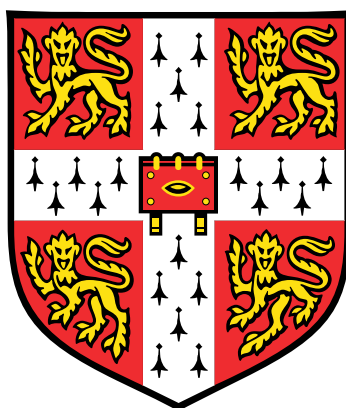


Computer Science Tripos - Part II Project Proposal

# 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 Eyal and Dr Nada Amin





Downing College

October 19, 2018

# Introduction

## Background



TCP (Transmission Control Protocol) is the protocol of choice in many data centers.


 However, it is very sensitive to losses (by design - as many packet drops happen in the network interface card (NIC) of the server, due to Direct Memory Access (DMA)), which can degrade the performance within the data centers significantly [1]. As the load increases, it becomes even 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. 

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), distributed systems functions such as consensus (NetPaxos[?], P4xos), various caching (netCache, netChain) 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 recovery in order to improve cross-datacentre performance.

## The Project

Currently, when there is packet loss, signaled by duplicate ACKs (DUP ACKs), the DUP ACKs will traverse all the way back to the host. The host receives the DUP ACKs, then resends the packet (Fig1 - sketch.png). The goal of my project is to implement a fast TCP recovery mechanism, which aims to reduce the response time to DUP ACK  by retransmitting packets from within the network (e.g., from the switch) instead of sending the DUP ACKs back to the host. The implementation will be based on KVS concepts 

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

# Starting Point

## Platform & Language

This project will work mainly with a NetFPGA SUME<sup>1</sup> 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 supervisor.

## 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 responsibility 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 harddrive, 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.



---

<sup>1</sup><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, where the sequence number and flow Id of a packet are the key and the packet is the value, to implement TCP fast recovery. 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 familiar with, I first have to study them thoroughly and be proficient working with the platform.
2. I would also need to study and gain a better understanding of KVS applications, TCP congestion control and recovery mechanisms, as well as their use cases (e.g. High Frequency Trading, in-datacenter latency sensitive applications, etc.).
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 pipeline.
4. The fourth stage involves implementing the architecture in code. The program, which will be written in P4, will have the basic functionalities such as:
  - A basic L3 switch function and TCP decoding (send TCP, read the flow information from the table/register)
  - Matching packets to a "*key*" (its flow Id):
    - If the key is of a new packet, store it (SET())
    - If the key is of a DUP ACK, read the packet and re-send (GET())
  -  If the number of DUP ACKs is greater than  $N$ , don't resend. Instead, forward to host as the standard conventions of handling TCP.
  -  Also, don't resend if the flow Id is not in a "selected" table. In other words, we only resend if the packets belongs to a pre-selected group (!?)

5. **The simulation stage:** perform functionality test by simulating the program using bmv2 or Xilinx simulators to ensure correctness.
6. After it runs smoothly in the software, it will then be compiled to the hardware. Ensure that the basic functionalities mentioned above work in hardware.
7. **The hardware stage:** demonstrate interoperability with a software-based client/application.

Possibly, I will implement a drop  $1 : N$  packets at the server to force DUP ACKs, and I will also be using a benchmark such as mutilate (<https://github.com/leverich/mutilate> - a memcached bm) or a tool such as OSNT (synthetic benchmark) as the applications on top of the network. The criterion to evaluate the performance is the flow completion time. There is no intention to use network simulators such as ns2, omnet++, etc. This is outside the scope of this project.

The aim for this stage is to get a working prototype in hardware that supports a single flow and single packet size, and evaluate on its performance.

8. 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.
2. The implementation works in simulation (using bmv2/Xilinx simulators).
3. I have a working prototype. In other words, my design runs on the hardware.
4. I am able to demonstrate interoperability with a software-based client/application.
5. 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. Supporting more than a single flow, and supporting the configuration of flows to monitor (& retransmit)
2. Supporting different packet sizes (depends on workflow limitations).
3. Sending a notification to the source if multiple retransmits fail.
4. Adaptively installing and deleting from the list of flows to monitor.
5. A performance evaluation comparison to existing TCP recovery mechanisms.

## 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 TCP congestion control and recovery mechanisms beyond the material taught in *Computer Networking* and *Principles of Communications* courses.
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/Xilinx 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. Write a hardware-test program (Python-based).
6. **Michaelmas vacation weeks 5–7 [27/12–16/1]:** Demonstrate interoperability with a software-based client/application. Start writing 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]:** Work on the design’s performance testing and evaluation.
8. **Lent weeks 3–4 [31/1–13/2]:** Continue working on testing and evaluation of the design.  
**Milestone:** The project is finished and meets the success criteria.
9. **Lent weeks 5–6 [14/2–27/2]:** Start working on extensions, based on project’s progress.  
**Milestone:** The prototype continues to work well with the extensions implemented.
10. **Lent weeks 7–8 [28/2–13/3]:** Possible overflow from the previous weeks. Clean up codes and repository. Start writing dissertation main chapters.  
**Milestone:** Completed working prototype with core components and extensions in place. First draft of dissertation
11. **Easter vacation:** Continue writing dissertation. Review cycles and corrections to dissertation towards the end of the vacation.
12. **Easter term 1–2 [25/4–8/5]:** Completed dissertation. Proof reading and then an early submission so as to concentrate on examination revision.
13. **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.