

IKN Journal 1: Øvelse 3, 6 og 7

Gruppe: 50

I4IKN

05-04-2018

Navn	Studienummer
Fatima Kodro	201609565
Søren Bech	201604784
Daniel Pat Hansen	201601915

Øvelse 3

1+2. Mål den tid der går fra en ping kommando startes i H1 til ping-respons fra H2 modtages i H1. Mål minimum-/maksimum-/gennemsnits-forsinkelsestider og standardafvigelsen for 10 på hinanden følgende ping-kommandoer.

På Figur 1 ses terminalen efter ping er blevet kaldt en gang. Dette er 1,23 ms. Efter der er blevet hentet 10 packets kan min/avg/max aflæses. Dette aflæses som:

- Min: 1,124
- Avg: 1,456
- Max: 2,146

```
root@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=1.23 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.234/1.234/1.234/0.000 ms
root@ubuntu:~# ping 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=1.97 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.21 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=1.24 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=1.18 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=1.16 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=1.19 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=2.08 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=2.14 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=1.22 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=1.12 ms
^C
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9128ms
rtt min/avg/max/mdev = 1.124/1.456/2.146/0.406 ms
root@ubuntu:~#
```

Figur 1 Pinger fra H1 til H2

3. Mål den tid der går fra kommandoen ping -c 1 www.tv2.dk startes i H1 til ping-respons fra web-serveren www.tv2.dk modtages i H1.

På Figur 2 ses terminalen efter der bliver pinget til www.tv2.dk. Det ses at det tager 11.9 ms for at sende og modtage packets.

```
root@ubuntu:~# ping -c 1 www.tv2.dk
PING web.pool.public.tv2net.dk (91.224.211.71) 56(84) bytes of data.
64 bytes from web-lb.pool.public.tv2net.dk (91.224.211.71): icmp_seq=1 ttl=128 time=11.9 ms

--- web.pool.public.tv2net.dk ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 11.985/11.985/11.985/0.000 ms
```

Figur 2: Pinger til www.tv2.dk

4. Mål minimum-/maksimum-/gennemsnits-forsinkelsestider og standardafvigelsen for 10 på hinanden følgende ping-kommandoer, udført som i punkt 3.

Figur 3 viser minimum/gennemsnit/maksimum efter der bliver pinget til www.tv2.dk. Det ses at der vises 11.813/12.034/12.253 ms

```
root@ubuntu:~# ping www.tv2.dk
PING web.pool.public.tv2net.dk (91.224.211.71) 56(84) bytes of data:
64 bytes from web-lb.pool.public.tv2net.dk (91.224.211.71): icmp_seq=1 ttl=128 time=12.2 ms
64 bytes from web-lb.pool.public.tv2net.dk (91.224.211.71): icmp_seq=2 ttl=128 time=11.8 ms
64 bytes from web-lb.pool.public.tv2net.dk (91.224.211.71): icmp_seq=3 ttl=128 time=11.8 ms
64 bytes from web-lb.pool.public.tv2net.dk (91.224.211.71): icmp_seq=4 ttl=128 time=12.2 ms
64 bytes from web-lb.pool.public.tv2net.dk (91.224.211.71): icmp_seq=5 ttl=128 time=12.0 ms
64 bytes from web-lb.pool.public.tv2net.dk (91.224.211.71): icmp_seq=6 ttl=128 time=12.0 ms
64 bytes from web-lb.pool.public.tv2net.dk (91.224.211.71): icmp_seq=7 ttl=128 time=12.0 ms
64 bytes from web-lb.pool.public.tv2net.dk (91.224.211.71): icmp_seq=8 ttl=128 time=12.1 ms
64 bytes from web-lb.pool.public.tv2net.dk (91.224.211.71): icmp_seq=9 ttl=128 time=11.9 ms
64 bytes from web-lb.pool.public.tv2net.dk (91.224.211.71): icmp_seq=10 ttl=128 time=12.0 ms
^C
--- web.pool.public.tv2net.dk ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9042ms
rtt min/avg/max/mdev = 11.813/12.034/12.253/0.198 ms
```

Figur 3: Gennemsnittet af 10 packets sendt til www.tv2.dk

5. Mål den tid der går fra en web-side i en dansk web-server ønskes modtaget i H1, til web-serveren responderer:

Der bruges www.kims.dk som hjemmeside

På Figur 4 ses et udsnit af wireshark, hvor den første DNS er en forespørgsel efter kims.dk.

Herefter modtages respons fra kims.dk. Det tager ca. 28ms før den første respons kommer fra kims.dk

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.74.128	151.236.219.8	TCP	74	53346 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=254004854 TSecr=0 WS=128
2	0.000252633	192.168.74.128	192.168.74.2	DNS	82	Standard query 0xc623 A www.kims.dk OPT
3	0.002949890	192.168.74.2	192.168.74.128	DNS	98	Standard query response 0xc623 A www.kims.dk A 151.236.219.8 OPT
4	0.003208978	192.168.74.128	192.168.74.2	DNS	82	Standard query 0xe728 AAAA www.kims.dk OPT
5	0.005505935	192.168.74.2	192.168.74.128	DNS	139	Standard query response 0xe728 AAAA www.kims.dk SOA ns1.rrprrproxy.net OPT
6	0.027523715	151.236.219.8	192.168.74.128	TCP	60	443 → 53346 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
7	0.027548649	192.168.74.128	151.236.219.8	TCP	54	53346 → 443 [ACK] Seq=1 Ack=1 Win=29200 Len=0
8	0.027930884	192.168.74.128	151.236.219.8	TLSv1.2	571	Client Hello
9	0.028121042	151.236.219.8	192.168.74.128	TCP	60	443 → 53346 [ACK] Seq=1 Ack=518 Win=64240 Len=0
10	0.074023959	151.236.219.8	192.168.74.128	TLSv1.2	208	Server Hello, Change Cipher Spec, Encrypted Handshake Message
11	0.074040289	192.168.74.128	151.236.219.8	TCP	54	53346 → 443 [ACK] Seq=518 Ack=155 Win=30016 Len=0
12	0.074651094	192.168.74.128	151.236.219.8	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
13	0.074872808	151.236.219.8	192.168.74.128	TCP	60	443 → 53346 [ACK] Seq=155 Ack=569 Win=64240 Len=0
14	0.075094994	192.168.74.128	151.236.219.8	TLSv1.2	231	Application Data
15	0.075197136	151.236.219.8	192.168.74.128	TCP	60	443 → 53346 [ACK] Seq=155 Ack=746 Win=64240 Len=0
16	0.075248210	192.168.74.128	151.236.219.8	TLSv1.2	343	Application Data

Figur 4: Wireshark billede af når man åbner browseren og går ind på www.kims.dk

6. Mål vha. samme målemetode den tid der går fra en australsk web-side (som er tilfældigt valgt) ønskes modtaget i H1 web-serveren, til web-serveren responderer.

Figur 5 viser Wireshark, når browseren åbnes og www.australia.gov.au forespørges.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.74.128	192.168.74.2	DNS	91	Standard query 0x8940 A www.australia.gov.au OPT
2	0.000105778	192.168.74.128	151.101.86.217	TLSv1.2	143	Application Data
3	0.000388133	192.168.74.128	192.168.74.2	DNS	91	Standard query 0x476f AAAA www.australia.gov.au OPT
4	0.000465807	151.101.86.217	192.168.74.128	TCP	60	443 → 53892 [ACK] Seq=1 Ack=90 Win=64240 Len=0
5	0.022101758	192.168.74.2	192.168.74.128	DNS	148	Standard query response 0x8940 A www.australia.gov.au CNAME b2.shared.global.fastly.net A 151.101.86.217 OPT
6	0.028069161	192.168.74.2	192.168.74.128	DNS	193	Standard query response 0x476f AAAA www.australia.gov.au CNAME b2.shared.global.fastly.net SOA ns1.fastly.net OPT
7	0.028356155	192.168.74.128	192.168.74.2	DNS	98	Standard query 0x23d2 AAAA b2.shared.global.fastly.net OPT
8	0.032165796	192.168.74.2	192.168.74.128	DNS	159	Standard query response 0x23d2 AAAA b2.shared.global.fastly.net SOA ns1.fastly.net OPT
9	0.403612147	151.101.86.217	192.168.74.128	TLSv1.2	160	Application Data
10	0.429240231	192.168.74.128	151.101.86.217	TLSv1.2	197	Application Data
11	0.429542644	151.101.86.217	192.168.74.128	TCP	60	443 → 53892 [ACK] Seq=107 Ack=233 Win=64240 Len=0
12	0.430179468	192.168.74.128	151.101.86.217	TLSv1.2	198	Application Data
13	0.430284866	151.101.86.217	192.168.74.128	TCP	60	443 → 53892 [ACK] Seq=107 Ack=377 Win=64240 Len=0
14	0.430454681	192.168.74.128	151.101.86.217	TLSv1.2	199	Application Data

Figur 5: Wireshark billede af når man åbner browseren og går ind på www.australia.gov.au

7. Er der forskel på tidsforsinkelses-målingerne i punkt 5 og punkt 6? Hvis der er forskel, hvad kan årsagen være til at der denne forskel?

På Figur 5 vises at det tager længere tid at anmode om at hente en pakke ved den australske hjemmeside. Begrundelse for dette er at den australske server er længere væk end den danske server, hvor propagation delay bliver meget større. Det tager ca. 41ms for at modtage den første respons.

8. Undersøg vha. Wireshark hvad der sker, når denne web-side hentes vha. en Web Browser: kurser.iha.dk/eit/it-dkt1/test.htm

På Figur 6 ses Wireshark når www.kurser.iha.dk forespørges. Læg særlig opmærksom på nummer 10 på Figur 6. Der bliver hentet noget fra denne side imellem 7 og 17. Det der specifikt bliver hentet er en http side med version 1.1 som er ved nummer 10. Der kommer et TCP three-way handshake imellem H1 og hjemmesiden ved 7 og 8. Igennem 11 og 16 bliver der sendt konstant noget data hvor 17 fortæller med FIN at hjemmesiden er modtaget.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.74.128	192.168.74.2	DNS	84	Standard query 0xb3f3 A kurser.iha.dk OPT
2	0.000281333	192.168.74.128	192.168.74.2	DNS	84	Standard query 0x4cb6 AAAA kurser.iha.dk OPT
3	0.002609988	192.168.74.2	192.168.74.128	DNS	124	Standard query response 0xb3f3 A kurser.iha.dk CNAME webhotel5.iha.dk A 10.20.255.5 OPT
4	0.002616774	192.168.74.2	192.168.74.128	DNS	164	Standard query response 0x4cb6 AAAA kurser.iha.dk CNAME webhotel5.iha.dk SOA iha01.iha.dk OPT
5	0.003224523	192.168.74.128	192.168.74.2	DNS	87	Standard query 0xcc59 AAAA webhotel5.iha.dk OPT
6	0.005511970	192.168.74.2	192.168.74.128	DNS	143	Standard query response 0xcc59 AAAA webhotel5.iha.dk SOA iha01.iha.dk OPT
7	0.005903930	192.168.74.128	10.20.255.5	TCP	74	55960 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1483644146 TSecr=0 WS=128
8	0.008327729	10.20.255.5	192.168.74.128	TCP	60	80 → 55960 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
9	0.008353021	192.168.74.128	10.20.255.5	TCP	54	55960 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
10	0.008468231	192.168.74.128	10.20.255.5	HTTP	519	GET /eit/it-dkt1/test.htm HTTP/1.1
11	0.008566775	10.20.255.5	192.168.74.128	TCP	60	80 → 55960 [ACK] Seq=1 Ack=466 Win=64240 Len=0
12	0.014579709	10.20.255.5	192.168.74.128	TCP	1514	80 → 55960 [ACK] Seq=1 Ack=466 Win=64240 Len=1460 [TCP segment of a reassembled PDU]
13	0.014589760	192.168.74.128	10.20.255.5	TCP	54	55960 → 80 [ACK] Seq=466 Ack=1461 Win=32120 Len=0
14	0.014641862	10.20.255.5	192.168.74.128	TCP	1354	80 → 55960 [PSH, ACK] Seq=1461 Ack=466 Win=64240 Len=1300 [TCP segment of a reassembled PDU]
15	0.014647847	192.168.74.128	10.20.255.5	TCP	54	55960 → 80 [ACK] Seq=466 Ack=2761 Win=35040 Len=0
16	0.015757853	10.20.255.5	192.168.74.128	HTTP	1074	HTTP/1.1 200 OK (text/html)
17	0.015945038	192.168.74.128	10.20.255.5	TCP	54	55960 → 80 [FIN, ACK] Seq=466 Ack=3782 Win=37960 Len=0
18	0.016045279	10.20.255.5	192.168.74.128	TCP	60	80 → 55960 [ACK] Seq=3782 Ack=467 Win=64239 Len=0

Figur 6: Wireshark billede af når man går ind på kurser.iha.dk

10. Etabler en LAN-forbindelse mellem en web-serveren (H1) og en web-client (H2) I første omgang anvendes en simpel, telnet baseret web-client i H2.

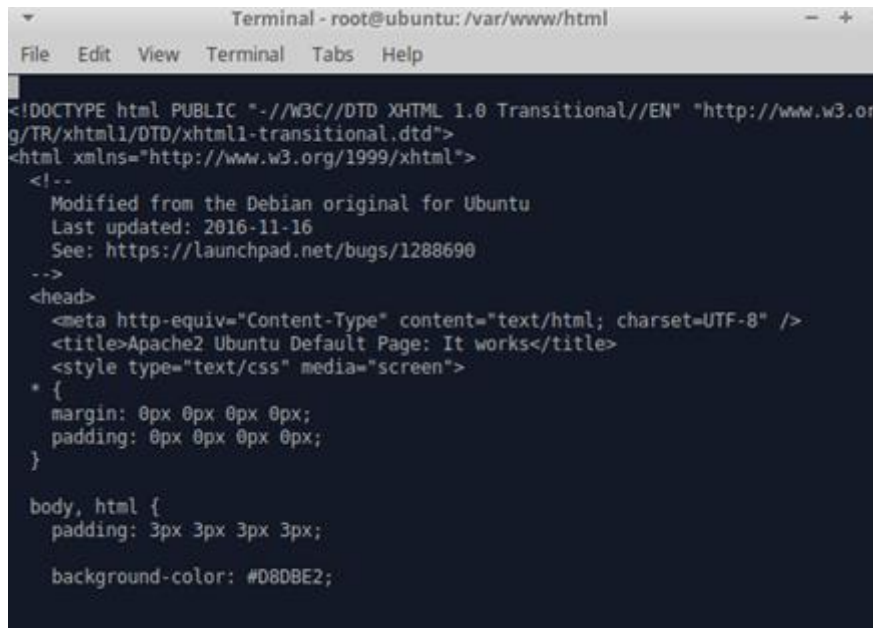
Figur 7 viser GET / med 127.0.0.01

```
Escape character is '^]'.
Connection closed by foreign host.
root@ubuntu:~# telnet 127.0.0.1 80
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
GET /

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<!--
  Modified from the Debian original for Ubuntu
  Last updated: 2016-11-16
  See: https://launchpad.net/bugs/1288690
-->
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Apache2 Ubuntu Default Page: It works</title>
  <style type="text/css" media="screen">
  * {
    margin: 0px 0px 0px 0px;
    padding: 0px 0px 0px 0px;
  }
</style>
</head>
```

Figur 7: H1 - GET/

Figur 8 viser indholdet af filen



```
Terminal - root@ubuntu: /var/www/html
File Edit View Terminal Tabs Help

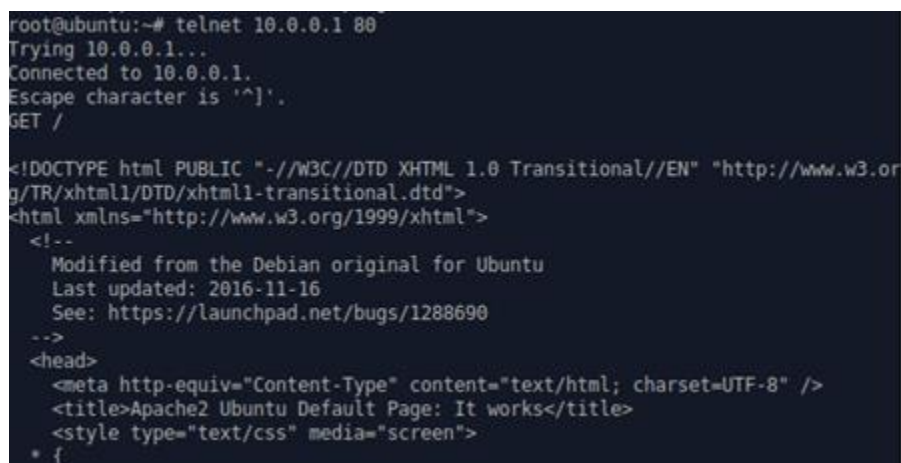
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <!--
    Modified from the Debian original for Ubuntu
    Last updated: 2016-11-16
    See: https://launchpad.net/bugs/1288690
  -->
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Apache2 Ubuntu Default Page: It works</title>
    <style type="text/css" media="screen">
      * {
        margin: 0px 0px 0px 0px;
        padding: 0px 0px 0px 0px;
      }

      body, html {
        padding: 3px 3px 3px 3px;

        background-color: #D8DBE2;
      }
    </style>
  </head>
  <body>
  </body>
</html>
```

Figur 8: H1 - /var/www/html

Figur 9 viser GET / med http fra H2 til H1.



```
root@ubuntu:~# telnet 10.0.0.1 80
Trying 10.0.0.1...
Connected to 10.0.0.1.
Escape character is '^'.
GET /

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <!--
    Modified from the Debian original for Ubuntu
    Last updated: 2016-11-16
    See: https://launchpad.net/bugs/1288690
  -->
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Apache2 Ubuntu Default Page: It works</title>
    <style type="text/css" media="screen">
      * {
        margin: 0px 0px 0px 0px;
        padding: 0px 0px 0px 0px;
      }

      body, html {
        padding: 3px 3px 3px 3px;

        background-color: #D8DBE2;
      }
    </style>
  </head>
  <body>
  </body>
</html>
```

Figur 9: H2 – telnet 10.0.0.1 80

11. Test protokollerne: HTTP 0.9, 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 (uden/med pipelining).

Figur 10 viser test af GET med version HTTP 0.9, og det ses her at TCP-forbindelsen automatisk lukkes ved anvendelse

```

root@ubuntu:~# telnet 127.0.0.1 80
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
GET / HTTP/0.9
HTTP/1.1 400 Bad Request
Date: Thu, 08 Feb 2018 09:05:46 GMT
Server: Apache/2.4.27 (Ubuntu)
Content-Length: 301
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.27 (Ubuntu) Server at 127.0.1.1 Port 80</address>
</body></html>
Connection closed by foreign host.
root@ubuntu:~#

```

Figur 10: HTTP 0.9

Figur 11 viser test af GET med version HTTP 1.0, og det ses her at TCP-forbindelsen automatisk lukkes ved anvendelse

```

root@ubuntu:~# telnet 127.0.0.1 80
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
GET / HTTP/1.0
host: 10.0.0.1

HTTP/1.1 200 OK
Date: Thu, 08 Feb 2018 09:04:31 GMT
Server: Apache/2.4.27 (Ubuntu)
Last-Modified: Thu, 08 Feb 2018 08:34:22 GMT
ETag: "2aa6-564af475e6749"
Accept-Ranges: bytes
Content-Length: 10918
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

```

Figur 11: HTTP 1.0

Figur 12 viser test af GET med version HTTP 1.1, og det ses her at TCP-forbindelsen ikke lukkes med det samme.

```

root@ubuntu:~# telnet 127.0.0.1 80
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^['.
GET / HTTP/1.1
host: 10.0.0.1

HTTP/1.1 200 OK
Date: Thu, 08 Feb 2018 08:59:29 GMT
Server: Apache/2.4.27 (Ubuntu)
Last-Modified: Thu, 08 Feb 2018 08:34:22 GMT
ETag: "2aa6-564af475e6749"
Accept-Ranges: bytes
Content-Length: 10918
Vary: Accept-Encoding
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.or

```

Figur 12: HTTP 1.1

Figur 13 viser en fortsættelse af Figur 12 efter der er gået få, og det ses her at forbindelsen bliver lukket af H1 (server).

```

</body>
</html>

Connection closed by foreign host.
root@ubuntu:~#

```

Figur 13: Viser hvordan det ser ud når serveren lukker forbindelsen

Figur 14 viser apache server versionen som er Apache/2.4.27

```

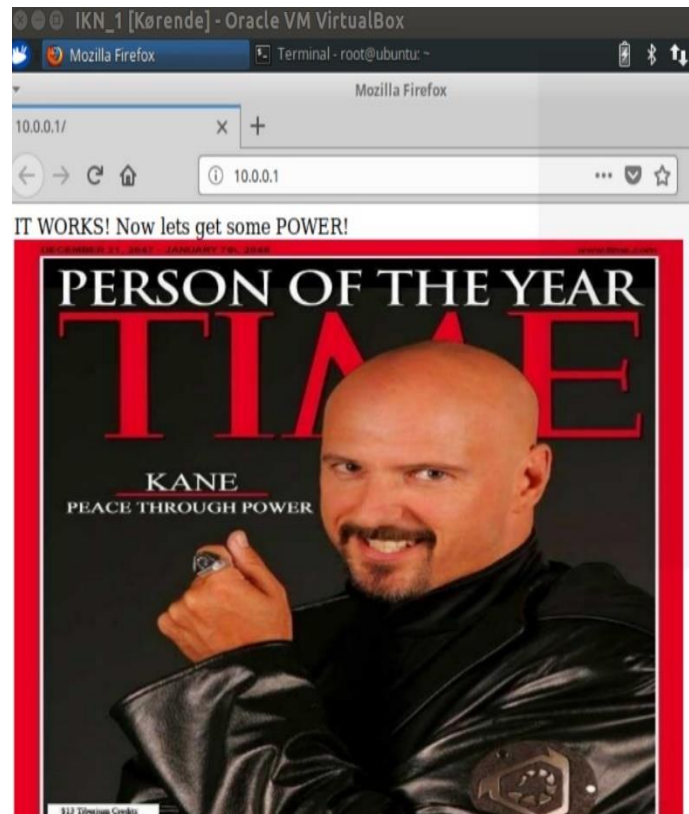
root@ubuntu:~# apache2 -v
Server version: Apache/2.4.27 (Ubuntu)
Server built: 2017-09-18T15:05:48
root@ubuntu:~#

```

Figur 14: Fremvisning af apache server version

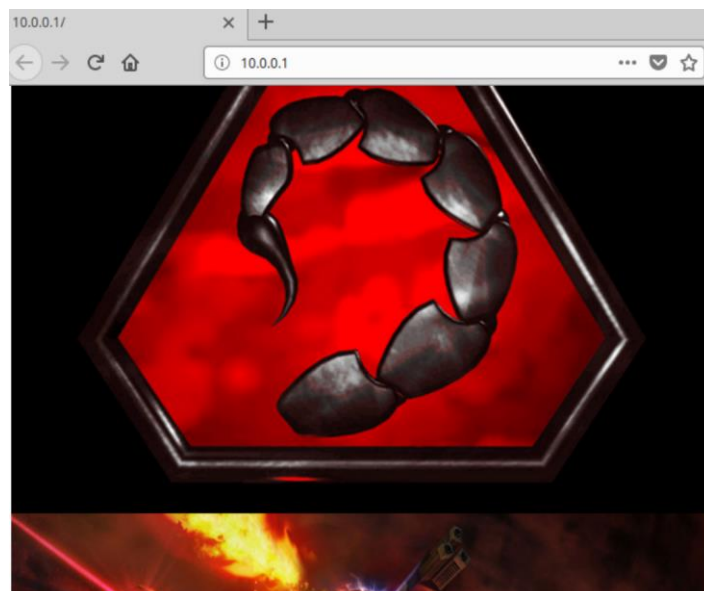
12. Anvend Firefox web-browser som web-client i H2 sammen med Apache web-server i H1

På Figur 15 fremvises billede (fil) 1.



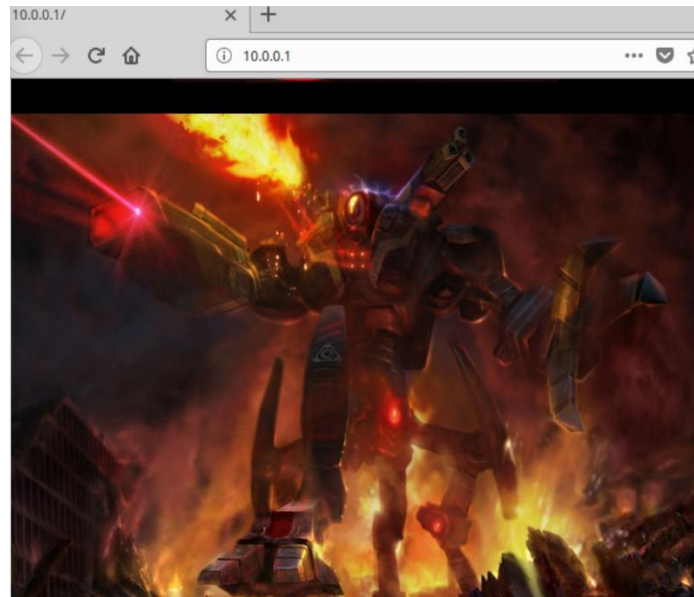
Figur 15: Åbnes igennem terminal med Firefox 10.0.0.1, med lidt magi får man nogle flotte billeder op

På Figur 16 fremvises billede (fil) 2.



Figur 16: Åbnes igennem terminal med Firefox 10.0.0.1, med lidt magi får man nogle flotte billeder op

På Figur 17 fremvises billede (fil) 3.



Figur 17: Åbnes igennem terminal med Firefox 10.0.0.1, med lidt magi får man nogle flotte billeder op

Apache server platform er en Linux baseret server/host platform der har en masse gode funktioner til at oprette forskellige type server, som eks. en webserver, FTP, osv. Den har også et funktionelt GUI og er som mange andre platformer afbenyttes med PHP og HTML kodning af hjemmesiderne. På Figur 15 fremvises en modificeret Apache server med nogle billeder og tekst. Standard udgaven vil have en vejledning til hvordan man afbenytter sig af Apache serveren såvel hvilken version Apache server der bliver brugt. I det her tilfælde kan det ikke ses at der bruges en Apache2 server pga. modifikationen.

Som der kan ses på Figur 15, Figur 16 og Figur 17, så er der lagt 3 billeder ind på serveren. Disse 3 billeder bliver hentet som der vises på Figur 18 ved no 13, 17 og 22, hvor H2 anmoder om at få billederne. Derefter kan de ses på Figur 19 at her svarer H1 med at den har fuldført at sende det der blev bedt om.

På Figur 20 vises koden til hjemmesiden.

På Figur 18 bliver der vist at H2 laver en forespørgsel på at få de 3 forskellige billeder.

No.	Time	Source	Destination	Protocol	Length	Info
13	1.734523922	10.0.0.2	10.0.0.1	HTTP	326	GET /Peace.jpg HTTP/1.1
14	1.738374149	10.0.0.2	10.0.0.1	TCP	74	59644 → 80 [SYN] Seq=0 Win=29200 Len=0
15	1.738585311	10.0.0.1	10.0.0.2	TCP	74	80 → 59644 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0
16	1.738601391	10.0.0.2	10.0.0.1	TCP	66	59644 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
17	1.739401847	10.0.0.2	10.0.0.1	HTTP	328	GET /NodLogo.jpg HTTP/1.1
18	1.739604652	10.0.0.1	10.0.0.2	TCP	66	80 → 59644 [ACK] Seq=1 Ack=263 Win=32768 Len=0
19	1.740383417	10.0.0.2	10.0.0.1	TCP	74	59646 → 80 [SYN] Seq=0 Win=29200 Len=0
20	1.740655320	10.0.0.1	10.0.0.2	TCP	74	80 → 59646 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0
21	1.740688348	10.0.0.2	10.0.0.1	TCP	66	59646 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
22	1.740973094	10.0.0.2	10.0.0.1	HTTP	330	GET /NODreedeem.jpg HTTP/1.1
23	1.741249885	10.0.0.1	10.0.0.2	TCP	66	80 → 59646 [ACK] Seq=1 Ack=265 Win=32768 Len=0
24	1.778367347	10.0.0.1	10.0.0.2	TCP	66	80 → 59632 [ACK] Seq=708 Ack=575 Win=0 Len=0

Frame 13: 326 bytes on wire (2608 bits), 326 bytes captured (2608 bits) on interface 0
Ethernet II, Src: PcsCompu_31:60:41 (08:00:27:31:60:41), Dst: PcsCompu_11:f9:c4 (08:00:27:11:f9:c4)
Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.1
Transmission Control Protocol, Src Port: 59632, Dst Port: 80, Seq: 315, Ack: 708, Len: 260
Hypertext Transfer Protocol

Figur 18: Wireshark billede af at H2 anmoder om billederne med GET /

På Figur 19 kan der ses på numrene 55 og 61 at H1 har fuldført sin opgave med at sende billederne til H2.

No.	Time	Source	Destination	Protocol	Length	Info
51	1.847618694	10.0.0.1	10.0.0.2	TCP	21786	80 → 59646 [ACK] Seq=43441 Ack=265
52	1.847624409	10.0.0.2	10.0.0.1	TCP	66	59646 → 80 [ACK] Seq=265 Ack=65161
53	1.847631895	10.0.0.1	10.0.0.2	TCP	14546	80 → 59644 [ACK] Seq=43441 Ack=263
54	1.847636400	10.0.0.2	10.0.0.1	TCP	66	59644 → 80 [ACK] Seq=263 Ack=57921
55	1.847679060	10.0.0.1	10.0.0.2	HTTP	12459	HTTP/1.1 200 OK (JPEG JFIF image)
56	1.847683562	10.0.0.2	10.0.0.1	TCP	66	59644 → 80 [ACK] Seq=263 Ack=70314
57	1.847721881	10.0.0.1	10.0.0.2	TCP	18890	80 → 59646 [ACK] Seq=65161 Ack=265
58	1.847726317	10.0.0.2	10.0.0.1	TCP	66	59646 → 80 [ACK] Seq=265 Ack=83985
59	1.847733664	10.0.0.1	10.0.0.2	TCP	10202	80 → 59632 [ACK] Seq=44148 Ack=575
60	1.847737215	10.0.0.2	10.0.0.1	TCP	66	59632 → 80 [ACK] Seq=575 Ack=54284
61	1.847954945	10.0.0.1	10.0.0.2	HTTP	5971	HTTP/1.1 200 OK (JPEG JFIF image)
62	1.847960580	10.0.0.2	10.0.0.1	TCP	66	59632 → 80 [ACK] Seq=575 Ack=60189

Figur 19: Wireshark billede af at H1 har fuldført sin opgave med at sende billederne til H2

På Figur 20 er HTML filen index.html og dens indhold. Billederne er hardcoded ind så apache serveren går efter de filer der er ved selve HTML filen (sti: var/www/). Billederne er også formatteret således at det er hardcoded size og kunne blive ændret til procenter for mere dynamisk system.

```

File Edit Search View Document Help
Warning, you are using the root account, you may harm your system.
1
2<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3
3<html xmlns="http://www.w3.org/1999/xhtml">
4
5  <!--
6    Modified from the Debian original for Ubuntu
7    Last updated: 2016-11-16
8    See: https://launchpad.net/bugs/1288690
9  -->
10 IT WORKS!
11 Now lets get some POWER!
12
13 
14
15 
16 
17
18
19 </html>
20
21

```

Figur 20: HTML filen der er blevet redigeret således at billederne kommer på hjemmesiden

Øvelse 6

Opgaveformulering

Der bruges 2 virtuelle maskiner. En maskine, der fungerer som en server, og en anden maskine, der fungerer som en klient. Serveren har det ansvar at skulle kunne sende filer af vilkårlige typer/størrelser til klienten, ud fra en forespørgsel fra klienten. Forespørgslen foregår således at klienten sender en tekststreng, der udskrives på terminalen, til serveren. Denne tekststreng skal indeholde et filnavn + en eventuel sti angivelse til en fil, der eksisterer i serveren.

Serveren skal sende filen i segmenter på max. 1000 bytes ad gangen indtil filen er overført, hvis filen altså eksisterer, ellers skal der udskrives en fejlmeddelelse. Uanset om filen eksisterer eller ej, skal serveren kunne håndtere en ny forespørgsel, når den er klar på til at sende en ny fil, dvs. den skal være iterativ.

Kodeforklaring

For at få udarbejdet en TCP server/client er disse to blevet udarbejdet parallelt. En simpel server og klient blev skrevet hvor der blev forsøgt at sende en simpel karakter eller streng fra klient til server. Når dette lykkedes, blev resten af applikationen udført.

Server

Forløbet er udført således, at serveren starter med at bruge klassen `TcpListener` til at lytte efter en TCP-forbindelse fra klienten som kan ses på følgende kodeudsnit:

```
TcpListener serverSocket = new TcpListener(IPAddress.Any, PORT);
TcpClient clientSocket = default(TcpClient);
serverSocket.Start();
clientSocket = serverSocket.AcceptTcpClient();
```

Serveren åbner en forbindelse med en port defineret som 9000, og venter herefter på en klient. Dette kald er blokerende indtil en klient forbindes til serveren.

Når en klient er forbundet, bliver der etableret en `NetworkStream` for klienten for at gøre det muligt at sende og modtage data igennem en stream. Der modtages en streng fra klienten som indeholder en sti. Der tjekkes om filen, som stien referer til, findes. Findes denne fil, vil en filstørrelse returneres, ellers vil en værdi på 0 returneres:

```
// Get the network file stream
NetworkStream clientStream = clientSocket.GetStream();

string filename = LIB.readTextTCP (clientStream);
long fileSize = LIB.check_File_Exists (filename);
```

Hvis filen ikke findes, vil serveren udskrive en fejlmeddelelse til konsollen samt en fejlmeddelelse til klienten som spørger om en anden sti. Stien tjekkes indtil en valid fil findes:

```

while (fileSize == 0)
{
    string errorMsg = "File '" + filename + "' not found";
    Console.WriteLine(errorMsg);

    LIB.writeTextTCP (clientStream, fileSize.ToString());

    filename = LIB.readTextTCP (clientStream);
    fileSize = LIB.check_File_Exists (filename);
}

```

Herefter vil en fil blive overført:

```

Console.WriteLine ("Filename: " + filename);
Console.WriteLine("Size: " + fileSize);
LIB.writeTextTCP (clientStream, fileSize.ToString());

sendFile (filename, clientStream);

```

hvor sendFile() er defineret som følgende:

```

private void sendFile (String fileName, NetworkStream io)
{
    Console.WriteLine ("Sending file ..");

    FileStream fs = new FileStream (fileName, FileMode.Open, FileAccess.Read);
    Byte[] fileToSend = new Byte[BUFSIZE]; //Changed to bufsize maks send6

    int bytesToSend = 0;

    while ((bytesToSend = fs.Read (fileToSend, 0, fileToSend.Length)) > 0) //I
    {
        io.Write (fileToSend, 0, bytesToSend); //I must send that byte

        Console.WriteLine ($"Send {bytesToSend} bytes");
    }
    Console.WriteLine ("File sent");
}

```

Denne kode opretter en FileStream på filnavnet. En buffer på 1000 bytes oprettes, hvilket er den største mængde bytes der bliver sendt per sending. FileStream læser på filen og indsætter dette i bufferen. NetworkStream skriver herefter til klienten indeholdende bufferen. Hvis fs.Read() returnerer et tal over 0, betyder dette at der stadig er data som skal sendes. Når al data er overført (fs.Read() returnerer 0), teminerer while-loopet og filen er overført.

Forbindelsen lukkes herefter så en ny klient kan forbindes:

```

clientSocket.Close();
serverSocket.Stop();

```

For at gøre serveren iterativ er den blot indsat i en while-løkke, som vil starte en forbindelse og lukke den herefter.

Client

Følgende kode er på klientens side, hvor der oprettes en forbindelse til serveren. Argumentet tjekkes først:

```
TcpClient clientSocket = new TcpClient();

if (args.Length != 2)
{
    Console.WriteLine ("Please supply two arguments: IP-address of the server and path + filename");
    Environment.Exit (0);
}

clientSocket.Connect(args[0], PORT);
Console.WriteLine ($"Connected to '{args[0]}'");
```

Herefter oprettes en `NetworkStream` ligesom i serveren. Følgende kodeudsnit viser at klienten sender en tekststreng til at forespørge om et filnavn. Hvis filen ikke findes, bliver det muligt at sende en ny forespørgsel:

```
NetworkStream serverStream = clientSocket.GetStream();

Console.WriteLine($"Requesting filename '{args[1]}'");

string filename = args [1];
LIB.writeTextTCP (serverStream, filename);

long fileSize = LIB.getFileSizeTCP (serverStream);

while (fileSize == 0)
{
    Console.WriteLine ("File not found. Input a valid file");
    filename = Console.ReadLine ();

    Console.WriteLine($"Requesting filename '{filename}'");

    LIB.writeTextTCP (serverStream, filename);
    fileSize = LIB.getFileSizeTCP (serverStream);
}

Console.WriteLine ("File size: " + fileSize);

receiveFile (filename, serverStream, fileSize);
```

Det meste af klienten er ret trivielt, udover `receiveFile()`:

```
receiveFile (filename, serverStream, fileSize);
```

Følgende kode viser der, hvor klienten modtager filen fra serveren. Klienten anmoder om et filnavn, der findes i serveren, og herefter bliver størrelsen af filen registreret. Klienten sørger for at oprette et directory til at opretholde filen. En `FileStream` oprettes som udskriver til mappen. While-løkken bliver ved med at læse fra `NetworkStream` til `FileStream` indtil hele filen er overført. Dette sker med bufferen på max 1000 bytes.


```

private void receiveFile (String fileName, NetworkStream io, long fileSize)
{
    fileName = LIB.extractFileName(fileName);
    string dataDir = "/root/Desktop/ExFiles/";
    Directory.CreateDirectory (dataDir);
    FileStream file = new FileStream (dataDir + fileName, FileMode.Create, FileAccess.Write);
    byte[] data = new byte[BUFSIZE]; //Vi modtager kun 1k bytes af gangen

    int totalBytes = 0;
    int bytesRead;

    Console.WriteLine ("Reading file " + fileName + " ... ");

    //while ((bytesRead = io.Read(data, 0, data.Length)) > 0) //Nu bliver den ved indtil længden af det d
    while(fileSize > totalBytes)
    {
        bytesRead = io.Read (data, 0, data.Length);
        file.Write (data, 0, bytesRead);

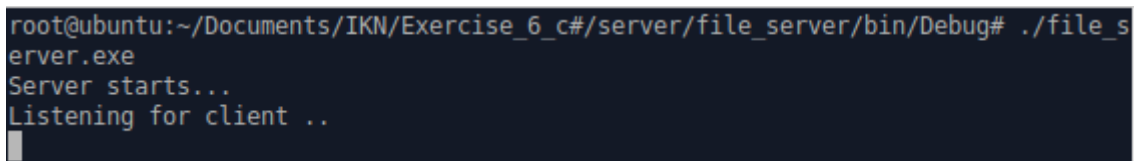
        totalBytes += bytesRead;

        Console.WriteLine ("Read bytes: " + bytesRead.ToString () + "\t Total bytes read:" + totalBytes);
    }
}

```

Resultater

På Figur 21 ses at serveren venter på en TCP-connection fra klienten.



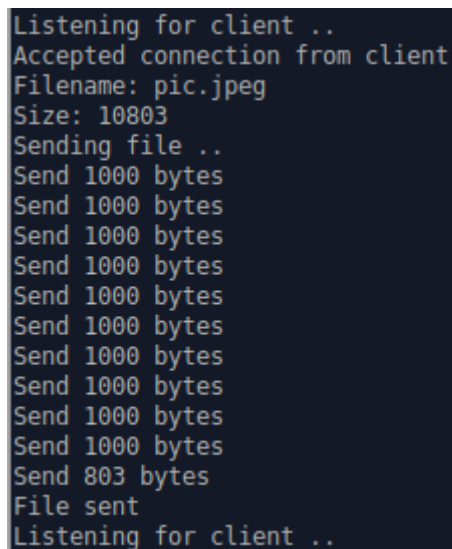
```

root@ubuntu:~/Documents/IKN/Exercise_6_c#/server/file_server/bin/Debug# ./file_s
erver.exe
Server starts...
Listening for client ..

```

Figur 21 Server venter på en klient

Figur 22 viser overførslen af en fil fra serveren. Filen er en jpeg-format på ~11 kb. Det ses at der sendes 1000 bytes ad gangen, indtil der til sidst kun er 803 bytes tilbage af filen.



```

Listening for client ..
Accepted connection from client
Filename: pic.jpeg
Size: 10803
Sending file ..
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 803 bytes
File sent
Listening for client ..

```

Figur 22 Overførsel af en fil fra serveren

På Figur 23 ses den måde klienten modtager denne jpeg fil på ~11kb. Øverst i terminalen anmoder klienten denne "pic.jpeg" fra serveren som er 10.0.0.1. Herefter sender overføres filen fra serveren og der ses at klienten til sidst modtager denne fil.

```
root@ubuntu:~/Documents/IKN/Exercise_6_c#/client/file_client/bin/Debug# ./file_client.exe 10.0.0.1 /root/Documents/IKN/Exercise_6_c#/server/file_server/index.jpeg
Client starts...
Connected to '10.0.0.1'
Requesting filename '/root/Documents/IKN/Exercise_6_c#/server/file_server/index.jpeg'
File size: 10803
Reading file index.jpeg ...
Read bytes: 1000      Total bytes read:1000
Read bytes: 1000      Total bytes read:2000
Read bytes: 1000      Total bytes read:3000
Read bytes: 1000      Total bytes read:4000
Read bytes: 1000      Total bytes read:5000
Read bytes: 1000      Total bytes read:6000
Read bytes: 1000      Total bytes read:7000
Read bytes: 1000      Total bytes read:8000
Read bytes: 1000      Total bytes read:9000
Read bytes: 1000      Total bytes read:10000
Read bytes: 803      Total bytes read:10803
File received
root@ubuntu:~/Documents/IKN/Exercise_6_c#/client/file_client/bin/Debug#
```

Figur 23 Klient modtager en fil

Som kvalitetskontrol, bliver der sikret, at den sendte fil og den modtagende fil er ens. Her bruges diff-kommandoen i terminalen som ses på Figur 24 til at tjekke om de to filer er ens. Der ses her at der bliver udskrevet "are identical", hvilket vil sige at de to filer er identiske.

```
root@ubuntu:~/Documents/IKN/Exercise_6_c#/client/file_client/bin/Debug# diff -s pic.jpeg ~/Documents/IKN/Exercise_6_c#/server/file_server/pic.jpeg
Files pic.jpeg and /root/Documents/IKN/Exercise_6_c#/server/file_server/pic.jpeg are identical
root@ubuntu:~/Documents/IKN/Exercise_6_c#/client/file_client/bin/Debug#
```

Figur 24 Tjek af at filerne er ens

På Figur 25 bliver der sendt en mp4-fil på ~1mb til klienten fra serveren. Dette bliver overført på samme måde som jpeg-filen, nemlig 1000 bytes ad gangen. Dette er kun et udsnit af slutningen af overførslen.

```
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 736 bytes
File sent
Listening for client ..
```

Figur 25 Overførsel af en stor fil (1 mb)

På Figur 26 ses starten af, hvordan klienten modtager mp4-filen som blev sendt fra serveren.

```
Read bytes: 1000      Total bytes read:122448
Read bytes: 1000      Total bytes read:123448
Read bytes: 344      Total bytes read:123792
Read bytes: 1000      Total bytes read:124792
Read bytes: 1000      Total bytes read:125792
Read bytes: 1000      Total bytes read:126792
Read bytes: 656      Total bytes read:127448
Read bytes: 1000      Total bytes read:128448
Read bytes: 1000      Total bytes read:129448
Read bytes: 1000      Total bytes read:130448
Read bytes: 1000      Total bytes read:131448
Read bytes: 1000      Total bytes read:132448
Read bytes: 792      Total bytes read:133240
Read bytes: 1000      Total bytes read:134240
```

Figur 26 Klient er i gang med at modtage filen

På Figur 27 ses der, hvor klienten er færdig med at modtage mp4-filen som blev sendt fra serveren.

```
Read bytes: 1000      Total bytes read:1048832
Read bytes: 1000      Total bytes read:1049832
Read bytes: 1000      Total bytes read:1050832
Read bytes: 1000      Total bytes read:1051832
Read bytes: 1000      Total bytes read:1052832
Read bytes: 32      Total bytes read:1052864
Read bytes: 1000      Total bytes read:1053864
Read bytes: 1000      Total bytes read:1054864
Read bytes: 872      Total bytes read:1055736
File received
root@ubuntu:~/Documents/IKN/Exercise_6_c#/client/file_client/bin/Debug#
```

Figur 27 Klient har modtaget filen

På Figur 28 ses serveren, hvor den prøver at sende en fil der ikke findes, hvorefter det bliver muligt at sende en ny fil. Efter der er blevet skrevet et rigtigt filnavn, nemlig pic.jpeg, bliver filen overført til klienten, og derefter lytter den igen til klienten. Dette er fordi serveren er iterativ, så den bliver hele tiden ved med at lytte på forespørgsler fra klienten efter hver filoverførsel indtil serveren lukkes ned.

```
root@ubuntu:~/Documents/IKN/Exercise_6_c#/server/file_server/bin/Debug# ./file_server.exe
Server starts...
Listening for client ..
Accepted connection from client
File '/root/Documents/IKN/Exercise_6_c#/server/file_server/video.mp' not found
Filename: /root/Documents/IKN/Exercise_6_c#/server/file_server/pic.jpeg
Size: 10803
Sending file ..
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 1000 bytes
Send 803 bytes
File sent
Listening for client ..
```

Figur 28 Blå pil peger på en invalid fil

På Figur 29 klienten, hvor den anmoder om ikke eksisterende filer. Derefter bliver det muligt for den igen at lave en forespørgsel. Selv efter den rigtige fil er blevet overført, er det muligt for klienten at sende en ny forespørgsel.

```
root@ubuntu:~/Documents/IKN/Exercise_6_c#/client/file_client/bin/Debug# ./file_client.exe 10.0.0.1 /root/Documents/IKN/Exercise_6_c#/server/file_server/video.mp
Client starts...
Connected to '10.0.0.1'
Requesting filename '/root/Documents/IKN/Exercise_6_c#/server/file_server/video.mp'
File not found. Input a valid file
/root/Documents/IKN/Exercise_6_c#/server/file_server/pic.jpeg
Requesting filename '/root/Documents/IKN/Exercise_6_c#/server/file_server/pic.jpeg'
File size: 10803
Reading file pic.jpeg ...
Read bytes: 1000      Total bytes read:1000
Read bytes: 1000      Total bytes read:2000
Read bytes: 1000      Total bytes read:3000
Read bytes: 1000      Total bytes read:4000
Read bytes: 1000      Total bytes read:5000
Read bytes: 1000      Total bytes read:6000
Read bytes: 1000      Total bytes read:7000
Read bytes: 1000      Total bytes read:8000
Read bytes: 1000      Total bytes read:9000
Read bytes: 1000      Total bytes read:10000
Read bytes: 803      Total bytes read:10803
File received
root@ubuntu:~/Documents/IKN/Exercise_6_c#/client/file_client/bin/Debug#
```

Figur 29 Klient forespørger om invalid fil og herefter om en valid fil

På Figur 30 laves der en kvalitetskontrol, for at tjekke, om de to filer er identiske. Der bliver derefter skrevet "are identical" på terminalen, hvilken betyder at begge filer er helt identiske som forventet.

```
root@ubuntu:~/Documents/IKN/Exercise_6_c#/client/file_client/bin/Debug# diff -s  
video.mp4 ~/Documents/IKN/Exercise_6_c#/server/file_server/video.mp4  
Files video.mp4 and /root/Documents/IKN/Exercise_6_c#/server/file_server/video.m  
p4 are identical  
root@ubuntu:~/Documents/IKN/Exercise_6_c#/client/file_client/bin/Debug#
```

Figur 30 Kvalitetskontrol på filerne

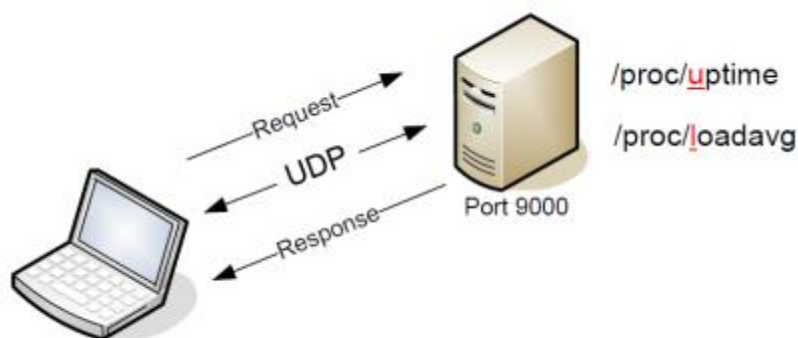
Øvelse 7

Opgaveformulering

Der bruges 2 virtuelle maskiner. En maskine bruges som server og en anden maskine bruges som klient.

Serveren har det ansvar, at skulle kunne modtage en forespørgsel fra klienten, bestående af enten et "U" eller "L". Hvis serveren modtager et "U" skal den returnere /proc/uptime tilbage til klienten, som indeholder information om den samlede tid, serveren har været kørende siden start. Hvis serveren modtager et "L" skal den returnere /proc/loadavg, som indeholder information om serverens aktuelle CPU-load.

Klienten skal kunne sende "u", "U", "l" eller "L" til UDP-serveren, og derefter modtage det som serveren returnere, som nævnt tidligere.



Figur 31 Opstilling

Kodeforklaring

En server og klient er blevet implementeret hvor serveren er iterativ. En UDP server/client forbindelse er meget mere løs end en TCP forbindelse og kræver ikke at man vedligeholder en forbindelse mellem server og klient som der sker i TCP. Serveren skal etablere et end point med et portnummer som en klient kan forbindes på. Herefter er der mulighed for at sende data til og fra hinanden.

Server

Koden til serveren etablerer en ny UDPClient med et portnummer som anvendes i sit end point.

```
var server = new UdpClient (PORT);

// Establish end point
IPEndPoint endPoint = new IPEndPoint(IPAddress.Any, PORT);
```

Herefter vil serveren blot stå i et while-loop og modtage data fra en client. Dette input vil valideres hvorefter der vil sendes en streng tilbage med uptime eller loadavg (eller en fejlbesked hvis inputtet ikke er valid):

```

while (true)
{
    // Receive client input
    byte[] receivedData = server.Receive (ref endPoint); //listen on port 9000
    string data = Encoding.ASCII.GetString (receivedData);

    Console.WriteLine ("Received client input: " + data);

    // Get the requested info
    string measure = GetMeasurement (data);

    // Send data back
    byte[] sentdataback = Encoding.ASCII.GetBytes (measure);
    server.Send (sentdataback, sentdataback.Length, endPoint);
}

```

GetMeasurement() vil returnere en string baseret på inputtet:

```

public string GetMeasurement(string letter)
{
    string filePath;

    switch (letter = letter.ToUpper())
    {
        case "U":
            filePath = "/proc/uptime";
            Console.WriteLine ("Sending uptime");
            break;
        case "L":
            filePath = "/proc/loadavg";
            Console.WriteLine ("Sending loadavg");
            break;
        default:
            Console.WriteLine ("Bad input. Sending error message");
            return "Bad input. Valid inputs are l (L) or u (U)\n";
    }
    return "Reading from " + filePath + ": " + File.ReadAllText (filePath);
}

```

Som der kan ses vil uptime returneres ved "u", loadavg på "l" og default en fejlmeddelelse. File.ReadAllText() anvendes for at læse fra en fil.

Client

Klienten vil også oprette en ny UdpClient, dog med intet portnummer registreret. Herefter vil klienten så simpelt som skrive en besked til serveren og få returneret et svar tilbage baseret på input:

```
// Create new UDPClient
var client = new UdpClient ();

// Connect to end point on host with PORT 9000
IPEndPoint endPoint = new IPEndPoint(IPAddress.Parse(args[0]), PORT);
client.Connect(endPoint);

// Send reading request to server
byte[] sendData = Encoding.ASCII.GetBytes (args[1]);
client.Send (sendData, sendData.Length);

// Receive data from server and print on screen
byte[] receivedData = client.Receive (ref endPoint);
string data = Encoding.ASCII.GetString (receivedData);
Console.WriteLine (data);
```

Resultater

Figur 32 viser klienten som sendet et U til serveren og modtager uptime.

```
root@ubuntu:~/Documents/IKN/Exercise_7_c#/UDPclient/UDPclient/bin/Debug# ./UDPcl
ient.exe 10.0.0.1 U
Client starts...
Reading from /proc/uptime: 26757.89 212441.82
root@ubuntu:~/Documents/IKN/Exercise_7_c#/UDPclient/UDPclient/bin/Debug#
```

Figur 32 Klient modtager uptime

Figur 33 viser serveren som modtager U fra klienten og sender uptime.

```
root@ubuntu:~/Documents/IKN/Exercise_7_c#/UDPServer/UDPServer/bin/Debug# ./UDPSe
rver.exe
Server starts...
Received client input: U
Sending uptime

```

Figur 33 Server sender uptime

Figur 34 og Figur 35 viser at serveren sender loadavg og klienten modtager dette.

```
root@ubuntu:~/Documents/IKN/Exercise_7_c#/UDPclient/UDPclient/bin/Debug# ./UDPcl
ient.exe 10.0.0.1 l
Client starts...
Reading from /proc/loadavg: 0.01 0.02 0.00 1/472 14482
root@ubuntu:~/Documents/IKN/Exercise_7_c#/UDPclient/UDPclient/bin/Debug#
```

Figur 34 Klient modtager loadavg

```
Sending uptime
Received client input: l
Sending loadavg

```

Figur 35 Server modtager et l og sender loadavg

Det kan ses at serveren er iterativ idét den ikke lukker forbindelsen, men en ny klient kan forbindes.

En sidste test foretages hvor der sendes et invalid input. I dette tilfælde er "k" ikke et valid input, se Figur 36 og Figur 37.

```
root@ubuntu:~/Documents/IKN/Exercise_7_c#/UDPclient/UDPclient/bin/Debug# ./UDPclient.exe 10.0.0.1 k
Client starts...
Bad input. Valid inputs are l (L) or u (U)
root@ubuntu:~/Documents/IKN/Exercise_7_c#/UDPclient/UDPclient/bin/Debug#
```

Figur 36 Klient sender "k" og modtager en fejlmeddelelse

```
Received client input: k
Bad input. Sending error message
```

Figur 37 Server modtager "k" og sender en fejlmeddelelse