Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Networked Systems
ETH Zürich — seit 2015

Bachelor Thesis

Deploying a mini-Internet instance across multiple servers

Alex Studer

Supervision: Tobias Bühler, Thomas Holterbach

Professor: Prof. Dr. Laurent Vanbever

Networked Systems Group

Department of Information Technology and Electrical Engineering

Eidgenössiche Technische Hochschule, Zürch

Spring Term 2021

Abstract

The mini-Internet is a “Open platform to teach how the internet practically works” [1] and currently used for the communication networks lecture at the ETH Zürich [1]. As of now the mini-Internet can only run on a single server and its size is thus limited by the computational capacity of the server used. For classes exceeding 150 participating students or in research with a need of bigger and more complex setups, this limitation would most likely be reached [1]. Hence, to improve the scalability of the mini-Internet we implemented the functionality into the mini-Internet to deploy it across multiple servers. This thesis describes how we were able to create an implementation that makes using a multi-server deployment rather easy and invisible to the users of the mini-Internet. Currently the improved mini-Internet not only achieves these objectives, but it also preserves backward compatibility and source code maintainability. In a next step getting the monitoring and debugging features such as the “connectivity-matrix” ready for multi-server deployments, would enable us to offer a holistic package for teaching even really big classes with the mini-Internet.

1 Introduction

The mini-Internet project has become an important part of the communication networks lecture at the ETH Zürich [1, 2]. As of now the mini-Internet runs on a single server and therefore the size of the mini-Internet is limited by the computational capacity of the server used. This was already pointed out in the paper "An Open Platform to Teach How the Internet Practically Works" which initially presented the mini-Internet to the public. Hence, it has suggested to add support for a multi-server deployment [1]. We can see this need more clearly when we have a look at the concrete setup of the mini-Internet used for the communication networks lecture. For around 100–150 students the mini-Internet consists of about 60 ASes, each composed of 8 routers and 4 switches. Using a server with an Intel Xeon CPU with 24 cores @2.30 GHz and 256 GB of memory running on Ubuntu Server 18.04.3 results in a CPU load of 51% and a memory usage of 58%. In case of malicious BGP traffic caused by student misconfigurations we expect the server to crash as high BGP traffic (>15000 advertisements) leads to a strong increase of CPU load (>80%) [1]. Further, some classes easily exceed 150 participating students and we could also imagine using the mini-Internet for research purposes where bigger and more complex setups are needed. In all these cases a single server would not be sufficient anymore and the deployment of the mini-Internet over multiple servers would become a necessity.

This project aims to improve the scalability of the mini-Internet by adding the necessary functionality to deploy it over multiple servers. We want to create an implementation that makes using a multi-server deployment rather easy and invisible to the users of the mini-Internet. Furthermore, we want to be able to create single-server setups as before and preserve good maintainability of the source code by keeping the newly introduced technologies to a minimum.

This thesis will first focus on describing the existing implementation of the mini-Internet and introducing the new technologies important for the extended implementation e.g., Generic Route Encapsulation (GRE) (Chapter 2 – Theory). After understanding the current state and the new technologies used, we discuss how the

new multi-server deployment feature works (Chapter 3 – Implementation). Then we go on presenting how the multi-server mini-Internet was tested and how it performs (Chapter 4 – Result). Chapter 5 highlights the difference between the old and the new mini-Internet setup and discusses practical deployment scenarios (Chapter 5 – Deploy a mini-Internet over multiple servers). Finally, a brief summary provides an overview of the achievements of this work and suggests what could be improved in the future (Chapter 6 – Summary).

The source code, the raw testing data and further materials can be found in a git repository online. You can access it with the following link.
[https://gitlab.ethz.ch/nsg/students/projects/2021/ba-2021-12_multi_server_mini_internet/-/tree/master/]

2 Theory

A proper understanding of the mini-Internet and some of the technologies used forms the basis to understand the decisions made within this project. The following sections provide a summary of these topics.

The mini-Internet. The mini-Internet consist of three basic components: host, switches and routers. Each of them runs in its own dedicated Docker container [1]. Open vSwitch bridges and virtual ethernet links are then used to connect the containers. Open vSwitch is also used for the switch components of the mini-Internet. For the routers, the Internet routing protocol suite FRRouting is used, and the hosts are simple Debian Stretch machines with some networking tools added (e.g., traceroute) [1, 2]. Additional components and features can be added if needed (e.g., monitoring tools, VPN support) [1, 2]. Finally, besides accessing the components directly using the Docker tools to interact with containers, there is an option to use SSH. For example, in class where different groups should be able to access different components of the mini-Internet the SSH option is to prefer. In this case, for each group a SSH proxy container would be created which can be accessed remotely. Starting out from the proxy container the students can then access only the components they are allowed to use [1].

Container and namespaces. Central to the implementation of the mini-Internet are container. They are a more lightweight solution than using virtual machines for each component since container do not run their own full operating system but make use of namespaces instead. Namespaces are a feature of the Linux kernel allowing to isolate software from one another and dynamically allocate system resources to them [1, 3].

Virtual ethernet devices (veth). Using veth devices one can create an interconnected pair of them. The resulting link could be called a virtual ethernet link. This link can act as a tunnel between network namespaces [4]. The mini-Internet makes heavy use of it to interconnect its components with each other [1].

Open vSwitch (OvS). Open vSwitch is a free and open-source multilayer virtual switch [5]. It supports all the features needed for a switch in the mini-Internet. For example, VLANs, the Spanning Tree Protocol (STP), tunneling protocols and OpenFlow to name a few [6]. The mini-Internet uses Open vSwitch bridges to a high extent. Not only are the switch components of the network realized with an OvS bridge in a container, but the containers are connected to each other using OvS bridges [1]. Instead of directly connecting the different containers together the setup used with the OvS bridges in-between makes online manipulation in the mini-Internet possible e.g., simulate link corruption. This project makes special use of this setup to enable remote connection between a pair of servers.

FRRouting (FRR). "FRRouting is a free and open-source Internet routing protocol suit for Linux and Unix platforms." [7] FRRouting provides a full command line interface (CLI) which resembles CLIs of commonly used routers in the Internet. Hence, it is used in the mini-Internet for the implementation of the Routers [1].

Generic Route Encapsulation (GRE). GRE is a protocol for encapsulating one protocol with another protocol. It was first introduced by Cisco in 1994 [8]. As it has been around for a relatively long time, it is widely supported by devices. In contrast to protocols which encapsulate a specific protocol X in a protocol Y GRE is more generic and can be used to encapsulate an arbitrary protocol over another arbitrary network layer protocol [9].

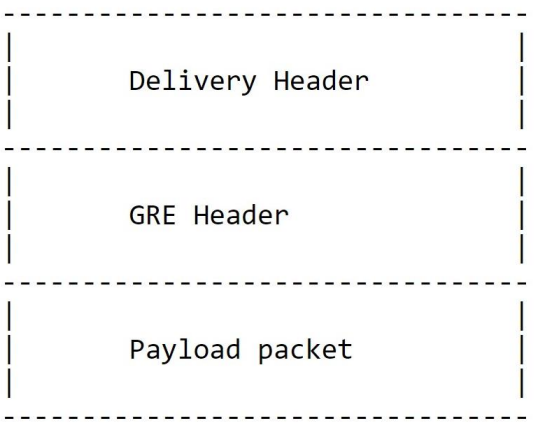

Figure 2.1: Structure of a GRE encapsulated packet [8]

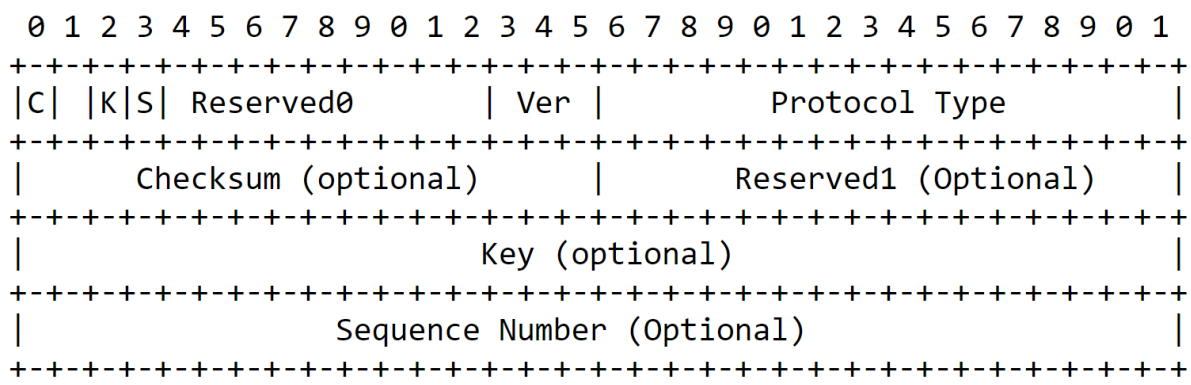

Figure 2.2: Structure of a GRE header [9]

In this project we use GRE to encapsulate an IP packet of network A with an IP header of network C. This way the IP packet from network A can be transported over the network C and then decapsulated in network B and be delivered to the destination IP address specified in network A which lies in network B. Hence, GRE is used for tunneling between two local IP networks which are connected by another IP network.

Special GRE interfaces can be added to an OvS bridge when corresponding GRE tunnels are configured in the Linux IP network stack. This way the incoming traffic to an OvS bridge can be directed to use the GRE tunnel and thereby create arbitrary connections between two OvS bridges even if they are not running on the same server. Exactly this concept of OvS bridges connected by a GRE tunnel will be used in this project to enable a multi-server deployment of the mini-Internet.

3 Implementation

This section carries out in detail how the deployment of the mini-Internet over multiple servers conceptually works and discusses the most important design choices. For a more in-dept view of the implementation we recommend having a look at the actual code.

Separate between ASes only. The mini-Internet like the real internet is composed of multiple ASes which represent interconnected closed entities. A single AS will not exceed the capability of a single server unless it is huge and consists of thousands of components. Therefore, we decided not to allow the possibility to distribute a single AS over multiple servers. Instead, you can decide on an AS per AS basis on which server an AS shall exist. This approach has the advantage to minimize the change of the existing scripts as only the "external links" are subject to the distribution over the servers. The internal AS links stay unaffected. Consequently, the maintainability of the scripts is not affected too much.

Global configuration files. We introduced three new configuration files. They form the "GLOBAL" configuration. The previously existing AS and external links configuration file represent henceforth the "LOCAL" configuration. The new configuration files consist of a global AS configuration (AS_config_GLOBAL.txt), a global external links configuration (external_links_config_GLOBAL.txt) and a global server configuration file (server_config_GLOBAL.txt). The global AS configuration is almost identical to the previous AS configuration only that it has an additional column specifying on which server the AS should run. The global external links configuration looks identical to the previous external links configuration. Here, however, if a GLOBAL configuration is running, the local external links configuration will append additional columns automatically. They will then contain the information needed to establish correct remote external links to other servers.

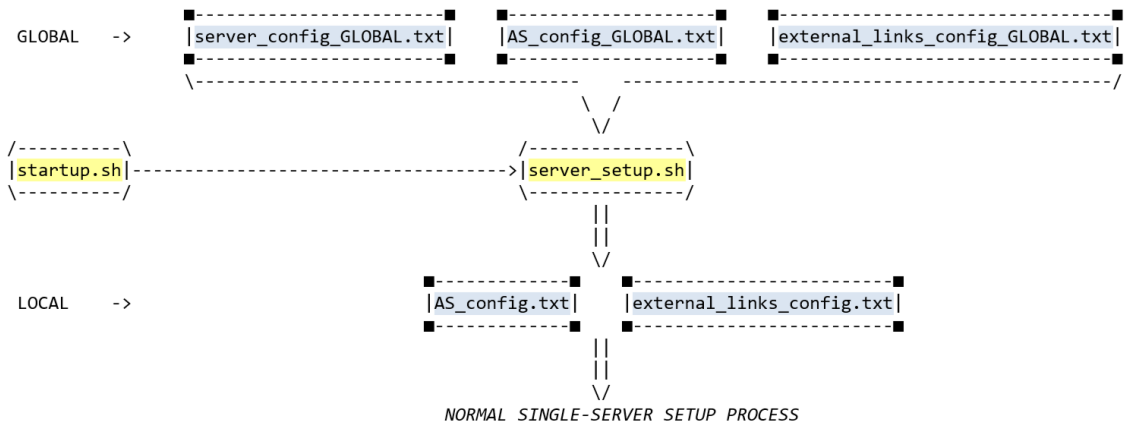

Figure 3.1: Visualization of the processing of the global configuration files to local configuration files and the interconnection of specific scripts and the configuration files.

The global server configuration is a completely new file consisting of a list of all servers. For each server, the information necessary for the creation of remote external links e.g., network interface and IP address are specified. Further, the information to establish an SSH connection for maintenance is specified as well.

Finally, when the startup script is started, a new server setup script checks if all the necessary configurations for a multi-server mini-Internet are available. If this is not the case, the startup script will proceed as before and start creating a single-server mini-Internet according to the local AS and local external links configurations. If indeed a global configuration is recognized, the server setup scripts open SSH connections to all the other servers, copy all the files needed to them and start the startup process there as well. It then automatically creates correct local AS and local external links configuration files. Afterwards all the setup routines can run as if it were a single-server mini-Internet just now according to the generated local configuration files.

This procedure we have chosen here guarantees us that we can continue using the mini-Internet in the same way as before. We now just have the additional option to create GLOBAL configurations instead of LOCAL configurations and then automatically deploy our mini-Internet over multiple servers.

Remote external links. If a GLOBAL configuration is running, we have a new type of links. We call them "remote external links" meaning links between ASes which run on different servers. These special links are indicated in the autogenerated local external links configuration file and treated separately during the setup process. They constitute the actual key part of a distributed mini-Internet.

For each server pair a GRE tunnel in the Linux IP network stack and an OvS bridge are automatically created. Then a GRE port corresponding to the created GRE tunnel is added to each OvS bridge. You can see this visualized in the figure below. In this example for the remote connection between VM1 and VM3, an OvS bridge called "remote-VM3" is created, and a GRE tunnel called "greVM1-VM3" specifying 10.0.0.1 as the local tunnel endpoint and 10.0.0.3 as the remote tunnel endpoint is created. Afterwards a GRE type port "greVM1-VM3" is added to the OvS bridge "remote-VM3" to connect the OvS bridge to the GRE tunnel. On VM3 the implementation is mirrored. We now have OvS bridges on each VM which are connected by a GRE tunnel.

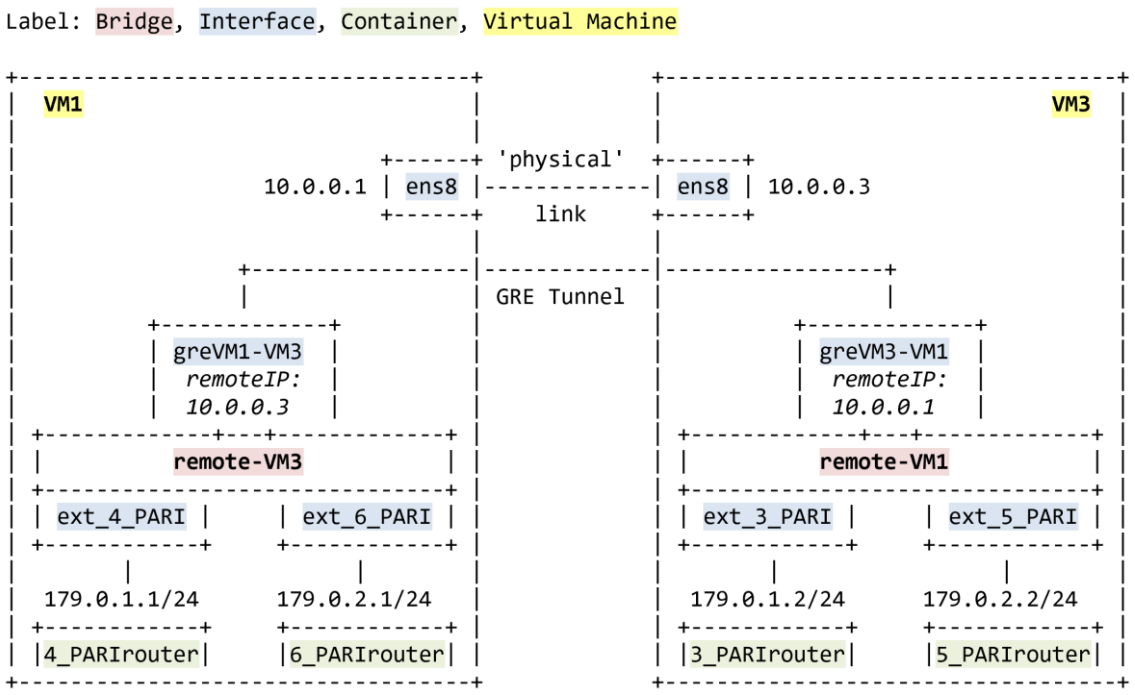

Figure 3.2: Remote external links implementation visualized. You can see the OvS bridges (red) on each VM which are connected by a GRE tunnel.

For every external link it is indicated in the configuration files between which servers the connection needs to be established. For example, the peer link between the

PARIS router of AS 5 and the PARIS router of AS 6 goes from VM1 to VM3. Thus, the PARIS router of AS 5 will be connected to the remote-VM3 bridge with a virtual ethernet link (veth link) and the PARIS router of AS 6 will be connected to the remote-VM1 bridge with a veth link. The same is done for each external link which needs a connection between VM1 and VM3. As a result, all the endpoints of these external links connected to the same OvS bridge are now able to communicate with each other. As this should not be possible, we need to implement a way to isolate the different connections which use the same GRE tunnel.

We make use of OpenFlow rules on the OvS bridges and the TOS (type of service) field of the IP header to achieve isolation between the different external links. The IPv4 TOS field consists of 8 bits where the 2 bits used for ECN (Explicit Congestion Notification) must be cleared to zero [10]. OvS OpenFlow implementation can access the TOS field and manipulate it as well. As only the upper 6 bits can be unequal to zero, we can set every multiple of 4 between 0 and 255 as the TOS value (0, 4, 8, 12, 16, …) when using OpenFlow rules.

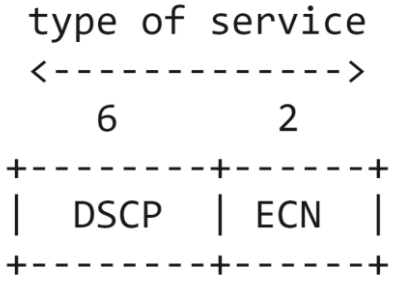

Figure 3.3: IPv4 TOS field structure.

This means that e.g., for a packet coming from the interface ext_4_PARI a TOS value is assigned on remote-VM3, and it is directed to the GRE interface. When arriving at the remote-VM1 bridge the packet incoming from the GRE interface is matched by the TOS value, the TOS value will be reset to zero and then the packet will be directed to the interface ext_3_PARI. We assign the TOS values by multiplying the number of the external links by 4. This results in having a unique TOS value for each external link but also limiting the number of external links supported by this implementation to 64. In most cases this will be enough as normally not each AS on one server has a

direct link to an AS on another server. Of course, using the TOS field this way prevents the current specified usage of the TOS field for DSCP. Nevertheless, the more commonly used ECN part is not affected and thus the tradeoff is worthwhile. With the OpenFlow rules in place to assign each link its unique TOS value and directing the traffic only to where it is supposed to go, we successfully isolated the different external links using the same GRE tunnel.

As the TOS field is part of the IP header and thus only available to IP traffic, we implemented additional OpenFlow rules for handling ARP packets. ARP is needed at the beginning when the routers not yet know the MAC address of the directly connected router which is on a different server. Hence, every incoming ARP packet from a veth interface of the remote-VM1 bridge will be forwarded to the GRE interface, and the ARP packets coming from the GRE interface are flooded to all the veth interfaces. The same applies for the remote-VM3 bridge. Alternatively, to this solution we could think of statically adding entries to the ARP table on each router connected to a router on another server or adding OvS rules which match the source MAC address. However, especially the first approach would greatly limit which IPs we can define on these interfaces.

You have seen how a single-connection between two servers is implemented. The same is done for all the pairs of servers. In case you have 4 VMs and all have connections to each other in a full mesh style, you would have three OvS “remote” bridges enabling to set up external links between ASes on different servers.

As we use GRE tunnels, the communicating routers on each server do not realize that there is anything different; it is as if they would run on the same server. Thus, we achieved our objective that for the user of the mini-Internet nothing has changed.

4 Results

In this section we discuss how the newly implemented functionality has been tested and show that our solution for a multi-server deployment works. We show how a multi-server mini-Internet performs compared to a single-server one.

Testing configuration and setup. For all the following tests we used the same mini-Internet configuration you can see illustrated below. Once run on a single VM with 20 cores at 2.3 GHz and once run distributed over 4 VMs with each 5 cores at 2.3 GHz. All machines run on Ubuntu 20.04 LTS and the 5.4.0 Linux Kernel. Thus, the 4 VMs together should have the same amount of computational power available as the single VM. Memory wise the single VM has about 37 GB available and the 4 VMs each have 8 GB available.

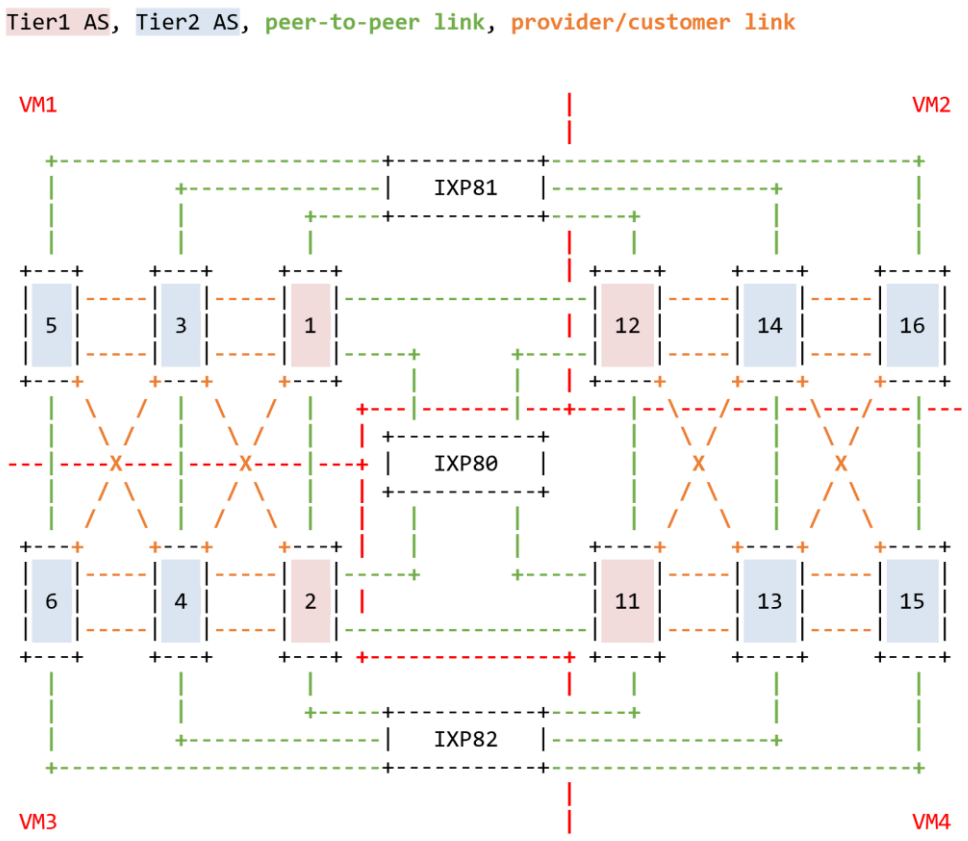

Figure 5.1: The blue and red boxes symbolize ASes, the green and orange lines indicate external links and the red lines show how the configuration is split up over the different servers (VMs).

Functionality. We run multiple traceroutes from a host inside one AS to the corresponding host on another AS. We did the identical traceroutes on the single-server setup and on the multi-server setup. As you can see by the example below the route taken between the ASes is the same whether you use a single-server or a multi-server setup. If you would like to see more traceroutes you can find the testing data in the git repository mentioned in the introduction. The difference you can see between these two traceroutes lays in the intra AS routes taken. They change when you run traceroute multiple times as there is no single preferred route due to load balancing configured in the intra AS network.

Reference single-server setup

```
root@staff_3:/# traceroute -n 16.200.20.5
traceroute to 16.200.20.5 (16.200.20.5), 30 hops max, 60 byte
packets
 1  5.200.20.1  6.669 ms  6.587 ms  6.552 ms
 2  5.0.2.2  8.637 ms  8.604 ms 5.0.3.1  5.653 ms
 3  5.0.12.1  6.789 ms  6.758 ms  6.727 ms
 4  180.81.0.16  12.949 ms  12.915 ms  12.884 ms
 5  16.0.12.2  11.104 ms 16.0.8.1  13.276 ms 16.0.12.2  11.089 ms
 6  16.0.3.2  12.192 ms 16.0.1.1  15.415 ms 16.0.2.1  15.253 ms
 7  16.200.20.5  21.511 ms  21.476 ms  19.365 ms
```

Tested multi-server setup

```
root@staff_3:/# traceroute -n 16.200.20.5
traceroute to 16.200.20.5 (16.200.20.5), 30 hops max, 60 byte
packets
 1  5.200.20.1  9.807 ms  9.589 ms  9.483 ms
 2  5.0.3.1  7.914 ms  7.859 ms  7.819 ms
 3  5.0.12.1  8.371 ms  8.237 ms  8.217 ms
 4  180.81.0.16  15.126 ms  15.079 ms  15.045 ms
 5  16.0.12.2  17.027 ms  16.993 ms 16.0.8.1  16.993 ms
 6  16.0.9.1  14.931 ms 16.0.2.1  16.861 ms 16.0.1.1  16.777 ms
 7  16.200.20.5  23.304 ms  22.687 ms  22.689 ms
```

Figure 5.2: Traceroute from AS 5 host at 5.200.20.5 to AS 16 host at 16.200.20.5. Identical traceroute once on the single-server setup and once on the multi-server setup.

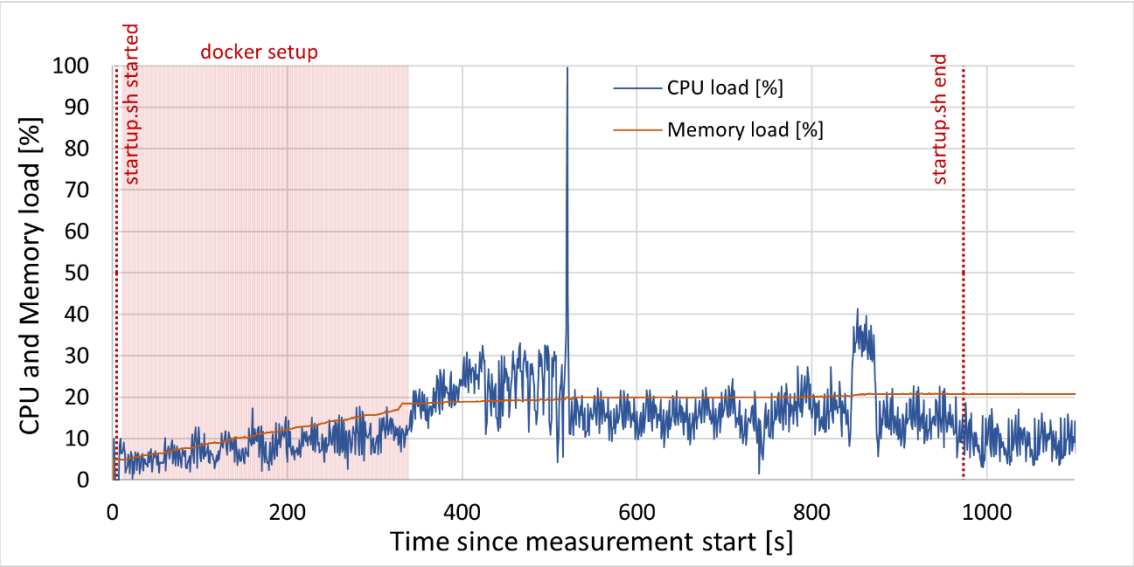

Figure 5.3: CPU and Memory load during startup process on a single-server mini-Internet setup. The blue line indicates the CPU load [%], the orange line indicates the memory load [%].

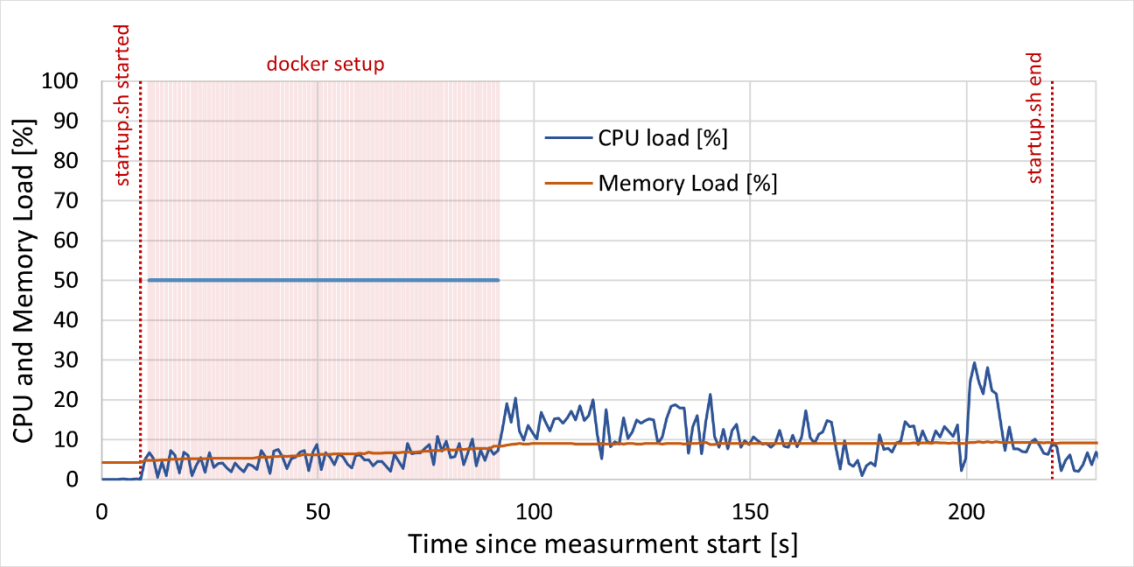

Figure 5.4: CPU and Memory load during startup process on a single-server mini-Internet setup whit a local configuration from a multi-server mini-Internet running. The blue line indicates the CPU load [%], the orange line indicates the memory load [%].

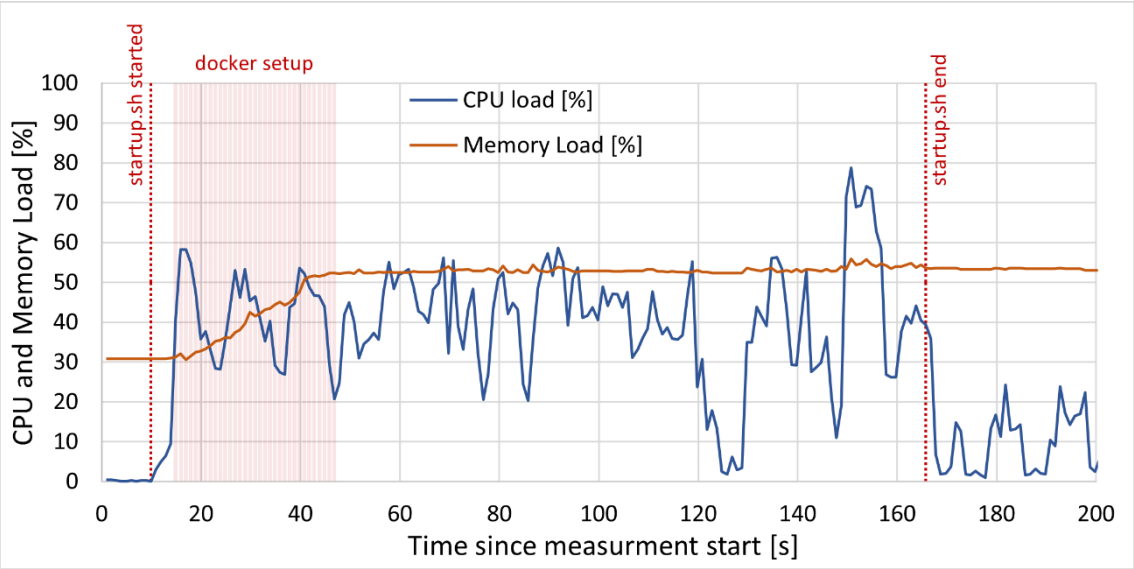

Figure 5.5: CPU and Memory load during startup process on an individual server in a multi-server mini-Internet setup.

System performance compared. You can recognize when comparing figure 5.3 and figure 5.5 that the setup process of the multi-server mini-Internet is about six times faster than the single server setup (156s vs. 969s). The multi-server deployment splits the configuration evenly over all four VMs and then runs the startup process in parallel on all of them. The single-server setup in contrast has a very low CPU load over the whole process and thus is not able to make use of the additional available cores. You can see in figure 5.4 that the single-server setup even takes longer to do the same task than the single VM in the multi-server setup. This again results from not leveraging the additional available cores as well as the memory intensive docker setup running slower. Therefore, it comes as no surprise that the multi-server mini-Internet is much faster initialized than the single-server one. Memory in general is not a limiting factor for a mini-Internet setup at all. In the single-server case as well as in the multi-server case a rather small amount of memory is available (34GB and 4x8GB) but this does not seem to be a problem as only about 15%–25% of the memory is actually used for the configuration. The figures 5.3 and 5.5 show very well that the memory usage primarily depends on the number of containers created for the setup (see reddish highlighted “docker setup” phase). Hence, with a significantly increased amount of docker containers i.e., components in the mini-Internet, you may still be able to use this rather small amount of memory or easily allocate more memory to the system you are using.

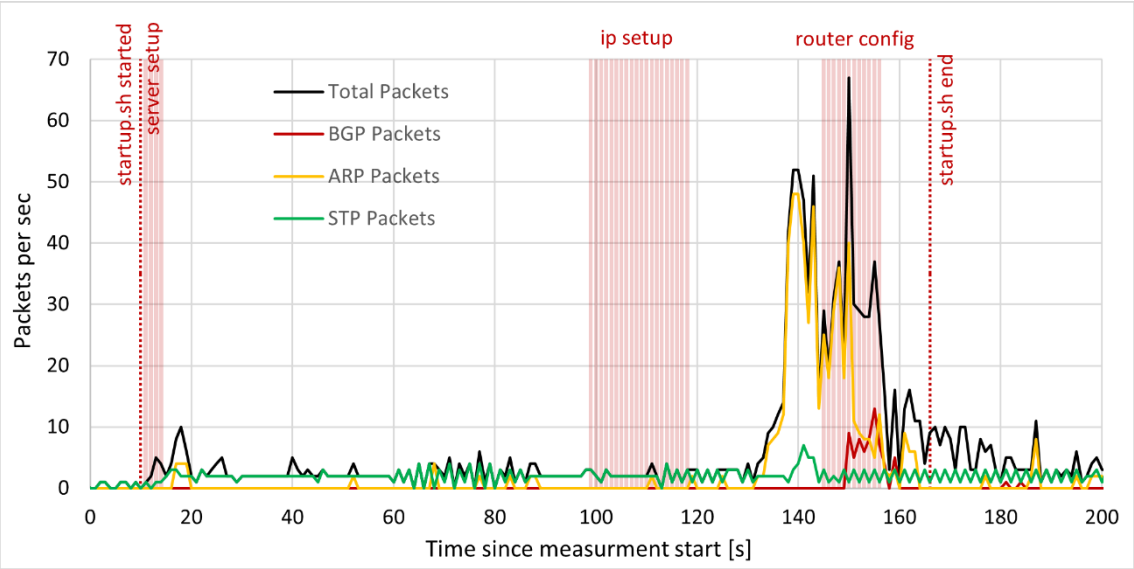

Figure 5.5: Network traffic on the interface of one VM used to communicate with the other VMs in a multi-server mini-Internet deployment. The black line indicates the total number of packets sent or received over the interface. Yellow indicates ARP (Address Resolution Protocol) packets, green STP (Spanning Tree Protocol) packets, and red BGP (Border Gateway Protocol) packets.

Network traffic between the servers. During the server setup process the OvS bridges with the GRE tunnel to all the other servers are created. After this one can see a baseline STP "Hello message" traffic between these bridges connected with a GRE tunnel. Then one can see that the IP setup, where the links between all the components are created, does not start any communication between the servers as the routers are not yet configured. During the router configuration process when the BGP sessions are established, we see a small increase of BGP packets. As there are not many ASes in this configuration and very small delays between the servers, BGP seems to have converged just after the routers are configured.

Most traffic arises from ARP packets. Especially due to identical ARP packets being sent multiple times. This is not a behavior we would expect and needs further investigation in case this really should happen. But even though we have unexpected ARP traffic, there is not a lot of traffic happening between the servers during startup and when the mini-Internet is idle.

5 Deploy a mini-Internet over multiple servers

When deploying the mini-Internet over multiple servers in practice there are differences in the process one must consider. Hence this section is dedicated to give guidance for a seamless setup process of a multi-server mini-Internet.

Correct Global Configurations. The key for a correctly set up mini-Internet are correct configuration files. Firstly, the name chosen for the server in the global server configuration must match the specified server name in the global AS configuration. Secondly, the server connections information provided in the global server configuration should be tested in advance. The server which will be used to start the setup process, must have key based SSH access to the other servers. You should use a different interface for the SSH connection than for the connection of the mini-Internet to guarantee connectivity in case of troubles. As you can see in the figure below the `<server_INTERFACE>` and the `<server_IP>` are indicating the interface used to connect to the mini-Internet.

server_config_GLOBAL.txt

<server_NAME>	<server_INTERFACE>	<server_IP>	<ssh_CONNECTION>	<ssh_PORT>	<sudo_PASSWORD>
VM1	ens8	10.0.0.1/24	alex@pisco.ethz.ch	4020	alex

AS_config_GLOBAL.txt

<AS_number>	<AS_type>	<AS_config>	<router_config>	<internal_links_config>	<…>	<server_NAME>
1	AS	Config	router_config_small.txt	internal_links_config_small.txt	…	VM1

Figure 4.1: Special global configuration files and their composition

Cleaning up reliably. Before a mini-Internet configuration can be started and deployed over multiple servers, you should remove possible old configuration on the servers. Otherwise, unexpected malfunctions can arise. Further, we recommended to comment out the start of the cleanup.sh script in the startup.sh script, as it is not yet fully capable of handling multi-server setups. Instead, you should use the

hard_reset_GLOBAL.sh script to clean a server. ATTENTION: The global hard reset might not only clean configuration created on the system by the mini-Internet but all configurations on the system related to e.g., Docker containers, OvS and IP tunnels. Thus, we strongly recommend running the mini-Internet in a VM or a server where nothing else is running.

There are two ways to use the hard_reset_GLOBAL.sh. You can start it on one server and add some arguments to also clean all the other servers specified in the server_config_GLOBAL.txt.

```
sudo ./hard_reset_GLOBAL.sh <server_NAME> master
```

Alternatively, you can run it without any additional arguments, and it only cleans the server where it was started. In this case you also have to remove the folder of the mini-Internet containing the configurations manually if a full clean is needed.

```
sudo ./hard_reset_GLOBAL.sh
```

Additional startup arguments. To start a deployed mini-Internet one must provide some arguments with the startup script. The first one specifies the name of the server where the startup script is started, and the second argument must be “master” to specify that this server starts the setup process on the other servers.

```
sudo ./startup.sh <server_NAME> master
```

In case you only want to install the local part of a multi-server mini-Internet, you can leave out the word “master”.

If you want to start just a single-server mini-Internet, it can be done as before by running the startup script without any argument.

```
sudo ./startup.sh
```

6 Summary

By adding the option to deploy the mini-Internet over multiple servers new possible use cases arise for the mini-Internet. It now can scale better and it might be used with bigger classes for teaching or research projects requiring more complex and detailed networks for simulation purposes. Performance wise a multi-server deployment needs a significantly smaller amount of time for the setup process, as it runs parallel on each server. With this rather small mini-Internet configuration used for testing the network traffic between the servers poses no further limitations.

Invisible scaling for the user. Using GRE tunnels enabled us to create an overlay network and thus a deployment over multiple servers is invisible to the user of the mini-Internet. Moreover, for people already familiar with the mini-Internet the additional creation of global configuration files should be easy as it is very similar to the previous process of creating configurations. This is also due to the high grade of automation which was implemented and reduces the amount of information which must be specified additionally in the global configurations.

Backward compatibility. With the implementation chosen in this project we were able to achieve that one can still use the mini-Internet in the same way as before this upgrade. This has multiple advantages: In many cases a multi-server deployment will not be necessary and only creating local configurations needs less time and therefore one might prefer to use the classical single-server setup procedure. Furthermore, as the multi-server deployment is not yet as rich in features as the single-server version, one might like to have the single-server setup option besides running multi-server deployments. Preferably this should be possible without having to maintain two sets of code. Thanks to the backward compatible implementation this is achieved.

Improve connection reliability. With the current implementation of handling, the ARP with OpenFlow seems to have convergence difficulties. This results in temporary loss of connection between ASes (ping) but not in downtime of BGP sessions. As this can be rather annoying, further investigations and improvements would be beneficial to most users of the mini-Internet.

Upgrade monitoring and debugging tools. When used for teaching the monitoring and debugging tools such as the connectivity matrix or the measurement platform play a crucial part in making configuring the mini-Internet a less frustrating experience for the students. With the multi-server deployment as it is implemented now, the tools currently are limited to monitor each server on its own.

Improved cleanup. In case the mini-Internet shall be deployed on servers where already docker containers are running or OvS bridges are used, the currently available global hard reset is not a sufficient solution, as it would unintentionally clear them too. A cleanup routine as it is used for the single-server setup adapted to a multi-server deployment would be a better solution. This should be easily implementable by merging the functionality of the single-server cleanup with the global hard reset procedure.

References

[1] Thomas Holterbach, Tobias Bühler, Tino Rellstab, Laurent Vanbever, *An Open Platform to Teach How the Internet Practically Works*, 2020

[2] The mini-Internet project. http://mini-inter.net/, 26.05.2021

[3] The Linux man-pages project, https://man7.org/linux/man-pages/man7/namespaces.7.html, 26.05.2021

[4] The Linux man-pages project, https://man7.org/linux/man-pages/man4/veth.4.html, 26.05.2021

[5] The Linux Foundation, https://www.openvswitch.org/, 26.05.2021

[6] The Linux Foundation, https://www.openvswitch.org/features/, 26.05.2021

[7] FRRouting Project, https://frrouting.org/, 26.05.2021

[8] IETF, https://datatracker.ietf.org/doc/html/rfc1701, 26.05.2021

[9] IETF, https://datatracker.ietf.org/doc/html/rfc2890, 26.05.2021

[10] https://man7.org/linux/man-pages/man7/ovs-fields.7.html,