UNIVERSITY OF STAVANGER

BACHELOR THESIS

DATBAC

Make the Internet Faster!
Improving Alternative Backoff with ECN in Linux

Students
Dan Erik RAMSNES
Erlend Moen AL-KASIM

Supervisor
Naeem KHADEMI

February 28, 2020

Abstract

Lorem ipsum sit amet, consectetur adipiscing elit. Sed mollis dolor risus, a pulvinar quam pretium et. Suspendisse ut dolor arcu. Nulla facilisi. Nullam finibus vestibulum nulla, ac sollicitudin neque pretium vitae. Donec in est pretium, elementum velit et, pulvinar enim. Vestibulum consectetur et lectus ut fringilla. Donec malesuada, ante quis ultricies feugiat, leo tortor egestas nunc, non sagittis nisl lorem ac felis. Donec venenatis eget tortor et aliquam. Integer a sapien ultricies, dapibus erat sit amet, luctus ex.

Donec vitae metus pretium, tempus sapien eget, maximus quam. Nam dictum aliquam mi, ut fringilla nunc vehicula a. Morbi vitae dictum eros. Phasellus non urna felis. Etiam eu lectus justo. Etiam et ex ultrices, elementum erat laoreet, hendrerit lorem. Vivamus imperdiet consectetur dictum. Sed ac orci placerat quam rutrum pellentesque. Proin sollicitudin diam et erat feugiat feugiat. Quisque tempus velit viverra sem aliquet feugiat vitae et augue. Donec at odio viverra, posuere tortor eu, volutpat quam. Proin id rutrum metus, id mollis ipsum.

Sed pulvinar tristique nibh eu convallis. Integer luctus pretium massa sit amet placerat. Donec tincidunt consectetur efficitur. Nam tincidunt libero ut nisi lacinia, rhoncus cursus elit lobortis. Interdum et malesuada fames ac ante ipsum primis in faucibus. Duis vel semper lectus. Donec et ullamcorper turpis.

Contents

Abstract i

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Goals and Research Questions . . . 1
- 1.3 Research Methodology . . . 1
- 1.4 Contributions . . . 1
- 1.5 Thesis Structure . . . 1

2 Literature Review 2
- 2.1 Network Delay and Latency . . . 2
- 2.2 Transmission Control Protocol . . . 2
  - 2.2.1 Network Congestion . . . 3
  - 2.2.2 Congestion Control . . . 3
  - 2.2.3 TCP Tahoe and Reno . . . 5
- 2.3 Active Queue Management . . . 6
- 2.4 Explicit Congestion Notification . 6
  - 2.4.1 Legacy ECN . . . 6
  - 2.4.2 Accuracy ECN . . . 6
- 2.5 Alternative Backoff with ECN . . . 6

3 Methodology 7
- 3.1 Network Topology . . . 7
  - 3.1.1 Raspberry Pi 4 Cluster . . . 7
- 3.2 TCP Experimentation with TEACUP 7
  - 3.2.1 Exposing TCP State with web10g . . . 7
- 3.3 Achieving Low Latency with ABE 7
- 3.4 Improving ABE by Adapting Its Reduction Factor β . . . 7

4 Results 8

5 Conclusion 9

A The PI4-Cluster Testbed 10
- A.1 Setting Up Dual Boot . . . 10
- A.2 Compiling Mainline Kernel 5.5 for Raspberry Pi 4 . . . 11
- A.3 Patching web10g on Mainline Kernel 5.5 . . . 11

Terms 12

References 13

Chapter 1

Introduction

This chapter aims at giving an introduction and overview of the thesis. It starts with a brief explanation of why Internet today still feels slow despite major advances in technology, followed up by establishing the goals and research questions for the entire thesis. To address the research questions, a small look into the research methodology is presented. In the final section, an outline of the thesis structure is discussed.

Donec vitae metus pretium [2], tempus sapien eget, maximus . Nam dictum aliquam mi, ut fringilla nunc vehicula a. Morbi vitae dictum eros.

1.1 Motivation

Maecenas pulvinar quis quam eu convallis. Fusce vulputate sodales suscipit. Nullam porttitor hendrerit lacinia. Cras tincidunt ultrices lobortis. Sed ac sem efficitur, faucibus nunc vehicula, elementum arcu. Aenean pellentesque augue ut massa ullamcorper congue. Phasellus varius at erat at lobortis. Ut nunc metus, consequat nec blandit in, laoreet nec ligula. Integer interdum, massa eu accumsan eleifend, nunc nunc ultrices augue, at commodo risus ligula varius ex. Praesent blandit risus sit amet tellus semper, a suscipit libero malesuada. Mauris nec lacus ipsum. Nullam at ipsum rhoncus, mattis nibh vitae, placerat odio. Sed nec leo id dui vulputate accumsan. Pellentesque placerat congue arcu id pharetra. Maecenas imperdiet ex sed vehicula euismod.

1.2 Goals and Research Questions

Aliquam quis imperdiet orci, nec vestibulum risus. Curabitur vitae euismod mauris, ut iaculis erat. Ut interdum ex ac elit ultrices, at mattis magna ultricies. Duis ac suscipit nibh. Mauris sagittis consequat elit eget euismod. Mauris tincidunt dui ex, sit amet vulputate sem pretium non. Quisque at nulla lobortis, facilisis ligula eu, efficitur ante. Praesent bibendum dolor purus, non placerat felis pellentesque eu.

1.3 Research Methodology

1.4 Contributions

1.5 Thesis Structure

Chapter 2

Literature Review

This chapter presents the background theory for which this thesis is based upon.

TODO: improve introduction to chapter...

2.1 Network Delay and Latency

The time it takes for a bit of data to travel across the network from one communication endpoint to another is known as *delay*. The process for such a transmission involves many components. A typical example where the data traverses an intermediate device before reaching destination follows. First, the data to be sent is usually created by an application. The data will then be handed over to the Operating System (OS) which passes it to a network card. From there, the data will be encoded and transmitted over a physical medium and eventually received by an intermediate device, such as a router. The router will then analyze the data and retransmit it over another medium that points to the destination. Finally, the data reaches the receiver. The whole process can happen in either multiples or fractions of seconds.

TODO: maybe show a picture of the typical example

Network delay is therefore divided into the following four parts:

- Processing delay — time it takes router to process the packet header
- Queuing delay — time the packet spends in routing queues
- Transmission delay — time it takes to push the packet's bits onto the link
- Propagation delay — time for a signal to reach its destination

It is common to notify the sender that the receiver actually got the data. This is done by sending a signal from the receiver to the sender, known as an acknowledgement (ACK). The total time it takes for a sender to send data *and* receive back an ACK is known as *latency* or Round Trip Time (RTT).

In this thesis, we are mainly concerned with the *queuing* delay part.

2.2 Transmission Control Protocol

Whenever a user sends an email, or any data over the network for that matter, one should expect some kind of assurance that the delivery of the data was successful. This notion is known as *reliability*, and is one of the key components of the Transmission Control Protocol (TCP) and why it is one of the main protocols for transmitting data on the Internet. In essence, TCP is a communication protocol that provides reliable, ordered, and error-checked delivery of data between applications such as World Wide Web (WWW), email and file transfer.

TODO: write a bit more about tcp in general such a connection managent and flow control

2.2.1 Network Congestion

In the same sense that traffic on the road can come to a halt, the same is true for traffic on a network. This is known as *congestion*, and is usually caused by overutilization. That is, network devices such as a router have finite resources, and thus too much traffic will cause the device to carry more data than it can handle which leads to congestion on the network. Typical effects include queueing delay, packet loss or the blocking of new connections.

TODO: show image illustrating congestion

TODO: write more about general network congestion...

2.2.2 Congestion Control

Another key compononent of TCP is the ability to either prevent congestion or mitigate it after it occurs, known simply as Congestion Control (CC). The means of applying CC is simple; ensure that the sender does not overflow the network. In other words, the sender's rate needs to be adjusted based on the condition of the network. Now to the hard part; how to adjust it?

To shed some light on this, TCP includes a state variable called Congestion Window (CWND) which limits the amount of data that a sender can send before receiving an ACK back. This variable is known only to the sender, and serves as the key role for attaining CC. The means of adjusting CWND is referred to as an *congestion-control algorithm*, and consists of four intertwined parts:

- Slow start —
- Congestion avoidance —
- Fast retransmit —
- Fast recovery —

The following sections will describe each part in more detail.

Slow Start and Congestion Avoidance

Slow start and congestion avoidance are two independent algorithms with different objectives, but in practice are implemented together and so will be discussed together.

Upon a TCP connection, the CWND value is initialized to a single *segment* with a specified size. This size is either announced by the the other end, known as the Receiver Window (RWND), or set to a typical default [1]. In the same sense that CWND is a sender-side limit, RWND is a receiver-side limit. Together, the minimum of CWND and RWND governs the data transmission in slow start. However, at the beginning of a transmission, the network conditions are unknown, so the goal of the slow start phase is simple; slowly probe the network to determine the available capacity.

To find the capacity, slow start works as follows; the sender starts by transmitting one segment. For every received ACK, increment the CWND by another segment, effectively doubling it every RTT. Repeat this process until a timeout occurs. That is, until the capacity has been reached that is signaled by a router starting to drop packets, which tells the sender that its CWND has grown too large.

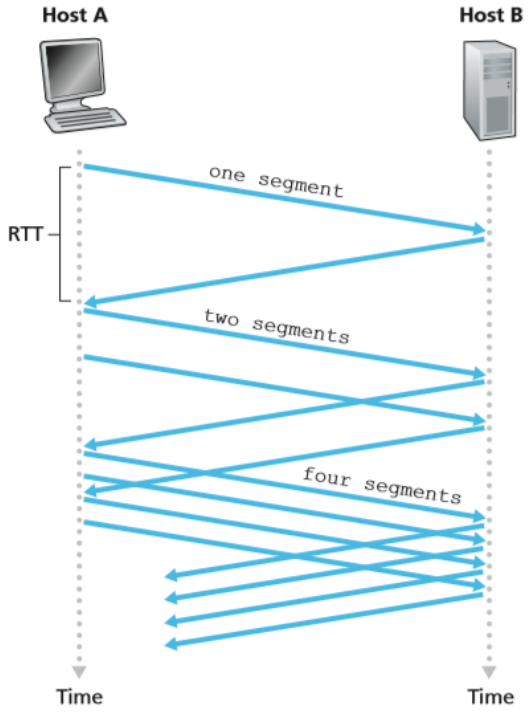

Figure 2.1: TCP slow start. Despite the name, the sending rate experiences exponential growth. TODO: update image

As a response to hitting the threshold, TCP will now halve the CWND since that is the last known value to not induce a timeout. This new value is known as the slow start threshold (ssthresh), and is another TCP state variable used to determine whether the slow start or congestion avoidance algorithm is used to control the data transmission. In other words, the slow start phase ends when CWND reaches or exceeds ssthresh.

The goal of congestion avoidance is also simple; avoid congestion, as the name implies. But the challenge here is to maintain a transmission rate that it not too low, otherwise the link becomes underutilized, and at the same time probe for available bandwidth without overflowing the network too quickly. To do so, a feedback control algorithm known as additive-increase/multiplicative-decrease (AIMD) is used. AIMD provides linear growth of the CWND when probing for available bandwidth, and a multiplicative reduction when congestion is detected.

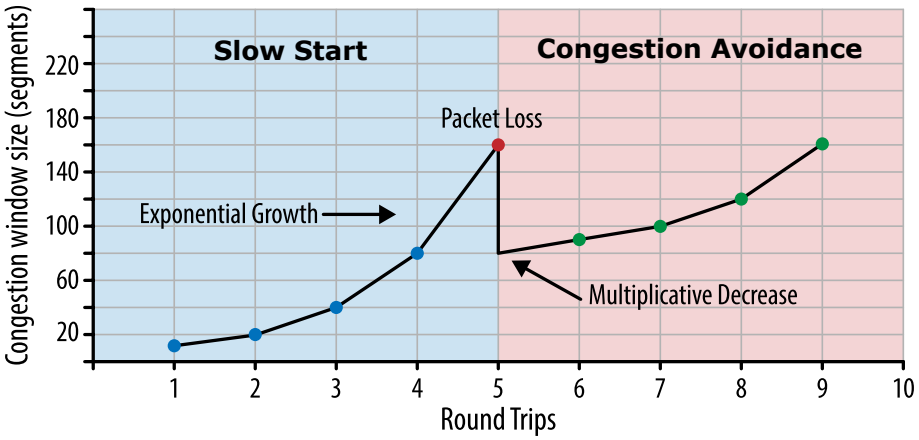

Figure 2.2: The two main parts of CC; the slow start phase where CWND grows exponentially, and the congestion avoidance phase where the transmission rate is increased more conservatively. (*source*)

TODO: maybe write more about AIMD with some math TODO: addd conclusion to this section

Fast Retransmit and Fast Recovery

Every packet sent using TCP has a *sequence number*, so that the data can reassembled in correct order on the receiver side. But it is important to note that TCP may send an out-of-order packet, to which the receiver should act upon immediately by sending back a duplicate ACK.

TODO: follow up...

2.2.3 TCP Tahoe and Reno

TODO: show some early examples TCP congestion-control algorithms

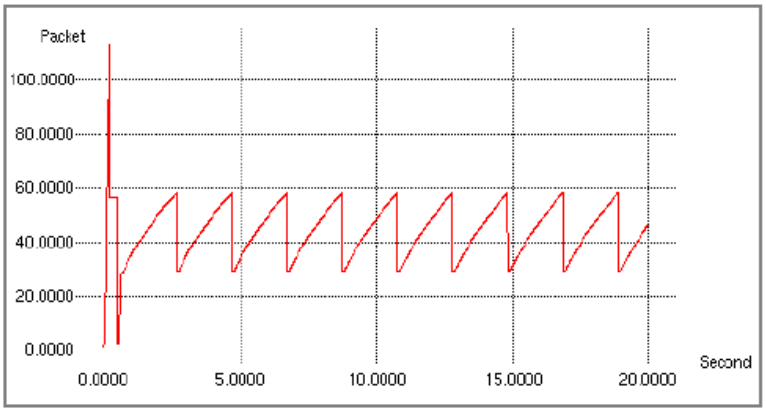

Figure 2.3: The classic *sawtooth* pattern of CWND using an older TCP congestion-algorithm known as *NewReno*. TODO: update image

2.3 Active Queue Management

2.4 Explicit Congestion Notification

Explicit Congestion Notification (ECN) is an extension to TCP that allows routers to notify end points on impending congestion *without* dropping packets.

2.4.1 Legacy ECN

In legacy ECN, the router notifies end hosts of congestion by setting a Congestion Encountered (CE) flag in the IP header on ECN enabled packets when experiencing congestion. The reciever of the packet then reflects this back to the sender by setting an ECN-Echo (ECE) in the TCP header. It keeps doing this until the sender responds back with a segment with Congestion Window Reduced (CWR) set, indicating that the sender has backed off.

2.4.2 Accuracy ECN

2.5 Alternative Backoff with ECN

Chapter 3

Methodology

3.1 Network Topology

3.1.1 Raspberry Pi 4 Cluster

3.2 TCP Experimentation with TEACUP

3.2.1 Exposing TCP State with web10g

3.3 Achieving Low Latency with ABE

3.4 Improving ABE by Adapting Its Reduction Factor β

Chapter 4

Results

Chapter 5

Conclusion

Appendix A

The PI4-Cluster Testbed

A.1 Setting Up Dual Boot

First install Ubuntu. When asked for partitioning the disk, choose manual, select the disk and confirm creating a new empty partition with yes. Select the newly created empty partition followed by create a new partition and set a size for it. The type should be of primary, location at beginning and mounting point root. Finish off with done setting up the partition followed by finish partitioning and write changes to disk.

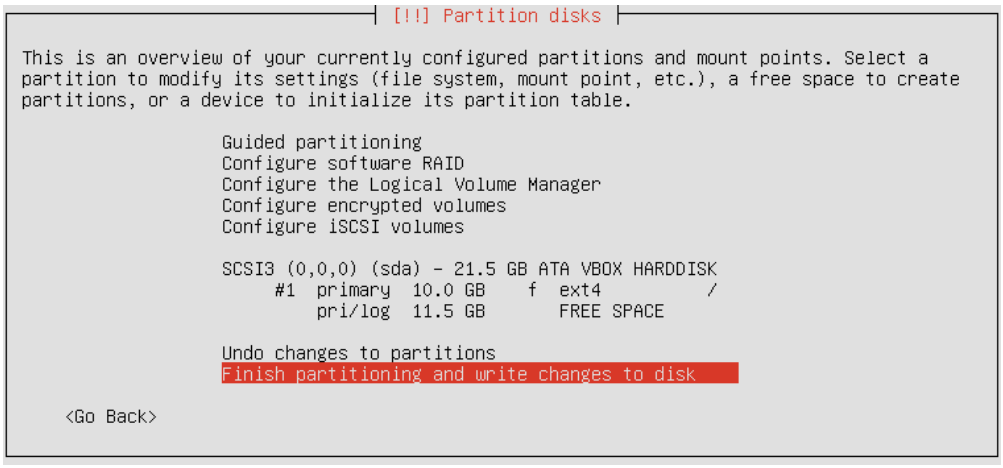

Figure A.1: The partition editor for Ubuntu.

Next, install FreeBSD. When asked for partitioning the disk, choose **auto** (UFS) followed by partition. Set a size, hit ok and finish.

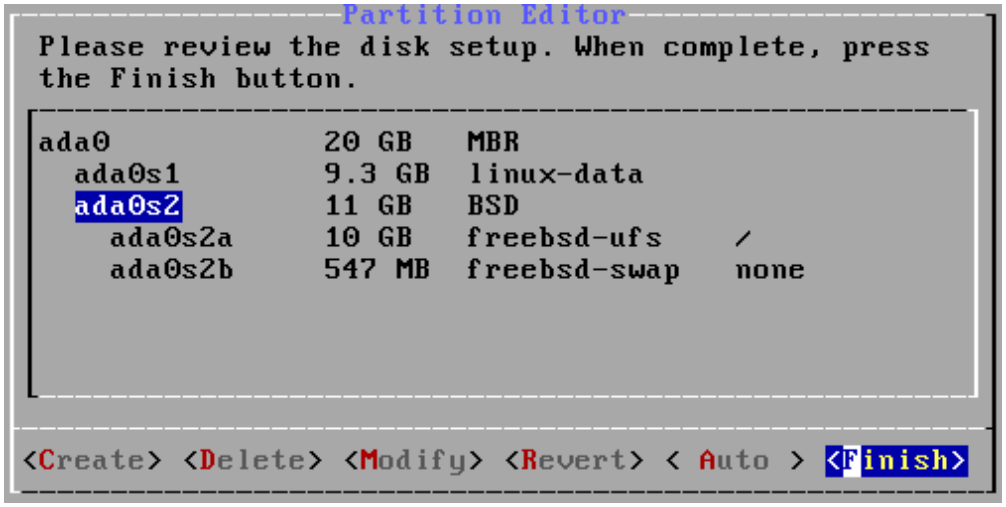

Figure A.2: The partition editor for FreeBSD.

After installing both systems, only Ubuntu is presented in the GRand Unified Bootloader (GRUB). To add FreeBSD as an option, run `sudo nano /etc/grub.d/40_custom` in Ubuntu, and add the following entry:

```
menuentry "FreeBSD" {
    insmod ufs2
    set root=(hd0,2)
    kfreebsd /boot/loader
}
```

Then update GRUB with `sudo update-grub`. The FreeBSD option should now be available when rebooting. If the bootloader won't display, hold the RIGHT SHIFT key upon booting.

To enable a one-time reboot into FreeBSD from Ubuntu, run the command `grub-editenv /boot/grub/grubenv set next_entry="FreeBSD"` and reboot with `sudo reboot`.

A.2 Compiling Mainline Kernel 5.5 for Raspberry Pi 4

A.3 Patching web10g on Mainline Kernel 5.5

Terms

Acknowledgement (ACK) A signal that is passed between communicating processes, computers, or devices to signify acknowledgement, or receipt of message, as part of a communications protocol.

Additive-increase/Multiplicative-decrease (AIMD) A feedback control algorithm best known for its use in TCP congestion control. AIMD combines linear growth of the congestion window with an exponential reduction when congestion is detected.

Congestion Control (CC) The process of managing the sender's packet rate to not overwhelm the network.

Congestion Window (CWND) A TCP state variable that limits the amount of data the sender can send into the network before receiving an ACK.

Explicit Congestion Notification (ECN) An extension to IP and TCP that allows end-to-end notification of network congestion without dropping packets.

GRand Unified Bootloader (GRUB) A Multiboot boot loader. It was derived from GRUB, the GRand Unified Bootloader, which was originally designed and implemented by Erich Stefan Boleyn.

Operating System (OS) System software that manages computer hardware, software resources, and provides common services for computer programs.

Receiver Window (RWND) A TCP state variable that advertises the amount of data that the receiver can receive.

Round Trip Time (RTT) The time it takes for a signal to be sent plus the time for an ACK of that signal to be received.

Slow start threshold (ssthresh) A TCP state variable used to determine whether the slow start or congestion avoidance algorithm is used to control data transmission.

Transmission Control Protocol (TCP) One of the main communication protocols of the Internet that defines how to establish and maintain a network conversation through which applications can exchange data.

World Wide Web (WWW) Commonly known as the Web; an information system in the form of webpages that web browsers can read and interact with.

References

[1] M. Allman, V. Paxson, and E. Blanton. *TCP Congestion Control*. RFC 5681. RFC Editor, Aug. 2009. URL: https://tools.ietf.org/html/rfc5681.

[2] Naeem Khademi et al. "Alternative Backoff: Achieving Low Latency and High Throughput with ECN and AQM". In: (2017), p. 9.