The Interdisciplinary Center, Herzliya
Efi Arazi School of Computer Science
M.Sc. program

GRAMI Software Enhancement on SDN Networks

by
Shlomi Nissim

Final project, submitted in partial fulfillment of the requirements
for the M.Sc. degree, School of Computer Science
The Interdisciplinary Center, Herzliya

January 2018

Acknowledgments

This work was carried out under the supervision of Prof. Anat Bremler-Bar from Efi Arazi School of Computer Science, the Interdisciplinary Center, Herzliya. I want to thank her for giving me the opportunity to work on such an interesting project, and the help to face the challenges I have encountered along the way.

I would also like to thank Mr. Alon Atari for his explanations and advices that were crucial for the success of this project.

Abstract

GRAMI, the Granular RTT Monitoring Infrastructure tackles the problem of monitoring round trip time and uses the ability of OpenFlow to control the routing. GRAMI, that was presented in [1] uses active probing from selected vantage points for efficient RTT monitoring of all the links and any round-trip path between any two switches in the network.

The previous implementation of GRAMI had couple of limitations, GRAMI was implemented on CPqD switch, a virtual switch that is not commonly used in the industry and GRAMI experiments ran on a simulated environment without comparing them to a real RTT measurement tool.

In this work, we first implemented GRAMI on the commonly used OpenVSwitch, which required us to do some adaptation of the original GRAMI algorithm. In addition, we implemented also round trip (RTP) measurement, in OpenVSwitch, that measures RTT between any two switches in the network. In the original paper RTP was designed but not implemented. GRAMI updated source code can be found in [2].

As second step in this work, GRAMI algorithm accuracy was confirmed by running several experiments in a real SDN network. The RTT results were compared with traceroute. Traceroute, is a common diagnostic tool for RTT monitoring for IP network, but it suffers from limitations, such that it can measure only some of the paths in the network, and only in IP level. This is as opposed to GRAMI that can measure any path in the network, and also works on L2 SDN network. The comparison to traceroute, in the scenario that traceroute can measured, shows that the results are very similar with deviation of maximum 0.1 milliseconds from the traceroute result.

Table of Contents

Acknowledgments 1
Abstract 2
Table of Contents 3
List of Figures 4
1 Introduction 5
2 Background 7
2.1 GRAMI Overview 7
2.2 GRAMI support on virtual switches 8
2.3 Previous software-defined network setup 10
2.4 RTP measurements 11
3 Implementation 12
3.1. OpenVSwitch Integration 12
3.1.1 Choosing tagging mechanism for GRAMI 12
3.1.2 GRAMI code modifications 13
3.2 Round trip path (RTP) 14
4 Evaluation 16
4.1 Mininet with OpenVSwitch setup 16
4.2 Using "traceroute" for comparison 18
4.3 Separate virtual machines setup 19
4.3.1 GRAMI vs. Traceroute without network load 21
4.3.2 GRAMI vs. Traceroute with network load 21
4.3.3 GRAMI vs. Traceroute overtime 22
4.4 Separate servers network setup 23
5 Summary and Conclusions 26
References 27
תקציר 28
תודות 29

List of Figures

Figure 1: Running GRAMI on OpenVSwitch on Mininet 17

Figure 2: Running GRAMI on multiple virtual machines without load..... 20

Figure 3: Running GRAMI on multiple virtual machines with a file transfer..... 20

Figure 4: GRAMI vs Traceroute - without any load..... 21

Figure 5: GRAMI vs Traceroute – with network load..... 22

Figure 6:GRAMI vs Traceroute – over time..... 22

Figure 7: GRAMI network setup on a real physical network..... 23

Figure 8: GRAMI vs Traceroute – running on separate hardware. 25

1 Introduction

Monitoring Round-Trip Time provides important insights for network troubleshooting and traffic engineering. The common monitoring technique is to actively send probe packets from selected vantage points (hosts or middleboxes). In traditional networks, the control over the network routing is limited, making it impossible to watch every selected path. The emerging concept of Software Defined Networking simplifies network control. However, OpenFlow, the common SDN protocol, does not support RTT monitoring as part of its specification.

GRAMI was designed to be resource efficient. It requires only four flow entries installed on every switch to enable RTT monitoring of all the links. For every round-trip path selected by the user, it requires a maximum of two additional flow entries installed on every switch along the measured path. Moreover, GRAMI uses a minimal number of probe packets, and does not require the involvement of the controller during online RTT monitoring.

One of the GRAMI implementation limitation in the original paper was that algorithm tested and implemented on a network emulated with Mininet [3] and based on CPqD OpenFlow [4] virtual switches controlled by a single Ryu controller [5]. But, this way of implementation had a major drawback, GRAMI could not run on a real SDN network, so all the results were simulated by one local machine.

Since the original paper GRAMI was implemented on CPqD virtual switches, and this virtual switch implementation is no longer supported, and is not popular among the SDN

users. SDN common users did not want to change their network to contain unsupported and old-fashioned virtual switches such as CPqD.

In this paper, we implemented GRAMI on OpenVSwitch, the most popular virtual switch software today makes it easier to integrate with GRAMI on existing networks that commonly used OpenVSwitch switches. GRAMI is tested in an environment that contains OpenVSwitch virtual switch with version was updated and now its supports QinQ, appending two VLAN headers, that was required in GRAMI for tagging data in the probe packets. This version was released in March 2017. Now, the source code was changed for fully support OpenVSwitch.

Moreover, the ability to calculate RTP was added, which is not available in other common RTT calculation tools. RTP was suggested in the previous article and was not implemented, but now RTP support added and tested successfully.

In the evaluation phase, we verify that GRAMI is a reliable tool for testing RTT between links on a real software-defined network. We created many network setups to examine various types of links: a virtual link within Mininet, a link between two separate VMs and a physical link between two separate servers. This selection of links simulates all the scenarios which a software-defined network can run. To ensure the reliability of GRAMI, we compared the experiment results to another RTT common diagnostic tool, traceroute. GRAMI successfully identified the loads and returned similar results to traceroute (by deviation of 0.1 milliseconds).

2 Background

2.1 GRAMI Overview

The previous paper presented GRAMI, the Granular RTT Monitoring Infrastructure. GRAMI uses active probing from selected vantage points for efficient RTT monitoring of all the links and any round-trip path between any two virtual switches in a software defined network. Then, it ran on a network emulated with Mininet and based on CPqD OpenFlow virtual switches.

In a classic SDN scenario, rules for packet handling are sent to the switch from a controller, an application running on a server somewhere, and switches (aka data plane devices) query the controller for guidance as needed, and provide it with information about traffic they are handling. Controllers and switches communicate via a controller's "south bound" interface, usually OpenFlow, although other protocols exist. OpenFlow does not support RTT monitoring as part of its specification. We use GRAMI for adding support of RTT monitoring using the OpenFlow protocol.

GRAMI was designed to be resource efficient. It requires only four flow entries installed on every switch in order to enable RTT monitoring of all the links. For every round-trip path selected by the user, it requires a maximum of two additional flow entries installed on every switch along the measured path. Moreover, GRAMI uses a minimal number of probe packets, and does not require the involvement of the controller during online RTT monitoring

GRAMI is composed of two phases, an offline phase and an online phase. In the offline phase, an application installed on the controller builds a single overlay network and installs its corresponding flow entries, which define the routing for the probe packets.

The overlay network enables monitoring of all the links in the network and all the RTPs selected by the user.

In the online phase the measurement points (MPs) repeatedly send probe packets. The probe packets are distributed over the overlay network to every switch. Along the way, the switches use tagging in order to identify the path traversed by each probe packet and the path it should traverse. Each switch sends the probe packet in return to the MPs. These packets contain two adjacent virtual switches identifiers. The MPs extract the path from the tagged data and by computing the receive time difference these packets, the links RTT time is calculated.

In the original paper, GRAMI code was ran on RYU SDN Framework. The application selects unique IDs for the selected RTPs, unique IDs for the switches, and a NULL ID to indicate an empty ID value. Note that probe packets with RTP information contain two IDs; the RTP ID and the first switch ID. Probe packets with link information contain two IDs as well; those of the switches at the link's endpoints. Thus, GRAMI uses two fields of IDs; (ID1; ID2), to enable tagging of RTPs or links according to the *RTPFlag*. Since the probe packets are created in the MPs and used only for RTT monitoring, they can be independent for a specific protocol. Therefore, GRAMI can add any payload, and select any field for tagging, if the OpenFlow version supports tagging and matching for that field.

2.2 GRAMI support on virtual switches

Virtual switch is a software layer that resides in a server that is hosting virtual machines (VMs). VMs, and now also containers, such as Docker, have logical or virtual Ethernet ports. These logical ports connect to a virtual switch.

There are three popular virtual switches: VMware virtual switch (standard & distributed), Cisco Nexus 1000V, and OpenVSwitch (OVS). OVS was intended to meet the needs of

the open source community, since there was no a feature-rich virtual switch offering designed for Linux-based hypervisors. OpenVSwitch is meant to be controlled and managed by third party controllers and managers. OVS is critical to many SDN deployments in data centers because it ties together all the virtual machines (VMs) within a hypervisor instance on a server. In the previous article, the implementation did not support OVS, to get GRAMI evaluation be real as possible, GRAMI integration with OVS is a main task of this project.

For running GRAMI on a real SDN network, GRAMI source code needs to be changed to support OpenVSwitch tool. OpenVSwitch is the most popular implementation of virtual switches, have stable releases and update support. Afterwards, running GRAMI on a real SDN network is a simple task because OpenVSwitch is wildly supported in the industry. Also, companies and research institutes could use GRAMI algorithm as well. To sum up, enhancing GRAMI to work on OpenVSwitch, cause it to be a flexible and compatible tool for computing RTT between virtual switches.

The probe packets tagging fields were *ethernetType* (16 bits) and two VLAN IDs (12 bits each). IEEE 802.1ad (QinQ tunneling) [6] is ethernet networking technique to tag two or more VLAN headers in packets. OpenVSwitch did not support QinQ, therefore the previous work cannot use two VLAN headers and is not compatible for implementation on this kind of virtual switch. Moreover, the. The *DirectionFlag* and the *SetIDFlag* were encoded by four different *ethernetType* values that are not correlated with any protocol. The *ParentFlag*, *RTPFlag* and ID1 were encoded in one VLAN header. ID2 was encoded in the other VLAN. In ID1, 10 bits were used for switch ID or the NULL ID. In ID2, 12 bits for switch ID, RTP ID or the NULL ID. Hence, GRAMI implementation is limited to (2^10-1) = 1023 switches and (2^12-1) = 4095 RTPs, but choosing other fields for tagging is possible for bigger networks.

2.3 Previous software-defined network setup

Software-defined networking (SDN) is a term encompassing several kinds of network technology aimed at making the network as agile and flexible as the virtualized server and storage infrastructure of the modern data center. The goal of SDN is to allow network engineers and administrators to respond quickly to changing business requirements. In a software-defined network, a network administrator can control the traffic forwarding, from a centralized control console without having to touch individual switches and can deliver services to wherever they are needed in the network, without regard to what specific devices a server or other device is connected to.

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking.

Unlike a simulator, Mininet doesn't have a strong notion of virtual time; this means that timing measurements are based on real time, and that faster than real time results (e.g. 100 Gbps networks) cannot easily be emulated. In the previous paper, GRAMI tests were running on Mininet. Hence, all the virtual switches and links were implemented on a single machine with shared computational resources. The final results were simulated as well and did not demonstrate RTT of a real SDN network.

One this project tasks is to run some tests on real software-defined network. The last work ran on Mininet and the tests were run on one single virtual machine with shared computational resources. Mininet is a virtual environment not suitable for measuring time or performance accurately. However, it was a proof of concept and gave us a sense on the impact of different network parameters. In the previous article, we tried to estimate

GRAMI's overhead in Mininet followed by an assessment of how different network parameters might affect the accuracy of the RTT measurements.

2.4 RTP measurements

Round trip path (RTP) time between any two switches in the network was not implemented in the previous work. One of GRAMI features is to calculate packet preconfigured RTP time, the algorithm is described in the article. This task provides important insights for network troubleshooting.

When a probe packet is received at a switch, it triggers the measurement of every egress link of s according to the overlay network, and of all the preconfigured RTPs that start at that switch. The RTP algorithm in GRAMI is well explained in the previous article. This is a programming task, by adding this feature, GRAMI algorithm have a great advantage compared to other algorithms and tools that do not contain this feature.

3 Implementation

3.1. OpenVSwitch Integration

One of the project tasks is to find a way to implement GRAMI on OpenVSwitch. OpenVSwitch did not support QinQ tagging, although that RYU controller is already supports it. First, we will describe the number of dilemmas to overcome it. Then we will explain what adaptation in the algorithm code we made for this integration.

3.1.1 Choosing tagging mechanism for GRAMI

We consider the following options for tagging in GRAMI:

A. Using different packet fields for tagging GRAMI data

We consider replacing other fields in the probe packet that OpenFlow supports. In that way, the computation overhead time for pushing and popping VLAN headers will be saved and we will use minimal bytes in our probe packets. We proposed that we use in the *ethernetType* for GRAMI protocol flags, *sourceMAC* for the first switch ID and *destinationMAC* for the second switch ID. This way is much faster and can support many more switches. The disadvantage of using MAC fields for tagging is possible collision of MAC addresses that might cause network malfunction. Another disadvantage that is that there is no history or research justification for using these fields for tagging data.

Later, we found an article, that uses tagging data in the *sourceMAC* address [7]. Since MAC address has enough bits (6 bytes= 6*8 bits), it can contain all GRAMI data (30 bits).

We used to tag in parts of MAC header for implementing GRAMI on OpenVSwitch. We found that, In the latest version of OpenFlow 1.5[8] supports the OpenFlow "set partial field" action. RYU and OpenVSwitch supports this feature. The problem was, that OpenVSwitch does not implement *OFPPacketOut* action (the controller supposed to use

this message to send a packet out through the switch, which is critical for virtual switch functionality) for OpenFlow 1.5. The outcome was that we cannot use this way of implementation.

B. Using P4 switch instead of OpenVSwitch

Another option is using P4, in the P4 language (also referred as" OpenFlow 2.0 API") [9], the user can define specific headers for tagging, and only set these headers to tag the packet. Implementing GRAMI with the P4 language should thus significantly reduce the overhead caused by the tagging mechanism. The problem is P4 not popular enough because it is uneasy to install on virtual switches.

C. Patching OpenVSwitch source code.

The final option and the most difficulty one way to patching OpenVSwitch source code to support QinQ tagging. Because OpenVSwitch is an open source project, after some research, we found mail discussion about developing QinQ tunneling support, they said that the first patch stable release was available in the beginning of 2017 [10], in March 2017 the patch was released. The integration was successful and GRAMI run in OpenVSwitch so that option was chosen.

3.1.2 GRAMI code modifications

After months of reading mail discussions, the support patch for QinQ was committed to OpenVSwitch GitHub. GRAMI uses the VLAN headers for saving two VLAN IDs, OpenVSwitch did not support double-tagging of VLAN headers - QinQ. We used this new version of OpenVSwitch under Mininet because the setup was under the same machine and it was easier to debug problems. The virtual switch could push two VLAN headers, which has not been possible in OpenVSwitch until now. Clearly, this was a breakthrough for the project.

Moreover, one of the OpenVSwitch integration problems was that run *OFPActionSetField* on the *ethernetType* field was not allowed, it was used to specify the probe packet type. By looking at the OpenVSwitch source code, it was confirmed that this field was blocked for changes. Previously, this field GRAMI's usage was to specify the direction and type of probe packet, so this field is necessary field to make GRAMI work.

One of the possible solutions was to use the VLAN's priority field. This field has been unused in the implementation of GRAMI until now. It is not commonly used for specifying the packet type. Eventually, we ran a few tests and the result was that the Linux driver was resetting this field, therefore we lose the type of the probe packet, so to sum up this field cannot be used.

At last, we found an article which uses *sourceMac* field for saving the protocol type. This field can be used because GRAMI uses Layer 2 only, and this field is unused too. The controller can decide to modify existing flow rules on one or more switches or add new rules, hence the virtual switches routing is determined by the controller. Accordingly, we use this field for specifying the probe packet type and direction.

As a result, GRAMI now can route packets by using *sourceMac* field. There are 4 types of probe packets, which means 4 MACs, so the chance of MAC collision is negligible. Furthermore, we are able to distinguish between regular packets and GRAMI packets more easily.

3.2 Round trip path (RTP)

First, for the RTP implementation, we use the algorithm that was suggested in the GRAMI article. The algorithm describes how to apply flaw rules for measuring RTP on the virtual switches. We assign to each RTP an ID, so it can match the packet RTP flaw rule according to the ID. When a probe packet arrives to a switch, and RTP starts in that

switch, it sends additional packet in the RTP path with *RTPType*, the switch passes the packet according to ID matching rules in the network topology according to the predefined route. Eventually, the RTP probe packet returns to the virtual switch that the path was started, the type becomes *ReturnNoTag* and the packet returns to the measurement point. By calculating the difference between the switch RTT and the time that the RTP packet arrives to the MP, we get the RTP time.

In addition, we add a feature that the MP could add RTP by sending a packet to the controller. Moreover, The MP can process and parse the RTP packets that arrives. Now, the RTP calculation results are simply shown in the MPs. In conclusion, after several tests we approve that the implementation worked, and we successfully added RTP time calculation support to GRAMI.

4 Evaluation

GRAMI was tested on some various software defined networks, in every setup the virtual switches were OpenVSwitch. The first setup is installing Mininet on a single virtual machine, the second is a couple virtual machines on a personal computer and the third one is by using separate physical servers.

In all setups, each machine that used run Ubuntu Server. OpenVSwitch software is installed on these endpoints. The links between the machines are emulated by internal virtual networks between the VMs or a ethernet cable. In addition, there are two hosts that create load on the switches for the tests. One machine for the controller and one for the measurement point. The main limitation using a single virtual machine is that heavy load on the single CPU created, which corrupt the test results.

Furthermore, Deepness Lab [11] servers used for the testing. By using this configuration, we receive fast and reliable results. However, there are not so many available Ethernet ports on these servers and only one switch is connecting between them. For GRAMI virtual switches link RTT measuring we must have at least two virtual switches. As a result, we configure the network and connect the servers physically.

4.1 Mininet with OpenVSwitch setup

After the integration with OpenVSwitch worked, the setup workspace had been upgraded, the operating system upgraded to Ubuntu 16.04, and we upgrade to OpenVSwitch 2.7.9 and to Mininet 2.3.0 which support the features that implemented. By using Mininet, it was easy to set up a network topology for tests. We choose the network topology from GRAMI presentation (figure 1), the advantages of this network topology are that it challenges the GRAMI algorithm. We added 10ms delay before

sending any packet in each switch port also added, in order to get results that could be verified.

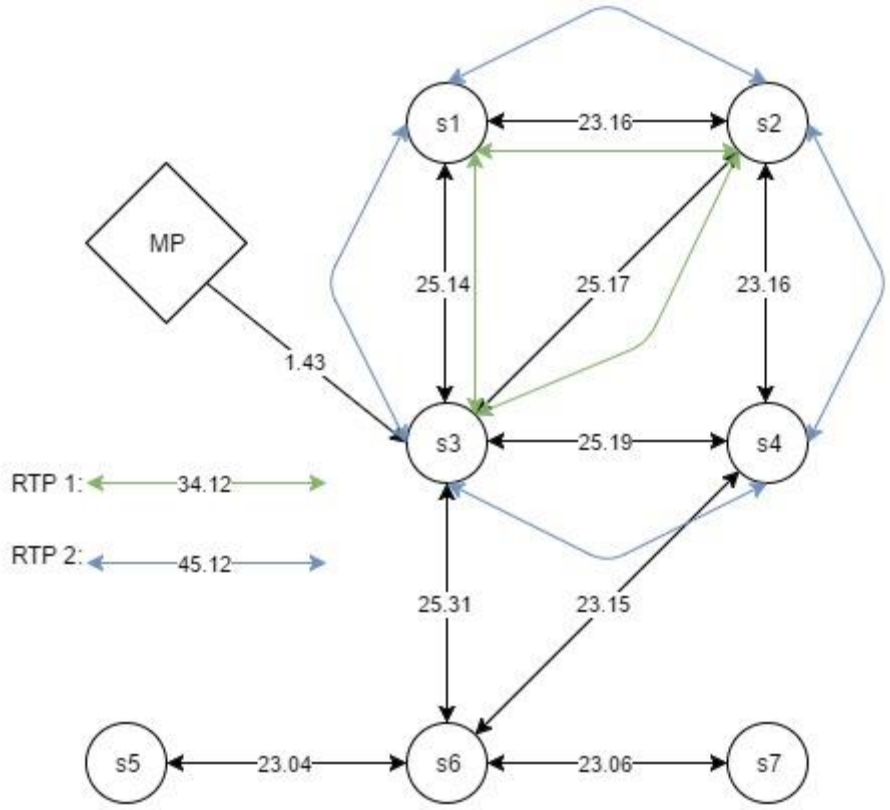

Figure 1: Running GRAMI on OpenVSwitch on Mininet

Finally, GRAMI ran on this topology. The result was that GRAMI supports OpenVSwitch under the Mininet virtual environment. The pre-configured RTP worked and returned reasonable results and the links RTT were calculated correctly as well. We received a significant overhead of 4ms in each link, in addition to the delay we added. This occurred because we ran under one machine with one processor, by tagging VLAN data and the processing time of the packets in the switches, the overhead increases due to the virtual processor load. The next step is to run GRAMI not in Mininet environment where we can get more reliable results.

4.2 Using "traceroute" for comparison

Traceroute is a computer network diagnostic tool for displaying the route (path) and measuring transit delays of packets across an Internet Protocol (IP) network. The history of the route is recorded as the round-trip times of the packets received from each successive host (remote node) in the route (path). The sum of the mean times in each hop is a measure of the total time spent to establish the connection. Traceroute proceeds unless all (three) sent packets are lost more than twice, then the connection is lost, and the route cannot be evaluated. Ping, on the other hand, only computes the final round-trip times from the destination point.

Moreover, OpenFlow switches do not have an IP address in their datapath. As a result, tools like Ping and Traceroute are not suitable for monitoring paths between two switches in the network. GRAMI measures the round-trip time between virtual switches. We need a network diagnostic tool to compare with GRAMI, and it seems that traceroute might do the work.

This tool should have the ability to calculate RTT between virtual switches. Since there is no public tool that calculates links RTT between virtual switches in SDN, we chose to use traceroute, which is commonly used to measure RTT. The disadvantage of this tool is that it is running on routers instead of virtual switches. In hence, at the same setup of tests we used, the endpoints are routers instead of switches, with the identical link conditions.

Traceroute tracks the route packets taken from an IP network on their way to a given host. It utilizes the IP protocol's time to live (TTL) field and attempts to elicit an ICMP TIME_EXCEEDED response from each endpoint along the path to the host. We get link RTT between endpoints by subtracting adjacent endpoint RTT.

The change to routers has forced us to migrate routing rules using the route command. Each endpoint port (NIC) has IP configured with its own subnet. For two linked

endpoints, we set different IPs but the same subnet, so they can communicate with each other. For two unlinked points, some routing manual packet forward rules was defined. Therefore, the router knows how to send the packet in a port that would reach its destination, just like a real Internet network. Furthermore, we succeed to do it by applying the IP forwarding option at the endpoints linux machines.

4.3 Separate virtual machines setup

We created multiple virtual (VM) machines in a personal computer using VMware software, the VMs are running Ubuntu Server 16.04.2 and OpenVSwitch v2.7.90, these are the most updated versions of the date of the tests. The links between the machines are configured by network adapters that are on the same virtual network.

The network contains the following Components: four switches: S1, S2, S3 and S4. Each switch run on separate VM, a controller and a MP. The controller and the other switches are connected to the management network. The network links are described in the results figure.

We had a couple of difficulties during the network creation. First, was running 6 VMs on a personal computer. The solution was by using Ubuntu Server on a virtual machine with minimum requirements, with four processors computer was enough. Second, the operating system, Ubuntu, by default doesn't support VLAN headers, so we added VLAN support by using 8021q kernel built-in module. We created multiple internal networks to simulate internal links in the host operating system. Moreover, we implement the automation to run multiple tests on multiple VMs, so we can run our tests more easily.

We ran two tests; the first test was without load on the network, and the other was with load. The results that are shown in figure 2 and 3, were the average between 100 measurement rounds, with one second delay between them. The load caused by file transferring between hosts h1 to h2, with maximum speed.

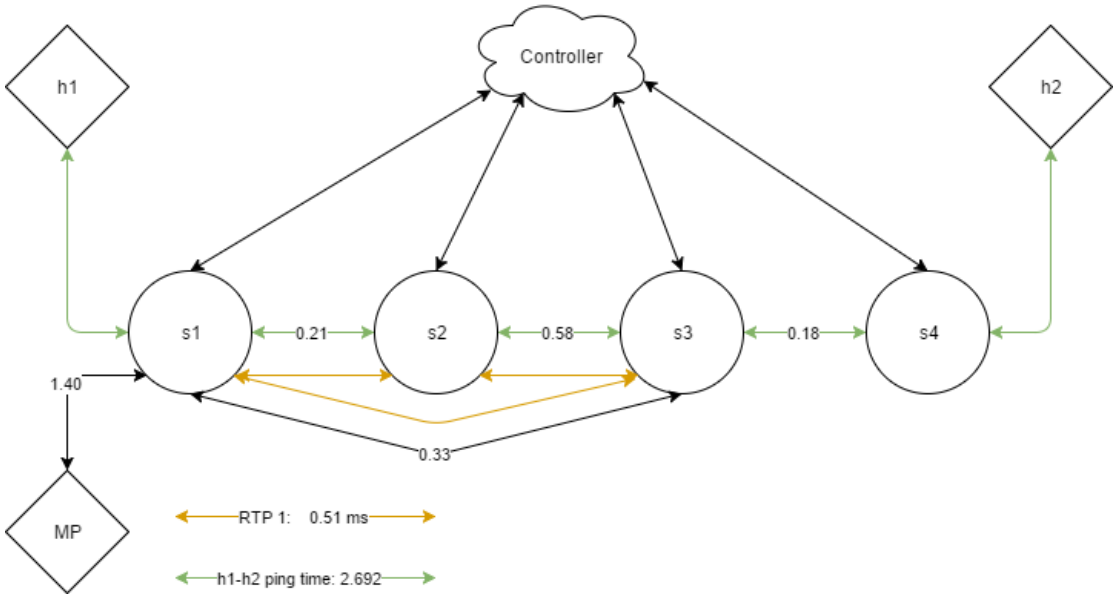

Figure 2: Running GRAMI on multiple virtual machines without load

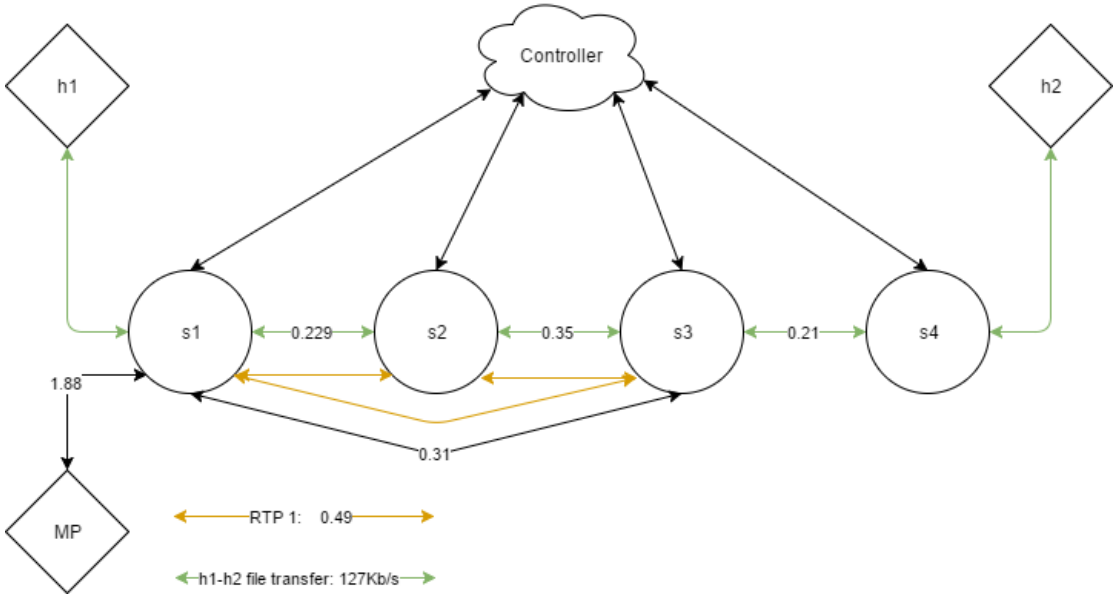

Figure 3: Running GRAMI on multiple virtual machines with a file transfer

The main achievement was that GRAMI works on a real network. GRAMI overhead is much lower on a real network than running on a single machine with Mininet. Furthermore, GRAMI algorithm does not damage the network bandwidth rates because

GRAMI send a little quantity of packets per second. Although the network load, the link RTT results stayed the same, because GRAMI packet flaw rules have higher priority.

4.3.1 GRAMI vs. Traceroute without network load

The tests purpose was to validate GRAMI virtual switches RTT results versus traceroute. Each test ran on virtual switches and calculated by GRAMI, then ran on routers using traceroute. The tests ran on the same VMs with the same conditions. We add 10ms delay between the endpoints for more stable results. If we did not do it, the result was unstable because of packet queuing and as a result some of the probe packets received in the same time. The test results are shown in figure 4.

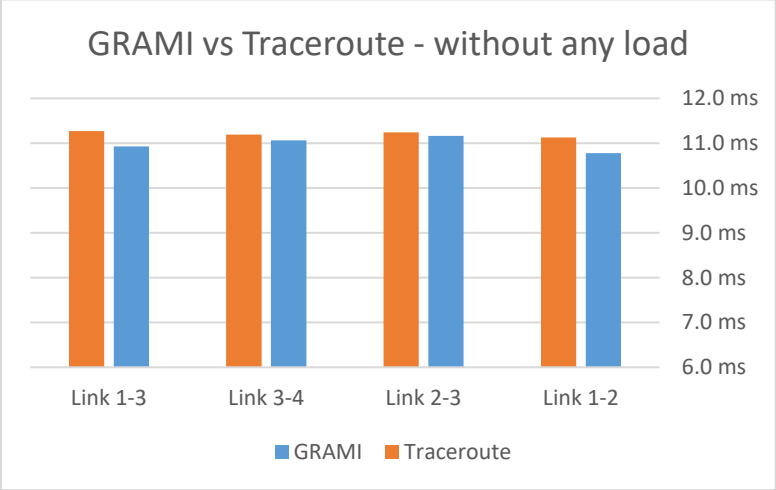

Figure 4: GRAMI vs Traceroute - without any load

4.3.2 GRAMI vs. Traceroute with network load

After some research, we understand that simulating the heavy load on links is possible by setting a bandwidth limit on the network interfaces. When we use VM configuration to limit the NIC bandwidth, then we see that the RTT time was increased significantly by simulating heavy traffic. Network load was simulated by transferring a file, at our network setup, the link bandwidth between s2 to s3 was configured to 1Mb/s. From figure 5 we can se that GRAMI identify the results successfully as traceroute.

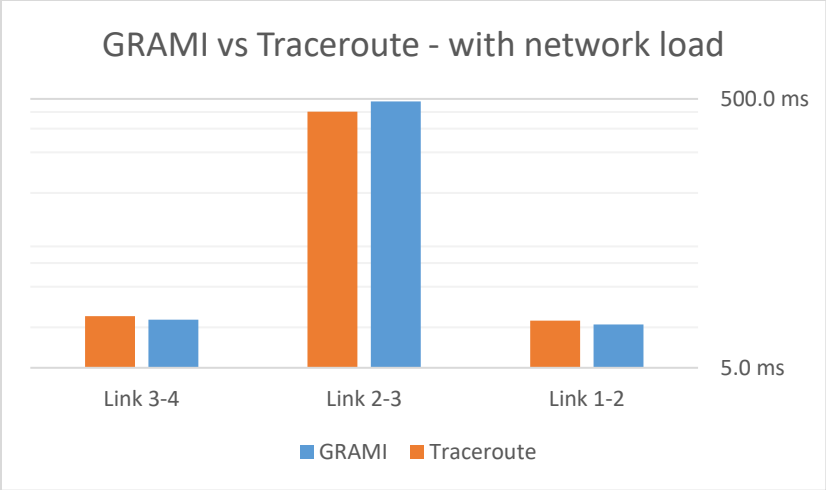

Figure 5: GRAMI vs Traceroute – with network load

4.3.3 GRAMI vs. Traceroute overtime

In this test, we sent a probe packet every second for 100 seconds and focused on a single link. At the middle of the test, we added a network load. The goal was that GRAMI could identify changes over time. Afterwards, we ran this test with traceroute with the same conditions and timing. The conclusion from figure 6 is that GRAMI and traceroute have indeed identified the change in load at the appropriate time. We see that GRAMI is faster than traceroute. The explanation for that is that traceroute works in IP layer which require much more work from the CPU than GRAMI, that works with simple packets by known route in the second layer.

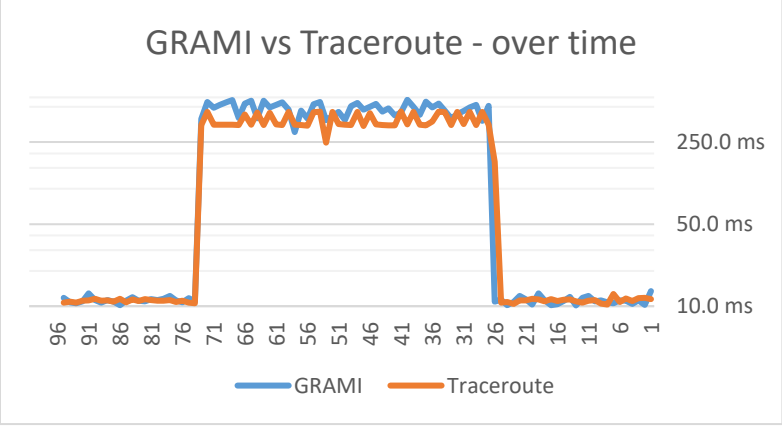

Figure 6:GRAMI vs Traceroute – over time

4.4 Separate servers network setup

We ran the tests on Deepness Lab servers. In the lab setup, there are three serves connected to each other by 10G optical cables. In addition, all the servers in the lab are connected to a management network, so they can be remote controlled by the Internet. There is an additional server that is not connected to any other server, but it is connected to the management computer too. We need to use this network setup to create the SDN topology with a controller, so we can run GRAMI tests on it. Figure 7 describes Deepness Lab Network Scheme. The challenges would be described at following section.

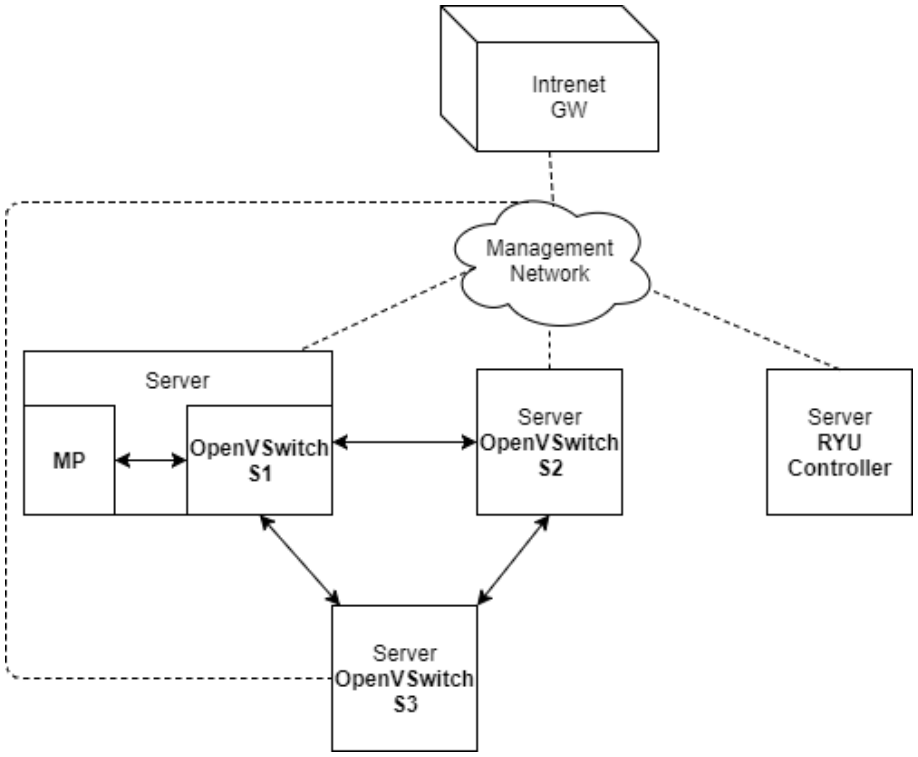

Figure 7: GRAMI network setup on a real physical network

1. The virtual switches and the controller placement.

There are three servers connected by 10G optical cable to each other. Within each server, we installed OpenVSwitch. Two of the network adapters of the server configured as ports of the switch. These ports are connected to the ports of the remaining switches. The advantage of this setup is that we can also measure RTP between these three switches.

The management network is on the same subnet, so we can use the fourth computer as RYU controller by using TCP connections for distribute forwarding rules to the switches. In conclusion, the controller and a virtual switch are installed in separate hardware which is the purpose of this test.

2. There are not enough computers to create MP.

We created a virtual link in one of the servers, the link connected to the virtual switch and the other end used to monitor GRAMI's management packets. In this scenario, the switch and the measurement point run on the same hardware, but that link is not relevant to our calculations, because only the RTT between the switches interest us.

3. Generate network load for GRAMI tests.

In the previous setup it was possible to limit the bandwidth of a network card by configuring the VM and create network load by transferring a file. Now the network cards are the hardware of the servers, and we can't change their bandwidth limit. We tried using existing tools such as "tc" or "wondershaper", but these tools only limit the network connection, and the GRAMI measurement packets is not even IP packets. Therefore, no matter how much we load the network card, we can't simulate a RTT significant changes. For test verification, we added 10ms delay on the interfaces.

In figure 8 we can see this test results, the test ran on the setups that described in figure 7. Moreover, RTP calculation time between S1 S2 and S3 virtual switches is: 30.41ms, the traceroute result for this test, which calculated manually is 30.91ms.

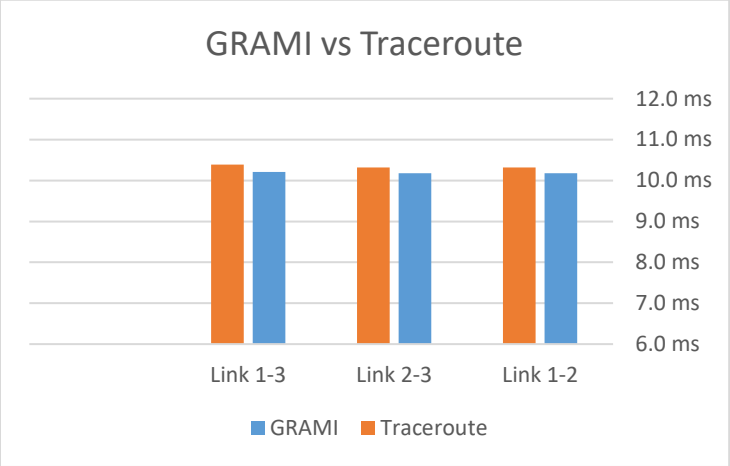

Figure 8: GRAMI vs Traceroute – running on separate hardware.

5 Summary and Conclusions

In this project, we learned a lot about SDN and its implementation methods. At first, we set up a Mininet network on a single computer and ran several GRAMI tests on it. Then, in the same setup, we did an integration with OpenVSwitch so that we could run GRAMI on more generic networks. At this point, we added support for the RTP calculation that was not implemented during the previous article. We have uploaded GRAMI updated source code to GitHub so that anyone who wants can use GRAMI or learn about SDN implementation method from it.

The main goal was to enhance GRAMI on various network setups. In these tests, we compared GRAMI to traceroute, one of the most common tools in the networking world, to verify the accuracy of the results. GRAMI was run on some virtual machines on a personal computer. In this setup, the loads between links was generated and increased RTT times. Finally, GRAMI have been tested on a network of servers, each server running on separate hardware.

There are several conclusions from our work. First, GRAMI works on the various setups that we have mentioned so far. Second, we verified the test results against traceroute tool and verified that the results we were getting were indeed reasonable. We succeeded in generating changes in the network load during the tests, and we verified that GRAMI can identify these changes. The bottom line, we confirmed that GRAMI does indeed correctly calculate the RTT links between virtual switches, and that GRAMI is a reliable tool for performing such tests.

References

[1] Alon Atari, Anat Bremler-Barr, "Efficient Round-Trip Time Monitoring in OpenFlow Networks", in INFOCOM: http://www.deepness-lab.org/pubs/infocom2016_grami.pdf

[2] GRAMI source code GitHub: https://github.com/alonatari1/GRAMI

[3] Mininet: http://mininet.org/overview/

[4] OpenFlow 1.3 Software Switch by CPqD: http://cpqd.github.io/ofsoftswitch13/

[5] Ryu Controller: https://github.com/osrg/ryu

[6] OpenVSwitch: Overview of 802.1ad (QinQ) Support: https://developers.redhat.com/blog/2017/06/06/open-vswitch-overview-of-802-1ad-qinq-support/

[7] Be Fast, Cheap and in Control with SwitchKV by Xiaozhou Li, Raghav Sethi, Michael Kaminsky, David G. Andersen, Michael J. Freedman : https://www.usenix.org/node/194905

[8] OpenFlow 1.5 Switch Specification: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

[9] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese et al., “P4: Programming protocol-independent packet processors,” ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, pp. 87–95, 2014

[10] OpenFlow Support in OpenVSwitch: https://github.com/openvswitch/ovs/blob/master/Documentation/topics/openflow.rst

[11] Deepness Lab Wiki [licensed for non-commercial use only]: http://deepness.pbworks.com/w/page/84759187/Welcome!

תקציר

אלגוריתם גראמי, הוא אלגוריתם המתמודד עם הבעיה של מדידת זמני שליחת חבילות הלוך חזור מרגע יציאתן עד חזרתן המשתמש בOpenFlow כדי לשלוט בחוקי הניתוב של הרשת. גראמי, אשר הוצג ב[1] עובד בצורה פעילה ברשת, על ידי שליחת חבילות ניטור ממספר נקודות קבועות ברשת למדידה של זמני שליחת חבילה הלוך וחזור בין כל שני מתגים ברשת.

המימוש הקודם של גראמי היה בעל מספר חסרונות. גראמי מומש על מתג שתוכנתו הייתה CPqD, שזה מתג תוכנתי שלא נפוץ בתעשייה ובנוסף, הבדיקות של גראמי רצו בסביבה מדומה מבלי השוואה לכלי ניטור רשת אחרים.

בעבודה זו, בשלב הראשון, מימשנו את גראמי על מתג תוכנתי הנפוץ ביותר, OpenVSwitch ושינוי זה הצריך מאתנו לעשות מספר התאמות לאלגוריתם גראמי המקורי. בנוסף, הוספנו מימוש לחישוב זמן של דרך מוגדרת מראש ברשת, המודד זמן בין כל שני מתגים תוכניתיים ברשת. במאמר המקורי, האלגוריתם לכך נכתב בעבודה הקודמת אך לא מומש. קוד המקור המעודכן נמצא ב[2].

בשלב השני של העבודה, הוכחנו את הנכונות של אלגוריתם גראמי על ידי הרצת מספר ניסויים במספר רשתות מבוססות תוכנה שונות. התוצאות שהתקבלו, הושוו לכלי אחר, traceroute, כלי הנפוץ לשם ניטור רשתות IP בלבד. בניגוד לגראמי, שזהו כלי היכול למדוד זמן מרגע שליחת חבילה עד לחזרתה בין כל זוג מתגים וירטואליים ברשת על ידי חבילות הנשלחות בשכבת הרשת השנייה. ההשוואה לכלי אחר, בתרחיש בו אנו יכולים להשתמש בו, הוכיחה שהתוצאות של גראמי היו זהות ובסטייה של עד 0.1 מילישניות בין שתי הבדיקות.

תודות

עבודה זו בוצעה בהדרכתו של פרופ' ענת ברמלר-בר מבי"ס אפי ארזי למדעי המחשב, המרכז הבינתחומי, הרצליה. בהזדמנות זו, אני רוצה להודות לה שהסכימה לעבוד איתי על הפרויקט זה, הנחייתה תרמה מאוד להכנת הפרויקט ועם התמודדות באתגרים שנתקלתי לאורך הדרך.

בנוסף, אני רוצה להודות לאלון אטרי, שכתב יחד עם פרופ' ענת ברמלר-בר את המאמר הקודם בנושא, על הסבלנות והעזרה שלו לאורך כל הפרויקט.

המרכז הבינתחומי בהרצליה

בית-ספר אפי ארזי למדעי המחשב

התכנית לתואר שני (M.Sc.)

הרחבת התוכנה של גראמי על רשתות מבוססת תוכנה

מאת

שלומי ניסים

פרויקט גמר, מוגש כחלק מהדרישות לשם קבלת תואר מוסמך M.Sc.,
בית ספר אפי ארזי למדעי המחשב, המרכז הבינתחומי הרצליה

ינואר 2018