

Großes Studienprojekt

Aaron Winziers
(1176638)

Thomas Schimper
(1184921)

Christopher Schmitt
(1192978)

Amir Durguti
(1172920)

Rudolf Barth
(1100075)

10. November 2017

Zusammenfassung

Da IPv4 aufgrund von immer weiter steigenden Nutzerzahlen und internetfähigen Geräte bald nicht mehr als Standardadressraum verwendet werden kann und IPv6 eine mögliche Lösung dieses Problems ist, haben wir uns innerhalb unseres großen Studienprojektes mit dem Aufsetzen und Verwalten eines IPv4 bzw. IPv6 Netzwerkes auseinandergesetzt.

1 Fragestellung

Um sich mit dem Zusammenspiel von Geräten innerhalb eines Netzwerkes vertraut zu machen, war gefordert das folgende Diagramm als Netzwerk umzusetzen:

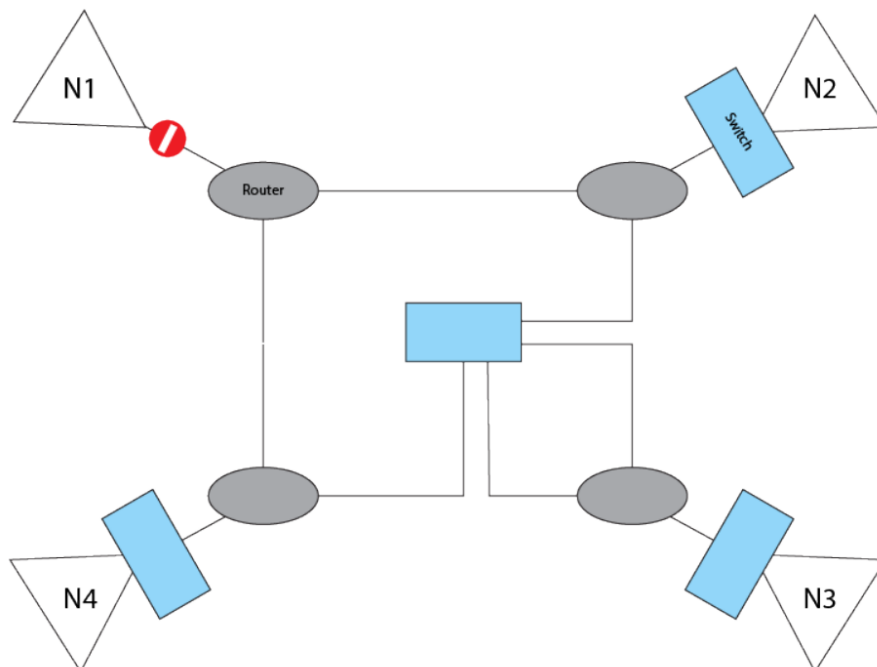


Abbildung 1: Aufbau des Netzwerks

Die mit N1 bis N4 beschriftete Dreiecke bezeichnen die von uns erstellten Subnetze, die durch die Router (graue Ovale, R1 bis R4) verwaltet werden, und die blauen Vierecke repräsentieren jeweils einen Switch. Das „Einfahrt Verboten“ Schild symbolisiert zusätzlich eine von uns konfigurierte Firewall.

Jeder Router verwaltet die Vergabe der IPs innerhalb der eigenen Subnetze über einen eigenen DHCP-Server. Alle Router erkennen sich und kommunizieren untereinander über statische Routen, wobei die Kommunikation zwischen R2 und R4 zwei Hops (über R1) benötigt. Außerdem werden die Verbindungen zwischen R2 und R3, und R3 und R4 über einem Switch und durch zwei unabhängige VLANs realisiert, die über den Switch in der Mitte konfiguriert werden.

Hinter R1 ist eine Verbindung zum Internet, die durch die Firewall geschützt wird, welche durch den Gebrauch von `iptables` realisiert wurde. Die Firewall ist so konfiguriert, dass sie den ausgehenden Verkehr nicht einschränkt, jedoch eingehende Pakete weiterleitet, wenn sie im Zusammenhang mit ausgehenden Paketen stehen.

Der innere Switch sollte zusätzlich so konfiguriert sein das er die Verbindungen zu den Routern über die verbliebene Ports spiegelt.

2 Herangehensweise

Nach ersten Recherchen bezüglich der Router und deren Funktionsweise fiel uns auf, dass diese mit einer veralteten Firmwareversion ausgestattet waren. Deshalb war der erste Arbeitsschritt das Aktualisieren der Firmware. Die jetzt aktualisierte Firmware verfügte über eine GUI, mit deren Hilfe jegliche Einstellungen bezüglich der IPv4-Einstellungen konfiguriert werden konnten.

Bevor die Aufgabe bearbeitet wurde war eine Einarbeitung in die Funktionen und Eigenschaften des Systems erforderlich, um eine vollständige Umsetzung der Fragestellung zu realisieren. Dazu wurde ein simples Netzwerk mit Hilfe der Konsole konfiguriert. In diesem Netzwerk wurden alle für die Aufgabenstellung wichtige Komponenten (z.B. DHCP, Firewall, Static Routes, usw.) eingebunden, um ein tieferes Verständnis dieser zu gewinnen.

Nach ersten Konfigurationen wurde mit Hilfe der GUI geprüft, ob alle Einstellungen erfolgreich übernommen wurden. Weil das Konfigurieren der Router über die GUI deutlich einfacher und übersichtlicher ist, wurde zur Umsetzung der Aufgabenstellung eine Kombination aus Konsole und GUI verwendet.

Bei der Bearbeitung der Aufgabe musste zu aller erst das Netzwerk auf physischer Ebene, wie auf Abbildung 1 zu sehen, zusammengesetzt werden. Dabei war eine übersichtliche Verkabelung wichtig, um die Subnetze auch unproblematisch unterscheiden und visualisieren zu können. Des Weiteren wurden die Router etikettiert um weitere Übersicht und Organisation zu erschaffen.

Wichtig war die Dokumentation des Verlaufs der Umsetzung, als auch eine Möglichkeit ältere Konfigurationen zu sichern und wieder herzustellen. Hierzu wurde Git auf dem berühmten Hostingplattform Github verwendet, dies erlaubte eine bessere Organisation des Projekts und effektivere Zusammenarbeit.

3 Hardware

Um die Fragestellung beantworten zu können, haben wir folgende Hardware zur Verfügung gestellt bekommen.

- 4× EdgeRouter Lite
- 4× Netgear ProSAFE Plus Switch GS108E
- LAN-Kabel

Zunächst haben wir die Software auf den Routern analysiert. Diese verwenden als Basissystem EdgeOS, was im Grunde ein von Ubiquiti entworfenes Ubuntu-Derivat, welches sich bequem durch Verwendung von `apt` und `aptitude` verwalten lässt. Um jetzt auf das System zugreifen zu können haben wir uns über SSH auf dem Router angemeldet. Dazu mussten wir uns über das `eth0`-Interface mit dem Router verbinden. Um dies zu erreichen, haben wir uns eine statische IP im gleichen Netz, in dem sich auch der Router befindet, zugewiesen. Als Standardzugangsdaten verwendet das System folgende Daten :

`ubnt:ubnt`

Meldet man sich mit den folgenden Daten am Router per SSH an, so erkennt man schnell, das EdgeOS als Standardshell die **Vyatta Shell**¹ verwendet, die auf der GNU Bash aufbaut. Diese hatte allerdings die normale Bash-Vervollständigung deaktiviert und wir verfügten nur über die zur Konfiguration vorgesehenen Befehle. Um dies zu beheben haben wir folgende Zeile in der `.bashrc` hinzugefügt.

```
source /etc/bash_completion
```

Wollen wir jetzt den Router per CLI konfigurieren, so können wir über den Befehl `configure` die Konfigurationsumgebung betreten. Hier kann man jetzt die Einstellung interaktiv einstellen. Ist man mit dem Einstellen fertig, so kann man mit dem Befehl `commit & save` permanent abspeichern. Wollen wir alle Daten auf einmal abändern so finden wir die Konfigurationsdatei an folgender Stelle im System: `/config/config.boot`. Die Umgebung kann man darauf mit `exit` wieder verlassen. Öffnet man die Datei mit `vim`, so kann man die Werte der Einstellungen direkt in der Datei eintragen, abspeichern und das System neustarten, um die Einstellungen zu übernehmen.

Listing 1: config.boot von Router1

```
interfaces {
    ethernet eth0 {
        address 192.168.1.1/24
        description LAN
        duplex auto
        speed auto
    }
    ethernet eth1 {
        address 192.168.253.11/24
        duplex auto
        speed auto
    }
    ethernet eth2 {
        address 192.168.252.12/24
        duplex auto
        speed auto
    }
    loopback lo {
    }
}
```

¹<https://github.com/vyos/vyatta-bash>

Wir haben um den Konfigurationsprozess zu beschleunigen, die Konfigurationsdatei mit der zweiten Variante direkt editiert, da wir diese auch später noch mit **Git** synchronisieren wollten. Die Vorteile der direkten Konfiguration des Routers ergibt sich dadurch, da es deutlich schneller als das Webinterface, oder überhaupt eine **GUI** zu verwenden, ist und da **IPv6** nicht über das Webinterface konfigurierbar ist.

Um im Folgenden **Git** zu verwenden muss man die offiziellen Debian Quellen in der Datei `/etc/apt/sources.list` eintragen. Dies geht aber auch bequem über die **configure**-Umgebung mit:

Listing 2: Hinzufügen der Debian Quellen

```
configure
set system package repository wheezy components 'main contrib non-free '
set system package repository wheezy distribution wheezy
set system package repository wheezy url http://http.us.debian.org/debian
commit
save
exit
```

Jetzt kann man wie auch unter Debian und Ubuntu bekannt mit **sudo apt-get update** die Quellen synchronisieren und mit **sudo apt upgrade** aktualisieren. Wollen wir jetzt ein bestimmtes Paket installieren, so können wir per **apt-cache search** nach diesem suchen.

In unserem Fall wollen wir **Git** installieren. Wir installieren dieses mit **sudo apt-get install git** und bestätigen die Rückfrage mit Yes. Ist **Git** installiert, so können wir mit **git init** ein neues Repository erstellen. Parallel dazu haben wir auf Github ein **private** Repository erstellt und alle Teammitglieder als **Contributer** eingeladen. Danach kann man per **git remote add origin** die URL zum Repository eintragen. Hat man so das locale Repository konfiguriert, fügt man neue Dateien mit **git add** neue Dateien versionieren. Mit **git commit** können wir einen Commit erstellen und diesen mit **git push origin master** in das Remote-Repository pushen. Wenn wir jetzt zusammenarbeiten wollen muss, sollte man nach dem Klonen des Repositories noch mit **git pull** das Repository aktualisieren. Werden unsere erstellten Änderungen von **Git** mit einem Mergekonflikt zurückgewiesen, so müssen wir diesen noch vorher lokal lösen und danach die Lösung commiten.

4 Konfiguration

R0 dient als Schnittstelle zum Internet, deshalb ist er der einzige Router, welcher kein eigenes Subnetz aufspannt. Im Gegensatz zu den anderen Routern konnte an **eth1** und **eth2** (die Verbindung zu R1 und R2) keine festen IP-Adressen vergeben werden. Stattdessen läuft ein **DHCP**-Server, der dem jeweiligen Anschluss eine passende IP-Adresse vergibt. Das ist notwendig, da sonst kein Zugriff auf das Webinterface des Routers mehr möglich wäre. Die drei anderen Router sind über eine IP-Adresse aus dem Adressraum ihres jeweiligen Subnetzes erreichbar (z.B. Router 3: 192.168.3.1)

R1-R3 haben alle eine recht ähnliche Konfiguration: alle bieten auf **eth0** ein eigenes Subnetz. Über **eth1** und **eth2** sind sie mit den anderen Routern verbunden. Die Router sowie die Geräte in den Subnetzen können sich über statische Routen erreichen.

Die Firewall wurde entsprechend der Vorgabe in der Aufgabenstellung konfiguriert. Die Konfiguration erfolgte hauptsächlich über die in der Router-Software verfügbare Shell. Diese ist wesentlich mächtiger als das integrierte Webinterface, insbesondere bei seltener verwendeten Features des Routers.

Das VLAN wurde durch die Konfiguration des entsprechenden Switches realisiert. Dies lässt sich mit der Konfigurationssoftware bewerkstelligen, die allerdings nur für Windows vorliegt (alternativ geschieht die Konfiguration über ein Web-Interface). Unsere Lösung sah zuerst vor, die Ports 1-4 und 5-8 als zwei VLANs zu trennen. Da wir jedoch noch einen Port für das Monitoring benötigten, mussten wir das zweite VLAN um einen Port verkleinern. Somit kann nun der gesamte Traffic des Switches an Port 8 abgegriffen werden.

5 Netzwerkanalyse

Um die realisierte Topologie auf Korrektheit und Funktionalität zu überprüfen, wurde nach der Konfiguration des gesamten Netzwerkes, sowohl die Erreichbarkeit unter den einzelnen Geräten, als auch die Verbindungsrouten selbst mit Hilfe verschiedener Methoden getestet.

Um die Erreichbarkeit zu prüfen, haben wir versucht von einem Gerät aus ein weiteres Gerät, welches in einem anderen Subnetz registriert hatte, mittels des Unix-Tools **ping** zu erreichen. Um zu verhindern, dass ein falsches Gerät als Ziel angegeben wurde, haben wir uns zunächst die einzelnen IP-Adressen der jeweiligen Rechner mit dem Konsolen-Befehl **ifconfig** beschafft und diese mit den DHCP-Leases im entsprechenden Router überprüft. Die Tests verliefen alle erfolgreich, was uns dazu führte, das Netzwerk weitergehend zu untersuchen.

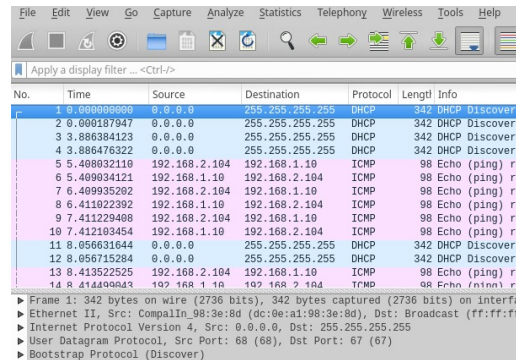


Abbildung 2: Ping-Test von Wireshark aufgenommen

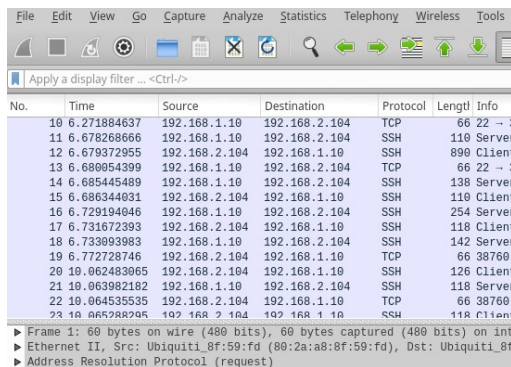
Test	Ausgangsrechner	Zielrechner	Ergebnis
Ping	hinter R2	hinter R1	erfolgreich
Ping	hinter R2	hinter R3	erfolgreich
SSH	hinter R2	hinter R1	erfolgreich
SFTP	hinter R2	hinter R1	erfolgreich

Abbildung 3: Durchgeführte Tests

Bevor wir jedoch TCP oder UDP basierende Pakete über die internen Leitungen versendet haben, richteten wir zunächst einen uns zur Verfügung gestellten Raspberry Pi ein, sodass dieser einen OpenSSH-Server bereitstellte. Dieser wurde dazu benutzt um SSH und SFTP-Anfragen zu realisieren. Um weitergehende Informationen zu erhalten, wurde ein dritter Rechner (A1) am Mirror-Port des Switches mit den VLANs eingebunden, dieser mit dem Tool **Wireshark** die pVerbindungen und Übertragungen analysierte.

Die Verbindung über Ping (ICMP), SSH oder auch SFTP von R1 nach R3 verursachte bei A1 keinen angezeigten Datenverkehr, was zu erwarten war, da R0, R1 und R3 über Static Routes direkt kommunizieren können und keine Daten über den Switch versendet werden. Alle anderen Verbindungen, welche über oder mit R2 hergestellt wurden zeigten bei den jeweiligen Tests die zu erwartenden Protokolle und Daten an. Bei den Versuchen mit **ping** kam das ICMP-Protokoll zum tragen. Bei den SSH und SFTP Verbindungen wurde TCP und SSH verwendet, jedoch war die Gewichtung der einzelnen Pakete unterschiedlich. So war die Paketverteilung von SSH zu TCP beim SSH-Test ungefähr 2 zu 1, wohingegen beim SFTP-Test der Anteil an SSH Paketen wesentlich höher war.

6 Evaluation & Fazit



No.	Time	Source	Destination	Protocol	Length	Info
10	6.271884637	192.168.1.10	192.168.2.104	TCP	66	22 → 3
11	6.678268666	192.168.1.10	192.168.2.104	SSH	110	Server
12	6.679372955	192.168.1.10	192.168.1.10	SSH	890	Client
13	6.680954399	192.168.1.10	192.168.2.104	TCP	66	22 → 3
14	6.685445489	192.168.1.10	192.168.2.104	SSH	138	Server
15	6.686344031	192.168.2.104	192.168.1.10	SSH	110	Client
16	6.729194046	192.168.1.10	192.168.2.104	SSH	254	Server
17	6.731672393	192.168.2.104	192.168.1.10	SSH	118	Client
18	6.733093983	192.168.1.10	192.168.2.104	SSH	142	Server
19	6.772728746	192.168.2.104	192.168.1.10	TCP	66	38760
20	10.062483065	192.168.2.104	192.168.1.10	SSH	126	Client
21	10.063982182	192.168.1.10	192.168.2.104	SSH	118	Server
22	10.064535535	192.168.2.104	192.168.1.10	TCP	66	38760
23	10.065282905	192.168.2.104	192.168.1.10	SSH	118	Client

Abbildung 4: SSH-Test von Wireshark aufgenommen

zunächst einen Router so, dass dieser über einen DHCP-Server automatisch IP-Adressen an Clients vergeben konnte. Als sich unsere Geräte eine IP-Adresse automatisch vom Router beziehen konnten, gingen wir dazu über zwei Router miteinander zu verbinden. Damit Geräte in den zwei verschiedenen Subnetzen der Router untereinander kommunizieren konnten und sich gegenseitig finden, bedarf es dem Gebrauch von statischen Routen, damit die Router wussten, wohin sie die Anfrage weiterleiten mussten.

Mit dem neuerlangten Wissen begannen wir jetzt unser Projekt mit dem Verbinden des Routers 0, der nicht nur als Gateway ins Internet fungieren sollte, sondern auch eine Brücke zwischen Router 1 und 3 bildete. Wichtig dabei war es zu erkennen, dass der Router 0 nach Abschluss der Konfiguration nicht mehr auf direktem Weg über den Ethernet-Port **eth0** erreicht werden konnte, da dieser dafür benutzt wurde, um eine Verbindung zum Internet herzustellen. Nach ausreichender Konfiguration überbrückte der Router Anfragen auf **eth1** und **eth2** und andersherum. Über die gemeinsam genutzte Schnittstelle **eth0** konnte eine Verbindung mit dem Internet hergestellt werden.

Als letzten Teil der Aufgabe kam der 8-Port-Switch von NetGEAR zum Einsatz. Dieser sollte die Ports eins und zwei zu einem VLAN hinzufügen und die Ports drei und vier in ein zweites. In dem ersten VLAN sollten die Router 2 und 3 kommunizieren und auf dem zweiten Router 1 und 2. Zusätzlich wurden alle Ports auf denen Traffic erfolgen könnte auf den Port 8 gespiegelt. Über diesen Port konnten wir mittels Wireshark Traffic analysieren und speichern, was wir auch im Teil der Netzwerkanalyse getan haben.

Der ersten Teil des Projekts wies uns in die Grundlagen des IPv4-Protokoll ein und bereitete uns darauf vor, das vorgegebene Netzwerk auf das IPv6-Protokoll zu portieren. Wir eigneten uns nicht nur neue Fähigkeiten im Umgang mit Netzarchitekturen, sondern lernten auch den Umgang mit embedded Linuxsystemen und die Verwendung von Git.

Der erste Teil des großen Praktikums erwies sich als günstiger Einstieg, um effizient den Umgang mit dem IPv4-Protokoll und allgemeinen Netzwerkarchitekturen zu erlernen. Wenn man zu beginnt vielleicht gedacht haben sollte, dass das Einrichten eines Netzwerkes bloß Plug-and-Play sei, mag dies vielleicht für einfache Netzwerke mit nur einem Router gelten. Betreibt man jedoch wie bei uns verlangt mehrere Router mit verschiedenen Subnetzen, so zeigt sich die Komplexität der Netzwerkkonfiguration. Um die Aufgabenstellung wie gefordert absolvieren zu können mussten wir zunächst lernen, wie wir die Router miteinander kommunizieren lassen konnten. Wir konfigurierten