

SW3NGK

Besvarelse af opgave

1.modul - Øvelse 2 - 7

Afleveret: [11-10-21]

Afleveret af: Rasmus Baunvig Aagaard

Gruppe: 22

Deltagere i afleveringen

Studienummer	AU-id	Navn	Studieretning
201510642	AU532459	Rasmus Baunvig Aagaard	SW

Indhold

Øvelse 2 - OPSÆTNING AF EN ETHERNET-FORBINDELSE MELLEM 2 HOSTS	5
Indledning:.....	5
Opgave 1:.....	5
Opgave 2:.....	5
Opgave 3:.....	6
Opgave 4:.....	6
Øvelse 3 - Delay	7
Indledning:.....	7
Opgave 1:.....	7
Opgave 2:.....	7
Opgave 3:.....	8
Opgave 4:.....	8
Opgave 5:.....	8
Opgave 6:.....	11
Opgave 7:.....	11
Øvelse 4 - HTTP Client/Server.....	12
Indledning:.....	12
Opgave 1:.....	12
Opgave 2:.....	15
Opgave 3:.....	15
Opgave 4:.....	16
Opgave 5:.....	19
Øvelse 5 - DNS client	23
Indledning:.....	23
Opgave 1:.....	23
Konklusion:	26
Øvelse 6 - Application Layer, Transport Layer, TCP Socket Programming	27
Indledning:.....	27
Opgave 1:.....	27
Opgave 2:.....	28
Opgave 3:.....	29
Opgave 3:.....	30
Øvelse 7 - UDP/IP_Socket_Programming.....	31

Indledning:.....	31
Opgave 1:.....	31
Opgave 2:.....	32
Opgave 3:.....	32
Opgave 4:.....	32
Opgave 5:.....	33
Billagsliste	33

Figur 1 - virtual machine settings	5
Figur 2 - Tildel ip-adress og netmask.....	5
Figur 3 - Tildelte ip-adresser.....	6
Figur 4 - Ping mellem H1 og H2	6
Figur 5 - Wireshark m. ping	6
Figur 6 - Tid ved 10 ping	7
Figur 7 - Ping til www.google.dk.....	8
Figur 8 - 10 ping til www.google.dk.....	8
Figur 9 - TCP three-way handshake.....	8
Figur 10 - Time display format.....	9
Figur 11 - Wireshark conversation filter	10
Figur 12 - TCP filter	10
Figur 13 - Responsetid til australsk web-side	11
Figur 14 - Responsetime	12
Figur 15 - DNS opslag og response	12
Figur 16 - Http GET	13
Figur 17 - Http response OK	14
Figur 18 - LAN forbindelse	15
Figur 19 - http/1.0 a.....	16
Figur 20 - http/1.1 b	16
Figur 21 - http/1.1 c.....	17
Figur 22 - Server-client	17
Figur 23 - Server version.....	18
Figur 24 - 3 time image.jpeg.....	19
Figur 25 - http version page1.html	20
Figur 26 - http response page1.html	20
Figur 27 - 3 times jpeg	21
Figur 28 - Terminal response.....	21
Figur 29 - host options.....	23
Figur 30 - host kommando	24
Figur 31 - nslookup options	25
Figur 32 - command nslookup	26
Figur 33 - TCP Serverside	28
Figur 34 - TCP Client.....	29
Figur 35 - TCP Server - client.....	29
Figur 36 - TCP send image	29
Figur 37 - TCP compare diff	30
Figur 38 - UDP server client opstilling	33

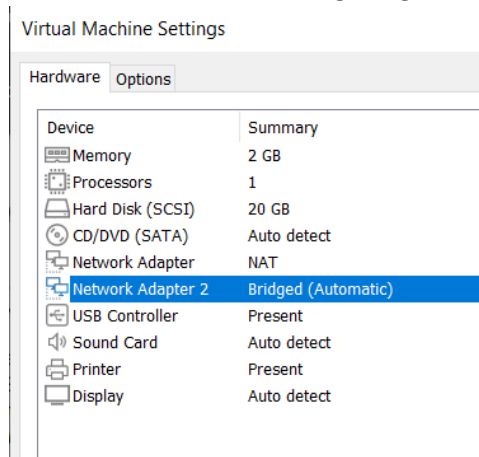
Øvelse 2 - OPSÆTNING AF EN ETHERNET-FORBINDELSE MELLEM 2 HOSTS

Indledning:

Formålet med øvelsen er at få to virtuelle Linux-computere til at kommunikere indbyrdes via Ethernet.

Opgave 1:

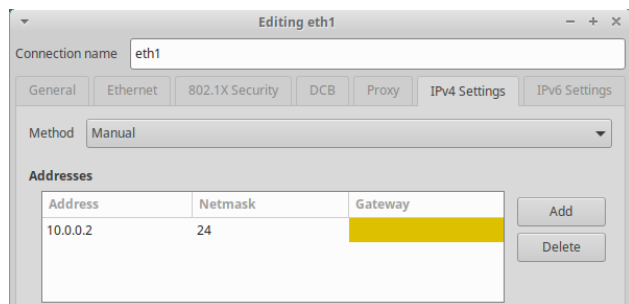
På hver enkel virtuelle Linux-computer oprettes en ekstra Ethernet-port, eth1, under "Edit virtual machine settings" Figur 1



Figur 1 - virtual machine settings

Opgave 2:

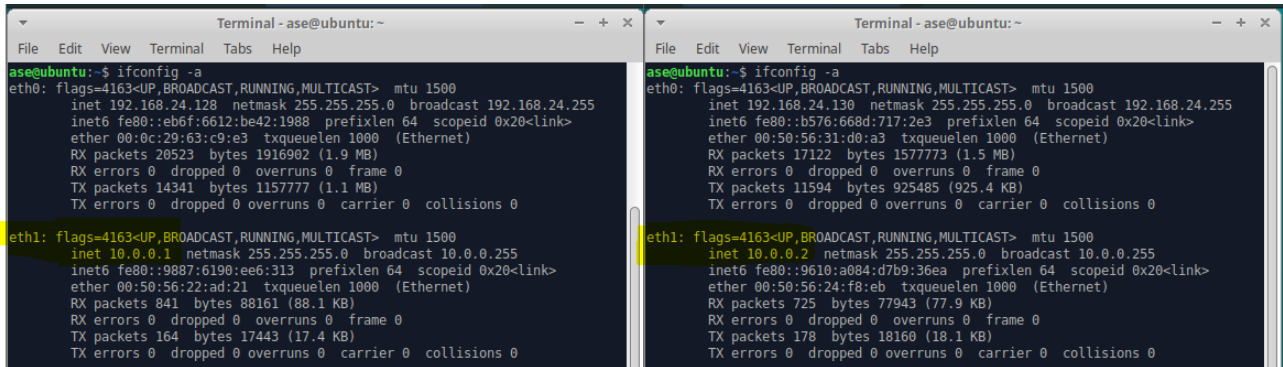
Der oprettes nu mulighed for kommunikation mellem de to virtuelle Linux-computere vist ved Figur 2: "eth1 på H1 får IP-adressen 10.0.0.1, netmask 255.255.255.0 og eth1 på H2 får IP-adressen 10.0.0.2, netmask 255.255.255.0"



Figur 2 - Tildel ip-adress og netmask

Opgave 3:

Der verificeres nu kommunikation mellem de to virtuelle Linux-computere ved at "ping" mellem H1 og H2 ved den tildelte ip-adresse. Resultatet vises i Figur 4



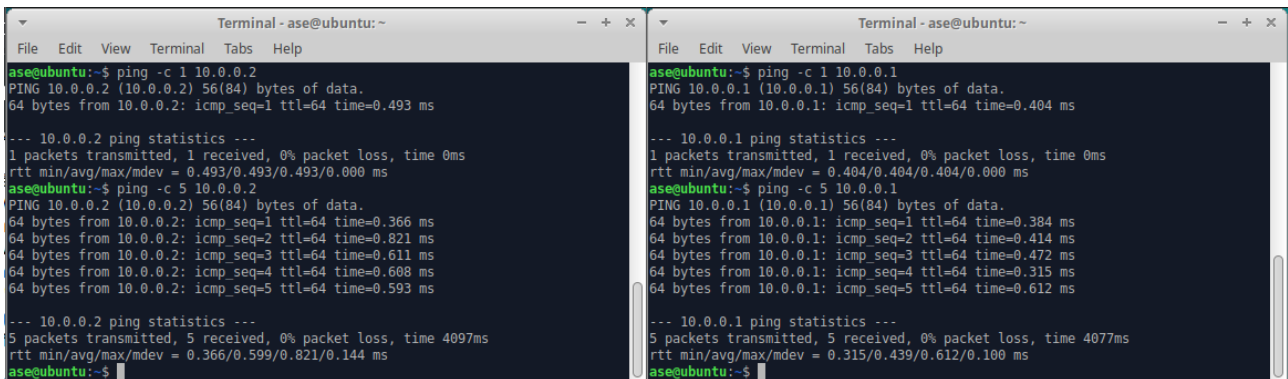
```
ase@ubuntu:~$ ifconfig -a
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.24.128 netmask 255.255.255.0 broadcast 192.168.24.255
    inet6 fe80::eb6f:6612:be42:1988 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:63:c9:e3 txqueuelen 1000 (Ethernet)
    RX packets 20523 bytes 1916902 (1.9 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 14341 bytes 1157777 (1.1 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.255.255.0 broadcast 10.0.0.255
    inet6 fe80::9887:6190:ee6:313 prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:22:ad:21 txqueuelen 1000 (Ethernet)
    RX packets 841 bytes 88161 (88.1 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 164 bytes 17443 (17.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
ase@ubuntu:~$ ifconfig -a
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.24.130 netmask 255.255.255.0 broadcast 192.168.24.255
    inet6 fe80::b576:668d:717:2e3 prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:31:d0:a3 txqueuelen 1000 (Ethernet)
    RX packets 17122 bytes 1577773 (1.5 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 11594 bytes 925485 (925.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.2 netmask 255.255.255.0 broadcast 10.0.0.255
    inet6 fe80::9610:a084:d7b9:36ea prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:24:f8:eb txqueuelen 1000 (Ethernet)
    RX packets 725 bytes 77943 (77.9 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 178 bytes 18160 (18.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figur 3 - Tildelte ip-adresser



```
ase@ubuntu:~$ ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.493 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.493/0.493/0.493/0.000 ms

ase@ubuntu:~$ ping -c 5 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.366 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.821 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.611 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.608 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.593 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4097ms
rtt min/avg/max/mdev = 0.366/0.599/0.821/0.144 ms

ase@ubuntu:~$
```

```
ase@ubuntu:~$ ping -c 1 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=64 time=0.404 ms

--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.404/0.404/0.404/0.000 ms

ase@ubuntu:~$ ping -c 5 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=64 time=0.384 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.414 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.472 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.315 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.612 ms

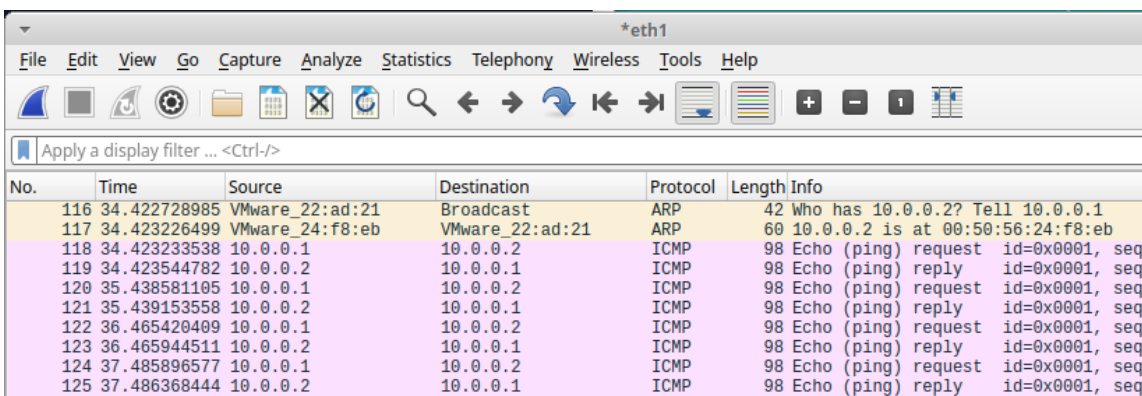
--- 10.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4077ms
rtt min/avg/max/mdev = 0.315/0.439/0.612/0.100 ms

ase@ubuntu:~$
```

Figur 4 - Ping mellem H1 og H2

Opgave 4:

Wireshark benyttes til at opsamle hændelser på netværket mellem H1 og H2 og vises i Figur 5



No.	Time	Source	Destination	Protocol	Length	Info
116	34.422728985	VMware_22:ad:21	Broadcast	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
117	34.423226499	VMware_24:f8:eb	VMware_22:ad:21	ARP	60	10.0.0.2 is at 00:50:56:24:f8:eb
118	34.423233538	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0001, seq=
119	34.423544782	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0001, seq=
120	35.438581105	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0001, seq=
121	35.439153558	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0001, seq=
122	36.465420409	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0001, seq=
123	36.465944511	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0001, seq=
124	37.485896577	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0001, seq=
125	37.486368444	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0001, seq=

Figur 5 - Wireshark m. ping

Øvelse 3 - Delay

Indledning:

Formålet med øvelsen er at anvende Wireshar til analyse af netværk og samtidig undersøge http-protokollens funktionalitet.

Opgave 1:

Mål tiden ved ping mellem H1 og H2

Ud fra Figur 4, findes tiderne for hhv. 1 og 5 ping.

For 1 ping:

--- 10.0.0.2 ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.493/0.493/0.493/0.000 ms

for 5 ping:

--- 10.0.0.2 ping statistics ---

5 packets transmitted, 5 received, 0% packet loss, time 4097ms

rtt min/avg/max/mdev = 0.366/0.599/0.821/0.144 ms

Opgave 2:

Mål tiden ved 10 følgende ping mellem H1 og H2. Resultaterne vises nedenfor i Figur 6.

```
ase@ubuntu:~$ ping -c 10 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.412 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.509 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.576 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.682 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.503 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.487 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.512 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.582 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.600 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.608 ms
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9209ms
rtt min/avg/max/mdev = 0.412/0.547/0.682/0.072 ms
ase@ubuntu:~$
```

Figur 6 - Tid ved 10 ping

Opgave 3:

Mål tiden ved ping kommandoen til en server på internettet. Resultaterne for tiderne ved ping til www.google.dk vises nedenfor i Figur 7.

```
ase@ubuntu:~$ ping -c 1 www.google.dk
PING www.google.dk (142.250.185.67) 56(84) bytes of data.
64 bytes from fra16s48-in-f3.1e100.net (142.250.185.67): icmp_seq=1 ttl=128 time
=25.1 ms

--- www.google.dk ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 25.142/25.142/25.142/0.000 ms
ase@ubuntu:~$
```

Figur 7 - Ping til www.google.dk

Opgave 4:

Mål tiden ved 10 ping til www.google.dk. Resultaterne vises nedenfor i Figur 8

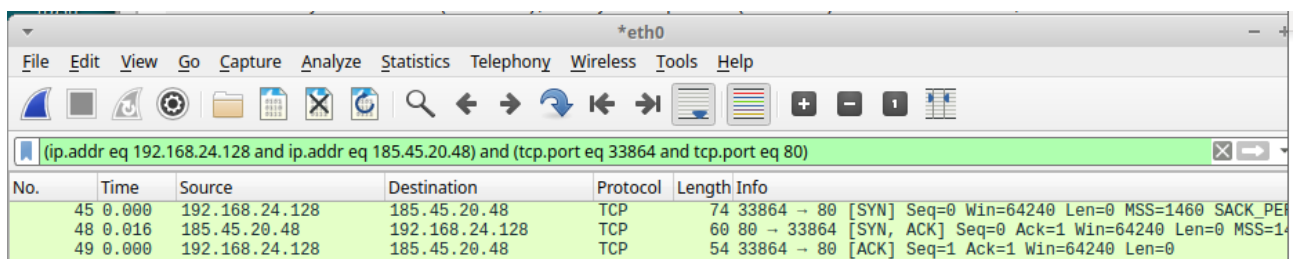
```
ase@ubuntu:~$ ping -c 10 www.google.dk
PING www.google.dk (142.250.186.131) 56(84) bytes of data.
64 bytes from fra24s07-in-f3.1e100.net (142.250.186.131): icmp_seq=1 ttl=128 time=25.4 ms
64 bytes from fra24s07-in-f3.1e100.net (142.250.186.131): icmp_seq=2 ttl=128 time=25.9 ms
64 bytes from fra24s07-in-f3.1e100.net (142.250.186.131): icmp_seq=3 ttl=128 time=34.6 ms
64 bytes from fra24s07-in-f3.1e100.net (142.250.186.131): icmp_seq=4 ttl=128 time=26.8 ms
64 bytes from fra24s07-in-f3.1e100.net (142.250.186.131): icmp_seq=5 ttl=128 time=25.7 ms
64 bytes from fra24s07-in-f3.1e100.net (142.250.186.131): icmp_seq=6 ttl=128 time=27.4 ms
64 bytes from fra24s07-in-f3.1e100.net (142.250.186.131): icmp_seq=7 ttl=128 time=27.7 ms
64 bytes from fra24s07-in-f3.1e100.net (142.250.186.131): icmp_seq=8 ttl=128 time=26.9 ms
64 bytes from fra24s07-in-f3.1e100.net (142.250.186.131): icmp_seq=9 ttl=128 time=25.7 ms
64 bytes from fra24s07-in-f3.1e100.net (142.250.186.131): icmp_seq=10 ttl=128 time=26.5 ms

--- www.google.dk ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9029ms
rtt min/avg/max/mdev = 25.446/27.278/34.620/2.556 ms
ase@ubuntu:~$
```

Figur 8 - 10 ping til www.google.dk

Opgave 5:

Anvend Wireshark til at måle tidsforsinkelsen. TCP three-way handshake består af [SYN] [SYN, ACK] og [ACK]. Tidforsinkelsen / responstiden fra [SYN] til [SYN, ACK] vises i Figur 9.



No.	Time	Source	Destination	Protocol	Length	Info
45	0.000	192.168.24.128	185.45.20.48	TCP	74	33864 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PER
48	0.016	185.45.20.48	192.168.24.128	TCP	60	80 → 33864 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=14
49	0.000	192.168.24.128	185.45.20.48	TCP	54	33864 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0

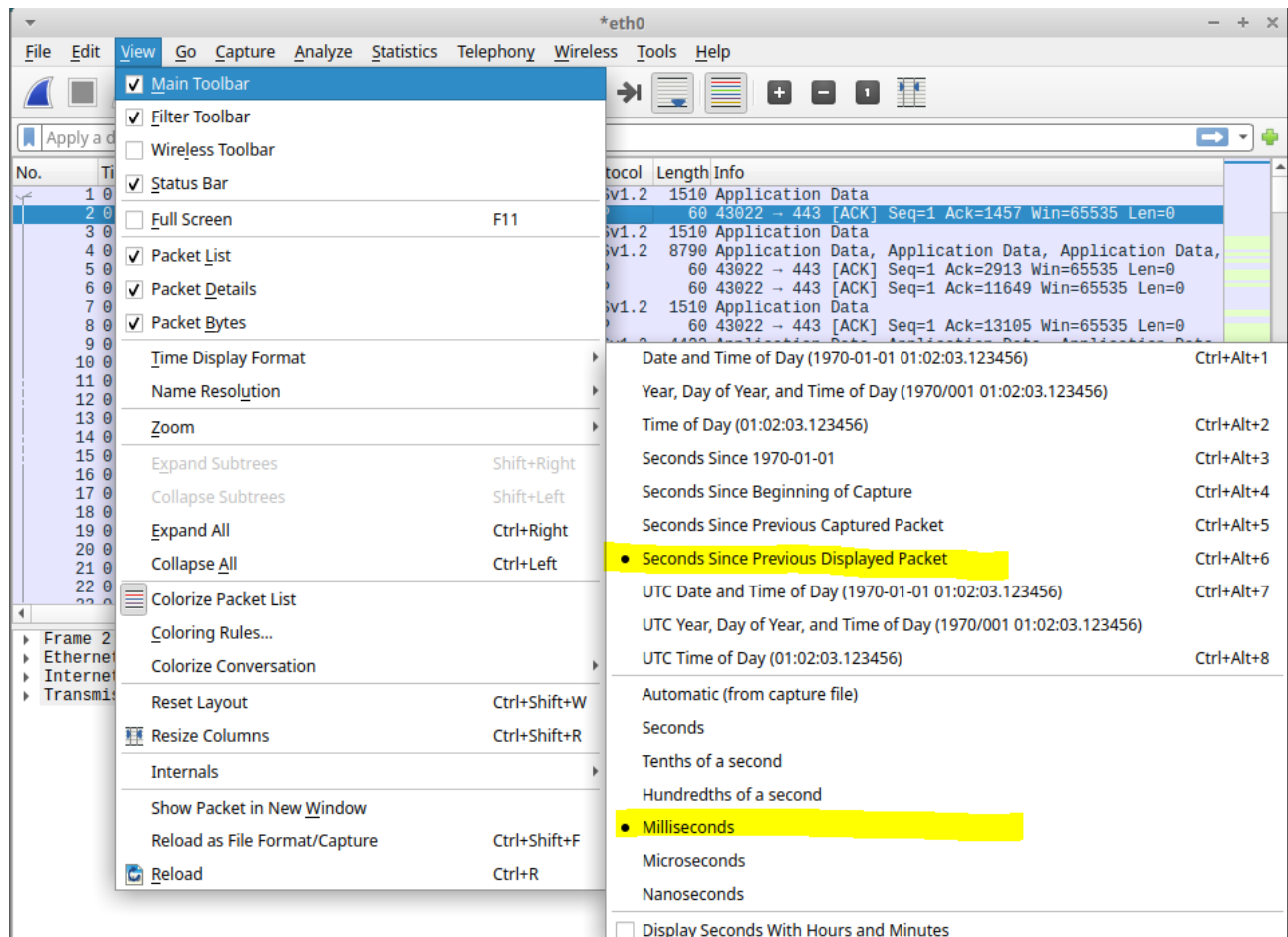
Figur 9 - TCP three-way handshake

Responstiden fra [SYN] til [SYN, ACK] her er:

0,016 milliseconds

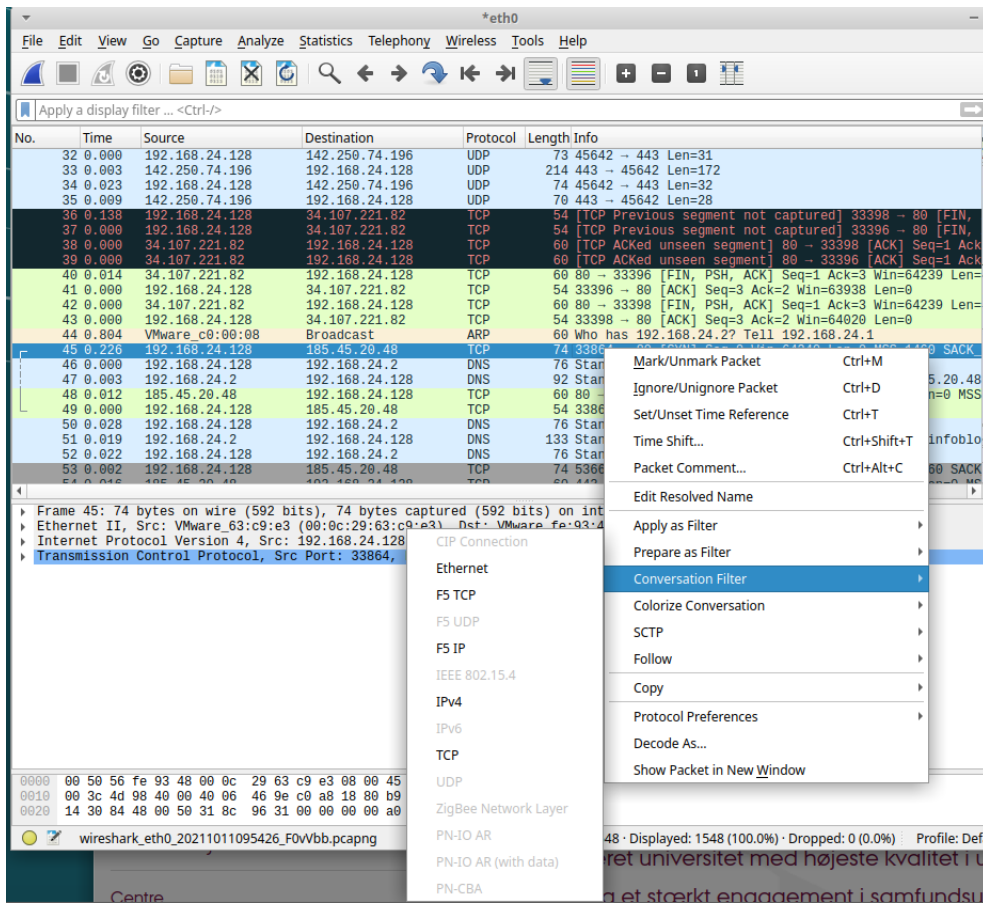
I denne opgave benyttes webbrowseren til at connect til www.au.dk

I Wireshark laves følgende indstillinger for nemmere at finde/aflæse responstiden. Først ændres Time Display Format til Milliseconds og måles som tiden siden sidst viste package. Dette er vist i Figur 10.

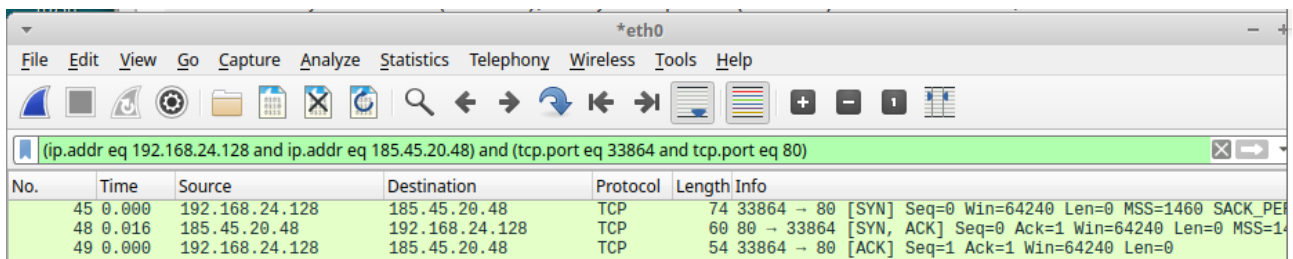


Figur 10 - Time display format

Samtidig benyttes et conversation filter: (TCP) for at isolere de oplysninger vi ønsker i Wireshark. Tilføjelse af filteret er vist i Figur 11 og resultatet er vist i Figur 12.



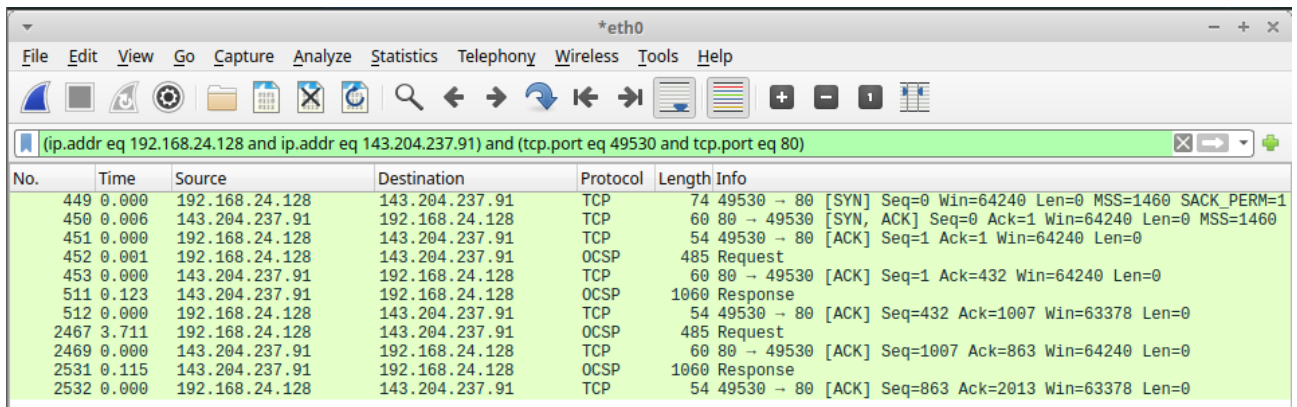
Figur 11 - Wireshark conversation filter



Figur 12 - TCP filter

Opgave 6:

Anvend Wireshark til at måle tidsforsinkelsen til en australsk web-side. Der benyttes samme opsætning som i opgave 5, men nu tilgås en australsk side og responstiden måles i Wireshark



No.	Time	Source	Destination	Protocol	Length	Info
449	0.000	192.168.24.128	143.204.237.91	TCP	74	49530 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
450	0.006	143.204.237.91	192.168.24.128	TCP	60	80 → 49530 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
451	0.000	192.168.24.128	143.204.237.91	TCP	54	49530 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
452	0.001	192.168.24.128	143.204.237.91	OCSP	485	Request
453	0.000	143.204.237.91	192.168.24.128	TCP	60	80 → 49530 [ACK] Seq=1 Ack=432 Win=64240 Len=0
511	0.123	143.204.237.91	192.168.24.128	OCSP	1060	Response
512	0.000	192.168.24.128	143.204.237.91	TCP	54	49530 → 80 [ACK] Seq=432 Ack=1007 Win=63378 Len=0
2467	3.711	192.168.24.128	143.204.237.91	OCSP	485	Request
2469	0.000	143.204.237.91	192.168.24.128	TCP	60	80 → 49530 [ACK] Seq=1007 Ack=863 Win=64240 Len=0
2531	0.115	143.204.237.91	192.168.24.128	OCSP	1060	Response
2532	0.000	192.168.24.128	143.204.237.91	TCP	54	49530 → 80 [ACK] Seq=863 Ack=2013 Win=63378 Len=0

Figur 13 - Responsetid til australsk web-side

Responstiden fra [SYN] til [SYN, ACK] her er:

0,006 milliseconds

Í denne opgave benyttes webbrowseren til at connect til www.9news.com.au

Opgave 7:

Er der en forskel i Tidforsinkelsen / responstiden i opgave 5 og 6?

Resultaterne for de to hjemmesider i de to opgaver har vist at tidforsinkelsen i opgave 6, siden fra Australien, er mindre end i opgave 5, hvor Au's hjemmeside tilgås.

Øvelse 4 - HTTP Client/Server

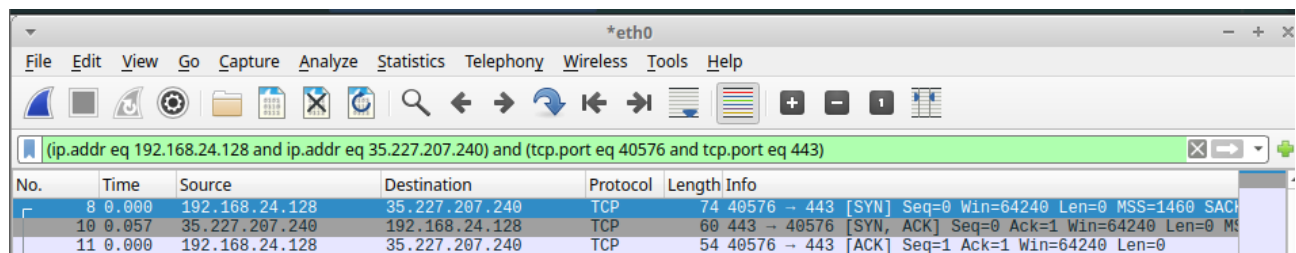
Indledning:

Formålet med øvelsen er at undersøge http protokollens funktionalitet ved brug af Wireshark .

Opgave 1:

Wireshark benyttes til at undersøge web-siden.

Tidsforsinkelse:



The image shows a Wireshark packet capture window titled '*eth0'. The filter bar contains the expression '(ip.addr eq 192.168.24.128 and ip.addr eq 35.227.207.240) and (tcp.port eq 40576 and tcp.port eq 443)'. The packet list shows three packets:

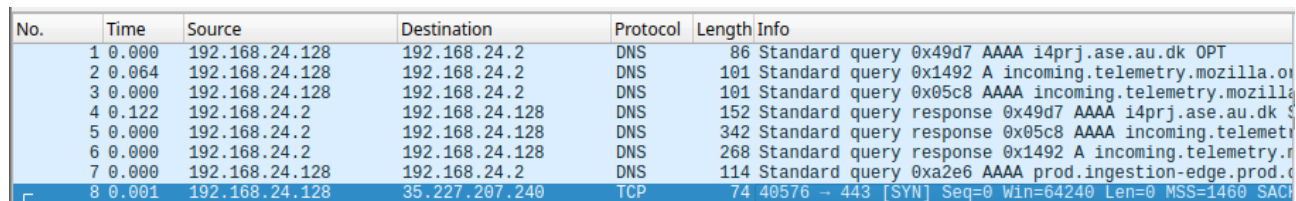
No.	Time	Source	Destination	Protocol	Length	Info
8	0.000	192.168.24.128	35.227.207.240	TCP	74	40576 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK
10	0.057	35.227.207.240	192.168.24.128	TCP	60	443 → 40576 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MS
11	0.000	192.168.24.128	35.227.207.240	TCP	54	40576 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0

Figur 14 - Responsetime

Responstiden fra [SYN] til [SYN, ACK] her er:

0,057 milliseconds

DNS opslag og DNS response:



The image shows a Wireshark packet capture window with a list of 8 packets:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000	192.168.24.128	192.168.24.2	DNS	86	Standard query 0x49d7 AAAA i4prj.ase.au.dk OPT
2	0.064	192.168.24.128	192.168.24.2	DNS	101	Standard query 0x1492 A incoming.telemetry.mozilla.or
3	0.000	192.168.24.128	192.168.24.2	DNS	101	Standard query 0x05c8 AAAA incoming.telemetry.mozilla
4	0.122	192.168.24.2	192.168.24.128	DNS	152	Standard query response 0x49d7 AAAA i4prj.ase.au.dk \$
5	0.000	192.168.24.2	192.168.24.128	DNS	342	Standard query response 0x05c8 AAAA incoming.telemetry
6	0.000	192.168.24.2	192.168.24.128	DNS	268	Standard query response 0x1492 A incoming.telemetry.r
7	0.000	192.168.24.128	192.168.24.2	DNS	114	Standard query 0xa2e6 AAAA prod.ingestion-edge.prod.c
8	0.001	192.168.24.128	35.227.207.240	TCP	74	40576 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK

Figur 15 - DNS opslag og response

Indhold af http request header:

HTTP GET vises i Figur 16

The image shows a Wireshark network traffic capture on interface eth0. The packet list at the top displays various network protocols including DNS, TCP, and HTTP. The selected packet is a GET request (No. 7) from 192.168.24.128 to 10.29.0.30. The packet details pane below shows the structure of the HTTP request, including the request line, headers, and body.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000	192.168.24.128	192.168.24.2	DNS	86	Standard query 0x46eb AAAA i4prj.ase.au.dk OPT
2	0.003	192.168.24.128	10.29.0.30	TCP	74	50648 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK
3	0.251	192.168.24.128	10.29.0.30	TCP	74	50650 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK
4	0.117	10.29.0.30	192.168.24.128	TCP	60	80 → 50648 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MS
5	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
6	0.000	192.168.24.2	192.168.24.128	DNS	152	Standard query response 0x46eb AAAA i4prj.ase.au.dk S
7	0.000	192.168.24.128	10.29.0.30	HTTP	404	GET /I4IKN HTTP/1.1
8	0.000	10.29.0.30	192.168.24.128	TCP	60	80 → 50648 [ACK] Seq=1 Ack=351 Win=64240 Len=0
9	0.244	10.29.0.30	192.168.24.128	TCP	60	80 → 50650 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MS
10	0.000	192.168.24.128	10.29.0.30	TCP	54	50650 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
11	0.000	10.29.0.30	192.168.24.128	TCP	1404	80 → 50648 [PSH, ACK] Seq=1 Ack=351 Win=64240 Len=135
12	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=1351 Win=63450 Len=0
13	0.000	10.29.0.30	192.168.24.128	TCP	1404	80 → 50648 [PSH, ACK] Seq=1351 Ack=351 Win=64240 Len=
14	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=2701 Win=62100 Len=0
15	0.000	10.29.0.30	192.168.24.128	TCP	1514	80 → 50648 [ACK] Seq=2701 Ack=351 Win=64240 Len=1460
16	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=4161 Win=62100 Len=0
17	0.000	10.29.0.30	192.168.24.128	TCP	1294	80 → 50648 [PSH, ACK] Seq=4161 Ack=351 Win=64240 Len=
18	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=5401 Win=60860 Len=0
19	0.047	192.168.24.128	192.168.24.2	DNS	101	Standard query 0xc935 A incoming.telemetry.mozilla.or
20	0.000	192.168.24.128	192.168.24.2	DNS	101	Standard query 0xd7da AAAA incoming.telemetry.mozilla
21	0.202	10.29.0.30	192.168.24.128	TCP	2754	80 → 50648 [PSH, ACK] Seq=5401 Ack=351 Win=64240 Len=
22	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=8101 Win=62780 Len=0
23	0.000	10.29.0.30	192.168.24.128	TCP	1404	80 → 50648 [PSH, ACK] Seq=8101 Ack=351 Win=64240 Len=
24	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=9451 Win=61430 Len=0
25	0.000	10.29.0.30	192.168.24.128	TCP	1404	80 → 50648 [PSH, ACK] Seq=9451 Ack=351 Win=64240 Len=
26	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=10801 Win=63450 Len=0
27	0.000	10.29.0.30	192.168.24.128	TCP	1514	80 → 50648 [ACK] Seq=10801 Ack=351 Win=64240 Len=1460
28	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=12261 Win=62780 Len=0

Frame 7: 404 bytes on wire (3232 bits), 404 bytes captured (3232 bits) on interface eth0, id 0
Ethernet II, Src: VMware_63:c9:e3 (00:0c:29:63:c9:e3), Dst: VMware_fe:93:48 (00:50:56:fe:93:48)
Internet Protocol Version 4, Src: 192.168.24.128, Dst: 10.29.0.30
Transmission Control Protocol, Src Port: 50648, Dst Port: 80, Seq: 1, Ack: 1, Len: 350
Hypertext Transfer Protocol
GET /I4IKN HTTP/1.1
[Expert Info (Chat/Sequence): GET /I4IKN HTTP/1.1
Request Method: GET
Request URI: /I4IKN
Request Version: HTTP/1.1
Host: i4prj.ase.au.dk
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:93.0) Gecko/20100101 Firefox/93.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
[Full request URI: http://i4prj.ase.au.dk/I4IKN]
[HTTP request 1/1]
[Response in frame: 198]

Figur 16 - Http GET

Og HTTP response 200 OK vises Figur 17

*eth0						
No.	Time	Source	Destination	Protocol	Length	Info
190	0.000	10.29.0.30	192.168.24.128	TCP	2644	80 → 50648 [PSH, ACK] Seq=248633 Ack=351 Win=64240 Len=0
191	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=251223 Win=65535 Len=0
192	0.000	10.29.0.30	192.168.24.128	TCP	1404	80 → 50648 [PSH, ACK] Seq=251223 Ack=351 Win=64240 Len=0
193	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=252573 Win=65535 Len=0
194	0.000	10.29.0.30	192.168.24.128	TCP	1404	80 → 50648 [PSH, ACK] Seq=252573 Ack=351 Win=64240 Len=0
195	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=253923 Win=65535 Len=0
196	0.000	10.29.0.30	192.168.24.128	TCP	1404	80 → 50648 [PSH, ACK] Seq=253923 Ack=351 Win=64240 Len=0
197	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=255273 Win=65535 Len=0
198	0.000	10.29.0.30	192.168.24.128	HTTP	334	HTTP/1.1 200 OK (text/html)
199	0.000	192.168.24.128	10.29.0.30	TCP	54	50648 → 80 [ACK] Seq=351 Ack=255553 Win=65535 Len=0
200	0.023	192.168.24.128	35.227.207.240	TCP	74	40586 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK
201	0.032	35.227.207.240	192.168.24.128	TCP	60	443 → 40586 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS
202	0.000	192.168.24.128	35.227.207.240	TCP	54	40586 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
203	0.000	192.168.24.128	35.227.207.240	TLSv1.2	571	Client Hello
204	0.000	35.227.207.240	192.168.24.128	TCP	60	443 → 40586 [ACK] Seq=1 Ack=518 Win=64240 Len=0
205	0.040	35.227.207.240	192.168.24.128	TLSv1.2	210	Server Hello, Change Cipher Spec, Encrypted Handshake
206	0.000	192.168.24.128	35.227.207.240	TCP	54	40586 → 443 [ACK] Seq=518 Ack=157 Win=64084 Len=0
207	0.000	192.168.24.128	35.227.207.240	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
208	0.000	35.227.207.240	192.168.24.128	TCP	60	443 → 40586 [ACK] Seq=157 Ack=569 Win=64240 Len=0
209	0.001	192.168.24.128	35.227.207.240	TLSv1.2	231	Application Data
210	0.000	192.168.24.128	35.227.207.240	TLSv1.2	405	Application Data
211	0.000	35.227.207.240	192.168.24.128	TCP	60	443 → 40586 [ACK] Seq=157 Ack=746 Win=64240 Len=0
212	0.000	35.227.207.240	192.168.24.128	TCP	60	443 → 40586 [ACK] Seq=157 Ack=1097 Win=64240 Len=0
213	0.000	192.168.24.128	35.227.207.240	TLSv1.2	637	Application Data
214	0.000	35.227.207.240	192.168.24.128	TCP	60	443 → 40586 [ACK] Seq=157 Ack=1680 Win=64240 Len=0
215	0.022	192.168.24.128	192.168.24.2	DNS	78	Standard query 0x12be A asp.net OPT
216	0.000	192.168.24.128	192.168.24.2	DNS	78	Standard query 0x7786 AAAA asp.net OPT
217	0.007	35.227.207.240	192.168.24.128	TLSv1.2	123	Application Data

Hypertext Transfer Protocol	
HTTP/1.1 200 OK\r\n	
[Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]	
Response Version: HTTP/1.1	
Status Code: 200	
[Status Code Description: OK]	
Response Phrase: OK	
Cache-Control: private\r\n	
Content-Type: text/html; charset=utf-8\r\n	
Server: Microsoft-IIS/8.0\r\n	
X-AspNetMvc-Version: 5.2\r\n	
X-AspNet-Version: 4.0.30319\r\n	
X-Powered-By: ASP.NET\r\n	
Date: Mon, 11 Oct 2021 08:32:17 GMT\r\n	
Content-Length: 255303\r\n	
\r\n	
[HTTP response 1/1]	
[Time since request: 0.725088886 seconds]	
[Request in frame: 7]	
[Request URI: http://i4prj.ase.au.dk/I4IKN]	
File Data: 255303 bytes	

Figur 17 - Http response OK

Apache2 installeres og opsættes i den virtuelle maskine

Etabler en LAN-forbindelse mellem en web-server (H1) og en web-client (H2). Resultatet af dette er vist nedenfor i Figur 18.

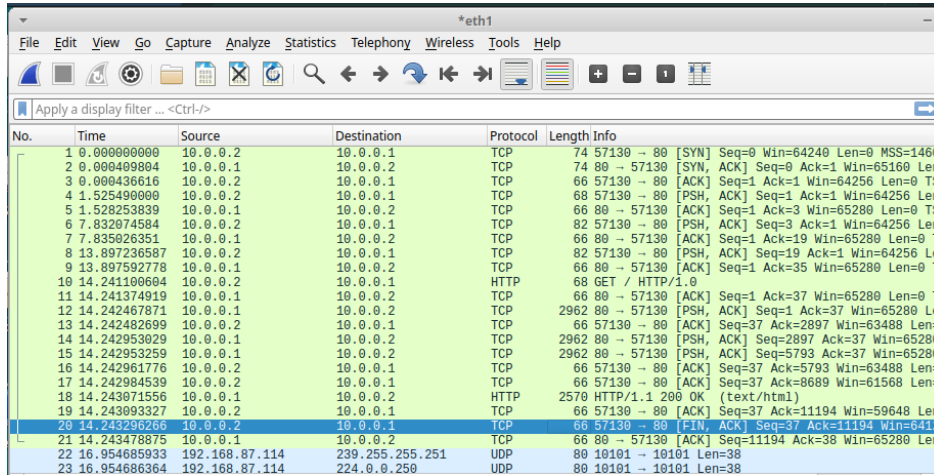
Figur 18 - LAN forbindelse

Figur 18 - LAN forbindelse

Opgave 4:

Test protokollerne: HTTP 1.0 og HTTP 1.1.vha. telnet med fokus på oprettelse/nedlukning af TCP-connection og på persistent/non-persistent HTTP-kommunikation vha. HTTP-protokollen

- a) Lukkes TCP-forbindelsen straks når HTTP 1.0 anvendes?

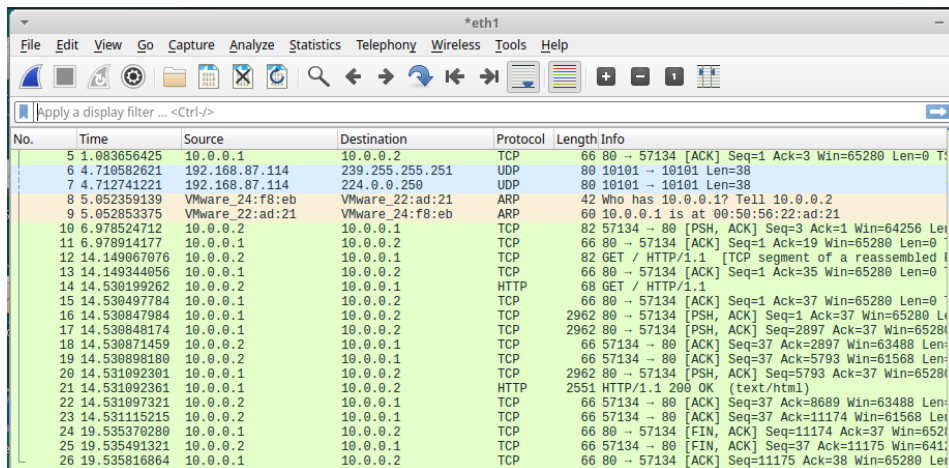


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.2	10.0.0.1	TCP	74	57130 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
2	0.000409804	10.0.0.1	10.0.0.2	TCP	74	80 → 57130 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
3	0.000436616	10.0.0.2	10.0.0.1	TCP	66	57130 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
4	1.525490000	10.0.0.2	10.0.0.1	TCP	68	57130 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=0
5	1.528253839	10.0.0.1	10.0.0.2	TCP	66	80 → 57130 [ACK] Seq=1 Ack=3 Win=65280 Len=0
6	7.832674584	10.0.0.2	10.0.0.1	TCP	82	57130 → 80 [PSH, ACK] Seq=3 Ack=1 Win=64256 Len=0
7	7.835026351	10.0.0.1	10.0.0.2	TCP	66	80 → 57130 [ACK] Seq=1 Ack=19 Win=65280 Len=0
8	13.897236587	10.0.0.2	10.0.0.1	TCP	82	57130 → 80 [PSH, ACK] Seq=19 Ack=1 Win=64256 Len=0
9	13.897592778	10.0.0.1	10.0.0.2	TCP	66	80 → 57130 [ACK] Seq=1 Ack=35 Win=65280 Len=0
10	14.241106064	10.0.0.2	10.0.0.1	HTTP	68	GET / HTTP/1.0
11	14.241374919	10.0.0.1	10.0.0.2	TCP	66	80 → 57130 [ACK] Seq=1 Ack=37 Win=65280 Len=0
12	14.242467871	10.0.0.1	10.0.0.2	TCP	2962	80 → 57130 [PSH, ACK] Seq=1 Ack=37 Win=65280 Len=0
13	14.242482699	10.0.0.2	10.0.0.1	TCP	66	57130 → 80 [ACK] Seq=37 Ack=2897 Win=63488 Len=0
14	14.242953029	10.0.0.1	10.0.0.2	TCP	2962	80 → 57130 [PSH, ACK] Seq=2897 Ack=37 Win=65280
15	14.242953259	10.0.0.1	10.0.0.2	TCP	2962	80 → 57130 [PSH, ACK] Seq=5793 Ack=37 Win=65280
16	14.242961776	10.0.0.2	10.0.0.1	TCP	66	57130 → 80 [ACK] Seq=37 Ack=5793 Win=63488 Len=0
17	14.242984539	10.0.0.2	10.0.0.1	TCP	66	57130 → 80 [ACK] Seq=37 Ack=8689 Win=61568 Len=0
18	14.243071556	10.0.0.1	10.0.0.2	HTTP	2570	HTTP/1.1 200 OK (text/html)
19	14.243093327	10.0.0.2	10.0.0.1	TCP	66	57130 → 80 [ACK] Seq=37 Ack=11194 Win=59648 Len=0
20	14.243292656	10.0.0.2	10.0.0.1	TCP	66	57130 → 80 [FIN, ACK] Seq=37 Ack=11194 Win=64112
21	14.243478875	10.0.0.1	10.0.0.2	TCP	66	80 → 57130 [ACK] Seq=11194 Ack=38 Win=65280 Len=0
22	16.954685933	192.168.87.114	239.255.255.251	UDP	80	10101 → 10101 Len=38
23	16.954686364	192.168.87.114	224.0.0.250	UDP	80	10101 → 10101 Len=38

Figur 19 - http/1.0 a

Forbindelsen lukkes straks og TCP protokollen er vist ovenfor i Figur 19, hvor der afsluttes med [FIN, ACK]

- b) Lukkes TCP-forbindelsen straks når HTTP 1.1 anvendes?



No.	Time	Source	Destination	Protocol	Length	Info
5	1.083656425	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=1 Ack=3 Win=65280 Len=0
6	4.710582621	192.168.87.114	239.255.255.251	UDP	80	10101 → 10101 Len=38
7	4.712741221	192.168.87.114	224.0.0.250	UDP	80	10101 → 10101 Len=38
8	5.052359139	VMware_24:f8:eb	VMware_22:ad:21	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
9	5.052853375	VMware_22:ad:21	VMware_24:f8:eb	ARP	60	10.0.0.1 is at 00:50:56:22:ad:21
10	6.978524712	10.0.0.2	10.0.0.1	TCP	82	57134 → 80 [PSH, ACK] Seq=3 Ack=1 Win=64256 Len=0
11	6.978914177	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=1 Ack=19 Win=65280 Len=0
12	14.149067076	10.0.0.2	10.0.0.1	TCP	82	GET / HTTP/1.1 [TCP segment of a reassembled
13	14.149344056	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=1 Ack=35 Win=65280 Len=0
14	14.530199262	10.0.0.2	10.0.0.1	HTTP	68	GET / HTTP/1.1
15	14.530497784	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=1 Ack=37 Win=65280 Len=0
16	14.530847994	10.0.0.1	10.0.0.2	TCP	2962	80 → 57134 [PSH, ACK] Seq=1 Ack=37 Win=65280 Len=0
17	14.530848174	10.0.0.1	10.0.0.2	TCP	2962	80 → 57134 [PSH, ACK] Seq=2897 Ack=37 Win=65280
18	14.530871459	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=37 Ack=2897 Win=63488 Len=0
19	14.530898180	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=37 Ack=5793 Win=61568 Len=0
20	14.531092301	10.0.0.1	10.0.0.2	TCP	2962	80 → 57134 [PSH, ACK] Seq=5793 Ack=37 Win=65280
21	14.531092361	10.0.0.1	10.0.0.2	HTTP	2551	HTTP/1.1 200 OK (text/html)
22	14.531097321	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=37 Ack=8689 Win=63488 Len=0
23	14.531115215	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=37 Ack=11174 Win=61568 Len=0
24	19.535370280	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [FIN, ACK] Seq=11174 Ack=37 Win=6521
25	19.535491321	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [FIN, ACK] Seq=37 Ack=11175 Win=64112
26	19.535816864	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=11175 Ack=38 Win=65280 Len=0

Figur 20 - http/1.1 b

Forbindelsen lukkes ikke staks, men efter kort tid, som vist ovenfor i Figur 20

- c) Hvis TCP-forbindelsen ikke lukkes umiddelbart, lukkes den så automatisk lidt senere? Hvor lang tid går der?

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.2	10.0.0.1	TCP	74	57134 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
2	0.000638843	10.0.0.1	10.0.0.2	TCP	74	80 → 57134 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
3	0.000674271	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSv
4	1.003031139	10.0.0.2	10.0.0.1	HTTP	68	Continuation
5	1.003056425	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=1 Ack=3 Win=65280 Len=0 TSv
6	6.978524712	10.0.0.1	10.0.0.2	TCP	82	57134 → 80 [PSH, ACK] Seq=1 Ack=19 Win=64256 Len=0
7	6.978914177	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=1 Ack=19 Win=65280 Len=0 TSv
8	14.149067076	10.0.0.1	10.0.0.2	TCP	82	57134 → 80 [PSH, ACK] Seq=19 Ack=1 Win=64256 Len=0
9	14.149344056	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=1 Ack=35 Win=65280 Len=0 TSv
10	14.530199262	10.0.0.2	10.0.0.1	HTTP	68	GET / HTTP/1.1
11	14.530497784	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=1 Ack=37 Win=65280 Len=0 TSv
12	14.530847984	10.0.0.1	10.0.0.2	TCP	2962	80 → 57134 [PSH, ACK] Seq=1 Ack=37 Win=65280 Len=0
13	14.530848174	10.0.0.1	10.0.0.2	TCP	2962	80 → 57134 [PSH, ACK] Seq=2897 Ack=37 Win=65280
14	14.530871459	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=37 Ack=2897 Win=63488 Len=0
15	14.530898180	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=37 Ack=5793 Win=61568 Len=0
16	14.531092301	10.0.0.1	10.0.0.2	TCP	2962	80 → 57134 [PSH, ACK] Seq=5793 Ack=37 Win=65280
17	21.000000000	10.0.0.1	10.0.0.2	HTTP	2551	HTTP/1.1 200 OK (text/html)
18	22.0000004960	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=37 Ack=8689 Win=63488 Len=0
19	23.0000022854	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=37 Ack=11174 Win=61568 Len=0
20	24.5004277919	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [FIN, ACK] Seq=11174 Ack=37 Win=65280
21	25.004398960	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [FIN, ACK] Seq=37 Ack=11175 Win=64128
22	26.5004724503	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=11175 Ack=38 Win=65280 Len=0

Figur 21 - http/1.1 c

Som vist i Figur 21, hvor der html response er sat som reference ifht. tiden, går der 5,0047 milliseconds.

- d) Hvad er fordelene ved at nedlukningen af TCP-forbindelsen udskydes?

Vis der benytte en pipeline, kan der sendes en ny GET inden nedlukningen, dette kræver en Persistent connection og kræver at der bruges http/1,1

- e) Er det web-server eller web-client, der lukker TCP-forbindelsen?

Som vist i tidligere resultater fra Wireshark er det Serveren på den virtuelle maskine H1 som afbryder forbindelsen. Dette ses ved at Source: 10.0.0.1 sender en request til at nedlukke forbindelsen og 10.0.0.2 bekræfter. Et uddrag fra tidligere er vist nedenfor som eksempel på dette i Figur 22.

24	5.004277919	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [FIN, ACK] Seq=11174 Ack=37 Win=65280
25	5.004398960	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [FIN, ACK] Seq=37 Ack=11175 Win=64128
26	5.004724503	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=11175 Ack=38 Win=65280 Len=0

Figur 22 - Server-client

f) Fremgår version af Apache serveren af http respons, og I givet fald hvor?

*eth1						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
(ip.addr eq 10.0.0.1 and ip.addr eq 10.0.0.2) and (tcp.port eq 80 and tcp.port eq 57134)						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.2	10.0.0.1	TCP	74	57134 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
2	0.000638843	10.0.0.1	10.0.0.2	TCP	74	80 → 57134 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0
3	0.000674271	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSv
4	1.083031139	10.0.0.2	10.0.0.1	HTTP	68	Continuation
5	1.083656425	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=1 Ack=3 Win=65280 Len=0 TSv
10	6.978524712	10.0.0.2	10.0.0.1	TCP	82	57134 → 80 [PSH, ACK] Seq=3 Ack=1 Win=64256 Len=0
11	6.978914177	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=1 Ack=19 Win=65280 Len=0 TSv
12	14.149067076	10.0.0.2	10.0.0.1	TCP	82	57134 → 80 [PSH, ACK] Seq=19 Ack=1 Win=64256 Len=0
13	14.149344056	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=1 Ack=35 Win=65280 Len=0 TSv
14	14.530199262	10.0.0.2	10.0.0.1	HTTP	68	GET / HTTP/1.1
15	14.530497784	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=1 Ack=37 Win=65280 Len=0 TSv
16	14.530847984	10.0.0.1	10.0.0.2	TCP	2962	80 → 57134 [PSH, ACK] Seq=1 Ack=37 Win=65280 Len=0
17	14.530848174	10.0.0.1	10.0.0.2	TCP	2962	80 → 57134 [PSH, ACK] Seq=2897 Ack=37 Win=65280 Len=0
18	14.530871459	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=37 Ack=2897 Win=63488 Len=0
19	14.530898180	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=37 Ack=5793 Win=61568 Len=0
20	14.531092301	10.0.0.1	10.0.0.2	TCP	2962	80 → 57134 [PSH, ACK] Seq=5793 Ack=37 Win=65280 Len=0
21	*REF*	10.0.0.1	10.0.0.2	HTTP	2551	HTTP/1.1 200 OK (text/html)
22	0.000004960	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=37 Ack=8689 Win=63488 Len=0
23	0.000022854	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [ACK] Seq=37 Ack=11174 Win=61568 Len=0
24	5.004277919	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [FIN, ACK] Seq=11174 Ack=37 Win=65280 Len=0
25	5.004398960	10.0.0.2	10.0.0.1	TCP	66	57134 → 80 [FIN, ACK] Seq=37 Ack=11175 Win=64128 Len=0
26	5.004724503	10.0.0.1	10.0.0.2	TCP	66	80 → 57134 [ACK] Seq=11175 Ack=38 Win=65280 Len=0

Frame 21: 2551 bytes on wire (20408 bits), 2551 bytes captured (20408 bits) on interface eth1, id 0
Ethernet II, Src: VMware_22:ad:21 (00:50:56:22:ad:21), Dst: VMware_24:f8:eb (00:50:56:24:f8:eb)
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
Transmission Control Protocol, Src Port: 80, Dst Port: 57134, Seq: 8689, Ack: 37, Len: 2485
[4 Reassembled TCP Segments (11173 bytes): #16(2896), #17(2896), #20(2896), #21(2485)]
Hypertext Transfer Protocol
HTTP/1.1 200 OK\r\n
Date: Mon, 11 Oct 2021 09:02:09 GMT\r\n
Server: Apache/2.4.41 (Ubuntu)\r\n
Last-Modified: Mon, 13 Sep 2021 08:13:08 GMT\r\n
ETag: "2aa6-5cbdc0904aa98"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 10918\r\n

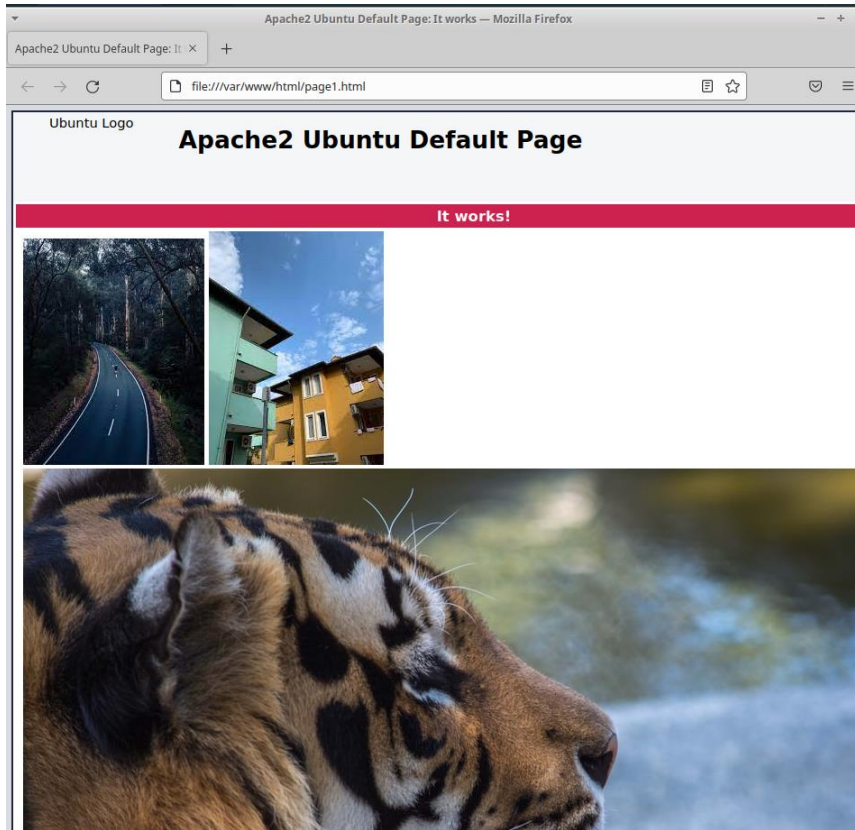
Figur 23 - Server version

Apache server version vises i http response, og som vist ovenfor i Figur 23, i Hypertext Transfer Protocol kan der aflæses:
server: Apache/2.4.41 (ubuntu)

Opgave 5:

Anvend Firefox web-browser som web-client i H2 sammen med apache-server i H1

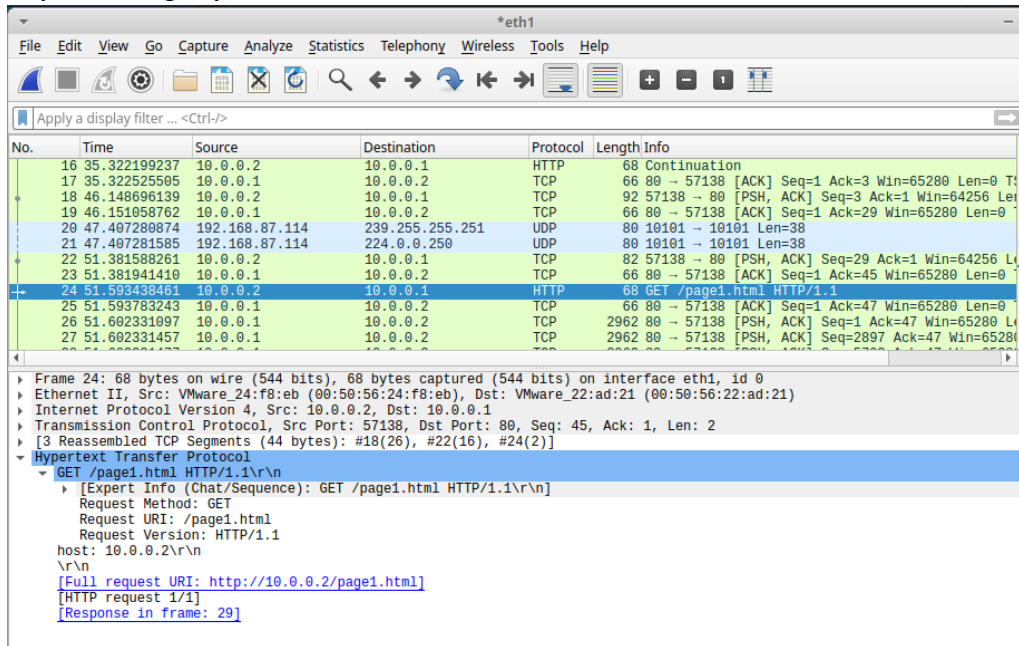
Der indsættes 3 .jpeg billeder i html filen, og resultatet fra webbrowseren vises i Figur 24



Figur 24 - 3 time image.jpeg

Der laves nu en analyse ved hjælp af Wireshark over relevante hændelser

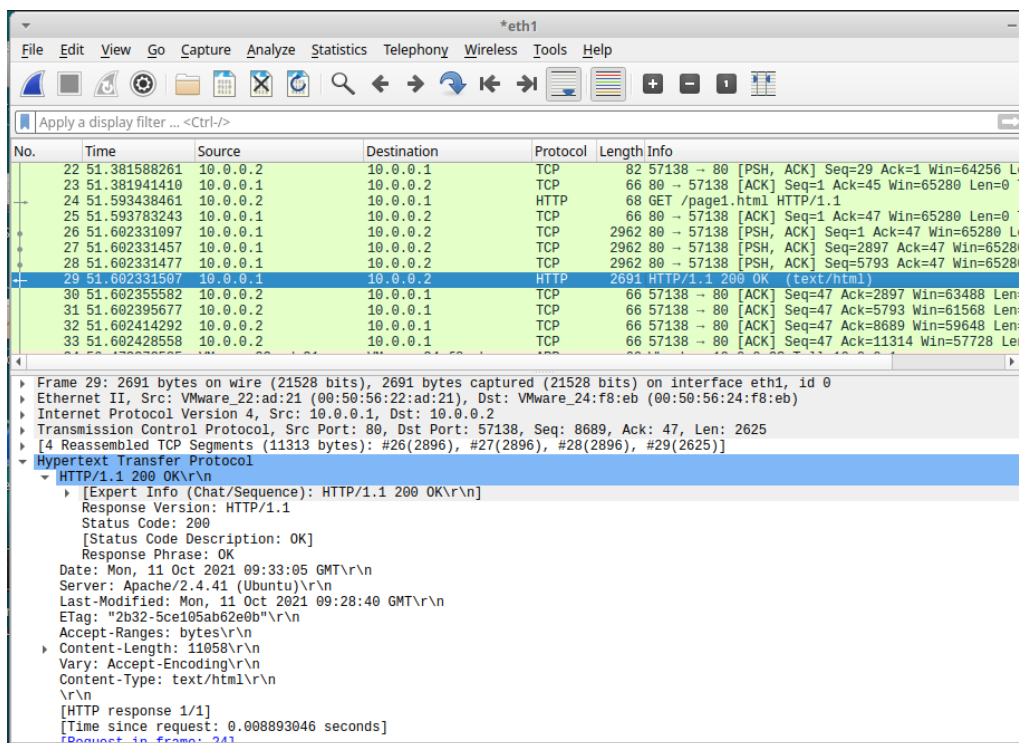
http-version og request header



Figur 25 - http version page1.html

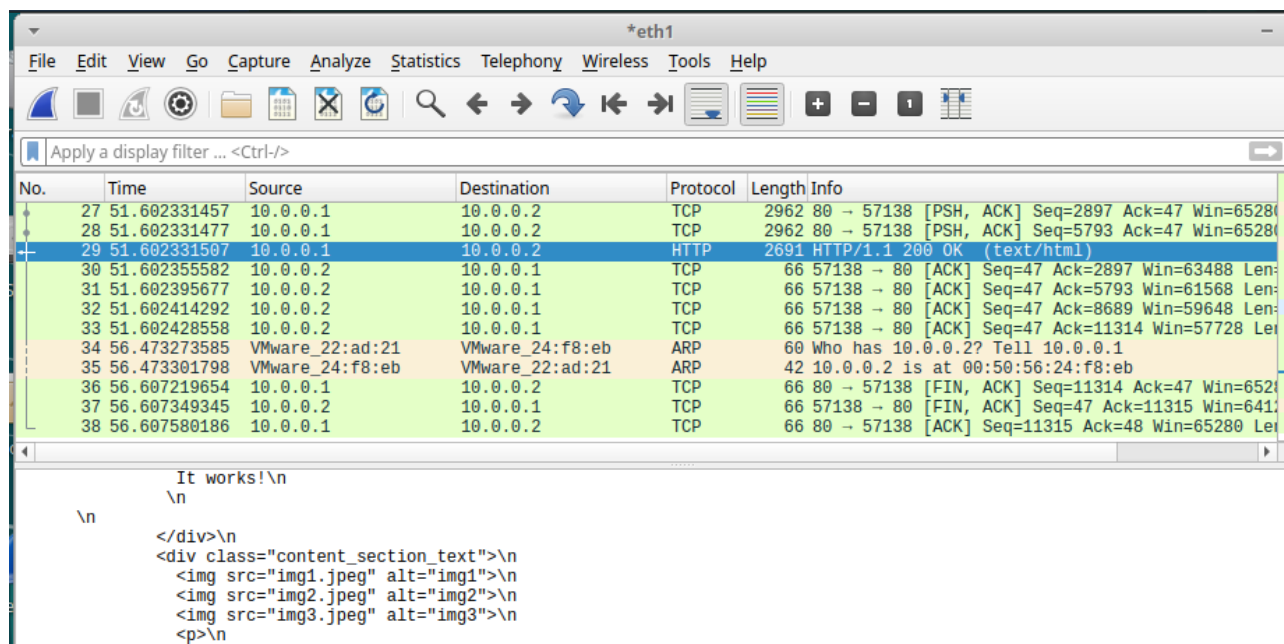
Version af hrml er 1.1 og der kaldet GET på siden /page1.html hvor de 3 jpeg billeder er sat ind.

Response header

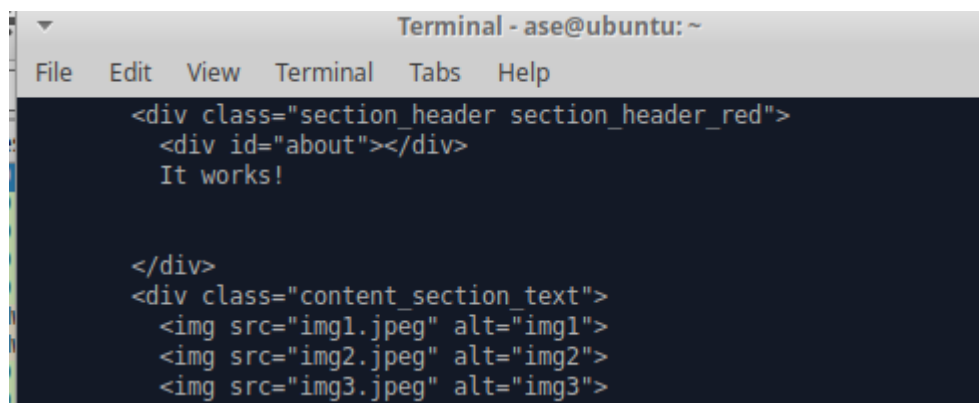


Figur 26 - http response page1.html

Html response indeholder version og serverstatus 200 OK udvides indholdet, vises html koden hvor de 3 billeder korrekt ligger. Dette er vist i Figur 27 fra Wireshark og i Figur 28 ses samme resultat fra terminalen.



Figur 27 - 3 times jpeg



Figur 28 - Terminal response

Connection

36	56.607219654	10.0.0.1	10.0.0.2	TCP	66	80 → 57138	[FIN, ACK] Seq=11314 Ack=47 Win=65280 Len=0
37	56.607349345	10.0.0.2	10.0.0.1	TCP	66	57138 → 80	[FIN, ACK] Seq=47 Ack=11315 Win=64112 Len=0
38	56.607580186	10.0.0.1	10.0.0.2	TCP	66	80 → 57138	[ACK] Seq=11315 Ack=48 Win=65280 Len=0

Da version af html/1.1 er connection persistent, men lukkes ned kort tid efter response. Dette observeres som tidligere i Wireshark, hvor der sendes en forespørgsel fra serveren om nedlukning.

Øvelse 5 - DNS client

Indledning:

Formålet med øvelsen er at undersøge/anvende DNS-protokollen.

Opgave 1:

Kommandoerne nslookup og host anvendes på forskellige web-sites og resultaterne vises nedenfor.

Først undersøges options for host kommandoen i Figur 29.



```
Terminal - ase@ubuntu: ~
File Edit View Terminal Tabs Help
HOST(1) BIND9 HOST(1)

NAME
  host - DNS lookup utility

SYNOPSIS
  host [-aACdlrrsTUwv] [-c class] [-N ndots] [-R number] [-t type] [-W wait] [-m flag]
    [[-4] | [-6]] [-v] [-V] {name} [server]

DESCRIPTION
  host is a simple utility for performing DNS lookups. It is normally used to convert
  names to IP addresses and vice versa. When no arguments or options are given, host
  prints a short summary of its command line arguments and options.

  name is the domain name that is to be looked up. It can also be a dotted-decimal IPv4
  address or a colon-delimited IPv6 address, in which case host will by default perform
  a reverse lookup for that address. server is an optional argument which is either
  the name or IP address of the name server that host should query instead of the
  server or servers listed in /etc/resolv.conf.

OPTIONS
  -4
    Use IPv4 only for query transport. See also the -6 option.

  -6
    Use IPv6 only for query transport. See also the -4 option.

  -a
    "All". The -a option is normally equivalent to -v -t ANY. It also affects the
    behaviour of the -l list zone option.

  -A
    "Almost all". The -A option is equivalent to -a except RRSIG, NSEC, and NSEC3
    records are omitted from the output.

  -c class
    Query class: This can be used to lookup HS (Hesiod) or CH (Chaosnet) class
    resource records. The default class is IN (Internet).

Manual page host(1) line 1 (press h for help or q to quit)

ase@ubuntu:~$ host
Usage: host [-aCdilrTvVw] [-c class] [-N ndots] [-t type] [-W time]
    [-R number] [-m flag] hostname [server]
  -a is equivalent to -v -t ANY
  -A is like -a but omits RRSIG, NSEC, NSEC3
  -c specifies query class for non-IN data
  -C compares SOA records on authoritative nameservers
  -d is equivalent to -v
  -l lists all hosts in a domain, using AXFR
  -m set memory debugging flag (trace|record|usage)
  -N changes the number of dots allowed before root lookup is done
  -r disables recursive processing
  -R specifies number of retries for UDP packets
  -s a SERVFAIL response should stop query
  -t specifies the query type
  -T enables TCP/IP mode
  -U enables UDP mode
  -v enables verbose output
```

Figur 29 - host options

Herefter testes host kommandoen og resultater ses i Figur 30

```
ase@ubuntu:~$ man host
ase@ubuntu:~$ host www.google.dk
www.google.dk has address 142.250.185.99
www.google.dk has IPv6 address 2a00:1450:4001:82a::2003
ase@ubuntu:~$ host -4 www.google.dk
www.google.dk has address 142.250.184.195
www.google.dk has IPv6 address 2a00:1450:4001:830::2003
ase@ubuntu:~$ host -a www.google.dk
Trying "www.google.dk"
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 3074
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.google.dk.                IN      ANY

;; ANSWER SECTION:
www.google.dk.                5       IN      AAAA    2a00:1450:4001:830::2003
www.google.dk.                5       IN      A       142.250.184.195

Received 75 bytes from 127.0.0.53#53 in 12 ms
ase@ubuntu:~$ host -A www.google.dk
Trying "www.google.dk"
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 25300
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

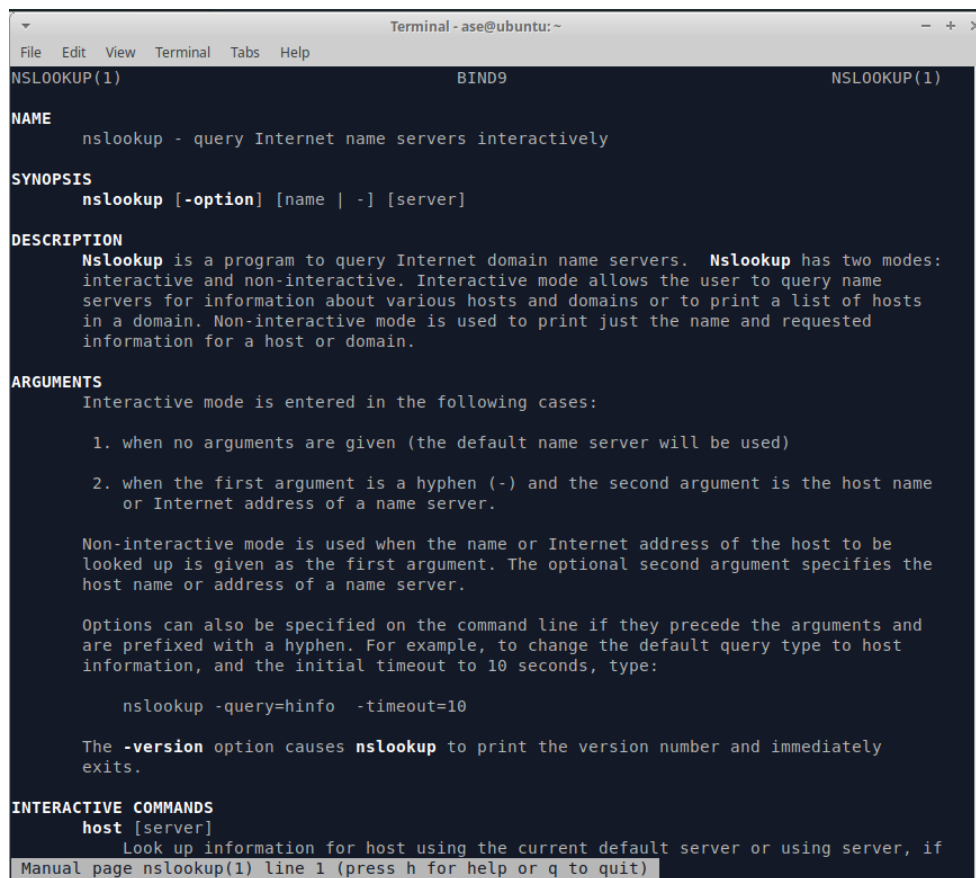
;; QUESTION SECTION:
;www.google.dk.                IN      ANY

;; ANSWER SECTION:
www.google.dk.                5       IN      AAAA    2a00:1450:4001:830::2003
www.google.dk.                5       IN      A       142.250.184.195

Received 75 bytes from 127.0.0.53#53 in 24 ms
```

Figur 30 - host kommando

Det samme gøres nu for nslookup hvor options vises i Figur 31

A terminal window titled "Terminal - ase@ubuntu: ~" displays the manual page for nslookup(1). The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal content shows the manual page structure: NAME, SYNOPSIS, DESCRIPTION, ARGUMENTS, and INTERACTIVE COMMANDS. The DESCRIPTION section explains that nslookup is a program to query Internet domain name servers, with interactive and non-interactive modes. The ARGUMENTS section lists cases for interactive mode and provides an example command: nslookup -query=hinfo -timeout=10. The INTERACTIVE COMMANDS section shows the 'host' command and its description. The bottom of the terminal shows a prompt: "Manual page nslookup(1) line 1 (press h for help or q to quit)".

```
Terminal - ase@ubuntu: ~
File Edit View Terminal Tabs Help
NSLOOKUP(1) BIND9 NSLOOKUP(1)

NAME
    nslookup - query Internet name servers interactively

SYNOPSIS
    nslookup [-option] [name | -] [server]

DESCRIPTION
    Nslookup is a program to query Internet domain name servers. Nslookup has two modes:
    interactive and non-interactive. Interactive mode allows the user to query name
    servers for information about various hosts and domains or to print a list of hosts
    in a domain. Non-interactive mode is used to print just the name and requested
    information for a host or domain.

ARGUMENTS
    Interactive mode is entered in the following cases:

    1. when no arguments are given (the default name server will be used)

    2. when the first argument is a hyphen (-) and the second argument is the host name
       or Internet address of a name server.

    Non-interactive mode is used when the name or Internet address of the host to be
    looked up is given as the first argument. The optional second argument specifies the
    host name or address of a name server.

    Options can also be specified on the command line if they precede the arguments and
    are prefixed with a hyphen. For example, to change the default query type to host
    information, and the initial timeout to 10 seconds, type:

        nslookup -query=hinfo -timeout=10

    The -version option causes nslookup to print the version number and immediately
    exits.

INTERACTIVE COMMANDS
    host [server]
        Look up information for host using the current default server or using server, if

Manual page nslookup(1) line 1 (press h for help or q to quit)
```

Figur 31 - nslookup options

Og resultaterne vises nedenfor i Figur 32

```
ase@ubuntu:~$ nslookup www.au.dk
Server:      127.0.0.53
Address:     127.0.0.53#53

Non-authoritative answer:
Name:   www.au.dk
Address: 185.45.20.48

ase@ubuntu:~$
```

```
ase@ubuntu:~$ nslookup www.google.dk
Server:      127.0.0.53
Address:     127.0.0.53#53

Non-authoritative answer:
Name:   www.google.dk
Address: 142.250.185.67
Name:   www.google.dk
Address: 2a00:1450:4001:80e::2003

ase@ubuntu:~$
```

```
ase@ubuntu:~$ man nslookup
ase@ubuntu:~$ nslookup
> www.dmi.dk
Server:      127.0.0.53
Address:     127.0.0.53#53

Non-authoritative answer:
www.dmi.dk   canonical name = niavarnish.dmi.dk.
Name:   niavarnish.dmi.dk
Address: 130.226.71.229
Name:   niavarnish.dmi.dk
Address: 5.56.149.254
Name:   niavarnish.dmi.dk
Address: 5.56.149.237
Name:   niavarnish.dmi.dk
Address: 130.226.71.226
>
```

Figur 32 - command nslookup

Konklusion:

Host og nslookup er undesøgt at benyttet på forskellige sites. DNS informationerne er varierende fra site til site, hvor forskellige adresser og informationer er tilgængeligt. I opgaven er manualerne for kommandoerne undersøgt og forskellige options er forsøgt benyttet til at få yderlige DNS information fra de forskellige sites.

Øvelse 6 - Application Layer, Transport Layer, TCP Socket Programming

Indledning:

Formålet med øvelsen er at udvikle en server og client der vha. TCP skal sende en sting med "filnavn" og få returneret den ønskede fil fra serveren i chunks af 1000 bytes.

Opgave 1:

Der udvikles en server som kører på H1. Serveren skal modtage en tekststreng med et filnavn. Hvis filen findes skal filens størrelse sendes til client hvorefter selve filen sendes i chunks af 100 bytes af gangen. Serveren læser 1000 bytes fra filen, sender disse til clien, modtager og gemmer disse chunks indtil hele filen er sendt.

```
import sys
import os
from socket import socket, AF_INET, SOCK_STREAM
from lib import Lib
```

```
HOST = '0.0.0.0'
PORT = 9000
BUFSIZE = 1000
```

```
def main(argv):

    # Create TCP welcoming socket
    serverSocket = socket(AF_INET, SOCK_STREAM)
    serverSocket.bind((HOST, PORT))
    serverSocket.listen(1) # Server begins listening for incoming TCP requests

    while 1:
        print("The server is ready to accept file request")
        connectionSocket, addr = serverSocket.accept()
        print("Socket accept", addr)

        data = connectionSocket.recv(BUFSIZE).decode('utf-8') #File request

        print(data)
        sendFile(data, os.path.getsize(data), connectionSocket)

        connectionSocket.close()
```

```
def sendFile(fileName, fileSize, conn):
    # TO DO Your Code

    myfile = open(fileName, "rb")
    print (fileSize, "bytes")
    # https://thepythonguru.com/python-how-to-read-and-write-files/
    # https://www.youtube.com/watch?v=Kg-sxVmCt5Q
    chunk = myfile.read(BUFSIZE) #Read 1000bytes
    n = 1 #counter for print status
    while chunk:
        conn.send(chunk) # Send 1000 bytes
        chunk = myfile.read(BUFSIZE) # Read next
        print (f'Send {n}. chunk') #Print status
        n = n + 1 # Increment counter

    myfile.close()
    conn.close()

if __name__ == "__main__":
    main(sys.argv[1:])
```

Figur 33 - TCP Serverside

På serversiden oprettes en Socket, som begynder at "lytte" efter TCP requests.

Serveren accepterer request og læser filen/tekststreng til variabelen "data".

Hvis tekststrengen er en fil der findes udskrives filstørrelsen

Filen sendes nu til client ved sendFile(), som læser første chunk af 1000 bytes og sender disse indtil hele filen er sendt modtaget og gemt hos client.

Opgave 2:

Der udvikles in client på H2, som sender en tekststreng til server med et filnavn. Herefter skal client modtage filen i chunks af 1000 bytes som løbende gemmes.

```
import sys
from socket import socket, AF_INET, SOCK_STREAM
from lib import Lib
import os.path
import datetime

PORT = 9000
serverName = "10.0.0.1"
BUFSIZE = 1000

def main(argv):
    # TO DO Your Code
    clientSocket = socket(AF_INET, SOCK_STREAM)
    print("The client is connecting...")
    clientSocket.connect((serverName, PORT))
    print("Client connected")

    messageToServer = input("Input filepath:") #Get filepath

    receiveFile(messageToServer, clientSocket)
```

```
def receiveFile(fileName, conn):

    conn.send(fileName.encode())#'utf-8'

    file = open('my_image.jpeg', "wb") #open new image file with
    image_chunk = conn.recv(BUFSIZE) #recieve first 1000 bytes
    n = 1 #counter for status print
    while image_chunk:
        file.write(image_chunk) #Write 1000 bytes to new image
        print (f'Recieved {n}. chunk')# Print status
        image_chunk = conn.recv(BUFSIZE) # Recieve next 1000 bytes
        n = n +1 #Increment counter

    file.close()
    conn.close()

if __name__ == "__main__":
    main(sys.argv[1:])
```

Figur 34 - TCP Client

På client siden oprettes en socket og der forsøges at connect til serveren med servername("10.0.0.1") og port 9000.

Herefter benyttes receiveFile() til at modtage fra serveren.

Der oprettes en ny fil svarende til typen der ønskes modtaget, som data løbende bliver gemt i. I denne øvelse oprettes et jpeg, ved navnet my_image.jpeg.

Så længe der kommer data chunks fra serveren gemmes/skrives de til my_image

Opgave 3:

Der laves control/test af implementeringen og at det nye billede er identisk til det originale.

Connecting client to server:

```
ase@ubuntu:~/NGK/Exercise_6_py$ ./file_server.py
The server is ready to accept file request
Socket accept ('10.0.0.2', 43942)

ase@ubuntu:~/NGK/Exercise_6_py$ ./file_client.py
The client is connecting...
Client connected
Input filepath:
```

Figur 35 - TCP Server - client

Input valid filepath

```
ase@ubuntu:~/NGK/Exercise_6_py$ ./file_server.py
The server is ready to accept file request
Socket accept ('10.0.0.2', 43980)
/home/ase/Downloads/index.jpeg
5484 bytes
Send 1. chunk
Send 2. chunk
Send 3. chunk
Send 4. chunk
Send 5. chunk
Send 6. chunk
The server is ready to accept file request

ConnectionRefusedError: [Errno 111] Connection refused
ase@ubuntu:~/NGK/Exercise_6_py$ ./file_client.py
The client is connecting...
Client connected
Input filepath:/home/ase/Downloads/index.jpeg
Recieved 1. chunk
Recieved 2. chunk
Recieved 3. chunk
Recieved 4. chunk
Recieved 5. chunk
Recieved 6. chunk
ase@ubuntu:~/NGK/Exercise_6_py$
```

Figur 36 - TCP send image

Størrelsen på billedet er 5,484 bytes, derfor sendes der 6 pakker

Efterfølgende testes det sendte og modtagende billeder vha. kommandoen diff, og resultatet er at de 2 billeder er identiske og data derfor ikke gået tabt.

```
ase@ubuntu:~/NGK/Exercise_6_py$ diff -s index.jpeg my_image.jpeg
Files index.jpeg and my_image.jpeg are identical
ase@ubuntu:~/NGK/Exercise_6_py$
```

Figur 37 - TCP compare diff

Opgave 3:

Der er optaget en video, som vedlægges journalen hvor TCP server og client vises og at filen sendes korrekt.

Øvelse 7 - UDP/IP_Socket_Programming

Indledning:

Formålet med øvelsen er at udvikle en server og client der vha. UDP skal kommunikere mellem 2 virtuelle maskiner H1 og H2.

Opgave 1:

Der udvikles en UDP server, som skal modtage en request fra client om enten uptime eller loadavg for serveren.

```
HOST = ""
PORT = 9000

def main(argv):

    # Create UDP welcoming socket
    UDPServerSocket = socket(AF_INET, SOCK_DGRAM) #Datagram socket
    UDPServerSocket.bind((HOST,PORT)) #Bind address and IP

    print ("Server is running")

    while 1:
        bytesAddressPair = UDPServerSocket.recvfrom(2048)
        message = bytesAddressPair[0] #Extract message from UDP packet
        clientAddress = bytesAddressPair[1] #Extract client address from UDP packet
        print ("Message received from client", clientAddress, ":", message.decode('utf-8') )

        message = message.decode('utf-8').upper()
        print( message)

        answerRequest(UDPServerSocket,message, clientAddress)
```

```
def answerRequest(UDPServerSocket,request, clientAddress):

    if request == 'U':
        print( "Uptime was requested")
        myfile = open("/proc/uptime", "r")
        chunk = myfile.read(2048)
        print(chunk)

        UDPServerSocket.sendto(chunk.encode(), clientAddress)
        myfile.close()

    if request == 'L':
        print("Loadavg was requested")
        myfile = open("/proc/loadavg", "r")
        chunk = myfile.read(2048)
        print(chunk)

        UDPServerSocket.sendto(chunk.encode(), clientAddress)
        myfile.close()
```

På serversiden oprettes en socket som afventer forespørgsel fra client. Afhængig af request åbnes den ønskede fil og indholdet sendes til client.

Opgave 2:

Der udvikles en UDP client, som skal sende en request til serveren om enten uptime eller loadavg. Svaret skal herefter udskrives.

```
PORT = 9000

def main(argv):

    serverName = "10.0.0.1"

    print ("Client is starting...")
    clientSocket = socket(AF_INET, SOCK_DGRAM)

    x = 0
    while x < 1:
        messagetoserver = input('Input command: ')

        if messagetoserver == 'u':
            x = x + 1
        elif messagetoserver == 'U':
            x = x + 1
        elif messagetoserver == 'l':
            x = x + 1
        elif messagetoserver == 'L':
            x = x + 1
        else:
            print ("No valid input")

    clientSocket.sendto(messagetoserver.encode(), (serverName, PORT))
    receiveInfo(clientSocket)
    clientSocket.close()
```

```
def receiveInfo(clientSocket):
    file = open('data.txt',"wb")
    chunk = clientSocket.recvfrom(2048)
    messagefromchunk = chunk[0]
    print(messagefromchunk)
    file.close()
```

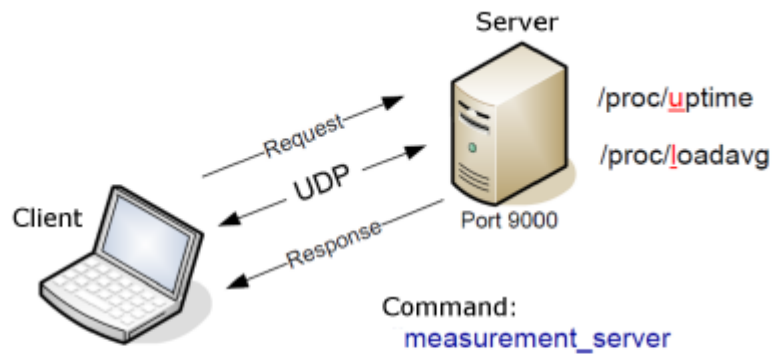
På serversiden sikres at userinput er enten u eller l, lowercase allowed. Dette sendes til serveren, og responses aflæses gemmes og udskrives.

Opgave 3:

Programmeringssproget benyttet er Python

Opgave 4:

Figur over opstillingen fra øvelsesbeskrivelsen Figur 38:



Command:
`get_measurement 10.0.0.1 U`
 or
`get_measurement 10.0.0.1 L`

Figur 38 - UDP server client opstilling

Opgave 5:

Der er optaget en video, som vedlægges journalen hvor UDP server og client vises og request og modtagelse er implementeret korrekt.

Billagsliste

- | | |
|-------------------------|-----------|
| 1) NGK-Exercise6_7_code | ZIP-mappe |
| 2) Exercise6 | MP4 |
| 3) Exercise7 | MP4 |