The Interdisciplinary Center, Herzlia
Efi Arazi School of Computer Science
M.Sc. program - Research Track

Efficient Round-Trip Time Monitoring in OpenFlow Networks

by

Alon Atary

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

January 2016

This work was carried out under the supervision of Prof. Anat Bremler-Barr from the Efi Arazi School of Computer Science, The Interdisciplinary Center, Herzliya.

Acknowledgments

This research was supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC Grant agreement no. 259085, and by the Neptune Consortium, administered by the Office of the Chief Scientist of the Israeli Ministry of Industry, Trade, and Labor.

Abstract

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 monitor 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. In this work, we leverage the ability of OpenFlow to control the routing, and present 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 switches* in the network.

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.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Round Trip Time, the time required to send a packet towards a specific destination and receive a response, is frequently used as a metric for network performance assessment. The common technique for RTT monitoring is to send probe packets from vantage points in the network and monitor their RTT. In order to monitor a specific path, control over the routing is required. However, in traditional networks, the routing is determined by traditional routing protocols; thus, monitoring every path in the network is practically impossible. Moreover, the monitored paths can change due to the dynamic nature of the traditional routing protocols, creating unstable paths and inconsistent RTT measurements [1, 2].

In recent years, SDN networks are becoming more common, and promise easier control over the network. However, OpenFlow, the common SDN protocol, does not provide any support of RTT measurements as part of its specification [3]. 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. In this work, we present GRAMI, an infrastructure that leverages the abilities of OpenFlow to fix paths in the network and duplicate packets within the switches for two purposes. First, it enables RTT monitoring of *any round-trip path* (RTP) *between any two switches* in the network, i.e., any path that starts in switch s_i, leads to switch s_j, and returns to s_i, not necessarily in a symmetric path. Consequently, GRAMI can provide useful information for assessing the quality of different routing policies. Second, it efficiently monitors the RTT of *all the links in the network*. Thereby, GRAMI enables better anomaly detection and bottleneck identification.

To conduct the monitoring, a monitoring application is installed on hosts at preselected vantage points, turning them into monitoring points (MPs). The MPs send probe packets and monitor their RTT. A single MP is capable of monitoring the entire network, but using multiple MPs can reduce the number of links monitored by each MP, and improve the accuracy of the measurements.

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 order that every link be measured *exactly once*, the overlay network is composed of DAGs with the MPs as their starting vertices. As the number of links between the MP and the switch/link (i.e., the depth of the switch/link) increases, there is more likely to be noise in the monitored RTT. Hence, the path from every switch and link to its closest MP in the overlay network is the shortest possible. Whenever a dynamic change in the network occurs, the controller application automatically recalculates the overlay network.

In the online phase the MPs repeatedly send probe packets. The probe packets are distributed over the overlay network to every switch, using the shortest path. Along the way, the switches use tagging in order to identify the path traversed by each probe packet and the path it should traverse. When a probe packet is received at a switch s, it triggers the measurement of every egress link of s according to the overlay network, and of all the preconfigured RTPs that start at s. The probe packet is duplicated and tagged, and one tagged clone is sent back to its original MP on the same shortest path of the overlay network. The other tagged clones are sent to the egress links and to the preconfigured RTPs. The duplication capability of the switches reduce the load caused by probe packets since only one probe packet is sent from each MP. The duplication also increases the accuracy because the path is partly shared by both probe packets.

The RTT of specific a RTP P that starts at s_p is estimated as the difference between the RTT of two probe packets: (1) the probe packet that returned directly from s_p and (2) the probe packet that first traversed the RTP P, returned to s_p, and then went back to its original MP using the same shortest path of the overlay network. The MP can estimate the RTT of each probe packet as the delta between the time it sent the original probe packet and the return time of that probe packet.

For example, in Figure 1.1 GRAMI estimates the symmetric RTT of the path s_2, s_4, s_6 (dotted path) by subtracting the RTT of the symmetric path m_1, s_3, s_2 (solid path) from the RTT of the symmetric path m_1, s_3, s_2, s_4, s_6 (dashed path).

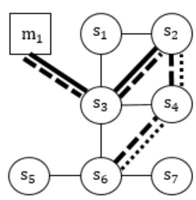


Figure 1.1: Monitoring the symmetric RTP: s_2, s_4, s_6

Estimating the RTT by subtraction of two RTTs is very common and was also used in [4, 5, 6]. However, in traditional networks, this technique is very limited, since the path cannot be controlled.

Monitoring the RTT of a single link is equivalent to monitoring an RTP P, where P is the path back and forth on a single link. We note that GRAMI can also monitor the links to hosts which are not MPs by installing virtual switches on the hosts, and monitoring them as part of the network.

GRAMI is very efficient: First, it requires only four flow entries installed on every switch to construct the overlay network and to enable monitoring of all the links. Any additional RTP requires a maximum of two more flow entries on every switch in the path. Second, it uses only a small number of probe packets; only one probe packet is sent from each MP, and the number of return probe packets is equal to the number of measured RTPs plus the number of measured links. Finally, the controller, which is often the most busy component in the network, is not involved in the online RTT monitoring.

We implemented GRAMI and ran simulations on a network emulated with Mininet [7] and based on CPqD OpenFlow virtual switches [8]. The code can be found in [9]. We demonstrated the efficiency of GRAMI on different topologies and examined its overhead on a hardware switch [10]. The results indicate that GRAMI adds short latency to the measurements with Mininet ($\sim 12\mu s$ for every packet duplication and $\sim 55\mu s$ for every tagging operation) and even shorter latency in the hardware switch ($< 4\mu s$ for every packet duplication and $< 1\mu s$ for every tagging operation).

The remainder of this work is organized as follows. Chapter 2 elaborates on relevant background and related work. Chapter 3 provides an overview of GRAMI. Chapters 4 and 5 describe the offline and online phases respectively, while Chapter 6 outlines additional technical details. Chapter 7 presents the evaluation of GRAMI, and Chapter 8 concludes.

Chapter 2

Background and related work

2.1 Time measurements in the internet

Several factors might impact the experienced RTT: link latency and bandwidth, queuing delays, overloaded network, etc. While some factors are properties of the network and remain constant, others can rapidly change due to the network traffic, significantly affecting the measured RTT. Therefore, the RTT must be monitored constantly in order to track changes.

Multiple innovative and sophisticated approaches tried to overcome the limited control over routing obtainable with classic routing protocols. Most of them focus on inter-domain measurements, while our work focuses on intra-domain measurements. King [11] and IDMaps [12] are mechanisms for estimating the RTT of paths between any two hosts in the internet. Paris Traceroute [13] creates symmetric paths in the internet by manipulating load balancers in the monitored paths and comparing their RTT. Network Radar [14] uses network tomography [15] and the RTT of different paths for one-way delay estimations. Still, these solutions solve problems in an environment in which the control over routing is limited, and therefore cannot monitor the RTT of every path in the network.

2.2 Measurements in OpenFlow networks

OpenFlow enables control over the routing in the network datapath by allowing the controllers to install flow entries on the switches. OpenFlow adds a lot of useful information with multiple counters and meters. Yet it does not supply any time measurement API as part of its specification.

Tools like Ping and Traceroute, the common tools for RTT monitoring [16], or more recently proposed tools such as PingMesh [17], are not suitable for OpenFlow networks, since they cannot observe layer-2 hop and the datapath has no IP address. Therefore, researchers have tried to create tools more suitable for OpenFlow networks. Several works [18, 19, 5] used the controller to send probe packets and measure their delay. However, the control path has different delays than the data path and it frequently becomes a bottleneck [20], making it harder to produce accurate results. Van Adrichem et al. [18] received noisy results and concluded that "The control plane is unsuitable to use as a medium for time-accurate delay measurements." In [21] Agarwal et al. proposed SDN Traceroute, which uses PACKET_IN messages to determine the paths traversed by specific packets in OpenFlow networks. However, it also relies on the control plane and therefore cannot be used for accurate RTT monitoring.

In the closest work to ours, Shibuya et al. [6] enabled all physical links RTT monitoring by setting paths for probe packets sent from a single point in the network, other than the controller. Table 2.1 compares GRAMI to the solution in [6]. GRAMI can monitor the network from any number of MPs and balance the overload between them. Moreover, GRAMI reduces the number of probe packets sent from the MPs per active measurement to a single probe packet from each MP. Finally, GRAMI is more resource efficient, it requires only four flow entries on every switch in order to monitor all the links, and it requires two additional flow entries to monitor any additional RTP that traverses through this switch. In comparison, the solution in [6] requires the number of flow entries on the switches to be proportional to the number of links in the network for monitoring the links only. To the best of our knowledge, GRAMI is the first infrastructure that enables RTT monitoring of any RTP.

	MPs	Probe packets	Flow entries	Measuring
GRAMI	Any k	Send k, Return $n+r$	$4+2r$	Links & RTPs
[6]	1	Send n, Return n	$O(n)$	Links Only

Table 2.1: GRAMI vs. Shibuya et al.'s solution for network with n links and r RTPs.

2.3 Estimating one-way delay

In order to accurately measure one-way delay, a time-stamped packet should be sent from source to destination. For example, Consistent NetFlow [22] sends time-stamped packets between routers. The main problem with this approach is time-synchronization, which adds a lot of complexity to the network.

To overcome this problem, many approaches halved the RTT of the selected path. As discussed in [23], the accuracy of this approach is highly dependent on path symmetry and symmetric load in both directions of the path. GRAMI adds small but asymmetric overload. Therefore it is not suitable for one-way delay estimation.

Chapter 3

GRAMI Overview

3.1 Goal and considerations

Our main goal is to create a generic, scalable and efficient infrastructure that enables RTT monitoring for any RTP and for all the links in the network. To achieve this goal, GRAMI meets the following criteria:

1. **Compatibility:** GRAMI should work with every OpenFlow network as is. Therefore, it makes no assumptions on the network topology and does not require changes to the OpenFlow protocol or to the switches.

2. **No Time-Synchronization:** Time-synchronization adds a lot of complexity to the network. Therefore, GRAMI does not count on time-synchronization.

3. **Active Probing:** Passively sampling packets in the vantage points is not sufficient for covering every path in the network. Thus, GRAMI uses active probing.

4. **Minimal Controller Involvement:** As explained in [18], the control plane is not suitable for time-accurate measurements. In addition, the control path frequently becomes a bottleneck [20]. For that reason, the controller does not participate in RTT monitoring in GRAMI.

5. **Flexibility:** GRAMI should be flexible. Thus, it can turn any vantage point into an MP and it can monitor from any number of MPs.

6. **Resource Efficiency:** First, the flow table capacity is often limited and multiple flow entries can degrade the switch performance [24]. Therefore, GRAMI installs only a small number of flow entries on the network switches. Second, overloading the network may decrease the accuracy of the measurements. Hence, GRAMI sends only one probe packet from every MP.

7. **Accuracy:** GRAMI builds the overlay network and uses packet duplication to optimize the accuracy of the results.

8. **Dynamic Updates:** Configurations are a burden to the network operators. Therefore, GRAMI automatically adapts to dynamic network modifications.

3.2 The workflow of GRAMI

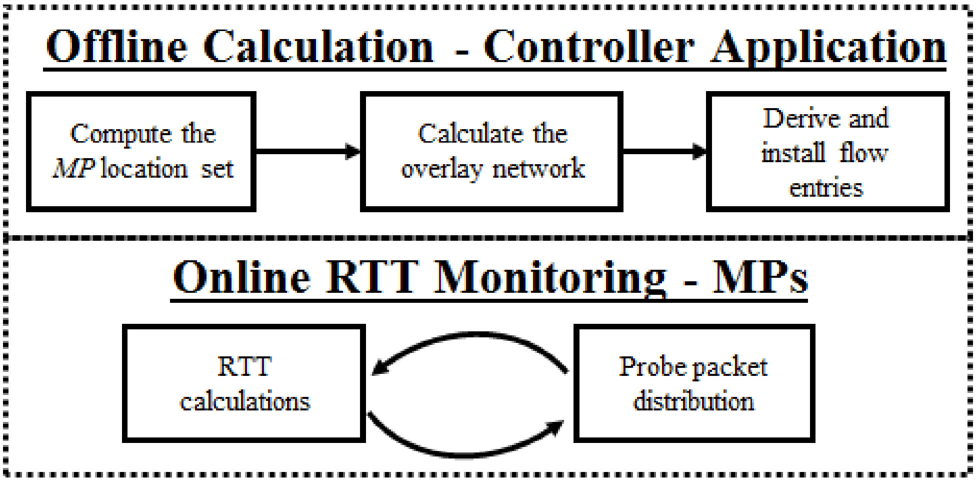


Figure 3.1: The workflow of GRAMI

GRAMI allows the user to select any RTP in the network, and the desired number of MPs. Then, GRAMI works in two phases: an offline calculation phase, in which the controller application sets the routing for the probe packets, and an online RTT monitoring phase in which each MP sends probe packets for RTT monitoring (see Figure 3.1). The offline calculation phase contains several steps: in the beginning, the controller application calculates the MP location set, which optimizes the overlay network. Afterwards, according to the locations of the MPs, it computes the shortest path towards every switch and link in the network, and calculates the overlay network. Eventually, the controller application translates the overlay network and the given RTPs into flow entries, and installs them on the OpenFlow switches.

The MPs are in charge of the online RTT monitoring phase. Each MP periodically sends a single probe packet. The probe packets are duplicated, distributed and tagged within the switches according to the flow entries. All of the probe packets return to the MP from which they originated. For each of the returned probe packets, the MP extracts the tags in order to identify the path the probe packet traversed. Eventually, it uses the RTTs of the returning probe packets to estimate the RTTs of all the links and specified RTPs.

Once in a predefined time, a summary of the results is sent by the MPs to the controller or to any application that might require the information. In the case of dynamic network modifications such as link failure or the addition of a new switch, the overlay network is recalculated and the flow entries are reinstalled.

Chapter 4

GRAMI offline phase

The offline phase is composed of three steps: (1) computing the MP location set, (2) calculating the overlay network, and (3) deriving and installing the corresponding flow entries on the network switches. These flow entries ensure that the probe packets will traverse the network according to the overlay network and RTPs. In this Chapter we elaborate on steps (1) and (2). For the ease of reading, we explain step (3) in Section 6.2, after elaborating on the online phase in Chapter 5.

4.1 Computing the MP location set

The controller application learns about the network topology by using a topology discovery application. It receives as an input the desired number of MPs, k, and in some cases, a fixed set of vantage points to be MPs. The controller application completes this set to a set of size k by computing the locations of the rest of the MPs. The MP location set should minimize the maximal depth (i.e., the number of links from the closest MP). GRAMI selects the location set that balances the number of links monitored by each MP, as long as it does not increase the depth of any link.

This type of optimal location set problem is known to be *NP-hard* [25]. GRAMI chooses the best location set among those found by two algorithms: farthest-first traversal greedy algorithm [25][1] and local-search heuristic[2]. This approach bounds the maximal link depth to be $2d + 1$ where d is the maximal link depth in the optimal solution.

[1] GRAMI uses the given set or chooses the first MP randomly and then iteratively adds the farthest MP in every iteration.

[2] GRAMI uses the given set or starts with an empty set and then iteratively adds the optimal local MP in every iteration.

4.2 Calculating the overlay network

The overlay network sets a *singular shortest path* from every link in the network to its closest MP (see Figure 4.1).

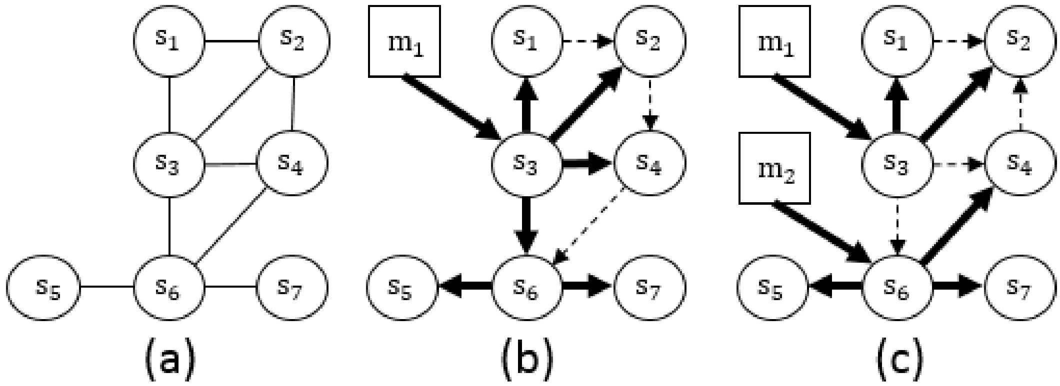


Figure 4.1: (a) Example topology of network switches.

(b) The overlay network after connecting MP m_1 to s_3.

(c) The overlay network after connecting MP m_1 to s_3 and MP m_2 to s_6.

In case of a single MP, the overlay network is a Direct Acyclic Graph that covers all the links in the network.

To construct the overlay network for a single MP, GRAMI first builds a shallow spanning tree, i.e., a shortest path spanning tree. The links of the spanning tree are marked as solid links. We denote the *single* ingress solid link of every switch as its *parent link.* Then GRAMI adds the rest of the links and marks them as dashed links. In order to minimize the depth of the dashed links, GRAMI sets the direction of the dashed links from the switch with the lower depth to the switch with the higher depth. Note that the difference between these depths is at most one.

If there are multiple MPs, GRAMI divides the links between them so that every link is monitored by its closest MP. If there are several closest MPs, GRAMI tries to balance the number of links connected to each one. In this case, the overlay network is composed of multiple DAGs with the MPs as their starting vertices. The solid links create a single path to every switch from exactly one MP and the dashed links cover the remaining links. This overlay network is calculated in four steps (see Figure 4.2):

1. The links are divided between the MPs, creating k connected sub-networks. Every link is connected to its closest MP. In case of a tie, the link is added to the MP that has fewer links in its DAG. Note that in this step, a switch can be covered by multiple MPs.

2. An overlay network with a single MP is calculated for every sub-network.

3. All the sub-networks are merged into a single network.

4. For every switch with multiple *parent links*, the *parent link* with the minimal depth remains solid and the rest of the ingress links become dashed.

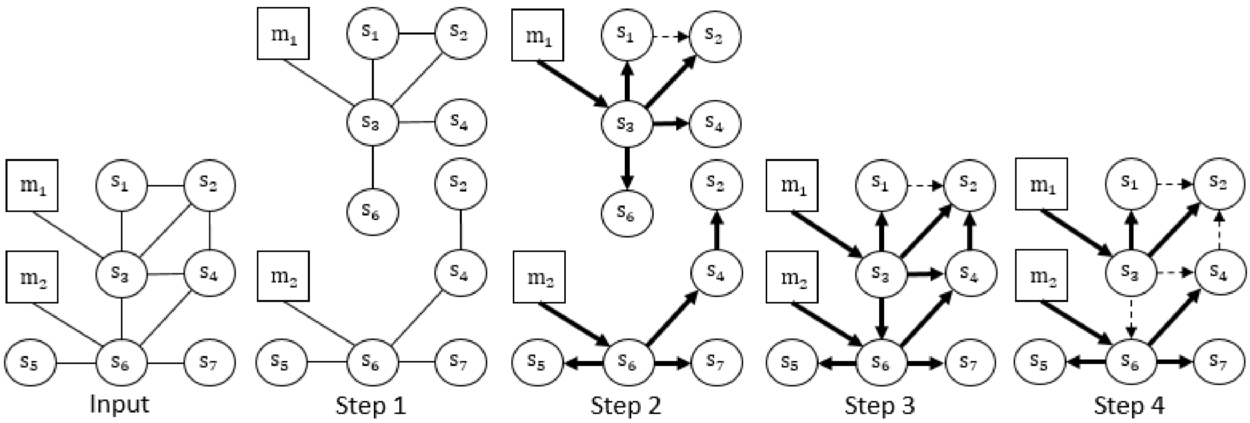


Figure 4.2: calculating the overlay network for two MPs.

GRAMI automatically adapts to dynamic modifications caused by failures or additions of network components. Those modifications are detected by the controller, either by packets that arrive at the controller to inform it that a new switch was added ($OFTP\_HELLO$ message), or by using a topology discovery application. After the modification has been detected, the controller application recalculates the overlay network, derives the flow entries, and installs them without any manual configuration.

Chapter 5

GRAMI online phase

In the online phase, the MPs work in measurement rounds. In every round, each MP sends a single probe packet with the measurement round number as payload. This number is used to match the sent probe packet to the return probe packets. The sent probe packet is duplicated and tagged within the switches and distributed over the overlay network. The duplication mechanism does add a short latency to the processing time in the switches (see Section 7.2); however, it also obviates the need for sending multiple probe packets.

The probe packets cover all the links and RTPs, and return to their original MP. Each MP extracts the tags and the measurement round number from the returning probe packets, and saves their RTT as the time elapsed since sending the probe packet with the same measurement round number. In this way, the MP can calculate the RTT of every link and RTP in its sub-network. In this Chapter, we first explain the probe packet distribution and then the RTT calculations.

5.1 Probe packet distribution over the overlay network

The probe packet distribution is determined by the probe packet's tags while arriving at a specific switch. The tags are encoded in the probe packet headers (see details in Section 6.1). According to the headers and the ingress port, the switch *matches* the probe packet to a specific *flow entry* and executes the corresponding *actions*. In this section we explain the general idea behind the probe packet distribution and in Section 6.2 we describe the flow entries.

Each probe packet contains a $directionFlag$ that indicates the direction of distribution. Probe packets that traverse the overlay network in the direction of the links are denoted as *forward probe packets*. The *return probe packets* are denoted correspondingly. The MPs send only forward probe packets.

To ensure coverage of all the links, probe packets are distributed over the overlay network as follows. When a *forward probe packet* arrives at switch s_i from switch s_j, where the link from s_j

to s_i is the parent link of s_i in the overlay network, s_i duplicates the probe packet and distributes the clones through all of its egress links according to the overlay network. Note that the egress links can be either solid or dashed. In addition, s_i sends a return probe packet back to s_j. The return probe packet is tagged with the IDs of both switches (s_i, s_j) to identify the last link and its direction. Additionally, the probe packet also contains a $ParentFlag$ that indicates whether the link is a parent link. In this case, the $ParentFlag$=True in the return probe packet.

When a *forward probe packet* arrives at a switch s_i from switch s_j, where the link from s_j to s_i is not s_i's parent link, s_i *only* sends the return probe packet back to s_j with $ParentFlag$=False. This mechanism ensures that every link and switch will be covered, *but only once*.

When a return probe packet arrives at a switch, the switch sends it through its parent link. The return probe packet thus traverses the shortest path back to its original MP (note that the return path is composed of solid links only). Figure 5.1 shows the distribution process over the example network.

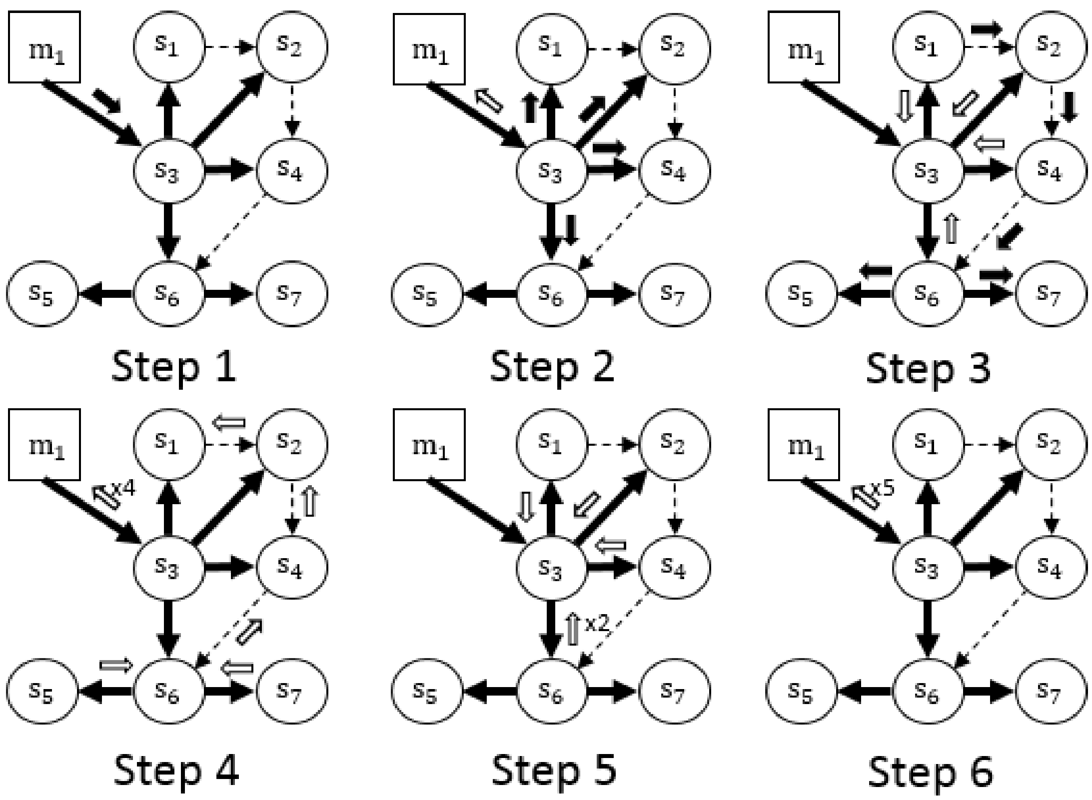


Figure 5.1: Step-by-step distribution process for the network links. The black arrows represent forward probe packets and the white arrows represent return probe packets. For convenience, the steps are presented synchronously.

The RTPs must be preconfigured in the network so that the probe packets will be able to traverse them. When a forward probe packet arrives at a switch s_p from its *parent link*, s_p sends, in addition to the aforementioned clones, a single probe packet to each RTP P that starts at s_p. Each probe packet contains an $RTPFlag$ indicating whether the packet measures an RTP or a link. The probe packets that measure RTPs are tagged with $RTPFlag$=True (for links $RTPFlag$=False). Additionally, the probe packets are tagged with the ID of s_p and the ID of the RTP P. The switches along the RTP use the tag P to send the probe packet along P until it returns to s_p. Then, s_p tags the probe packet as a return probe packet, and sends it back to the MP via its parent link.

Figure 5.2 illustrates the process of measuring an asymmetric RTP P that starts at s_2. In step 1, probe packets arrives on the same path as the corresponding probe packet in steps 1-2 in Figure 5.1. In steps 2-4, s_2 sends the probe packet to traverse P. In step 5, s_2 identifies the probe packet that returns from P and sends it back to the MP in the same path as the corresponding probe packet in steps 5-6 in Figure 5.1. The process focuses on traversing P and ignores the duplication according to the overlay network.

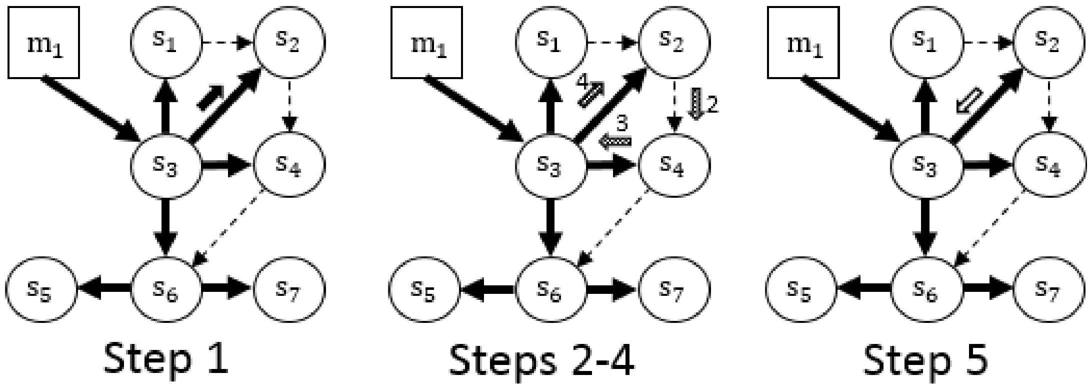


Figure 5.2: Monitoring RTP of the asymmetric RTP P=$\{s_2, s_4, s_3, s_2\}$. The numbers on the arrows represent steps in a synchronous network. The black arrows represent forward probe packets, the dotted arrows represent probe packets traversing P, and the white arrows represent return probe packets.

5.2 RTT Calculations

In every measurement round, each MP sends a single *forward* probe packet but receives multiple *return* probe packets.[1] The MPs have a configurable measurement round timeout; probe packets that exceed this timeout will be considered lost.

As explained, the return probe packets contain the information of a link or that of an RTP. In the case of a link, the return probe packet contains an $RTPFlag$=False. The RTT of the link from s_j to s_i is equal to the RTT of the probe packet tagged with (s_i,s_j)[2] minus the RTT

[1]We assume that under similar conditions, similar packets experience similar processing time in the switches. However, for specific scheduling techniques such as: Virtual Output Queues or Active Queue Management, the time in queue may vary for different ingress ports; other techniques should be used in that case. We moreover assume that middleboxes will use consistent routing and will not change the tagging of the probe packets.

[2]The probe packets are tagged on their way back; therefore, (s_i,s_j) represents the link from s_j to s_i in the

of the probe packet that measured the shortest path to s_j, which is the RTT of a probe packet that is tagged with (s_j,s_k) and $parentFlag$=True.

In the case of an RTP, the return probe packet contains an $RTPFlag$=True and the IDs (s_p,P), where s_p is the first switch in the RTP P. The RTT of P is equal to the RTT of the probe packet tagged with (s_p,P) minus the RTT of the probe packet that measured the shortest path to s_p.

Note that the MP does not need to receive the network topology from the controller application to perform RTT calculations. However, during topology changes, some of the probe packets might traverse unexpected paths, possibly leading to calculation errors. Therefore, unstable networks require a tight connection between the controller and the MPs.

overlay network.

Chapter 6

GRAMI - technical details

In this chapter we describe the tagging mechanism and the flow entries installed on the network switches.

6.1 Tagging mechanism

The controller 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 $RTPFlag$.

To enable the tagging only in the desired switches, GRAMI uses $SetIDFlag$ to indicate whether the IDs should be tagged (i.e., at least one ID has not yet been tagged).

GRAMI uses the following tags (summarized also in Table 6.2):

1. $DirectionFlag$ indicates whether the direction of the probe packet is *forward* or *return*. It is used by the switches to match the probe packets to a flow entry.

2. $SetIDFlag$ indicates whether the packet still has to be tagged with an RTP ID or a switch ID. It is used by the switches to match the probe packets to a flow entry.

3. $ParentFlag$ indicates whether the last link in the path was a parent link. It is used by the MPs for RTT calculations.

4. $RTPFlag$ indicates whether the information in the return probe packet is related to an RTP or to a link. It is used by the MPs for RTT calculations.

5. $(ID1, ID2)$ are used by the MPs for RTT calculations. If $RTPFlag$=False, the IDs are the endpoints of a link; otherwise, $ID2$ is the ID of the RTP, and $ID1$ is the ID of the

first switch in the RTP. Note that $ID2$ is also used by the switches to match the probe packet and forward it along the RTP.

Since the probe packets are created in the MPs and used only for RTT monitoring, they can be independent of a specific protocol. Therefore, GRAMI can add any payload, and select any field for tagging, as long as the OpenFlow version supports tagging and matching for that field. We implemented GRAMI with OpenFlow1.3 and used ETH_TYPE (16 bits) and two VLAN headers (12 bits each). The $DirectionFlag$ and the $SetIDFlag$ were encoded by four different ETH_TYPE 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.[1] Hence, the current 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.

VLAN headers are commonly used for tagging in OpenFlow networks [26]. However, tagging with VLANs has the overhead of using the "PUSH_VLAN" and "POP_VLAN" actions, in addition to setting the field with the relevant tag. In the P4 language (also referred as "OpenFlow 2.0 API") [27], the user can define specific headers for tagging, and only set these headers in order to tag the packet. Implementing GRAMI with the P4 language should thus significantly reduce the overhead caused by the tagging mechanism.

6.2 The flow entries

The controller application calculates the overlay network and derives the corresponding flow entries. In addition, it finds the relevant ingress and egress ports of every switch along each of the selected RTPs. Then, the controller application installs the flow entries on the network switches.

Table 6.1 describes in detail all the flow entries that implement the distribution, duplication, and tagging mechanisms. If a probe packet matches several flow entries, the one with the highest priority will be executed.

The purpose of flow entries 1-4 is to distribute the probe packets according to the overlay network; therefore, these flow entries are installed on all the switches. The controller application installs flow entries 5-6 for every selected RTP. The purpose of these flow entries is to distribute the probe packets over a specific RTP P which starts in switch s_p. Therefore, these flow entries are installed only on switches along P. Flow entry 5 is installed on every switch in P except s_p in order to create P (it can be installed twice if the switch appears twice in P). Flow entry 6 is installed on s_p so the probe packet will return to the MP after traversing P.

[1] $ID2$ requires at least $log(max(r, n) + 1)$ bits for a network with n switches and r RTPs.

No.	Purpose	Name	Switch	Match	Priority	Action
1	Traverse the overlay network.	Distribute	All	i.$DirectionFlag$=Forward ii.$SetIDFlag$=True iii.Ingress port = parent port	2	1. Duplicate the probe packet and send the clones to all the egress ports with no tag changes. 2. Send probe to the ingress port with $DirectionFlag$=Return, $SetIDFlag$=True, $parentFlag$=True and $ID1$= ID of s. 3. For every RTP P which starts in s, send a probe packet to the first link in P with $RTPFlag$=True, $DirectionFlag$=Forward, $SetIDFlag$=False, $ID1$=ID of s and $ID2$=RTP ID of P.
2		Do not dis-tribute	All	i.$DirectionFlag$=Forward ii.$SetIDFlag = True$ (Ingress port $\neq$ parent port)	1	1. Send probe to the ingress port with $DirectionFlag$=Return, $SetIDFlag$=True, $parentFlag$=False and $ID1$= ID of s.
3		Return And Tag	All	1.$DirectionFlag$=Return ii.$SetIDFlag$=True	1	1. Send to the parent port with $SetIDFlag$=False and $ID2$ = ID of s.
4		Return No Tag	All	i.$DirectionFlag$=Return ii.$SetIDFlag$=False	1	1. Send to parent port with no tag changes.
5	Traverse RTP P which starts in switch s_p.	Traverse P	Not s_p	i.$DirectionFlag$=Forward ii.$SetIDFlag$=False iii.$ID2$ = RTP ID of P iv. Ingress port value	1	1. Forward according to the RTP ID and the ingress port value to the next link in P through specific egress port with no tag changes.
6		Return From P	s_p only	i.$DirectionFlag$=Forward ii.$SetIDFlag$=False iii.$ID2$=RTP ID of P	1	1. Send to the parent port with $DirectionFlag$=Return.

Table 6.1: Flow entries installed on switch s. The egress ports and parent ports are derived from the overlay network. The ingress port is the port from which the packet entered.

Below we describe the life-cycle of a probe packet. In parentheses we note the state of the probe packet according to Table 6.2 and the flow entry number according to Table 6.1.

	$DirectionFlag$ (1)	$SetIDFlag$ (1)	$ParentFlag$ (1)	$RTPFlag$ (1)	$ID1$ (log(n+1))	$ID2$ (log(max(r,n)+1))
(a)	$Forward$	$True$	$False$	$False$	$NULL$	$NULL$
(b)	$Return$	$True$	$True$\False	$False$	$Switch\ ID$	$NULL$
(c)	$Return$	$False$	$True$\False	$False$	$Switch\ ID$	$Switch\ ID$
(d)	$Forward$	$False$	$False$	$True$	$Switch\ ID$	$RTP\ ID$
(e)	$Return$	$False$	$False$	$True$	$Switch\ ID$	$RTP\ ID$

Table 6.2: The tagging of a probe packet in every possible state. We note in parentheses the number of bits required for each tag in a network with n switches and r RTPs.

The *forward* probe packet is sent from the MPs with $(ID1, ID2)$=(NULL,NULL) and $SetIDFlag$=True (a). When it arrives at a switch from its parent link, the switch distributes the probe packet to its egress links (a,1) and sends a *return* probe packet through its ingress port after tagging $ID1$ and $ParentFlag$=True (b,1). If a *forward* probe packet with $SetIDFlag$=True arrives at a switch from a link which is not its parent link, the switch tags $ID1$, $ParentFlag$=False, and sends a *return* probe packet (b,2). The first switch on the return path to the MP tags $ID2$ and sets $SetIDFlag$=False (c,3), so the probe packet will not be tagged until it returns to the MP (c,4).

When the probe packet start traversing an RTP P that originates in a switch s_p, s_p sets $RTPFlag$ = True, $(ID1, ID2)$=(s_p, P) and $SetIDFlag$=False (d,1). The packet is still a *forward* probe packet but it will not be tagged. The switches along the P forward the probe packet until it returns to s_p (d,5). Then, s_p sets $DirectionFlag$=Return and the probe packet returns to the MP (e,6) along the shortest path.

Chapter 7

Evaluation and discussion

We tested GRAMI on a network emulated with Mininet and based on CPqD OpenFlow virtual switches controlled by a single Ryu controller [28]. All links were set with a $100Mbps$ bandwidth and $20ms$ latency. The only traffic in the network was OpenFlow communication between the controller and the switches.

7.1 Building the overlay network

No.	Name	Links	Switches	Max Depth	Calc Time (ms)
1	GetNet	8	7	3	2.82
2	Peer1	20	16	4	11.79
3	Airtel	37	16	3	15.37
4	BT Europe	37	24	3	30.02
5	BICS	48	33	6	55.39
6	ATT	57	25	4	47.89
7	GEANT	61	40	5	87.49
8	Deutsche...	62	39	5	101.13
9	Forthnet	62	62	5	106.48
10	BTN	65	53	6	120.8

Table 7.1: Calculation times of the overlay network for various topologies and a single MP.

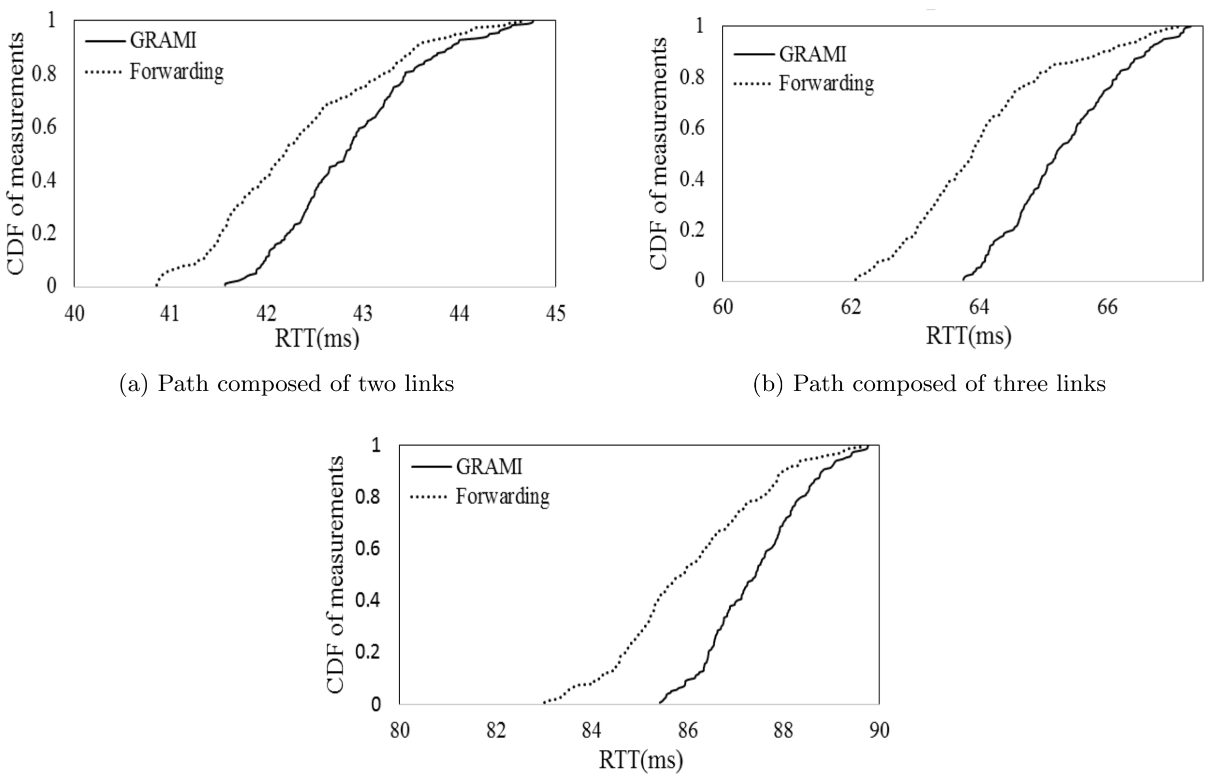
We tested GRAMI on topologies taken from Topology Zoo [29]. In all of the tested topologies, GRAMI successfully monitored the RTT of all the links in the network and the RTT of different RTPs we preconfigured. Table 7.1 describes the topologies, the overlay network calculation time, and the maximal depth when a single MP is placed in the optimal location. As shown in the table, the calculation time tends to grow when the network size and maximal depth increases.

Installing the flow entries took an additional 2.7ms per switch.

In the following tests we used topology 5. In every test we conducted 200 measurement rounds and sent a single forward probe packet from every MP with 1 second interval between rounds. Note that Mininet is a virtual environment not suitable for measuring time or performance accurately. However, it is a proof of concept and gives us a sense on the impact of different network parameters. In the next sections, we try to estimate GRAMI's overhead in Mininet followed by an assessment of how different network parameters might affect the accuracy of the RTT measurements.

7.2 Overhead analysis

GRAMI has the overhead of duplication and tagging within the switches. We measured the overhead in our emulated Mininet network with virtual switches and found that the average latency for adding a single VLAN header tag is equal to $\sim 55\mu s$, and the average latency for a single packet duplication is equal to $\sim 12\mu s$. To estimate the overhead of GRAMI for paths with different lengths, we selected three switches with depths of 2-4 in topology 5. Then, we installed flow entries to set the shortest symmetric path toward each switch.



(a) Path composed of two links

(b) Path composed of three links

(c) Path composed of four links

Figure 7.1: GRAMI vs. forwarding monitored RTT CDFs for paths composed of different number of links.

We monitored the RTT of those paths with simple forwarding (the packets were forwarded between the switches with no further actions) and with GRAMI. Figure 7.1 compares the monitored RTT and shows that GRAMI adds small overhead to the results, which increases for longer paths. Note that the same link latency was emulated for all the links in the network. As a result, the probe packets that were sent from a certain switch to its egress links returned to that switch within a short time of one another. Since in our emulated network tagging takes much longer than duplication, each probe packet had to wait in the switch queue until the switch tagged the probe packets that preceded it in the queue. Thus, even though the mentioned latencies were relatively small, they accumulated and increased with the length of the measured path.

We measured these mentioned latencies for NoviFlow Novikit 250 switch [10]. The packet duplication took less then $4\mu s$ and tagging took less than $1\mu s$. We expect the overhead to be significantly lower also on other hardware switches.

7.3 Link depth

In order to estimate the impact of depth on the accuracy of the measurements, we calculated the standard deviation of the RTT measurements for every link. The standard deviation provides a good estimation of measurement noise. As Figure 7.2 shows, for deeper nodes, the RTT measurements tend to be more noisy.

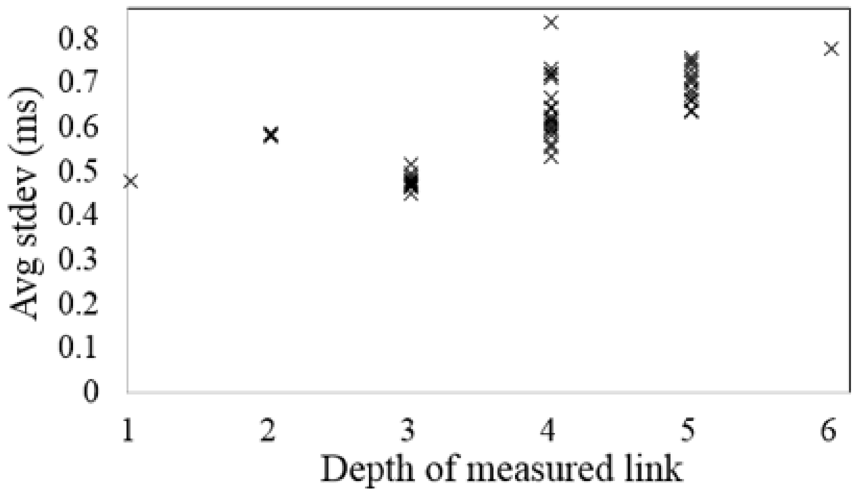


Figure 7.2: Scatter plot of the average standard deviation in the RTT measurements as a function of the link's depth. Every x is a link in the network.

7.4 MP location set

To check how the location of an MP affects the monitored RTT, we used GRAMI to find the optimal and worst locations to place a single MP. Note the optimal location for a single MP

can be found in polynomial time.

Location	Avg depth	Max depth	Avg RTT(ms)	Avg Stdev(ms)
Optimal	3.91	6	21.88	0.75
Worst	7.125	9	21.97	1.2

Table 7.2: Optimal vs. worst location for a single MP.

Table 7.2 shows that the optimal location reduces the depth of the links, GRAMI's overhead, and measurement noise.

We also monitored all the links in the network for 1-6 MPs connected to the network. As shown in Figure 7.3, the average RTT monitored by GRAMI decreased as the number of MPs grew. The dashed line represents the link latency that was emulated for all the links in the network. Since the RTT also includes the processing time in the switches, the average RTT cannot reach the dashed line. However, as it gets closer, the overhead of GRAMI decreases.

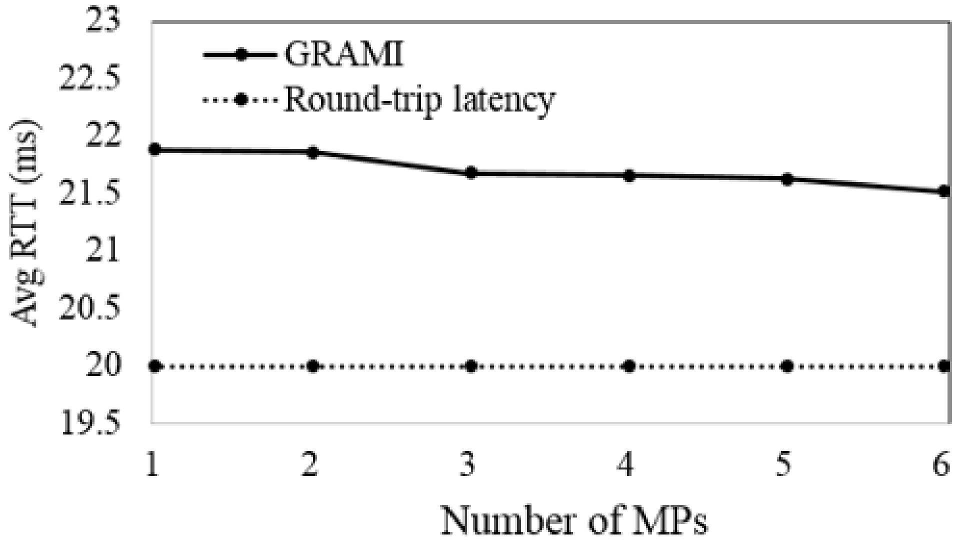


Figure 7.3: Average RTT of the links in the network as a function of the number of MPs.

Figure 7.4 shows that the measurements tend to be less noisy as the number of MPs increases. We assume these more stable results are due to the use of multiple MPs, which reduces the average depth in the network.

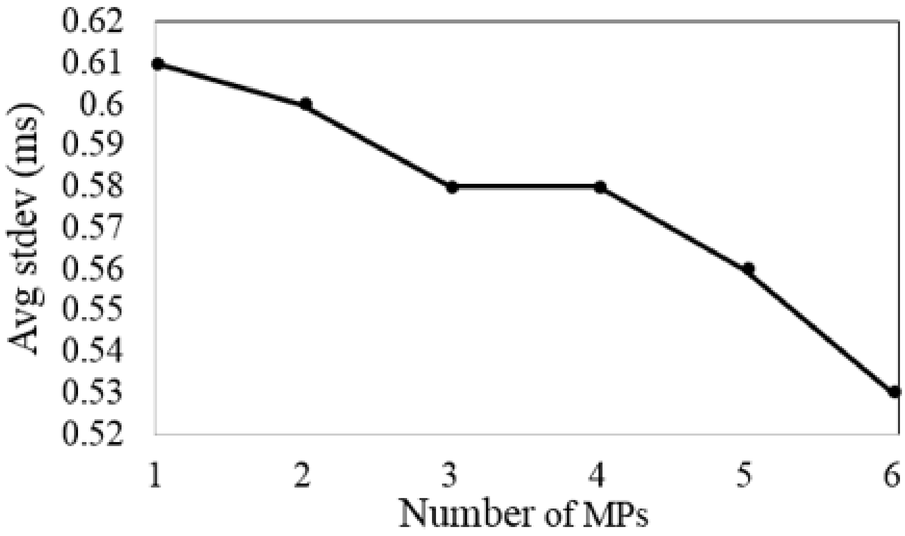


Figure 7.4: Average standard deviation in the RTT measurements of all the links as a function of the number of MPs.

7.5 Sensitivity to network conditions and dynamic changes

We tested how GRAMI responds to dynamic changes in the network by selecting a specific link and monitoring its RTT for 15 seconds with a 200ms interval between probe packets.

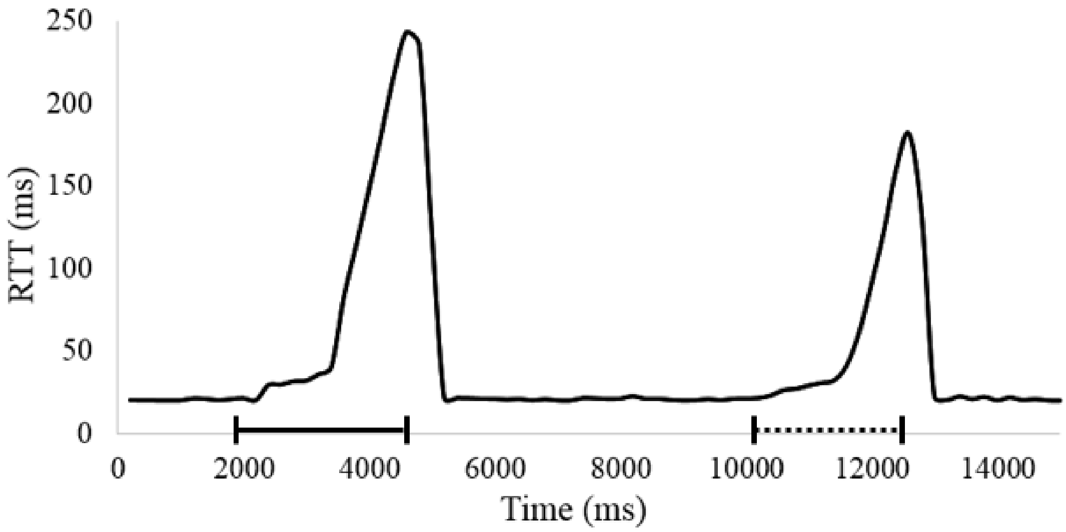


Figure 7.5: RTT over time for a single link that was overloaded at known times. In the solid segment, we overloaded the network with $57Mbps$, as opposed to $54Mbps$ in the dotted segment.

First, we tested whether GRAMI can detect a flooded link. For that reason, we used the Ipref tool to overload the specific link with known data rates as shown in Figure 7.5. The graph shows that GRAMI monitored an increasing RTT during overloading. Moreover, the graph shows that GRAMI immediately detected when we stopped overloading the link.

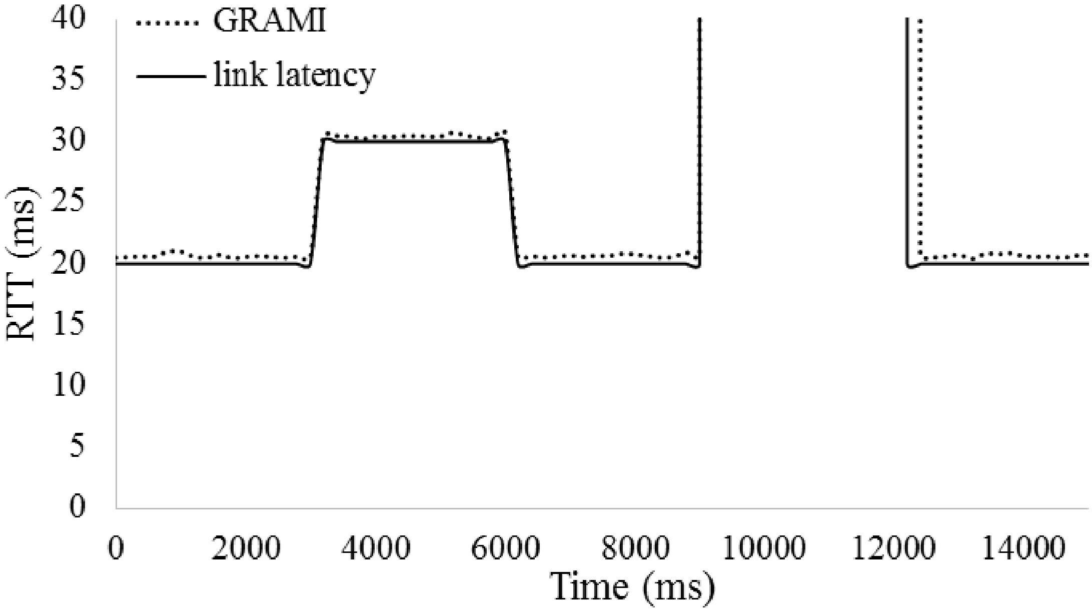


Figure 7.6: RTT over time for a single link with dynamic changes.

In addition, we tested whether GRAMI can detect changes in the latency of a link or link failure. As Figure 7.6 shows, GRAMI immediately detected the change in the link's latency, and when the link failed, it did not receive a return probe packet that measured the link, and thus could not monitor the RTT. As the graph shows, GRAMI did not immediately identify the recovery of the link. In fact, it took the controller application 89.24ms to detect the recovery and calculate the new overlay network, and 188.11ms to install the flow entries on all the switches.

Chapter 8

Conclusions

We introduced GRAMI, an infrastructure that enables RTT monitoring all the links and of all the RTPs preconfigured in OpenFlow networks. GRAMI is easy to operate and supplies important information for network operators. Moreover, GRAMI is resource efficient and does not involve the controller in the online RTT monitoring. This work demonstrates the power of OpenFlow and SDN concepts, and uses the new capabilities of OpenFlow to enable accurate RTT monitoring in granularity and stability that could not be achieved in traditional networks.

Bibliography

[1] C. Pelsser, L. Cittadini, S. Vissicchio, and R. Bush, "From paris to tokyo: On the suitability of ping to measure latency," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 427–432.

[2] M. Crovella and B. Krishnamurthy, *Internet measurement: infrastructure, traffic and applications*. John Wiley & Sons, Inc., 2006.

[3] O. N. Foundation, "Openflow switch specification (version 1.3.0)," 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

[4] A. B. Downey, "Using pathchar to estimate internet link characteristics," in *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4. ACM, 1999, pp. 241–250.

[5] C. Yu, C. Lumezanu, A. Sharma, Q. Xu, G. Jiang, and H. V. Madhyastha, "Software-defined latency monitoring in data center networks," in *Passive and Active Measurement*. Springer, 2015, pp. 360–372.

[6] M. Shibuya, A. Tachibana, and T. Hasegawa, "Efficient performance diagnosis in openflow networks based on active measurements," *ICN 2014*, p. 279, 2014.

[7] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[8] ""cpqd openflow software switch", http://cpqd.github.io/ofsoftswitch13/."

[9] "https://github.com/alonatari1/grami."

[10] "noviflow novikit 200", http://www.nvc.co.jp/pdf/product/noviflow/novikit250datasheet.pdf.

[11] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 5–18.

[12] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "Idmaps: A global internet host distance estimation service," *Networking, IEEE/ACM Transactions on*, vol. 9, no. 5, pp. 525–540, 2001.

[13] B. Augustin, T. Friedman, and R. Teixeira, "Measuring load-balanced paths in the internet," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM, 2007, pp. 149–160.

[14] Y. Tsang, M. Yildiz, P. Barford, and R. Nowak, "Network radar: tomography from round trip time measurements," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 175–180.

[15] Y. Vardi, "Network tomography: Estimating source-destination traffic intensities from link data," *Journal of the American Statistical Association*, vol. 91, no. 433, pp. 365–377, 1996.

[16] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "A survey on network troubleshooting," Technical Report Stanford/TR12-HPNG-061012, Stanford University, Tech. Rep., 2012.

[17] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 2015, pp. 139–152.

[18] N. L. Van Adrichem, C. Doerr, F. Kuipers *et al.*, "Opennetmon: Network monitoring in openflow software-defined networks," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–8.

[19] K. Phemius and M. Bouet, "Monitoring latency with openflow," in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, 2013, pp. 122–125.

[20] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Scotch: Elastically scaling up sdn control-plane using vswitch based overlay," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 403–414.

[21] K. Agarwal, E. Rozner, C. Dixon, and J. Carter, "Sdn traceroute: Tracing sdn forwarding without changing network behavior," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 145–150.

[22] M. Lee, N. Duffield, and R. R. Kompella, "Opportunistic flow-level latency estimation using

consistent netflow," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 1, pp. 139–152, 2012.

[23] O. Gurewitz, I. Cidon, and M. Sidi, "One-way delay estimation using network-wide measurements," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2710–2724, 2006.

[24] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "Openflow switching: Data plane performance," in *Communications (ICC), 2010 IEEE International Conference on.* IEEE, 2010, pp. 1–5.

[25] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985.

[26] S. Narayana, J. Rexford, and D. Walker, "Compiling path queries in software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking.* ACM, 2014, pp. 181–186.

[27] 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.

[28] "ryu sdn framework", http://osrg.github.io/ryu-book/en/ryubook.pdf.

[29] "the internet topology zoo", http://www.topology-zoo.org/.

תקציר

ניטור הזמן הלוקח להודעות לעבור הלוך ושוב (Round-Trip Times) על גבי מסלולים שונים ברשת מספק מידע חיוני לצורך פתרון בעיות ברשת ותכנון תעבורת הרשת. השיטה הנפוצה ביותר למדידת זמנים אלו היא שליחת הודעות בדיקה מנקודות נבחרות ברשת. ברשתות קלאסיות, השליטה על הניתוב ברשת מוגבלת מאוד, לכן בלתי אפשרי לנטר את כל הדרכים הקיימות ברשת.

הקונספט החדשני של Software Defined Networking מפשט את השליטה על הניתוב ברשת. עם זאת, OpenFlow, הפרוטוקול הנפוץ מבין פרטוקולי ה-SDN, לא מספק תמיכה בניטור זמנים ברשת. בעבודה זו , אנחנו נמנף את היכולות של OpenFlow לשלוט בניתוב ברשת ונציג את GRAMI – תשתית המשמשת למדידת RTTs ברשת. GRAMI בוחר נקודות ברשת שיהוו נקודות מדידה ושולח מהן הודעות בדיקה באופן אקטיבי. הודעות אלו מאפשרות מדידה של RTT עבור כל מסלול מעגלי ברשת שנבחר על ידי המשתמש ושל כל אחד מהלינקים ברשת.

GRAMI תוכנן להיות יעיל במשאבים: הוא דורש ארבע רשומות בלבד שיותקנו על כל נתב ברשת במטרה למדוד RTTs של כל הלינקים ברשת. עבור כל מסלול מעגלי שבוחר המשתמש, GRAMI דורש עד שתי רשומות נוספות שיותקנו על כל נתב ברשת. יתר על כן, GRAMI משתמש במספר מינימלי של הודעות בדיקה, ולא דורש מעורבות של הcontroller לצורך המדידות.

עבודה זו בוצעה בהדרכתה של פרופ' ענת ברמלר בר מבי"ס אפי ארזי למדעי המחשב, המרכז הבינתחומי, הרצליה.

המרכז הבינתחומי בהרצליה
בית-ספר אפי ארזי למדעי המחשב
התכנית לתואר שני (M.Sc.) - מסלול מחקרי

ניטור RTT יעיל ברשתות OpenFlow

מאת
אלון אטרי

עבודת תיזה המוגשת כחלק מהדרישות לשם קבלת תואר מוסמך M.Sc. במסלול
המחקרי בבית ספר אפי ארזי למדעי המחשב, המרכז הבינתחומי הרצליה

ינואר 2016