INSTITUT FÜR INFORMATIK

ragged2e verbatim

DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Ausarbeitung

zum

Rechnernetzepraktikum

im WS 24/25

Aufgabenblock 1

Gruppe: 05

Zeyu Li

Hanzhang Ma

Contents

1 A100 Adressierung und Wegewahl (Theorie)

i) Beschreiben Sie kurz die Bedeutung des Felds TTL im IPv4-Header und erklären Sie dabei, wie es von Schicht 3-Komponenten benutzt wird!

Antwort:

The TTL (Time to Live) field in the IPv4 header is used to limit the time to live of a packet in the network. It initially indicates the maximum number of routers that a packet can pass through, preventing an infinite loop in the network. The initial value of the TTL field is usually set by the sender (e.g., 64 or 128) and is reduced by 1 each time it passes through a router (i.e., a Layer 3 device). Once the TTL is reduced to 0, the router drops the packet and typically returns an ICMP "timeout" message to the sender.

When a Layer 3 device, such as a router, receives a packet, it checks the TTL field first. Then the TTL value decreases by 1 for each router that passes through; if the TTL value is reduced to 0, the packet has reached the hop limit. Packets with a TTL of 0 are dropped to avoid looping in the network. The router may send an ICMP Timeout message to the source host informing the sender that the packet failed to reach its destination.

This mechanism effectively prevents packets from looping indefinitely in the network and improves the reliability of the network.

ii) Erklären Sie kurz das Verfahren der Unterteilung des IPv4-Adressraums in Klassen!

Antwort:

The process of classifying IPv4 address space is to distinguish different address classes by the first few bits of the address, so as to adapt to the needs of networks of different scales. An IPv4 address is an address with a length of 32 bits (4 bytes). IPv4 addresses are classified into 5 categories:

Type A addresses have an address range from 0.0.0.0 to 127.255.255.255. They are characterized by a first byte that begins with 0, meaning the first bit is 0. In Type A addresses, the first 8 bits (the first byte) are used to identify the network, while the remaining 24 bits are used to identify the host. These addresses are suitable for large-scale networks, such as national or large institutional networks, with each network capable of supporting over 16 million hosts.

Type B addresses range from 128.0.0.0 to 191.255.255.255, with the first byte starting with 10 (the first bit pattern is 10). In this address type, the first 16 bits are used for network identification, and the remaining 16 bits for host identification. Type B addresses are suitable for medium-sized networks, each capable of accommodating approximately 65,000 hosts.

Type C addresses fall within the range of 192.0.0.0 to 223.255.255.255, with the first byte beginning with 110 (indicating the first bit pattern is 110). Here, the first 24 bits are used to identify the network, and the last 8 bits identify the host. Type C addresses are ideal for small networks, with each network supporting up to 254 hosts.

Type D addresses, ranging from 224.0.0.0 to 239.255.255.255, are identified by a first byte that starts with 1110. These addresses are used exclusively for multicast and do not identify individual networks or hosts.

Finally, Type E addresses, with a range from 240.0.0.0 to 255.255.255.255, start with the bit pattern 1111. These addresses are reserved for experimental purposes and are not intended for public network communication.

Through this division design, IPv4 can adapt to the needs of networks of different scales and avoid the waste of address space. This partitioning method allows the network to flexibly select A, B, and C addresses according to actual needs, so as to achieve efficient management of address resources.

iii) Welche Probleme / Nachteile wurden durch die Einführung von CIDR behoben?

Antwort:

Classless Inter-Domain Routing (CIDR) is an address allocation and route aggregation method that does not use traditional Class A, B, and C partitions, and uses the notation of "IP address/prefix length" (e.g., 192.168.1.0/24). The introduction of CIDR mainly solves the problems of IPv4 address space waste and route table bloat.

CIDR can solve these problems: **1. Solve the waste of address space** Under the division of A, B, and C, the allocation of address space is inflexible, resulting in a large number of addresses being wasted. If a network requires 500 hosts, using Class B addresses would result in a significant waste of addresses, while using Class C addresses would be insufficient to meet the demand. CIDR allows flexible allocation of address blocks based on actual needs, providing more accurate subnet allocation. For example, you can use a /27 prefix to assign 32 addresses, or a /30 prefix to assign 4 addresses, reducing address space waste and improving address utilization.

2. Reduce routing table bloat In classified networks, each independent network needs a separate route, which will cause the routing table to bloat rapidly on the Internet, increasing the storage and computing pressure on the router. CIDR uses "route aggregation" or "supernetting" to reduce the size of the routing table. It allows multiple consecutive IP prefixes to be merged into one larger CIDR block for advertising. For example, multiple contiguous /24 networks (e.g., 192.168.0.0/24 to 192.168.3.0/24) can be aggregated into a single /22 network (192.168.0.0/22), reducing the number of entries in the routing table and optimiz-

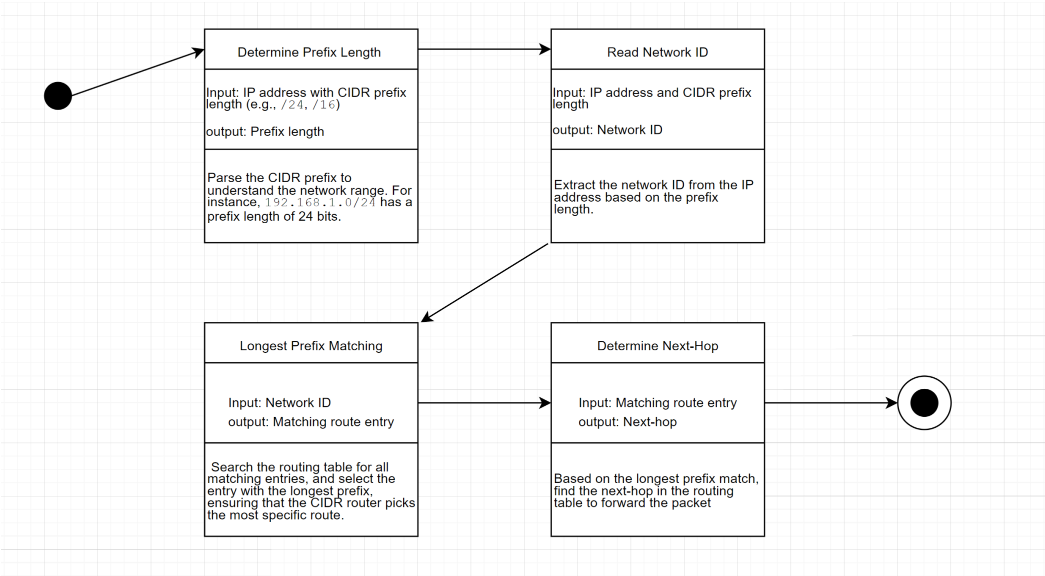


Figure 1.1: Automaton

ing routing efficiency

The disadvantages of CIDR are mostly about computation costs. CIDR makes IP addressing and subnetting more flexible, but it requires network administrators to have more expertise in subnet planning to avoid conflicts and wastage. We also found that, some older network devices and software may not support CIDR notation or handle variable-length subnet masks correctly, particularly when integrating with legacy systems. At the same time, the route aggregation and classless subnetting features of CIDR require routers to perform more calculations, especially for longest-prefix matching, which increases computational demands.

iv) Erstellen Sie analog zu Abbildung 1.6 einen Automaten zur Wegewahl für CIDR!

Antwort:

The CIDR automaton is designed to enable path selection using flexible subnet masks. First, the automaton receives the full IP address and CIDR prefix length as input, and it parses the CIDR prefix to determine the subnet range. Then, it separates the IP address and the parsed prefix length to read the network ID. Next, the automaton performs longest prefix matching in the routing table to find the most suitable route entry. Finally, based on the match result, it determines the next hop and forwards the packet toward its destination.

The automaton can be found at figure1.1. The figure has been uploaded to the website

2 A101 Topology Discovery

Kapitel beschreibt die virtuelle Infrastruktur, die Ihnen zur Verfügung steht und wie Sie Zugang zu Ihren virtuellen Maschinen erlangen.

Erstellen Sie einen Netzplan der Ihnen zur Verfügung stehenden virtuellen Maschinen!

i) Konfigurieren Sie Ihre virtuellen Maschinen mit IPv4 Adressen aus Ihrem Subnetz! Benutzen Sie dafür den Befehl `ip` (Anmerkung: `ip help`)

Antwort:

We used the `ip addr add` command to assign an IP address to the device's Ethernet interface. This command configures the specified IP on the desired network interface, making it available for network communication. Below is the relevant output from the `ip addr` command, with unrelated information omitted for clarity.

```
root@router1:~# ip link set dev eth2 up
root@router1:~# ip addr add 10.5.1.1/24 dev eth2
root@router1:~# ip addr show
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
↪  state UP group default qlen 1000
    link/ether 00:16:3e:00:00:09 brd ff:ff:ff:ff:ff:ff
    inet 10.5.1.1/24 scope global eth2
       valid_lft forever preferred_lft forever
    inet6 fe80::216:3eff:fe00:9/64 scope link
       valid_lft forever preferred_lft forever

root@router2:~# ip link set dev eth2 up
root@router2:~# ip addr add 10.5.1.2/24 dev eth2
root@router2:~# ip addr show
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
↪  state UP group default qlen 1000
    link/ether 00:16:3e:00:00:14 brd ff:ff:ff:ff:ff:ff
    inet 10.5.1.2/24 scope global eth2
       valid_lft forever preferred_lft forever
    inet6 fe80::216:3eff:fe00:14/64 scope link
       valid_lft forever preferred_lft forever
```

ii) Ermitteln Sie Verbindungen zwischen zwei VMs, indem Sie mittels `ping`-Befehl Daten zwischen diesen hin und her schicken! (Anmerkung: `man ping`)

Antwort:

```
root@router1:~# ping -I eth2 10.5.1.2
PING 10.5.1.2 (10.5.1.2): 56 data bytes
64 bytes from 10.5.1.2: seq=0 ttl=64 time=1.148 ms
64 bytes from 10.5.1.2: seq=1 ttl=64 time=0.301 ms
64 bytes from 10.5.1.2: seq=2 ttl=64 time=0.407 ms
64 bytes from 10.5.1.2: seq=3 ttl=64 time=0.371 ms
64 bytes from 10.5.1.2: seq=4 ttl=64 time=0.319 ms
64 bytes from 10.5.1.2: seq=5 ttl=64 time=0.347 ms
64 bytes from 10.5.1.2: seq=6 ttl=64 time=0.321 ms
64 bytes from 10.5.1.2: seq=7 ttl=64 time=0.324 ms
64 bytes from 10.5.1.2: seq=8 ttl=64 time=0.325 ms
64 bytes from 10.5.1.2: seq=9 ttl=64 time=0.281 ms
^C
--- 10.5.1.2 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 0.281/0.414/1.148 ms
```

iii) Erstellen Sie einen Netzplan, der Ihre Konfigurationen und Ergebnisse widerspiegelt!

Antwort:

To determine which interfaces are connected to each other, we developed a script that verifies connectivity by testing if two Ethernet interfaces can successfully ping each other.

The script operates as follows: The script takes two parameters, specifying the types of the sender and receiver devices. Here, "sender" and "receiver" refer to network devices, such as router or pc. The script begins by powering on the devices involved in the test. It then assigns IP addresses to both the sender and receiver devices to establish a unique network configuration for each.

Once configured, the script initiates a ping command from the sender to the receiver, measuring packet loss to assess connectivity. Finally, the script captures and outputs the results, showing whether the devices are able to communicate over the tested interfaces.

The output of the script is shown as follows. We paste the full script in the appendix. You can find the script on the website as well: `https://gitea.mhrooz.xyz/iicd/RN/src/branch/main/Blatt01/scripts/testpc.sh`

```
rnp@Gruppe05:~$ bash testpc.sh router router
#Script to test the router to router connection status
results
sender #   sender eth #    receiver #   receiver eth #  losses #
1          2               2            2               20
```

```
1           3               3            2               20
1           4               4            1               0
2           2               1            2               20
2           3               3            3               0
2           4               4            2               0
3           2               1            3               0
3           3               2            3               0
3           4               4            3               0
4           1               1            4               0
4           2               2            4               0
4           3               3            4               0
rnp@Gruppe05:~$ bash testpc.sh router pc
#Script to test the router to pc connection status
results
sender #   sender eth #    receiver #   receiver eth #  losses #
1           1               1            1               0
2           1               2            1               20
3           1               3            1               0
rnp@Gruppe05:~$ bash testpc.sh pc router
#Script to test the pc to router connection status
results
sender #   sender eth #    receiver #   receiver eth #  losses #
1           1               1            1               0
2           1               2            1               0
3           1               3            1               0
```

Using these results, we constructed the network topology shown in Figure 2.1. Each device in the network can successfully ping at least one other device, indicating that with a correctly configured routing table, full connectivity can be achieved across all devices in the network.

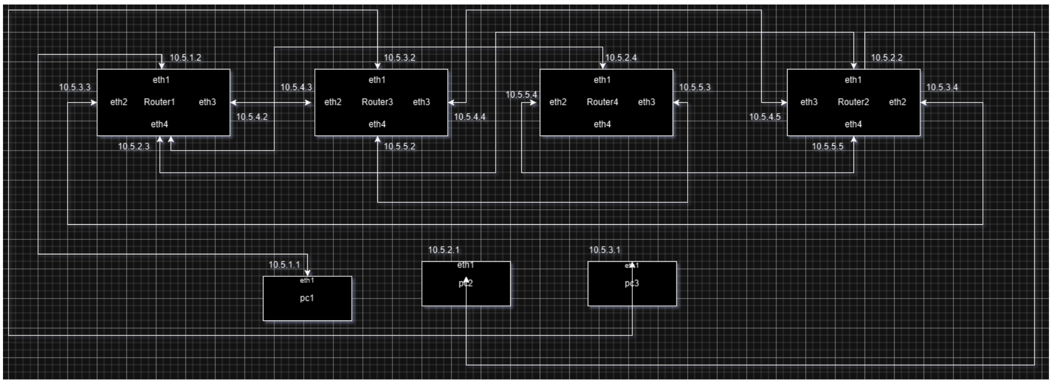


Figure 2.1: A101 iii) Network Plan

3 A102 Fehlerdiagnose mit `tcpdump`

Benutzen Sie in dieser Aufgabe `tcpdump` um ICMP “echo request” PDUs sichtbar zu machen. Starten Sie dazu einen `ping` zwischen `pc1` und `router1`. Starten Sie nun auf einem der beiden Rechner `tcpdump`, um die ICMP PDUs mitlesen zu können.

Erläutern Sie die Ausgabe von `tcpdump` wenn ...

i) ... Sie **nicht** die Option `-e` angeben.

Antwort:

```
root@pc1:~# tcpdump -i eth1
tcpdump: verbose output suppressed, use -v[v]... for full protocol
↪  decode
listening on eth1, link-type EN10MB (Ethernet), snapshot length
↪  262144 bytes
21:54:52.787780 IP 10.5.1.1 > 10.5.1.2: ICMP echo request, id 51307,
↪  seq 1, length 64
21:54:52.789878 IP 10.5.1.2 > 10.5.1.1: ICMP echo reply, id 51307,
↪  seq 1, length 64
21:54:53.789472 IP 10.5.1.1 > 10.5.1.2: ICMP echo request, id 51307,
↪  seq 2, length 64
21:54:53.790142 IP 10.5.1.2 > 10.5.1.1: ICMP echo reply, id 51307,
↪  seq 2, length 64
21:54:54.797728 IP 10.5.1.1 > 10.5.1.2: ICMP echo request, id 51307,
↪  seq 3, length 64
21:54:54.799794 IP 10.5.1.2 > 10.5.1.1: ICMP echo reply, id 51307,
↪  seq 3, length 64
21:54:57.965328 ARP, Request who-has 10.5.1.2 tell 10.5.1.1, length
↪  28
21:54:57.965838 ARP, Reply 10.5.1.2 is-at 00:16:3e:00:00:08 (oui
↪  Unknown), length 28
21:54:58.039126 ARP, Request who-has 10.5.1.1 tell 10.5.1.2, length
↪  28
21:54:58.039142 ARP, Reply 10.5.1.1 is-at 00:16:3e:00:00:02 (oui
↪  Unknown), length 28
```

From the `tcpdump`’s output, we can tell that there are two parts in it: ICMP and ARP parts.

For the ICMP part, it shows that every approximately 1 second, 10.5.1.1 sends an ICMP Echo Request (Ping Request) to 10.5.1.2, specifying id 51307. `seq` represents the sequence number, incremented from 1 (seq=1, seq=2...), that identifies each request. which is used to identify each request. 10.5.1.2 receives the request and replies with an ICMP echo reply to 10.5.1.1. `length 64` means the length of the ICMP packet (64 bytes). With these `ICMP echo request` and `ICMP echo reply`, we can confirm that the network between 10.5.1.1 and 10.5.1.2 is open.

For the ARP part, at 21:54:57.965328, 10.5.1.1 broadcasts an ARP request to the network asking “Who is 10.5.1.2?” and informs itself that it is 10.5.1.1. 10.5.1.2 replied to the ARP request, informing 10.5.1.1 that its own MAC address is 00:16:3e:00:00:08. Immediately after that, we can see that 10.5.1.2 sent a similar ARP request to 10.5.1.1, asking for 10.5.1.1’s MAC address. 10.5.1.1 replies to the request, informing that its MAC address is 00:16:3e:00:00:02.

ii) . . . Sie die Option `-e` angeben.

Antwort:

The `tcpdump` command’s output is as follows:

```
root@pc1:~# tcpdump -i eth1 -e
tcpdump: verbose output suppressed, use -v[v]... for full protocol
↪  decode
listening on eth1, link-type EN10MB (Ethernet), snapshot length
↪  262144 bytes
14:28:59.207532 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02
↪  (oui Unknown), ethertype IPv4 (0x0800), length 98: 10.5.1.1 >
↪  10.5.1.2: ICMP echo request, id 8384, seq 23, length 64
14:28:59.207566 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08
↪  (oui Unknown), ethertype IPv4 (0x0800), length 98: 10.5.1.2 >
↪  10.5.1.1: ICMP echo reply, id 8384, seq 23, length 64
14:29:00.207639 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02
↪  (oui Unknown), ethertype IPv4 (0x0800), length 98: 10.5.1.1 >
↪  10.5.1.2: ICMP echo request, id 8384, seq 24, length 64
```

The biggest difference between with or without `-e` option is that these entries above show the source MAC address and the destination MAC address. For the first entry, `00:16:3e:00:00:08` is the source MAC address, which corresponds to IP address `10.5.1.1`. `00:16:3e:00:00:02` corresponds to `10.5.1.2` respectively.

`ICMP echo request` or `ICMP echo reply` indicates the type of ICMP. `id` is the identifier of the ICMP packet, used to match requests and replies. `seq` is the sequence number of the ICMP packet. `seq=23` indicates that this is the 23rd `Ping` request, and is usually used to mark the order of Pings. `length` is the payload length of the ICMP packet. These are basically the same with the no `-e` option.

iii) Führen Sie nun auf `pc1` den Befehl `ifup eth1` aus. Damit erhält der Rechner die IP-Adresse `172.16.1.100`. Vergeben Sie auf dem entsprechenden Interface von `router1` die IP-Adresse `172.16.1.1`. Wiederholen Sie den `ping`-Vorgang mit diesen Adressen.

Antwort:

`Ping` command basically has 100% losses. With these two IP, `Ping` has no response at all.

iv) Verwenden Sie `tcpdump` um den Unterschied zwischen den IP-Adressen zu sehen und erklären Sie warum keine Antwort ankommt.

Antwort:

We started a `ping` command on `router1` and a `tcpdump` command on `pc1`. The `tcpdump`'s output is as follows:

```
tcpdump -i eth1 -e
tcpdump: verbose output suppressed, use -v[v]... for full protocol
↪  decode
listening on eth1, link-type EN10MB (Ethernet), snapshot length
↪  262144 bytes
14:18:42.413051 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02
↪  (oui Unknown), ethertype IPv4 (0x0800), length 98: 172.16.1.1 >
↪  172.16.1.100: ICMP echo request, id 7117, seq 39, length 64
14:18:42.413085 00:16:3e:00:00:02 (oui Unknown) > 00:de:ad:be:ef:00
↪  (oui Unknown), ethertype IPv4 (0x0800), length 98: 172.16.1.100
↪  > 172.16.1.1: ICMP echo reply, id 7117, seq 39, length 64
14:18:43.413174 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02
↪  (oui Unknown), ethertype IPv4 (0x0800), length 98: 172.16.1.1 >
↪  172.16.1.100: ICMP echo request, id 7117, seq 40, length 64
14:18:43.413199 00:16:3e:00:00:02 (oui Unknown) > 00:de:ad:be:ef:00
↪  (oui Unknown), ethertype IPv4 (0x0800), length 98: 172.16.1.100
↪  > 172.16.1.1: ICMP echo reply, id 7117, seq 40, length 64
```

After the comparison of the output of `tcpdump` between the normal IP and the abnormal IP, we observed an issue with the MAC address. The reply from `172.16.1.100` is being sent to an incorrect MAC address, likely due to an outdated or incorrect ARP table entry, which resulted in an incorrect IP-to-MAC address mapping.

v) Korrigieren Sie den Fehler und zeigen Sie dass der `ping`-Vorgang nun erfolgreich ist.

Antwort:

To address the issue encountered previously on pc1, where the reply from `172.16.1.100` was being sent to an incorrect MAC address due to an outdated or incorrect ARP table entry, we used the following command:

```
ip neigh replace 172.16.1.1 lladdr 00:16:3e:00:00:08 dev eth1
```

This command manually updates the ARP table on pc1, replacing the MAC address associated with the IP address `172.16.1.1` to `00:16:3e:00:00:08` on the `eth1` interface.

After the modification of the ARP table, we made the `ping` command on `pc1` again. The new output is:

```
root@pc1:~# ping 172.16.1.1
PING 172.16.1.1 (172.16.1.1) 56(84) bytes of data.
64 bytes from 172.16.1.1: icmp_seq=1 ttl=64 time=1.73 ms
64 bytes from 172.16.1.1: icmp_seq=2 ttl=64 time=0.268 ms
64 bytes from 172.16.1.1: icmp_seq=3 ttl=64 time=0.255 ms
64 bytes from 172.16.1.1: icmp_seq=4 ttl=64 time=0.305 ms
64 bytes from 172.16.1.1: icmp_seq=5 ttl=64 time=0.260 ms

--- 172.16.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4072ms
rtt min/avg/max/mdev = 0.255/0.563/1.731/0.583 ms
```

All packets are transmitted.

4 A103 Statisches Routing

Konfigurieren Sie nun die VMs, um Nachrichten über mehrere Hops zu transportieren. *Hinweis*: Alle Teilaufgaben sollen so gelöst werden, dass IP-Pakete „hin und zurück", d.h. in beide Richtungen zwischen Sender und Empfänger vermittelt werden.

Um die Routing-Tabelle eines Rechners einsehen und administrieren zu können, benutzen Sie das Werkzeug `ip`:

i) Konfigurieren Sie **Host-Routen**, so dass `pc1` und `pc2` Daten austauschen können! Weisen Sie mittels ICMP echo request/reply nach, dass dies der Fall ist! Nehmen Sie die nach erfolgreicher Konfiguration geltenden Routing-Tabellen der beteiligten VMs in Ihre Ausarbeitung auf! Einzelne exemplarische Tabellen reichen, sofern daraus der Ablauf ersichtlich wird.

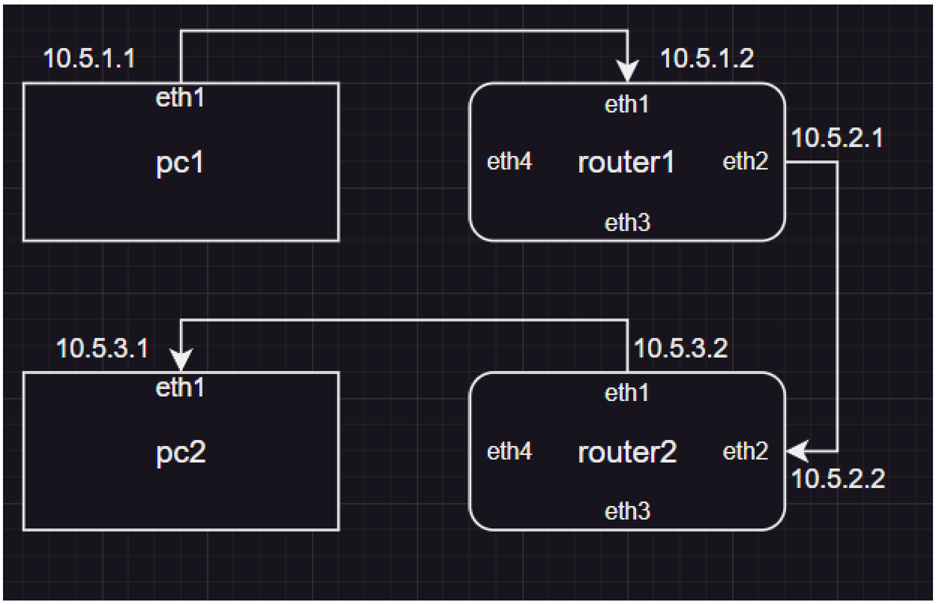
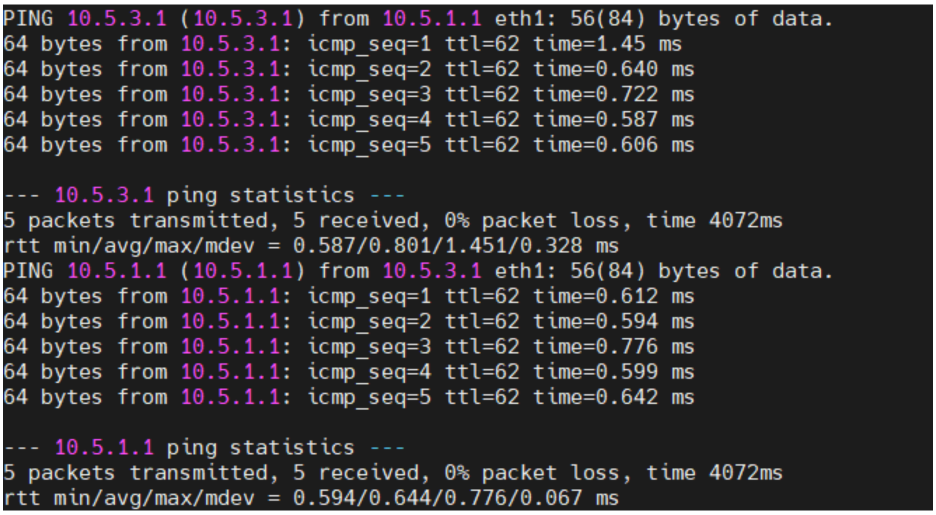


Figure 4.1: A103 i) Network Topology

Antwort: We have implemented a network topology as the figure 4.1. We first show the result of `ping` in the figure 4.2 since the `ping` command uses ICMP Echo Requests and Echo Replies to detect whether a target device is reachable, and network latency. We transmitted 5 packets. You can find that 0% packet loss is reached.
Since we need to use **host route**, we need to specify a `/32` subnet mask to indicate a route for a single IP address only in the `ip route add` command. The route table for each of the four devices is shown as follows:



```
PING 10.5.3.1 (10.5.3.1) from 10.5.1.1 eth1: 56(84) bytes of data.
64 bytes from 10.5.3.1: icmp_seq=1 ttl=62 time=1.45 ms
64 bytes from 10.5.3.1: icmp_seq=2 ttl=62 time=0.640 ms
64 bytes from 10.5.3.1: icmp_seq=3 ttl=62 time=0.722 ms
64 bytes from 10.5.3.1: icmp_seq=4 ttl=62 time=0.587 ms
64 bytes from 10.5.3.1: icmp_seq=5 ttl=62 time=0.606 ms

--- 10.5.3.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4072ms
rtt min/avg/max/mdev = 0.587/0.801/1.451/0.328 ms
PING 10.5.1.1 (10.5.1.1) from 10.5.3.1 eth1: 56(84) bytes of data.
64 bytes from 10.5.1.1: icmp_seq=1 ttl=62 time=0.612 ms
64 bytes from 10.5.1.1: icmp_seq=2 ttl=62 time=0.594 ms
64 bytes from 10.5.1.1: icmp_seq=3 ttl=62 time=0.776 ms
64 bytes from 10.5.1.1: icmp_seq=4 ttl=62 time=0.599 ms
64 bytes from 10.5.1.1: icmp_seq=5 ttl=62 time=0.642 ms

--- 10.5.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4072ms
rtt min/avg/max/mdev = 0.594/0.644/0.776/0.067 ms
```

Figure 4.2: A103 i) Ping Results

```
root@router1:~# ip route show
10.5.1.1 via 10.5.1.2 dev eth1
10.5.3.1 via 10.5.2.2 dev eth2

root@router2:~# ip route show
10.5.1.1 via 10.5.2.1 dev eth2
10.5.3.1 via 10.5.3.2 dev eth1

root@pc1:~# ip route show
10.5.3.1 via 10.5.1.2 dev eth1

root@pc2:~# ip route show
10.5.1.1 via 10.5.3.2 dev eth1
```

Here we hide the `192.168.0.0/24` entry of the route table since it is used for the management machine. For `router1` and `router2`, we must first add the host route and delete the kernel automatical generated route rules, for example, `10.5.3.0/24 dev eth1 proto kernel scope link src 10.5.3.1`. If we delete the rule first and add the host route rule second, the os will give an error: Nexthop has invalid gateway.

ii) Ersetzen Sie die Host-Routen auf `pc1` und `pc2` durch default Routen!

Antwort: We use the same network topology as figure4.1. Since the `ping` command's output is similar to the figure4.2, we just paste the results of the default route in the appendix (See 5.1).
The route table of pc1 and pc2 is as follows, again, we hide the `192.168.0.0/24` entry:

```
root@pc1:~# ip route show
default via 10.5.1.2 dev eth1
10.5.1.0/24 dev eth1 proto kernel scope link src 10.5.1.1

root@pc2:~# ip route show
default via 10.5.3.2 dev eth1
10.5.3.0/24 dev eth1 proto kernel scope link src 10.5.3.1
```

The route table of `router1` and `router2` are same as the A103 i)

iii) Ersetzen Sie die Host-Routen auf den Routern durch Routen in die jeweiligen Subnetze von `pc1` und `pc2`!

Antwort: We use the same network topology as figure4.1. Since the `ping` command's output is similar to the figure4.2, we just paste the results of the default route in the appendix (See 5.2).
The route table of router1 and router2 is as follows, again, we hide the `192.168.0.0/24` entry:

```
root@router1:~# ip route show
10.5.1.0/24 via 10.5.1.2 dev eth1
10.5.3.0/24 via 10.5.2.2 dev eth2

root@router2:~# ip route show
10.5.1.0/24 via 10.5.2.1 dev eth2
10.5.3.0/24 via 10.5.3.2 dev eth1
```

`pc1` and `pc2` use the same configuration with the

iv) Starten Sie auf `pc1` einen `traceroute` nach `pc2`. Erläutern Sie anhand eines `tcpdump`-Mitschnitts die Funktionsweise von traceroute!

Antwort: The configurations of route table between devices and ip assignment are the same as the A103 iii).
We start the `traceroute` command. The traceroute outputs are:

```
root@pc1:~# traceroute 10.5.3.1
traceroute to 10.5.3.1 (10.5.3.1), 30 hops max, 60 byte packets
 1  10.5.1.2 (10.5.1.2)  1.543 ms  1.496 ms  1.470 ms
 2  10.5.2.2 (10.5.2.2)  4.906 ms  4.881 ms  4.855 ms
 3  10.5.3.1 (10.5.3.1)  4.829 ms  4.806 ms  4.783 ms
```

At the same time, we start the tcpdump command. Since the full outputs is too long, we just put the important part there:

```
15:30:12.566315 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08
↪  (oui Unknown), ethertype IPv4 (0x0800), length 74:
↪  10.5.1.1.43648 > 10.5.3.1.33434: UDP, length 32
15:30:12.566379 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08
↪  (oui Unknown), ethertype IPv4 (0x0800), length 74:
↪  10.5.1.1.55791 > 10.5.3.1.33435: UDP, length 32
...
15:30:12.569112 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02
↪  (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.1.2 >
↪  10.5.1.1: ICMP time exceeded in-transit, length 68
...
15:30:12.572024 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02
↪  (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.2.2 >
↪  10.5.1.1: ICMP time exceeded in-transit, length 68
...
15:30:12.572465 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02
↪  (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.3.1 >
↪  10.5.1.1: ICMP 10.5.3.1 udp port 33440 unreachable, length 68
```

The full output can be found in the Appendix 5.2.3
Depending on the output of the `tcpdump`, we can find the process of the `traceroute`. First, it sends several UDP packets. The TTL of each packet increases successively. Second, Each intermediate router returns `ICMP Time Exceeded` messages when the TTL reaches 0. These messages help the traceroute determine each hop on the path. Third, When the packet successfully reaches the target host 10.5.3.1, the target host returns `ICMP Destination Unreachable`, signalling the end of path probing.

v) Konfigurieren Sie Ihr Routing so, dass Daten von `pc1` an `pc2` **immer über router4** vermittelt werden und Daten von `pc2` an `pc1` **nie über router4** vermittelt werden! Zeigen Sie, dass Ihre Konfiguration funktioniert, indem Sie entsprechende traceroute-Ausgaben erzeugen. Zeichnen Sie ein **gerichteten Netzplan** für Ihren Aufbau.

Antwort: Based on the ping results, we construct a new network topology as in figure 4.3. The final ping results and traceroute outputs are shown in 4. From pc1 to pc2, there are four hops. These are router1's eth1, router4's eth1, router2's eth2, pc2's eth1 respectively. with the third hop being particularly noteworthy. The traceroute indicates that the response comes from 10.5.2.2, which corresponds to the eth2 interface of router2. However, the packet from pc1 to pc2 initially reaches router2 through its eth4 interface. Since the return packet can only exit via eth2 depending on the route table, the traceroute displays 10.5.2.2 instead of 10.5.5.2.
The packets from pc2 to pc1 encounter a similar situation. There are only three hops: router2's eth1, router1's eth4, and pc1's eth1. Since the packets from router1 to pc2 can only be routed through eth4, the traceroute output displays router1's eth4 interface in the results.

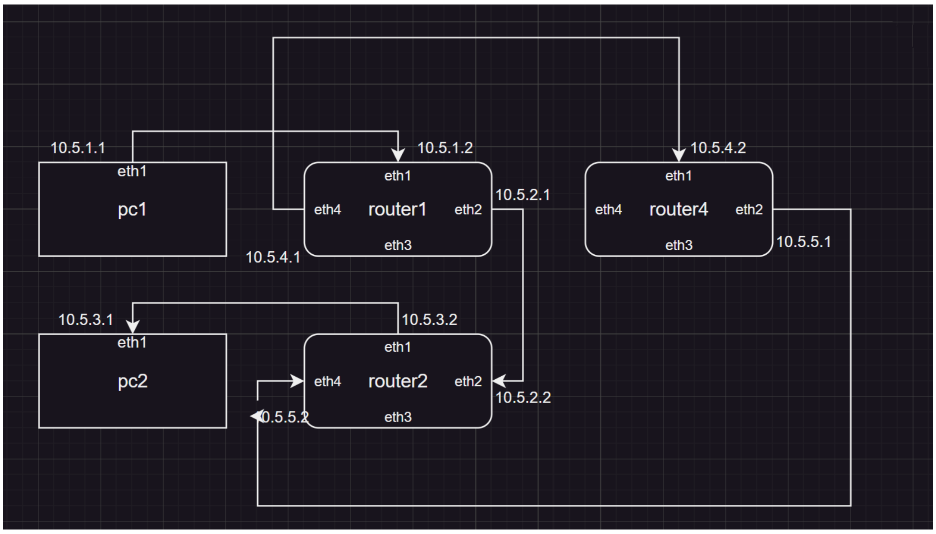


Figure 4.3: A103 v) Network Topology

```
PING 10.5.3.1 (10.5.3.1) from 10.5.1.1 eth1: 56(84) bytes of data.
64 bytes from 10.5.3.1: icmp_seq=1 ttl=62 time=2.19 ms
64 bytes from 10.5.3.1: icmp_seq=2 ttl=62 time=18.1 ms
64 bytes from 10.5.3.1: icmp_seq=3 ttl=62 time=0.965 ms
64 bytes from 10.5.3.1: icmp_seq=4 ttl=62 time=0.894 ms
64 bytes from 10.5.3.1: icmp_seq=5 ttl=62 time=0.784 ms

--- 10.5.3.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 0.784/4.577/18.056/6.758 ms

PING 10.5.1.1 (10.5.1.1) from 10.5.3.1 eth1: 56(84) bytes of data.
64 bytes from 10.5.1.1: icmp_seq=1 ttl=61 time=0.891 ms
64 bytes from 10.5.1.1: icmp_seq=2 ttl=61 time=0.869 ms
64 bytes from 10.5.1.1: icmp_seq=3 ttl=61 time=0.870 ms
64 bytes from 10.5.1.1: icmp_seq=4 ttl=61 time=0.735 ms
64 bytes from 10.5.1.1: icmp_seq=5 ttl=61 time=20.0 ms

--- 10.5.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 0.735/4.673/20.002/7.664 ms

traceroute to 10.5.3.1 (10.5.3.1), 30 hops max, 60 byte packets
 1  10.5.1.2 (10.5.1.2)  2.266 ms  2.214 ms  2.180 ms
 2  10.5.4.2 (10.5.4.2)  2.145 ms  2.110 ms  2.075 ms
 3  10.5.2.2 (10.5.2.2)  2.039 ms  2.008 ms  2.004 ms
 4  10.5.3.1 (10.5.3.1)  1.967 ms  1.917 ms  2.002 ms
```

```
traceroute to 10.5.1.1 (10.5.1.1), 30 hops max, 60 byte packets
 1  10.5.3.2 (10.5.3.2)  2.110 ms  2.068 ms  2.035 ms
 2  10.5.4.1 (10.5.4.1)  1.995 ms  1.961 ms  1.930 ms
 3  10.5.1.1 (10.5.1.1)  1.898 ms  1.870 ms  1.867 ms
```

The routing tables are shown as follows:

```
root@router1:~# ip route show
10.5.1.0/24 dev eth1 proto kernel scope link src 10.5.1.2
10.5.3.0/24 via 10.5.4.2 dev eth4
10.5.4.0/24 dev eth4 proto kernel scope link src 10.5.4.1

root@pc1:~# ip route show
default via 10.5.1.2 dev eth1
10.5.1.0/24 dev eth1 proto kernel scope link src 10.5.1.1

root@router2:~# ip route show
10.5.1.0/24 via 10.5.2.1 dev eth2
10.5.2.0/24 dev eth2 proto kernel scope link src 10.5.2.2
10.5.3.0/24 dev eth1 proto kernel scope link src 10.5.3.2

root@pc2:~# ip route show
default via 10.5.3.2 dev eth1
10.5.3.0/24 dev eth1 proto kernel scope link src 10.5.3.1

root@router4:~# ip route show
10.5.1.0/24 via 10.5.4.1 dev eth1
10.5.3.0/24 via 10.5.5.2 dev eth2
10.5.4.0/24 dev eth1 proto kernel scope link src 10.5.4.2
10.5.5.0/24 dev eth2 proto kernel scope link src 10.5.5.1
```

In router1's routing table, there is no entry for the 10.5.2.0/24 network. This means router1 will not send packets directly to router2. Similarly, router2 cannot forward packets directly to router4, as it lacks a route to reach router4 in its routing table.

5 Appendix

5.1 A101

5.1.1 A101 iii)

```
#!/bin/bash

show_ip(){
	local output=$1
	ips=()
	interfaces=()
	echo $output
	while read -r line; do
		ip=$(echo "$line" | awk '{print $2}' | cut -d'/' -f1)
		interface=$(echo "$line" | awk '{print $5}')
		ips+=("$ip")
		interfaces+=("$interface")
	done <<< "$output"
	for i in "${!ips[@]}"; do
		echo "${ips[i]} ${interfaces[i]}"
	done

}

ip_cmd='ip address show | grep 10.5'
sender_ip='10.5.1.1/24'
rec_ip='10.5.1.2/24'
rec_ip_no_code='10.5.1.2'
senders=()
recs=()
sender_eths=()
rec_eths=()
losses=()

sender_type="${1:-router}"
receiver_type="${2:-router}"
if [ "$sender_type" = "router" ]; then
	sender_ub=4
else
	sender_ub=3
fi
```

```
if [ "$receiver_type" = "router" ]; then
	rec_ub=4
else
	rec_ub=3
fi
for sender in $(seq 1 $sender_ub);
do
	sender_eth_nums=$(ssh $sender_type$sender "ip address show" | grep -c '^.*eth[0-9]:')
	sender_eth_nums=$(($sender_eth_nums-1))
	echo "$sender_type$sender has $sender_eth_nums eths"
	for sender_eth_num in $(seq 1 $sender_eth_nums);
	do
		# turn on the device
		ssh $sender_type$sender "ip link set dev eth$sender_eth_num up"
		ssh $sender_type$sender "ip address add $sender_ip dev eth$sender_eth_num"
		# for receiver in {1..4}
		for receiver in $(seq 1 $rec_ub);
		do
			receiver_eth_nums=$(ssh $receiver_type$receiver "ip address show" | grep -c '^.*eth[0-9]:')
			receiver_eth_nums=$(($receiver_eth_nums-1))
			echo "$receiver_type$receiver has $receiver_eth_nums eths"
			if [ "$sender_type" = "$receiver_type" ]; then
				if [ "$sender" -eq "$receiver" ]; then
					continue
				fi
			fi
			for receiver_eth_num in $(seq 1 $receiver_eth_nums);
			do
				ssh $receiver_type$receiver "ip link set dev eth$receiver_eth_num up"
				ssh $receiver_type$receiver "ip address add $rec_ip dev eth$receiver_eth_num"
				echo "sen $sender_type$sender eth$sender_eth_num ip $sender_ip"
				echo "rec $receiver_type$receiver eth$receiver_eth_num ip $rec_ip"
				ip_output=$(ssh $sender_type$sender $ip_cmd)
				result=$(show_ip "$ip_output")
				echo $result

				# test loss
```

```
                                loss=$(ssh $sender_type$sender "ping -c 5 -W 2 -I eth$sender_eth_num $rec_ip_no_code | awk -F', ' '/packet loss/ {print \$3}' | awk '{print int(\$1)}'")
                                echo $loss
                                if [ "$loss" -ne 100 ]; then
                                        senders+=("$sender")
                                        recs+=("$receiver")
                                        sender_eths+=("$sender_eth_num")
                                        rec_eths+=("$receiver_eth_num")
                                        losses+=("$loss")
                                fi


                                ssh $receiver_type$receiver "ip address del 10.5.1.2/24 dev eth$receiver_eth_num"
                        done
                done
                ssh $sender_type$sender "ip address del 10.5.1.1/24 dev eth$sender_eth_num"
        done
done

echo "results"

printf "%-10s %-15s %-12s %-15s %-10s\n" "sender #" "sender eth #" "receiver #" "receiver eth #" "losses #"
for i in "${!senders[@]}"; do
    printf "%-10s %-15s %-12s %-15s %-10s\n" "${senders[i]}" "${sender_eths[i]}" "${recs[i]}" "${rec_eths[i]}" "${losses[i]}"
done

```

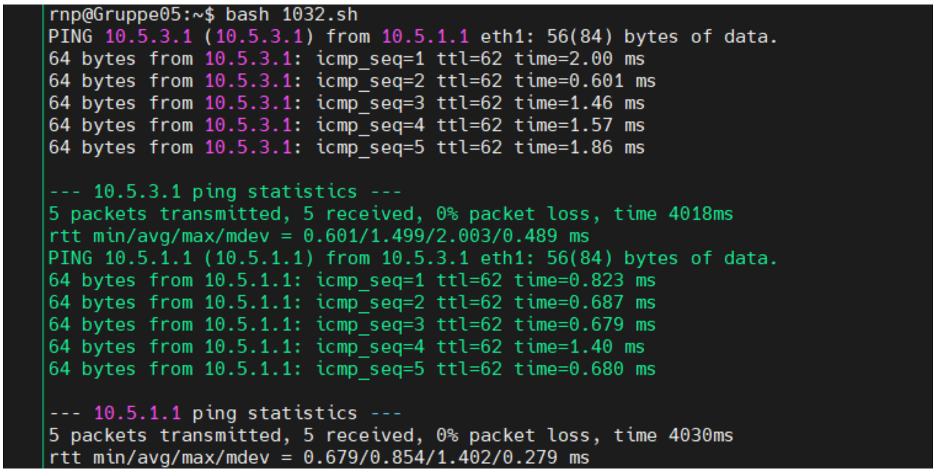
5.2 A103

5.2.1 A103 ii)

5.2.2 A103 iii)

5.2.3 A103 iv)

The output of the tcpdump command:



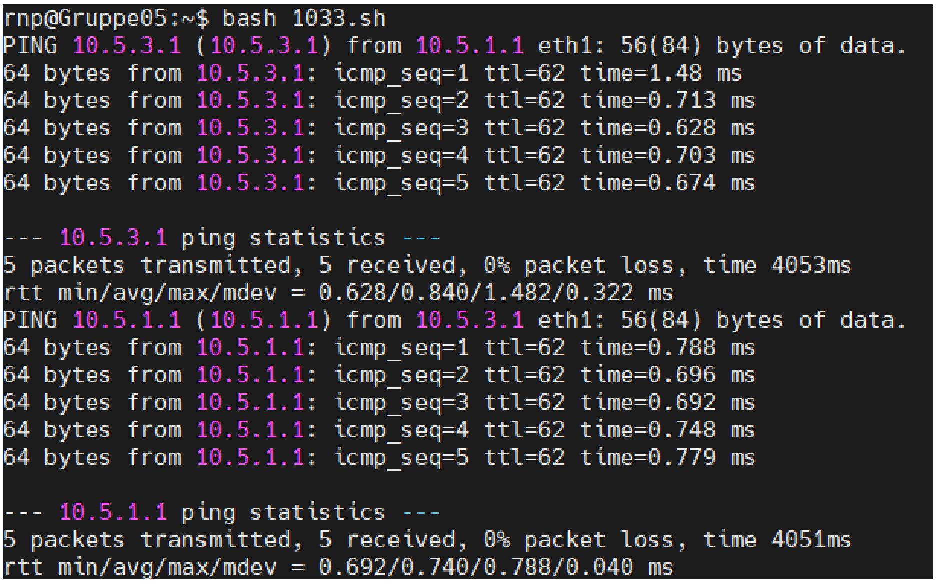
```
rnp@Gruppe05:~$ bash 1032.sh
PING 10.5.3.1 (10.5.3.1) from 10.5.1.1 eth1: 56(84) bytes of data.
64 bytes from 10.5.3.1: icmp_seq=1 ttl=62 time=2.00 ms
64 bytes from 10.5.3.1: icmp_seq=2 ttl=62 time=0.601 ms
64 bytes from 10.5.3.1: icmp_seq=3 ttl=62 time=1.46 ms
64 bytes from 10.5.3.1: icmp_seq=4 ttl=62 time=1.57 ms
64 bytes from 10.5.3.1: icmp_seq=5 ttl=62 time=1.86 ms

--- 10.5.3.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4018ms
rtt min/avg/max/mdev = 0.601/1.499/2.003/0.489 ms
PING 10.5.1.1 (10.5.1.1) from 10.5.3.1 eth1: 56(84) bytes of data.
64 bytes from 10.5.1.1: icmp_seq=1 ttl=62 time=0.823 ms
64 bytes from 10.5.1.1: icmp_seq=2 ttl=62 time=0.687 ms
64 bytes from 10.5.1.1: icmp_seq=3 ttl=62 time=0.679 ms
64 bytes from 10.5.1.1: icmp_seq=4 ttl=62 time=1.40 ms
64 bytes from 10.5.1.1: icmp_seq=5 ttl=62 time=0.680 ms

--- 10.5.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4030ms
rtt min/avg/max/mdev = 0.679/0.854/1.402/0.279 ms
```

Figure 5.1: A103 ii) Ping Results

```
15:30:12.566315 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.43648 > 10.5.3.1.33434: UDP, length 32
15:30:12.566379 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.55791 > 10.5.3.1.33435: UDP, length 32
15:30:12.566428 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.48341 > 10.5.3.1.33436: UDP, length 32
15:30:12.566475 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.37163 > 10.5.3.1.33437: UDP, length 32
15:30:12.566524 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.57251 > 10.5.3.1.33438: UDP, length 32
15:30:12.566572 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.59408 > 10.5.3.1.33439: UDP, length 32
15:30:12.566619 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.47726 > 10.5.3.1.33440: UDP, length 32
15:30:12.566666 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.34959 > 10.5.3.1.33441: UDP, length 32
15:30:12.566713 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.52377 > 10.5.3.1.33442: UDP, length 32
15:30:12.566760 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.52845 > 10.5.3.1.33443: UDP, length 32
```



```
rnp@Gruppe05:~$ bash 1033.sh
PING 10.5.3.1 (10.5.3.1) from 10.5.1.1 eth1: 56(84) bytes of data.
64 bytes from 10.5.3.1: icmp_seq=1 ttl=62 time=1.48 ms
64 bytes from 10.5.3.1: icmp_seq=2 ttl=62 time=0.713 ms
64 bytes from 10.5.3.1: icmp_seq=3 ttl=62 time=0.628 ms
64 bytes from 10.5.3.1: icmp_seq=4 ttl=62 time=0.703 ms
64 bytes from 10.5.3.1: icmp_seq=5 ttl=62 time=0.674 ms

--- 10.5.3.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4053ms
rtt min/avg/max/mdev = 0.628/0.840/1.482/0.322 ms
PING 10.5.1.1 (10.5.1.1) from 10.5.3.1 eth1: 56(84) bytes of data.
64 bytes from 10.5.1.1: icmp_seq=1 ttl=62 time=0.788 ms
64 bytes from 10.5.1.1: icmp_seq=2 ttl=62 time=0.696 ms
64 bytes from 10.5.1.1: icmp_seq=3 ttl=62 time=0.692 ms
64 bytes from 10.5.1.1: icmp_seq=4 ttl=62 time=0.748 ms
64 bytes from 10.5.1.1: icmp_seq=5 ttl=62 time=0.779 ms

--- 10.5.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4051ms
rtt min/avg/max/mdev = 0.692/0.740/0.788/0.040 ms
```

Figure 5.2: A103 iii) Ping Results

```
15:30:12.566813 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.37429 > 10.5.3.1.33444: UDP, length 32
15:30:12.566869 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.60746 > 10.5.3.1.33445: UDP, length 32
15:30:12.566920 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.54456 > 10.5.3.1.33446: UDP, length 32
15:30:12.566970 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.37812 > 10.5.3.1.33447: UDP, length 32
15:30:12.567024 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.51663 > 10.5.3.1.33448: UDP, length 32
15:30:12.567070 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.58697 > 10.5.3.1.33449: UDP, length 32
15:30:12.569112 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.1.2 > 10.5.1.1: ICMP time exceeded in-transit, length 68
15:30:12.569114 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.1.2 > 10.5.1.1: ICMP time exceeded in-transit, length 68
15:30:12.569116 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.1.2 > 10.5.1.1: ICMP time exceeded in-transit, length 68
```

```
15:30:12.570491 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 81: 10.5.1.1.57334 > 10.153.211.1.domain: 30412+ PTR? 2.1.5.10.in-addr.arpa. (39)
15:30:12.572024 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.2.2 > 10.5.1.1: ICMP time exceeded in-transit, length 68
15:30:12.572026 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.2.2 > 10.5.1.1: ICMP time exceeded in-transit, length 68
15:30:12.572028 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.2.2 > 10.5.1.1: ICMP time exceeded in-transit, length 68
15:30:12.572030 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 109: 10.5.1.2 > 10.5.1.1: ICMP net 10.153.211.1 unreachable, length 75
15:30:12.572129 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 81: 10.5.1.1.39035 > 10.153.211.1.domain: 30412+ PTR? 2.1.5.10.in-addr.arpa. (39)
15:30:12.572465 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.3.1 > 10.5.1.1: ICMP 10.5.3.1 udp port 33440 unreachable, length 68
15:30:12.572467 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.3.1 > 10.5.1.1: ICMP 10.5.3.1 udp port 33441 unreachable, length 68
15:30:12.572469 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.3.1 > 10.5.1.1: ICMP 10.5.3.1 udp port 33442 unreachable, length 68
15:30:12.572471 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.3.1 > 10.5.1.1: ICMP 10.5.3.1 udp port 33443 unreachable, length 68
15:30:12.572473 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.3.1 > 10.5.1.1: ICMP 10.5.3.1 udp port 33444 unreachable, length 68
15:30:12.572475 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.3.1 > 10.5.1.1: ICMP 10.5.3.1 udp port 33445 unreachable, length 68
15:30:13.272480 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 81: 10.5.1.1.39351 > 10.153.211.1.domain: 10747+ PTR? 1.3.5.10.in-addr.arpa. (39)
15:30:17.577940 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.51830 > 10.5.3.1.33450: UDP, length 32
15:30:17.577998 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.49257 > 10.5.3.1.33451: UDP, length 32
```

```
15:30:17.578056 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 74: 10.5.1.1.58847 > 10.5.3.1.33452: UDP, length 32
15:30:17.578466 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 81: 10.5.1.1.38198 > 10.153.211.1.domain: 47433+ PTR? 2.2.5.10.in-addr.arpa. (39)
15:30:17.580510 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 109: 10.5.1.2 > 10.5.1.1: ICMP net 10.153.211.1 unreachable, length 75
15:30:17.580621 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 81: 10.5.1.1.51597 > 10.153.211.1.domain: 47433+ PTR? 2.2.5.10.in-addr.arpa. (39)
15:30:17.581351 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 109: 10.5.1.2 > 10.5.1.1: ICMP net 10.153.211.1 unreachable, length 75
15:30:17.582081 00:16:3e:00:00:02 (oui Unknown) > 00:16:3e:00:00:08 (oui Unknown), ethertype IPv4 (0x0800), length 81: 10.5.1.1.39284 > 10.153.211.1.domain: 20390+ PTR? 1.3.5.10.in-addr.arpa. (39)
15:30:17.583108 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.3.1 > 10.5.1.1: ICMP 10.5.3.1 udp port 33450 unreachable, length 68
15:30:17.583110 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.3.1 > 10.5.1.1: ICMP 10.5.3.1 udp port 33451 unreachable, length 68
15:30:17.583112 00:16:3e:00:00:08 (oui Unknown) > 00:16:3e:00:00:02 (oui Unknown), ethertype IPv4 (0x0800), length 102: 10.5.3.1 > 10.5.1.1: ICMP 10.5.3.1 udp port 33452 unreachable, length 68
```

List of Figures