ScuDo

Scuola di Dottorato ~ Doctoral School

WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation

Doctoral Program in Energy Engineering (29^{th}cycle)

Writing Your Ph.D. Thesis in LaTeX

Using the POLITO Template

By

Mario Rossi

******

Supervisor(s):

Prof. A.B., Supervisor

Prof. C.D., Co-Supervisor

Doctoral Examination Committee:

Prof. A.B. , Referee, University of...

Prof. C.D, Referee, University of...

Prof. E.F, University of...

Prof. G.H, University of...

Prof. I.J, University of...

Politecnico di Torino

2017

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Mario Rossi
2017

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

I would like to dedicate this thesis to my loving parents

Acknowledgements

And I would like to acknowledge ...

Abstract

The field of software defined networking (SDN) has been rapidly evolving in the recent years. Multiple per-packet processing functions are being defined in order to introduce a tighter control over the network. However, these new functionalities can create considerable overhead and significant delays for control information exchanged with the controller. These factors lead to a reduced reactivity to network events and an overall performance degradation.

In order to overcome these limitations a big effort is being devoted towards the definition of a novel approach based on stateful functionalities. The use of stateful approaches allows to offload some of the controller functions directly to the switches by defining persistent states that can hold state information related to per-packet processing rules. The proposed solutions however make use of a centralized state placement, eventually forcing the majority of traffic to pass through a single device which can create single points of failure and reduced performance.

The purpose of my research was to design and implement a scalable stateful solution that would lead to an increased performance and reliability. The idea behind the proposed approach was to distribute the state information across multiple forwarding devices while guaranteeing high consistency among the states. The proposed solution is able to provide lower overhead and thus better performance in respect to the solutions present in literature and is able to prevent state information loss in case of isolated failures.

The implementation and evaluation has been performed in an emulated network environment with virtual switches programmed in P4, an emerging data-plane programming language.

In a scenario of a DDoS attack towards an autonomous system by neighboring autonomous systems the solution based on distributed states led to a substantial improvement in terms of reactiveness and introduced overhead in respect to solutions

based on classical SDN. The obtained results are also compared to the results produced by a collaborating research group whose effort was devoted to a yet non-standardized extension of an existing technology (OPP). Thanks to its flexibility the results obtained with P4 exhibited better properties in terms of overhead and performance when compared to the aforementioned technology.

Contents

Chapter 1

Introduction

Since the birth of Internet in the early 70's the global infrastructure has been evolving anarchically without any coordination. New autonomous systems were being added to the network without taking into account the global architecture which led to a complex and globally unknown structure. In order to compensate the lack of a global view of the network a lot of effort has been devoted into developing complex distributed protocols and local algorithms which conclusively led to the Internet of today: a chaotic entanglement of interconnections with hundreds of obscure vendor-specific protocols and expensive proprietary hardware that was designed to do what it should and nothing more.

The industry continued to move towards the same direction until the recent years during which the research community started to develop new paradigms in order to considerably simplify the world of networking and at the same time guarantee tighter control over the whole system.

1.1 SDN

Software Defined Networking (SDN) has been rapidly attracting attention of research communities and industries. The main idea behind SDN is to deprive the forwarding devices, namely switches and routers, of any decision capability and move all the intelligence to a single central entity, the controller, while allowing the switches to simply forward and process data packets. The forwarding capability is usually referred to as the forwarding (data) plane while the capability of performing complex

decisions like changes in forwarding policies is referred to as control plane. What we observe in SDN is essentially the decoupling of the two and the centralization of the control plane.

By centralizing the control plane it is possible but especially necessary to have a global view of the whole network in order to guarantee correct functionality of the whole system. A global real-time view of the network with all the active links and switches and their corresponding occupancy allows to constantly tune and optimize the whole system and dynamically respond to network events in the best possible way. This is achieved by continuously performing optimizations at the controller and then distributing the control information among all switches. Thanks to this SDN allows to achieve levels of optimization and control over the network that was not easily achievable in legacy systems with integrated control plane.

As a direct consequence of the decoupling of data and control plane it is possible to unbind the switches from vendor specific software and considerably reduce the architectural complexity leading to the possibility of development, without too much effort, open-source software and hardware for forwarding devices at a relatively low cost.

OpenFlow (OF) [1] is one of the most diffused standards for SDN switches and control plane. It has had a very large popularity among the industries thanks to a simple and consequently cheap to manufacture switch architecture that at the same time allows to use almost all of the features required by SDN. In order to hold control information OF switches comprise a set of volatile TCAM memories that are commonly adressed to as *Flow Tables*. The use of TCAMs allows to perform rapid matchings of the packets' fields against the control information and if no matching is found a dialogue with the controller will be initialized. This dialogue is composed by three message types: when there is a miss during a matching the switch will issue a *Message In* packet to the controller and carry the main information related to the packet that caused the miss. The controller will decide on the destination of the packet and will reply with a *Packet Out* message that will allow the switch to perform a required action. If the action needs to be performed on all packets of that type the controller will additionlly issue a *FlowMod* message that will embed the rule inside the switch's flow table.

1.2 Stateful SDN

SDN as any other technology while introducing novel functionalities carries also intrinsic limitations. Consider a classic functionality of SDN: whenever a switch encounters a flow for which no existing rule is present inside the forwarding tables. Since the switch is not able to perform any decision at its own it will generate a request to the controller asking for the set of actions to perform. This communication is required only once per flow and if we consider the flows to be sufficiently big in terms of packets the overhead and delay introduced by the dialogue with the controller is negligible when its spread over all the packets belonging to that flow. Now consider a scenario of a big company with thousands of servers and hundreds of SDN-enabled switches. A Network Engineer wants to keep track of per-packet information of each flow; for example the number of packets of each flow. This means that at least one switch that is traversed by that flow must keep count of the number of packets that has been seen by it. In classic SDN the only solution to this problem is to send a notification message to the controller for each received packet. Although this will not increase the delay of that flow if the forwarding and the notification operations are done concurrently it will considerably increase the network overhead since the switch will generate an additional amount of information equals to $OH = \alpha f$ i.e. proportional to the packet rate of the flow f. It is evident that the solution does not scale and it is too penalizing for the network.

In order to overcome this issue an extensive effort have been devoted into defining what is now called *Stateful SDN*, an extension of the classic SDN that introduces stateful memories and operations inside the switch. With the introduction of these functionalities the Network Engineer from the previous example can instruct the switches to count the packets and locally save the associated counters without forwarding any information to the controller unless explicitly required.

There have been some proposals aiming to bring stateful functionalities inside OF like OpenState [2] that is based on already present primitives and requires little to no modification to the actual hardware architecture. Another recent proposal, Open Packet Processor (OPP) [3] aims to extend the functionalities of OpenState by introducing more complicate and tunable stateful operations, however OPP relies on considerable changes inside the hardware architecture.

One of the major limitations of OF and its proposed extensions is the fact that it is de facto a standard that is under continuous development by the Open Network Foundation (ONF) which is responsible of deploying new versions of OF. The presence of such a big body behind a widely popular standard is for sure assuring the consumers about the quality of the product but at the same time introduces a significant moratorium of circa one year in the releases of new versions and consequently the support of new features. If a company decided to use custom labeling system for the packet inside its data-center or if it was to develop a completely new transport protocol they would need to apply for standardization procedure at ONF or spending their own resources in order to bring changes in the existent release of OF.

In 2014 a group of researchers in Stanford University came up with the idea of a protocol independent programming language for forwarding devices: P4 [4]. P4 has been introduces with the intent of solving the present issues and limitations of OF, in particular the absence of any possibility of defining parsing operations on non-standardized header instances.

1.3 Network programming and State sharding

A novel trend in the field of networking is so called "Network programmability" that introduces a concept of "One big switch" i.e. the network is exposed to the operator as a black-box with a set of capabilities. In a scenario in which an operators wants to define a network policy she is not required anymore to manually configure it on a device-per-device basis but she can delegate the actual matching of the rule to a network optimizer. The optimizer will translate the required policy into a set of constraints and requirements and will perform the most optimal placement inside the network hardware.

When dealing with stateful data plane operations an intrinsic limitations emerges when a global network-wide per-packet processing rule needs to be applied. Consider a scenario in which a policy on the maximum amount of packets inside the network must be defined, e.g. if we observe more than N packets inside the network we impose a dropping policy for all packets that arrive after the detection. With state of the art network programming mechanisms This type of policy becomes feasible only when all of the traffic entering our network converges to a single stateful switch. It is easy to observe that this solution will eventually lead to a scheme where the original

topology with multiple independent ingresses and with a balanced network resource utilization collapses to a topology with a single highly congested ingress, i.e. the selected stateful switch.

In this work it is presented a solution to this problem that exploits state replication and distribution. The main idea behind the proposed approach is to distribute states related to the network-wide policies on multiple switches, thus avoiding the congestion introduced with the approach using only one switch and at the same time guaranteeing network-wide policies correct functionalities with a deterministic error.

References

[1] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[2] Giuseppe Bianchi, Marco Bonola, Antonio Capone, and Carmelo Cascone. Openstate: programming platform-independent stateful openflow applications inside the switch. *ACM SIGCOMM Computer Communication Review*, 44(2):44–51, 2014.

[3] Giuseppe Bianchi, Marco Bonola, Salvatore Pontarelli, Davide Sanvito, Antonio Capone, and Carmelo Cascone. Open packet processor: a programmable architecture for wire speed platform-independent stateful in-network processing. *arXiv preprint arXiv:1605.01977*, 2016.

[4] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

Appendix A

How to install LaTeX

Windows OS

TeXLive package - full version

1. Download the TeXLive ISO (2.2GB) from
   https://www.tug.org/texlive/
2. Download WinCDEmu (if you don't have a virtual drive) from
   http://wincdemu.sysprogs.org/download/
3. To install Windows CD Emulator follow the instructions at
   http://wincdemu.sysprogs.org/tutorials/install/
4. Right click the iso and mount it using the WinCDEmu as shown in
   http://wincdemu.sysprogs.org/tutorials/mount/
5. Open your virtual drive and run setup.pl

or

Basic MikTeX - TeX distribution

1. Download Basic-MiKTeX(32bit or 64bit) from
   http://miktex.org/download

2. Run the installer
3. To add a new package go to Start » All Programs » MikTex » Maintenance (Admin) and choose Package Manager
4. Select or search for packages to install

TexStudio - TeX editor

1. Download TexStudio from
   http://texstudio.sourceforge.net/#downloads
2. Run the installer

Mac OS X

MacTeX - TeX distribution

1. Download the file from
   https://www.tug.org/mactex/
2. Extract and double click to run the installer. It does the entire configuration, sit back and relax.

TexStudio - TeX editor

1. Download TexStudio from
   http://texstudio.sourceforge.net/#downloads
2. Extract and Start

Unix/Linux

TeXLive - TEX distribution

Getting the distribution:

1. TexLive can be downloaded from http://www.tug.org/texlive/acquire-netinstall.html.
2. TexLive is provided by most operating system you can use (rpm,apt-get or yum) to get TexLive distributions

Installation

1. Mount the ISO file in the mnt directory

   ```
   mount -t iso9660 -o ro,loop,noauto /your/texlive####.iso /mnt
   ```

2. Install wget on your OS (use rpm, apt-get or yum install)
3. Run the installer script install-tl.

   ```
   cd /your/download/directory
   ./install-tl
   ```

4. Enter command 'i' for installation
5. Post-Installation configuration: http://www.tug.org/texlive/doc/texlive-en/texlive-en.html#x1-320003.4.1
6. Set the path for the directory of TexLive binaries in your .bashrc file

For 32bit OS

For Bourne-compatible shells such as bash, and using Intel x86 GNU/Linux and a default directory setup as an example, the file to edit might be

```
edit $~/.bashrc file and add following lines
PATH=/usr/local/texlive/2011/bin/i386-linux:$PATH;
export PATH
MANPATH=/usr/local/texlive/2011/texmf/doc/man:$MANPATH;
export MANPATH
INFOPATH=/usr/local/texlive/2011/texmf/doc/info:$INFOPATH;
export INFOPATH
```

For 64bit OS

```
edit $~/.bashrc file and add following lines
PATH=/usr/local/texlive/2011/bin/x86_64-linux:$PATH;
export PATH
MANPATH=/usr/local/texlive/2011/texmf/doc/man:$MANPATH;
export MANPATH
INFOPATH=/usr/local/texlive/2011/texmf/doc/info:$INFOPATH;
export INFOPATH
```

Fedora/RedHat/CentOS:

```
sudo yum install texlive
sudo yum install psutils
```

SUSE:

```
sudo zypper install texlive
```

Debian/Ubuntu:

```
sudo apt-get install texlive texlive-latex-extra
sudo apt-get install psutils
```