69% of the total time in the different networks.

5.3.5 Summary

The network configurations that were chosen were the biggest reason on why TCP struggle with its R2 result in the MS 2, MS 3, MS 4 and MS 5 scenarios. We saw in MS 2 that two TCP connections failed to give TCP a boost in the positive direction in any network. Using more TCP connections did not improve TCP situation. The networks forced TCP to show all the bad things with using multiple connections. Such things are the overhead of initiating multiple connection, idle networks and time-outs with retransmissions. The GSM network was the network that showed to be the least suitable for concurrent connections. A comparison between SCTP's and TCP's R2 result is available in Figure 5.5, containing all scenarios and network combinations. The figure shows SCTP's result as a percentage of TCP's result in the same scenario.

The sequential initiation of TCP connections was identified as the main reason why some network capacity was left unused in MS6, MS7 and MS8. However, initiating the connection in parallel might have its own issues and perform equally bad. For more details on the initiation of the TCP connections read Section 4.3.4.

The negative issue observed for SCTP is the extra transmission delay due to the cookie mechanism of SCTP. Otherwise SCTP has showed itself to be more efficient than TCP, but not by much, see the results from MS 1, 2, 3, 4 and 5. The use of parallel TCP connection did not increase the overhead so that it becomes significant until 50 or more connections were used to transfer 300KB of user data.

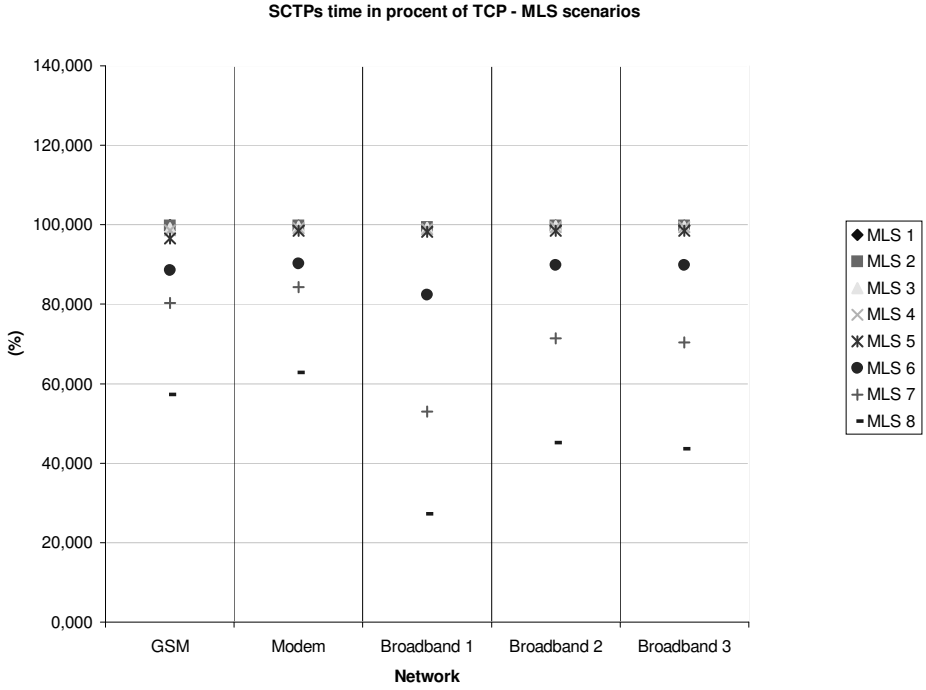


Figure 5.5: SCTP R2 result in percentage of TCP

5.4 Conclusion

The analysis method is regarded as accurate and helped during the study of concurrent connections initiating, transferring data and closing. A number of parameters have been identified that affect the performance, most for TCP but also some for SCTP. The scenarios that were chosen also gave an acceptable perspective on the performance of TCP and SCTP when using a varying amount of connection/streams and data. The results did sadly not show that multiple TCP connections can make use of the available bandwidth faster than a single TCP or SCTP connection, which was as described in Section 4.1.1 expected. However, the network logs from the scenarios indirectly confirmed it. One could see that TCP in MS 2, 3 and 4 tried more aggressively to make use of the available bandwidth than the single SCTP association did.

The MS scenarios were designed to mimic HTTP1.0 with either SCTP or TCP but it is difficult to say if a user regards HTTP1.0 with SCTP as faster than TCP. The results show that SCTP finished its first stream together with all other streams during a short interval at the end of the entire transmission. TCP did on the other hand finish its connections over a larger interval and therefore were many of TCP's connections finished before the corresponding SCTP streams. A user of HTTP1.0 might therefore regard HTTP1.0 over TCP faster than HTTP1.0 over SCTP due to that some files are retrieved faster.

In general, the results strongly indicate that an application, which uses TCP as its transport protocol, will reach the best throughput performance if it uses long TCP sessions instead of short sessions as in test scenario MS 6, 7 and 8. Such a recommendation is also valid for SCTP, which almost have an identical congestion control as TCP. Longer sessions make the header overhead from the initiations become less significant. However, the most important reason why long sessions are preferable is that all transmission resources are more likely utilized if the congestion control algorithms are allowed to work longer.

The tests does not cover all the networks used for Internet communication but it is believed, by the author, that one can now with the knowledge from this study extrapolate

what will happen if we go outside the domain of the scenarios in this study.

Chapter 6

Summary

This chapter summarizes the achievements of this study and look at possible future work. The chapter finishes with the authors final thoughts.

6.1 Accomplishments

There were two major implementations of SCTP available when this study started early in year 2002. The Siemens implementation was regarded at that time as the best and most active project. This implementation had though a number of bugs that hindered this study for some time. When looking back at the decision the Siemens implementation is still seen as the right choice, much because of its socket API implementation.

The stream feature of SCTP was in advance identified as an appropriate feature to test in this performance study. It was picked for the experiment after some other possibilities were examined. The implementation of the test software for TCP and SCTP was successfully performed, but some issues were discovered when verifying the results of the experiment. The issues were resolved and at last were the TCP and SCTP version regarded equivalent and the R1 and R2 results from both versions could be used in the comparison.

A test environment was successfully built and several scenarios were designed to test

SCTP and TCP against each other. The environment has worked fine and the analysis has found many interesting factors that had an effect on the results. However, the experiment specification was found falling short of the expectations. It turned out that the network never allowed TCP to use the potential it had when using multiple connections. A more wide spread set of network configurations would have given a richer view of the differences between using one SCTP association instead of multiple TCP connections.

This document describes the experimental experience gained on SCTP and the knowledge about SCTP. Chapter 2 summarizes the knowledge gained before and after the execution of the experiment, which verified our expectation and also revealed new information that was not known, as the slop-over rule.

6.2 Future work

This study on SCTP is narrow and there are many new studies on SCTP that can be done. This section lists what the author has identified during this study as potential continuations to this study.

6.2.1 A continuation with HTTP1.0

Goal 2 not completely satisfactory fulfilled in this study, as mentioned in section 6.1. By designing new test with different network configurations one could find out more about the performance of HTTP1.0 with SCTP. New network configurations should be chosen from a new survey of networks. A network configuration with a higher bandwidth delay product can verify whether TCP can benefit from its concurrent connections. The characteristics of typical HTTP1.0 real world transfers should be identified with more details, which would strengthen the research and help make stronger conclusions.

We also need to check if the use of concurrent TCP connections is suitable on Internet and identify a limit on how many that is suitable to use. Many TCP connections are more

aggressive and might have a negative impact on concurrent network traffic. The use of multiple TCP connections is controversial and seen by some as TCP-unfriendly[7].

6.2.2 HTTP1.1

The problem with HTTP1.0 is that it sometimes use short TCP sessions for short data streams and this is addressed by the successor of HTTP1.0. HTTP1.1 has a concept called persistent connections[11], where different data streams share a single TCP session. Instead of tearing down the TCP connection after each data object could the connection be reused, which results in longer TCP sessions. This allows the congestion algorithms to receive more data about the state of the network and multiple initial slow-starts can be avoided. The TCP connection will have a better chance to stabilize itself with an accurate value of cwnd.

HTTP1.0/SCTP has also addressed this problem and several data streams can share an association by using its streams. This has the same positive effects as HTTP1.1/TCP, but with SCTP comes also more positive effects. SCTP delivers the different streams in parallel in opposite to HTTP1.1/TCP, which delivers the streams in a sequence. The sequential delivery is vulnerable to HLB and SCTP has therefore a possibility to perform better.

6.2.3 Signalling traffic

Tests with telephony signalling applications are also very interesting to perform, due to that signalling was initially the main application for SCTP. To test signalling one needs to collect information about the traffic characteristics of signalling in an IP network. A possible opponent to SCTP for transporting signalling traffic over IP networks is unknown right now, but it would be very interesting to study if SCTP can meet the demands from signalling applications.

6.3 Final thoughts

SCTP is new and there was not much material published about it in the beginning of this study. On the IETF web page, one could find the RFC and a few other documents on SCTP. Several of these were drafts on specifying extensions and corrections to the proposed standard. The book Stream Control Transmission Protocol[23] has given me many insights in the subject. All together, I believe I have found the information I needed for this study.

Analyzing tests with multiple TCP connections active at the same time turned out to be far harder than I expected at the start. In the beginning, it was even hard to see what was happening with only one connection active, but as I gained experience I learned what to look for. This is believed to be an important experience for future experiments that might contain many concurrent flows.

Performing research in the form of practical experiments was new for me and some things have naturally gone wrong. When I chose between the different APIs that were available in the Siemens implementation of SCTP I first chose the SCTPd API. This decision put me on a wrong path and I had to return after some time and restart. The decision to restart with the socket-API was done after a discussion with the crew assigned to the Siemens implementation and naturally should this discussion have been made earlier. Regarding the experiment design, I have also made a mistake that I regret. The network configurations, chosen for the experiment, were not as different as I believed in the beginning. All networks had similar bandwidth delay products and this hindered me to see all aspects of using many concurrent TCP connections.

However, I still believe several interesting aspects have been found in this study.

References

[1] Ethereal, a network protocol analyzer. *http://www.ethereal.com/*.

[2] SCTP, a prototype implementation. *http://www.sctp.de/*.

[3] Katarina Asplund, Anna Brunstrom, Johan Garcia, and Karl-Johan Grinnemo. Analysis and implementation of a partially reliable transport protocol for multimedia applications: Project report. *Project Report, Karlstad University*, Dec 2000.

[4] T. Berners-Lee, R. Fielding, and H. Frystyk. RFC 1945: Hypertext Transfer Protocol HTTP/1.0, May 1996. Status: INFORMATIONAL.

[5] P. Conrad, G. Heinz, A. Caro, P. Amer, and J. Fiore. SCTP in battlefield networks. *Proceedings MILCOM 2001, Washington, DC*, October 2001.

[6] Tom Dunigan. Tcp auto-tuning zoo. *http://www.csm.ornl.gov/ dunigan/net100/auto.html*.

[7] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, August 1999.

[8] J. Garcia and J. Gustafsson. Data communication in GSM networks. *Project Report, Karlstad University*, 2000.

[9] Johan Garcia. Integrated testing - test setup. *Working Report, Karlstad University*, October 2000.

[10] M. Allman BBN/NASA GRC. Tcp congestion control with appropriate byte counting (abc). *http://www.ietf.org/*, page 10, February 2003.

[11] James F. Kurose and Keith W. Ross. Computer networking: A top-down approach featuring the internet. *Addison Wesley*, page 688, 2001.

[12] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. RFC 2018: TCP selective acknowledgment options, October 1996.

[13] Shawn Ostermann. Tcptrace. *http://www.tcptrace.org/*.

[14] Avi Silberschatz, Peter B. Galvin, and Greg Gagne. Operating system concepts, sixth edition. *John Wiley & Sons, Inc*, page 803, 2002.

[15] W. Stevens. RFC 2001: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, January 1997. Status: PROPOSED STANDARD.

[16] W. Richard Stevens. UNIX network programming, volume 1: Networking APIs - sockets and XTI. *Prentice Hall PTR, Upper Saddle River, NJ*, 1998.

[17] Randal Stewart, Lyndon Ong, Ivan Arias-Rodriguez, Kacheong Roon, Phillip T. Conrad, Armando L. Caro Jr., and Michael Tuexen. Stream control transmission protocol (SCTP) implementers guide - work in progress. *http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-sctpimpguide-07.txt*, page 62, October 2002.

[18] Randal Stewart, M. Ramalho, Q. Xie, M. Tuexen, I. Rytina, M. Belinchon, and P. Conrad. Stream control transmission protocol (SCTP) dynamic address reconfiguration. *http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-addip-sctp-06.txt*, page 36, September 2002. Work in Progress.

[19] Randall Stewart. Stream control transmission protocol (SCTP). *http://www.sctp.org/*.

[20] Randall Stewart. RFC 2960: Stream Control Transmission Protocol. October 2000.

[21] Randall Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad. SCTP partial reliability extension. *Work in progress, draft-stewart-tsvwg-prsctp-00.txt*, May 2002.

[22] Randall Stewart, Q. Xie, L Yarroll, J. Wood, K. Poon, and K. Fujita. Sockets API extensions for stream control transmission protocol (work in progress). *IETF draft*, May 2002.

[23] Randall Stewart and Qiaobing Xie. Stream control transmission protocl (SCTP) - a reference guide. *Addison Wesley*, November 2001.

[24] Guanhua Ye, Tarek Saadawi, and Myung Lee. SCTP congestion control performance in wireless multi-hop networks. *MILCOM*, 2002.

Appendix A

List of Abbreviations

BDP	Bandwidth Delay Product
CARL	Communications Applied Research Laboratory
cwnd	Congestion window
DISCO	Distributed System Communication
HLB	Head of Line Blocking
HTML	HyperText Markup Language
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
MAC	Message Authentication Code
MS	Multiple Streams
MSS	Maximum Segment Size
MTU	Maximum Transfer Unit
NIC	Network Interface Card
OOTB	Out of the blue
PRTP	Partial Reliable Transport Protocol
rtt	Round Trip Time
rwnd	Receiver Window
SACK	Selective Acknowledgement
SCTP	Stream Control Transmission Protocol
ssthresh	Slow Start threshold
TCP	Transmission Control Protocol

UDP User Datagram Protocol

ULP Upper Layer Protocol