Efficient Paxos on a software defined network
** DRAFT **

Christian Stigen Larsen

May 15th, 2014

Contents

Chapter 1

Introduction

1.1 Software defined networking

Software defined networking emerged from research at Berkeley and Stanford around 2008 as a way to enable networks to be defined and managed using software.

Central to the idea is to split the *control* and *data* planes. While routers traditionally contained both, one would now implement the controller in *software* running on servers and have them talk to each other using a predetermined protocol.

One such protocol is *OpenFlow*. It allows the two planes to interact using a predetermined set of commands. The controller configures so–called *flow tables* in the data plane which contain matching patterns and actions for incoming packets. These tables are designed for high–speed routing, and are thus quite simple. On the other hand, the controller can be arbitrarily complex.

To maintain high networking speeds, one wants to restrict the interaction between the data and control planes to a minimum. This happens when the data plane is able to route packets entirely on its own. The control plane update the flow tables as seldomly as possible.

The data plane can also forward packets to the control plane for routing decisions. The extreme case is when *all* packets are forwarded to the control plane. While this allows very advanced behaviour — such as deep packet inspection, for instance — it obviously comes at the cost of lower networking performance. Therefore, one must be diligent in designing such a network.

Today, several SDNs have been deployed in production networks and power some of the biggest services on the internet[1].

[1]Examples are Google's *Chubby* distributed lock service and the *Megastore* distributed, high–availability storage system.

1.2 Paxos

Paxos is a family of distributed, fault–tolerant consensus algorithms. It allows network nodes to reach *agreement* even in the face of intermittent network failures. For example, one can design a database system using Paxos to make sure that transactions are executed in the same order on all nodes.

Originally published in 1989 by Leslie Lamport[1], Paxos has spawned numerous extensions like Byzantine tolerance and so on.

1.3 A Paxos–enabled software defined network

Our aim is to build an efficient, *Paxos–enabled software defined network* where nodes are able to leverage the guarantees of Paxos without needing to handle the details.

For simplicity, we will constrain our scope to Paxos *phase two* where we have steady–state flow with no failures.

We will implement this in progressive stages:

- Stage 1: Implement Paxos entirely on the controller.
- Stage 2: Move parts of Paxos down from the controllers to the switches.
- Stage 3: Look for possible expansion of the OpenFlow protocol to facilitate running Paxos on the switches.

Stage 1 should be the easiest to implement, and will be used as a baseline for comparing networking performance with later stages.

Additionally, we will implement Paxos entirely on the network nodes, using the same topology, to get an idea of how much of a performance hit we take for running it on the controller.

In stage 2, we will look at how we can improve efficiency by moving parts of the algorithm down into the switches using the flow tables.

Finally, in stage 3 we will investigate whether it would make sense to enhance the OpenFlow protocol with new primitives to build fast networking services such as Paxos.

Our hypothesis is that by moving parts of the Paxos implementation down to the switches, we will get a more performant system than by running Paxos either on the controller or on the network nodes.

The thesis will therefore be a study of *feasibility.*

Chapter 2

Theory

2.1 Software defined networks

2.2 OpenFlow

2.3 Paxos

Appendix A

Acronyms

Bibliography

[1] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998. `http://doi.acm.org/10.1145/279227.279229`.