IKN Journal 1: Øvelse 3, 6 og 7

Gruppe: 50

I4IKN

05-04-2018

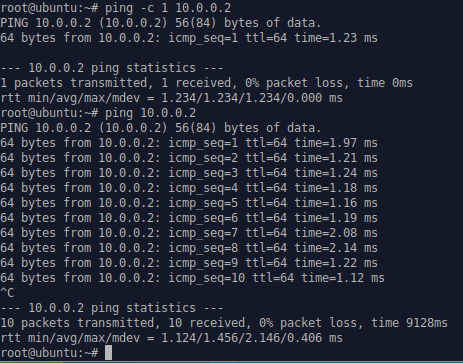
|  |  |
| --- | --- |
| Navn | Studienummer |
| Fatima Kodro | 201609565 |
| Søren Bech | 201604784 |
| Daniel Pat Hansen | 201601915 |

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:

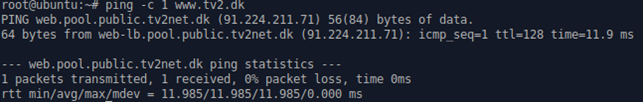
* Min: 1,124
* Avg: 1,456
* Max: 2,146



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.**

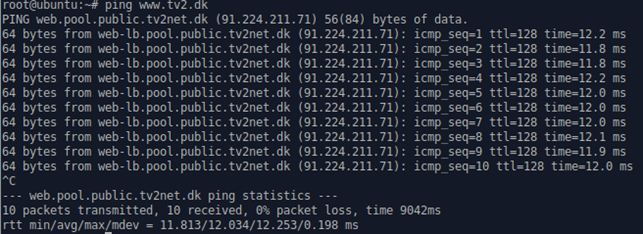
På Figur 2 ses terminalen efter der bliver pinget til [www.tv2.dk](http://www.tv2.dk). Det ses at det tager 11.9 ms for at sende og modtage packets.



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](http://www.tv2.dk). Det ses at der vises 11.813/12.034/12.253 ms

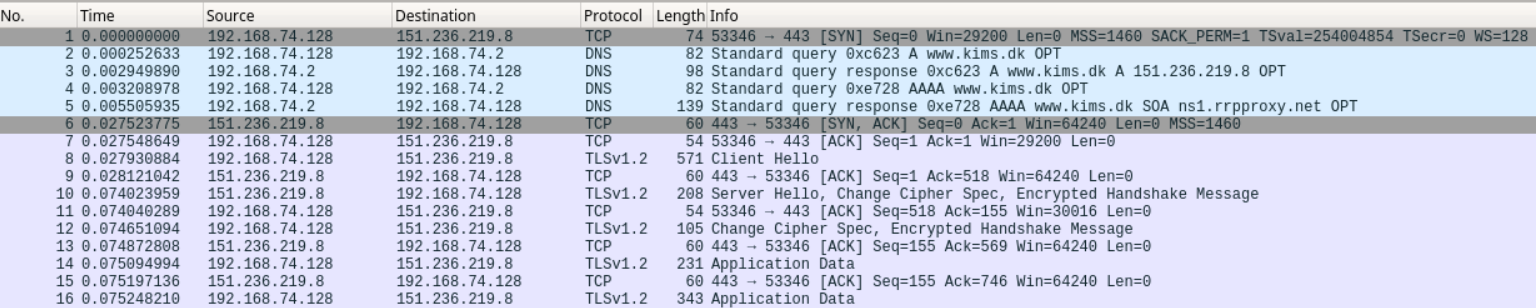


Figur 3: Gennemsnittet af 10 packets sendt til www.tv2.dk

**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](http://www.kims.dk/) som hjemmeside

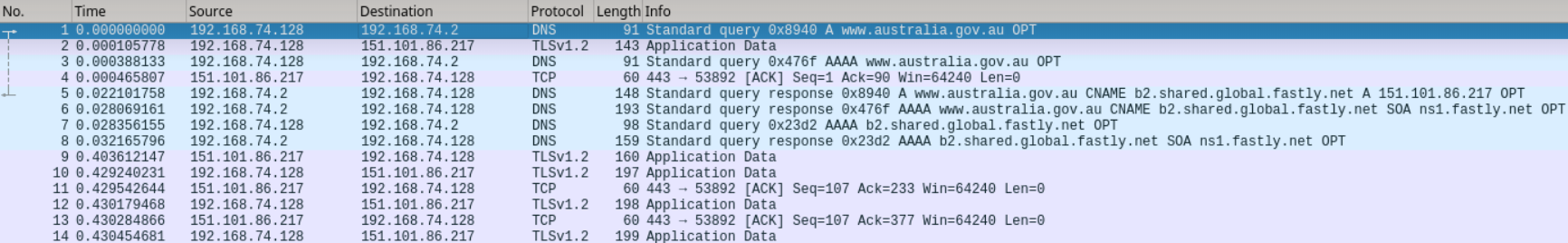
På Figur 4 ses et udsnit af wireshark, hvor den første DNS er en forespørgsel efter kims.dk. Herefter modtages respons fra kims.dk. Det tager ca. 28ms før den første respons kommer fra kims.dk



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 5viser Wireshark, når browseren åbnes og [www.australia.gov.au](http://www.australia.gov.au) forespørges.



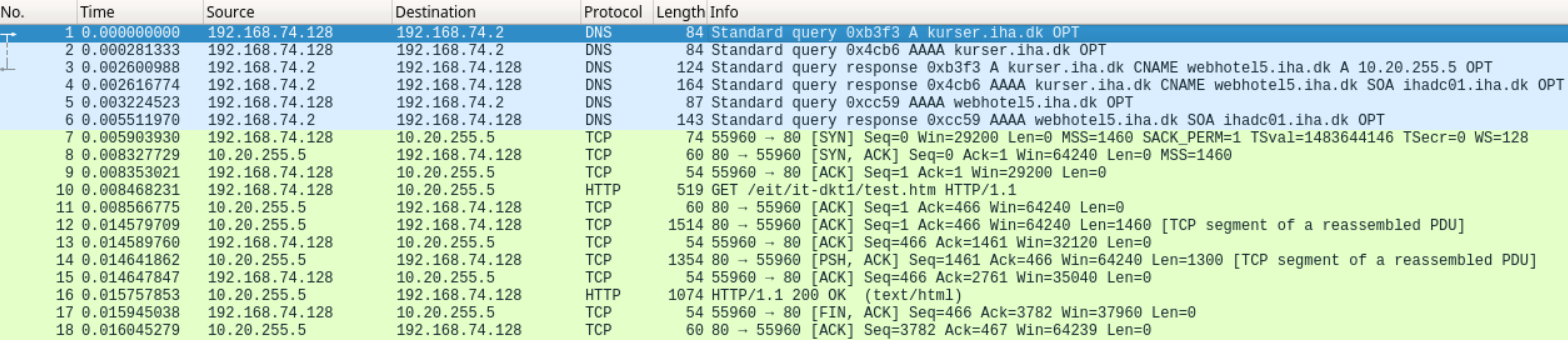
Figur 5: Wireshark billede af når man åbner browseren og går ind på www.australia.gov.au

**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**

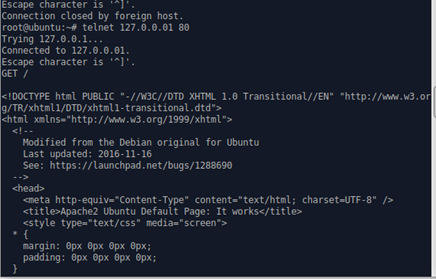
På Figur 6 ses Wireshark når [www.kurser.iha.dk](http://www.kurser.iha.dk) forespørges. Læg særlig opmærksom på nummer 10 på Figur 6. Der bliver hentet noget fra denne side imellem 7 og 17. Det der specifikt bliver hentet er en http side med version 1.1 som er ved nummer 10. Der kommer et TCP three-way 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.



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

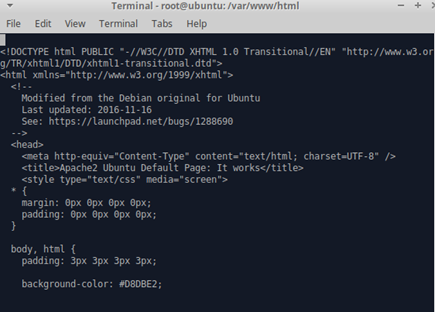
**10. Etabler en LAN-forbindelse mellem en web-serveren (H1) og en web-client (H2) I første omgang anvendes en simpel, telnet baseret web-client i H2.**

Figur 7 viser GET / med 127.0.0.01



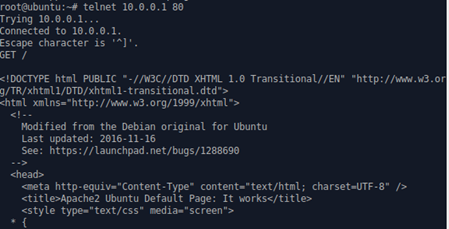
Figur 7: H1 - GET/

Figur 8 viser indholdet af filen



Figur 8: H1 - /var/www/html

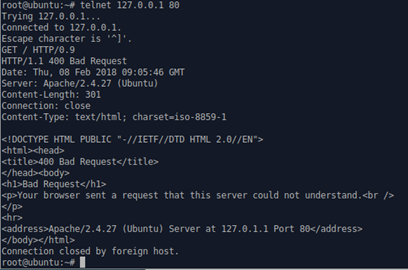
Figur 9 viser GET / med http fra H2 til H1.



Figur 9: H2 – telnet 10.0.0.1 80

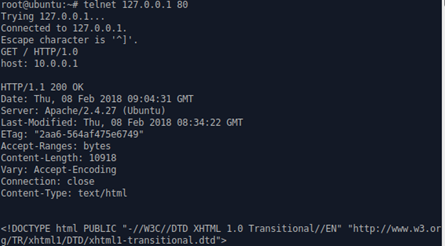
**11. Test protokollerne: HTTP 0.9, HTTP 1.0 og HTTP 1.1.vha. telnet med fokus på oprettelse/nedlukning af TCP-connection og på persistent/non-persistent HTTP-kommunikation vha. HTTP-protokollen (uden/med pipelining).**

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

****

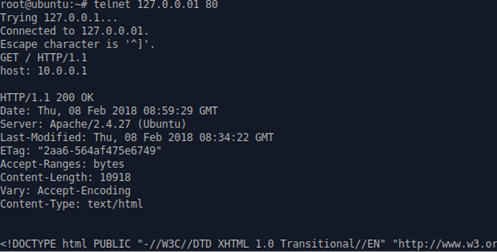
Figur 10: HTTP 0.9

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

****

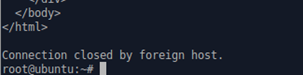
Figur 11: HTTP 1.0

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

****

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).

****

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

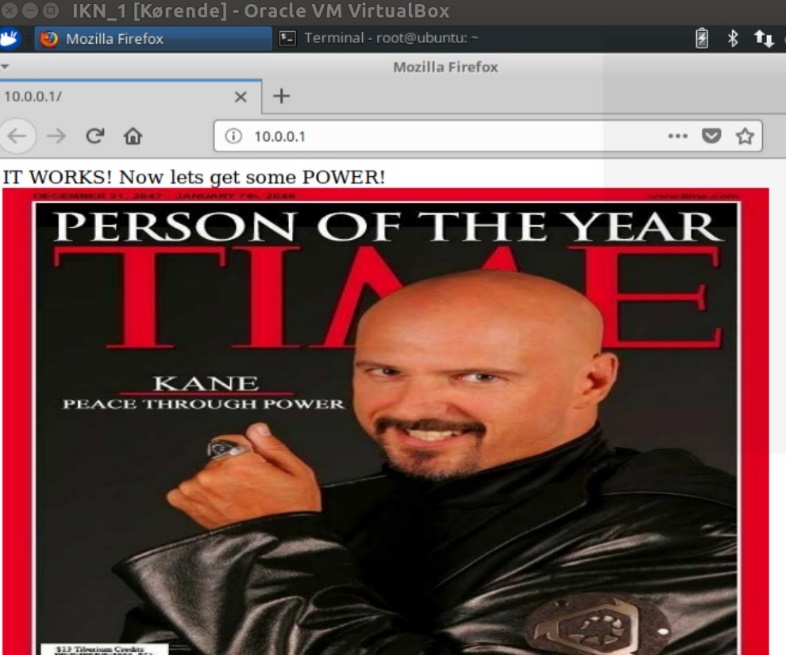
Figur 14viser apache server versionen som er Apache/2.4.27



Figur 14: Fremvisning af apache server version

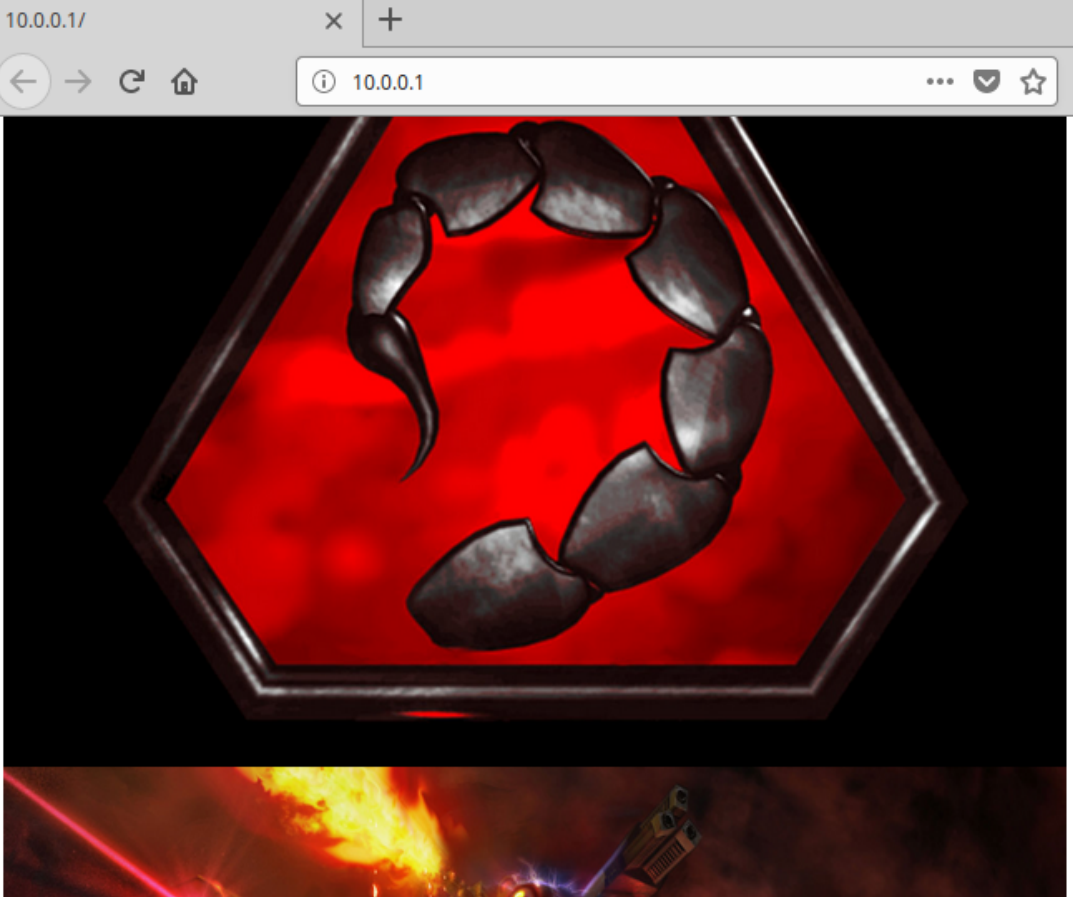
12.   Anvend Firefox web-browser som web-client i H2 sammen med Apache web-server i H1

På Figur 15 fremvises billede (fil) 1.



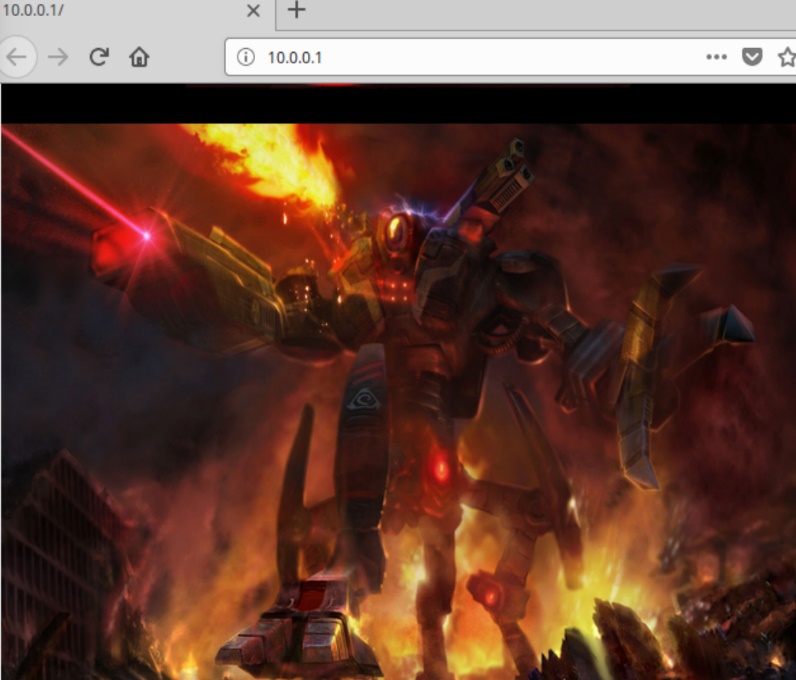
Figur 15: Åbnes igennem terminal med Firefox 10.0.0.1, med lidt magi får man nogle flotte billeder op

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.



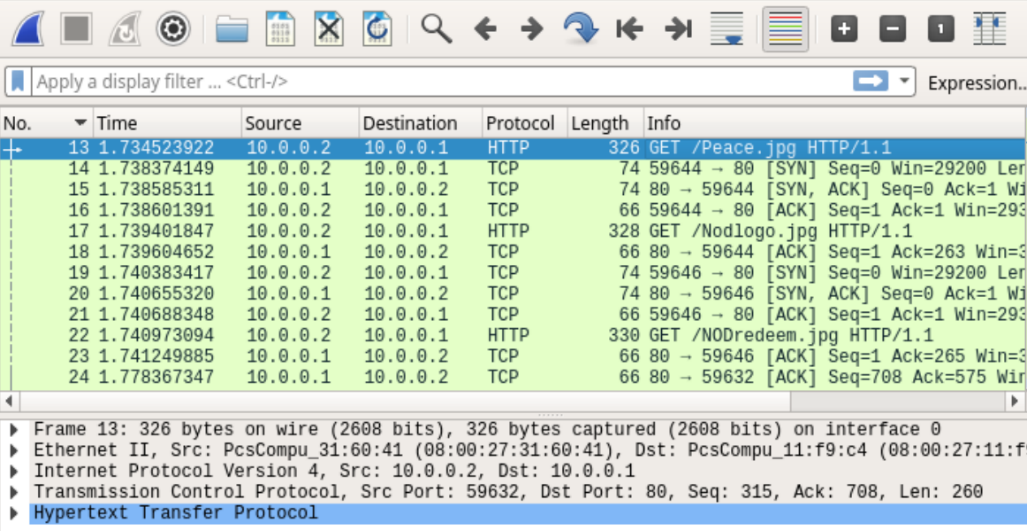
Figur 17: Åbnes igennem terminal med Firefox 10.0.0.1, med lidt magi får man nogle flotte billeder op

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 modificere 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.

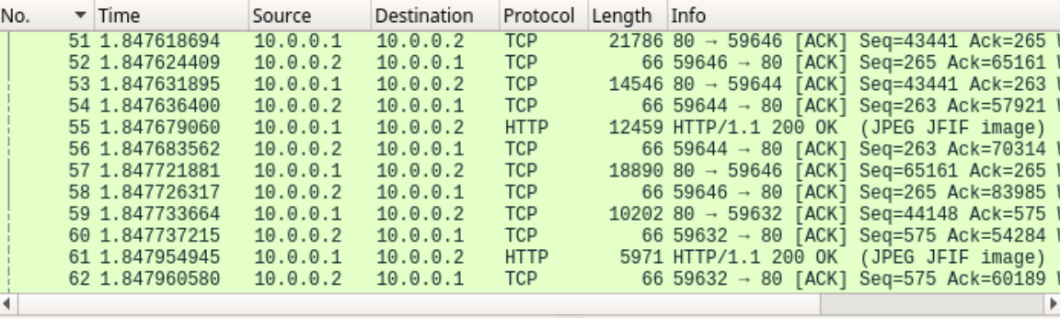
På Figur 20 vises koden til hjemmesiden.

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



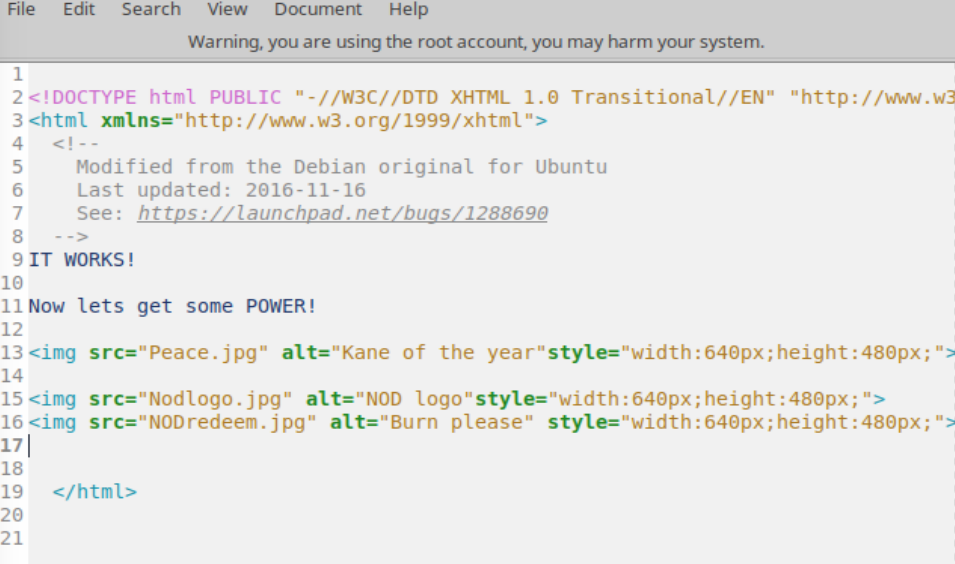
Figur 18: Wireshark billede af at H2 anmoder om billederne med GET /

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



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.



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 stiangivelse til en fil, der eksisterer i serveren.

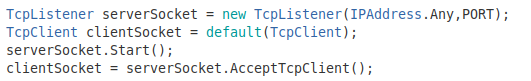
Serveren skal sende filen i segmenter på max. 1000 bytes ad gangen indtil filen er overført, hvis filen altså eksisterer, ellers skal der udskrives en fejlmeddelelse. Uanset om filen eksisterer eller ej, skal serveren kunne håndtere en ny forespørgsel, når den er klar på til at sende en ny fil, dvs.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

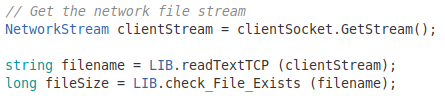
**Server**

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:

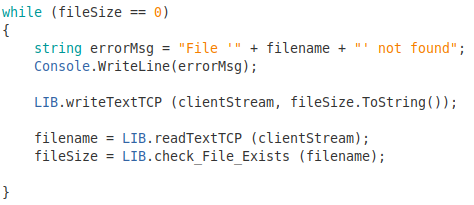


Serveren åbner en forbindelse med en port defineret som 9000, og venter herefter på en klient. Dette kald er blokerende indtil en klient forbindes til serveren.

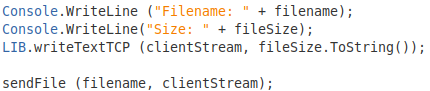
Når en klient er forbundet, bliver der etableret en NetworkStream for klienten for at gøre det muligt at sende og modtage data igennem en stream. Der modtages en streng fra klienten som indeholder en sti. Der tjekkes om filen, som stien referer til, findes. Findes denne fil, vil en filstørrelse returneres,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



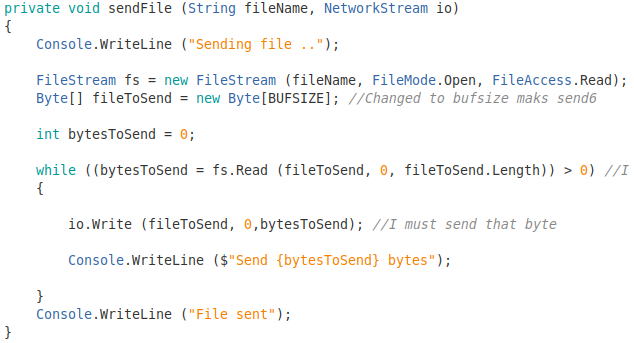
Hvis filen ikke findes, vil serveren udskrive en fejlmeddelelse til konsollen samt en fejlmeddelelse til klienten som spørger om en anden sti. Stien tjekkes indtil en valid fil findes:



Herefter vil en fil blive overført:



hvor sendFile() er defineret som følgende:



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.

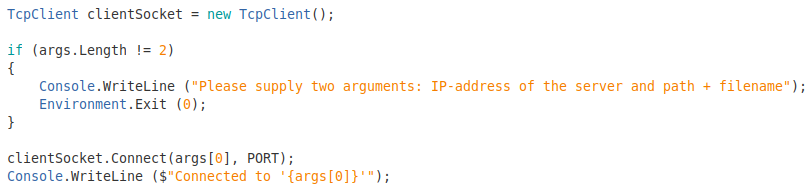
Forbindelsen lukkes herefter så en ny klient kan forbindes:



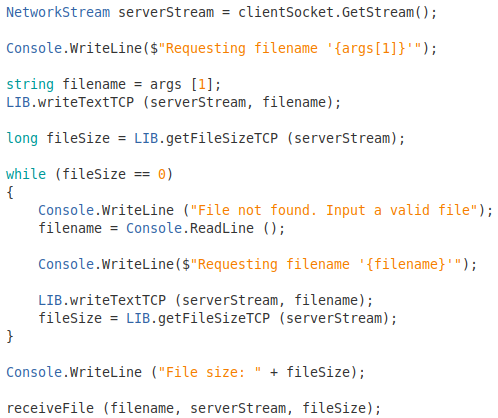
For at gøre serveren iterativ er den blot indsat i en while-løkke, som vil starte en forbindelse og lukke den herefter.

**Client**

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



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:



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

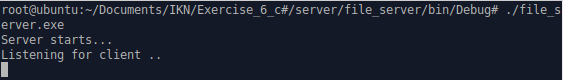


Følgende kode viser der, hvor klienten modtager filen fra serveren. Klienten anmoder om et filnavn, der findes i serveren, og herefter bliver størrelsen af filen registreret. Klienten sørger for at oprette et directory til at opretholde filen. En FileStream oprettes som udskriver til mappen. While-løkken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



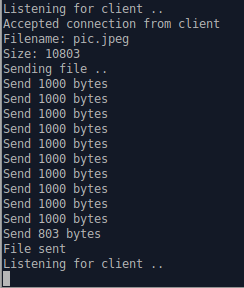
Resultater

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



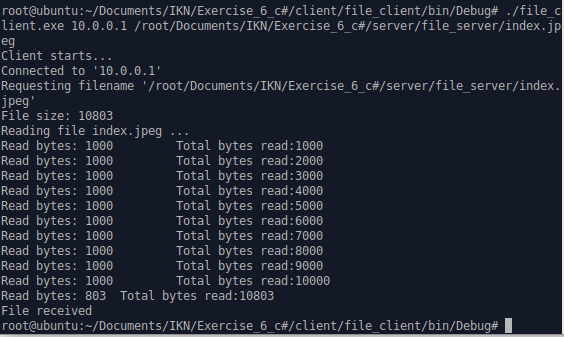
Figur 21 Server venter på en klient

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.



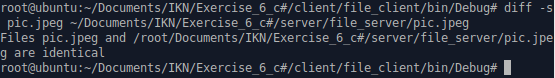
Figur 22 Overførsel af en fil fra serveren

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.



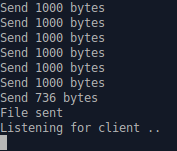
Figur 23 Klient modtager en fil

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.



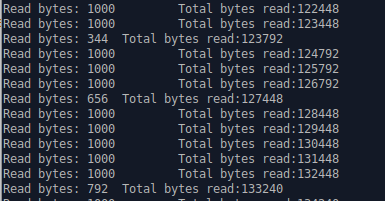
Figur 24 Tjek af at filerne er ens

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.



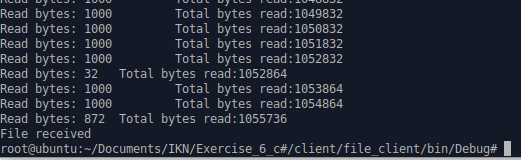
Figur 25 Overførsel af en stor fil (1 mb)

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



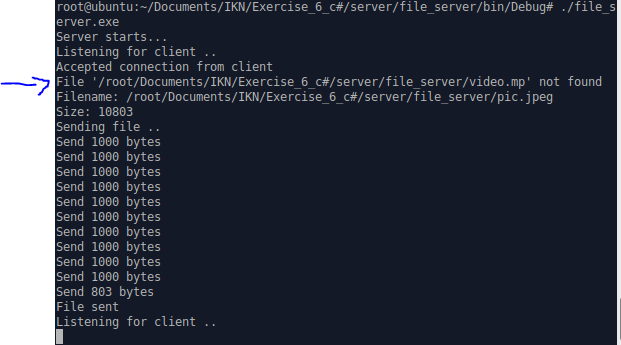
Figur 26 Klient er i gang med at modtage filen

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



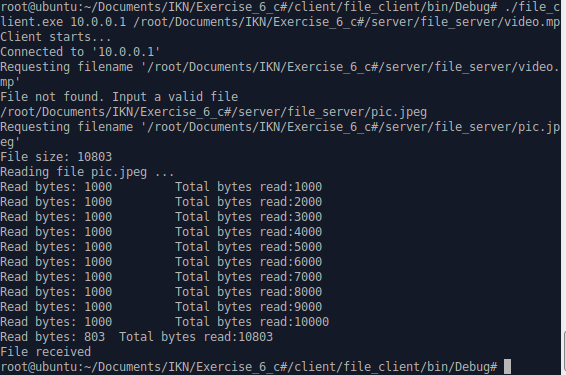
Figur 27 Klient har modtaget filen

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.



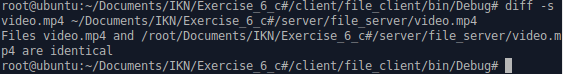
Figur 28 Blå pil peger på en invalid fil

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.



Figur 29 Klient foresprøger 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.



Figur 30 Kvalitetskontrol på filerne

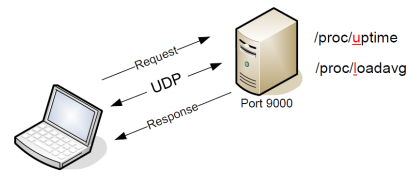
Øvelse 7

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 til, 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.



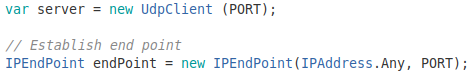
Figur 31 Opstilling

Kodeforklaring

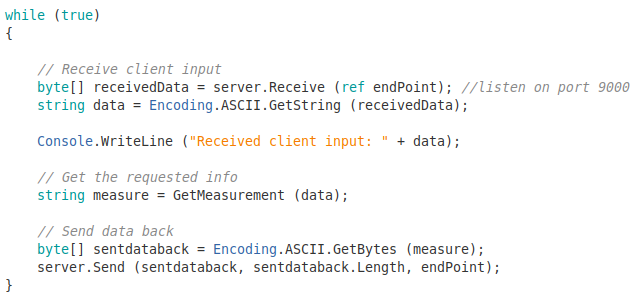
En server og klient er blevet implementeret hvor serveren er iterativ. En UDP server/client forbindelse er meget mere løs end en TCP forbindelse og kræver ikke at man vedligeholder en forbindelse mellem server og klient som der sker i TCP. Serveren skal etablere et end point med et portnummer som en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

**Server**

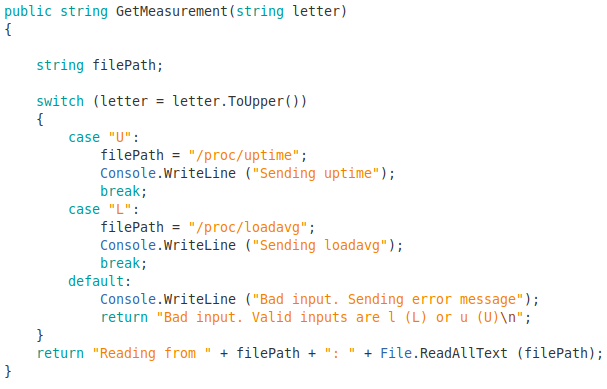
Koden til serveren etablerer en ny UDPClient med et portnummer som anvendes i sit end point.



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):



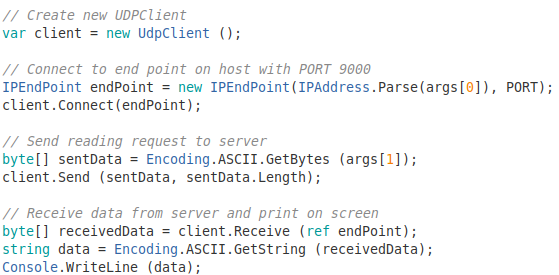
GetMeasurement() vil returnere en string baseret på inputtet:



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.

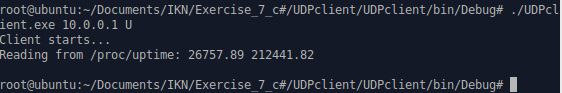
**Client**

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:



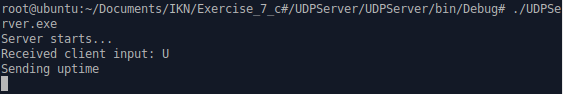
Resultater

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



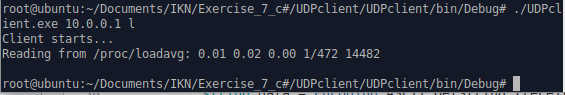
Figur 32 Klient modtager uptime

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



Figur 33 Server sender uptime

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



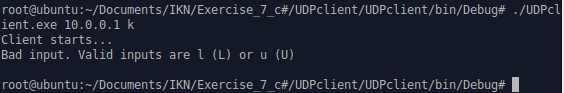
Figur 34 Klient modtager loadavg



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.



Figur 36 Klient sender ”k ”og modtager en fejlmeddelelse



Figur 37 Server modtager ”k” og sender en fejlmeddelelse