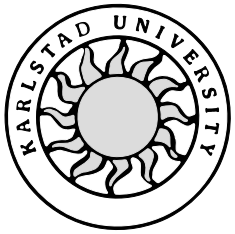


Department of Computer Science

Johan Eklund

Evaluation of Emulab as Experimental Platform by Comparing TCP and SCTP

D-level Thesis

2004:07

Evaluation of Emulab as Experimental Platform by Comparing TCP and SCTP

Johan Eklund

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.

_____

Johan Eklund

Approved, 2004-12-10

_____

Advisor: Katarina Asplund

_____

Examiner: Donald F Ross

Abstract

To be able to evaluate new protocols it is important to have access to good experimental environments. Several experiments are needed to verify different aspects of protocol performance as well as robustness under various network conditions. Emulab is a new public experimental platform which is available for remote users. The ambition is that Emulab should offer the user the possibility to perform both simulation and emulation of a network. In addition, Emulab offers access to an experimental live network. This thesis presents a study where a series of tests are performed on the Emulab platform and also gives an introduction to SCTP. The first objective of the thesis is to obtain practical experience and to evaluate the usability of Emulab and the second objective is to compare the throughput between the transport protocols TCP and SCTP. The experiences from using Emulab are very positive. The results show that Emulab is a reliable and robust platform with high availability. The throughput comparison did not reveal significant differences between SCTP and TCP under moderate traffic load. Further tests and analyses are necessary to obtain a clear view of the situation in a heavily loaded network.

Acknowledgements

First I would like to thank my supervisor Katarina Asplund for good supervision and encouragement during this drawn out work. I am also thankful to Annika Wennström, Anna Brunström, Johan Garcia, Hannes Persson, Karl-Johan Grinnemo and Stefan Alfredsson in the Distributed System and Communication Research Group for helpful answers to many of my questions and for encouraging comments during the work.

Last but not least, I want to thank Torbjörn Andersson for giving me great support in how to solve many of the technical challenges that have turned up during this project and also for providing me with some of the application programs used in the experiment. Without his support I would have spent several nights and days finding solutions to technical problems.

Contents

List of Figures

Abbreviations

Association - A relationship established and maintained between two communicating peers. An association in SCTP is semantically equivalent to a connection in TCP

Chunk - The SCTP datagram is comprised of a common header and chunks. The chunks contain either control information or user data.

Firewalling - Setting up rules to protect your system from specific traffic

IETF - Internet Engineering Task Force. An open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture.

MTU - Maximum Transmission Unit. The largest size of IP datagram which may be transferred using a specific data link connection

Multistreaming - In SCTP the term is used for the possibility to have several independent logical streams in the same association for transfer of data or control information between endpoints

Multihoming - Assigning more than one IP network interface to a single endpoint. This means that a connection endpoint may have alternate IP addresses.

NFS - Network File System. A protocol for sharing files over a network

RTO - Retransmission Time Out. A counter which purpose is to differentiate the cases where

- acknowledgment is delayed to due random delay fluctuations
- network is congested and the sent segment has been lost

SCTP -Stream Control Transmission Protocol. A reliable transport protocol with partial ordering of data.

SIP -Session Initiation Protocol. An application protocol used to set up, tear down and negotiate options for a session

Stream - In SCTP: a sequence of user messages that are to be delivered to the upper-layer protocol

1 Introduction

Before a new transport protocol can come into general use, the new protocol is usually subject to several experiments. The requirements on a protocol may concern reliability, throughput, robustness, and fairness to competing traffic. To perform these experiments, it is possible to test the protocol in a network emulator, a network simulator or in a live network. All these approaches have advantages as well as limitations.

A new experimental environment for networks and distributed systems, Netbed [23], has been available since the year 2000. The group responsible for Netbed claims that it is an environment that integrates the three approaches above and therefore makes it easier, faster and less costly to perform experiments and to get valid results compared to traditional testing in the separate environments. One part of Netbed is a cluster of computers connected to a network called Emulab. Emulab is free to use for research purposes and it is run remotely using a web interface on the Internet. Emulab has not previously been used by the *Distributed System Communication Research Group (DISCO)* in Karlstad.

For many years, the two transport protocols Transmission Control Protocol (TCP) [15] and User Datagram Protocol (UDP) [13] have served applications as transport providers in a sufficient way. However, as computer networking has evolved, the demands on the service provided by the network protocols have become more divergent.

Today, neither TCP nor UDP fulfills the requirements of, for instance, applications like IP-telephony signaling. This type of application is not very sensitive to small data losses, but instead extremely sensitive to delays. However, IP-telephony signaling needs a basic level of reliability; otherwise the application will not be useful.

To meet the demands from the new type of applications, new transport protocols are being developed. The Stream Control Transmission Protocol (SCTP) [19] is one of these transport protocols that claims to meet some of these new demands. The goal when developing SCTP has been to present a protocol that offers reliability and high throughput but only partial ordering. A transport service designed this way would be a better solution for IP telephony signaling applications as compared to one using TCP and UDP.

1.1 Objectives

The objectives of this thesis are twofold. The first objective is to evaluate the benefits and drawbacks from using the remote experimental platform Emulab as a base for experiments with network applications and protocols. The second objective is to undertake a performance study of SCTP. The aim is to compare the throughput of SCTP and TCP.

1.2 Disposition

The rest of this thesis is structured as follows. Chapter 2 presents background information on the Netbed and Emulab platforms. It gives an overview on how the platforms are designed, configured and how they are to be used by remote users. Other public research labs are also briefly presented. Chapter 3 gives an introduction to the transport protocol SCTP, and presents an overview of how it is designed. Moreover, the chapter covers related research performed on SCTP and experiments performed on Emulab. Chapter 4 describes the experimental design. It begins with a specification of the aim of the study. The scenario, which has served as inspiration for the experiments is then presented as well as the hardware and software required to perform the experiment. Furthermore, the chapter presents the different parameters used. In chapter 5, the results of the experiments with SCTP and TCP are presented as well as comments and analyses. Chapter 6 presents the experiences from using the remote lab Emulab as platform for the experiments, and benefits and drawbacks are stated. Finally, in Chapter 7, conclusions from this study are drawn and areas for further studies are proposed.

2 Netbed

This chapter first introduces and gives an overview of some experimental environments available for testing new protocols. After this, background information concerning Netbed and Emulab is given.

2.1 The experiment procedure for a new protocol

All new protocols should be exposed to several tests to verify different aspects of the performance under various network conditions as well as robustness and fairness to other traffic, before being accepted as generally used protocols.

Traditionally, there have been three experimental environments supporting computer networking research. These are live networks, network simulators, and network emulators. Each of these environments has different advantages as well as drawbacks. Live networks naturally offer the most realistic experimental environment, but it is difficult to control the tests in this environment. That is, it is almost impossible to repeat the same network conditions twice. Therefore, different types of artificial solutions are often used. The most frequently used approaches are simulation and emulation.

2.1.1 Simulation

Simulation is often used by network researchers. Simulating a protocol is a way to test the behavior of a protocol artificially by using software acting as, i.e. simulating, a protocol in a real network. The advantage of simulation compared to experiments in a live network is that it is faster, that the experiments are repeatable and that the environment is controlled. For example, the user may change one parameter and then run the simulator again to rapidly obtain and analyze possible differences. By using simulation, the researcher obtains basic information concerning protocol behavior. The drawback of using a simulator compared to a live network is the loss of realism. Since simulation often makes simplifying assumptions of both the protocol and the network the results may be affected.

2.1.2 Emulation

It is often interesting to try to imitate a large network in an experimental lab by using a small experimental network, a network emulator. This could be done by inserting an extra node on a link between two hosts and on this node run software which makes the link look like a real network with bandwidth constraints, losses and delays. The software Dummynet [16] is commonly used for these purposes. By using Dummynet it is possible to set, for example, bandwidth constraint, delay, loss rate and router buffer size.

Emulation is often a more realistic way to test a protocol compared to simulation since a real protocol (sender and receiver) is used over a network. Nevertheless, also emulation is a simplification of reality and therefore also emulation results must be interpreted carefully.

In comparison to simulation, emulation takes more time to run and the tests are also more limited, i.e. it is possible to simulate situations that cannot be emulated.

2.1.3 Conclusion

There are both benefits and drawbacks with both simulation and emulation. When making use of a simulator it is possible to achieve a controlled, fast and repeatable environment. However, possibly the simulated situation is not very realistic, since using a simulator often implies simplifications. Using an emulator is a more realistic approach to obtain a view of how a protocol performs in a network, but emulating a network usually requires quite some time for manual configuration of the system.

To achieve information from as many aspects as possible, researchers sometimes make use of these two approaches. To get a proper view of the application or network protocol performance the information from the simulated and emulated situation also has to be verified by information from a real situation in a live network.

2.2 Netbed and Emulab

Netbed is a public research platform that intends to integrate the three experimental environments mentioned above, in order to streamline the evaluation of network scenarios. The use of the platform is free of charge for authorized users.

Netbed was founded in 1999 when a prototype for a large cluster of computers, Emulab, was compiled. After a time, it was made public to remote researchers via a web interface. Emulab was primarily intended to be an emulation platform, but there is no restriction against running a simulation on a machine in the test bed as well. Over time, Netbed has evolved and now also an experimental wide area network is available. This network consists of computers

in different parts of the world connected to the Internet, especially dedicated to research activities and running a special configuration. This network offers the ability to use a live network under controlled forms [23].

Netbed is intended to be an experimental platform available for users from all over the world. The intention is to let researchers, research groups as well as companies use the platform for performing their own experiments [10]. The ambition of the founders[1] was also to integrate the three test approaches, for computer networking research. Three goals were set up when designing the platform:

- "Ease of use". By using a web interface and also a Java GUI for users to allocate resources and to configure and run experiments
- "Control". An authorized user gets full control of the nodes in the allocated network during test performance
- "Realism". By offering both emulation, simulation and wide area facilities

Netbed consists of two parts. The first part is a number of computer clusters. Originally there was only one cluster called Emulab, situated at the University of Utah (168 PCs) [26]. Over time the cluster was cloned and today there exist two other clusters controlled by Netbed. One of them is situated at the University of Kentucky (48 PCs) [25] and the other at Georgia Institute of Technology (40 PCs) [24]. These clusters are configured in the same way as the Emulab cluster and are controlled by Netbed staff. Emulab is the cluster primarily intended for external users, but also the cluster in Kentucky may be used by externals, although it is used primarily as a teaching aid. The cluster in Georgia is used only for classes and for research purposes, not for external users.

The second part of Netbed is a distributed system of different test beds and separate nodes contributed by different organizations. This system is dynamic and nodes may be added and withdrawn dynamically from the system by the owners. All nodes in this network run a special UNIX configuration. This network is called PlanetLab [30] .

The scope of this thesis includes evaluation of only one part of Netbed, the public cluster of computers situated at the University of Utah, Emulab. Therefore, subsequent parts of this thesis concerning Netbed will focus on Emulab.

---

[1] www.cs.utah.edu/flux/index.html

2.2.1 Hardware

Emulab is composed of 168 PCs. Each computer has five 100 Mbps Ethernet network interfaces four of which are available for network project setup. The fifth network interface is used as a control interface to regulate access and communication to the outside world.

A server called userhost and a database are connected to the system. These components are used for starting up new experiments and for storing project information for the users. The user and experiment directories are exported via NFS to the nodes allocated for an experiment.

There is also a master server in the network, called masterhost. This server is not directly accessible by test bed users, but is used as a web server and as a name server, among other things.

All computers (nodes) in the network are connected to switches that function as a programmable panel. This means it uses Virtual LAN-technology (VLAN) [35] for isolating arbitrary topologies and for granting security to the different projects. An overview of the entire Netbed is shown in Figure 2.1. The figure focuses on Emulab, but the connection to the wide area network, PlanetLab, is also visible in the figure.

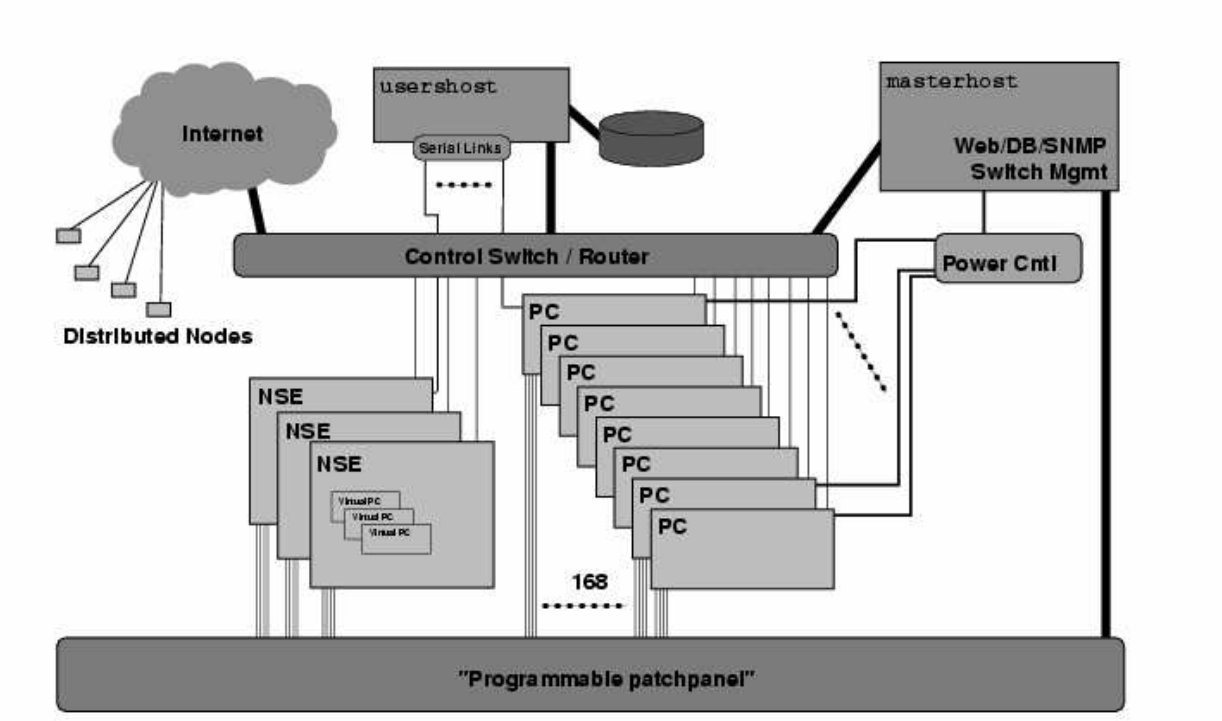


Figure 2.1 An overview of the Netbed architecture

2.2.2 Software

All the computers are able to boot on FreeBSD 4.7 and RedHat Linux 7.3 but the users are also free to load an operating system of their own. This thesis only covers usage of the default operating systems.

For emulation purposes Emulab uses Dummynet to set bandwidth constraints, loss rate, queue size and delay. The insertion of a node running Dummynet (to "shape the link") is transparent to the user.

If a user wants to use simulation, Emulab uses ns's[2] simulation facility nse [6]. Running several simulated nodes, links and traffic on one physical node gives the possibility to test scalability beyond the number of the nodes in the experimental network.

2.2.3 Access

To start experimenting on Emulab the user needs a user account. This account covers a project, and is requested for by a principal investigator of the project. The Emulab approval committee[3] is the authority that approves or rejects the request.

After approval, the requesting researcher is set as the person responsible for all activities performed in the project. This person also acquires authorization to accept new members of the project. In this way, the administrative load for the Emulab staff is kept low. After authorization, any researcher (accepted by the person responsible for the project) in the group is free to start as many experiments as is requested. Each project and each authorized user has a directory for storing files needed for the experiment.

2.2.4 Using the platform

Any authorized user may run an experiment. The Emulab administrative governance has set up a few administrative policies concerning acceptable use and referencing [31]. These policies are very general and they prevent serious misuse of the platform. If the user follows these policies there are no restrictions for using nodes in the cluster.

An experiment is created by using the web interface. The user sends information about the topology and the parameters of the network wanted for the experiment. Information about topology is submitted via an ns-script. This script could either be written in an editor, or it could be created using a Java GUI, called NetBuild. This GUI makes it possible to draw a

---

2 Network Simulator is a discrete event simulator targeted at networking research

3 testbed-approval@flux.cs.utah.edu

picture of the topology of the network including nodes, links, LANs, bandwidth constraints, loss rates and delays as well as naming of both nodes and links.

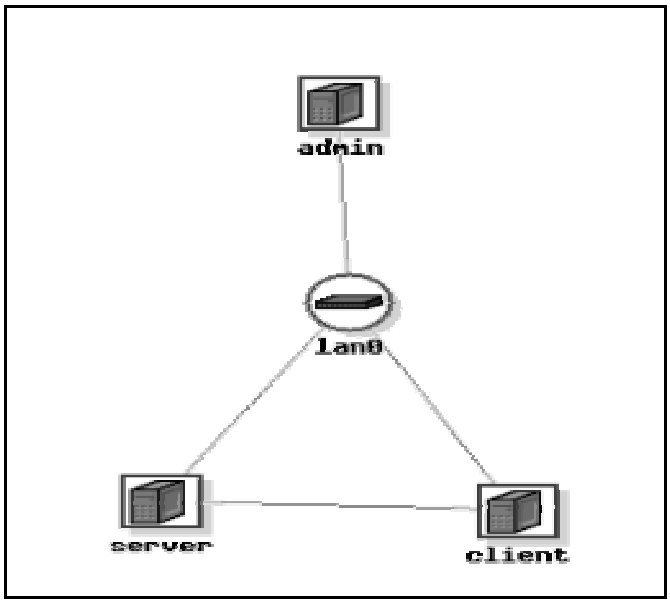


Figure 2.2 Graphic image of network for an experiment created by NetBuild

After processing this drawing, NetBuild produces a valid ns-script[4], usable for creation of an emulated network topology similar to the drawing. The topology shown in Figure 2.2 will generate the ns-script shown in Figure 2.3.

```
#generated by Netbuild 1.03
set ns [new Simulator]
source tb_compat.tcl

set server [$ns node]
set client [$ns node]
set admin [$ns node]

set testlink [$ns duplex-link $server $client 0.4Mb 25ms DropTail]

set lan0 [$ns make-lan "$server $client $admin " 100Mb 0ms]

$ns rtproto Static
$ns run
#netbuild-generated ns file ends.
```

Figure 2.3 The ns-script generated from the NetBuild-picture in Figure 2.2

[4] The NS script may be used the way it is generated, but it can also serve as a base and can be complemented by instructions for routing and automatic load and start of applications.

After submission of information about the requested project, all information is stored in the userhost database. A request may also contain information about swap ability and running of the experiment in batch mode. Allocation of resources and configuration of the experiment is performed automatically by Emulab after obtaining the requested information. No staff is needed for the processing.

After processing of the request, the user receives an e-mail informing him or her whether the setup of the experiment was successful or not. The request may be rejected or postponed if there are not enough free nodes available to construct the network. If the response is positive, the e-mail contains a listing of the nodes, DNS-names and IP-addresses to the nodes allocated to the project (see Appendix A). The user is then free to use the network.

Normally, the user allocates resources for the experiment immediately before the experiment is conducted. A different way of setting up an experiment is to request it to be set up in batch mode, i.e. that the experiment is run when enough nodes are available. This time may be immediately or sometime in the future. The batch system is then responsible for setting up and removing the experiment. Running the experiment in batch mode requires an extension in the ns-file that handles automatic load and start of programs as well as startup and completion information. This mode is beyond the scope of this thesis.

The user is free to log on to any node allocated to the experiment using secure shell (ssh), and to load and run any program. The user has full privileges over the node and may customize the configuration in any way as well as reboot the node. After experiment completion the user has to terminate the experiment (if it is not run in batch mode or in swap mode). After termination the user is requested to return all allocated resources to Emulab.

After any change concerning the allocated resources the user will receive an e-mail that informs him or her about the new network status. The only control performed by the Emulab is the network utilization status. If the experiment is found to be idle for a specified time period this may lead to automatic project swapout[5] and the resources are automatically returned to Emulab. The user may also manually swap a running experiment in or out by using the web interface.

---

[5] An automatic experiment swapout does not mean loss of experiment information. All swapped out data is saved in the Emulab database and may be swapped in again. Data stored locally on the allocated machine is lost when the experiment is swapped out.

2.2.4.1 Performing an experiment

The nodes allocated for the experiment are under full control by the user as the experiment is running. This control gives the user the ability to run programs to generate and analyze traffic through the network. Since the user has full privileges and the possibility to connect to every node in the allocated system it is now transparent that the experiment is performed remotely. The intention is that the user should be using the constructed network as if it was situated in a local lab.

The easiest way of storing data generated from the experiment is to store it on a specific node and then to copy it to a local machine after experiment termination.

2.2.5 Security

As a public network intended to serve as a research platform, Emulab might be attractive to attackers. An attacker might, for example, be interested in disturbing the current experiment or in stealing information concerning a special experiment. It is also most essential that the lab is available to the researcher whenever he or she wants.

To prevent attacks, Emulab blocks all ports below 1024 (with exception of port 20, 21 for ftp, port 22 for ssh and port 80 for http). This blocking is done for the protection of the experimenters, as well as to ensure that an errant application cannot become the source of a Denial-of-Service attack on sites outside of Emulab. This is done by firewalling traffic from outside the present experimental net.

To be able to control the use of Emulab, every user must have a personal identity and login name. Secure Sockets Layer (SSL) [32] is used to protect submitted data. SSL is a protocol that encrypts data as it is transferred across the Internet.

Emulab makes use of user and group mechanisms in UNIX to provide protection between users and projects. Each new user obtains a new Unix UID, and each new project obtains a new Unix GID. Users are members of the UNIX groups that correspond to each of the projects they are working on, and thus may share files with other members of those projects. The default directory permissions are set so that project files are not readable by members of other projects.

3 Stream Control Transmission Protocol (SCTP)

In this chapter the Stream Control Transmission Protocol (SCTP) is presented. The motivation for development of this new protocol is first discussed. Further, the different features of SCTP are described. Finally, a motivation for the choice of SCTP implementation for this project is presented.

3.1 History of SCTP

The process of developing the Stream Control Transmission Protocol (SCTP) [19] was started in 1997. At that time the protocol was intended to run on top of UDP. It was then called the Multi-Network Datagram Transmission Protocol (MDTP) and it was intended to be a protocol for telephony signaling transport. The requirements from telephony signaling traffic were that the transport service provided should offer reliability and high throughput, but only partial ordering. After submission to the Internet Engineering Task Force (IETF) [33], the work with MDTP was incorporated in the Signaling Transport working group (SIGTRAN) in the IETF. Over time, the scope and functionality of the protocol expanded, and therefore the design and the name of the protocol was changed. The name was changed to SCTP and it expanded from being a protocol running over UDP to be a general purpose transport protocol running directly over IP, not only serving telephony signaling.

After a few years of designing and reviewing, SCTP became an IETF Proposed Standard and was published as RFC 2960 in October 2000. The still continuing work on SCTP is now performed in the Transport Area Working Group (TSVWG) in the IETF [36] .

3.2 Motivation for SCTP

The original ambition of the founders of SCTP was to develop a transport protocol to serve the demands from telephony signaling applications. These applications need:

- full reliability
- stable connections between end points
- only partial ordering of messages

Neither of the traditional transport protocols, TCP and UDP could fulfill all of these demands; therefore the development of a new network protocol, SCTP was started.

3.2.1 Shortcomings of TCP according to IP-telephony signaling

TCP provides reliable data transfer and strict order-of-transmission delivery of data to the application. All data is sent as a stream of bytes between the sender and the receiver. However, for telephony signaling only partial ordering of data is necessary. The strict sequence number delivery in TCP could be a problem, since it can cause delays. Loss of a single TCP segment may block delivery of all subsequent data until the lost segment is retransmitted. This situation is called head-of–line blocking [21].

To achieve stabile connections between endpoints, it is necessary to introduce path-level redundancy. TCP is not designed for path level redundancy, since it does not support multi-homing, which SCTP does. This implies a problem; if one part of the network breaks down the connection will be unusable.

3.2.2 Shortcomings of UDP according to IP-telephony signaling

The most critical drawback regarding UDP is the lack of reliability. Further, UDP is not connection oriented, which is a requirement for IP telephony signaling. Nevertheless, UDP has some advantages over TCP, since it is faster than TCP due to no extra delay for the connection procedure, and, furthermore, UDP introduces no extra delay for retransmissions and for ordering before delivery.

3.2.3 SCTP components

There are many similarities between SCTP and TCP, but also a number of differences. An overview of the functions of an SCTP endpoint is seen in Figure 3.1.

SCTP has inherited the congestion and flow control mechanisms from TCP. Like TCP, SCTP is connection oriented and fully reliable. However, the connection is established a bit differently. Some TCP implementations use the selective acknowledgement (SACK) mechanism [11]. When using SACK, the receiver sends back an ACK informing the sender of all the data that has been received both in order and out of order. The sender may then retransmit only the missing data segments. The SACK mechanism has been adopted and extended by SCTP. When using SCTP, SACK is the only option.

What is new in SCTP is the support for multihoming, the concept of associations and multistreaming. Other specific features in SCTP are the message boundary conservation and the ability to deliver messages out of order to the receiving application. Below, in Figure 3.1, the most important mechanisms in SCTP are shown. In the coming subsections these mechanisms are described.

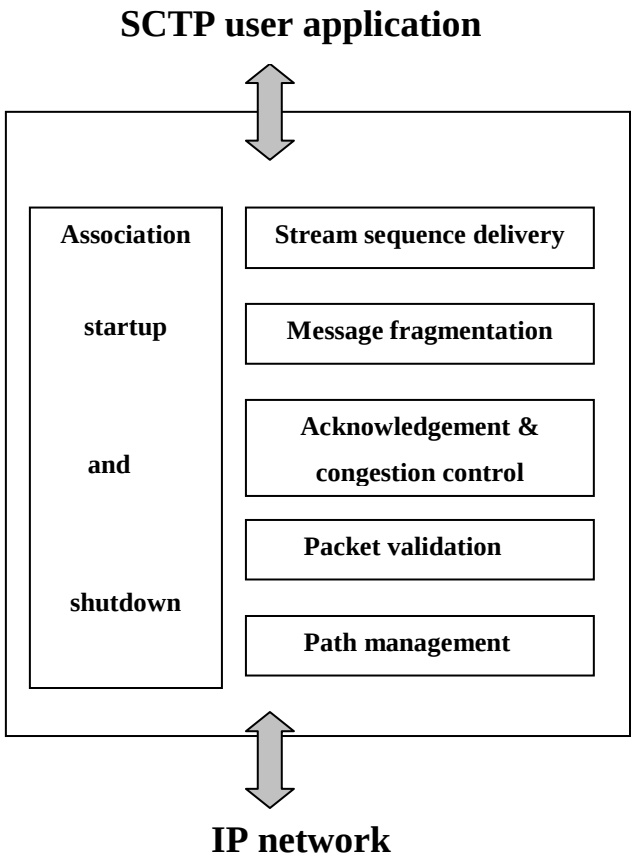


Figure 3.1 Overview of functions of SCTP

3.2.3.1 The association setup

SCTP is connection oriented, which means that a connection is established between sender and receiver before data is sent. The connection is established using a four-way handshake, see Figure 3.2. This procedure is different from that in TCP, which uses a three-way handshake. The four-way handshake was introduced to protect from Denial-of-Service attacks, such as SYN flooding [9] .

An established connection between hosts is, when using SCTP, called an association.

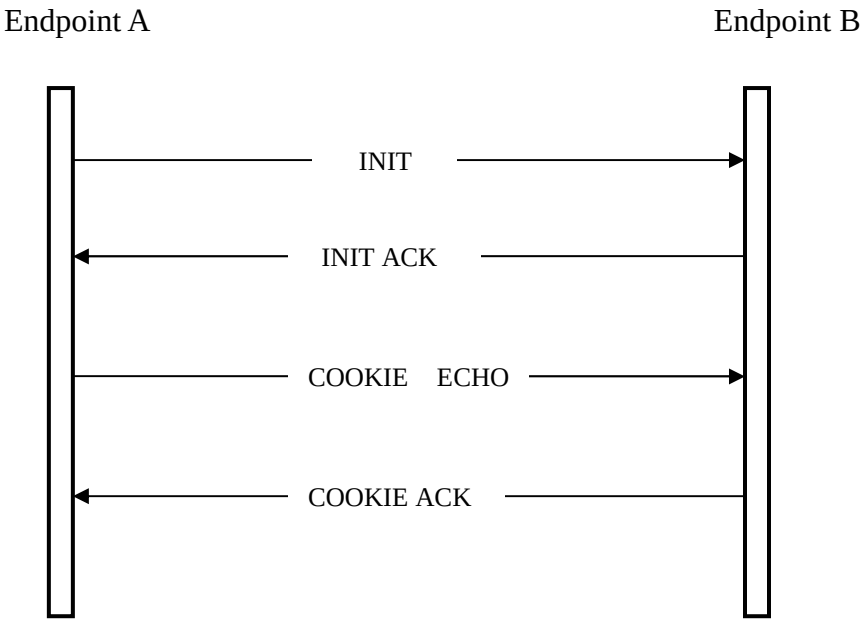


Figure 3.2 SCTP four-way handshake

3.2.3.2 Unordered data delivery

An application using SCTP has the possibility to choose to deliver the message in ordered or in unordered delivery mode. When using unordered delivery mode the receiving SCTP host will deliver the messages to the application as soon as they arrive, even if the sequence number is not the next expected. This is suitable for IP telephony signaling messages, which does not require strict ordering[6].

3.2.3.3 Message boundary conservation

SCTP is message oriented whereas TCP is byte stream oriented. This means that SCTP preserves the message boundaries throughout the transport between peers. The message boundary control makes the message delineation information, needed by the application when using TCP, unnecessary. The whole message is kept together and the receiver will deliver the assembled message to the application, whereas TCP delivers a stream of bytes.

[6] If the message is bigger than the underlying path MTU, SCTP also has support for fragmentation of messages.

3.2.3.4 Multistream support

Within one SCTP association there can be multiple logical parallel multiplexed streams. These streams are unidirectional. The number of inbound and outbound streams is set during the connection procedure. The streams may be used in parallel for simultaneous data transfer. The ordering of data in a single stream is strict, but there is no ordering of data between separate streams, therefore partial ordering is provided. If the message is sent in ordered mode, the receiver will use a mechanism to reorder data if necessary, but this reordering process is bypassed if the message is sent in unordered mode. Multistreaming was invented to prevent head-of-line blocking when one or more packets were lost.

3.2.3.5 Multihomed host support

Multihomed hosts are hosts that have multiple IP interfaces. The multihoming support provided by SCTP improves the robustness of the association because there may be several paths between the same end points. A multihomed SCTP end point can be represented as a list of SCTP transport addresses that share a single SCTP port. If one path fails, traffic can still be sent using another path. In an association, one path is marked as primary and if this path suffers a breakdown the sender starts sending data using one of the other connections. Figure 3.3 shows a multihomed association between two processes. If the path using the interfaces IP2 will break down, the association will still be working using either the IP1 or the IP3 paths (on both machines). The multihoming feature thus improves the association robustness [14].

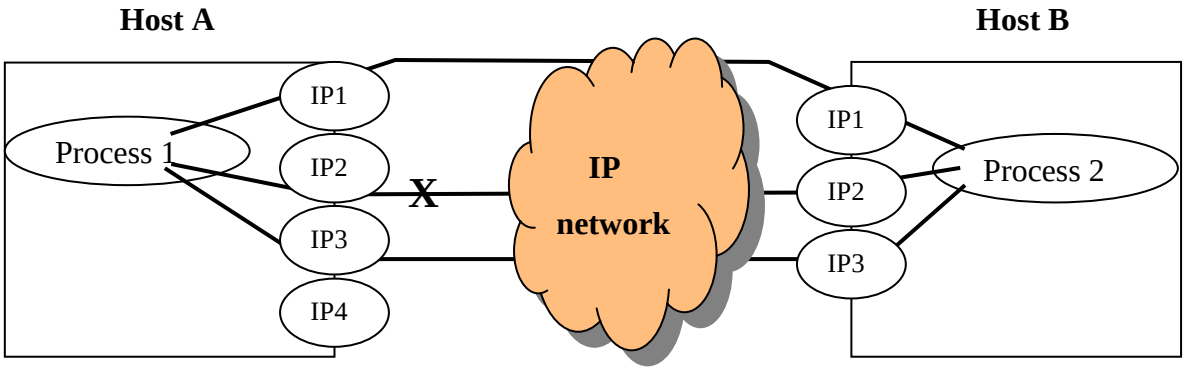


Figure 3.3 Multihoming between two hosts

3.2.4 SCTP packets

An SCTP packet has a header, called the common header. The SCTP packets sent over the networks are composed of the common header and of one or more specific building blocks called chunks (containing control information or data). An SCTP packet is seen in Figure 3.4.

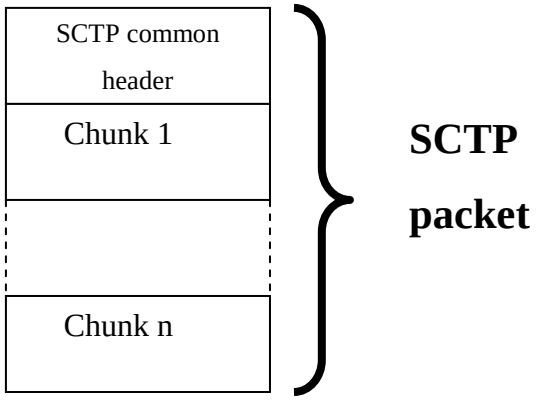


Figure 3.4 View of an SCTP packet

3.2.4.1 The common header

The SCTP common header is 12 bytes long and it provides three basic services: the method to associate an SCTP packet with an association (source and destination addresses and port numbers), the verification tag to validate the sender of the SCTP packet and a checksum to verify that the data is correct.

3.2.4.2 The Chunk

Chunks are the SCTP messages sent over the network. There are basically two types of chunks, data chunks and control chunks. The chunks are used for connection setup and shut down, heartbeat messages (used as a keep-alive message sent to idle destination addresses) and for data messages. The chunk includes information about chunk type (16 chunk types have been defined), chunk flags (used by the specific chunk type) and chunk length. If it is a data chunk, data is included.

3.2.4.3 Message bundling

An SCTP packet is designed to carry multiple chunks, so that multiple user messages may be bundled into a single SCTP packet. By bundling several small user messages into a single

SCTP packet, there is an improvement in network bandwidth efficiency. Bundling is the default for data chunks, but not for control chunks.

3.2.5 SCTP implementations

A few implementations of SCTP are as yet available and several projects are currently implementing SCTP [34]. The most well known implementation is the one in FreeBSD [27]. Another running project is implementing SCTP in a coming Linux kernel (Linux 2.6) [29], but so far there is only an alpha version available. A reference implementation, run in user space, is being developed by the research group run by Randall Stewart [20]. Another reference implementation has been produced by Siemens in cooperation with Computer Networking Technology Group of the University of Essen [22].

4 Experimental Setup

In this chapter, first the aims of the study are described, and then there is a brief introduction to the Session Initiation Protocol (SIP) [17], which has served as inspiration for the study. Next, the network topology and the different test parameters are described and motivated.

4.1 Aims of the study

The aims of this study were twofold. One aim was to evaluate the experimental platform Emulab, i.e. to investigate which benefits and drawbacks that can be seen from using Emulab as a platform for experiments compared to using a local lab.

Specific questions raised were:

- How difficult is it to allocate resources for an experiment?
- How difficult is it to perform an experiment on Emulab compared to in a local lab and what are in such case the drawbacks/advantages?
- How well documented is Emulab?
- How is the support situation of Emulab?
- How long will response to a support question take?

One way to evaluate Emulab is to do the same tests in both Emulab and in a local lab and to compare the results. This corresponded well with the second aim of this thesis, to undertake a performance study on SCTP.

Specific questions were:

- Does SCTP in ordered or unordered mode offer higher or lower throughput than TCP?
- If there are any differences, how will these differences vary under various network conditions?

4.2 Inspiration for the study

A new application protocol for initiation of multimedia sessions, the Session Initiation Protocol (SIP), became available a few years ago. SIP is transport independent, but today the mostly used transport protocol is UDP. The SIP architecture consists of end points, called user

agents (UA) and SIP proxies. The proxies receive a message from a user agent or another proxy and forward the message to another proxy or the destination user agent. Figure 4.1 shows a SIP message exchange to establish a session. In this figure there are two user agents and one proxy.

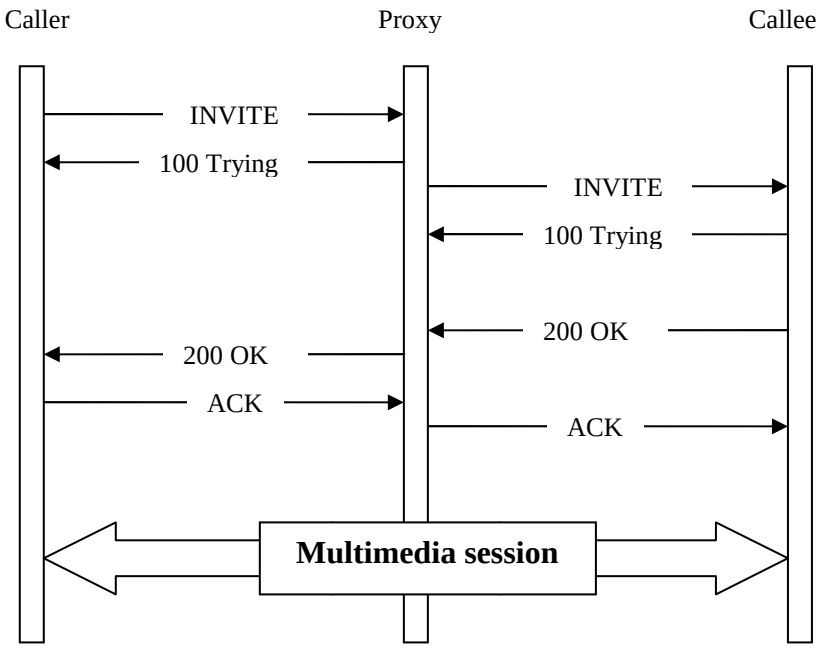


Figure 4.1 SIP message exchange

The idea was to perform an experiment simulating SIP traffic between two proxies, using the different transport protocols and then to compare the throughput. The aim of SIP is to initiate and negotiate options for a session. SIP messages are therefore usually quite small, since they only carry a session description, written in Session description protocol (SDP) [39].

UDP is usually used for the transport of SIP messages. UDP is suitable for transport of individual small messages, when no fragmentation is necessary. UDP has also the advantage that no latency for connection establishment is introduced. However, one of the drawbacks of UDP is that no reliability is provided. This means that the end points must provide reliability in each application layer.

Due to the lack of reliability, the loss of one message may take a long time before detection. Nevertheless, UDP is the most suitable protocol for these types of small messages between end point and proxy, at least under moderate traffic load.

The situation is a bit different in the transmission between proxies, where many messages may be sent between the same two hosts. In this situation, TCP or SCTP may become a

competitive alternative to UDP, since many small messages may be assembled and sent in the same packet. The scenario our experimental study focuses on is emulation of several SIP messages sent between the same hosts.

4.3 Studies on SCTP

SCTP is quite a new protocol, but there are nevertheless several studies of SCTP. A few of them have been focused on aspects related to this study.

In one study, a comparison between SCTP (using one single stream in unordered mode), TCP and UDP has been made to see which transport service is best for transporting multiple SIP messages between proxies [5]. In this study the protocols and the network were simulated and the experiments indicated that SCTP has some advantages over TCP. The protection against Denial-of-Service attacks, the multihoming facility and the delivery of complete signaling messages to the application provided by SCTP were some of the advantages pointed out in the report. According to throughput there was no significant difference between the different transport protocols under moderate traffic, but for higher levels of packet loss SCTP was shown to have significantly higher throughput.

A performance study comparing SCTP and TCP over a satellite link has also been performed [1]. This study indicated that SCTP has slightly higher throughput than TCP. The difference between TCP and SCTP was found to depend on differences in the congestion control mechanism.

Another study compares SCTP to TCP for mobile IP [7]. This study shows that SCTP has slightly higher throughput than TCP when the bandwidth is low (under 200 Kbps).

The same scenario, as described for the experiment in this thesis, has been studied in a local lab at Karlstad University [4]. In the study, TCP and SCTP were used and one aim of this study was to investigate the fairness in sharing bandwidth between the two protocols in a concurrent situation. Further, another aim was to see if the unordered delivery gives a shorter message delay compared to ordered delivery. Since the same scenario has been used in that experiment, the results from that study will be compared to the results from Emulab (see Section 5.2).

4.4 Studies performed on Emulab

Emulab has been open for public use for a few years. Several studies where Emulab has been used as experimental platform have been performed. Most of the studies have naturally

been focusing on networking problems. As an example, one study focused on problems with differential services for TCP flows over Internet [18]. Another study focused on reliable communications in overlay networks [3]. Most of the studies performed on Emulab can be found at the Emulab homepage [39].

None of these studies has been made in order to evaluate Emulab, but the researchers of these studies all seem to be confident in the high performance of the platform.

4.5 Scenario

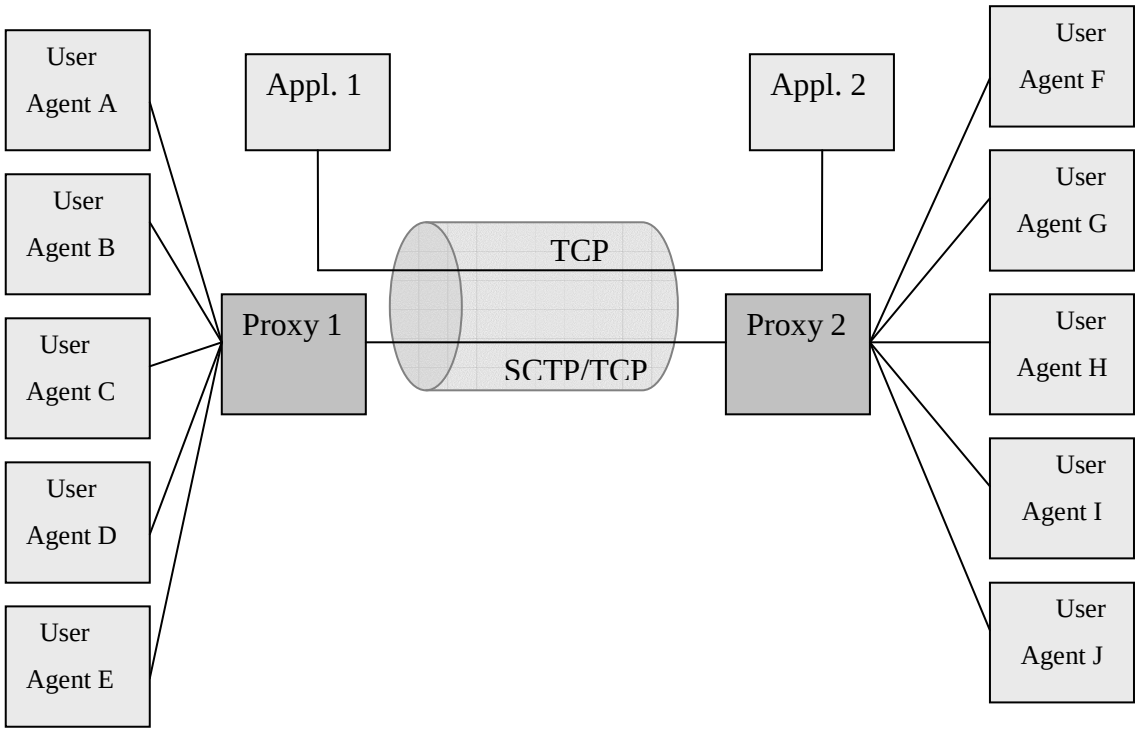


Figure 4.2 Overview of scenario for the experiment

One aim of the study was to evaluate whether there is an improvement in throughput by using SCTP compared to TCP, for transport between proxies.

Three scenarios were used in this study. In the first scenario, the proxy makes use of SCTP in ordered mode, meaning that SCTP delivers messages to the application in the same order as they were sent. In the second scenario, SCTP delivers messages in unordered mode, and in the third scenario TCP is used.

To make the experiment a bit more realistic, some of the tests performed include concurrent traffic. To achieve this, TCP-traffic will be sent as fast as possible in parallel to the emulated SIP traffic (Appl. 1 and Appl. 2 in Figure 4.2).

Since the experiments focus on the transport between proxies the endpoints (User Agents) will not be a part of the experiment. This exclusion of endpoints makes it possible to emulate a number of User Agents in one application, generating messages and transmitting them to the SIP proxy. In the different scenarios, either SCTP or TCP will be used as transport service provider. The content of the messages is not important, just the message size. This message size is to be in the same range as the size of a SIP message and is therefore set to 500 bytes.

The one-way delay and the bandwidth over the link will be set within a realistic range. The decision is to use two different delays, 25 and 75 ms. The first delay is a realistic delay for sending between two proxies inside the same country, and 75 ms is reasonable delay between two more distant proxies. The bandwidth is decided to be 400Kbps. The motivation for using this bandwidth is that the minimum limited interval used between messages is 10 ms. This implies that 100 messages will be sent every second. Since the message size is 500 bytes this implies 400 Kbit of data sent every second, which is 100% of the bandwidth. (Messages are also sent at full speed, the results from this transmission are to be seen as a reference value, since the bandwidth constraint is set to 400 Kbps).

4.5.1 Simplifications

Compared to a real situation a few simplifications were made. These were the following:

- Traffic was sent only in one direction. Real SIP-traffic is bidirectional in a query-response-pattern, see Figure 4.1.
- The emulated SIP messages were, when making use of SCTP, transported on a single stream inside the SCTP association. This implied that the multi-streaming facility of SCTP was not used. In the situation when TCP was used as transport service provider, the SIP messages were transported using a single TCP connection. In a real situation probably several streams inside a SCTP association would be used, and perhaps also several TCP connections in the TCP scenario.
- There was a fixed interval between messages generated by the application. In a real situation the SIP messages would be generated with varying intervals.
- The multihoming facility of SCTP was not used in this study. This is a limitation and the use of this option was left to further studies.

4.6 Background traffic

In real life the links between hosts are usually busy. To make the experiment a bit more realistic some of the emulated SIP traffic had to compete with parallel traffic. The competing traffic was controlled by applications, which were sending TCP data in parallel to the SIP traffic over the same link. The applications were able to vary the amount of competing traffic by varying the number of TCP connections. The competing traffic was always sent at full speed and the amount of traffic was controlled by setting up different number of parallel connections. In the case where competing traffic was used it was started a few seconds before the emulated SIP traffic.

4.7 Nagle´s algorithm

The sending of small TCP or SCTP packets over a network introduces quite a big overhead, since each packet has a header. If the network is heavily loaded there is a risk of congestion, which may result in lost packets and retransmissions. If the traffic consists mostly of small packets, the introduction of many headers increases the overhead for the delivery and decreases the data throughput. This is called “the small packet problem”. To overcome this problem, an algorithm to reduce the overhead has been introduced. This algorithm, called Nagle’s algorithm [12], reduces the overhead by first sending the first packet, and then queuing the other packets until an acknowledgement arrives for the first packet. When this happens all queued messages are sent in a single packet (up to the MTU) Nagle’s algorithm introduces a potential delay for the individual small packet, but not for the entire transmission. The reduction of risk of congestion has made Nagle’s algorithm standard in TCP implementations. This feature has also been inherited by SCTP. Nevertheless, it is possible to disable Nagle’s algorithm when using both TCP and SCTP.

The delay of packets may have an impact on the results in the tests of this study since 1000 packets of 500 Bytes are sent in sequence with a specific interval in between. It is unclear whether the message bundling facility of SCTP has any impact on the throughput when Nagle’s algorithm is not used. It seems possible that the usage of Nagle’s algorithm may increase throughput when bundling of messages is used. Due to this, the ambition was to perform the experiment both with and without using Nagle’s algorithm to see whether the usage of this algorithm entailed any differences. When Nagle’s algorithm is enabled it is referred to as *delay*, and when disabled referred to as *nodelay* (see Section 4.10).

4.8 The network

A logical connection between two computers, a client and a server, was allocated at Emulab, as described in Section 2.2.4. The client and the server ran the test programs and programs generating background traffic. The data was sent from the server to the client over a link called testlink, see Figure 4.3. A third computer, admin, was also allocated and connected to the network. This server ran scripts to manage the different programs and stored the test results.

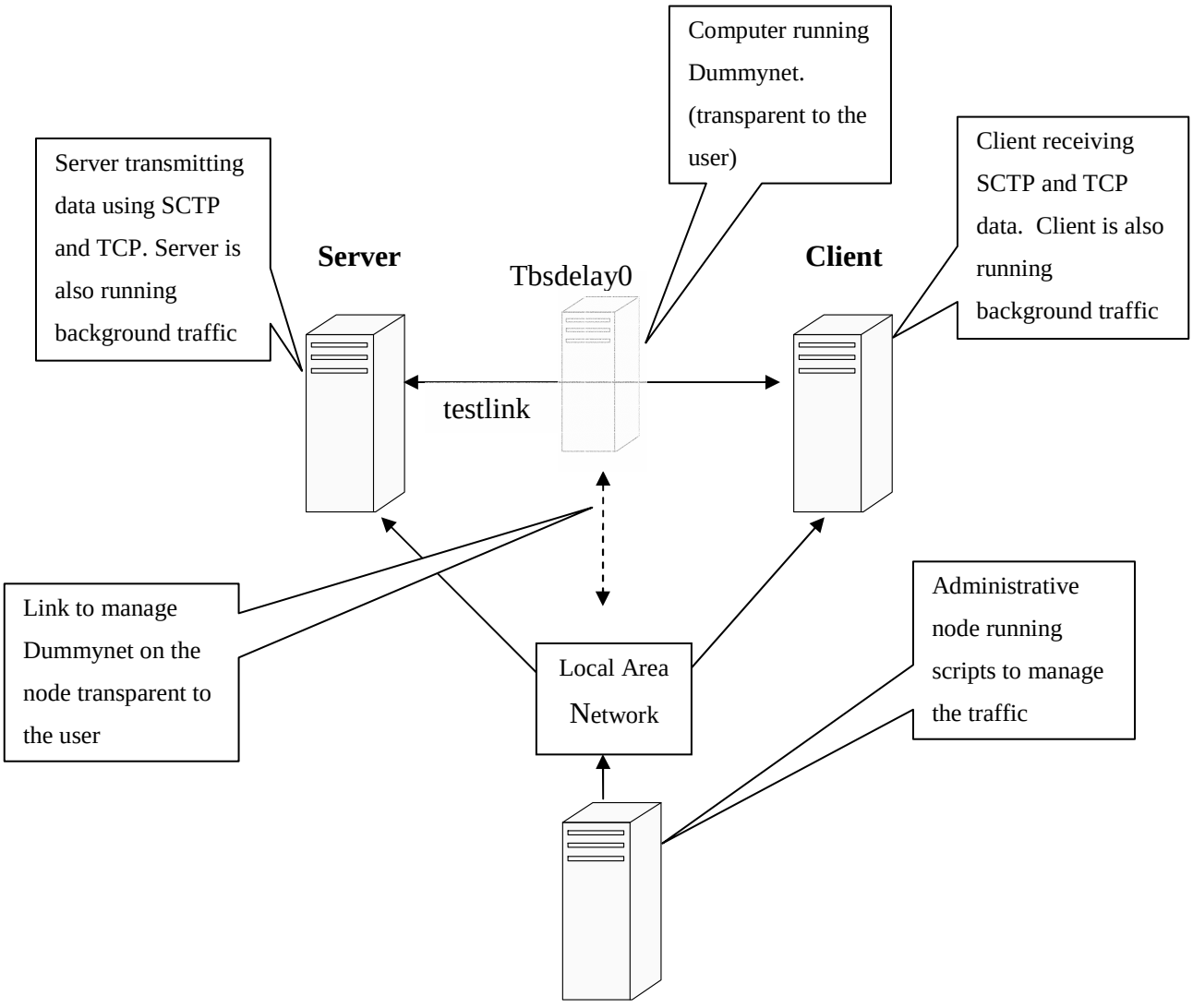


Figure 4.3 Logical view of network setup

The physical setup included a fourth computer running Dummynet marked as Tbsdelay0 in Figure 4.3. The computer was implicitly allocated, since the request for the resources included a request for bandwidth constraint on the testlink between server and client.

4.9 Software

The ambition was to compare the results of the tests in this study with the results from the tests performed on the local platform [4]. The experiment on the local platform was to be made using the Siemens implementation of SCTP run on a Linux platform, in user space. The choice made for this study was to use the same SCTP implementation to make the comparison as fair as possible. An advantage from using this implementation was that there was a socket API with useful functionality and test programs available. The prototype implementation of SCTP made by Siemens, together with the API, was installed on both the server and the client. For some reasons the SCTP implementation used in the experiment in the local lab was changed. That experiment was performed using the FreeBSD implementation of SCTP, which is a kernel implementation, see Section 4.11.4.

The experiment with TCP as transport protocol was performed using the kernel implementation of Red Hat Linux 7.3.

There were 2 + 2 applications running during the experiment. Two of them were the concurrent traffic generators (unidirectional TCP-traffic). The third application was the generator of SIP traffic, and the fourth application was the receiver of SIP data.

4.10 Experimental parameters

The tests were to be performed in six different modes in total:

- SCTP in unordered mode
- SCTP in ordered mode
- TCP

All of these modes were also to be performed with Nagle's algorithm both enabled and disabled (referred to as delay and nodelay).

The chosen SCTP-implementation is still under development. This implies that not all features of SCTP specified in the RFC 2960 are implemented. During the experiment it was found that the possibility to disable Nagle's algorithm was not yet implemented. As a result, the running of the experiment in nodelay mode was not possible when using SCTP. This implied that this mode was not used in any of the experiments and the tests were performed only in the three modes mentioned above.

To be able to compare the experimental results to the results from the local lab, this experiment was performed with the same parameters. In the local experiment only a network delay of 75 ms was emulated. To have the possibility to see differences in throughput when

the network delay was changed, tests with a network delay of 25 ms was performed. To be able to see a possible impact on throughout of a small router buffer, tests with a network router buffer of 6 packets was also performed. This resulted in many different parameter settings. All tests for each setting were repeated 100 times. All the different parameter combinations used in this experiment are presented in Appendix B.

The packet size was always 500 Bytes and for every parameter setting 100 packets were sent.

4.11 Problems

4.11.1 TCP cache

When using TCP, the slow start and congestion avoidance algorithm is used [2]. By using this algorithm the sender, after a while, gets to know a reasonable level for sending data over the specified connection. The TCP implementation in Linux 2.4 is caching data concerning transfer rates between subsequent connections between the same hosts, so that the size of the congestion window of the server is stored (for 10 minutes) and reused when setting up a new connection between the same hosts again [28]. Caching may impact the results, since the different tests are to be performed in sequence. There is no way to disable this feature, but there is a way to restore the congestion window between each test to the initial value, by using a specific command. This has to be done with root privileges. The overall recommendation by Emulab is to use the command sudo to get root privileges. When trying to type the command the response was "*permission denied*". This was expected, since the sudo command is just a redirection. After contacting the support team at Emulab there was a rapid reply that gave a proper answer which helped to solve the problem.

4.11.2 Halts in the test

The time to perform the whole experiment was estimated to take between 7 and 10 days. The experiment was run automatically by several management scripts. The scripts run on the admin node sent parameters as well as start and stop commands to the different nodes using ssh. The tests run smoothly most of the times, but occasionally there was a stop in a test due to ssh problems (the application suddenly asked for the authentication key). Problems with ssh caused several stops in the execution. These ssh problems prolonged the execution time dramatically.

The problem was presented to the Emulab staff and the answer was as follows:

"*It looks like the root of the problem is your NFS-mounted home directory disappearing occasionally on the admin node*".

They promised to look at the problem after experiment termination. After a while the following answer and suggestion came:

"*It's a race in the FreeBSD code. We have found it but it's tedious to fix so it has not been done yet. But if you remove your dependency on the NFS mount, you shouldn't have the problem. Copy all your stuff over to the local disk and run from there*."

At this time most of the tests were already completed and since it would have meant more work, the tests were not repeated.

4.11.3 Different implementations

The kernel implementation of TCP has further restrictions when sending data, since the buffer to store data until the ACK arrives is limited. This implies that the send call is blocked when the buffer is full. Nevertheless, the size of the send buffer doesn't seem to have any impact on throughput. There may also be other differences between the kernel implementation and the user space implementation used for SCTP that impacts the result. This means that the interpretation of the different results from using TCP and SCTP has to be done with care.

4.11.4 Change of implementation

The ambition was to compare the results of the SCTP tests in Emulab with those performed in the local lab. The intention was also to use the same implementation of SCTP and the same operating system on both platforms. For different reasons the experiment in the local lab was changed slightly and the local experiment was executed with the FreeBSD implementation of SCTP, which is a kernel implementation. It was not possible to change the SCTP implementation for the Emulab SCTP experiment, since the tests had already started. The use of different implementations makes it more difficult to compare the results. The difference in performance does not have to be a result of the different platforms but they can also be due to different implementations.

Another difference between the experiments was the operating system. In FreeBSD, the SACK functionality in TCP is not included. This implies that the concurrent traffic is using ordinary cumulative ACKs. The SACK-functionality is used in the Linux implementation used for the experiment on Emulab. Since the SACK functionality is more efficient than only cumulative ACKs the concurrent traffic is supposed to be more aggressive when using SACK.

Therefore, it is difficult to make a fair comparison between the results on Emulab and the local results when using concurrent traffic. The only possible fair comparison is when no concurrent traffic is used.

4.11.5 Connection problems

The ambition was to perform 100 repetitions for each setting, to get reliable results. From these results the plan was to calculate a valid mean value with a 95% confidence interval. This ambition was fulfilled when there was no concurrent traffic. When using a limited router buffer and concurrent traffic it was not always possible for the application to connect within reasonable time. The reason for these problems is not identified in the scope of this thesis. These connection problems lead to fewer samples (in some situations only about 50 repetitions were performed) and is one of the reasons why the confidence interval in some settings is rather large. The few samples and the big confidence intervals is one of the reasons why it was not possible to make a fair comparison to the test in the local lab.

5 Experimental results and analysis

This chapter presents and analyses the results obtained from the experiment. The evaluation of Emulab as an experimental platform is presented in chapter 6 .

In Section 5.1 the expected results are stated, followed by the presentation of the actual results in the form of graphs in Section 5.2. Finally, in this section a brief analysis of the results is made and specific throughput differences from using SCTP and TCP as transport services are pointed out.

5.1 Expected results from the experiment

There were many similarities in sending data over a network using SCTP with a single stream, and using a single TCP connection. However, one difference is that data delivered by SCTP may be delivered unordered, whereas data delivered by TCP always has to be delivered in order.

Because of this, the differences in throughput between TCP and SCTP in ordered mode were expected to be small since the functionalities are very much the same. Nevertheless, the results from using SCTP were expected to show a slightly higher throughput, at least in a congested network. The reason for this was that results were expected to be in line with those achieved in the simulation study [5].

Another expectation was that there should be small differences in the result from using SCTP in the local lab compared to using SCTP at Emulab, since the lab platforms are similar. The possible differences in the results between the local lab and Emulab are not supposed to depend on the lab platform, but on the different implementations of SCTP that had been used. Moreover, the concurrent TCP traffic is not using the same ACK mechanism on FreeBSD as in Linux, why it is not reasonable to expect the same small differences between the results in the congested situations as in the situations with no concurrent traffic.

5.2 Experimental results

In this study, throughput was the main concern. That is the reason for presenting the throughput mean value of the 100 repetitions. In addition, the confidence interval at the 95% level was calculated and included.

The data from the experiments was written to a file. The data collected was:

- Departure time from the server application
- Arrival time to the client application
- Sequence number for the message (1-1000)
- Test number and repetition number (1-100)

For each test the time from sending the first message to arrival of the last packet was measured. From these values it was possible to calculate the throughput since the size of the data transferred was known. The average time and the confidence interval were calculated from the 100 repetitions of the same test. All calculations were made by home made perl scripts. From the calculations it was possible to plot the results in graphs by using gnuplot.

The results are presented in form of graphs, where each graph shows the throughput as a function of the interval between messages. In each figure the same number of concurrent connections and the same router buffer size is used, at different intervals between messages.

The interval between messages indicates the capacity used by the measured traffic. For example, when the interval is 20 ms, 200 Kbps is used. A result graph showing the maximal throughput at the given intervals is shown in Figure 5.1. The graph in this figure is shown as a reference graph, showing the maximal possible throughput when no concurrent traffic is in the network. This graph has no error bars for confidence interval, since it is a reference and in this ideal situation all messages have the same maximum throughput.

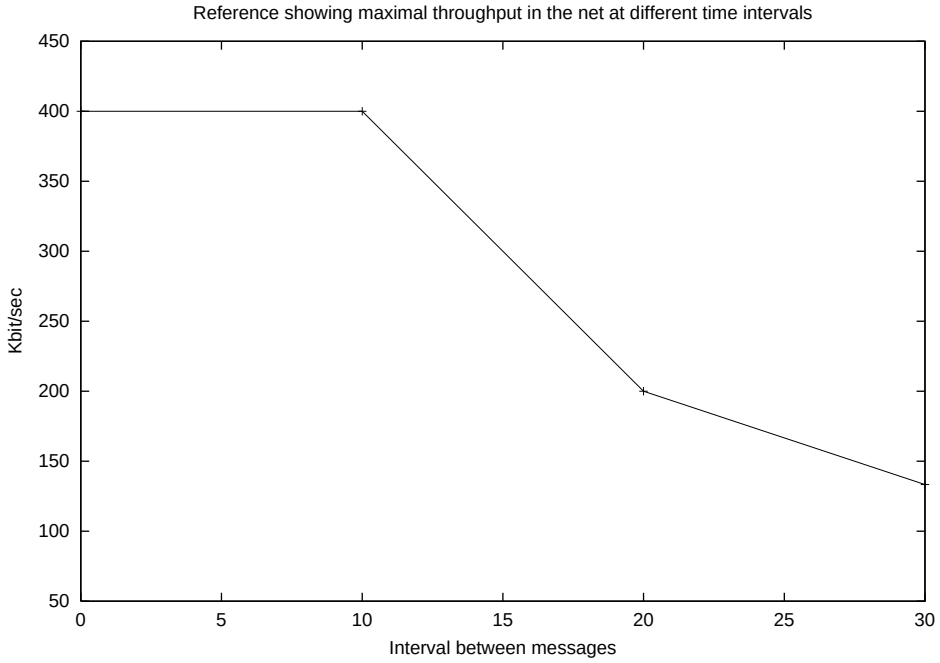


Figure 5.1 Reference graph showing maximal throughput

The results in the graph only show throughput values for the specific intervals 0, 10, 20 and 30 ms. It is not possible to say anything about the values between the specific intervals. The values are connected with lines, but these lines should be looked upon just as probable values.

In Figure 5.1 it can be seen that sending messages without any time interval in between does not increase the maximal throughput compared to sending the messages with 10 ms interval. This result is due to the bandwidth constraint, which is set to 400 Kbps. When sending the messages with 10 ms interval it implies full usage of the bandwidth. Further, 20 ms interval means usage of 50% of the maximal bandwidth.

5.2.1 Result graphs

Result graphs from some of the tests are shown in Figures 5.2-5.6. Each graph shows the throughput as a function of the intervals between messages using the same number of concurrent connections. The graphs commented in this section concern results with reasonably small confidence interval and also results that differ from the results from the local lab, see Appendix C.

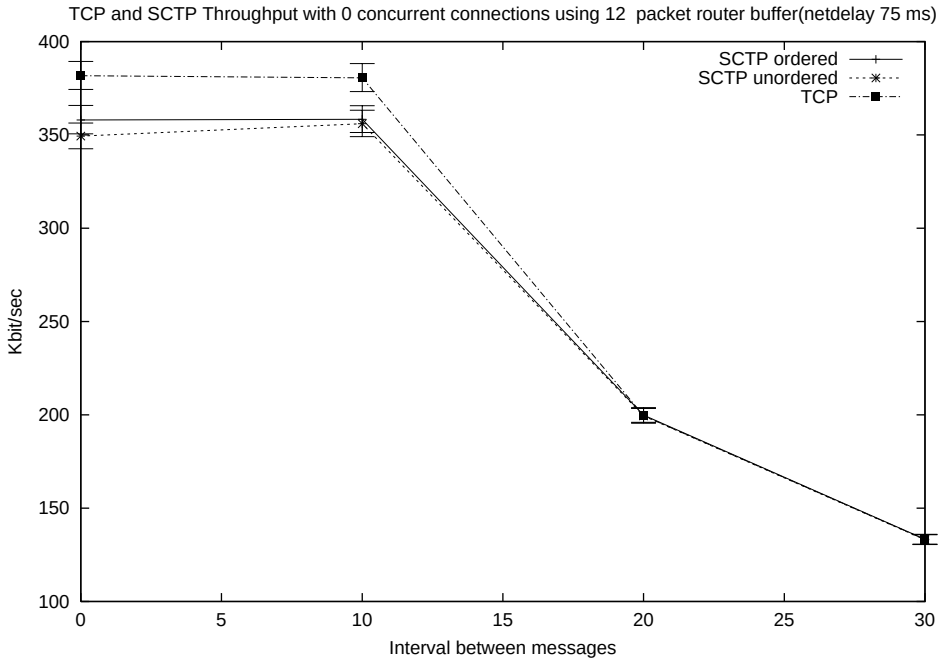


Figure 5.2 No concurrent traffic. 75 ms delay, 12 packet router buffer

When looking upon the first two of these figures (Figure 5.2 and 5.3), where the delay is set to 75 ms, there are no significant differences between the results and the reference values

shown in Figure 5.1, and there is also no significant difference between the different transport services. This is expected, since there is no other traffic competing for bandwidth. The results indicate that TCP is performing marginally better than SCTP in any mode when traffic is being sent at full speed and when the usage of the bandwidth is exactly 100% (= 10 ms interval).

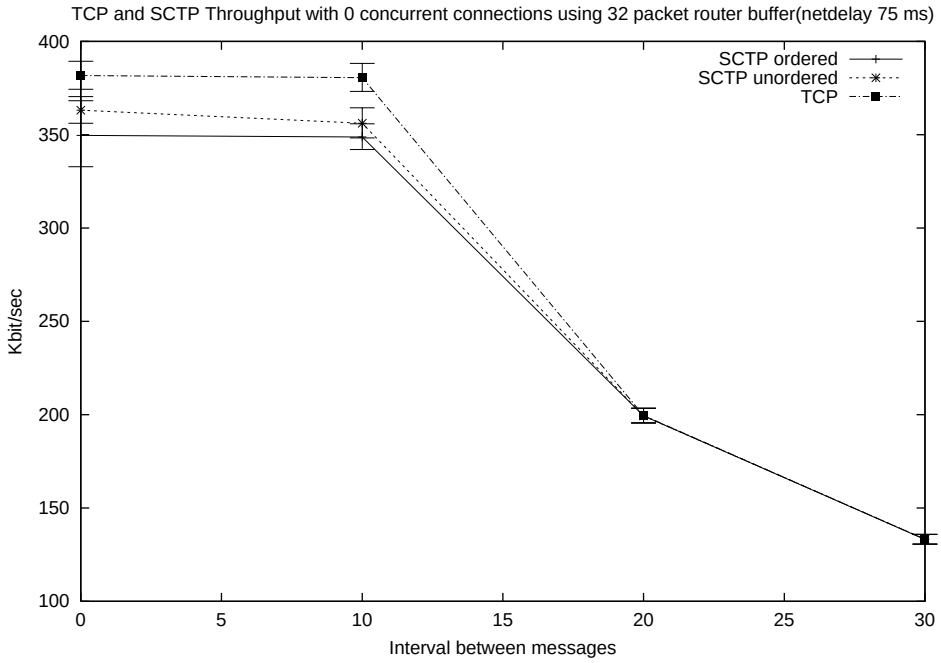


Figure 5.3 No concurrent traffic. 75 ms delay, 32 packet router buffer

The comparison shows that the different results from Emulab in Figure 5.2 to Figure 5.4 are in line with the local results in Appendix C. This indicates that the usage of the remote platform does not introduce any difference in the results.

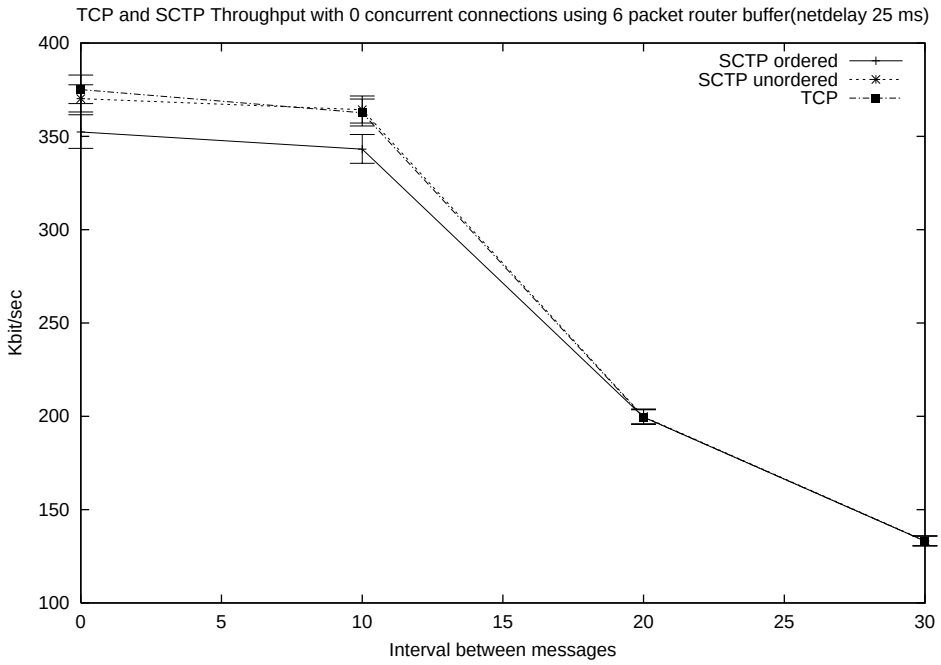


Figure 5.4 No concurrent traffic. 25 ms delay, 6 packet router buffer

Even after reducing the router buffer to 6 packets and the network delay to 25 ms (see Figure 5.4), the throughput of the protocols is almost identical. The graphs show that SCTP in ordered mode performs slightly poorer than the other protocols when the bandwidth of the network is fully utilized. The results from these settings are also in the same region as the results of the reference results of Figure 5.1.

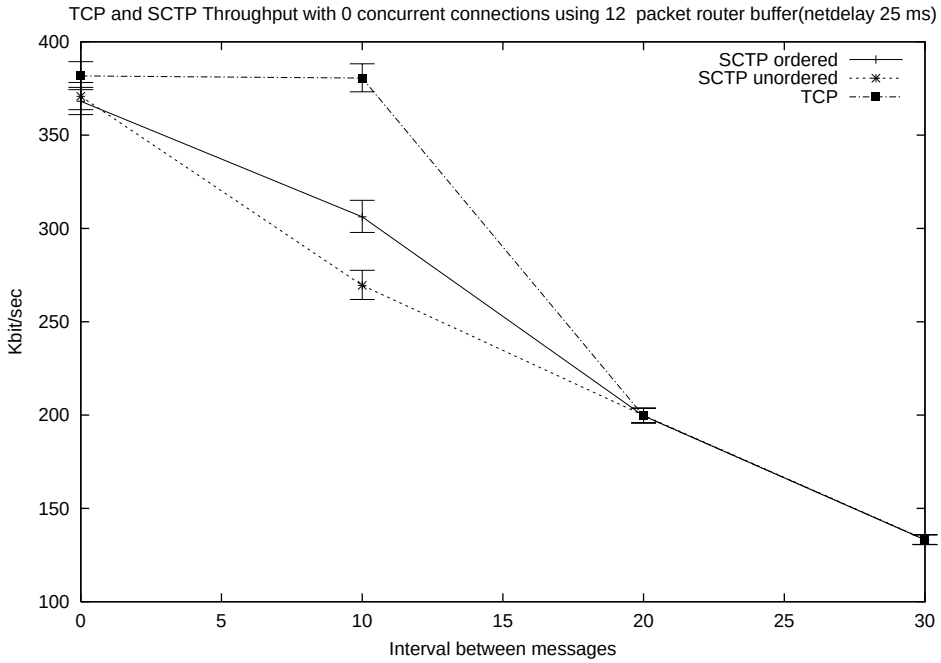


Figure 5.5. No concurrent traffic. 25 ms delay, 12 packet router buffer

Figure 5.5 and Figure 5.6 show results when almost all parameters are the same as in Figure 5.4. The delay is the same, as well as the number of concurrent connections (none). The only difference is the increase of the size of the router buffer of Dummynet. In Figure 5.5 the router buffer is set to 12 packets and in Figure 5.6 it is 32 packets.

These results show that the throughput of TCP remains unchanged, and the throughput is still almost the maximum possible. When using this setting there is a discrepancy seen in the results from using SCTP both in ordered, but even more in unordered mode. This is not expected and it is quite remarkable. The expectation was that the throughput in these cases should be at least as good as with a smaller router buffer. The results show that when using SCTP in unordered mode, with a router buffer of 12 packets (at 10 ms interval between messages), the average throughput is only 278 Kbps. and for SCTP in ordered mode the throughput is 315 Kbps.

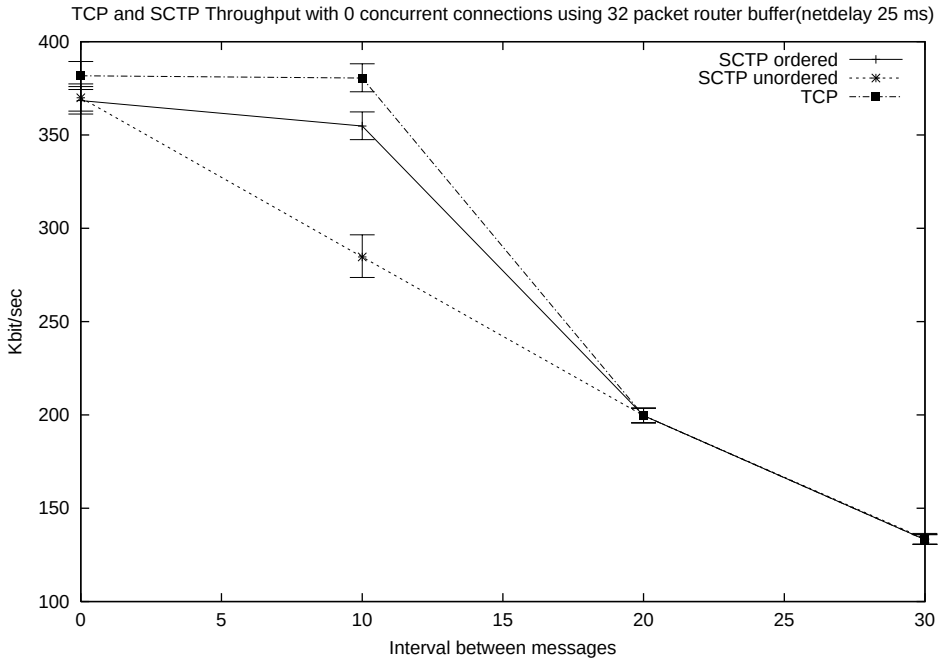


Figure 5.6 No concurrent traffic. 25 ms delay, 32 packet router buffer

When further increasing the buffer size, seen in Figure 5.6, the throughput for TCP remains unchanged and the throughput when using SCTP in ordered mode improves. Here the result when sending with 10 ms interval is 368 Kbps. On the other hand SCTP, in unordered mode at 10 ms interval is still not performing better than 285 Kbps, which is about 70% of the possible bandwidth. This is a remarkable loss in throughput compared to SCTP in ordered mode which shows utilization of about 88% and TCP which utilizes around 95% of the bandwidth. These decreases in throughput for SCTP were not expected, since there is no other

traffic in the net. These results raise a few questions which are further discussed in Section 5.2.2.

The results from using concurrent traffic all show different patterns and the confidence intervals tend to be larger than the results without concurrent traffic. The results from these experiments can all be seen in Appendix D. They are all very hard to analyse and it is also hard to see any tendencies. Further studies with different parameters have to be done to see any significant differences between the different protocols. In the same way it is very difficult to compare the results with the results in the local lab, since the prerequisites for the two parallel studies differ due to differences in concurrent traffic. What can be said as a general comment is that TCP with SACK functionality, which is implemented in Linux, but not in FreeBSD, has a more efficient loss recovery functionality than ordinary cumulative ACK. This usage of the SACK functionality makes the TCP traffic, used as concurrent traffic, use a larger share of the bandwidth compared to TCP without that functionality. The throughput for the measured traffic, when TCP without SACK is used (in the local lab), is generally higher than the throughput when TCP traffic with SACK is used for the concurrent traffic. These indications are in line with what was expected and no other comparison is made, since it is a bit like comparing pears with apples.

5.2.2 Results from the experiment in the local lab

The results from the experiment performed in the local lab have been compared to the results from the experiment performed in this study. To make this comparison possible the local results have been treated the same way as the results from Emulab. The local results are presented in the same form as the results from these tests. These graphs and are found in Appendix C.

5.2.3 Analysis of the results

Most of the results when there is no concurrent traffic are in line with the expected results. The only remarkable digressions are the results when using a net with 25 ms net delay and 12 or 32 packet router buffer, see Figures 5.5. and 5.6. In these cases, the throughput for SCTP in any mode is significantly lower than for TCP when there is 10 ms interval between the messages. At this interval exactly 400 Kbps is sent into the network, which means 100% utilization of the available bandwidth. The discrepancy is not seen at any other interval.

The drop in throughput for SCTP is not seen when the router buffer is as small as 6 packets, see Figure 5.4, and it is not seen when using TCP. Furthermore, the pattern is not

seen when having a longer net delay, 75 ms, see Figures 5.2 and 5.3. In this situation there is just a very small difference in throughput between the transport services.

It is not obvious why there is such a drop in throughput in this specific case and why this drop is not seen when the application is trying to utilize more or less than 100% of the bandwidth (at 0 ms or 20 ms or 30 ms interval) and why it is not seen when the router buffer is small.

There may be several reasons for this decline in throughput. When trying to search for a probable reason it is important to look at the differences in transport services. One aspect is that in these tests the results are all obtained with the Nagle's algorithm enabled. This introduces a potential delay (see Section 4.7). Nagle's algorithm is used both for TCP and SCTP. The difference between the protocols is that TCP is byte stream-oriented, whereas SCTP is message-oriented. The transport in SCTP is performed in the form of chunks. Usually one message is put in the same chunk (if the message is not larger than the path MTU). To reduce overhead, several chunks may be bundled into the same SCTP message. In this scenario the messages are 500 Bytes, which means that several messages are bundled into the same packet. This bundling also introduces a potential delay and a potential reduction in throughput. The theory that the bundling of messages together with Nagle's algorithm is responsible for the drop in throughput is nevertheless not likely, since that would also have reduced the throughput when messages are sent without intervals and even when the buffer queue is small. Nevertheless it would be interesting to see if this situation is repeated when Nagle's algorithm is disabled. That might be a subject for further studies.

Another theory is that, since the situation does not occur when the buffer is small, the increased size of the router buffer might entail the throughput reduction. It is a complex process to choose the optimal router buffer size, but there is a universally applied rule-of-thumb [38] that says that B= 2T*C, where B is the buffer size, 2T is the roundtrip time and C is the capacity of the link. This rule is based on the TCP congestion control and would in this specific case be 50 ms * 400Kbps = 20Kbit = 2500 Bytes = 5 packets. This implies that a 6 packet router buffer should be enough for this transport and the usage of 12 and 32 packets would be more than enough. Further analysis of each packet is needed to find the reason for this decrease in throughput.

Deciding the router buffer size is a delicate problem and in this case one aspect could be that a big router buffer may cause the packets to stay in the buffer queue longer than if the router buffer is small. This could affect the timer on the sender side so that the retransmission time out, RTO does not decrease properly. This would make the sender wait longer time

before resending a lost packet. Nevertheless, if this would be the only reason it is difficult to explain why the drop in throughput is different between SCTP in ordered and unordered mode, since the same implementation is used.

Another possible reason for this discrepant result is that the implementation used for this SCTP transport is a prototype implementation, which is probably not yet optimized.

To further analyze the result, there is a need for controlling each packet by looking at the network dump files. This is not possible within the scope of this project and is left as a challenge for further studies. From the results obtained in this study it is not possible to say whether SCTP is more efficient for this type of traffic than TCP. A few questions have come up and to answer these question and to say anything about which transport service is more efficient further tests and analysis have to be performed and perhaps also with a different SCTP implementation.

6 Results of using Emulab

The expectations on Emulab were quite high, since the documentation concerning the platform was very positive. The results from this study do not cover all the aspects and facilities of Emulab. For example, no operating system but the default ones has been used.

Before performing the study presented in this thesis a few expectations were stated. The result from using the platform shows that almost all the expectations were fulfilled.

As the experiments turned out, the only fair comparison between the results from the local lab and the results achieved on Emulab platform were the results from tests without concurrent traffic, since the situation when using concurrent traffic is not the same on the different platforms.

When looking at the results from the different platforms (see Appendix C and D) they look very much the same, which was expected. Based on this similarity between the results, it is possible to say that the remote platform performs the same way as a local lab.

Only four nodes were necessary for this experiment so there has been no pressure on the number of nodes from our side. Nevertheless, the number of free nodes has always been presented on the web-page and the number of free nodes has almost always been high. Only twice all the nodes have been occupied and at these occasions there has always been some special event going on and all users of Emulab has been informed in advance via mail. Maybe Emulab is mostly used by researchers in the USA and due to the time difference the number of available nodes has been high when we required resources in the daytime.

The documentation concerning the functionality of Emulab has been found to be clear and easy to follow. After reading the instructions found on the web site it was easy to start the experiments.

A few problems have come up during the experiments and a number of questions have been sent to the Emulab staff. The answers to these questions have always come rapidly, within 24 hours, and they have always been very clear and almost always have helped to solve the problem.

6.1 The most significant advantages of using Emulab

There have been several benefits from using Emulab compared to a local network. The most significant advantages found are:

- **Simple and fast configuration**
  - There has been no need for manual setup of the network and no manual configuration of the separate nodes. All allocation and configuration has been done via the web interface and if the requested resources were available the e-mail informing that the network is ready has come within two minutes.
- **The possibility to run several tests in parallel**
  - The execution of the experiments presented in this thesis took several days and nights. To speed up the execution, it was possible to allocate several different experimental setups and run experiments in parallel. The only part to be careful about was that the results should be stored on the local machine. If data was stored on the NFS mounted device there is a risk of overwriting data, since this device is the same for all allocated networks.
- **Good capacity**
  - Emulab consists of 168 PCs and it does not seem to be heavily loaded, and it is free to use. This makes it a good alternative to a local lab.

6.2 Disadvantages of using Emulab

For this study there have not been many drawbacks of using Emulab. The prolonged execution time does not seem to have anything to do with Emulab, but is a result of the implementation of NFS in FreeBSD and could easily have been overcome by mounting the program on the local node.

The only potential drawback is if Emulab comes to be very popular, since then it would be heavily loaded. This will make it more difficult to allocate the resources needed. Anyhow, today this does not seem to be a problem.

6.3 Recommendation

It is risky to say much about Emulab from just having gathered limited experience from using it. There are still many features that have not been tested in the scope of this thesis. Nevertheless, the impression of Emulab is positive and the recommendation is that Emulab is a good contribution to the academic environment. Based on the experience from the experiments presented it is possible to say, without having tried all facilities of Emulab, that it works well and it is well worth to continue to use the platform.

The author recommends the research staff at Karlstad University to perform further experiments on this experimental platform. Another step is perhaps to gather experience from the real network PlanetLab.

7 Conclusion and future work

In this chapter some conclusions are drawn. Most of the conclusions concern usability and availability in Emulab. Furthermore, a few parts of the experiment that could have been done in a different way in the experiments are commented. Finally, suggestions for further work are presented.

7.1 Conclusion

In this thesis I have compared the throughput of SCTP and TCP. Many tests with different parameters have been performed and for each test several repetitions have been conducted.

From this study it is not easy to draw general conclusions from the comparison between TCP and SCTP, neither is it possible to see any tendencies. To do this further analysis of the results on packet level would have been required.

The conclusion from the results is that when there was no concurrent traffic in the network SCTP shows no significant improvement over TCP. There are some deviating results when the router buffer size was reduced, but also the analysis of these results was beyond the scope of this thesis. More important than the choice of network protocol seems to be whether SACK is used or not for reporting missing packets.

The experiments have been performed on the remote platform, Emulab, and the conclusion from this study is that Emulab is a well working platform. The documentation of Emulab is easy to read and it gives a clear overview of the network. As a beginner, it is easy to start experimenting on the Emulab platform, since there is an online tutorial available [37]. By reading this, one could easily receive all necessary information.

The fears that Emulab would be overloaded and that the availability therefore would be limited turned out to be wrong. Only twice during these studies there have not been enough nodes available.

The conclusion is that it is easy to use Emulab, the configuration of the experimental network is simple, fast and the availability is high. From this study the experience is that Emulab is a good contribution for many researchers and a good complement to a local lab.

7.2 Parts of the study that could have been done differently

A few problems have turned up during this study. The first and the most serious was the choice of SCTP implementation. Due to the change of SCTP implementation for the experiments in the local lab, it was not possible to make a comparison between the different results. The impact of the change of SCTP implementation was not obvious to the author until it was too late. More open eyes and a better communication would have prevented this problem.

This study has included many different tests with different parameters. These tests have produced enormous amounts of data. Analyzing all these data has been time consuming. A better approach for a study like this would have been to perform fewer tests and instead put more effort on the analysis. That would probably have given the possibility to find an answer to why throughput for SCTP was unexpectedly low under some circumstances.

7.3 Future work

Further analyses to find the reasons behind some of the results in this study could be one challenge for future work, and perhaps also to use slightly different parameters (lower delay and smaller router buffer), to see whether any tendencies could be found. To perform the tests by using another SCTP implementation could perhaps also give some more validity to whether the results are representative for SCTP or just for this implementation.

As a comparison to this study, it would be interesting to perform the same study again, but using several streams and several TCP connections for the transport of the emulated SIP messages.

Another approach would be to compare throughput for TCP, SCTP and UDP. The results from such a study could serve as a more complete basis of whether a reliable transport protocol is better than UDP for the transmission of several packets between proxies in the SIP scenario.

If interesting results are found and verified in the emulated environment provided by Emulab another challenge would be to perform the same study in PlanetLab to see if the results are the same in a live network.

To get more results from Emulab it would be interesting to run an experiment that is run on another operating system than FreeBSD or Linux. This would require loading an OS image apart from the default ones. Such an activity would further indicate whether the Emulab documentation is good enough.

References

[1] Alamgir R Et al. *Effect of Congestion Control on the performance of TCP and SCTP over Satellite Networks*. NASA Earth Science Technology Conference. June 2002

[2] Allman M et al. RFC 2581 *TCP Congestion Control*. April 1999

[3] Amir Y, Danilov C. *Reliable Communication in Overlay Networks.* Proceedings of the IEEE International Conference on Dependable Systems and Networks, San Francisco, June 2003.

[4] Andersson T. An Evaluation: Multiple TCP connections vs Single SCTP Association with Multiple Streams. Karlstad University, October 2003

[5] Camarillo G, Schulzrinne H, Kantola R. *Signalling transport protocols*. Technical report, Dept. of Computer Science, Columbia University, CUCS-002-02, February 2002

[6] Fall K. *Network Emulation in the Wint/NS Simulator*. In Proc. IEEE ISCC '99, July 1999

[7] Fu S. Atiquzzaman M. *Improving End-to-End throughput of Mobile IP using SCTP*. 2003 Workshop on high performance switching and routing, Torino. June 2003

[8] Handley M, Jacobson V, *SDP Session Description Protocol RFC 2327*, April 1998

[9] Kurose J K. *Computer Networking A top-down approach featuring the Internet*. Pearson Education 2003

[10] Leprau J., et al. *An Emulation Testbed for Networks and Distributed Systems* http://www.cs.utah.edu/flux/testbed-docs/testbed-intel-jun01.htm September 2003

[11] Mathis M. et al. RFC 2018 *TCP Selective Acknowledgment Options*. October 1996

[12] Nagle J. RFC 896 *Congestion Control in IP/TCP Internetworks*. January 1984

[13] Postel J. RFC 768: *UDP User Datagram Protocol*. August 1980

[14] Ravier T., Brennan R., Curran T. *Experimental studies of SCTP Multi-homing*. First Joint IEI/IEE Symposium on Telecommunications Systems Research, Dublin, November 2001

[15] RFC 793: *TCP Transmission Control Protocol*. September 1981

[16] Rizzo L., *Dummynet and Forward error correction*. In Proc. Of the 1998 USENIX Technical Conf. New Orleans, US, June 1998

[17] Schulzrinne H. et al. RFC 2543 *SIP: Session initiation protoco*l. March 1999

[18] Singh M, Pradhan P, Francis P. *MPAT: Aggregate TCP Congestion Management as a Building Block for Internet QoS* In Proceedings of IEEE International Conference on Network Protocols (ICNP 2004), Berlin, Germany, October 2004.

[19] Stewart R. et al. RFC 2960 *Stream Control Transmission Protocol*. October 2000

[20] Stewart R. *Stream Control Transmission Protocol (SCTP).* http://www.sctp.org September 2003

[21] Stewart R. *Stream Control Transmission Protocol (SCTP)-a reference guide*. Adison Wesley, November 2001

[22] *SCTP-a prototype implementation* http://www.sctp.de September 2003

[23] White B., Lepreau J., Stoller L.,Ricci R., Guruprasad S., Newbold M., Hibler M., Barb C.and Joglekar A.. *An Integrated Experimental Environment for Distributed Systems and Networks*. 5th Symposium on Operating Systems Design & Implementation, Boston, US, December 2002

[24] *Emulab at Georgia Institute of Technology* http://www.netlab.cc.gatech.edu/ September 2003

[25] *Emulab at University of Kentucky* http://www.uky.emulab.net/ September 2003

[26] *Emulab at University of Utah* http://www.emulab.net/ September 2003

[27] FreeBSD implementing SCTP http://www.freeBSD.org November 2004

[28] *Linux 2.4 auto-tuning/caching* http://www.csm.ornl.gov/~dunigan/net100/auto.html February 2004

[29] *Linux Kernel SCTP* http://sourceforge.net/projects/lksctp September 2003

[30] *Planetlab* http://www.planet-lab.org/ September 2003

[31] *Policies for using Emulab* http://www.emulab.net/docwrapper.php3?docname=policies.html September 2004

[32] *Secure sockets layer* http://www.openssl.org/ September 2004

[33] *The Internet Engineering Task Force* http://www.ietf.org/ September 2003

[34] http://www.sctp.org/implementations.html May 2004

[35] http://www.cisco.com/warp/public/cc/pd/wr2k/cpbn/tech/vlan_wp.htm September 2004

[36] http://www.ietf.org/html.charters/tsvwg-charter.html September 2004

[37] http://www.emulab.net/tutorial/docwrapper.php3?docname=tutorial.html September 2003

[38] http://www.stanford.edu/class/ee384y/Handouts/EE384y_BufferSize.pdf February 2002

[39] http://www.emulab.net/doc/docwrapper.php3?docname=expubs.html November 2004

Appendix

A Information presenting details about allocated resources at Emulab

Your experiment `firsttest' in project `SCTP-KaU' has been started. Here is the experiment summary detailing the nodes that were allocated to you. You may use the `Qualified Name' to log on to your nodes. See /etc/hosts on your nodes (when running FreeBSD, Linux, or NetBSD) for the IP mapping on each node.

User: Johan Eklund
EID: firsttest
PID: SCTP-KaU
GID: SCTP-KaU
Name: first test to test an application
Swappable: Yes
Idle-Swap: Yes, at 1 hours
Auto-Swap: Yes, at 10 hours
Created: 2003-09-24 06:10:18
Directory: /proj/SCTP-KaU/exp/firsttest

Appended at the end is the output of the experiment setup. If you have any questions or comments, please include the output below in your message to testbed-ops@flux.cs.utah.edu

--------- firsttest.report --------
Experiment: SCTP-KaU/firsttest
State: active

Virtual Node Info:

ID	Type	OS	Qualified Name
node0	pc		node0.firsttest.SCTP-KaU.emulab.net
node1	pc		node1.firsttest.SCTP-KaU.emulab.net
node2	pc		node2.firsttest.SCTP-KaU.emulab.net
node3	pc		node3.firsttest.SCTP-KaU.emulab.net

Physical Node Mapping:

```
ID             Type        OS             Physical
-------------- ----------- -------------- -----------
node0          pc850       RHL73-STD      pc112
node1          pc850       RHL73-STD      pc113
node2          pc850       RHL73-STD      pc114
node3          pc850       RHL73-STD      pc110
tbdelay0       pc850       FBSD47-STD     pc111
tbdelay1       pc850       FBSD47-STD     pc149

Virtual Lan/Link Info:
ID             Member         IP/Mask        Delay     BW (Kbs)  Loss Rate
-------------- -------------- -------------- --------- --------- -----
lan0           node0:0        10.1.2.3       25.00     100000    0.100
                              255.255.255.0  25.00     100000    0.100
lan0           node1:0        10.1.2.4       25.00     100000    0.100
                              255.255.255.0  25.00     100000    0.100
lan0           node2:1        10.1.2.2       25.00     100000    0.100
                              255.255.255.0  25.00     100000    0.100
link0          node2:0        10.1.1.3       100.00    100000    0.010
                              255.255.255.0  100.00    100000    0.010
link0          node3:0        10.1.1.2       100.00    100000    0.010
                              255.255.255.0  100.00    100000    0.010

Virtual Queue Info:
ID             Member         Q Limit    Type    weight/min_th/max_th/linterm
-------------- -------------- ---------- ------- ----------------------
lan0           node0:0        50s        Tail    0/0/0/0
lan0           node1:0        50s        Tail    0/0/0/0
lan0           node2:1        50s        Tail    0/0/0/0
link0          node2:0        50s        Tail    0/0/0/0
link0          node3:0        50s        Tail    0/0/0/0

Physical Lan/Link Info:
ID             Member         Delay Node   Delay    BW (Kbs) PLR    Pipe
-------------- -------------- ------------ -------- -------- ------ ---
lan0           node0          tbdelay0     25.00    100000   0.100  130
                                           25.00    100000   0.100  140
lan0           node1          tbdelay1     25.00    100000   0.100  130
                                           25.00    100000   0.100  140
lan0           node2          tbdelay1     25.00    100000   0.100  110
```

			25.00	100000	0.100	120
link0	node2	tbdelay0	200.00	100000	0.020	110
link0	node3	tbdelay0	200.00	100000	0.020	120

Physical Queue Info:

ID	Member	Q Limit	Type	weight/min_th/max_th/linterm
lan0	node0	50s	Tail	0/0/0/0
lan0	node1	50s	Tail	0/0/0/0
lan0	node2	50s	Tail	0/0/0/0
link0	node2	50s	Tail	0/0/0/0
link0	node3	50s	Tail	0/0/0/0

B Test parameters used in the experiments

The parameter settings marked on this page are the same parameters as were used in the experiment in the local lab, the settings on next page are the ones added for this experiment.

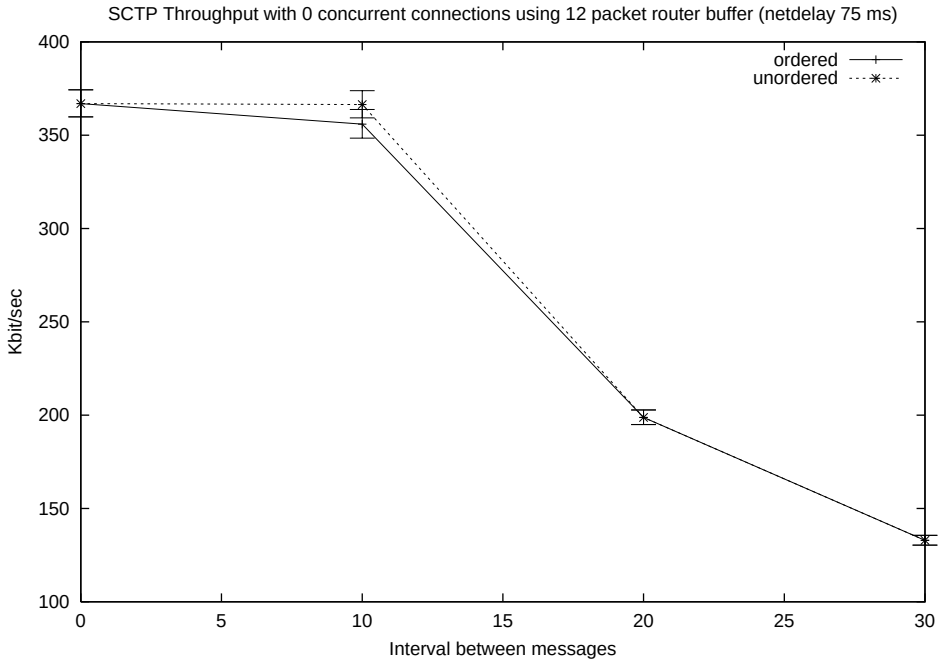
Test number	Competing traffic	message interval	net delay (ms)	Router Buffer
1	0	0	75	12
2	1	0	75	12
3	2	0	75	12
4	8	0	75	12
5	0	10	75	12
6	1	10	75	12
7	2	10	75	12
8	8	10	75	12
9	0	20	75	12
10	1	20	75	12
11	2	20	75	12
12	8	20	75	12
13	0	30	75	12
14	1	30	75	12
15	2	30	75	12
16	8	30	75	12
17	0	0	75	32
18	1	0	75	32
19	2	0	75	32
20	8	0	75	32
21	0	10	75	32
22	1	10	75	32
23	2	10	75	32
24	8	10	75	32
25	0	20	75	32
26	1	20	75	32
27	2	20	75	32
28	8	20	75	32
29	0	30	75	32
30	1	30	75	32
31	2	30	75	32
32	8	30	75	32

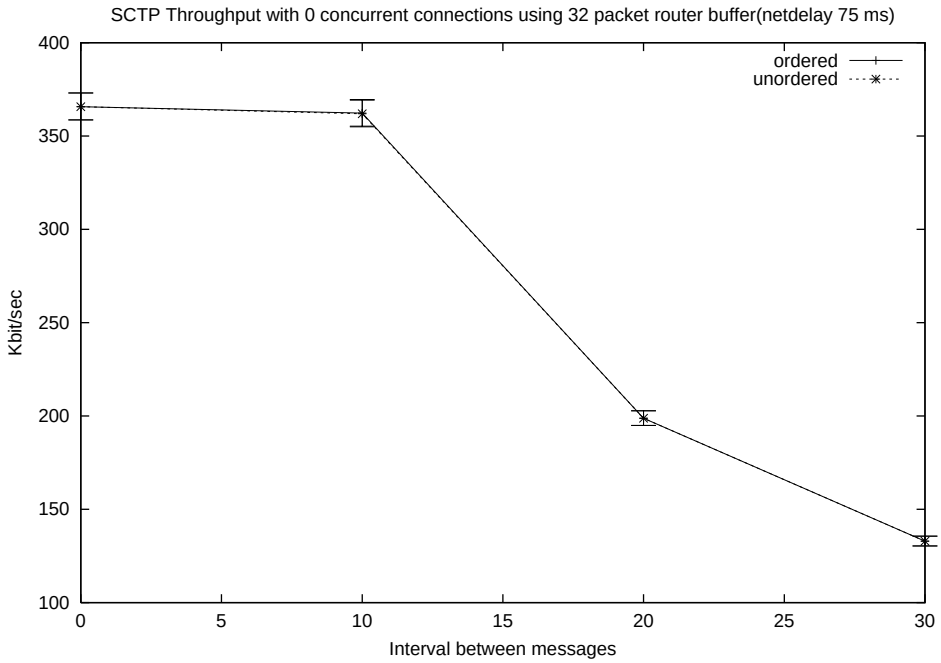
Test number	Competing traffic	message interval	net delay (ms)	Router Buffer
33	0	0	25	6
34	2	0	25	6
35	8	0	25	6
36	0	10	25	6
37	2	10	25	6
38	8	10	25	6
39	0	20	25	6
40	2	20	25	6
41	8	20	25	6
42	0	30	25	6
43	2	30	25	6
44	8	30	25	6
45	0	0	25	12
46	2	0	25	12
47	8	0	25	12
48	0	10	25	12
49	2	10	25	12
50	8	10	25	12
51	0	20	25	12
52	2	20	25	12
53	8	20	25	12
54	0	30	25	12
55	2	30	25	12
56	8	30	25	12
57	0	0	25	32
58	2	0	25	32
59	8	0	25	32
60	0	10	25	32
61	2	10	25	32
62	8	10	25	32
63	0	20	25	32
64	2	20	25	32
65	8	20	25	32
66	0	30	25	32
67	2	30	25	32
68	8	30	25	32

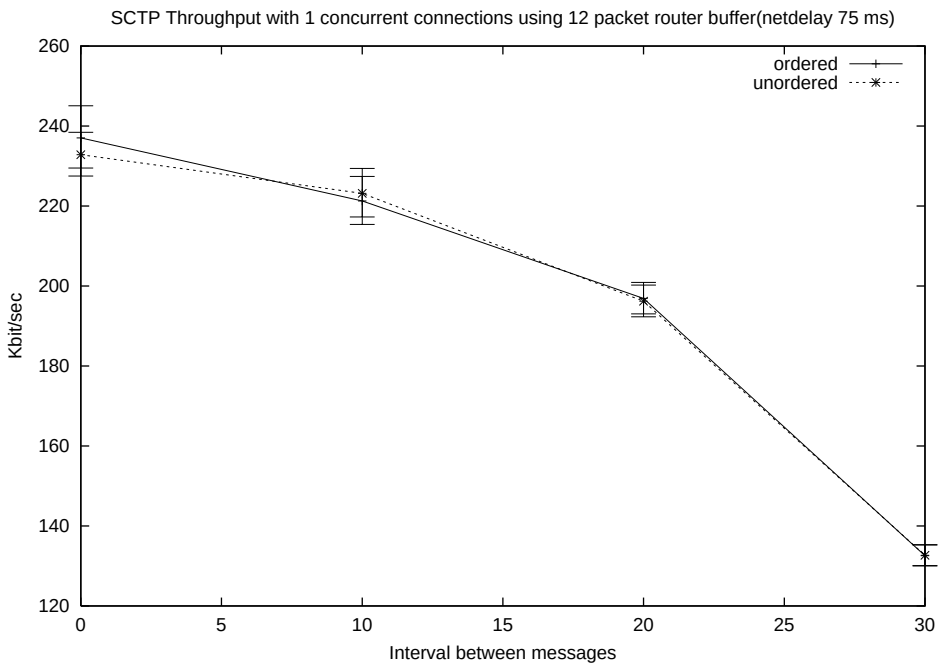
C Results from the experiment running SCTP performed in the local lab

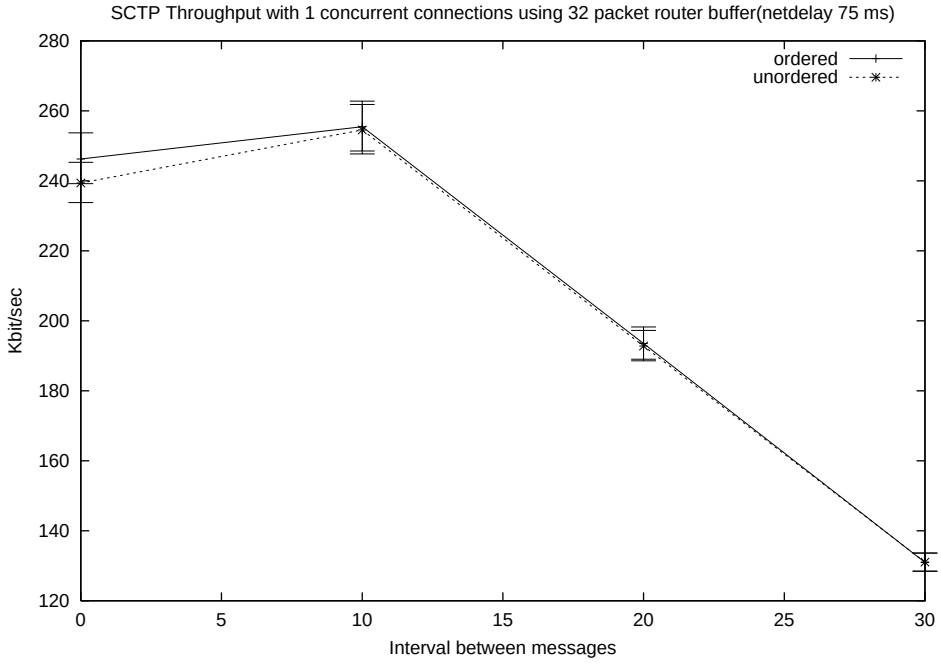
(Local lab = “CARL” at Karlstad university)

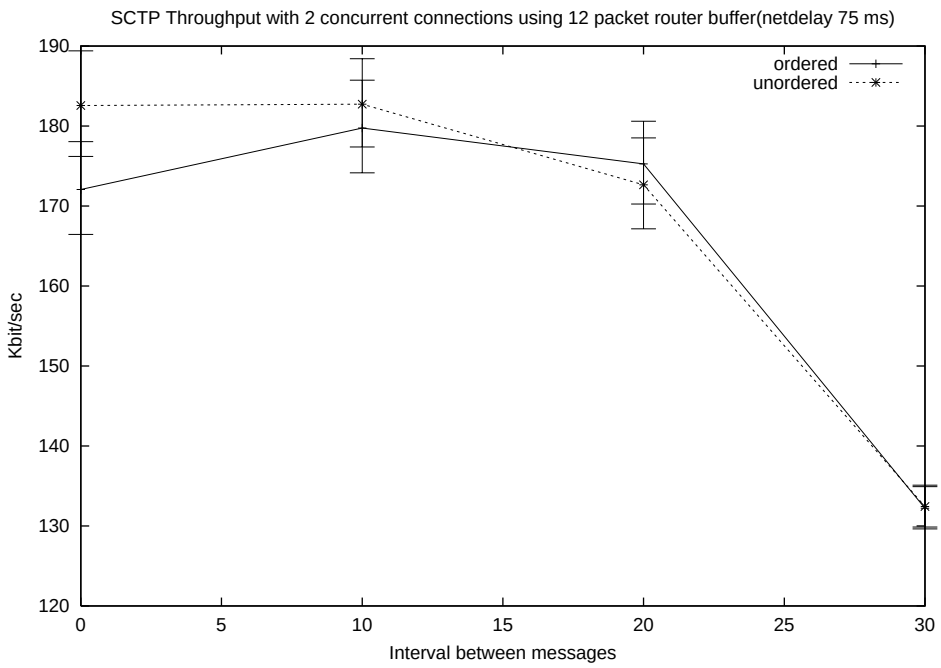
The graphs in this appendix show the results from the experiment performed in the Local lab. The graphs are to be compared to the results achieved in this thesis.

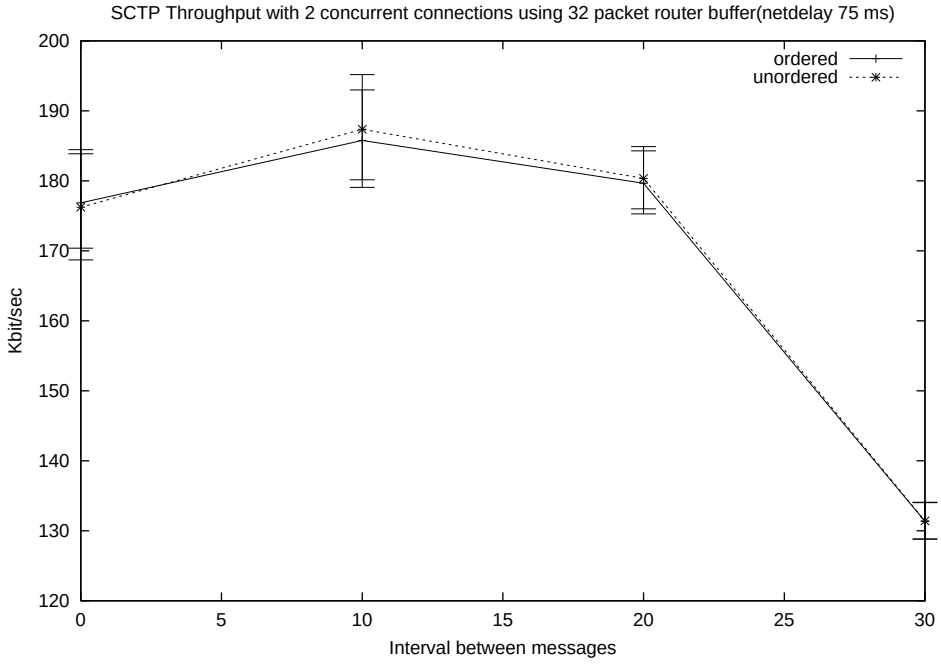


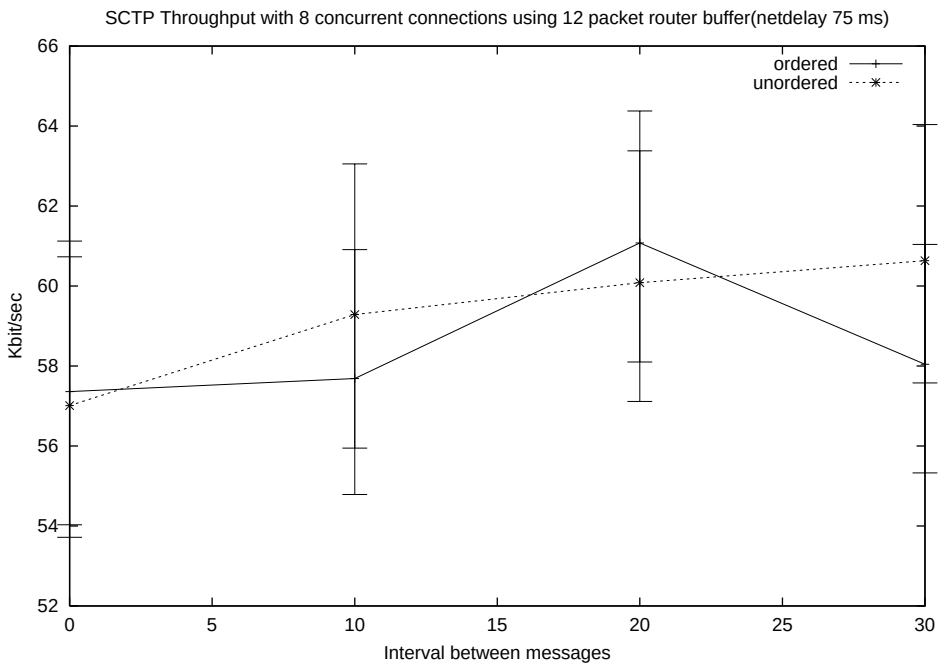


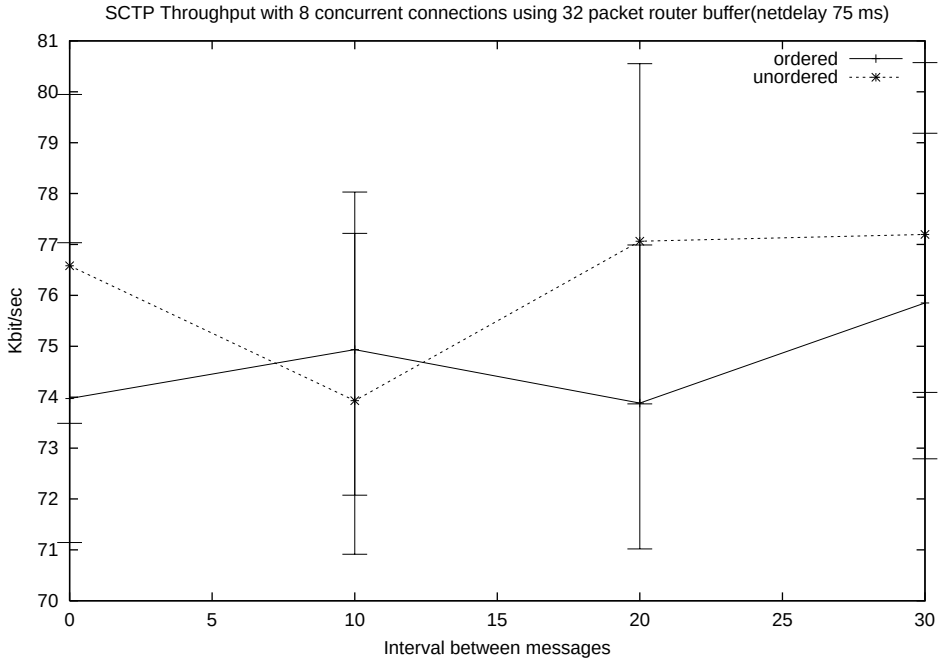

SCTP Throughput with 1 concurrent connections using 12 packet router buffer(netdelay 75 ms)
ordered
unordered
Kbit/sec
Interval between messages
260
240
220
200
180
160
140
120
0
5
10
15
20
25
30


SCTP Throughput with 1 concurrent connections using 32 packet router buffer(netdelay 75 ms)
ordered
unordered
Kbit/sec
Interval between messages
280
260
240
220
200
180
160
140
120
0
5
10
15
20
25
30

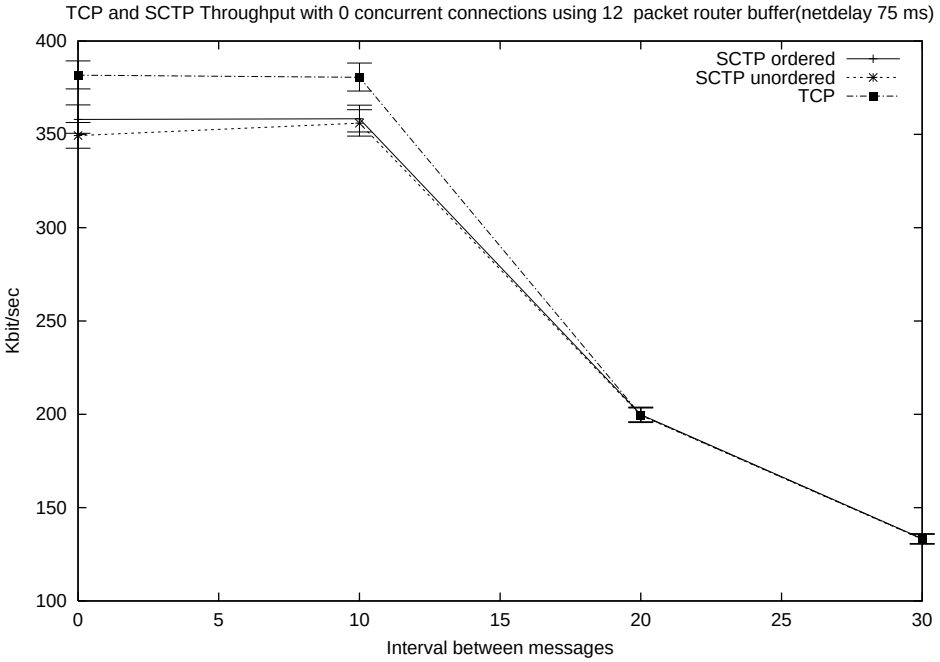

SCTP Throughput with 2 concurrent connections using 12 packet router buffer(netdelay 75 ms)
ordered
unordered
Kbit/sec
Interval between messages
190
180
170
160
150
140
130
120
0
5
10
15
20
25
30

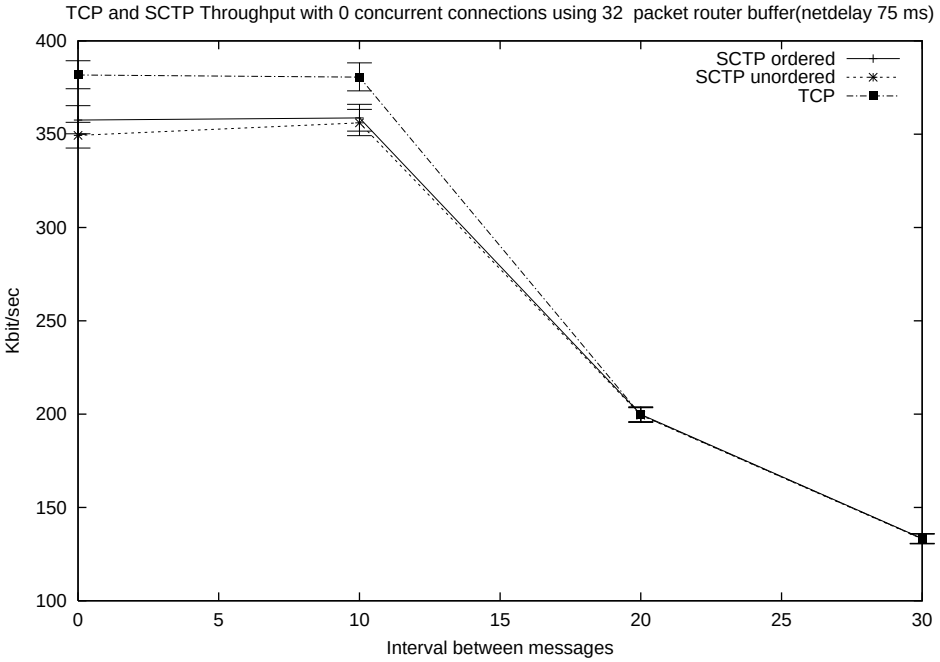

SCTP Throughput with 2 concurrent connections using 32 packet router buffer(netdelay 75 ms)
ordered
unordered
Kbit/sec
Interval between messages
200
190
180
170
160
150
140
130
120
0
5
10
15
20
25
30

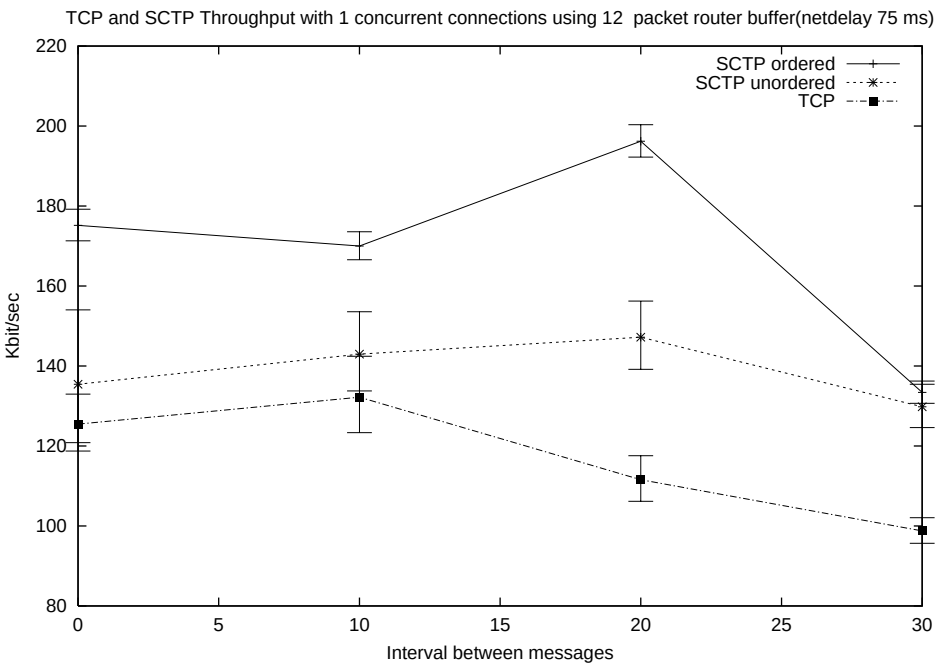

SCTP Throughput with 8 concurrent connections using 12 packet router buffer(netdelay 75 ms)
ordered
unordered
Kbit/sec
Interval between messages
66
64
62
60
58
56
54
52
0
5
10
15
20
25
30

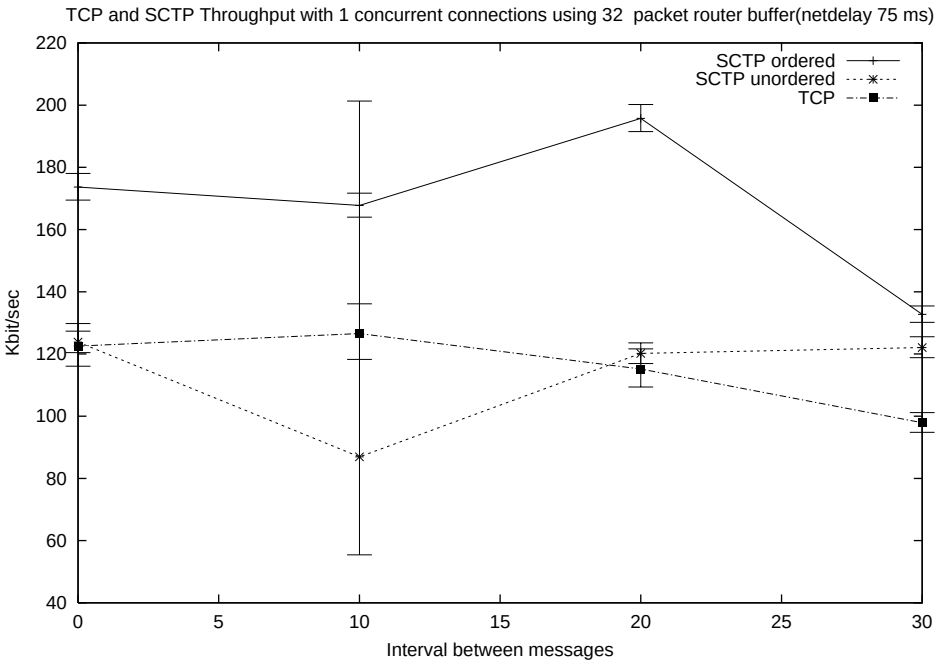

SCTP Throughput with 8 concurrent connections using 32 packet router buffer(netdelay 75 ms)
ordered
unordered
Kbit/sec
Interval between messages
81
80
79
78
77
76
75
74
73
72
71
70
0
5
10
15
20
25
30

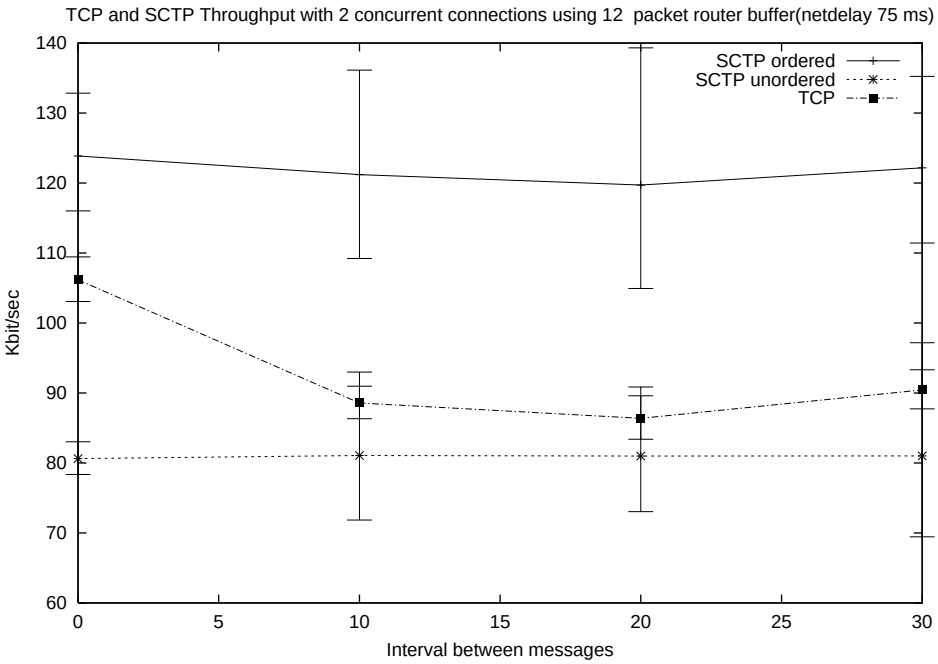
D Results from the experiment running SCTP and TCP performed on Emulab

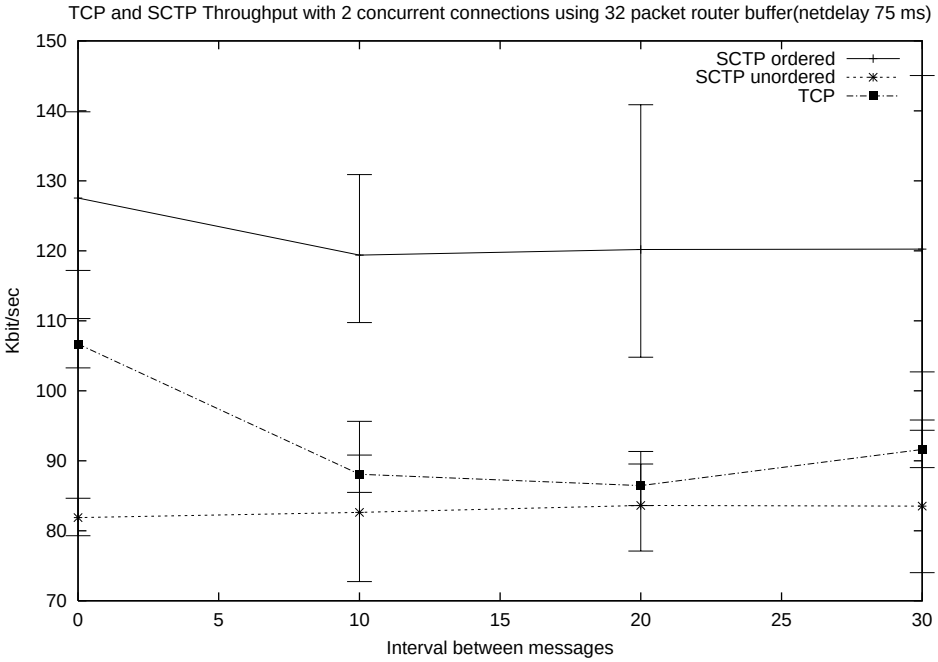


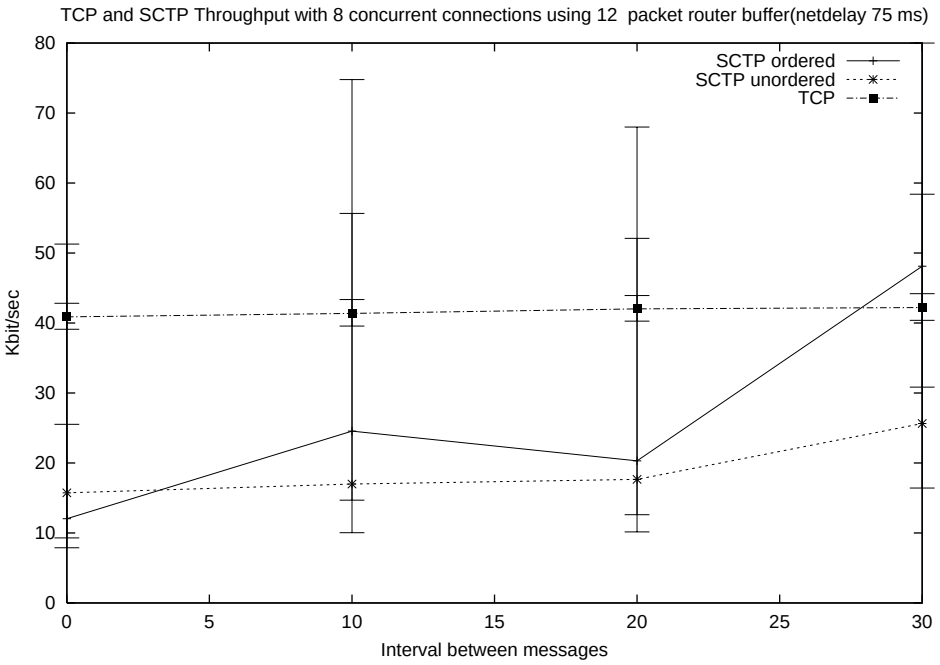


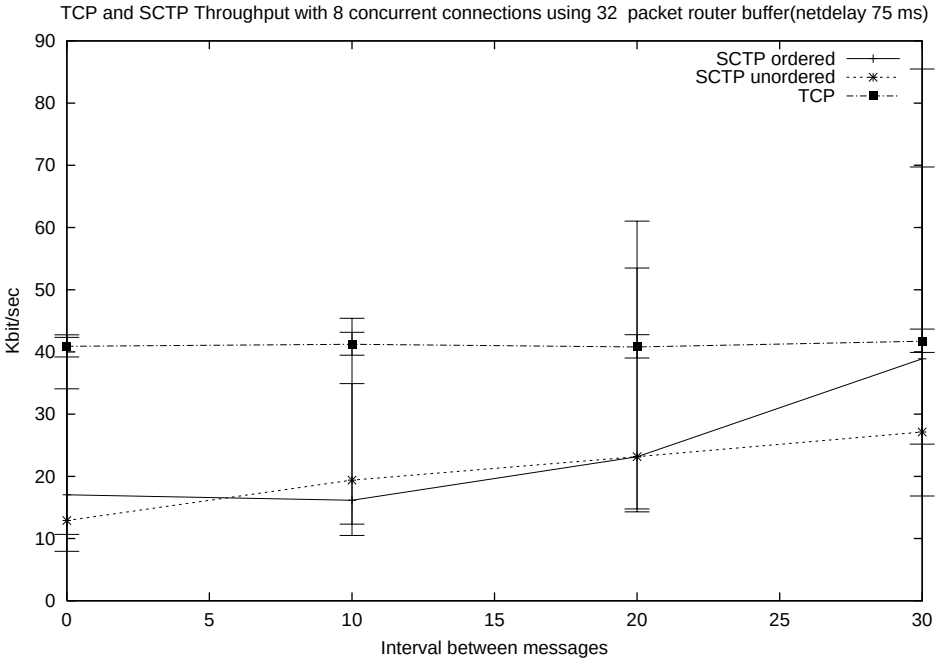

TCP and SCTP Throughput with 1 concurrent connections using 12 packet router buffer(netdelay 75 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
220
200
180
160
140
120
100
80
0
5
10
15
20
25
30

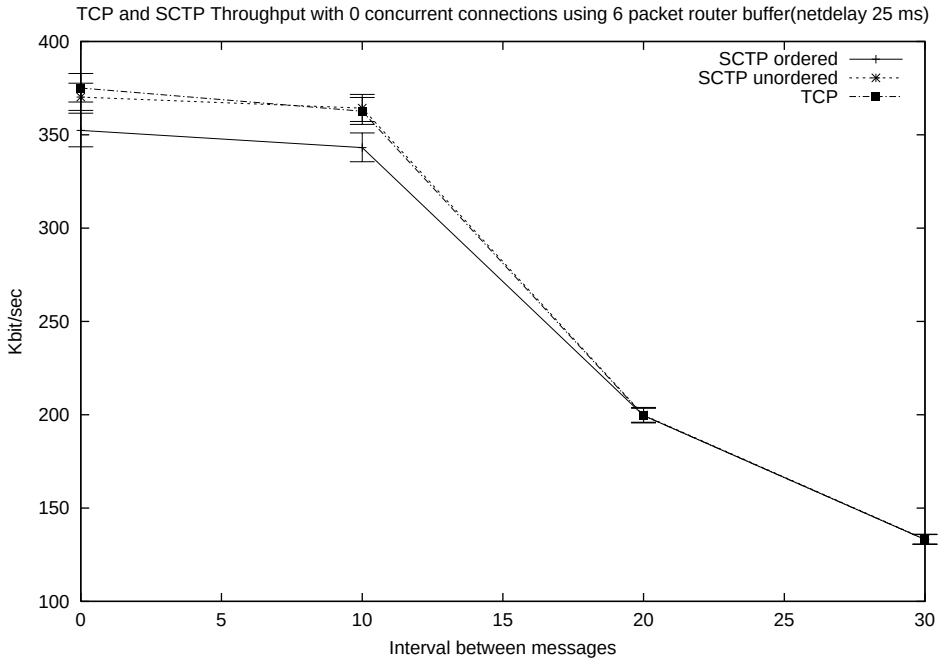

TCP and SCTP Throughput with 1 concurrent connections using 32 packet router buffer(netdelay 75 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
220
200
180
160
140
120
100
80
60
40
0
5
10
15
20
25
30

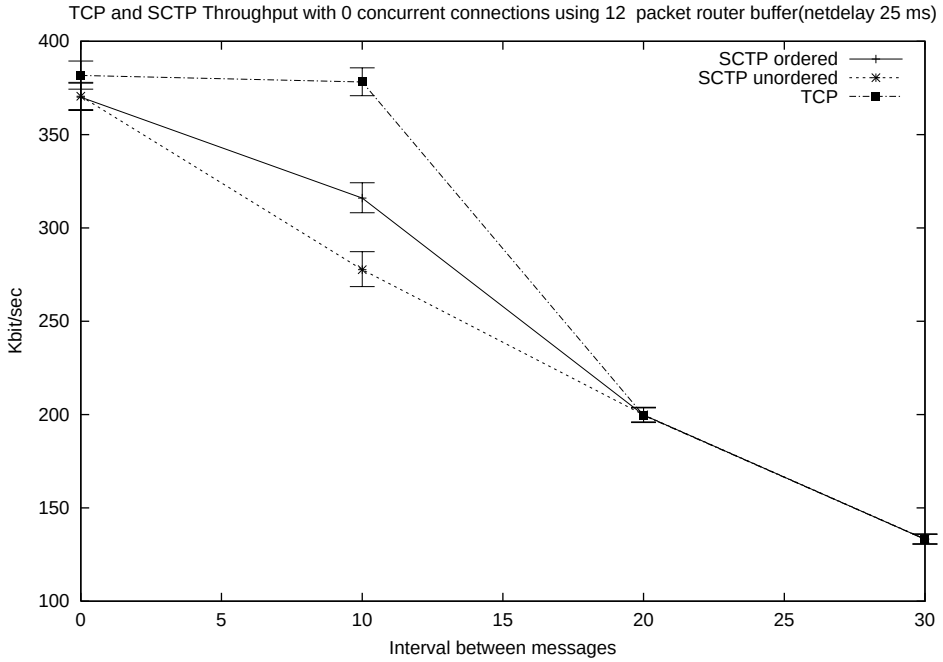

TCP and SCTP Throughput with 2 concurrent connections using 12 packet router buffer(netdelay 75 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
140
130
120
110
100
90
80
70
60
0
5
10
15
20
25
30

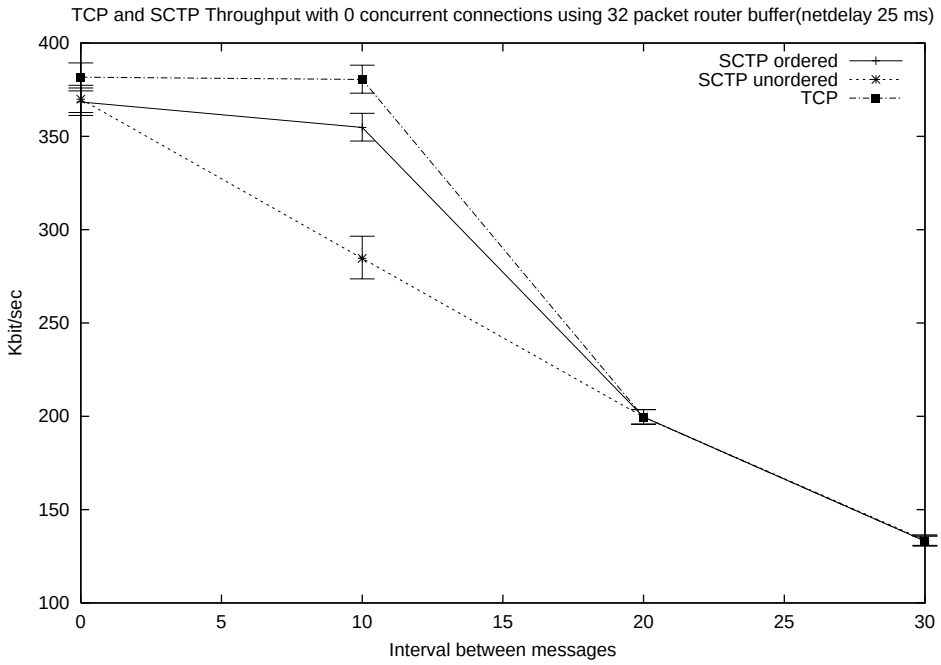

TCP and SCTP Throughput with 2 concurrent connections using 32 packet router buffer(netdelay 75 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
150
140
130
120
110
100
90
80
70
0
5
10
15
20
25
30

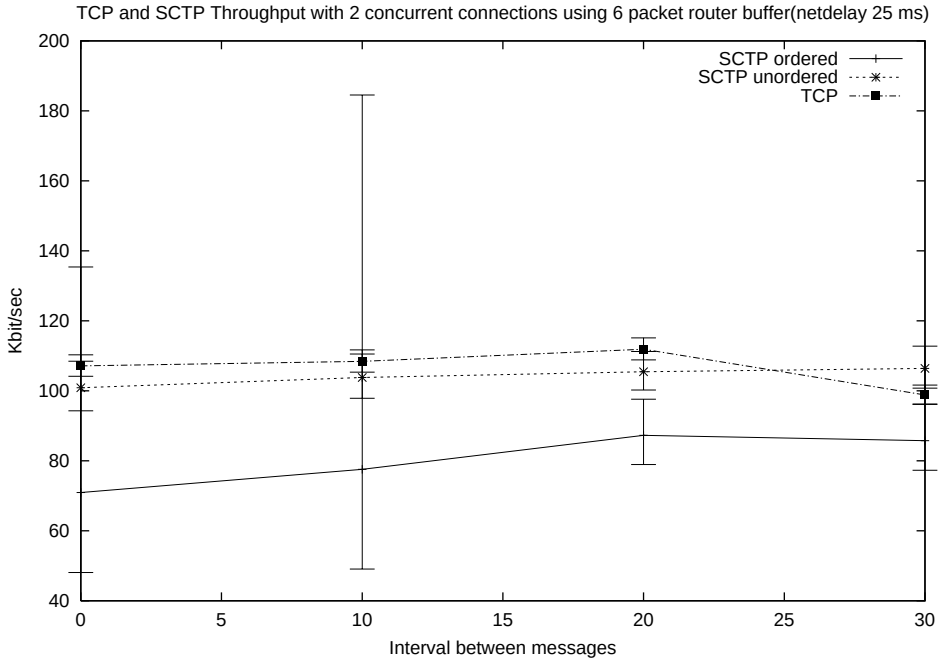

TCP and SCTP Throughput with 8 concurrent connections using 12 packet router buffer(netdelay 75 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
0
10
20
30
40
50
60
70
80
0
5
10
15
20
25
30

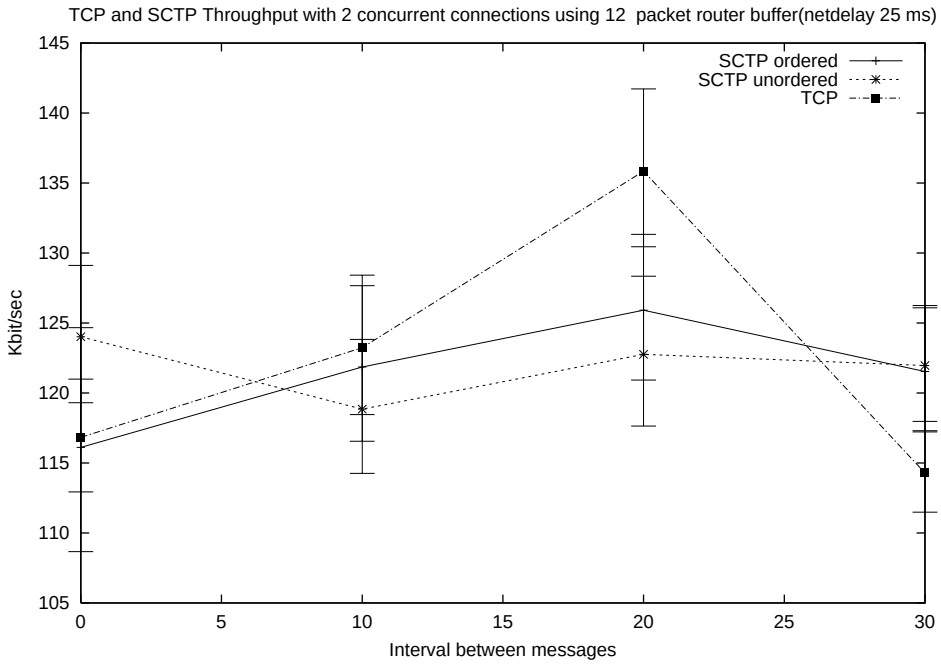

TCP and SCTP Throughput with 8 concurrent connections using 32 packet router buffer(netdelay 75 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
0
10
20
30
40
50
60
70
80
90
0
5
10
15
20
25
30

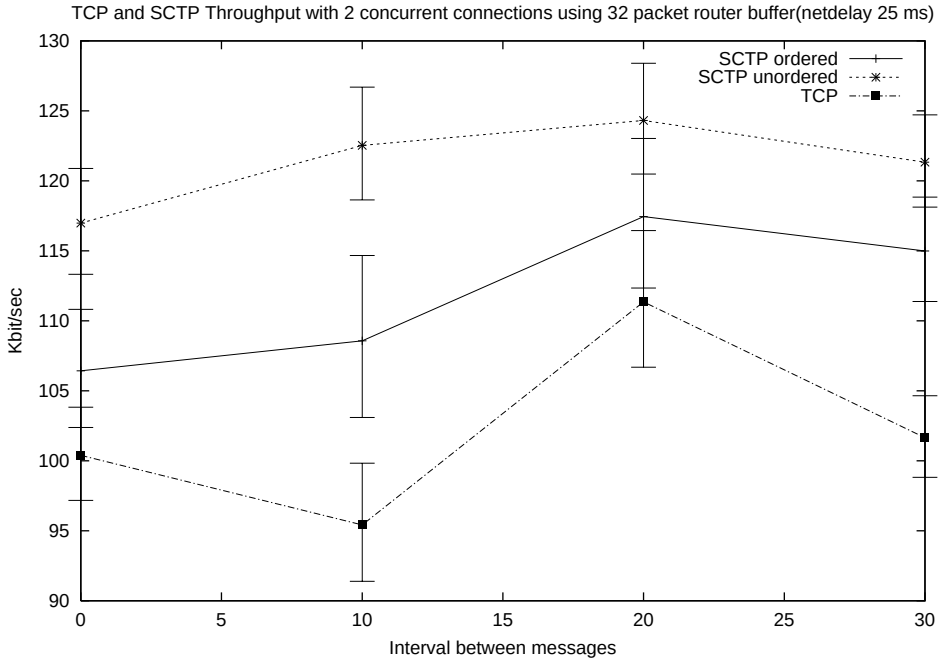

TCP and SCTP Throughput with 0 concurrent connections using 6 packet router buffer(netdelay 25 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
400
350
300
250
200
150
100
0
5
10
15
20
25
30

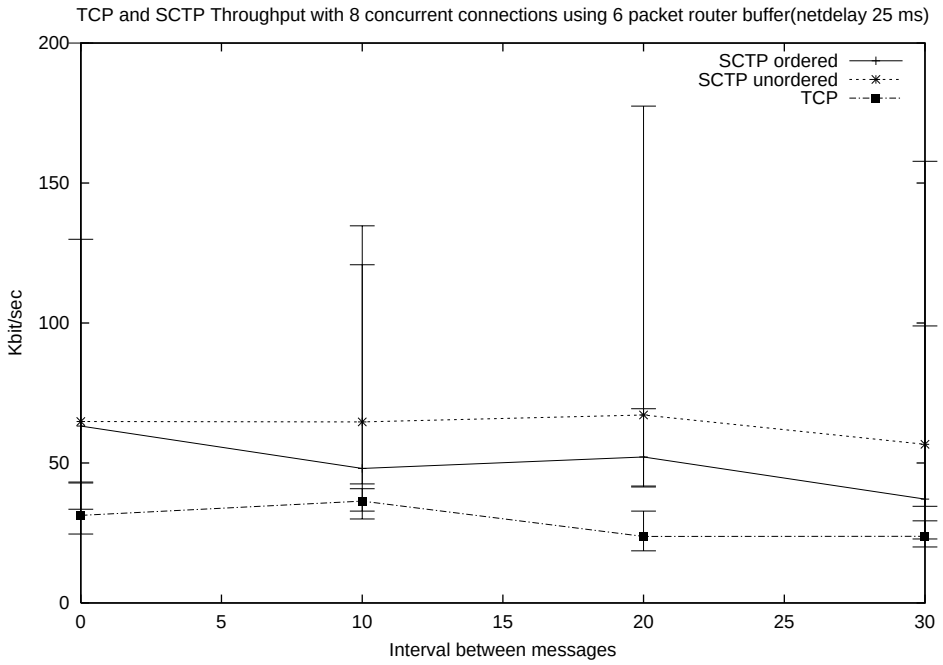

TCP and SCTP Throughput with 0 concurrent connections using 12 packet router buffer(netdelay 25 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
400
350
300
250
200
150
100
0
5
10
15
20
25
30

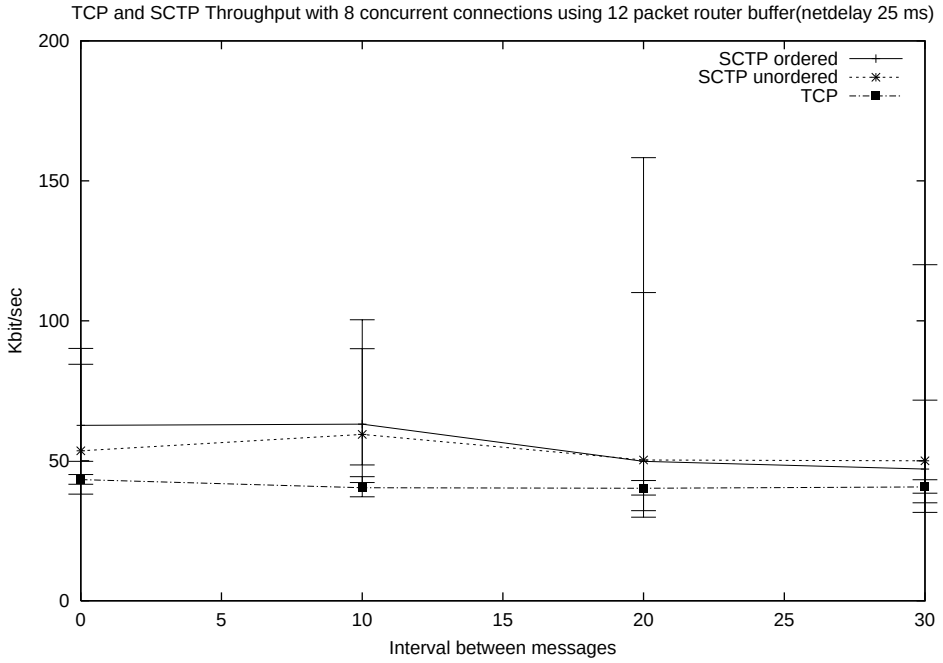

TCP and SCTP Throughput with 0 concurrent connections using 32 packet router buffer(netdelay 25 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
400
350
300
250
200
150
100
0
5
10
15
20
25
30

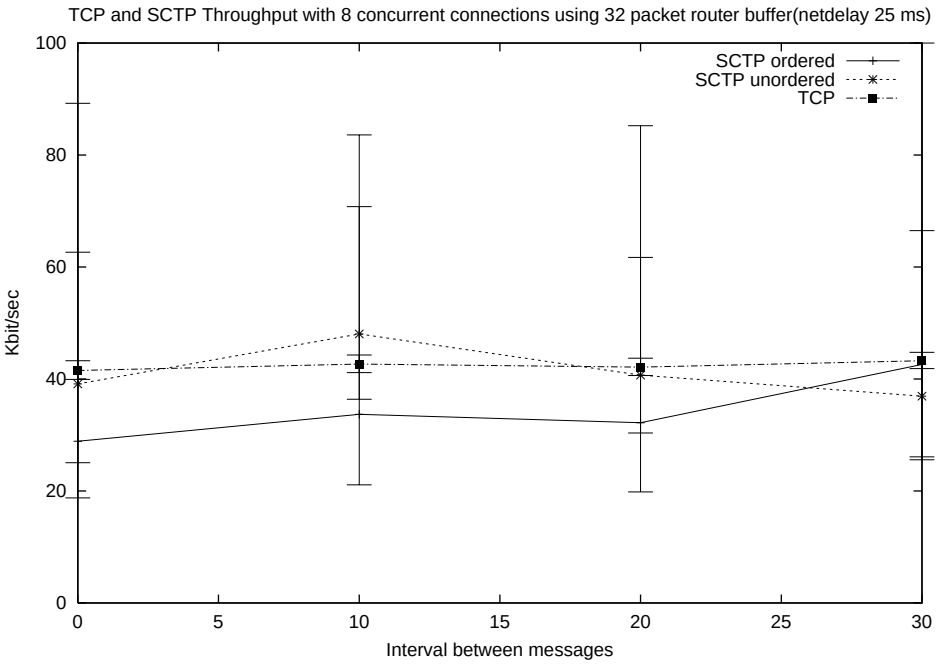

TCP and SCTP Throughput with 2 concurrent connections using 6 packet router buffer(netdelay 25 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
200
180
160
140
120
100
80
60
40
0
5
10
15
20
25
30


TCP and SCTP Throughput with 2 concurrent connections using 12 packet router buffer(netdelay 25 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
145
140
135
130
125
120
115
110
105
0
5
10
15
20
25
30


TCP and SCTP Throughput with 2 concurrent connections using 32 packet router buffer(netdelay 25 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
130
125
120
115
110
105
100
95
90
0
5
10
15
20
25
30


TCP and SCTP Throughput with 8 concurrent connections using 6 packet router buffer(netdelay 25 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
0
50
100
150
200
0
5
10
15
20
25
30


TCP and SCTP Throughput with 8 concurrent connections using 12 packet router buffer(netdelay 25 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
0
50
100
150
200
0
5
10
15
20
25
30


TCP and SCTP Throughput with 8 concurrent connections using 32 packet router buffer(netdelay 25 ms)
SCTP ordered
SCTP unordered
TCP
Kbit/sec
Interval between messages
0
20
40
60
80
100
0
5
10
15
20
25
30