Thanh Bui
Downing College
ttb29

Computer Science Tripos - Part II Project

An accelerated, network-assisted TCP fast retransmit

May 17, 2019

Proforma

Name:	**Thanh Bui**
College:	**Downing College**
Project Title:	**An accelerated, network-assisted TCP retransmit**
Examination:	**Computer Science Tripos – Part II, June 2019**
Word Count:	**1587**[1]
Project Originator:	Dr Noa Zilberman
Supervisor:	Dr Noa Zilberman

Original Aims of the Project

This project investigates the usage of a programmable switch to assist the Transmission Control Protocol fast retransmit

Work Completed

All that has been completed appears in this dissertation.

Special Difficulties

Learning how to incorporate encapulated postscript into a LaTeX document on both Ubuntu Linux and OS X.

[1] This word count was computed using `texcount -sum -inc -utf8 -sub=chapter diss.tex` for chapters 1–5.

Declaration

I, Thanh Bui of Downing College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

SIGNED DATE

Contents

List of figures

Chapter 1

Introduction

In this chapter, I provide the motivation for this project and setup the problem I am solving. I also explain some key algorithms involved. Finally, I cover some related work.

1.1 Motivation

Transmission Control Protocol (TCP) is the protocol of choice in many data centers. However, it is very sensitive to losses (by design, as a mean for congestion control), which can degrade the performance within the data centers significantly [1]. Various congestion control, avoidance and recovery mechanisms are thus of high importance in this field to minimise such loss rate. Still, not all TCP losses are born equal. For example, losses happening at the destination host's network interface card (NIC) are not an indication of congestion within the network. It is assumed that fast retransmission of such lost packets, from within the network, can increase the utilization of the network.

In-network computing is an emerging research area in systems and networking, where applications traditionally running on the host are offloaded to the network hardware (e.g. switch, NIC). Examples of applications offloaded in the past include network functions (DNS server [2]), distributed systems functions such as consensus (P4xos [3]), various caching (netCache [4], netChain [5]) and even a game (Tic-Tac-Toe). Key-Value Store (KVS) is also among the popular type of in-network applications.

Therefore, it is particularly interesting, and indeed challenging, to see how network-accelerated KVS concepts can be applied to TCP fast retransmit mechanism in order to improve cross-datacentre performance.

1.2 Project Aims

Fast retransmit is an enhancement to TCP that reduces the time a sender waits before retransmitting a lost segment. A TCP sender normally uses a simple timer to recognize lost segments. If an acknowledgement is not received for a particular segment within a specified time (a function of the estimated round-trip delay time), the sender will assume the segment was lost in the network, and will retransmit the segment.

Duplicate acknowledgement (DUP ACK) is the basis for the fast retransmit mechanism. After receiving a packet (e.g. with sequence number 1), the receiver sends an acknowledgement by adding 1 to the sequence number (i.e. acknowledgement number 2). This indicates to the sender that the receiver received the packet number 1 and it expects packet number 2. Suppose that three subsequent packets are lost. The next packets the receiver sees are packet numbers 5 and 6. After receiving packet number 5, the receiver sends an acknowledgement, but still only for sequence number 2. When the receiver receives packet number 6, it sends yet another acknowledgement value of 2. Duplicate acknowledgement occurs when the sender receives more than one acknowledgement with the same sequence number (2 in our example).

When a sender receives several DUP ACKs, it can be reasonably confident that the segment with the sequence number specified in the DUP ACK was dropped. A sender with fast retransmit will then retransmit this packet immediately without waiting for its timeout.

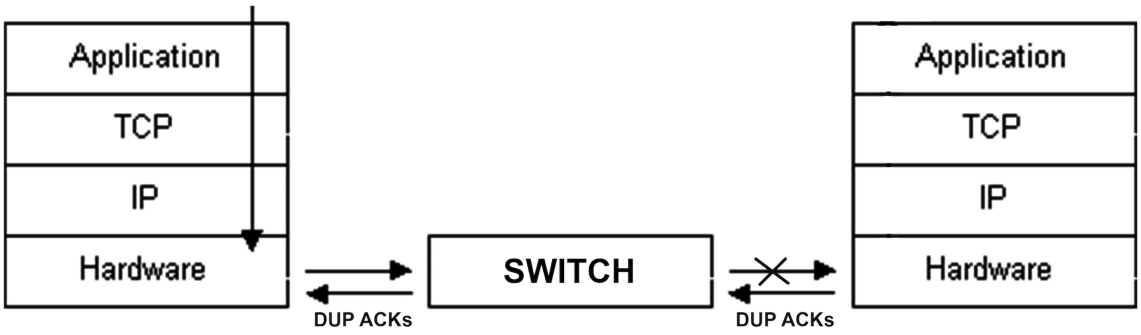

Figure 1.1: The standard convention of TCP handling.

Currently, the DUP ACKs will traverse all the way back to the sender (**Figure 1.1**). The sender receives the DUP ACKs, then retransmits the packet with the next higher sequence number.

This project aims to design and implement a programmable switch that assists the TCP fast retransmit algorithm. The programmable switch will be able to retransmit the packets from within the network, instead of waiting for the DUP ACKs to get back

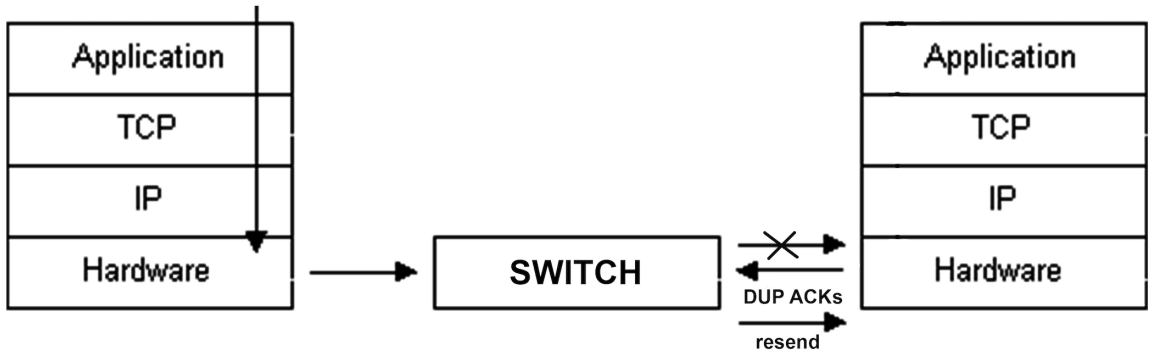

Figure 1.2: The proposed TCP handling.

to the host (**Figure 1.2**), thereby aims to reduce the response time to DUP ACKs and reduce unnecessary changes to the congestion window. The implementation will be based on the KVS concept, where the keys are the flow ID and the packet sequence number, and the value is the payload.

1.3 Related Work

1.3.1 TCP Congestion Control

One of the main aspects of TCP is congestion control, where a number of mechanisms are used to achieve high performance and avoid sending more data than the network is capable of forwarding, that is, to avoid causing network congestion. In particular, TCP uses a *congestion avoidance* algorithm that includes various aspects of an additive increase/multiplicative decrease (AIMD) scheme, with other schemes such as *slow start*, *fast retransmit* and *fast recovery* to achieve congestion avoidance.

The four intertwined algorithms are defined in more detail in RFC 5681[6]. In this project, we are mostly interested in the *fast retransmit* algorithm, which has been explained in the previous section.

1.3.2 Programmable Data Planes

The use of P4, netFPGA, etc.

Chapter 2

Preparation

In this chapter, I discuss the Transmission Control Protocol in more dteail. This is followed by a discussion of sftware deliverables, formal requirements, existing libraries and tools levarage, workflow and starting point.

2.1 Refinement of the project proposal

Empty for now.

2.2 Software Used

2.2.1 Programming Languages

In this project, I used a multitude of languages, including **P4**, **Python**, **Verilog** and **Tcl**.

P4: **P4** programming language [7]. It is a language designed to allow the programming of packet forwarding planes. Besides, unlike general purpose languages such as C or Python, P4 is domain-specific with a number of constructs optimized around network data forwarding, hence is well suited to such a network application.

Python: Python was used extensively in the evaluation because of `scapy` is a Python module that enables the user to send, sniff, dissect and forge network packets. This capability allows me to write unit tests for my program by building customised packets, sending and checking them.

Verilog: Verilog was used to implement certain HDL modules within the P4-NetFPGA platform, in order to add or modify certain functionalities to suit the purpose of my

design. It is the language of choice of the P4-NetFPGA platform.

Tcl:

I also made use of the `make` build automation tool to automate project builds, tests and benchmarks.

2.2.2 P4-NetFPGA Platform

a

2.2.3 The NetFPGA SUME board

2.2.4 Development Environment

Git: I used Git for the I used branches to implement large changes, allowing me to backtrack

2.2.5 High-level Architecture (network level and system level)

Empty for now

2.3 Starting Point

This project uses the knowledge about TCP introduced in the Part IB *Computer Networking* course and the experience in Electronic Computer-aided Design (ECAD) through learning a design-flow for Field Programmable Gate Arrays (FPGAs) from Part IB *ECAD and Architecture Practical Classes*.

During the development of this project, I acquired further knowledge from the materials covered in the following Part II and Part III courses:

- *High Performance Networking* — ;
- *Principle of Communications* — Test;

- *LaTeX and Matlab* — typesetting the project proposal and dissertation.

In terms of , I had little prior experience with P4 Programming Language and the P4-NetFPGA framework. I had some prior experience in Python and Git.

All code was written from scratch, using the . Apart from

2.4 Project Workflow

2.4.1 Setting Up The Development Environment

2.4.2 Getting

2.4.3 Design approach / workflow diagrams

2.4.4 Risk analysis

FFmpeg is a complex piece of software, mostly written in a style of C that sacrifices clarity for performance. A potential risk for the project was the difficulty of proper integration with FFmpeg and hence inability to access or reliably modify motion vectors. Complete failure to do so was unlikely, but it could have consumed a significant amount of development time. As suggested by the spiral development model [19], this high-risk part was scheduled early and some "catch-up" time was allocated in the project timetable in case it caused significant delays.

2.4.5 Backup Plan

Throughout the project development, I made sure to follow good backup procedure by keeping regular local weekly backups of my project using Time Machine for MacOS. This provides recent history through incremental backups. I ensured additional remote storage by backing up with Git, which also provided version control.

2.5 Requirements Analysis

Empty for now

2.5.1 Extensions

Chapter 3

Implementation

I give the overview of the project repository. I move on to explain the

3.1 Repository Overview

```
P4-NetFPGA
├── project
│   ├── simple_sume_switch
│   │   ├── hw
│   │   │   └── hdl
│   │   │       └── nf_datapath.v*
│   │   └── test
│   │       └── sim_switch_default
│   │           └── run.py*
│   ├── src
│   │   ├── tcp_retransmit.p4*
│   │   └── commands.txt*
│   ├── testdata
│   │   ├── gen_testdata.py*
│   │   ├── digest_data.py*
│   │   └── sss_sdnet_tuples.py*
│   └── templates
│       └── externs
│           └── <externs-name>
│               └── hdl
│                   └── <externs-name>_template.v*
└── lib
    └── hw
        ├── contrib
        │   └── cores
        │       └── sss_cache_queues_v1_0_0*
        └── std
            └── cores
                └── output_arbiter_v1_0_0*
```

This project will work mainly with a NetFPGA SUME board [8], using P4 programming language. I will be using the P4-NetFPGA workflow, which provides infrastructure to compile P4 programs to NetFPGA [9]. Apart from that, everything else will be built from scratch.

I have no prior experience with either NetFPGA or P4, but this will be mitigated through self-learning in which I will make use of the online tutorials, Google's resources and the P4 community documentation, as well as the experience of my supervisors.

3.2 Software Implementation

3.2.1 The Parser

3.2.2 The SimpleSumeSwitch

3.2.3 The Deparser

3.3 Hardware Implementation

Chapter 4

Evaluation

Tests

Chapter 5

Conclusion

5.1 Results

5.2 Lessons Learnt

5.3 Future Work

Bibliography

[1] N. Zilberman, M. Grosvenor, D. A. Popescu, N. Manihatty-Bojan, G. Antichi, M. Wójcik, and A. W. Moore, "Where has my time gone?" in *International Conference on Passive and Active Network Measurement*. Springer, 2017, pp. 201–214.

[2] Sapio, Amedeo, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-Network Computation is a Dumb Idea Whose Time Has Come," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 2017, pp. 150–156.

[3] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, H. Weatherspoon, M. Canini, F. Pedone, N. Zilberman, and R. Soulé, "P4xos: Consensus as a Network Service," ser. Tech Report, May 2018. [Online]. Available: http://web.inf.usi.ch/file/pub/105/p4xos.pdf

[4] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "NetCache: Balancing Key-Value Stores with Fast In-Network Caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: ACM, 2017, pp. 121–136. [Online]. Available: http://doi.acm.org/10.1145/3132747.3132764

[5] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica, "NetChain: Scale-Free Sub-RTT Coordination," in *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI '15. Renton, WA, USA: ACM, 2018. [Online]. Available: https://www.usenix.org/system/files/conference/nsdi18/nsdi18-jin.pdf

[6] E. Blanton, V. Paxson, and M. Allman, "TCP Congestion Control," RFC Editor, RFC 5681, September 2009. [Online]. Available: https://tools.ietf.org/html/rfc5681

[7] "The P4 Language Consortium," https://p4.org/.

[8] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as research commodity," *IEEE micro*, vol. 34, no. 5, pp. 32–41, 2014.

[9] "NetFPGA/P4-NetFPGA-public," https://github.com/NetFPGA/P4-NetFPGA-public/wiki.