SCTP vs MPTCP:

Evaluating Concurrent Multipath Protocols

Anthony Trinh and Richard Zieminski

Department of Computer Science, Columbia University

{ akt2105, rez2107 }@columbia.edu

Abstract

Using multiple network interfaces simultaneously has been shown to significantly increase network bandwidth. We evaluated two application-layer techniques concurrent multipath protocols, including Stream Control Transmission Protocol (SCTP) as well as Multipath TCP (MPTCP). Our experiments showed that SCTP has far higher throughput than MPTCP but at the cost of excessive CPU utilization. MPTCP might be the choice protocol for concurrent multipath transfer due to its backward compatibility with existing TCP applications and for its overall performance.

Contents

[1 Introduction 1](#_Toc312212255)

[2 Background 1](#_Toc312212256)

[2.1 TCP 1](#_Toc312212257)

[2.2 MPTCP 1](#_Toc312212258)

[2.3 SCTP 2](#_Toc312212259)

[3 Evaluation 3](#_Toc312212260)

[3.1 System Setup 3](#_Toc312212261)

[3.1.1 Network Topology 3](#_Toc312212262)

[3.1.2 Network Delay Settings 3](#_Toc312212263)

[3.1.3 Operating System Settings 4](#_Toc312212264)

[3.1.4 Hardware Specifications 5](#_Toc312212265)

[3.1.5 Software Test Tools 5](#_Toc312212266)

[3.2 Methodology 6](#_Toc312212267)

[3.3 Results 7](#_Toc312212268)

[4 Conclusion 13](#_Toc312212269)

[5 References 14](#_Toc312212270)

[6 Appendix 16](#_Toc312212271)

[6.1 Shell Scripts 16](#_Toc312212272)

[6.2 Emulab NS Scripts 23](#_Toc312212273)

# Introduction

The communication industry is going through a period of explosive change, where the available content on the Internet has grown exponentially. While content increases there also needs to be a change in the way to effectively access this content. Legacy protocols such as the Transmission Control Protocol are no longer efficient enough to handle the massive amount of data. To address these needs, newer methods are being introduced which both upgrade the transport methodologies, as well as provide more reliable mechanisms for transfer. In this paper, we will be investigating two of these protocols: Stream Control Transmission Protocol and Multipath Transmission Control Protocol.

# Background

## TCP

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet. It is one of the most widely used protocols, used for everything from web page access to file transfer and VPN connectivity. It provides for reliable delivery of data by using a mechanism of a positive acknowledgement (ACK) with retransmission system, in addition to flow control mechanisms in place to ensure no data is lost. With TCP, there is an orderly flow of bytes between hosts whereby data is transferred as a sequential stream flow of data between sender and receiver.

## MPTCP

Multipath Transmission Control Protocol (MPTCP) is an extension to the TCP protocol whereas multiple paths to a regular TCP session are used. The idea is to allow for increased throughput, as well as network resiliency. Using congestion control algorithms, MPTCP will ensure load balancing such that no worse than standard TCP throughput performance can be observed. The sub flow nature of MPTCP though makes it more prone to higher delay and jitter, which may occur due to the data arrival being dependent on another sub flow. Because of this it may not be suitable for all applications, such as VOIP or streaming video. Low level modifications to the TCP stack are necessary to support MPTCP, but it is backward compatible to both the application and network layers. Since MPTCP requires modifications to the TCP stack at both peers, it is generally more difficult to implement. Like SCTP, MPTCP can also take advantage of multi-homing, or multiple interface usage, to allow a single MPTCP association to run across multiple paths. As additional option fields in the TCP header are utilized, traffic may not pass through all middle boxes on the internet. What this means is without Internet infrastructure support, not all hosts can utilize MPTCP. While transparent, additional coding needs to occur to fully take advantage of the protocol benefits. Finally, failover is supported by design as multiple interfaces are required to implement MPTCP.

## SCTP

Stream Control Transmission Protocol (SCTP) is a multi-stream transport layer protocol that supports multiple independent streams per connection. While similar to TCP in that it provides a reliable, full duplex connection, it has significant differences. Unlike TCP where data delivery is purely stream-oriented, SCTP uses multi-byte chunks of data in sequence. Additionally, SCTP is able to deliver out-of-order data in parallel, which avoids head-of-line blocking as seen with TCP byte streams, to allow for greater throughput realizations. SCTP can also take advantage of multi-homing, or multiple interface usage, at the transport layer, to allow a single SCTP association to run across multiple paths. Interfaces can be through any combination of wired, wireless, and even multiple ISP based. Core support for SCTP is becoming mainstream with many Linux distributions having this natively present. This is not a transparent protocol however, and legacy code requires application layer changes to realize the benefits. Finally, failover mechanisms can also be supported through multi-homing where one interface can seamlessly switch in for another interface in failure. Currently, SCTP uses multi-homing for redundancy purposes only; the RFC2960 specification does not allow simultaneous transfer of new data to multiple destination addresses.

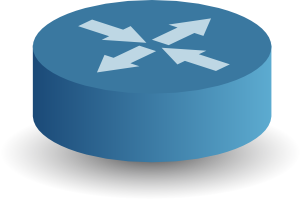
# Evaluation

## System Setup

### Network Topology

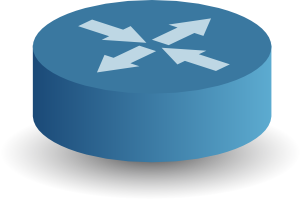
We used *Emulab* to emulate the network, shown in Figure 1, which creates a two-node network with 1 wired connection (Ethernet at 100 Mbit/s) and 3 software-delayed connections that mimic wireless bandwidth. That is, *wlan1*, *wlan2*, and *wlan3* are “Delay Nodes”, running *Dummynet* , all configured identically to simulate the bandwidth and latency of a specific wireless technology (based on the current test), including Wi-Fi 802.11g, 3G-HSPA+, and 4G-LTE.

Ethernet (*lan0*)



Delay Node

(*wlan1*)



Server (*node1*)

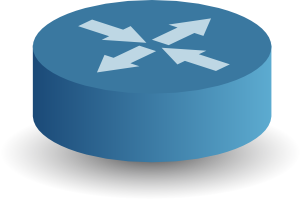


Client (*node0*)



Delay Node

(*wlan2*)



Delay Node

(*wlan3*)

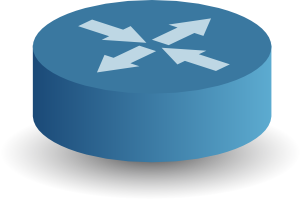


Figure : Emulated network topology

### Network Delay Settings

*Emulab* configured the Delay Nodes to match our bandwidth specifications (Table 1), and we did not directly interact with those nodes.

Table : Settings for *Delay Nodes* simulation[[1]](#footnote-1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Technology** | **Bandwidth (Mbit/s)** | | **Delay**  **(ms)** | **Packet Loss Ratio (%)** |
| **Downlink** | **Uplink** |
| Ethernet (ref) | 100 | 100 | 2 | 1 |
| 4G-LTE | 100 | 100 | 30 | 5 |
| 3G-HSPA+ | 56 | 22 | 80 | 9 |
| 802.11g | 54 | 54 | 2 | 2 |

### Operating System Settings

As shown in Table 2, our experiments used 64-bit operating systems in our client node and server node. The choices of operating system were limited to FreeBSD and Ubuntu because these were the only ones that supported the features we needed. FreeBSD 8.2 is currently the only [known] operating system to support the CMT mode of SCTP. The Linux MPTCP kernel is customized for Ubuntu 11.x only (but it may run in any Ubuntu variant that also supports the 3.0.0-14 kernel, such as Linux Mint 12).

Table : Operating systems used

|  |  |  |
| --- | --- | --- |
| **Node** | **Test** | **Operating System** |
| Delay Nodes | *all* | FreeBSD 4.10 (32-bit) |
| Client/Server Nodes | SCTP, TCP | FreeBSD 8.2 (64-bit) |
| MPTCP | Ubuntu Server 11.04 (64-bit)  w/Linux MPTCP 3.0.0-14 |

#### Network Throughput Tuning

The default network settings in the kernel were adjusted as necessary to maximize the throughput test results. In particular, the buffer sizes of the receive and transmit-windows were increased from a conservative 200KB to 16MB. See Appendix for the specific commands.

### Hardware Specifications

All nodes in our experiments (including the delay nodes) have the following hardware specifications:

* Dell Poweredge 2850
* 3.0 GHz 64-bit Xeon processors, 800Mhz FSB
* 2GB 400Mhz DDR2 RAM
* Multiple PCI-X 64/133 and 64/100 buses
* Six 10/100/1000 Intel NICs spread across the buses (one NIC is the control net)
* 2 x 146GB 10,000 RPM SCSI disks

### Software Test Tools

To test the network throughput between the client and server nodes, we used *NetPerfMeter*[[7](#Uni113)]. This tool is similar to *iperf* in that it connects to a server instance and transfers (or exchanges) data as fast as possible. During the tests, *NetPerfMeter* prints the current bandwidth in the TX and RX directions; data size sent and received; and CPU utilization. In addition, this tool is capable of creating multiple simultaneous flows of UDP, TCP, DCCP , and/or SCTP.

#### NetPerfMeter Settings

Our experiments used one or more concurrent flows, all of which were configured in *NetPerfMeter* as shown in Table 3. Each flow was tested as half-duplex (unidirectional from traffic-generating client to listening server) and was always set to the same protocol. That is, all flows were either set in *NetPerfMeter* to TCP (which internally used MPTCP in the transport layer for the MPTCP test) or SCTP.

Table : Settings used in *NetPerfMeter* for each traffic flow

|  |  |
| --- | --- |
| **Outgoing packet size** | 1452 bytes (based on 1500-byte MTU) |
| **Outgoing rate** | As many packets as possible per second |
| **Incoming packet size** | 0 bytes (off) |
| **Incoming rate** | 0 (off) |
| **Runtime** | 120 seconds |
| **Receive buffer** | 7000000 |
| **Send buffer** | 14000000 (larger sizes not allowed in FreeBSD) |
| **Packet-delivery order (SCTP only)** | Disabled (100% unordered) |

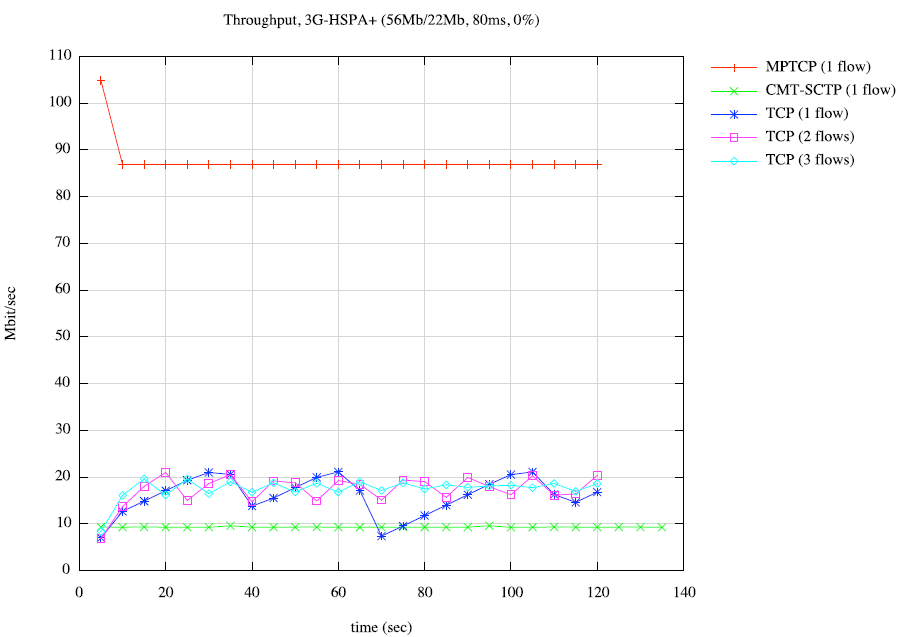
## Methodology

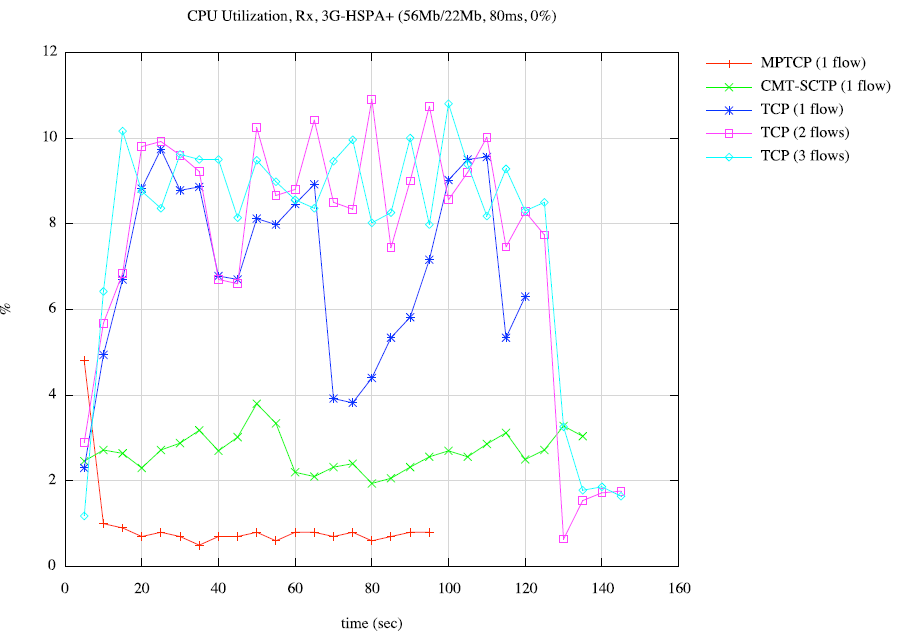
For each technology under evaluation (i.e., 802.11g, 3G, 4G, and Ethernet), we tested the throughput of SCTP and MPTCP in *Emulab*. Each experiment consisted of starting the client and server nodes, loading the appropriate kernel settings (and operating system), and then running *NetPerfMeter* on those nodes for 120 seconds. The statistics print-outs from *NetPerfMeter* were then downloaded to our local machines for post-processing. The client and server nodes were rebooted between experiments (i.e., upon switching link types). Four experiments, with 4 nodes each (16 total), ran in parallel on separate hosts and networks within *Emulab.*

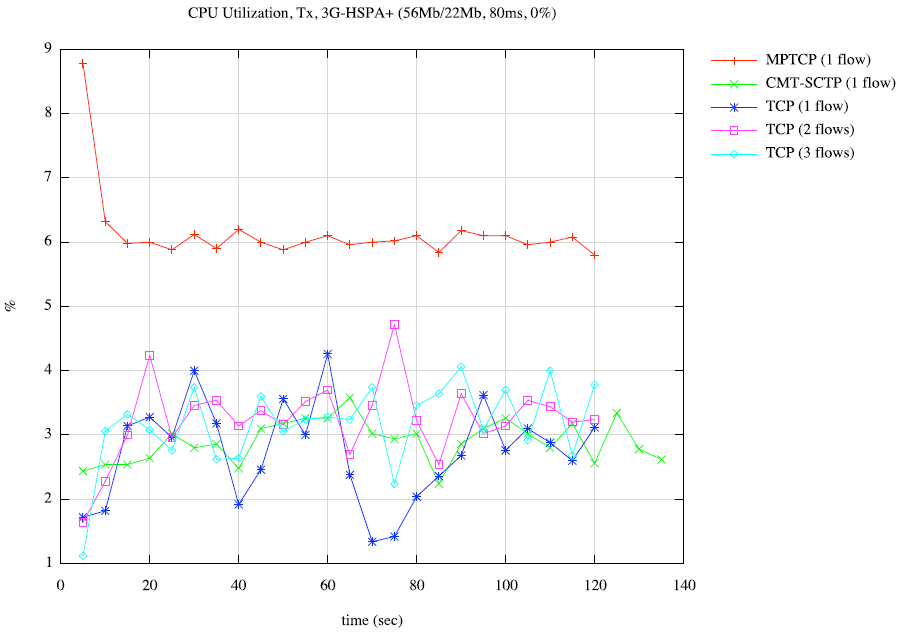
# Results

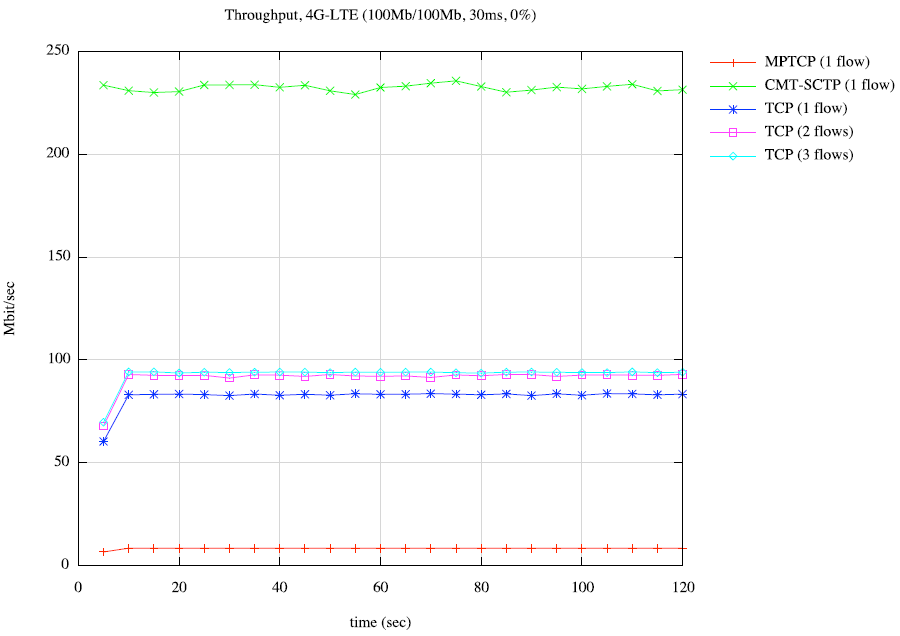
The plots shown below indicate SCTP has higher throughput than MPTCP in 4G, Wi-Fi, and Ethernet. For some reason, SCTP performed poorly in 3G. In addition, we discovered that SCTP showed alarmingly high CPU utilization in all link technologies except for 3G.

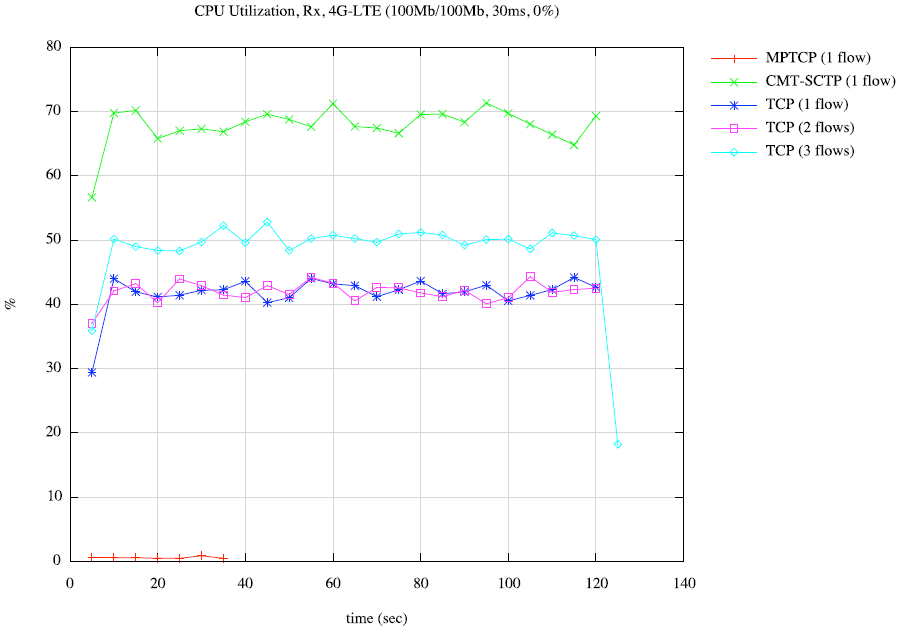
The parallel TCP flows were used as a lower-limit reference, and the Ethernet flows were used as an upper limit. That is, we expected 1-flow SCTP and 1-flow MPTCP to each have higher throughput than TCP but not exceed Ethernet performance, and this proved to be the case.

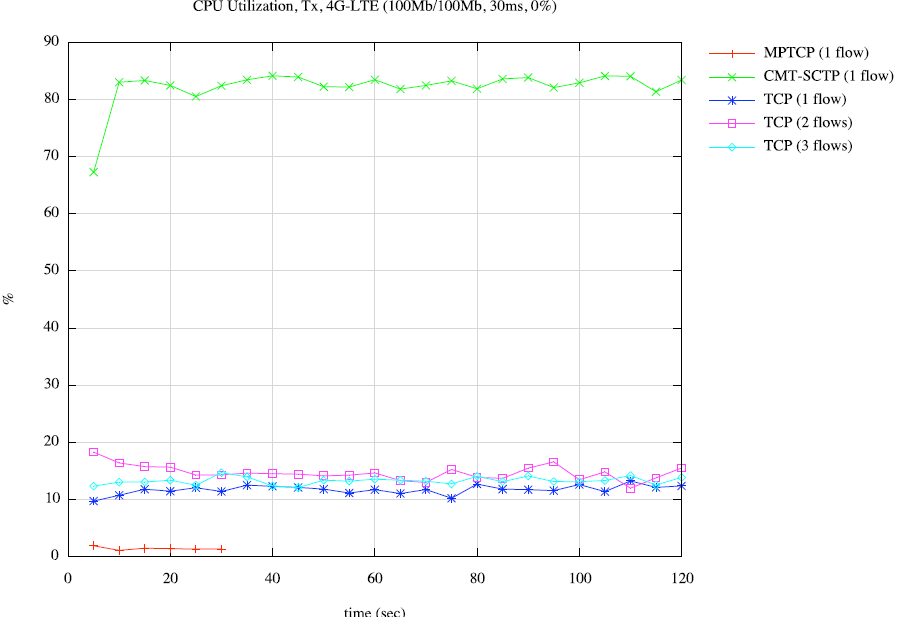


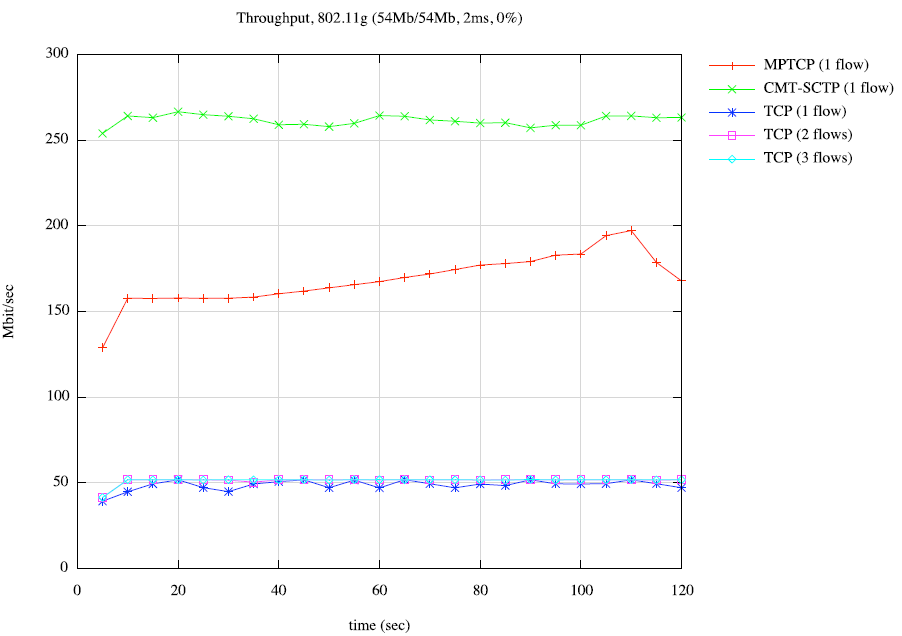


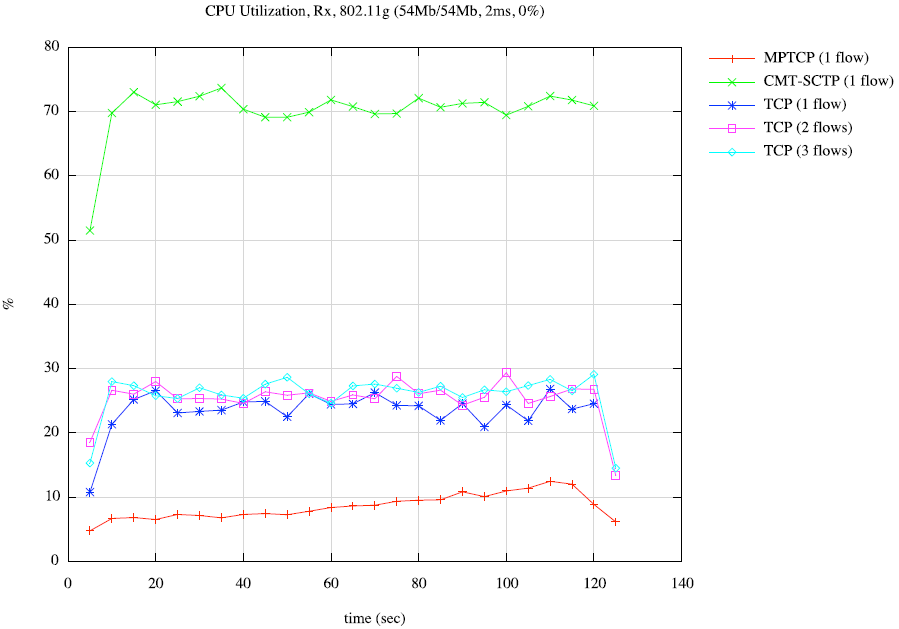


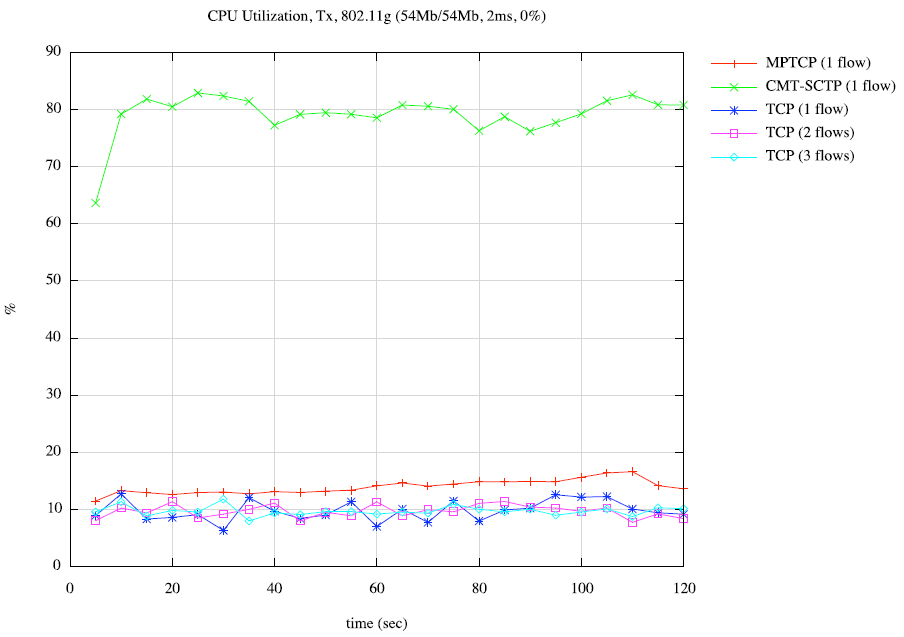


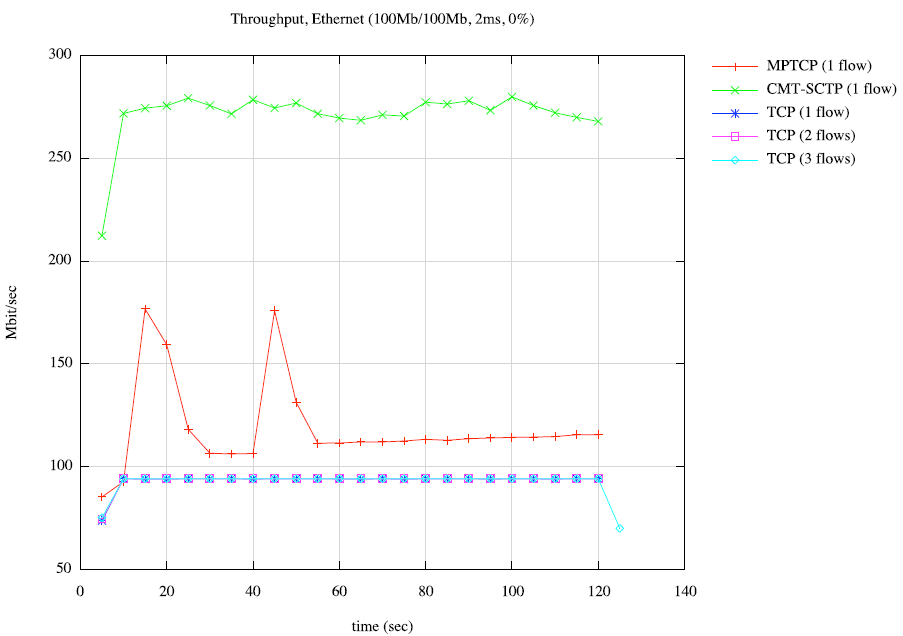


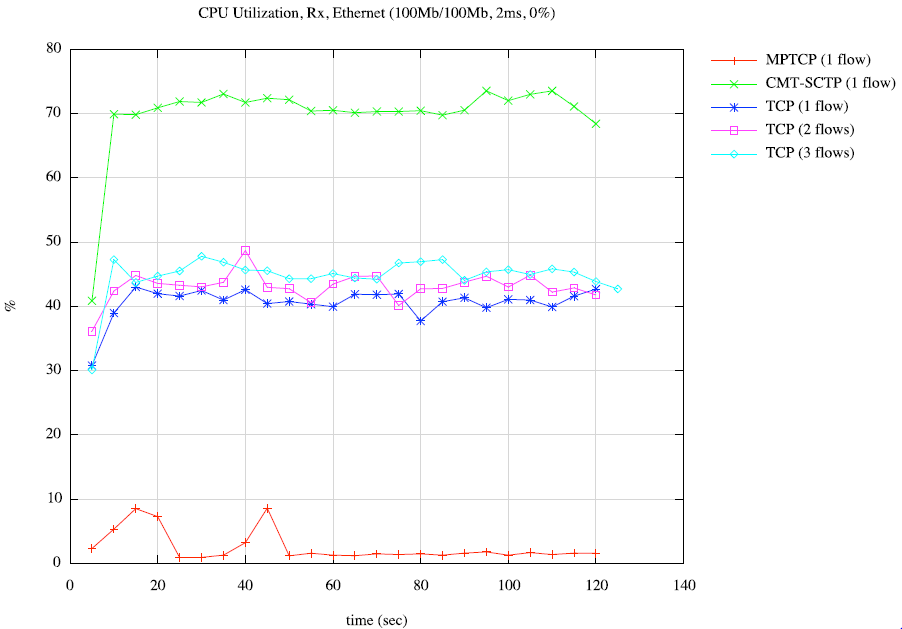


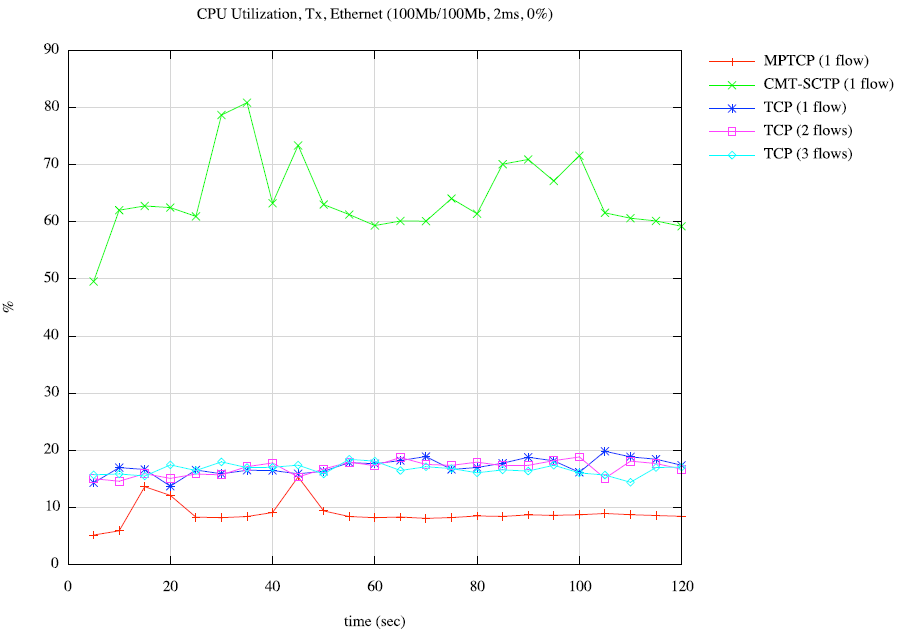












# Conclusion

This paper investigated and evaluated the potential usage of newer acceleration type protocols, to replace legacy protocols such as TCP. Based on our testing we have seen that in most cases, both MPTCP and SCTP outperform even multiple stream TCP implementations. While both have strengths, MPTCP is most attractive as it is backward compatible with existing TCP application. In addition, as MPTCP is an extension to the network stack itself, it tends to be faster and use less CPU cycles that SCTP. On the downside, as MPTCP uses non-common options flags available in the TCP header, it may not traverse all middle boxes and therefore may require internet infrastructure changes for mainstream implementation. This is where SCTP has an advantage, as it does not require any infrastructure changes, and utilizes the best parts of the TCP protocol itself. This includes a more secure security implementation along with not having to deal with head-of-line blocking due to streaming of byte data. Given SCTP requires changes to legacy application code, which may outweigh its benefit for mainstream deployment.

# References

x

|  |  |
| --- | --- |
| [1] | University of Southern California. (1981, Sep.) Transmission Control Protocol. [Online]. <http://www.ietf.org/rfc/rfc793.txt> |
| [2] | M Scharf. (2011, Nov.) MPTCP Application Interface Considerations, draft-ietf-mptcp-api-03. [Online]. <http://tools.ietf.org/html/draft-ietf-mptcp-api-03> |
| [3] | R Stewart et al. (2000, Oct.) Stream Control Transmission Protocol. [Online]. <http://www.ietf.org/rfc/rfc2960.txt> |
| [4] | University of Utah. (2011, Dec.) Emulab. [Online]. <http://www.emulab.net> |
| [5] | Luigi Rizzo. (2011, Dec.) Dummynet. [Online]. <http://info.iet.unipi.it/~luigi/dummynet/> |
| [6] | Universite Catholique de Louvain. (2011, Dec.) MultiPath TCP Linux Kernel Implementation. [Online]. <http://mptcp.info.ucl.ac.be/> |
| [7] | University of Duisburg-Essen. (2011, Aug.) NetPerfMeter. [Online]. <http://www.iem.uni-due.de/~dreibh/netperfmeter/> |
| [8] | University of Illinois. (2011, Dec.) Iperf. [Online]. <http://iperf.sourceforge.net/> |
| [9] | E. Kohler, M. Handley, and S. Floyd. (2006, Mar.) Datagram Congestion Control Protocol (DCCP). [Online]. <http://www.ietf.org/rfc/rfc4340.txt> |
| [10] | J. Iyengar, K. Shah, and P. Amer, "Concurrent Multipath Transfer using SCTP Multihoming," 2004. |
| [11] | M. Becke, T. Dreibholz, and others, "Load Sharing for the Stream Control Transmission Protocol (SCTP)," *draft-tuexen-tsvwg-sctp-multipath-00, Internet Draft, IETF*, July 2010. |

x

# Appendix

## Shell Scripts

**HOW TO RUN TESTS.txt**

##################################################################

# These instructions describe how to run MPTCP and SCTP experiments in

# Emulab. They're the same steps we used to generate the plots in our report.

# This was tested on Mac OSX Lion 10.7.2 but should work on most \*nix systems.

#

# Note that running all test one after another can take a long time, but beauty

# of Emulab is that you can run multiple experiments at once. Just be sure

# to keep track of all running experiments.

##################################################################

#######################################

# SETUP

#######################################

1. Load an experiment in Emulab (www.emulab.net).

a. Pick an NS file from ${project}/scripts/ns to be loaded for the experiment based on the test. See specific test instructions below.

b. One way to load the NS file into Emulab: modify an existing experiment. Navigate the following:

i. Login to: www.emulab.net

ii. Click "My Emulab".

iii. Click the name of the experiment to modify.

iv. Click "Modify Experiment".

v. Click "Choose File" button.

vi. Select the NS file to load (from step a).

vii. Click "Modify" button. This takes a few minutes.

c. Swap in the experiment.

i. Go back to the details page for the experiment.

ii. Click "Swap Experiment In", and confirm the prompts.

2. Get the root login info for the two nodes of interest (node0 and node1).

a. Open the details page for your experiment (one way is to navigate "www.emulab.net > Experimentation > Experiment List" and click the EID of the experiment).

b. For node0, click the Node ID to open the details page for the node.

c. Make note of the values for "Node ID" and "root\_password".

d. Repeat b and c for node1.

3. Open a serial line for each of the two nodes of interest (node0 and node1) in two separate terminals.

a. SSH into users.emulab.net (don't SSH into the node machine itself or else the SSH traffic could skew the results, but that might be acceptable if that noise traffic is accounted for).

NOTE: To be able to SSH, you need to generate a password-protected public key (the `ssh-keygen` command) and upload it to Emulab (www.emulab.net > My Emulab > Edit SSH Keys). Once the key is uploaded, you can SSH in from your personal machine using any of these commands:

% ssh ${emulab-user}@users.emulab.net

% ssh ${emulab-user}@node0.${test-name}.damnit.emulab.net

% ssh ${emulab-user}@node1.${test-name}.damnit.emulab.net

b. From the users.emulab.net terminal, enter:

% console -d pcXXX

where pcXXX is the "Node ID" value from step 3c.

This opens a telnet session to the host's serial line. Press ENTER a couple times to show the login prompt. To escape this telnet session, enter CTRL+] and then `quit`. That should bring you back to the users.emulab.net session in step a.

NOTE: The serial line is quirky. It sometimes kills telnet at the first login before even showing the login prompt, in which case you just try again. Also, the UP key in FreeBSD causes TCSH to core-dump, so switch to BASH at login. Finally, vi is un-usable in the serial line for FreeBSD (you'll have to login directly to the node to use vi successfully...just remember to logout before you start any network throughput tests in order to avoid skewing results...an alternative is to modify files locally and then scp it in).

c. For the login username, enter 'root'. For the password, enter the "root\_password" value from step 2c (ok to copy and paste).

d. Switch from root to your local user:

# su - ${emulab-user}

4. From your personal machine, upload all shell scripts and binaries to Emulab:

% scp ${scripts-dir}/\*.sh ${emulab-user}@users.emulab.net:scripts/.

% scp ${bin-dir}/\* ${emulab-user}@users.emulab.net:bin/.

#######################################

# MPTCP SETUP

#######################################

1. Follow the SETUP procedure above, but load one of the Ubuntu NS files.

2. In vanilla Ubuntu 11.04, run "setup\_linux\_mptcp.sh".

3. Run `sudo reboot`. Since you're on the serial line, you'll see the reboot process scroll by. This takes a few minutes, and then you'll see the login prompt.

4. Once logged in, enter `uname -r`. You should see "3.0.0-14-mptcp".

#######################################

# MPTCP TEST

#######################################

1. Follow the MPTCP SETUP procedure above.

2. Run "tune\_ubu.sh" to tune the Ubuntu network settings.

3. Run MPTCP test.

a. From node1, run "server\_mptcp.sh 1".

b. From node0, run "test\_mptcp.sh 1" to test MPTCP with 1 flow. Wait 600 seconds.

c. From your personal machine, download the test results with:

% cd ${results-dir}

% scp ${emulab-user}@users.emulab.net:scripts/results/\* .

4. (OPTIONAL) Plot the results with GnuPlot (assuming installed on host). The .gp file expects certain data files. If they're missing, a warning is printed, but it still shows a plot of other data found.

% cd ${results-dir}

% ${scripts-dir}/plot\_data.sh

#######################################

# SCTP TEST

#######################################

1. Follow the SETUP procedure above, but load one of the FreeBSD NS files.

2. Run "tune\_bsd.sh" to tune the FreeBSD network settings.

3. Run TCP baseline test.

a. From node0, run "server\_tcp.sh 1".

b. From node1, run "test\_tcp.sh 1" to record data for 1-flow TCP. Wait 130 seconds.

c. From your personal machine, download the test results with:

% cd ${results-dir}

% scp ${emulab-user}@users.emulab.net:scripts/results/\* .

d. Repeat a through c with 2 flows, then 3, and 4 flows (i.e., change the first argument to test\_tcp.sh to change the number of concurrent flows).

4. Run SCTP test.

a. From node0, run "server\_sctp.sh 1" to test SCTP with 1 flow.

b. From node1, run "test\_sctp.sh 1" to record data for 1-flow TCP. Wait 130 seconds.

c. From your personal machine, download the test results with:

% cd ${results-dir}

% scp ${emulab-user}@users.emulab.net:scripts/results/\* .

5. (OPTIONAL) Plot the results with GnuPlot (assuming installed on host). The .gp file expects certain data files. If they're missing, a warning is printed, but it still shows a plot of other data found.

% cd ${results-dir}

% ${scripts-dir}/plot\_data.sh

**setup\_linux\_mptcp.sh**

#!/bin/sh

#######################################################################

# This script installs MPTCP Linux Kernel from

# http://mptcp.info.ucl.ac.be/pmwiki.php?n=Users.AptRepository.

#######################################################################

# Get the key to validate the repository we're about to add

wget -q -O - http://mptcp.info.ucl.ac.be/mptcp.gpg.key | sudo apt-key add -

# Add repository to source list only if not present

grep -q 'deb http://mptcp.info.ucl.ac.be/repos/apt/debian orneic main' /etc/apt/sources.list || echo 'deb http://mptcp.info.ucl.ac.be/repos/apt/debian orneic main' | sudo tee -a /etc/apt/sources.list > /dev/null

# Make sure we have the sources for 11.10 (needed for linux-headers-3.0.0-14)

sudo sed -i 's/natty/oneiric/' /etc/apt/sources.list

sudo apt-get update

# Install the kernel and the SCTP library (for netperfmeter)

sudo apt-get -y install linux-image-3.0.0-14-mptcp libsctp1

# Set the following Grub parameters to 0 so that Linux MPTCP boots

# automatically.

# \* GRUB\_DEFAULT

# \* GRUB\_TIMEOUT

# \* GRUB\_HIDDEN\_TIMEOUT

echo Updating boot settings to auto-load Linux MPTCP...

sudo sed -i 's/\(GRUB\_DEFAULT=\|GRUB\_TIMEOUT=\|GRUB\_HIDDEN\_TIMEOUT=\).\*/\10/' /etc/default/grub

sudo update-grub

echo "OK. Enter 'sudo reboot' to load Linux MPTCP."

**tune\_ubu.sh**

#!/bin/sh

#####################################################################

# This script sets the kernel network settings for maximum throughput.

# http://www.techrepublic.com/blog/opensource/tuning-the-linux-kernel-for-more-aggressive-network-throughput/62

#####################################################################

sudo sysctl -w net.ipv4.tcp\_window\_scaling=1

sudo sysctl -w net.ipv4.tcp\_syncookies=1

sudo sysctl -w net.core.rmem\_max=16777216

sudo sysctl -w net.core.wmem\_max=16777216

# setting = 'min init max'

#sudo sysctl -w net.ipv4.tcp\_rmem="4096 87380 16777216"

#sudo sysctl -w net.ipv4.tcp\_wmem="4096 65536 16777216"

# setting = max

sudo sysctl -w net.ipv4.tcp\_rmem=16777216

sudo sysctl -w net.ipv4.tcp\_wmem=16777216

server\_mptcp.sh

**test\_mptcp.sh**

#!/bin/sh

#####################################################################

# This script creates an output directory, and runs a single

# 120-second MPTCP throughput test with a server host. The test uses

# netperfmeter 1.1.9 on Ubuntu.

#

# See manpage: http://dev.man-online.org/man8/netperfmeter/

#####################################################################

if [ ! $1 ]; then

echo "error: $0 {flowcount}"

exit 1

fi

# Only run if MPTCP is on. We can't enable it on the fly with the sysctl cmd.

if ! sysctl -n net.mptcp.mptcp\_enabled > /dev/null 2>&1; then

# key doesn't exist (not Linux MPTCP)...not ok

echo "error: MPTCP not detected"

exit 1

fi

if [ $(sysctl -n net.mptcp.mptcp\_enabled) -eq 0 ]; then

echo "error: MPTCP is disabled (sysctl.net.mptcp.mptcp\_enabled=0)"

exit 1

fi

# Install libsctp if not found

if [ ! "$(find /usr/lib -name 'libsctp\*')" ]; then

sudo apt-get -y -q install libsctp1

fi

outdir=results/$(basename ${0%.\*})\_$(uname -n | awk 'BEGIN { FS = "." } ; {print $2}')

npm=../bin/netperfmeter-ubu64

server=node1:9000

# Duration of throughput tests (in seconds)

runtime=120

# Outgoing rate. Set to 'const0' to send as much as possible.

# Packet size = 1452 (MTU 1500 - IP/UDP header)

outrate=const0

outlen=const1452

# No incoming flow (half-duplex). set both rate and size

# to 'const0' to disable

inrate=const0

inlen=const0

# Tx buffer should be set as high as allowed by kernel.

# Rx buffer should be half the Tx buffer.

rcvbuf=rcvbuf=7000000

sndbuf=sndbuf=14000000

flowspec=$outrate:$outlen:$inrate:$inlen:$rcvbuf:$sndbuf

# Create output dir if necessary

[ ! -d $outdir ] && mkdir -p $outdir 2>/dev/null

echo "\*\*\* Running MPTCP test (flows=$1) for $runtime seconds"

pre=mptcp$1

$npm $server $(yes " -tcp $flowspec" | head -n$1) -runtime=$runtime > ${outdir}/${pre}.tx.txt

**tune\_bsd.sh**

#!/bin/sh

######################################################

# This script sets the kernel network settings for maximum throughput.

# http://fasterdata.es.net/fasterdata/host-tuning/freebsd/

######################################################

# set to at least 16MB for 10GE hosts

sudo sysctl kern.ipc.maxsockbuf=16777216

# set autotuning maximum to at least 16MB too

sudo sysctl net.inet.tcp.sendbuf\_max=16777216

sudo sysctl net.inet.tcp.recvbuf\_max=16777216

# enable send/recv autotuning

sudo sysctl net.inet.tcp.sendbuf\_auto=1

sudo sysctl net.inet.tcp.recvbuf\_auto=1

# increase autotuning step size

sudo sysctl net.inet.tcp.sendbuf\_inc=16384

sudo sysctl net.inet.tcp.recvbuf\_inc=524288

# turn off inflight limitting

sudo sysctl net.inet.tcp.inflight.enable=0

# set this on test/measurement hosts

sudo sysctl net.inet.tcp.hostcache.expire=1

**server\_sctp.sh**

#!/bin/sh

######################################################

# This script starts a Netperfmeter server on port 9000 and

# redirects stdout to a file in the results directory. The server

# automatically stops after $runtime seconds.

#######################################################

if [ ! $1 ]; then

echo "Usage: $0 {flowcount}"

exit 1

fi

# Duration of throughput tests (in seconds) plus a few seconds

# of buffer time (to allow starting client on other host). This

# obviously requires coordination when running the two hosts.

runtime=150

npm=../bin/netperfmeter-bsd64

port=9000

outdir=results/$(basename ${0%.\*})\_$(uname -n | awk 'BEGIN { FS = "." } ; {print $2}')

pre=sctp$1

# Create output dir if necessary

[ ! -d $outdir ] && mkdir -p $outdir 2>/dev/null

echo "\*\*\* Starting server for SCTP test (flows=$1) for $runtime seconds"

sudo sysctl net.inet.sctp.cmt\_on\_off=1

${npm} ${port} -runtime=${runtime} > ${outdir}/${pre}.rx.txt

**test\_sctp.sh**

#!/bin/sh

#######################################################

# This script creates an output directory, and runs a single

# 120-second SCTP throughput test with a server host. The test uses

# netperfmeter 1.1.9 on FreeBSD.

#

# See manpage: http://dev.man-online.org/man8/netperfmeter/

######################################################

if [ ! $1 ]; then

echo "Usage: $0 {numflows}"

exit 1

fi

# Enable CMT-SCTP for the next benchmarks

sudo sysctl net.inet.sctp.cmt\_on\_off=1

outdir=results/$(basename ${0%.\*})\_$(uname -n | awk 'BEGIN { FS = "." } ; {print $2}')

npm=../bin/netperfmeter-bsd64

server=node1:9000

# Duration of throughput tests (in seconds)

runtime=120

# Outgoing rate. Set to 'const0' to send as much as possible.

# Packet size = 1452 bytes (MTU of 1500 - IP/UDP header)

outrate=const0

outlen=const1452

# No incoming flow (half-duplex). set both rate and size

# to 'const0' to disable

inrate=const0

inlen=const0

# CMT mode

# normal = Normal (independent paths)

# cmtrpv1 = Resource pooled (v1)

# cmtrpv2 = Resource pooled (v2)

# like-mptcp = Like MPTCP

# off = primary path (regular SCTP)

cmtmode=cmt=normal

# Tx buffer should be set as high as allowed by kernel.

# Rx buffer should be half the Tx buffer.

rcvbuf=rcvbuf=7000000

sndbuf=sndbuf=14000000

# Set fraction of traffic to be unordered (0 <= x <= 1.0)

# Setting to 1 disables packet ordering, reducing overhead

# and delay.

ord=unordered=1

flowspec=$outrate:$outlen:$inrate:$inlen:$cmtmode:$rcvbuf:$sndbuf:$ord

# Create output directory if necessary

[ ! -d $outdir ] && mkdir -p $outdir 2>/dev/null

echo "\*\*\* Running SCTP test (flows=$1) for $runtime seconds"

pre=sctp$1

$npm $server $(yes " -sctp $flowspec" | head -n$1) -runtime=$runtime > ${outdir}/${pre}.tx.txt

**plot\_data.sh**

#!/bin/bash

#####################################################################

# This script plots the throughput of MPTCP, CMT-SCTP, and

# TCP-FreeBSD data all in one graph. This runs from the results

# directory, which must have subdirs for each data set

# (i.e., 3g, 4g, wifi, etc.).

#####################################################################

parser=../scripts/parse\_data.sh

for dir in 3g 4g wifi wired

do

echo "Parsing results for ${dir}"

cd $dir && ../${parser} && cd ..

done

function plot {

type=$1

dat=$2

units=$3

title=$4

gnuplot <<EOT

reset

set xlabel "time (sec)"

set ylabel "$units"

set title "$title"

set key reverse Left outside

set grid

set style data linespoints

plot "$type/mptcp1.$dat.dat" using 1:2 title "MPTCP (1 flow)",\

"$type/sctp1.$dat.dat" using 1:2 title "CMT-SCTP (1 flow)",\

"$type/tcp1-bsd.$dat.dat" using 1:2 title "TCP (1 flow)",\

"$type/tcp2-bsd.$dat.dat" using 1:2 title "TCP (2 flows)",\

"$type/tcp3-bsd.$dat.dat" using 1:2 title "TCP (3 flows)"

EOT

}

plot 4g rx.bw Mbit/sec "Throughput, 4G-LTE (100Mb/100Mb, 30ms, 5%)"

plot 3g rx.bw Mbit/sec "Throughput, 3G-HSPA+ (56Mb/22Mb, 80ms, 9%)"

plot wifi rx.bw Mbit/sec "Throughput, 802.11g (54Mb/54Mb, 2ms, 2%)"

plot wired rx.bw Mbit/sec "Throughput, Ethernet (100Mb/100Mb, 2ms, 1%)"

plot 4g rx.cpu % "CPU Utilization, Rx, 4G-LTE (100Mb/100Mb, 30ms, 5%)"

plot 3g rx.cpu % "CPU Utilization, Rx, 3G-HSPA+ (56Mb/22Mb, 80ms, 9%)"

plot wifi rx.cpu % "CPU Utilization, Rx, 802.11g (54Mb/54Mb, 2ms, 2%)"

plot wired rx.cpu % "CPU Utilization, Rx, Ethernet (100Mb/100Mb, 2ms, 1%)"

plot 4g tx.cpu % "CPU Utilization, Tx, 4G-LTE (100Mb/100Mb, 30ms, 5%)"

plot 3g tx.cpu % "CPU Utilization, Tx, 3G-HSPA+ (56Mb/22Mb, 80ms, 9%)"

plot wifi tx.cpu % "CPU Utilization, Tx, 802.11g (54Mb/54Mb, 2ms, 2%)"

plot wired tx.cpu % "CPU Utilization, Tx, Ethernet (100Mb/100Mb, 2ms, 1%)"

## Emulab NS Scripts

**freebsd-3g.ns**

######################################################################

# This script creates a 2-node network with 4 NICs in Emulab. One of

# the NICs is a wired Ethernet connection while the others are some

# mix of wireless ones (WiFi, 3G, and 4G-LTE).

#

# See https://users.emulab.net/trac/emulab/wiki/nscommands for more

# details on the commands.

######################################################################

set ns [new Simulator]

source tb\_compat.tcl

set opt(OS) "FBSD82-64-STD"

# Settings for 3G

#

# http://en.wikipedia.org/wiki/3G

# http://www.ericsson.com/hr/about/events/archieve/2007/mipro\_2007/mipro\_1137.pdf

set opt(BW) "56Mb"

set opt(DELAY) "80ms"

set opt(DN\_BW) "56Mb"

set opt(DN\_DELAY) "40ms"

set opt(DN\_LOSS) "0"

set opt(UP\_BW) "22Mb"

set opt(UP\_DELAY) "40ms"

set opt(UP\_LOSS) "0"

# Nodes

set node0 [$ns node]

set node1 [$ns node]

# Set their OS

tb-set-node-os $node0 $opt(OS)

tb-set-node-os $node1 $opt(OS)

# Set their PC types (PC3000 has 64-bit Xeons)

tb-set-hardware $node0 pc3000

tb-set-hardware $node1 pc3000

# Wired LAN

set lan0 [$ns make-lan "$node0 $node1" 100Mb 0ms]

# Wireless LANs

set lan1 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

set lan2 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

set lan3 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

tb-set-lan-simplex-params $lan1 $node0 $opt(DN\_DELAY) $opt(DN\_BW) $opt(DN\_LOSS) $opt(UP\_DELAY) $opt(UP\_BW) $opt(UP\_LOSS)

tb-set-lan-simplex-params $lan2 $node0 $opt(DN\_DELAY) $opt(DN\_BW) $opt(DN\_LOSS) $opt(UP\_DELAY) $opt(UP\_BW) $opt(UP\_LOSS)

tb-set-lan-simplex-params $lan3 $node0 $opt(DN\_DELAY) $opt(DN\_BW) $opt(DN\_LOSS) $opt(UP\_DELAY) $opt(UP\_BW) $opt(UP\_LOSS)

tb-set-lan-simplex-params $lan1 $node1 $opt(DN\_DELAY) $opt(DN\_BW) $opt(DN\_LOSS) $opt(UP\_DELAY) $opt(UP\_BW) $opt(UP\_LOSS)

tb-set-lan-simplex-params $lan2 $node1 $opt(DN\_DELAY) $opt(DN\_BW) $opt(DN\_LOSS) $opt(UP\_DELAY) $opt(UP\_BW) $opt(UP\_LOSS)

tb-set-lan-simplex-params $lan3 $node1 $opt(DN\_DELAY) $opt(DN\_BW) $opt(DN\_LOSS) $opt(UP\_DELAY) $opt(UP\_BW) $opt(UP\_LOSS)

$ns rtproto Static

$ns run

**freebsd-4g.ns**

######################################################################

# This script creates a 2-node network with 4 NICs in Emulab. One of

# the NICs is a wired Ethernet connection while the others are some

# mix of wireless ones (WiFi, 3G, and 4G-LTE).

#

# See https://users.emulab.net/trac/emulab/wiki/nscommands for more

# details on the commands.

######################################################################

set ns [new Simulator]

source tb\_compat.tcl

set opt(OS) "FBSD82-64-STD"

# Settings for 4G-LTE

#

# http://en.wikipedia.org/wiki/4G

# http://www.pcmag.com/article2/0,2817,2372304,00.asp

set opt(BW) "100Mb"

set opt(DELAY) "30ms"

# Nodes

set node0 [$ns node]

set node1 [$ns node]

# Set their OS

tb-set-node-os $node0 $opt(OS)

tb-set-node-os $node1 $opt(OS)

# Set their PC types (PC3000 has 64-bit Xeons)

tb-set-hardware $node0 pc3000

tb-set-hardware $node1 pc3000

# Wired LAN

set lan0 [$ns make-lan "$node0 $node1" 100Mb 0ms]

# Wireless LANs

set lan1 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

set lan2 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

set lan3 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

$ns rtproto Static

$ns run

**freebsd-wifi.ns**

######################################################################

# This script creates a 2-node network with 4 NICs in Emulab. One of

# the NICs is a wired Ethernet connection while the others are some

# mix of wireless ones (WiFi, 3G, and 4G-LTE).

#

# See https://users.emulab.net/trac/emulab/wiki/nscommands for more

# details on the commands.

######################################################################

set ns [new Simulator]

source tb\_compat.tcl

set opt(OS) "FBSD82-64-STD"

# Settings for 802.11g

set opt(BW) "54Mb"

set opt(DELAY) "2ms"

# Nodes

set node0 [$ns node]

set node1 [$ns node]

# Set their OS

tb-set-node-os $node0 $opt(OS)

tb-set-node-os $node1 $opt(OS)

# Set their PC types (PC3000 has 64-bit Xeons)

tb-set-hardware $node0 pc3000

tb-set-hardware $node1 pc3000

# Wired LAN

set lan0 [$ns make-lan "$node0 $node1" 100Mb 0ms]

# Wireless LANs

set lan1 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

set lan2 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

set lan3 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

$ns rtproto Static

$ns run

**freebsd-wired.ns**

######################################################################

# This script creates a 2-node network with 4 NICs in Emulab. One of

# the NICs is a wired Ethernet connection while the others are some

# mix of wireless ones (WiFi, 3G, and 4G-LTE).

#

# See https://users.emulab.net/trac/emulab/wiki/nscommands for more

# details on the commands.

######################################################################

set ns [new Simulator]

source tb\_compat.tcl

set opt(OS) "FBSD82-64-STD"

# Settings for Ethernet

#

# We use a tiny delay here (smallest possible in Emulab)

# to add router nodes in between the client and server.

# Otherwise, the client and server nodes are connected

# directly (for 0 delay), which is probably unrealistic.

set opt(BW) "100Mb"

set opt(DELAY) "2ms"

# Nodes

set node0 [$ns node]

set node1 [$ns node]

# Set their OS

tb-set-node-os $node0 $opt(OS)

tb-set-node-os $node1 $opt(OS)

# Set their PC types (PC3000 has 64-bit Xeons)

tb-set-hardware $node0 pc3000

tb-set-hardware $node1 pc3000

# Wired LANs

set lan0 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

set lan1 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

set lan2 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

set lan3 [$ns make-lan "$node0 $node1" $opt(BW) $opt(DELAY)]

$ns rtproto Static

$ns run

1. Taken from Wikipedia (http://www.wikipedia.com) and other Internet sources [↑](#footnote-ref-1)