DATANETTVERK: Lab-oppgave 2

Gjennomført individuelt: Alexander Gilstedt

Forberedelser

Hadde ikke wireshark installert, og fulgte en fremgangsmåte på nettet for dette:

https://computingforgeeks.com/how-to-install-wireshark-on-ubuntu-desktop/

Møtte på noen problemer når jeg skulle starte wireshark (begynne å fange datapakker). Hadde ikke de nødvendige tillatelsene. Måtte da søke litt rundt og finne ut hvordan jeg skulle legge til brukeren i wireshark- gruppen. Fant en løsning på dette, og etter en omstart virket det som forventet.

1)

Startet wireshark, og pinget finn.no slik:

Her vises også resultatet

```
alexander@S540:~$ ping www.finn.no
PING www.finn.no (152.90.94.9) 56(84) bytes of data.
64 bytes from softwarelb-prod1.finn.no (152.90.94.9): icmp_seq=3 ttl=55 time=13.
6 ms
64 bytes from softwarelb-prod1.finn.no (152.90.94.9): icmp_seq=6 ttl=55 time=14.
2 ms
64 bytes from softwarelb-prod1.finn.no (152.90.94.9): icmp_seq=7 ttl=55 time=12.
5 ms
64 bytes from softwarelb-prod1.finn.no (152.90.94.9): icmp_seq=23 ttl=55 time=12
.8 ms
64 bytes from softwarelb-prod1.finn.no (152.90.94.9): icmp_seq=38 ttl=55 time=12
.4 ms
^C
--- www.finn.no ping statistics ---
44 packets transmitted, 5 received, 88,6364% packet loss, time 43854ms
rtt min/avg/max/mdev = 12.371/13.110/14.215/0.696 ms
```

I wireshark velger jeg en frame, og man kan da velge å se de ulike headerene. Velger Ethernet:

```
▸ Ethernet II, Src: IntelCor_e2:f7:7f (3c:f0:11:e2:f7:7f), Dst: JensenSc_14:40:a8 (34:21:09:14:40:a8)
```

Ekspansjonen av denne ser slik ut:

```
▾ Ethernet II, Src: IntelCor_e2:f7:7f (3c:f0:11:e2:f7:7f), Dst: JensenSc_14:40:a8 (34:21:09:14:40:a8)
  ▾ Destination: JensenSc_14:40:a8 (34:21:09:14:40:a8)
      Address: JensenSc_14:40:a8 (34:21:09:14:40:a8)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  ▾ Source: IntelCor_e2:f7:7f (3c:f0:11:e2:f7:7f)
      Address: IntelCor_e2:f7:7f (3c:f0:11:e2:f7:7f)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
```

Her kan man tydelig se de forskjellige delene Ethernet headeren består av

-**Blå markering** (Øverst): Dette er MAC-adressen til destinasjonen til datapakken. Altså mitt nettverk, da dette er en request

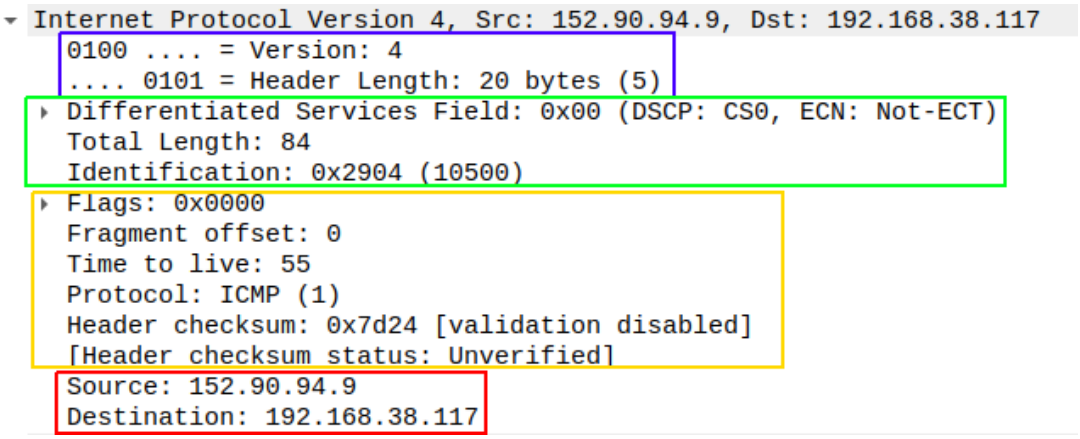
-**Rød markering** (Midten): Dette er MAC-adressen til kilden, altså der datapakken kommer fra. Dette kan for eksempel være en av serverne til finn.no.

-**Grønn markering** (Nederst): Dette indikerer hvilken ethertype som blir benyttet. I dette tilfellet IPv4. Denne informasjonen blir brukt av den mottakende enden til å bestemme hvordan informasjonen skal behandles. (Blir benyttet av data-link laget)

2)

Startet wireshark, og pinget finn på samme måte som forrige oppgave. Undersøkes en frame kan man se seksjonen/ headeren for IP.

En ekspansjon av denne headeren ser slik ut:



Blå firkant: finnes informasjon om **hvilken IP-protokoll** som blir benyttet. I dette tilfellet ser vi at den er av versjon 4. Altså IPv4. Rett under finnes verdien for **lengden av headeren.** Denne er 20 bytes, noe som også er minumus-verdien. Maks er 60 bytes.

Grønn firkant: Her spesifiseres først hvilken prioritet og type service som er gjeldende. Dette spesifiserer hvordan dataen skal bli behandlet. Videre ser vi den totale lengden på hele pakken. Dette innebærer da både header og data. Den er her på 84 byte. Helt på slutten av denne delen finnes en type identifikasjon. Denne brukes til til ...

Gul firkant: Denne delen begynner med **flags**. Denne informasjonen brukes til å kontrollere eller sjekke de forskjellige fragmentene i datapakken. Videre finner vi **fragment offset** denne informasjonen er viktig hvis pakken er for stor til å putte i en fram. Det når de ulike fragmentene skal settes sammen igjen.

Time to live: Dette er en viktig del av IP-headeren. Denne verdien forteller hvor lenge pakken kan "leve". Hvis ikke pakken er fremme før denne verdien er null blir den kastet. Hver ruter pakken er innom trekker ifra en verdi fra denne verdien. I dette tilfellet er denne verdien 55. En verdi ned finnes **protocol,** denne definerer hvilken protokoll som er brukt i data-delen av pakken. De to siste delene handler om **checksum.** Hvis pakken kommer frem og det blir kaklulert en annen checksum enn den som er spesifisert i dette feltet, vil pakken bli sett bort ifra. I dette tilfellet står det at dette er disabled.

Rød firkant: Her finner vi IP-adressen til source, i dette tilfellet finn.no. Rett nedenfor finner vi IP-en til destination som i denne testen var mitt hjemmenettverk. Begge adressene er **IP versjon 4. (IPv4)**

3)

Starter wireshark. Jeg prøvde først å pinge en tilfeldig IP på nettet, men slet med å få noe respons. Valgte derfor å pinge en host på mitt eget LAN. (En ipad). Leste av IP-en fra denne, og pinget dette i terminalen:

```
alexander@S540:~$ ping 192.168.38.105
PING 192.168.38.105 (192.168.38.105) 56(84) bytes of data
```

Etter å filtrere for arp i Wireshark, fant jeg igjen request og response for denne IP-en:

Request:

```
IntelCor_e2:f7:7f    Broadcast           ARP         42 Who has 192.168.38.105? Tell 192.168.38.117
```

Vi ser for øvrig at denne requesten er av typen Broadcast som betyr at den går ut til alle Host på nettverket.

Reply:

Når Hosten (enheten) som har denne IP-adressen får denne beskjeden, vil den svare med MAC-adressen til den aktuelle enheten. Svaret blir plukket opp som en frame i wireshark:

```
Apple_72:9d:34      IntelCor_e2:f7:7f    ARP         42 192.168.38.105 is at 34:12:98:72:9d:34
```

(Ipaden på nettverket svarer, og sender tilbake MAC-adressen)

Dette stemmer for øvrig overens med oppgitt adresse på Ipaden:

Request

```
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: IntelCor_e2:f7:7f (3c:f0:11:e2:f7:7f)
    Sender IP address: 192.168.38.117
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.38.105
```

Ovenfor er et utklipp av request- delen fangen i wireshark. De ulike delene kan identifiseres ovenfra og ned:

Hardware type: Denne delen forteller hva slags hardware som blir brukt til å sende arp-beskjeden. I dette tilfellet var det av typen Ethernet, som også er en av de vanligste.

Protocol type: Spesifiserer hvilken type protokoll som blir brukt, (IPv4)

Hardware size: Dette spesifiserer hvor stor adressen til hardwaren som er brukt er. For ethernet (og noen andre) er denne på 6. Noe vi kjenner igjen av informasjonen.

Protocol size: Spesifiserer hvor lang protokoll adresser (på lag 3) er i denne pakken. For IPv4 er dette alltid 4, noe vi kjenner igjen av informajonen.

Opcode: Dette feltet spesifiserer hva slags type arp- beskjed det er snakk om. 1 tilsvarer en request

Sender MAC address: Dette er MAC-adressen til enheten som sender forespørselen. I dette tilfellet er det min bærbare laptop.

Sender IP-adress: IP-adressen til avsender

Target MAC-adress: MAC-adressen til mottaker. Denne er følgelig 00.00 …. i dette tilfellet, fordi MAC –adressen ikke er kjent før mottaker har blitt identifisert og sendt et svar tilbake

Target IP-adress: IP-adressen for mottakeren. Det er denne som brukes til å identifisere enheten som skal svare med MAC-adresse

Reply-delen ser slik ut:

```
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: Apple_72:9d:34 (34:12:98:72:9d:34)
    Sender IP address: 192.168.38.105
    Target MAC address: IntelCor_e2:f7:7f (3c:f0:11:e2:f7:7f)
    Target IP address: 192.168.38.117
```

Inndeholder akkurat de samme spesifikasjonene. Forskjellene er at **Opcode** nå er 2, som indikerer en reply. Target –sender sine spesifikasjoner er reversert, og vi kan også identifisere MAC-adressen til avsender. (Ipaden er nå avsender, og svarer med MAC-adresse)

a)

Nei, IP-adressene som blir brukt i ARP er kun en del av ARP-headeren, og er ikke en egen IP-header. (Litt usikker på dette svaret)

4)

Startet wireshark, og terminalen og kjørte disse kommandoene:

```
root@S540:~# dhclient -v -r
```

```
root@S540:~# dhclient -v
```

Filtrerte for **bootp** i wireshark og tydelig se prossessen:

```
    33 8.946472209   0.0.0.0          255.255.255.255      DHCP     342 DHCP Discover - Transaction ID 0xddcc1876
    34 8.951529326   1.1.1.1          158.39.190.149       DHCP     354 DHCP Offer    - Transaction ID 0xddcc1876
    35 8.952032530   0.0.0.0          255.255.255.255      DHCP     342 DHCP Request  - Transaction ID 0xddcc1876
    36 8.955831952   1.1.1.1          158.39.190.149       DHCP     354 DHCP ACK      - Transaction ID 0xddcc1876
```

Første del av prosessen havner først. Altså **DHCP-discover.** Her sender klienten ut en Broadcast beskjed med forespørsel etter IP.

Rett under kan vi se **DHCP-Offer.** DHCP-serveren har ta mottatt beskjeden og sender "tilbud" tilbake. (Tilbys flere IP-adresser vil hosten velge den første som kommer frem)

Videre ser vi **DHCP-request.** Host sender her tilbake en beskjed om at denne adressen kan bli brukt

Helt nederst finner vi **DHCP ACK.** Dette er siste steg i prosessen. Serveren sender en siste gang tilbake til host. Her blir endelig IP-adresse og en rekke andre nødvendige nettverksparameterere gitt.

a)

Dette finner jeg ved å undersøke IP-headeren. Her kan vi se typen protokoll (som er brukt til overføring av data) som vi jo er ute etter:

```
Protocol: UDP (17)
```

Her kan vi se at DHCP-serveren bruker protokoll av typen **UPD.**

b)

UDP blir altså brukt til å sende disse beskjedene som er en del av DHCP-prosessen. (Det er også det som er hovedoppgaven til denne protokollen. Altså å tilknytte ulike applikasjoner som er en del av et nettverk, slik at de kan kommunisere)

Under **User Datagram Protocol** i wireshark kan vi identifisere portnummer til både server og klient. Den oppgir egentlig portnummeret til sender og mottaker, så for å identifisere server og klient gjelder det å se på riktig del av prosessen.

I eksempelvis første del av **DHCP- prosessen** er PC-en avsender, og serveren mottaker av beskjeden. Kan da lese ut portnummer:

```
▸ User Datagram Protocol, Src Port: 68, Dst Port: 67
```

Klient port -nummer: 67

c)

Server port- nummer: 68

5)

Jeg fant ikke ut hvordan jeg kunne se http request og response mot hiof.no da denne siden benytter seg av https protokoll.

Valgte derfor å prøve mot denne siden : http://scratchpads.eu/explore/sites-list. (Denne lister en rekke nettsteder som enda benytter seg av http protokollen.

Request:

Jeg finner da framen som er merket med GET, og utvider **Hyper Transfer Protocoll delen:**

```
Hypertext Transfer Protocol
  GET /sites/scratchpads.eu/files/css/css_0rdNKyAx9df4tRKovKjnVDsGvZwVuYAKO-m17AAJ1Dk.css HTTP/1.1\r\n
  Host: scratchpads.eu\r\n
  Connection: keep-alive\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.102 Safari/537.36\r\n
  Accept: text/css,*/*;q=0.1\r\n
  Referer: http://scratchpads.eu/explore/sites-list\r\n
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: en,nb-NO;q=0.9,nb;q=0.8,no;q=0.7,nn;q=0.6,en-US;q=0.5\r\n
  \r\n
  [Full request URI: http://scratchpads.eu/sites/scratchpads.eu/files/css/css_0rdNKyAx9df4tRKovKjnVDsGvZwVuYAKO-m17AAJ1Dk.css]
  [HTTP request 1/3]
  [Next request in frame: 1225]
```

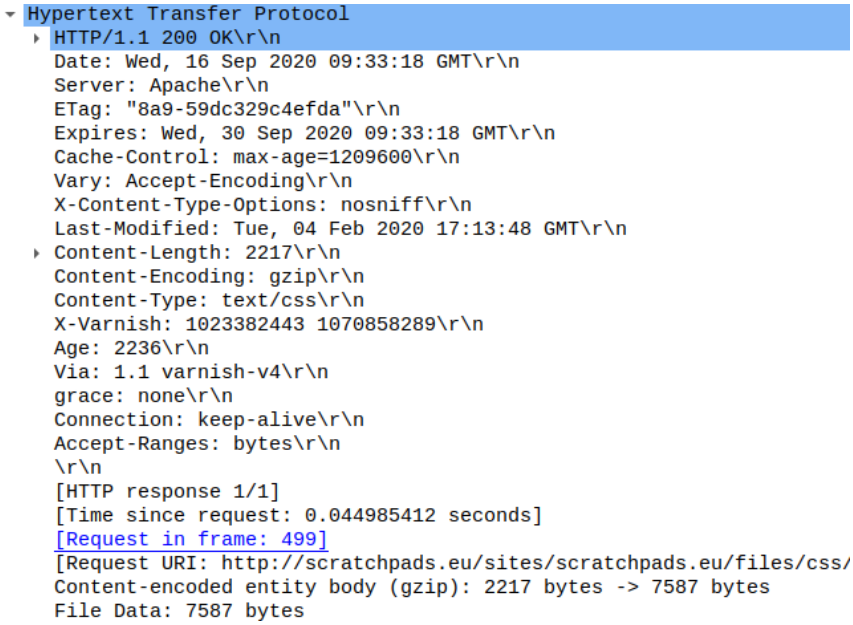
Den første linjen ovenfra kalles **request line.** Denne kan vi se er delt inn i 3. Første del spesifiserer metoden, i dette tilfellet GET. Denne metoden brukes når en browser forespør et objekt fra serveren. Dette objektet er gitt ved URL-en som følger. (vi ser her at det er et stilark (css) til nettsiden som blir forespurt). Siste del spesifiserer hvilken http versjon som blir brukt. I dette tilfellet versjon 1.1

Videre ser vi **Host** denne delen spesifiserer på hvilken host objektet befinner seg. Dette tilfellet serveren som siden jeg forespør befinner seg på. Under er **connection** spesifisert. Dette sier noe om hva som skjer med koblingen etter objektet er sendt. I dette tilfellet er den “keep alive” som sier at koblingen skal holdes gående etter objektet er sendt. Et altvernativ er at denne er “close” da brytes koblingen rett etter objektet er sendt.

User agent er i praksis browser typen som gjør requesten mot serveren. I dette tilfellet er den Mozilla/5.0 (en firefox browser). Nedenfor har vi **Accept-** informasjon. Den første spesifiserer hva slags type objekt som blir tatt imot. (Text/css). Til slutt har vi **Accept –Language.** Denne spesifiserer hvilke språk Browseren foretrekker å motta objektet på. Blant annet her: (Bokmål, Nynorsk, Engelsk)

Response

En response message ser slik ut:



Fra toppen ser vi en **status line.** Her ser vi som tidligere http-versjon. Det viktige er midterste del: **200 OK.** Denne sier at requesten var suksessfull og all forespurt informasjon ble returnert. Denne kan også være **301 Moved Permanently, og 400 Bad Request.** (Objektet har blitt flyttet, eller serveren kunne ikke forstå forespørselen)

Datoen forteller når responsen ble opprettet og sent av serveren. **Server** spesifiserer hva slags type server som sendte responsen. **Expires** forteller hvor lenge objektet er gyldig (før det må forespørres på nytt). **Last modified,** er viktig, og forteller når objektet ble opprettet eller sist endret.

Videre har vi **Content Length** som spesifiserer antall byte objektet er av. **Content-type** er igjen hvilken type objektet er av. Altså text/css. Dette er de viktigste delene i en HTTP –response, men som vi ser av bildet er det også en rekke annen info tilgjengelig.