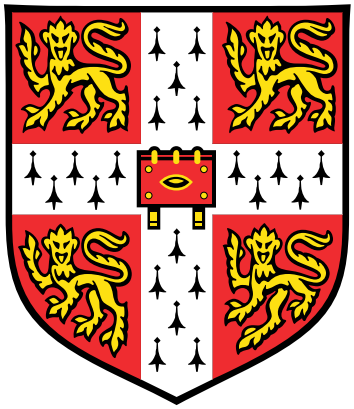
Computer Science Tripos - Part II Project Proposal

An accelerated, network-assisted TCP recovery



Thanh Bui

Supervisor: Dr Noa Zilberman

Department of Computer Science
University of Cambridge

Originator: Dr Noa Zilberman

Director of Studies: Dr Graeme Jenkinson

Project Overseers: Dr Markus Kuhn, Dr Ewal and Dr Nada Amin

Downing College October 19, 2018

Introduction

Background

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, network interface card (NIC)). Examples of applications offloaded in the past include consensus protocols (NetPaxos[1]), sorting and even a game (Tic-Tac-Toe). Key-Value Store (KVS) is also among the popular type of in-network applications. In particular, the performance within the data center is strongly affected by TCP losses [1]. As the load increases, it becomes more difficult to maintain an acceptable TCP loss rate without exceeding the host capacity. Various congestion control and avoidance recovery mechanisms are thus of high importance in this field to minimise such loss rate.

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

The Project

The goal of the project will be to implement fast recovery by retransmitting lost packets (based on DUP ACKs) from the switch. The key idea is to build upon KVS concepts.

For this project, I will be using the P4 programming language[2]. 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.

[1] https://dl.acm.org/citation.cfm?id=2774999
[2] https://p4.org/

Starting Point

Platform & Language

This project will work mainly with a NetFPGA SUME[3] board, using P4 programming language. I will be using the P4-NetFPGA workflow, which provides infrastructure to compile P4 programs to NetFPGA. 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.

Computer Science Tripos

The relevant Tripos courses that can serve as a starting point for this project are primarily: *Computer Networking*, *Principles of Communications* and *ECAD and Architecture Practical Classes*. Since the courses are introductory, I will also consult Part III's *High Performance Networking* course. I also plan to bridge any knowledge gap through extensive personal reading as well as help from my project supervisors.

Resources Required

For this project I will be using my own computer, a 2017 MacBook Pro with a 2.3 GHz Intel Core i5 processor and 16 GB of RAM, that runs macOS Mojave. I accept full responsiblity for this machine, and I have made contingency plans to deal with hardware and/or software failures. Should that machine suddenly fail, I have another 16Gb of RAM computer with a 2.6 GHz Intel Core i7 processor that runs Ubuntu 18.04 LTS. If all else fails, I can continue to work on an MCS machine. Backups will be done weekly to Microsoft OneDrive and my external hardrive, and all my codes will be uploaded to GitHub for version control.

For the hardware prototype, I will require a NetFPGA SUME board. I will need access to a machine that has the SUME installed, and a 10G NIC. These will be supplied by my supervisor. I will be using my own machine for development, with remote access to a server with the SUME board inside.

[3]https://github.com/NetFPGA/NetFPGA-SUME-public/wiki

I will also require access to the lab network in order to *ssh* to the server with the SUME board. Development on my machine will require VPN access, for using floating licenses of the development tools. Lastly, I will require a second server for testing some of the extensions of the project.

Work to be Done

The main core component of the project is to be able to apply the KVS concept to implement TCP fast recovery by retransmitting lost packets from the switch. In order to do that, the following sub-tasks must be done:

1. First, since the project is done on NetFPGA using P4 programming language, both of which I am not familar with, I first have to study them thoroughly and be proficient working with the platform.

2. I would also need to study the application, including TCP congestion control and recovery mechanisms.

3. The next stage will be to design the architecture for the application. This is a non-trivial process. Generally, it will be to try to map the application to a match-action table.

4. The fourth stage involves implementing the architecture. Firstly, it will be coded and simulate – in P4, using bmv2 or Xilinx simulators – to ensure correctness.

5. After it runs smoothly in the software, it will then be compiled to the hardware. The aim for this stage is to get a working prototype in hardware.

6. Once the prototype is up and working, I need to implement further extensions to allows the prototype to support a variety of parameters/conditions (which will be discussed in **Possible Extensions** section).

Success Criteria

This project will be deemed a success if I managed to study and design an architecture for the application, as well as succeeded to implement that design. More concretely:

1. I have an implementation of the application written in P4 and working in simulation.

2. I have a working prototype. In other words, my design runs on the hardware.

3. I am able to demonstrate interoperability with a software-based client/application.

4. I can provide a performance evaluation of the design.

Possible Extensions

If the core parts of the project are successful and completed within a reasonable time, I shall then try to investigate and implement further extensions. Some possible options are:

1.

2.

3. Extending to provide concurrent support to standard network forwarding (e.g. IPv4 in parallel).

4.

Timetable

The schedule is broken into 15 2-week periods, with the first period starting on 19/10/2018.

1. **Michaelmas weeks 2–4 [19/10–5/11]:** Preparatory reading on the P4 programming language and setting up the NetFPGA platform. Going through the tutorials and experimenting with some examples. Study the application: TCP congestion control and recovery mechanisms.

2. **Michaelmas weeks 5–6 [6/11–19/11]:** Design the architecture: mapping the application to a match-action pipeline.
   Milestone: Understand the application architecture. Be able to map the application to a match-action pipeline.

3. **Michaelmas weeks 7–8 [20/11–28/11]:** Start implementing of the design by writing the basic code.

4. **Michaelmas vacation weeks 1–2 [30/11–12/12]:** Simulate the application using bmv2/Xilins simulators.

Milestone: **Basic design written in P4 working in simulation, with minimal bugs left.** .

5. **Michaelmas vacation weeks 3–4 [13/12–26/12]:** Start to implement the working code into hardware. Start writing progress report.

6. **Michaelmas vacation weeks 5–7 [27/12–16/1]:** Continue with the hardware implemention and the progress report.
   Milestone: **Have the application working in hardware, a completed progress report and a presentation for demonstration purposes.**

7. **Lent weeks 1–2 [17/1–30/1]:** Start working on extension 1–2

8. **Lent weeks 3–4 [31/1–13/2]:** Working on extension 3–4.
   Milestone: **The prototype continues to work well with the extensions implemented.**

9. **Lent weeks 5–6 [14/2–27/2]:** Possible overflow from the previous weeks. Clean up codes and repository. Start writing dissertation main chapters.

10. **Lent weeks 7–8 [28/2–13/3]:** Continue writing dissertation.
    Milestone: **Completed working prototype with core components and extensions in place. First draft of dissertation**

11. **Easter vacation weeks 1–2 [14/3–27/3]:** Continue writing dissertation. Review cycles and corrections to dissertation towards the end of the vacation.

12. **Easter vacation weeks 3–4 [28/3–10/4]:** Continue writing dissertation. Review cycles and corrections to dissertation towards the end of the vacation.

13. **Easter vacation weeks 5-6 [11/4–24/4]:** Continue writing dissertation. Review cycles and corrections to dissertation towards the end of the vacation.

14. **Easter term 1–2 [25/4–8/5]:** Completed dissertation. Proof reading and then an early submission so as to concentrate on examination revision.

15. **Easter term 3 [9/5–17/5]:** Buffer week.

References

[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*, pp. 201–214, Springer, 2017.