



Aflevering 2

NGK

Genafleveret: 26 april 2025

Sidst rettet: 26 april 2025

Gruppe nr. 18

Studie nr.	Navn
202309622	Julia Aspoen Rasmussen
202310768	Kalja Blirup Grønning
202300114	Sophia Vagnsbjerg Bach



Øvelse 8

Transport Layer – UDP Client/Server, TCP Client/Server

UDP-Client/Server

Der opsættes en UDP-server på H1 med:

```
netcat -u -l -p 9000
```

Her angiver -u UDP, -l serverfunktion, og -p portnummer.

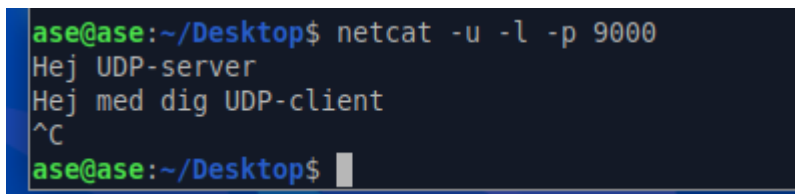
UDP-clienten på H2 startes med:

```
netcat -u 10.0.0.1 9000
```

Her angiver -u UDP, efterfulgt af serverens IP og port.

Analyser vha. Wireshark UDP-protokollen på kommunikationslinjen mellem H1 og H2 når teksten "Hej UDP-server" sendes fra client til server og når teksten "Hej med dig UDP-client" sendes fra server til client. *netcat afsluttes med "Ctrl+c"*

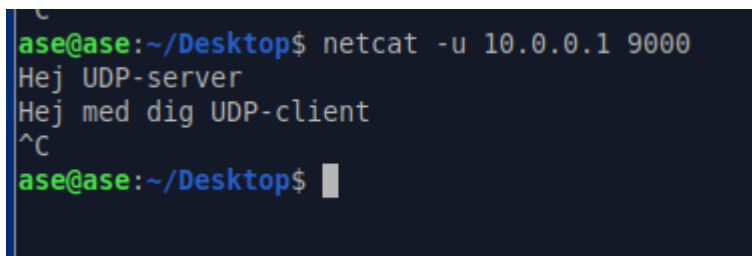
Terminalen set fra H1:



```
ase@ase:~/Desktop$ netcat -u -l -p 9000
Hej UDP-server
Hej med dig UDP-client
^C
ase@ase:~/Desktop$
```

Figur 1 - UDP-server terminal

Terminalen set fra H2:



```
ase@ase:~/Desktop$ netcat -u 10.0.0.1 9000
Hej UDP-server
Hej med dig UDP-client
^C
ase@ase:~/Desktop$
```

Figur 2 - UDP-server klient

Der fokuseres på følgende hændelser i transportlaget (UDP):

- Eventuel connection i starten af kommunikationsforløbet i stil med en TCP-connection?
- UDP-header indhold hvor hver plads i headeren analyseres
- UDP-payload indhold – Indholdet af payload analyseres – er indholdet krypteret, komprimeret eller er indholdet bare ”plain text”?
- Eventuel nedlukning af connection i slutningen af kommunikationsforløbet i stil med en TCP connection?

Er der nogen connection som i TCP i starten af kommunikationsforløbet?

Protokollerne kommer i rækkefølgen vist nedenfor:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.192.55.205	255.255.255.255	BJNP	60	Scanner Command: Discover
2	0.000000130	10.192.55.205	255.255.255.255	BJNP	60	Scanner Command: Discover
3	4.226837648	VMware_f4:be:1f	Broadcast	ARP	60	Who has 10.0.0.1? Tell 10.0.0.2
4	4.227077777	VMware_fc:28:b0	VMware_f4:be:1f	ARP	42	10.0.0.1 is at 00:0c:29:fc:28:b0
5	4.228129035	10.0.0.2	10.0.0.1	UDP	60	49355 → 9000 Len=15
6	13.105393550	10.0.0.1	10.0.0.2	UDP	65	9000 → 49355 Len=23
7	18.464365589	VMware_fc:28:b0	VMware_f4:be:1f	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
8	18.465640462	VMware_f4:be:1f	VMware_fc:28:b0	ARP	60	10.0.0.2 is at 00:0c:29:f4:be:1f

Figur 3 - Udsnit fra wireshark capture

De to UDP-protokoller viser at klienten sender en UDP-pakke til serveren og serveren svarer med en UDP-pakke tilbage til klienten. UDP sender pakkerne direkte og der fremgår ingen connection som i TCP, kun ARP til MAC-adresseopslag.

UDP-header indholdet

Headeren nedenfor viser kommunikationen fra klient til server:

▶	Frame 5: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth1, id 0
▶	Ethernet II, Src: VMware_9d:03:de (00:0c:29:9d:03:de), Dst: VMware_fc:28:b0 (00:0c:29:fc:28:b0)
▶	Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1
▼	User Datagram Protocol, Src Port: 42058, Dst Port: 9000
	Source Port: 42058
	Destination Port: 9000
	Length: 23
	Checksum: 0x7705 [unverified]
	[Checksum Status: Unverified]
	[Stream index: 1]
▶	[Timestamps]
	UDP payload (15 bytes)
▶	Data (15 bytes)

Figur 4 - UDP-header indhold for klient til server

Source Port: 42058 - En tilfældig kildeport, da vi ikke har specificeret den på forhånd.

Destination Port: 9000 - Serverens port, som vi selv specificerede.



Length: 23 - UDP-headeren består af 8 bits, så trækker man 8 fra 23 er der 15 bits tilbage, hvilket stemmer overens med længden på beskeden sendt fra klienten.

Checksum: 0x7705 [unverified] - Checksum er en kontrol der tjekker om det er den rigtige mængde data der bliver sendt. Da den siger "unverified" er det fordi den ikke kan bekræfte det.

Fra server til klient:

```
▼ User Datagram Protocol, Src Port: 9000, Dst Port: 42058
  Source Port: 9000
  Destination Port: 42058
  Length: 31
  Checksum: 0x1433 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 1]
  ▶ [Timestamps]
  UDP payload (23 bytes)
```

Figur 5 - UDP-header indhold for server til klient

Source Port og Destination Port er byttet rundt. Length er 31, hvilket betyder at tekst strengen der blev sendt, er på 23 tegn. Checksum er stadig unverified.

UDP-payload indholdet - Er indholdet krypteret, komprimeret eller plain tekst?

UDP payload for tekstrengen fra klient til server:

```
UDP payload (15 bytes)
▼ Data (15 bytes)
  Data: 48656a205544502d7365727665720a
  [Length: 15]
```

Figur 6 - UDP payload for klient til server

Indholdet har en længde på 15 tegn, hvilket vi bekræftede ovenfor. Data'en er den rå data der bliver sendt over netværket, og består af plain tekst i hexadecimal format. Da indholdet er 15 bits og der ikke er brug for nogen form for nøgle er det hverken komprimeret eller krypteret.



Når vi konverterer fra hexadecimal til ascii fås tekstrengen i det oprindelige format vi skrev det i:

Paste hex code numbers or drop file

48656a205544502d7365727665720a

Character encoding

ASCII

= Convert × Reset ↕ Swap

Hej UDP-server

Figur 7 - Konvertering fra <https://www.rapidtables.com/convert/number/hex-to-ascii.html>

Er der nogen eventuel nedlukning af connection i slutningen af kommunikationsforløbet i stil med en TCP connection?

Nej, som vi også konkluderede ved opstart af connection, så sendes UDP pakkerne direkte.



TCP-Client/Server

TCP-serveren opsættes på H1 med:

```
netcat -l -p 9000
```

Her angiver -l serverfunktion og -p portnummer.

TCP-clienten på H2 startes med:

```
netcat 10.0.0.1 9000
```

Her angives serverens IP og port.

Analyser vha. Wireshark TCP-protokollen på kommunikationslinjen mellem H1 og H2 når følgende sendes:

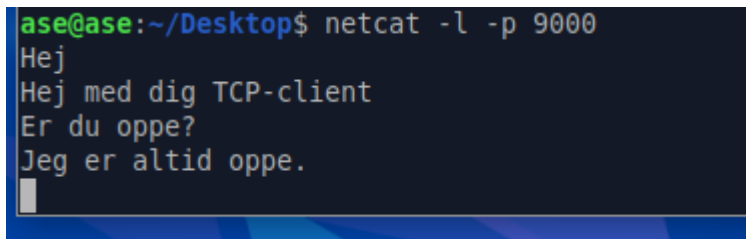
Fra client -> server: "Hej"

Fra server -> client: "Hej med dig TCP-client"

Fra client -> server: "Er du oppe?"

Fra server -> client: "Jeg er altid oppe"

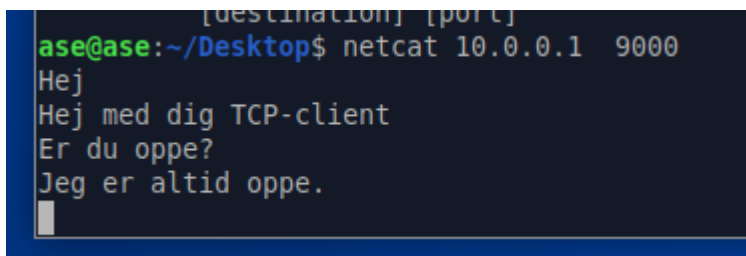
Terminalen set fra H1:



```
ase@ase:~/Desktop$ netcat -l -p 9000
Hej
Hej med dig TCP-client
Er du oppe?
Jeg er altid oppe.
```

Figur 8 - TCP-server terminal

Terminalen set fra H2:



```
[destination] [port]
ase@ase:~/Desktop$ netcat 10.0.0.1 9000
Hej
Hej med dig TCP-client
Er du oppe?
Jeg er altid oppe.
```

Figur 9 - TCP-klient terminal



Oprettelse af connection:

Undersøgelse af Flags, Sequence Counter og Acknowledge Counter

Der undersøges for den første interaktion fra klient til server:

```
Frame 3: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth1, id 0
Ethernet II, Src: VMware_76:9c:8e (00:0c:29:76:9c:8e), Dst: VMware_fc:d7:12 (00:0c:29:fc:d7:12)
Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1
Transmission Control Protocol, Src Port: 54428, Dst Port: 9000, Seq: 0, Len: 0
  Source Port: 54428
  Destination Port: 9000
  [Stream index: 0]
  [Conversation completeness: Incomplete, DATA (15)]
  [TCP Segment Len: 0]
  Sequence Number: 0 (relative sequence number)
  Sequence Number (raw): 3009580286
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 0
  Acknowledgment number (raw): 0
  1010 .... = Header Length: 40 bytes (10)
  Flags: 0x002 (SYN)
  Window: 32120
  [Calculated window size: 32120]
  Checksum: 0x50b0 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
  [Timestamps]
```

Figur 10 - TCP-headeren for første interaktion fra klient til server

Flag: SYN

Sequence Counter: 0.

Acknowledge Counter: 0.

Er det client eller server, der tager initiativ til oprettelsen af TCP-connection?

Det er klienten som beder om at komme på serveren, det ser vi på protokollerne, hvor source er 10.0.0.2 og destination er 10.0.0.1:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.192.116.195	224.0.0.251	MDNS	85	Standard query 0x0000 PTR _microsoft_mcc._tcp.local, "QM" question
2	0.001012327	fe80::23f9:cf24:407...	ff02::fb	MDNS	105	Standard query 0x0000 PTR _microsoft_mcc._tcp.local, "QM" question
3	37.076140696	10.0.0.2	10.0.0.1	TCP	74	54428 → 9000 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=1917827935 TSecr=0 WS=128
4	37.076197953	10.0.0.1	10.0.0.2	TCP	74	9000 → 54428 [ACK] Seq=0 Ack=1 Win=31856 Len=0 MSS=1460 SACK_PERM TSval=1065129721 TSecr=1
5	37.076554951	10.0.0.2	10.0.0.1	TCP	66	54428 → 9000 [ACK] Seq=1 Ack=1 Win=32128 Len=0 TSval=1917827935 TSecr=1065129721
6	37.076827907	10.0.0.1	10.0.0.2	TCP	89	9000 → 54428 [PSH, ACK] Seq=1 Ack=1 Win=31872 Len=23 TSval=1065129722 TSecr=1917827935
7	37.077041736	10.0.0.2	10.0.0.1	TCP	66	54428 → 9000 [ACK] Seq=1 Ack=24 Win=32128 Len=0 TSval=1917827936 TSecr=1065129722
8	40.562711203	10.0.0.2	10.0.0.1	TCP	70	54428 → 9000 [PSH, ACK] Seq=1 Ack=24 Win=32128 Len=4 TSval=1917831421 TSecr=1065129722
9	40.562775777	10.0.0.1	10.0.0.2	TCP	66	9000 → 54428 [ACK] Seq=24 Ack=5 Win=31872 Len=0 TSval=1065133208 TSecr=1917831421
10	42.140739145	VMware_76:9c:8e	VMware_fc:d7:12	ARP	60	Who has 10.0.0.1? Tell 10.0.0.2
11	42.140761803	VMware_fc:d7:12	VMware_76:9c:8e	ARP	42	10.0.0.1 is at 00:0c:29:fc:d7:12
12	150.231813067	10.192.116.195	10.192.127.255	BROWSER	243	Host Announcement ADMINIS-64HIL07, Workstation, Server, NT Workstation
13	260.341457779	fe80::23f9:cf24:407...	ff02::1	ICMPv6	86	Neighbor Advertisement fe80::23f9:cf24:4072:6511 (ovr) is at 04:ed:33:15:ce:30
14	301.045706330	10.192.116.195	224.0.0.251	MDNS	85	Standard query 0x0000 PTR _microsoft_mcc._tcp.local, "QU" question
15	301.047501304	fe80::23f9:cf24:407...	ff02::fb	MDNS	105	Standard query 0x0000 PTR _microsoft_mcc._tcp.local, "QU" question
16	302.049251190	10.192.116.195	224.0.0.251	MDNS	85	Standard query 0x0000 PTR _microsoft_mcc._tcp.local, "QM" question
17	302.050241021	fe80::23f9:cf24:407...	ff02::fb	MDNS	105	Standard query 0x0000 PTR _microsoft_mcc._tcp.local, "QM" question

Figur 11 - Protokolliste for TCP-connection



Overførsel af applikationslags-data:

Flags, Sequence Counter, Acknowledge Counter og Window Size undersøges for alle beskeder der sendes mellem server og klient.

PSH-flaget (Push) indikerer at modtager enheden burde videresende dataen til den modtagende applikation hurtigst muligt. Dette hjælper på at reducere delays og medfører uforstyrret dataflow.

Calculated window size hentyder til den maksimale mængde data i bytes den modtagende enhed er villig til at modtage.

Længden (Length) indikerer antal bytes beskeden fylder, så i tilfældet "Hej" ser vi en længde på 4, 1 byte for hver char, samt en fjerde byte til slut-byte.

De enkelte værdier ses nedenfor.

For "Hej":

```
Frame 8: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface eth1, id 0
Ethernet II, Src: VMware_76:9c:8e (00:0c:29:76:9c:8e), Dst: VMware_fc:d7:12 (00:0c:29:fc:d7:12)
Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1
Transmission Control Protocol, Src Port: 54428, Dst Port: 9000, Seq: 1, Ack: 24, Len: 4
  Source Port: 54428
  Destination Port: 9000
  [Stream index: 0]
  [Conversation completeness: Incomplete, DATA (15)]
  [TCP Segment Len: 4]
  Sequence Number: 1 (relative sequence number)
  Sequence Number (raw): 3009580287
  [Next Sequence Number: 5 (relative sequence number)]
  Acknowledgment Number: 24 (relative ack number)
  Acknowledgment number (raw): 1833438169
  1000 ... = Header Length: 32 bytes (8)
  Flags: 0x018 (PSH, ACK)
  Window: 251
  [Calculated window size: 32128]
  [Window size scaling factor: 128]
  Checksum: 0xcc37 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  [Timestamps]
```

Figur 12 - Header for "Hej" ved TCP-connection

Flags: 0x018 (PSH, ACK)

Length: 4

Sequence Counter: 1

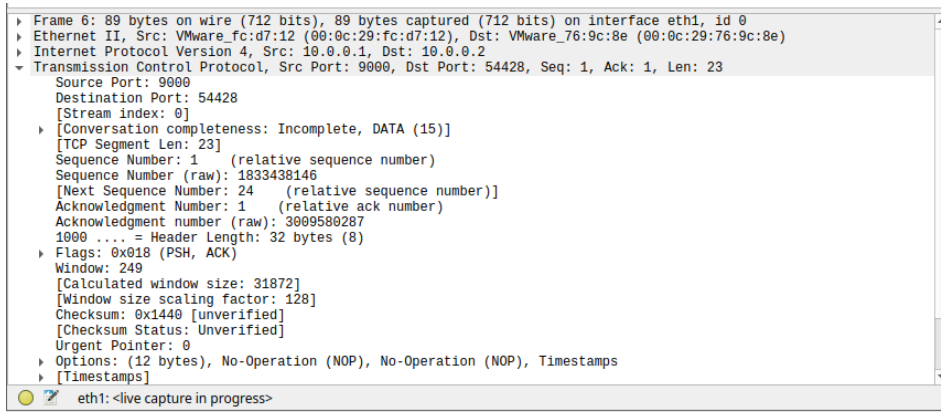
Acknowledge Counter: 24

Window Size: 251

Calculated window size: 32128



For "Hej med dig TCP-client":



Figur 13 - Header for "Hej med dig TCP-client" ved TCP-connection

Flags: 0x018 (PSH, ACK)

Sequence Counter: 1

Acknowledge Counter: 1

Window Size: 249

Calculated window size: 31872



For “Er du oppe?”:

```
Frame 21: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface eth1, id 0
Ethernet II, Src: VMware_76:9c:8e (00:0c:29:76:9c:8e), Dst: VMware_fc:d7:12 (00:0c:29:fc:d7:12)
Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1
Transmission Control Protocol, Src Port: 54428, Dst Port: 9000, Seq: 5, Ack: 24, Len: 12
  Source Port: 54428
  Destination Port: 9000
  [Stream index: 0]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 12]
  Sequence Number: 5 (relative sequence number)
  Sequence Number (raw): 3009580291
  [Next Sequence Number: 17 (relative sequence number)]
  Acknowledgment Number: 24 (relative ack number)
  Acknowledgment number (raw): 1833438169
  1000 .... = Header Length: 32 bytes (8)
  Flags: 0x018 (PSH, ACK)
  Window: 251
  [Calculated window size: 32128]
  [Window size scaling factor: 128]
  Checksum: 0x4ab6 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  [Timestamps]
```

Figur 14 - Header for "Er du oppe?" ved TCP-connection

Flags: 0x018 (PSH, ACK)

Sequence Counter: 5

Acknowledge Counter: 24

Window Size: 251

Calculated window size: 32128

For “Jeg er altid oppe”:

```
Frame 28: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on interface eth1, id 0
Ethernet II, Src: VMware_fc:d7:12 (00:0c:29:fc:d7:12), Dst: VMware_76:9c:8e (00:0c:29:76:9c:8e)
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
Transmission Control Protocol, Src Port: 9000, Dst Port: 54428, Seq: 24, Ack: 17, Len: 19
  Source Port: 9000
  Destination Port: 54428
  [Stream index: 0]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 19]
  Sequence Number: 24 (relative sequence number)
  Sequence Number (raw): 1833438169
  [Next Sequence Number: 43 (relative sequence number)]
  Acknowledgment Number: 17 (relative ack number)
  Acknowledgment number (raw): 3009580303
  1000 .... = Header Length: 32 bytes (8)
  Flags: 0x018 (PSH, ACK)
  Window: 249
  [Calculated window size: 31872]
  [Window size scaling factor: 128]
  Checksum: 0x143c [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  [Timestamps]
```

Figur 15 - Header for "Jeg er altid oppe" ved TCP-connection

Flags: 0x018 (PSH, ACK)

Sequence Counter: 24

Acknowledge Counter: 17

Window Size: 249

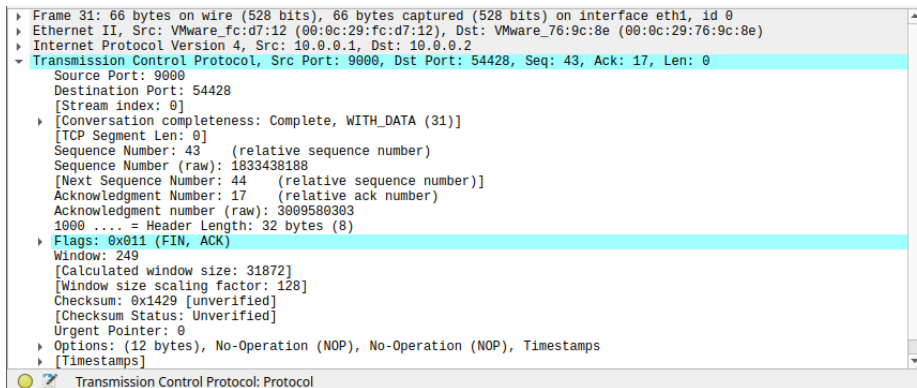
Calculated window size: 31872



Nedlukning af connection:

Flags, Sequence Counter og Acknowledge Counter undersøges.

Server header ved nedlukning:



Figur 16 - Header for server ved nedlukning af TCP-connection

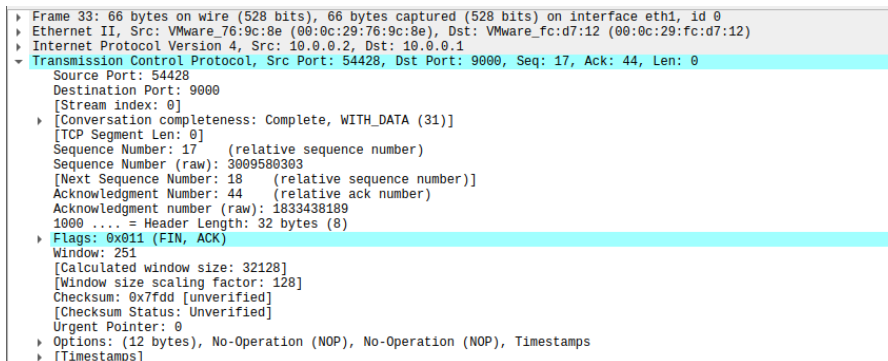
Flags: 0x11 (FIN, ACK)

Sequence Counter: 43

Acknowledge Counter: 17

Window Size: 249

Client header ved nedlukning:



Figur 17 - Header for client ved nedlukning af TCP-connection

Flags: 0x011 (FIN, ACK)

Sequence Counter: 17

Acknowledge Counter: 44

Window Size: 251



Er det client eller server, der tager initiativ til nedlukningen af TCP connection?

21	584.412637161	10.0.0.1	10.0.0.2	TCP	66 9000 → 54428	[FIN, ACK]	Seq=0 Ack=
22	584.412637161	10.0.0.1	10.0.0.2	TCP	66 9000 → 54428	[ACK]	Seq=24 Ack=17 W
28	592.415744785	10.0.0.1	10.0.0.2	TCP	85 9000 → 54428	[PSH, ACK]	Seq=24 Ack
29	592.416259273	10.0.0.2	10.0.0.1	TCP	66 54428 → 9000	[ACK]	Seq=17 Ack=43 W
31	596.810178566	10.0.0.1	10.0.0.2	TCP	66 9000 → 54428	[FIN, ACK]	Seq=43 Ack
32	596.851694547	10.0.0.2	10.0.0.1	TCP	66 54428 → 9000	[ACK]	Seq=17 Ack=44 W
33	602.783479993	10.0.0.2	10.0.0.1	TCP	66 54428 → 9000	[FIN, ACK]	Seq=17 Ack
34	602.783509007	10.0.0.1	10.0.0.2	TCP	66 9000 → 54428	[ACK]	Seq=44 Ack=18 W

Figur 18 - Protokolliste ved nedlukning

Serveren initialiserer lukning da den sender den første [FIN, ACK].

Hvilke afsluttende værdier har Sequence Counter og Acknowledge Counter på Server?

Sequence Counter: 43

Acknowledge Counter: 17

Hvilke afsluttende værdier har Sequence Counter og Acknowledge Counter på Klient?

Bemærk at H1 og H2, hver har et sæt af Sequence / Acknowledge counters – i alt 4 counterer!

Sequence Counter: 17

Acknowledge Counter: 44

Indsæt Flow Graph fra Wireshark der viser hvordan de fire afsluttende værdier er fremkommet

Flow Graph over forløbet i Wireshark:



Figur 19 - Flow Graph fra Wireshark

Tiderne på Flow Graph'en matcher ikke med tiderne på de ovenstående figurer, da denne Flow Graph er taget fra en anden connection.

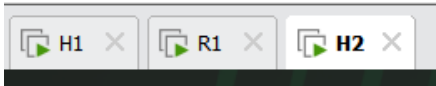


Øvelse 9

Opsætning af en simpel router (forwarder)

1. Konfigurer et internetwork bestående af en host (H1) på et LAN-segment, som via en simpel router (R1) skal kunne kommunikere med en host (H2) på et andet LAN-segment.

De tre virtuelle maskiner side om side i VMWare:



Figur 20 - De tre virtuelle maskiner

De virtuelle maskiner er konfigureret således:

	Address	Netmask	Gateway
H1	10.0.0.1	24	10.0.0.2
R1	10.0.0.2	24	
R1	10.9.8.2	24	
H2	10.9.8.1	24	10.9.8.2

Tabel 1 - Oversigt over netværksopsætningerne

2. Den virtuelle maskine R1 konfigureres til at have router/forwarder-tilstand vha. kommandoen: `sudo sysctl -w net.ipv4.ip_forward=1`

Indholdet af filen `ip_forward` kan læses/tjekkes vha. kommandoen: `cat /proc/sys/net/ipv4/ip_forward`

Kommandoerne kørt i terminalen i R1:

```
Terminal - ase@ase: ~/Desktop
File Edit View Terminal Tabs Help
ase@ase:~/Desktop$ sudo sysctl -w net.ipv4.ip_forward=1
[sudo] password for ase:
net.ipv4.ip_forward = 1
ase@ase:~/Desktop$ cat /proc/sys/net/ipv4/ip_forward
1
ase@ase:~/Desktop$
```

Figur 21 - R1 konfigureret til at være i router tilstand



3. Verificer at de 2 hosts H1 og H2, som er tilsluttet hvert sit LAN-segment, kan kommunikere indbyrdes vha. Linux-kommandoen ping

H2 ping'et fra H1:

```
Terminal - ase@ase: ~/Desktop
File Edit View Terminal Tabs Help
ase@ase:~/Desktop$ ping 10.9.8.1
PING 10.9.8.1 (10.9.8.1) 56(84) bytes of data.
64 bytes from 10.9.8.1: icmp_seq=1 ttl=63 time=2.16 ms
64 bytes from 10.9.8.1: icmp_seq=2 ttl=63 time=1.92 ms
64 bytes from 10.9.8.1: icmp_seq=3 ttl=63 time=1.37 ms
64 bytes from 10.9.8.1: icmp_seq=4 ttl=63 time=1.89 ms
64 bytes from 10.9.8.1: icmp_seq=5 ttl=63 time=1.35 ms
64 bytes from 10.9.8.1: icmp_seq=6 ttl=63 time=1.73 ms
```

Figur 22 - H1 pinger H2

H1 ping'et fra H2:

```
Terminal - ase@ase: ~/Desktop
File Edit View Terminal Tabs Help
ase@ase:~/Desktop$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=63 time=1.84 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=63 time=1.13 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=63 time=1.68 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=63 time=0.919 ms
```

Figur 23 - H2 pinger H1

4. Hvorfor skal "Gateway" konfigureres i hhv. H1 og H2?

Når man pinger en IP-adresse, og den ikke finder den, så søger den gateway i stedet for. Så når H1 ikke kan finde 10.9.8.1, så finder den 10.0.0.2 ved routeren, som finder 10.9.8.1 for den. Det samme gælder den anden vej.



5. Analyser hændelser på de to LAN-segmenter på netværkslaget vha. Wireshark.

Wireshark startes på R1 og der udføres en enkelt ping fra H1 via R1 til H2.

No.	Time	Source	Destination	Protocol	Length	Info
2	2.811000283	10.0.0.1	10.9.8.1	ICMP	98	Echo (ping) request id=0x0a6e, seq=1/256, ttl=64 (no response found!)
3	2.811736932	10.0.0.1	10.9.8.1	ICMP	98	Echo (ping) request id=0x0a6e, seq=1/256, ttl=63 (reply in 4)
4	2.811737077	10.9.8.1	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0a6e, seq=1/256, ttl=64 (request in 3)
5	2.811843363	10.9.8.1	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0a6e, seq=1/256, ttl=63

Figur 24 - 4 ICMP telegrammer ved ping fra H1 via R2 til H2

Hvorfor kommer der 4 ICMP-telegrammer?

Der bliver brugt 4 adresser når requesten skal fra H1 til H2, men skal igennem R1 først.

10.0.0.1 → 10.0.0.2

10.9.8.2 → 10.9.8.1

10.9.8.1 → 10.9.8.2

10.0.0.2 → 10.0.0.1



De 4 ICMP telegrammer analyseres på netværkslaget. Er IP-adresserne konstante på netværkslaget, eller udskiftes de undervejs, når en datapakke (ICMP) sendes fra H1 til H2 (IP-adresserne på de 4 ICMP pakker undersøges)? – dokumenteres med Wireshark!

Det første telegram fortæller at den ikke kunne finde en direkte forbindelse til destinationen.

```
▶ Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
▶ Ethernet II, Src: VMware_fc:28:b0 (00:0c:29:fc:28:b0), Dst: VMware_f4:be:1f (00:0c:29:f4:be:1f)
▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.9.8.1
▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x0955 [correct]
  [Checksum Status: Good]
  Identifier (BE): 2670 (0x0a6e)
  Identifier (LE): 28170 (0x6e0a)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
▼ [No response seen]
  ▶ [Expert Info (Warning/Sequence): No response seen to ICMP request]
  Timestamp from icmp data: Mar 24, 2025 12:44:50.506762000 CET
  [Timestamp from icmp data (relative): 0.051681123 seconds]
  ▶ Data (40 bytes)
```

Figur 25 - Første ICMP telegram

Det næste telegram viser at en request bliver sendt igen og at responsen findes i det næste telegram:

```
▶ Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
▶ Ethernet II, Src: VMware_f4:be:29 (00:0c:29:f4:be:29), Dst: VMware_9d:03:de (00:0c:29:9d:03:de)
▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.9.8.1
▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x0955 [correct]
  [Checksum Status: Good]
  Identifier (BE): 2670 (0x0a6e)
  Identifier (LE): 28170 (0x6e0a)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Response frame: 4]
  Timestamp from icmp data: Mar 24, 2025 12:44:50.506762000 CET
  [Timestamp from icmp data (relative): 0.052417772 seconds]
  ▶ Data (40 bytes)
```

Figur 26 - Det andet ICMP telegram

Det tredje telegram bekræfter at ping requesten fra det forrige telegram er gået igennem:

```
▶ Frame 4: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
▶ Ethernet II, Src: VMware_9d:03:de (00:0c:29:9d:03:de), Dst: VMware_f4:be:29 (00:0c:29:f4:be:29)
▶ Internet Protocol Version 4, Src: 10.9.8.1, Dst: 10.0.0.1
▼ Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
  Code: 0
  Checksum: 0x1155 [correct]
  [Checksum Status: Good]
  Identifier (BE): 2670 (0x0a6e)
  Identifier (LE): 28170 (0x6e0a)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Request frame: 3]
  [Response time: 0.000 ms]
  Timestamp from icmp data: Mar 24, 2025 12:44:50.506762000 CET
  [Timestamp from icmp data (relative): 0.052417917 seconds]
  ▶ Data (40 bytes)
```

Figur 27 - Tredje ICMP telegram



Den fjerde vises som et "reply", men indeholder ikke en request frame som den forrige respons. Dette kunne tyde på at den kun bekræfter tilstedeværelsen af adressen. Da den første siger "No response found" kunne den tomme reply være responsen til den første request. Hvis der ikke var forekommet en fejl i den første request, ville der kun være en enkelt request og respons:

```
▶ Frame 5: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
▶ Ethernet II, Src: VMware_f4:be:1f (00:0c:29:f4:be:1f), Dst: VMware_fc:28:b0 (00:0c:29:fc:28:b0)
▶ Internet Protocol Version 4, Src: 10.9.8.1, Dst: 10.0.0.1
▼ Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
  Code: 0
  Checksum: 0x1155 [correct]
  [Checksum Status: Good]
  Identifier (BE): 2670 (0x0a6e)
  Identifier (LE): 28170 (0x6e0a)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  Timestamp from icmp data: Mar 24, 2025 12:44:50.506762000 CET
  [Timestamp from icmp data (relative): 0.052524203 seconds]
▶ Data (40 bytes)
```

Figur 28 - Fjerde ICMP telegram



Øvelse 10

Opsætning af en simpel router (forwarder), og netværksanalyse af ARP-protokollens funktionalitet.

6. Giv en kort forklaring på ARP-protokollens formål og virkemåde?

ARP (Address Resolution Protocol) bruges til at oversætte en IP-adresse til en MAC-adresse i et lokalt netværk. Når en enhed (f.eks. H1) vil kommunikere med en anden enhed (f.eks. H2) i samme netværk, men kun har dens IP-adresse, sender den en ARP-request i broadcast-format (ff:ff:ff:ff:ff:ff – her sendes det til alle). Den enhed, der har den ønskede IP-adresse, svarer med sin MAC-adresse via en ARP-reply.

ARP hjælper med at forbinde IP-kommunikation med den rigtige fysiske hardwareadresse i et netværk.

7. Analyser hændelser på de to LAN-segmenter på linklaget vha. Wireshark.

Som i den tidligere øvelse udføres en enkelt ping fra H1 via R1 til H2:

```
Terminal - ase@ase: ~/Desktop
File Edit View Terminal Tabs Help
ase@ase:~/Desktop$ ping -c 1 10.9.8.1
PING 10.9.8.1 (10.9.8.1) 56(84) bytes of data.
64 bytes from 10.9.8.1: icmp_seq=1 ttl=63 time=2.75 ms

--- 10.9.8.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.749/2.749/2.749/0.000 ms
ase@ase:~/Desktop$
```

Figur 29 - Ping fra H1 til H2 via R1 i terminalen

Dokumentation i wireshark fremgår i besvarelserne nedenfor.



Analyse af relevante hændelser for ARP-protokollen (dokumenteres med Wireshark)

Hvilke ARP-telegammer observeres?

Protokollisten i wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
82	6.898846042	VMware_f4:be:1f	VMware_fc:28:b0	ARP	60	Who has 10.0.0.1? Tell 10.0.0.2
83	6.898869702	VMware_fc:28:b0	VMware_f4:be:1f	ARP	42	10.0.0.1 is at 00:0c:29:fc:28:b0
84	6.900011692	VMware_f4:be:29	VMware_9d:03:de	ARP	60	Who has 10.9.8.1? Tell 10.9.8.2
85	6.900994410	VMware_9d:03:de	VMware_f4:be:29	ARP	60	10.9.8.1 is at 00:0c:29:9d:03:de
86	7.036550164	VMware_fc:28:b0	VMware_f4:be:1f	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
87	7.037694412	VMware_f4:be:1f	VMware_fc:28:b0	ARP	60	10.0.0.2 is at 00:0c:29:fc:28:b0
88	7.372781784	VMware_9d:03:de	VMware_f4:be:29	ARP	60	Who has 10.9.8.2? Tell 10.9.8.1
89	7.373352865	VMware_f4:be:29	VMware_9d:03:de	ARP	60	10.9.8.2 is at 00:0c:29:9d:03:de

Figur 30 - Wireshark protokolliste med fokus på ARP-telegammer

Fra frame 82 til 89 forekommer 8 ARP-telegammer. Disse telegammer viser linking mellem IP- og MAC-adresserne for hver af de 3 virtuelle maskiner.

De første fire telegammer viser R1 ETH1 (f4:be:1f) der linker IP- og MAC-adressen for H1, hvorefter R1 ETH2 (f4:be:29) linker IP og MAC for H2.

Derefter ses det på de næste 2 telegammer, at H1 (fc:28:b0) linker IP og MAC for R1 ETH1.

På de sidste to telegammer linker H2 (9d:03:de) R1 ETH2's adresser.

Dokumenter kobling mellem IP-adresser og MAC-adresser? Fokuser på MAC-a, MAC-b, MAC-c og MAC-d kombineret med ARP-protokollen.

Ud fra protokollisten fra forrige figur, laver vi en tabel til at demonstrere koblingen mellem IP-adresser og MAC-adresser:

Navn	MAC-adresse	IP-adresse
H1 (MAC-a)	00:0c:29:fc:28:b0	10.0.0.1
R1-ETH1 (MAC-b)	00:0c:29:f4:be:1f	10.0.0.2
R1-ETH2 (MAC-c)	00:0c:29:f4:be:29	10.9.8.2
H2 (MAC-d)	00:0c:29:9d:03:de	10.9.8.1

Tabel 2 - Kobling mellem IP-adresser og MAC-adresser



De 4 ICMP-telegrammerne analyseres på linklaget

Er MAC-adresserne konstante på linklaget, eller udskiftes de undervejs, når en datapakke (ICMP) sendes fra H1 til H2 (MAC adresserne på de 4 ICMP pakker undersøges)? - dokumenteres med Wireshark!

MAC-adresser ændres undervejs på hvert netværkssegment, når en ICMP-pakke sendes fra H1 til H2 via R1.

Dette kan ses på figurerne nedenunder:

No.	Time	Source	Destination	Protocol	Length	Info
17	1.860541106	10.0.0.1	10.9.8.1	ICMP	98	Echo (ping) request id=0x0a61, seq=1/256, ttl=64 (no response found!)
18	1.861999613	10.0.0.1	10.9.8.1	ICMP	98	Echo (ping) request id=0x0a61, seq=1/256, ttl=63 (reply in 19)
19	1.862537496	10.9.8.1	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0a61, seq=1/256, ttl=64 (request in 18)
20	1.863020110	10.9.8.1	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0a61, seq=1/256, ttl=63

▶ Frame 17: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
▼ Ethernet II, Src: VMware_fc:28:b0 (00:0c:29:fc:28:b0), Dst: VMware_f4:be:1f (00:0c:29:f4:be:1f)
 ▶ Destination: VMware_f4:be:1f (00:0c:29:f4:be:1f)
 ▶ Source: VMware_fc:28:b0 (00:0c:29:fc:28:b0)

Figur 31 wireshark undersøgelse af MAC-adresser på ICMP 1

No.	Time	Source	Destination	Protocol	Length	Info
17	1.860541106	10.0.0.1	10.9.8.1	ICMP	98	Echo (ping) request id=0x0a61, seq=1/256, ttl=64 (no response found!)
18	1.861999613	10.0.0.1	10.9.8.1	ICMP	98	Echo (ping) request id=0x0a61, seq=1/256, ttl=63 (reply in 19)
19	1.862537496	10.9.8.1	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0a61, seq=1/256, ttl=64 (request in 18)
20	1.863020110	10.9.8.1	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0a61, seq=1/256, ttl=63

▶ Frame 18: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
▼ Ethernet II, Src: VMware_f4:be:29 (00:0c:29:f4:be:29), Dst: VMware_9d:03:de (00:0c:29:9d:03:de)
 ▶ Destination: VMware_9d:03:de (00:0c:29:9d:03:de)
 ▶ Source: VMware_f4:be:29 (00:0c:29:f4:be:29)

Figur 32 - wireshark undersøgelse af MAC-adresser på ICMP 2

No.	Time	Source	Destination	Protocol	Length	Info
17	1.860541106	10.0.0.1	10.9.8.1	ICMP	98	Echo (ping) request id=0x0a61, seq=1/256, ttl=64 (no response found!)
18	1.861999613	10.0.0.1	10.9.8.1	ICMP	98	Echo (ping) request id=0x0a61, seq=1/256, ttl=63 (reply in 19)
19	1.862537496	10.9.8.1	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0a61, seq=1/256, ttl=64 (request in 18)
20	1.863020110	10.9.8.1	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0a61, seq=1/256, ttl=63

▶ Frame 19: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
▼ Ethernet II, Src: VMware_9d:03:de (00:0c:29:9d:03:de), Dst: VMware_f4:be:29 (00:0c:29:f4:be:29)
 ▶ Destination: VMware_f4:be:29 (00:0c:29:f4:be:29)
 ▶ Source: VMware_9d:03:de (00:0c:29:9d:03:de)

Figur 33 - wireshark undersøgelse af MAC-adresser på ICMP 3

No.	Time	Source	Destination	Protocol	Length	Info
17	1.860541106	10.0.0.1	10.9.8.1	ICMP	98	Echo (ping) request id=0x0a61, seq=1/256, ttl=64 (no response found!)
18	1.861999613	10.0.0.1	10.9.8.1	ICMP	98	Echo (ping) request id=0x0a61, seq=1/256, ttl=63 (reply in 19)
19	1.862537496	10.9.8.1	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0a61, seq=1/256, ttl=64 (request in 18)
20	1.863020110	10.9.8.1	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0a61, seq=1/256, ttl=63

▶ Frame 20: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth1, id 0
▼ Ethernet II, Src: VMware_f4:be:1f (00:0c:29:f4:be:1f), Dst: VMware_fc:28:b0 (00:0c:29:fc:28:b0)
 ▶ Destination: VMware_fc:28:b0 (00:0c:29:fc:28:b0)
 ▶ Source: VMware_f4:be:1f (00:0c:29:f4:be:1f)

Figur 34 - wireshark undersøgelse af MAC-adresser på ICMP 4

På de fire ovenstående figurer, hvori MAC-adresserne undersøges, ses det at destinationen samt source adresserne ændres undervejs. Dette skyldes at R1 benyttes som gateway mellem H1 og H2, da H1 sender til R1, efterfulgt af R1 til H2 og tilbage igen.