Flow prioritization based on SCTP send buffer occupancy

A PROJECT REPORT

submitted by

Arpan Kapoor **B120555CS**

Deepak Sirone J **B120097CS**

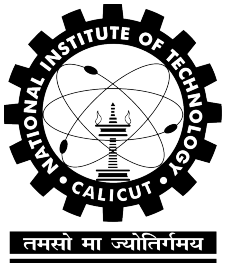
K Prasad Krishnan **B120128CS**

in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology
in
Computer Science and Engineering

under the guidance of

Dr. Vinod Pathari



Department of Computer Science and Engineering
National Institute of Technology Calicut
NIT Campus P.O., Calicut
Kerala, India 673601

May 2016

ACKNOWLEDGEMENTS

We thank Dr. Vinod Pathari, our project guide, for giving us directions and valuable suggestions to complete this project in a professional manner. We also thank Mr. V Anil Kumar, Principal Scientist, CSIR, Bengaluru for providing us with this topic and giving us the basic directions to start off our work. We would also like to thank Dr. K A Abdul Nazeer, Head of Department, CSE and Dr. A P Shashikala, Dean (Academic) for permitting us to work along with CSIR Fourth Paradigm Institute, Bengaluru for this project.

DECLARATION

"I/We hereby declare that this submission is my/our own work and that, to the best of my/our knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text".

Place: Calicut
Date: May 12, 2016

Signature:

Name: Arpan Kapoor
Roll No.: B120555CS

Signature:

Name: Deepak Sirone J
Roll No.: B120097CS

Signature:

Name: K Prasad Krishnan
Roll No.: B120128CS

CERTIFICATE

This is to certify that the project report entitled: "**Flow prioritization based on SCTP send buffer occupancy**" *submitted by* **Arpan Kapoor B120555CS, Deepak Sirone J B120097CS** and **K Prasad Krishnan B120128CS** *to National Institute of Technology Calicut towards partial fulfilment of the requirements for the award of Degree of Bachelor of Technology in Computer Science Engineering is a bonafide record of the work carried out by them under my supervision and guidance.*

Signed by Guide(s) with name(s) and date

Place: Calicut
Date: May 12, 2016

Signature of Head of the Department

Office Seal

Abstract

Network flows need to be prioritized dynamically in order to ensure fair sharing of bandwidth. Bandwidth estimation based on send buffer occupancy, i.e. the amount of backlogged data present in the sender's buffer has been proposed for TCP. This project proposes to advertise the same in SCTP and investigate whether this parameter is useful in prioritizing network flows (at routers) and hence ensure fair sharing of the bandwidth among all flows.

Contents

1 Introduction **1**
1.1 Problem Statement 1
1.2 Literature Survey 1

2 Design **2**
2.1 Prerequisite terms 2
2.2 Send buffer value 3
2.3 Previously Proposed Modification 3
2.4 Currently Proposed Modification 4
2.5 Test bed design 4

3 Implementation **6**
3.1 Linux Kernel Modification 6
3.2 Test bed Implementation Details 6

4 Results **8**
4.1 Use Case Description 8
4.2 Test Results 8
4.3 Explanation of Test Results 10

5 Conclusion **11**

Bibliography **12**

List of Figures

2.1 SCTP Packet Format 2
2.2 SCTP Chunk Format 3
2.3 Send buffer structure 3
2.4 Proposed Chunk for send buffer advertisement 4
2.5 Proposed Chunk for send buffer advertisement 4
2.6 Test bed topology 5

3.1 Test bed implementation 7
3.2 Hierarchy of classes 7

4.1 Percentage send buffer variation with 2 flows each having duration of 300 seconds using the Token Bucket Filter qdisc with rate limited to 1mbit/sec 8
4.2 Percentage send buffer variation with 2 flows each having duration of 300 seconds using Stochastic Fair Queuing qdisc with rate limited to 1mbit/sec 9

List of Tables

4.1 Average bandwidth (kbits/sec) with different number of flows, each lasting 300 seconds 9
4.2 Standard deviation in bandwidth (kbits/sec) 9

Chapter 1

Introduction

Stream Control Transport Protocol (SCTP) is a reliable transport protocol designed to transport Public Switched Telephone Network (PSTN) signaling messages over IP networks, but is capable of broader applications. Unlike TCP, SCTP offers sequenced delivery of user messages within multiple unidirectional logical channels called streams. Each SCTP endpoint is represented as a set of destination transport addresses, one of which is the primary address. If the primary address becomes unreachable SCTP chooses another destination transport address to route the messages thereafter. This provides network-level fault tolerance and is called multi-homing. It also employs a security cookie mechanism during association initialization to provide resistance to flooding and masquerade attacks.

Advertising the amount of backlogged data present in the sender's buffer can help network operators evaluate the end-to-end performance of a connection in a better way than that with the existing passive measurements like receive window and congestion window. This information can also be used to infer whether a connection is limited by the network or the application.

1.1 Problem Statement

To propose a scheme to advertise send buffer occupancy information in SCTP, implement it in the Linux kernel and study the network performance using this scheme by prioritizing the network flows in a congested network based on the send buffer occupancy.

1.2 Literature Survey

RFC 3286 [8] provides a high level introduction to the capabilities supported by SCTP, while RFC 4960 [9] describes the complete protocol. Agache and Raiciu [1] propose a scheme to advertise send buffer occupancy in TCP. [4] was used to study the state machine employed in the Linux SCTP implementation. It was also used to understand the SCTP packet flow within the kernel. [5] provided with an overview of the traffic control and routing mechanisms in the Linux kernel, along with the userspace tools available for shaping and controlling the traffic.

Chapter 2

Design

2.1 Prerequisite terms

- **SCTP packet** is the unit of data that is passed on to the lower layer protocol. It is composed of a common header followed by one or more chunks.

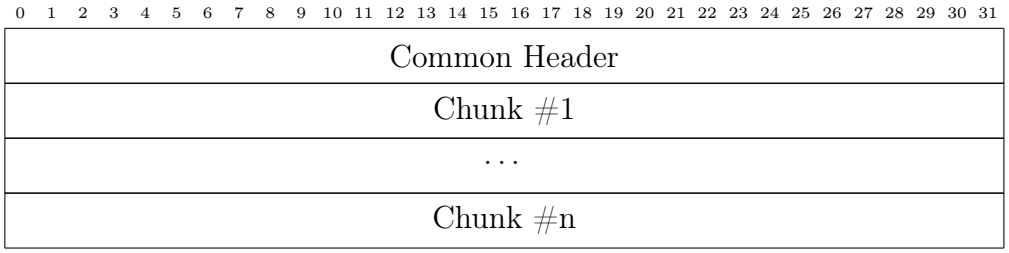


Figure 2.1: SCTP Packet Format

- **SCTP Chunk** is a unit of information within an SCTP packet, containing either control information or user data. It consists of the following fields:

  - **Chunk Type** identifies the type of information contained in the Chunk Value field. The chunk type values 16–62, 64–126, 129, 131, 133–190, 194–254 are currently unassigned[3].

    The highest-order 2 bits of this 8-bit field specify the action that must be taken if the processing endpoint does not recognize the Chunk Type.

    `00` - Stop processing this SCTP packet and discard it.

    `01` - Stop processing this SCTP packet and discard it and report the unrecognized chunk in an 'Unrecognized Chunk Type'.

    `10` - Skip this chunk and continue processing.

    `11` - Skip this chunk and continue processing, but report in an ERROR chunk using the 'Unrecognized Chunk Type' cause of error.

  - **Chunk Flags** usage depends on the Chunk Type.
  - **Chunk Length** represents the size of the chunk in bytes, including the Chunk Type, Chunk Flags, Chunk Length, and Chunk Value fields.
  - **Chunk Value** contains the actual information to be transferred in the chunk.

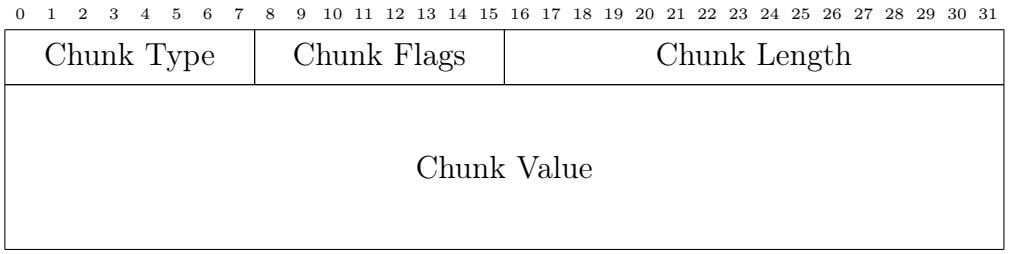


Figure 2.2: SCTP Chunk Format

- Network Flow - A sequence of packets in a transport connection.
- Qdisc (Queueing Discipline) - An algorithm that manages the queue of a device, either egress or ingress. It can be either be classless or classful. Examples of classless qdiscs include FIFO, Token Bucket Filter (TBF) and Stochastic Fair Queuing (SFQ). Hierarchy Token Bucket (HTB) is one of the classful qdiscs.
- Stochastic Fair Queueing (SFQ) - A classless queueing discipline that classifies the packets based on flows and then assigns to a hash bucket. Each flow is identified by its source address, destination address and the source port. Ideally each bucket should represent a unique flow, though multiple flows may get hashed to the same bucket especially when there are large number of flows.
- Classes - Some qdiscs can contain classes, which contain further qdiscs - traffic may then be enqueued in any of the inner qdiscs, which are within the classes.

2.2 Send buffer value

The kernel send buffer can be divided into 2 distinct parts:

- unacknowledged in-flight segments.
- segments waiting to be sent (backlog).

We propose to advertise the number of bytes in this backlog in the Chunk Value field of the proposed chunk.

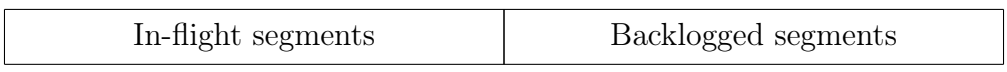


Figure 2.3: Send buffer structure

2.3 Previously Proposed Modification

For advertising the send buffer occupancy, we propose to add a new Chunk Type, with a 32-bit Chunk Value field containing the amount of backlogged data in the send buffer.

To ensure that hosts running an unmodified SCTP stack skip this chunk without returning an ERROR chunk, the highest-order 2 bits of the Chunk Type of this chunk

should be 10 (as explained in section 2.1). This limits the choice of the Chunk Type value between 128 and 190.

This chunk is sent at a specified interval. The total size of this chunk is 8 bytes, which is 0.53% of a typical 1500 byte packet.

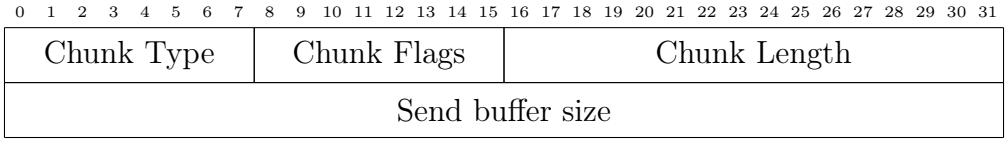


Figure 2.4: Proposed Chunk for send buffer advertisement

2.4 Currently Proposed Modification

For advertising the send buffer occupancy, we propose to add a new Chunk Type, with the Chunk Flags field containing the percentage occupancy of the send buffer.

To ensure that hosts running an unmodified SCTP stack skip this chunk without returning an ERROR chunk, the highest-order 2 bits of the Chunk Type of this chunk should be 10 (as explained in section 2.1). This limits the choice of the Chunk Type value between 128 and 190.

Each SCTP packet (and in turn every IP packet) contains this chunk as the first chunk. Since SCTP has path MTU detection, each SCTP packet fits into exactly 1 IP packet. The total size of this chunk is 4 bytes, which is 0.26% of a typical 1500 byte packet.

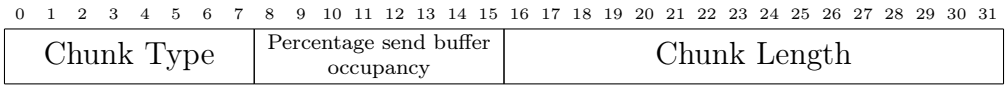


Figure 2.5: Proposed Chunk for send buffer advertisement

2.5 Test bed design

A dumbbell shaped network topology was created with two routers in the center, and multiple hosts connected to one end of each router via a switch. This ensures that we have a bottleneck link in all flows between end hosts on either side.

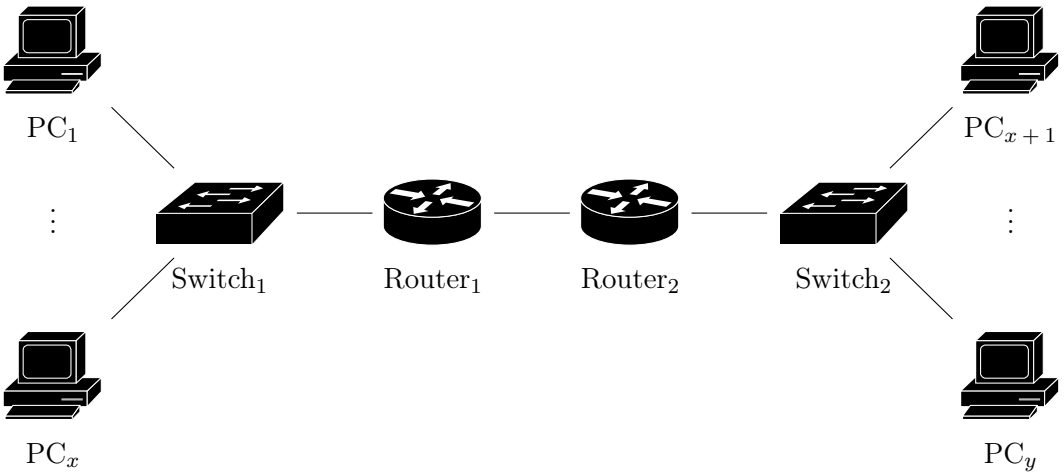


Figure 2.6: Test bed topology

Chapter 3

Implementation

3.1 Linux Kernel Modification

- A patch implementing the SCTP send buffer advertisement was created for Linux kernel `v4.6-rc4`.
- The send buffer advertisement chunk type value was set to 150.
- To modify the frequency at which send buffer advertisement chunks are sent, a `sysctl` interface was created. The default value was set to 5 seconds.
- A kernel timer was added corresponding to each SCTP association (within the `struct sctp_association`).
- A state table was created for this chunk, specifying the states in which the send buffer advertisement chunk should be generated and sent.

3.2 Test bed Implementation Details

Two Raspberry Pis running Arch Linux were configured as routers by enabling IP forwarding. Laptops running the modified Linux kernel were used as end hosts.

The hosts in the test bed were allocated IP addresses as in figure 3.1. From the figure, we observe that RPi_1 would not be able to route packets with destination address in the `192.168.50.0/24` subnet, and similarly RPi_2 wouldn't be able to route packets directed to `192.168.150.0/24` subnet. To enable end-to-end communication, appropriate static routes were added to both the RPis.

The `tc` utility was used in the both the RPis to shape the network flows in the bottleneck link. Larger bandwidth was provided to flows with higher values of the percentage send buffer occupancy.

The Hierarchy Token Bucket classful queuing discipline was used to classify the traffic into 11 classes, one of which is the default class used for non-SCTP packets. The remaining 10 classes correspond to the send buffer occupancy ranges of size 10.

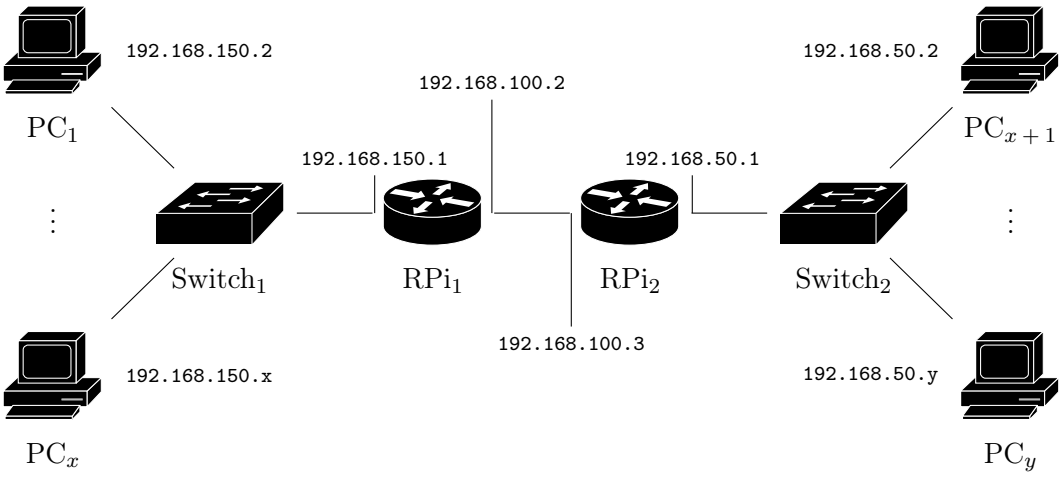


Figure 3.1: Test bed implementation

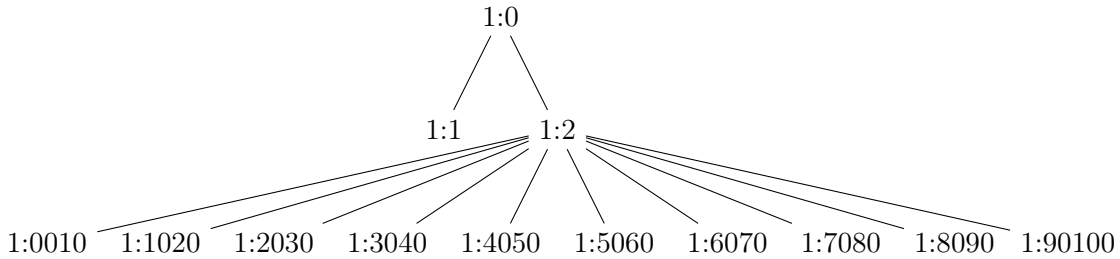


Figure 3.2: Hierarchy of classes

Chapter 4

Results

4.1 Use Case Description

The test bed was used to create a scenario in which there are multiple senders and multiple receivers. Multiple flows were created and the bottleneck between the two Raspberry Pis were fully utilized. The aim was to classify the flows according to the bandwidth requirements. The TBF qdisc did not ensure fair sharing of bandwidth. The SFQ qdisc gives very good performance but degrades for high number of flows [5]. We expect our method of classification can yield better results in reducing average flow completion time in the above case.

4.2 Test Results

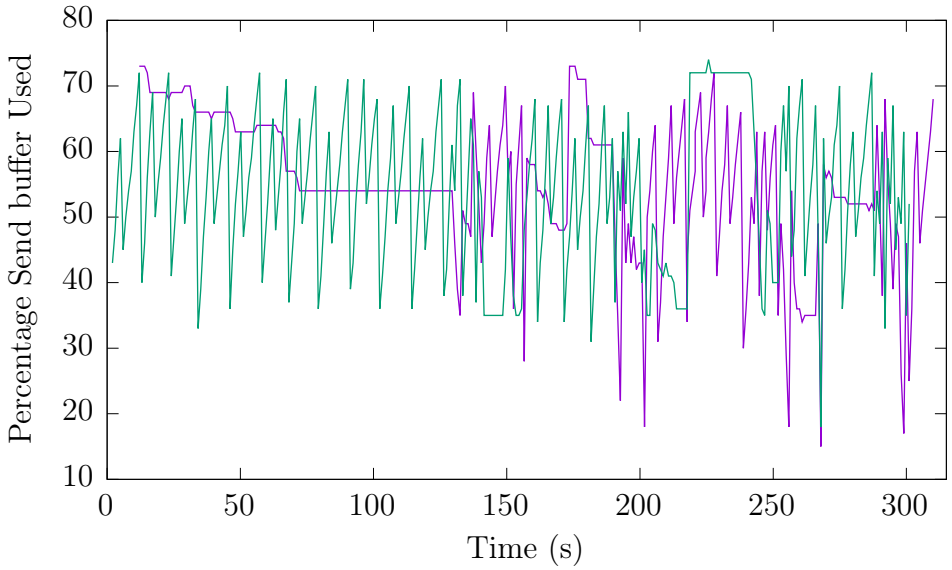


Figure 4.1: Percentage send buffer variation with 2 flows each having duration of 300 seconds using the Token Bucket Filter qdisc with rate limited to 1mbit/sec

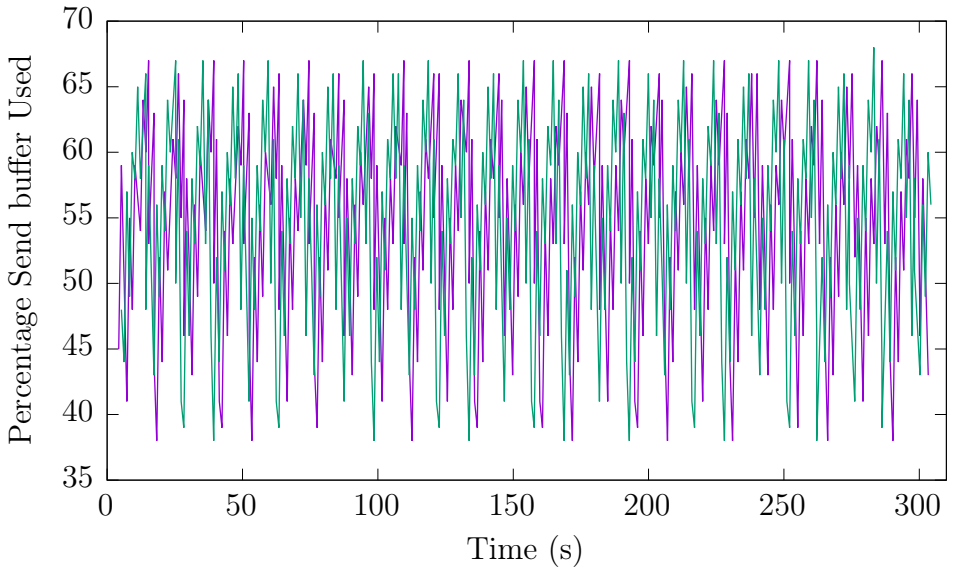


Figure 4.2: Percentage send buffer variation with 2 flows each having duration of 300 seconds using Stochastic Fair Queuing qdisc with rate limited to 1mbit/sec

Number of flows	Send buffer based hierarchy	SFQ
20	2909.75	3651.0
200	296.53	365.65
1000	64.195	76.0317
2000	31.85	52.028

Table 4.1: Average bandwidth (kbits/sec) with different number of flows, each lasting 300 seconds

Number of flows	Send buffer based hierarchy	SFQ
20	507.807	671.027
200	63.959	124.025
1000	29.193	107.742
2000	50.55	288.567

Table 4.2: Standard deviation in bandwidth (kbits/sec)

4.3 Explanation of Test Results

In figure 4.1, one of the flows uses majority of the bandwidth for a specific period of time during which the percentage send buffer occupancy of the other host remains almost constant. On the whole, one of the end hosts utilized four times the bandwidth utilized by the other end host.

In figure 4.2, the bandwidth was shared equally between both the flows. The percentage send buffer occupancy of both the hosts fluctuated periodically during the test period.

The bandwidth observed using our implementation (table 4.1) was lesser than the standard SFQ. This reduction can be mainly attributed to the overhead imposed by the computation involved in classifying the packets based on send buffer occupancy percentages. The additional overhead of 4 bytes per SCTP packet for the send buffer chunk also affects the usable bandwidth.

As the number of flows increase, the probability of two flows hashing to the same bucket also increases. This in turn increases the queuing delay for each of the flows. In a standard SFQ implementation, there is no explicit indicator for flows starved of bandwidth. Our implementation ensures that flows starved of bandwidth are assigned a high priority leaf in the classification scheme. Further each flow classified into a particular leaf is still fairly treated owing to the SFQ qdisc in the leaf. This causes our implementation to have a lesser variance in the bandwidth compared to the standard SFQ implementation.

Chapter 5

Conclusion

Send buffer advertisement was implemented in Linux kernel version 4.6. This information was used to classify packets to implement fair sharing of bandwidth. Our implementation was based on a modified version of the Stochastic Fairness Queuing(SFQ), qdisc in the linux kernel.

Our implementation gives a better standard deviation of bandwidth as compared to the standard SFQ implementation with multiple flows. However, the computation involved in our implementation degrades the average bandwidth when compared to the standard SFQ implementation.

We expect that a hardware implementation of the classifier(like an Application Specific Integrated Circuit) can significantly decrease the computation time required and hence obtaining a higher average bandwidth.

All code and test data used/produced in this project is available at [7].

Bibliography

[1] A. Agache and C. Raiciu. *TCP Sendbuffer Advertising*. Internet-Draft draft-agache-tcpm-sndbufadv-00.txt. IETF Secretariat, July 2015.

[2] Alexandru Agache and Costin Raiciu. "Oh Flow, Are Thou Happy? TCP Sendbuffer Advertising for Make Benefit of Clouds and Tenants". In: *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*. Santa Clara, CA: USENIX Association, July 2015. URL: `https://www.usenix.org/conference/hotcloud15/workshop-program/presentation/agache`.

[3] Internet Assigned Numbers Authority. *Stream Control Transmission Protocol (SCTP) Parameters*. 2015. URL: `https://www.iana.org/assignments/sctp-parameters/sctp-parameters.xhtml#sctp-parameters-1` (visited on 04/20/2016).

[4] Karthik Budigere. "Linux Implementation Study of Stream Control Transmission Protocol". In: *Proceedings of Seminar on Network Protocols in Operating Systems*, p. 22.

[5] Bert Hubert. *Linux Advanced Routing & Traffic Control HOWTO*. 2012. URL: `http://lartc.org/lartc.html` (visited on 04/20/2016).

[6] M. Tim Jones. *Better networking with SCTP*. Feb. 28, 2006. URL: `http://www.ibm.com/developerworks/library/l-sctp/` (visited on 04/20/2016).

[7] Arpan Kapoor, Deepak Sirone J, and K Prasad Krishnan. *btech-project*. 2016. URL: `https://github.com/deepaksirone/sctp_sndbuf_adv`.

[8] L. Ong and J. Yoakum. *An Introduction to the Stream Control Transmission Protocol (SCTP)*. RFC 3286. RFC Editor, May 2002, pp. 1–10. URL: `http://www.rfc-editor.org/rfc/rfc3286.txt`.

[9] R. Stewart. *Stream Control Transmission Protocol*. RFC 4960. RFC Editor, Sept. 2007, pp. 1–152. URL: `http://www.rfc-editor.org/rfc/rfc4960.txt`.

[10] R. Stewart et al. *Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)*. RFC 6458. RFC Editor, Dec. 2011, pp. 1–115. URL: `http://www.rfc-editor.org/rfc/rfc4960.txt`.

[11] *tc-sfq(8) Linux User's Manual*. Jan. 2012.