Ben-Gurion University of the Negev

Faculty of Engineering Science

School of Electrical and Computer Engineering

Dept. of Communication Systems Engineering

Fourth Year Engineering Project

Final Report

Distributed Caching-based Acceleration Mechanisms in Datacenter Networks

Project Number:	p-2022-093
Students:	Shir Granit 205531445, Anna Axalrod 324682475
Supervisors:	Prof. Chen Avin, Dr. Gabriel Scalosuv
Submitting Date:	27/07/2022

Table of Contents:

1. Project Summary

I. Project Summary (English):

Distributed Caching-based Acceleration Mechanisms in Datacenter Networks

Students Names: Axalrod Anna, Granit Shir

axalrod@post.bgu.ac.il

Switches in datacenter networks are required to store an enormous amount of traffic rules. Usually, those rules are stored in an external device to which the access slows the network's performance. The purpose of the project was to find a solution that prevents multiple access to the mentioned external device and speeds up the routing process.

In our project, we created a basic network, based on an SDN datacenter topology, with several switches that have cache storage for most used forwarding rules. In case of missing information, a switch sends the request to a higher-ranked switch. The highest rank switch is the SDN controller, which holds the entire forwarding policy.

By using caching technologies, and cleverly re-placing forwarding rules in distinct switches, we are able to reduce the number of accesses to a controller, thus speeding up the overall throughput in the network.

Keywords – Datacenter, Distributed Algorithms, Cache, Software-Defined Networking, Rule, Routing, Mininet, P4, Pipeline.

II. Project Summary (Hebrew):

<u>מנגנוני האצה מבוזרים מבוססי מטמון ברשתות מרכזי נתונים</u>

שמות הסטודנטים: אקסלרוד אנה, גרניט שיר

שמות המנחים: פרופ' אבין חן, ד"ר סקלוסוב גבי

axalrod@post.bgu.ac.il

מתגים במרכזי נתונים נדרשים לאחסן מספר רב מאוד של חוקי תעבורה. בדרך כלל חוקים אלו נשמרים במכשיר חיצוני, אליו הגישה מאטה את ביצועי הרשת. מטרת הפרויקט הייתה למצוא פתרון אשר ימנע את הגישה המרובה למכשיר החיצוני ובכך יאיץ משמעותית את תהליך הניתוב ברשת.

בפרויקט, יצרנו רשת מוגדרת תוכנה (SDN), על פי טופולוגיה מוכרת של מרכזי נתונים, עם מספר מתגים להם קיים זיכרון מטמון ששומר את חוקי התעבורה השכיחים ביותר. במקרה של מידע חסר, מתג מעביר את בקשת הניתוב למתג בדרגה הירככית גבוהה יותר. המתג בעל הדרגה ההירככית הגבוהה ביותר נקרא המתג המרכזי (controller), והוא מחזיק את כל חוקי התעבורה ברשת.

על ידי שימוש בטכנולוגיות מבוססות זיכרון מטמון והחלפת חוקים במתגים נפרדים בצורה חכמה, אנחנו יכולים לצמצם את מספר הפניות לבקר, ובכך להאיץ את התפוקה הכוללת ברשת.

<u>מילות מפתח</u>: מרכזי נתונים, אלגוריתמים מבוזרים, זיכרון מטמון, SDN (רשת מוגדרת תוכנה), חוקים, ניתוב, P4 (שפת תכנות לרשתות תקשורת), Mininet (תוכנת אמולציה לרשתות), צינור תעבורה.

2. Introduction

A datacenter (DC) is a facility, usually a building, a dedicated space within a building, or a group of buildings, which is used to house computer systems and associated components, such as switches, routers, and storage systems. In modern times, many companies and individuals use the services of DCs through different platforms. DCs rely on high connectivity to function properly.

Switches are hardware components that can connect to several servers and/or one another, and forward packets in a specific direction (through a specific exit port). Switches are necessary for DCs networks and influence the general throughput.

Due to the enormous number of servers in a DC, switches are required to store an enormous amount of forwarding rules. These rules tell the switch how to handle a packet that matches a certain rule. Like any other hardware, switches have memory limitation, therefore cannot poses all forwarding rules of the DC.

Modern DC components, such as Virtual machines (VMs) and cloud computing components, require the use of an approach called a Software-defined Network (SDN). In SDN, control and management of the network are run by centralized software [1]. To do so, a controller is usually used. The controller usually has high computing capabilities and functions as a key component in an SDN network.

DC, like other networks, relay on switches to navigate traffic through the network. SDN-DC networks usually use programmable switches to provide a flexible packet forwarding capability. Therefore, the switches in the network can be manipulated, by code, to operate innovatively and construct new features.

In the SDN-DC network, due to the switches' limitations, the controller is the component that holds the entire DC forwarding policy. The controller will insert and delete relevant forwarding rules from switches. Therefore, the controller's algorithm has a great influence on the behavior of the traffic in a DC.

One approach of SDN-DC switch will forward a packet with a known destination address, meaning a destination that can be found in the switch's forwarding table. In case of missing information, a switch will pass the message to the controller, and it will either forward the message to its destination or will insert a new rule to the switch.

The topology discussed can be examples in fig. 1 below. The controller is considered to be a part of the control plane, and the switches are considered to be in the data plane. In fig. 1, P objects are servers and S objects are switches.

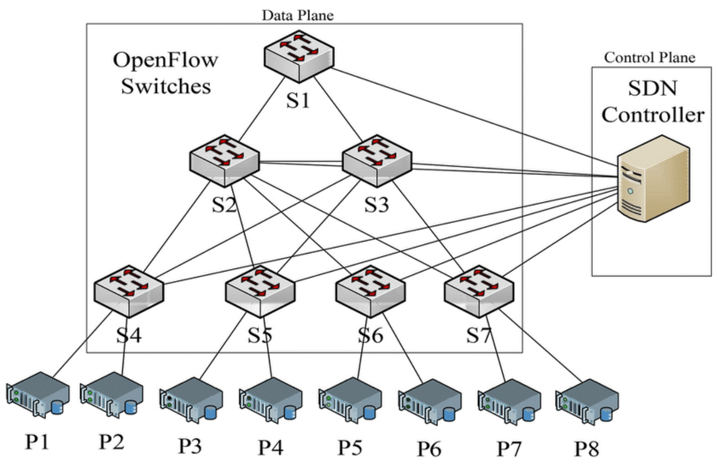


Figure 1: Software-defined load-balanced data center: design, implementation, and performance analysis. (From: Montazerolghaem, Ahmadreza., 2021)

2.1. Motivation

Cloud computing has become very popular in the past few years. Cloud computing servers are usually stored in DCs. As a result, DC networks are required to handle a great amount of traffic. To do so efficiently, inside communication must be efficient.

The main idea of our project was to use a sophisticated method to decide which rules will be placed inside the switches' forwarding rules. To do so, a proposed hardware addition was suggested, on top of the existing one (or on behalf of the existing one) and dedicated the additional memory space to serve as a caching mechanism [2].

Increasing the general throughput in a DC has great significance, financially mainly. The companies that maintain DCs will always prefer increasing their service quality, using the same resources.

2.2. Project Goal

In our project, we focused on a specific and common DC network layout, called 3-tier topology. 3-tier topology is a scalable and resilient network architecture for large enterprises [3]. Most networks use a 3-tier architecture that divides the network into three layers: core layer, distribution layer, and access layer [4]. Figure 2 illustrates the concept of division into different layers in a three-tier topology. The core layer consists of Core routers (or switches with CPU), the aggregation layer consists of Aggregation switches, and the access layer consists of switches called Top of Rack (ToR) switches. Each layer's purpose is different from the one below or under it. Core switches are usually switched with very high throughput and advanced routing capabilities, where Aggregation switches are mid-tier speed switches with an emphasis on uplink speeds, and ToR switches that gather all information from a certain rack, with emphasis on uplink speed [5].

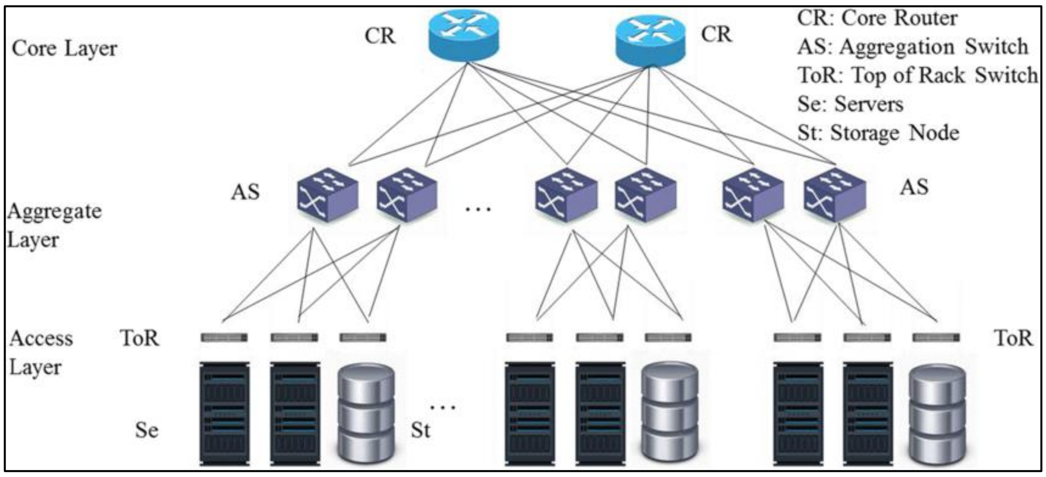


Figure 2: Datacenter 3-tier network architecture [6]

Our work was conducted as part of wider research, in collaboration with Nvidia. A cache mechanism was implemented on every switch in the network, which contained the adaptive forwarding rules needed by the switch. If the switch knows where to forward a certain packet, it will be considered a cache hit, otherwise it will be considered a cache miss.

The goal of this proposed solution is that the cache will hold the rules needed to handle as many packets as possible, thus maximizing the cache hit rate and reducing the number of hops to the higher layers in the network, thus reducing the overall unnecessary traffic in the network. When a cache miss happens, the unknown destination packet will be sent to a higher layer switch, hopefully, that this switch will contain the destination in its cache table. If a repetitive cache miss happens, the controller is our last line of defense, knowing all destinations in our network.

The controller manages our distributed algorithm, inserting rules to caches in switches belonging to different layers in the topology (Core, Aggregation, and ToR). The basic guideline consists of hit-threshold logic; if a certain address is being requested repeatedly, its forwarding rule will be inserted into the lower layer switch, thus getting a cache hit the next time this address will be requested.

2.3. Past work

To this day, two main approaches have been taken to solve the issue explained. The first approach is to use switches that can hold the entire forwarding policy necessary in the wanted environment within the DC. This solution requires a pre-known scale of the desired forwarding policy size and the correlated hardware memory in the switches. This solution faces a scalability issue.

The memory used in this solution is a Ternary Content Addressable Memory (TCAM) [7] which can perform fast parallel table lookup for stored entries. This memory is very expensive and consumes a high amount of electrical energy. Therefore, this solution cannot be scaled to the number of switches needed to operate nowadays DCs.

The second approach is to use cheaper switches with relatively small TCAM and offload the entire cloud-scale policy to some external device, with the sole purpose of storing these forwarding rules. Each switch will be connected to such a device (see Figure. 1), and upon packet arrival, it will ask the device for the correct forwarding rule for that certain packet.

This solution degrades the switches' response time to forwarding decisions. DC switches are required to perform at line rate when deciding where to forward each packet, and the channel between the switch and the external device creates a bottleneck in the network. This approach is the approach we tried to improve in our project. Lowering the bottleneck created in the network.

3. Technical Specifications

To prove that our improvement benefits the network by lowering the number of times a forwarding request is arriving at the SDN controller, we needed to emulate a DC network. To do so, we used an emulation environment.

Conducting an emulation and not a simulation gave us a perspective on real network connectivity and behavior. To our solution, a simulation would give a non-realistic result and we could not have been convinced of the nature of our solution the way we did.

3.1. Technologies and Methodologies

3.1.1. Mininet Emulation Environment

Mininet is a network emulator widely used to emulate virtual hosts, routers, network controllers, and switches. Mininet creates a realistic virtual network, running real kernel and application code, on a single machine (VM, cloud, or native) [8].

We built our network using Mininet in a private VM with ubuntu OS on our personal computers. Interacting with our environment through a built control line interface (CLI).

As seen in Fig. 3, a basic Mininet network can consist of virtual hosts, links, and switches and can be accessed through a CLI.

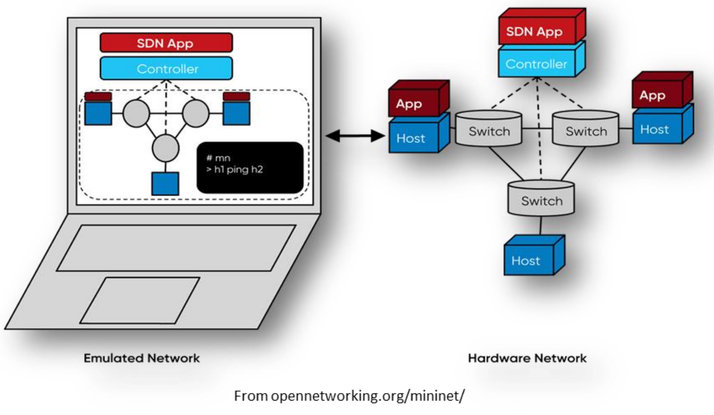


Figure 3: A Mininet network consists of isolated hosts, emulated links, and emulated switches [8]

3.1.2. P4 Programing Language

P4 is a programming language used to control the packet forwarding plane in programmable network devices, like routers and switches. P4 can be compiled in a Mininet emulation environment and program the network control plain behavior pre-run.

3.1.3. P4runtime API

p4runtime API is a control plane specification for controlling the data plane elements of a device defined or described by a P4 program. Through it, a user can redefine switches configured in P4. The protocol is still under development, but many specification sheets have already been published, strictly defining the API between data and control plane devices. [9]

3.1.4. SDN Controller

Our SDN controller is written in Python language, and loaded into a single-purposed switch, which is not defined as all other switches. The SDN controller listens to the entire network and gets unresolved destinations from lower layer switches in the network. The controller algorithm will be detailed in section 4.

3.1.5. BMv2 Programmable Switch

A version of a software switch called Behavioral Model version 2 (BMv2) [10]. This software is written in C++11, and it is widely used in the academy as well as the industry for research and development purposes, regarding P4 and SDN-related scenarios. We deployed the BMv2 on top of the Mininet Switches and used p4runtime to manage the control plane through the data plain.

3.2. Changes Occurred During the Year

As the researchers in NVIDIA worked alongside BGU research, attempts to improve and design our proof of concept (POC) reshaped. When we began our work, the wanted topology consisted only of a switch and 2 hosts; one functioning as the controller and the other as a rack (constructing traffic with heavy loads). We were requested to construct a 3-hosts topology, presented in Fig. 4. In this solution, a cache miss in host 2 or 3 would first be passed on to the correlated neighbor, instead of forwarding the query to host 1 – the controller.

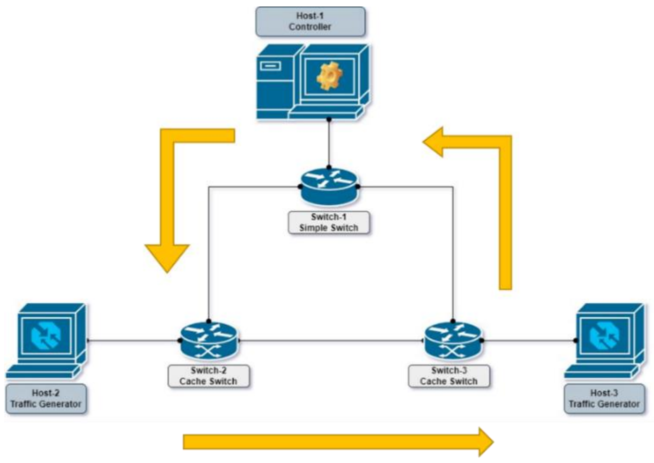


Figure 4: Previous Project Topology

During our year work, several other improvements have been suggested, the final construction is presented in the next section, section 4.

4. Approach and Design

Building a Mininet environment, we constructed a simple DC 3-tier topology, with an SDN controller. Using P4 we configured the initial configuration of the network switches. We designed the packet pipeline inside the switches and the cache mechanism using p4runtime API through the controller Python script.

Fig. 5 shows our topology, as it exists in Mininet. We created a network with 3 hosts (racks) and 7 BMv2 switches, with links as shown in the figure. Working with the objects' CLI, we loaded a Python program to the controller and a different one to each of the hosts, which represent racks.

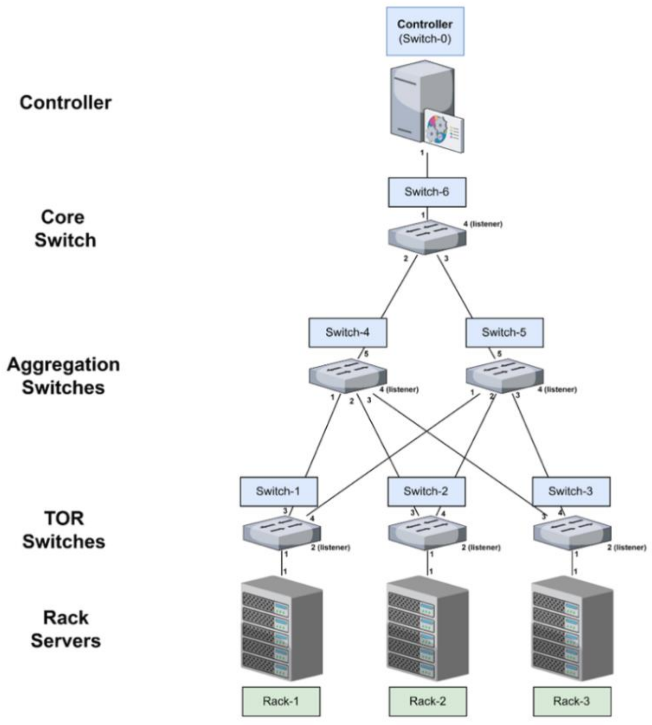


Figure 5: Our Network Topology

Flooding the network with DC emulated traffic the controller starts to react. At first, packets stream up through the layers until the controller receives a miss from switch #6 (S6) and inserts a new cache rule to S6. All switches count their cache hit counts, as thresholds.

The model suggests that if configured correctly, most cache hits will be in the ToR switches, then in the Aggregation switches, and then at the Core switches. The algorithm running in the controller can be explained in Fig. 6:

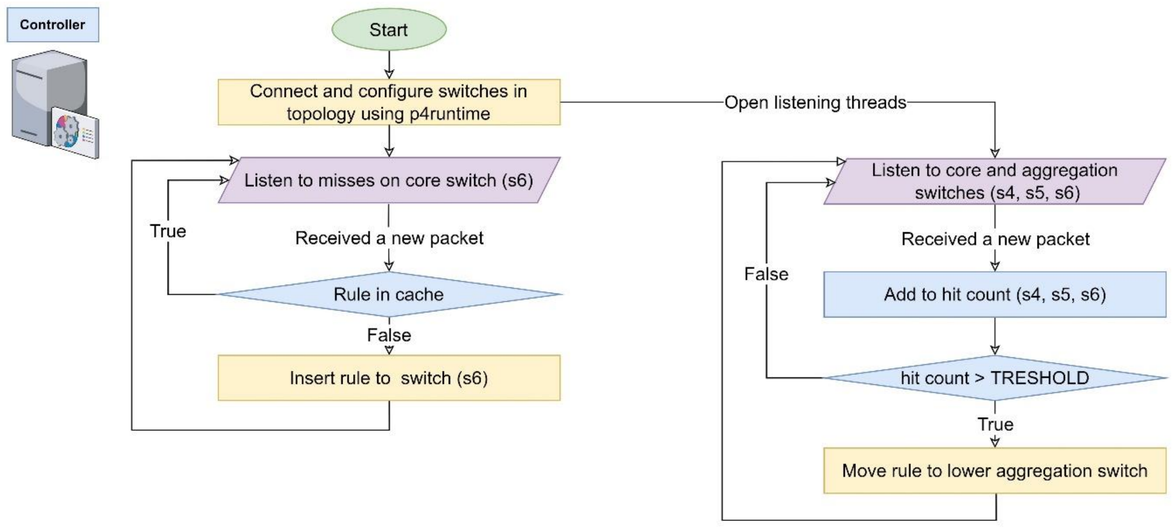

Controller
Start
Connect and configure switches in topology using p4runtime
Open listening threads
Listen to misses on core switch (s6)
Received a new packet
True
Rule in cache
False
Insert rule to switch (s6)
Listen to core and aggregation switches (s4, s5, s6)
Received a new packet
Add to hit count (s4, s5, s6)
False
hit count > TRESHOLD
True
Move rule to lower aggregation switch

Figure 6: Controller Algorithm

Our cache replacement policy is Least Recently Used (LRU), which is activated when a new rule is to be inserted into a cache, yet the cache is full.

5. Results

5.1. Eexperiments Parameters

Constant parameters during all experiments were the total cache size in the entire network, which is set to be 90, and the traffic generated into the system.

The traffic generated in each rack, in our emulation, a host, is generated in 40 packets per second (pps). Out of the total 60 destination IP addresses (flows) participating in the experiment, 10 are transmitted in 40pps, defined as large flows, 20 are transmitted in 20pps, defined as medium flows, and 10 are transmitted in 10pps, defined as small flows. The large and medium flows are shared IP destinations from all racks, meaning all racks send packets to those destinations. Small flows, on the other hand, are unique per rack, meaning each rack has a different set of destinations.

Other experiments parameters are changed throughout the different experiments to show our solution against a baseline network architecture. The cache distribution between the different switches in the different layers and the cache hit threshold bar vary through the different experiments.

To estimate our solution, we counted the number of cache hits in time intervals for each group of switches, divided according to the different layers. We then calculated the average number of hops a packet needs to reach a cache hit. Lowering the average number of hops in our cache mechanism model will prove our solution has improved the total throughput in the network.

5.2. Experiments Conducted

After performing several tests with different parameters, we chose to show the four most relevant experiments. The parameters set are presented in table 1 below:

Experiment Name	Cache Sizes (total: 90)			Threshold Bar	
	ToR Switches (3)	Agg. Switches (2)	Core Switch (1)	Agg. Switches	Core Switch
Baseline	15	15	15	10	20
Lower TH	15	15	15	7	3
Unequal Cache	10	20	20	7	3
Optimized	10	20	20	5	3

Table 1: Experiments Parameters

5.3. Experiments Results

In the next section, we present 4 sets of graphical results, followed by the calculation of our measurement parameters- the average number of hops. On the left of each set is the cache hit counts as a function of time. On the right, the bar graph is the percentage of the number of hops per flow type.

5.3.1. Experiment 1: Baseline

Even distributed cache between the switches in the network, and a high threshold bar.

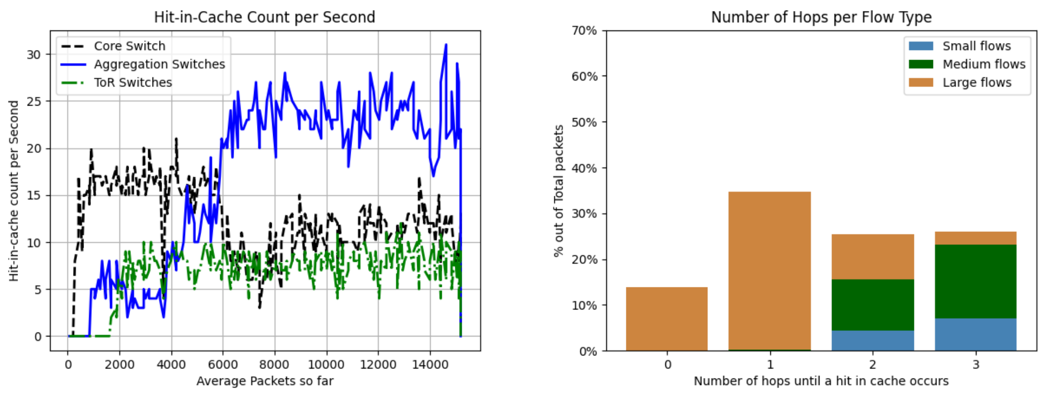


Figure 7: Baseline Results

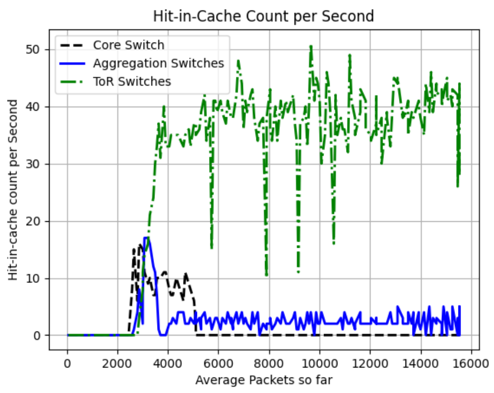
<u>The average number of hops in this experiment is 1.63</u>

In the first graphs set, we can see that this is not the graph we would like to see. The ToR switches get the lowest cache hit during the different time intervals.

We would like to see a bar graph of descending stairs when the large flows are captured in the ToR switches.

5.3.2. Experiment 2: Lower Threshold

Even distributed cache between the switches in the network, and a low threshold bar.



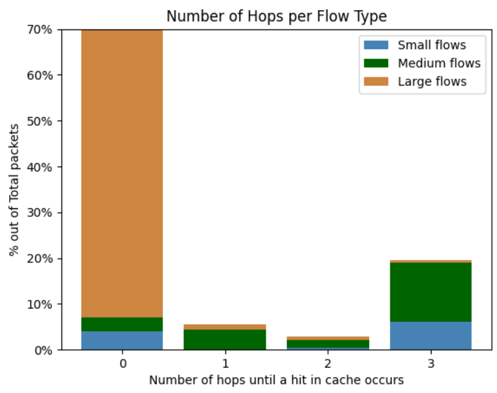


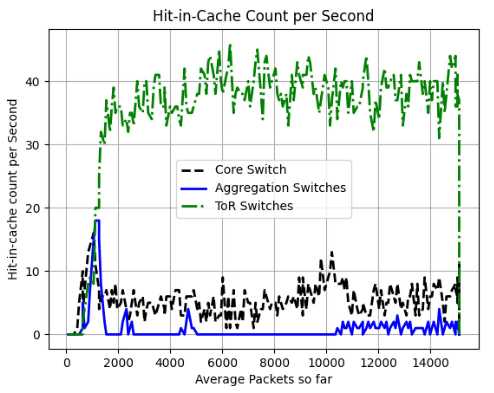
Figure 8: Lower Threshold Results

<u>The average number of hops in this experiment is 0.69</u>

From the results, we can see some improvement- almost all of the large flows get a cache hit in the ToR switches, but the bar graph is still not descending as we would like. For medium flows, we will need approximately, on average, 3 hops. The difference is mainly since we lowered the hit threshold.

5.3.3. Experiment 3: Unequal Cache

Unevenly distributed cache between the switches in the network, when ToR switches receive a lower portion of the total cache size in the network.



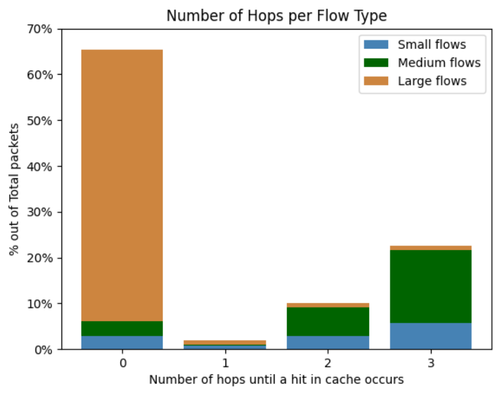
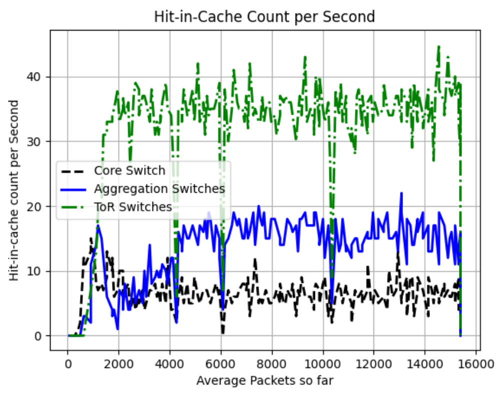


Figure 8: Lower Threshold Results

<u>The average number of hops in this experiment is 0.89</u>

These results are shown to prove that by changing only the distribution of the cache memory we don't improve much, validating our hypothesis for the optimized results.

5.3.4. Experiment 4: Optimized



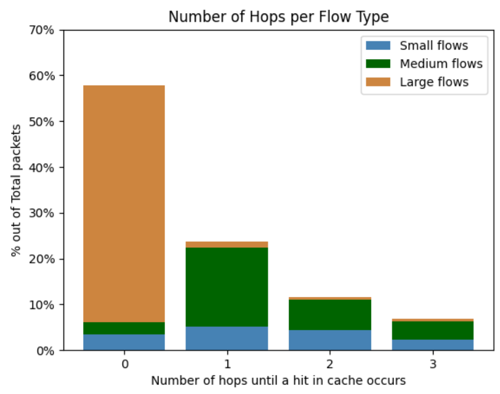


Figure 8: Lower Threshold Results

<u>The average number of hops in this experiment is 0.67</u>

This is the best scenario we have created so far. We reduced the size of the hit threshold in the Aggregation layer.

5.4. Summarizing the Results

Experiment Name	Cache Sizes (total: 90)			Threshold Bar		Average Number of Hops
	ToR Switches (3)	Agg. Switches (2)	Core Switch (1)	Agg. Switches	Core Switch	
Baseline	15	15	15	10	20	**1.63**
Lower TH	15	15	15	7	3	**0.69**
Unequal Cache	10	20	20	7	3	**0.89**
Optimized	10	20	20	5	3	**0.67**

Table 2: Experiments Parameters with Average Number of Hops

6. Challenges

6.1. Scalability

The ability to emulate a real DC network depends on the CPU of the machine running the emulation environment. Working with Mininet required working through a VM with 4 cores on a personal PC and therefore large-scale experiments were not possible in our case.

There is not currently a way to examine our model in a real-life DC network since we do not have access to one.

6.2. Implementation within the P4 framework

The P4 language contains programming challenges, it is a developing language and it defines code for all the switches together with the same prewritten program.

7. Conclusions

We showed that our model is visible, in our environment. The solution we proposed is not a trivial one, before starting the experiment we first needed to think about all the parameters properly to choose the right work environment.

In further research we would recommend changing the controller algorithm, to suit a better distributed algorithm in the network. Such distributed mechanism should be implemented in the BMv2 switches to manage their cache memory, or switches in their local environment.

8. Bibliography

[1] D. Y. Huang, K. Yocum and A. C. Snoeren, "**High-fidelity Switch Models for Software-Defined Network Emulation**" in HotSDN, Aug. 2013.

[2] S. Jiang, "**Efficient Caching Algorithms for Memory Management in Computer Systems**", College of William & Mary - Arts & Sciences, 2004.

[3] de Souza Dias, Daniel & Costa, Luís. (2012). **Online traffic-aware virtual machine placement in data center networks**. 1-8. 10.1109/GIIS.2012.6466665.

[4] "**Network Architectures: Collapsed Core and Three-Tier Architectures**" ad-net.com. https://www.ad-net.com.tw/network-architectures-collapsed-core-and-three-tier-architectures/ (accessed: June. 26, 2022).

[5] "**Comparing Two-Tier and Three-Tier Data Center Networks**" wwt.com. https://www.wwt.com/article/comparing-two-tier-three-tier-data-center-networks (accessed: June. 26, 2022).

[6] Rawas, Soha. "**Energy, network, and application-aware virtual machine placement model in SDN-enabled large scale cloud data centers.**" Multimedia Tools and Applications 80.10 (2021): 15541-15562.

[7] V. Ravikumar and R. Mahapatra, "**TCAM architecture for IP lookup using prefix properties**", EEE Micro, vol. 24, no. 2, pp. 60-69, March-April 2004.

[8] "**Mininet Overview**" mininet.org. http://mininet.org/overview (accessed: June. 26, 2022).

[9] "**P4Runtime-Spec**" p4.org. https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html (accessed: June. 26, 2022).

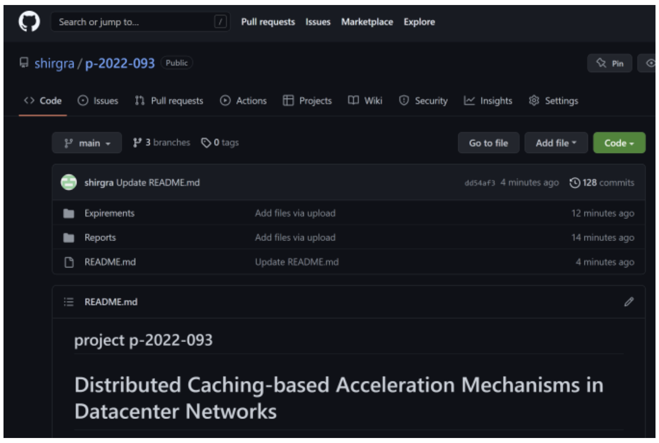
[10] "**GitHub: p4lang/behavioral-model**" github.com. https://github.com/p4lang/behavioral-model (accessed: June. 26, 2022).

9. Appendix:

9.1. Code Source

Our project is presented as a project on Github and public to the world:

https://github.com/shirgra/p-2022-093



shirgra / p-2022-093 Public

3 branches 0 tags

shirgra Update README.md dd54af3 4 minutes ago 128 commits

Expirements	Add files via upload	12 minutes ago
Reports	Add files via upload	14 minutes ago
README.md	Update README.md	4 minutes ago

README.md

project p-2022-093

Distributed Caching-based Acceleration Mechanisms in Datacenter Networks

In there, you will find all files connected to this project and a detailed guide mentioned here in section 9.2.

9.2. Reconstruction of Our Project (From Readme in our Github Project)

Environment Setup:

- Download P4 VM from here (Use your post.bgu account to gain access).
- If you don't have VirtualBox, install VirtualBox and run this VM.
- Clone this repo to the vm that you've just downloaded:

```
$ git clone https://github.com/shirgra/p-2022-093.git
```

- In the repository, p-2022-093/ open a terminal, run:

```
$ cd Experiments
$ chmod +x pip_install.sh
$ ./pip_install.sh
```

Experiment Setup:

- In the NetCache_2022 folder, open a terminal and run:

```
$ chmod +x build.sh
  $ ./build.sh
```

- Mininet CLI will come up, new open 4 objects CLI:

```
$ > xterm s0 h1 h2 h3
```

  - In s0 run:

```
  $ chmod +x switch_controller.py
  $ chmod +x host_traffic_generator.py
  $ ./switch_controller.py
```

  - Wait a few seconds, the in each host 1 2 or 3 run, for example in host 1:

```
$ ./host_traffic_generator.py 1
```

Note:

You can insert your own topology! create a new folder in src/ folder and run:

```
$ ./build.sh my_topology_folder
```

9.2. Final Report Evaluation

המלצת ציון (ע"י מנחה אקדמי) לדו"ח מסכם

אם יש צורך, לכל סטודנט/ית בנפרד

מספר הפרויקט:	p-2022-093		
שם הפרויקט:	מנגנוני האצה מבוזרים מבוססי מטמון ברשתות מרכזי נתונים.		
שם המנחים מהמחלקה:	פרופסור חן אבין, ד"ר גבריאל סקלוסוב.		
שם הסטודנטית:	אנה אקסלרוד	ת.ז.:	324682475
שם הסטודנטית:	שיר גרניט	ת.ז.:	205531445

%		חלש 55-64	בינוני 65-74	טוב 75-84	ט"מ 85-94	מצוין 95-100	
20	הצגת הגישה והתכנון ההנדסי						
20	הצגת התוצאות וניתוח השגיאות						
20	הסקת מסקנות						
10	גילוי יוזמה וחריצות						
20	פתרון בעיות, מקוריות ותרומה אישית (מעבר למילוי ההנחיות)						
10	עמידה בלו"ז ורמת הביצוע המעשי						

אם יש כוונה לפרסם/ יפורסם מאמר, שם כתב העת ומועד משוער להגשה:

ציין אם יש כוונה לשקול המלצה כפרויקט מצטיין:

הערות נוספות: