Rapport over simuleringsøvelser

Daniel Brasholt s214675 Johannes Bjørn Nielsen s214671
Frederik Bondo Palmqvist s202157

Afleveret 3. December 2021

Indhold

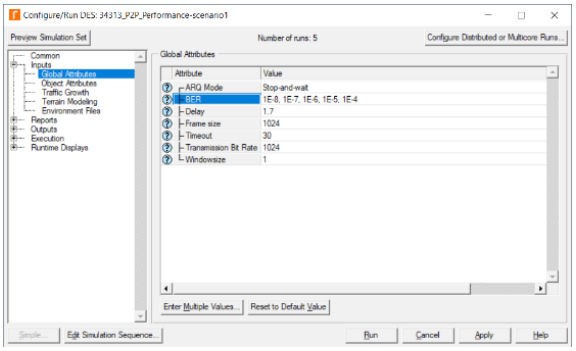
Introduktion

Denne øvelse er delt op i 9 dele. Johannes har stået for del 1-3, Frederik har stået for del 4-6 og Daniel har stået for del 7-9. Vi Vi har i denne rapport lavet en simulering, over for hvilke begrænsninger og muligheder et kommunikations netværk har, i forhold til hvordan det udveksler informationer. Vi vil også kigge på hvordan et kommunikationsnetværk er sat op, og hvad for en indflydelse det har på den virkelige verden. Dette gøres, eftersom simuleringer er kraftfulde værktøjer, når man skal bygge eller fejlfinde i et netværk. Selvom simuleringerne aldrig vil blive helt virkelighedstro, kan man nemt estimere, hvilke protokoller og kommunikationsteknologier, man skal implementere. Dette er for eksempel en fordel for virksomheder, der gerne vil spare så mange penge som muligt på deres netværk, men stadig gerne vil have det bedste, de kan få.
Denne rapport vil indeholde simuleringer af 9 forskellige dele, der hver er vigtig, når man skal bygge et netværk. Først kigges der på forskellene i *flow-control* algoritmer, navnligt *stop-and-wait* og *sliding-window*; disse testes også med bitfejl. Efterfølgende simuleres forskellene mellem hubs, switches og WiFi; hvilke fordele og ulemper, der opstår, når man anvender en af de tre teknologier. Til sidst simuleres forskellene mellem et netværk, der benytter sig at statiske ruter, og netværk, der dynamisk kortlægger netværkets topologi.

Del 1: Punkt-til-punkt med *Stop-and-Wait*

Del 1.1: Gennemsnitlig udnyttelse over tid

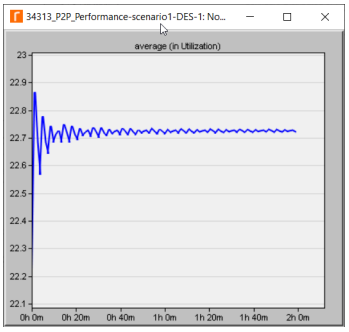
Figur 1: Opbygning af simuleringen



Figur 2: Værdier indsat til første simulering

I denne øvelse har vi et setup med to computere, der er forbundet til hinanden via et direkte kabel som vist på nedenstående figur 1. Derfor skal parametre som routing og IP ikke konfigureres. Til at starte med, skal der laves en simulering af punkt-til-punkt-kommunikation med *stop-and-wait*-algoritmen. Til dette skal følgende parametre indstilles (dette gøres i et vindue vist på figur 2):

- *BER*: Bit Error Rate. Dette er sandsynligheden for, at en bitfejl opstår under kommunikationen. Til denne simulering antager vi fejlfri overførsel.
- *Delay*: Hvor lang tid, der går, før information kommer fra node A til node B. I dette scenarie er denne sat urealistisk højt - et delay på $1,7$ sekunder.
- *Framesize*: Antallet af bits i hver meddelelse. I denne simulering er den sat til 1024.
- *Timeout*: Hvor lang tid Host A venter på en kvittering, før retransmission påbegyndes. Sat til 30 sekunder.
- *Transmission Bit Rate*: Transmissionshastighed målt i $\frac{bit}{sek}$. Til at starte med sat til 1024.
- *Window size*: Vinduesstørrelse til brug for *Sliding Window* flow control. Denne har ingen indvirkning på dette scenarie.



Figur 3: Gennemsnitlig udnyttelse over tid

Målet med simuleringen er at finde udnyttelsen af transmissionen mellem node A og B. Simuleringen viser 2 timers kommunikation og viser gennemsnitsudnyttelsen. På y-aksen er udnyttelsen markeret og x-aksen viser tiden. I starten er gennemsnitsudnyttelsen, som man kan se på figur 3, da der ikke er nok målinger. Den stabiliserer sig dog over tid omkring en gennemsnitsudnyttelse på ca. $22,75\%$. Denne værdi kan estimeres teoretisk ved at bruge formlen for utilization som vist i forelæsning 2:

$$U = \frac{1}{1 + 2 \cdot \frac{t_p}{t_{tx}}} = \frac{1}{1 + 2 \cdot \frac{D \cdot R}{V \cdot L}} \tag{1}$$

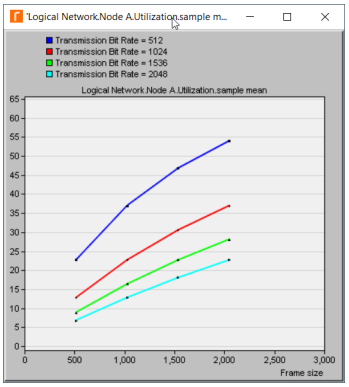
I formel 1 er R bitraten, som er sat til $1024\frac{bit}{sek}$ og L er længden af meddelelsen i bit, her $1024bit$. D og V er i simuleringen ukendte, men $\frac{D}{V}$ er udtryk for forsinkelsen (delay), som er sat til $1,7$ sekunder. Indsættes disse værdier i (1), fås:

$$U = \frac{1}{1 + 2 \cdot 1,7 \cdot \frac{1024}{1024}} \cdot 100\% \approx 22,73\% \tag{2}$$

Denne værdi for udnyttelse stemmer overens med resultatet fra simuleringen, ca. $22,75\%$.

Del 1.2: Udnyttelse som funktion af bitrate og beskedlængde

Det næste, simuleringen skulle vise, var sammenhængen mellem frame size, transmissionshastighed og netværkets udnyttelse. Dette blev gjort ved at køre 16 simuleringer i alt, hvor frame size og transmissionsrate blev varierert i 4 trin hver. Dette gav følgende graf:

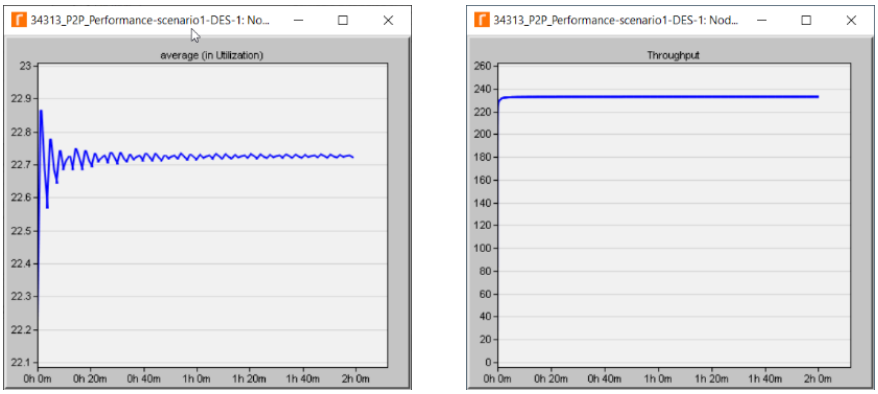


Figur 4: Udnyttelse som funktion af transmissionsrate og frame size

Det kan ses på graferne på figur 4, at der er en sammenhæng mellem resultaterne af simuleringerne og det matematiske udtryk for netværkets udnyttelse (1); jo større, frame size (x-aksen) er, desto større er udnyttelsen i procent. Dette sker, da en stigning i L (frame size) vil gøre brøken $\frac{R}{L}$ mindre. Når denne brøk bliver mindre, bliver det samlede udtryk større, da man så dividerer 1 med et mindre tal. På figuren kan det også ses, at jo højere bitrate, der sendes med, desto lavere bliver udnyttelsen. Igen kan dette forklares ud fra formlen; her er R bitraten. Bliver denne større, bliver brøken $\frac{D \cdot R}{V \cdot L}$ mindre. Dette vil betyde, at man dividerer 1 med en større værdi, hvorfor udnyttelsen så bliver lavere. Dette giver også mening i den virkelige verden; sender man hurtigere data, går der længere tid før man får en `ACK`-pakke i forhold til mængden af tid, man har brugt på at sende pakken.

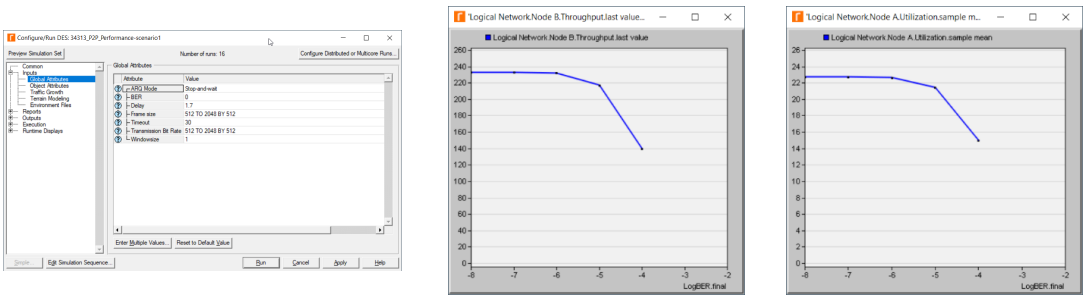
Del 2: Effektivitet af *Stop-and-Wait* ved transmissionsfejl

I del 2 skal udnyttelsen, bitraten (R) og frame size fra del 1 igen bruges. Denne gang skal forbindelsens throughput dog bestemmes. Da netværket har en udnyttelse på $22,73\%$ og en bitrate på $1024\frac{bit}{sek}$, vil throughput teoretisk være givet ved formlen $R \cdot U = 1024\frac{bit}{sek} \cdot 0,2273 \approx 232,76\frac{bit}{sek}$. Fra simuleringen fås følgende graf for udnyttelse og throughput:



Figur 5: Gennemsnitlig udnyttelse og throughput

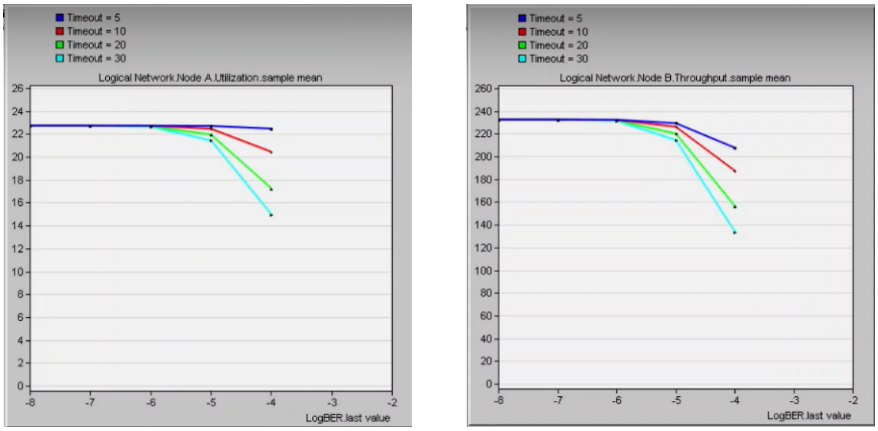
Dette forudsætter dog en perfekt (fejlfri) forbindelse mellem node A og B; dette er da throughput er defineret som mængden af brugbar data, der bliver sendt mellem de to maskiner. Introduceres fejl på netværket, vil der være en vis risiko for, at en given pakke går tabt og at throughput derfor falder, da pakken skal retransmitteres. I simuleringen introduceres dette ved at ændre BER-værdien, der indtil nu har været sat til 0. Hæves denne, vil vi forvente et fald i throughput.



Figur 6: Indstillinger, utilization og output med BER over 0

Simuleringen køres med BER sat til henholdsvist 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5} og 10^{-4}. Disse værdier er sat ud ad x-aksen på figur 6. På y-aksen er henholdsvist throughput og utilization vist.

Det kan ses på resultaterne, at throughput falder, når `BER` hæves. Hvis vi sammenligner throughput med Transmission Bit Rate, ligner de ikke hinanden. Det betyder, at nå der er en `BER` over 0, kan vi ikke bruge den simple formel $U \cdot R = throughput$ for at finde throughput ud fra udnyttelsesprocenten. Dog giver det god mening, hvis man kigger på definitionen af utilization og throughput; utilization betyder blot hvor stor en del af tiden, der sendes over netværket, inklusive de bit, der fejler. Throughput tæller dog kun det, der når frem intakt, hvilket vil sige eksklusivt de bit, der fejler. Dette kan også vises ved at ændre time-out værdierne. Definitionen af time-out værdien er hvor lang tid host A venter på at en kvittering ankommer, før den påbegynder en retransmission. Hvis der da sker en fejl, vil man gerne have en lavere timeout-værdi, så retransmissionen påbegyndes hurtigere; dog skal den ikke sættes for lavt, da der så er en risiko for, at host A sender en pakke to gange, selvom host B har modtaget den første. Simuleringen af ændring af time-out værdien kan ses på nedenstående figurer:

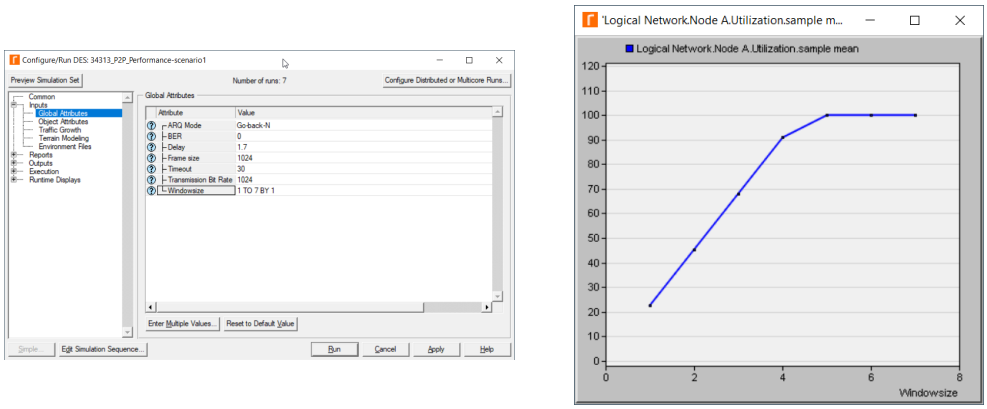


Figur 7: Udnyttelse og throughput som funktion af `BER` og timeout-værdi

Som det kan ses på ovenstående figur 7, falder throughput, hvis timeout-værdien hæves. Den falder også mere end utilization, der stadig er på næsten 100%, selvom der er en `BER` på 10^{-4}. Dette er igen eftersom utilization også inkluderer beskeder med fejl.

Del 3: Effektivitet af *Sliding-Window* flow control

Til del 3 går vi tilbage til en fejlfri linje og en timeout-værdi på 30 sekunder. Der ændres dog også på flow-control algoritmen; i stedet for at kigge på *stop-and-wait*, anvendes *go-back-N* (sliding window). Sliding window bør være mere effektivt, da denne gør det muligt at sende flere bit ad gangen, da vi har et større vindue at sende igennem. Følgende grafer fås, når denne simulering køres:

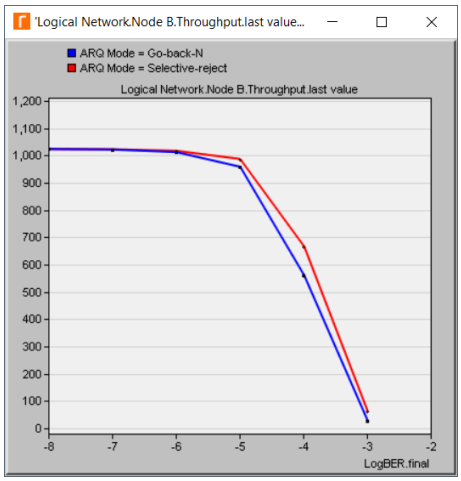


Figur 8: Input og utilization ved forskellige insdtillinger af window size

Det kan ses på figur 8, at udnyttelsen kommer helt op på 100% (perfekt), når vinduet sættes tilstrækkeligt stort. Når ikke udnyttelsen er 100%, er den bestemt ved en lineær funktion af typen $u = a \cdot x + b$. Udnyttelsen starter, når vinduesstørrelsen er på 1, på $22,73\%$. Dette stemmer overens med det, vi fandt i del 1. Formlen på udnyttelsen som funktion af vinduesstørrelsen lyder da:

$$U = 22,73\% \cdot W \leq 100 \quad , \quad W \in \mathbb{N} \tag{3}$$

Til sidst sammenlignes *go-back-N* og *selective-reject*. Dette gøres, da det er relevant at vide, hvor stor forskel disse to algoritmer gør for et netværk, hvis man skulle bygge det i en virksomhed. Det ville naturligvis være bedst at implementere *selective-reject*, hvis man blot skulle optimere throughput. Dog skal man være opmærksom på, at implementeringen af *selective-reject* ville kræve en buffer i kommunikationsenhederne. Uheldigvis for firmaet betyder dette, at implementeringen bliver dyrere. Derfor er det en vurdering af, hvorvidt ændringen i throughput er den ekstra omkostning værd.

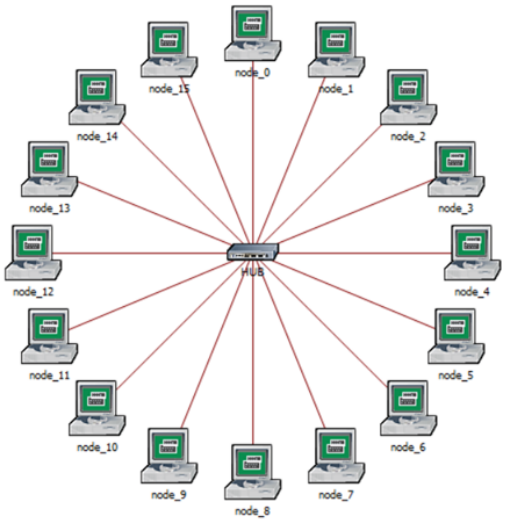


Figur 9: Throughput med *go-back-N* og *selective-reject* som funktion af $logBER$

På figur 9 vises throughput for både *go-back-N* og *sliding-window* som funktion af *Bit Error Rate (BER)*. Det kan ses, at forskellen i throughput på de to algoritmer først er betydelig, hvis man har en *BER* på omkring 10^{-4}. Dette er dog igen en vurdering, den enkelte må lave, når vedkommende skal budgettere opbygningen af et netværk.

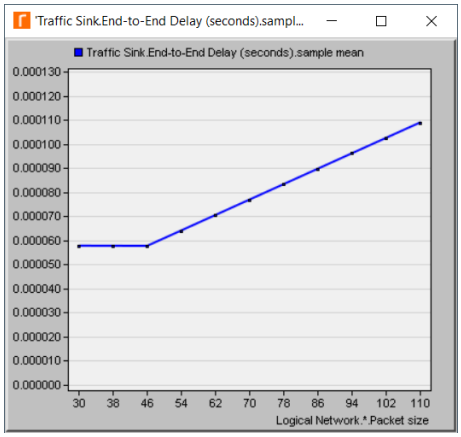
Del 4: *End-to-end* forsinkelse som funktion af pakkestørrelse

I denne simulering skal *end-to-end* forsinkelsen af brugerpakker bestemmes som funktion af pakkestørrelsen. Dette gøres på et kabelnetværk med en central hub, som det kan ses på figur 10



Figur 10: Opbygning af netværk til del 4

Resultaterne fra simuleringen er vist på nedenstående figur 11



Figur 11: *End-to-end* forsinkelse (μs) som funktion af pakkestørrelse (*byte*)

Grafen på figur 11 viser *End-to-End* forsinkelsen, hvilket er den tid, der går fra bruger A sender en pakke data, der går gennem en hub, til den modtages hos bruger B. Dette er målt i mikrosekunder (μs). Det, grafen viser, er hvor lang tid, der går for de forskellige pakkestørrelser at blive sendt. Den mindste pakke, der bliver sendt, er en pakke på 30 byte og den største er har størrelsen 110 byte. Dette gøres i 8 trin. Der er en vandret del på grafen, hvor transmissionstiden på alle pakker mellem 30 byte og 46 byte ligger på ca. $57\mu s$. Den vandrette del af grafen ser sådan ud, fordi der er en minimumsstørrelse på 46 byte, hvilket vil sige, at alle pakker under denne længde bliver fyldt op med polstring indtil pakkerne er store nok til at blive sendt. Dette forklarer også knækket på grafen. Hældningen på det skrå stykke af grafen kan udregnes med følgende formel ud fra to punkter $P(x_1, y_1)$ og $P(x_2, y_2)$:

$$\frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \tag{4}$$

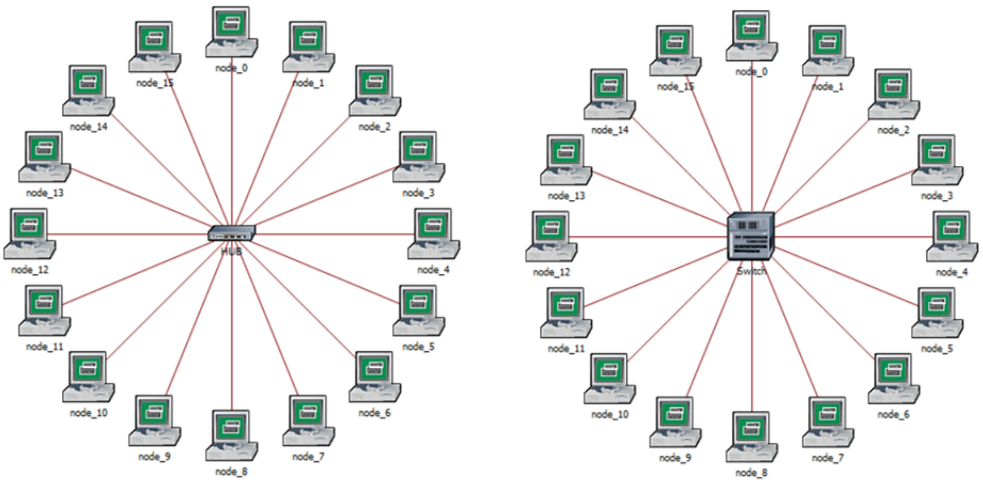
Vælges to punkter fra det skrå stykke på grafen og sætter værdierne ind i (4), fås følgende hældning (efter omregning fra byte til bit):

$$1,000984602 \cdot 10^{-7} \frac{bit}{s}$$

Del 5: *End-to-end* forsinkelse med hub og switch

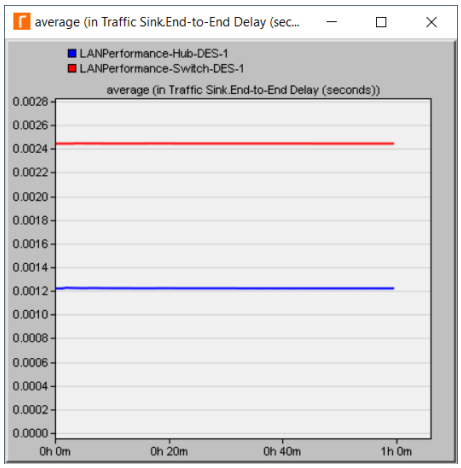
Del 5.1: Hub vs switch

Denne simulering går ud på at sammenligne *end-to-end* forsinkelsen mellem et hub- og et switch-baseret netværk. Strukturen af netværkene kan ses på nedenstående figur 12:



Figur 12: Opbygning af hub- og switchnetværk

Simuleringen gav følgende resultat:

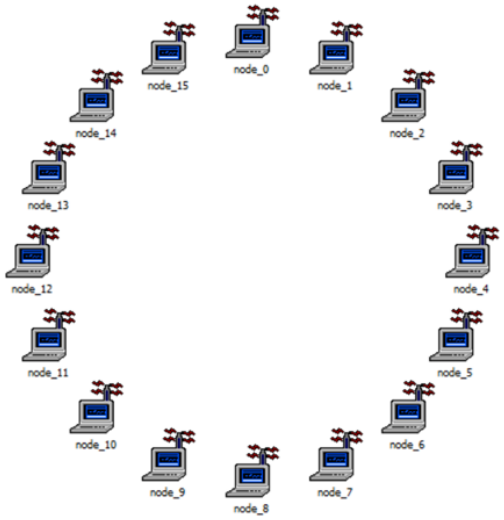


Figur 13: *End-to-end* forsinkelse over tid med hub og switch

Resultatet fra simuleringen, som kan ses på figur 13, viser, at forsinkelsen ved et hub-baseret netværk er lavest. Dette vil sige, at hub-netværket er det hurtigste, hvorimod switchen er langsommest. Denne forskel i hub-netværk og switch-netværk, hvor det hub-baserede netværk er dobbelt så hurtigt, skyldes transmissionstiden. Når hubben modtager en datapakke, sender den pakken videre til modtageren med det samme. Switchen, derimod, skal modtage pakken, læse headeren fra pakken, finde modtageradressen og til sidst sende pakken til den rette bruger. De sender da lige hurtigt pakkerne, men switchen skal bruge noget ekstra tid på at finde modtageren. Dette forklarer den tydelige forskel, som simuleringen viste.

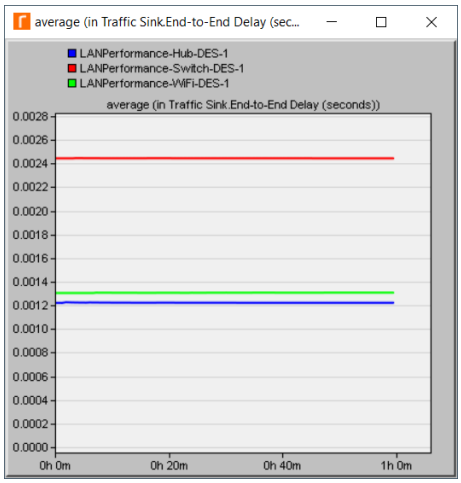
Del 5.2: Hub, switch og WiFi

Den næste simulering skulle vise forskellen mellem *end-to-end* forsinkelsen i kablede og trådløse netværk. For at gøre dette, blev netværksstrukturen af del 5.1 genanvendt, denne gang dog med yderligere en struktur, som kan ses på nedenstående figur 14:



Figur 14: Opbygning af WiFi netværk

Dette gav følgende resultater:

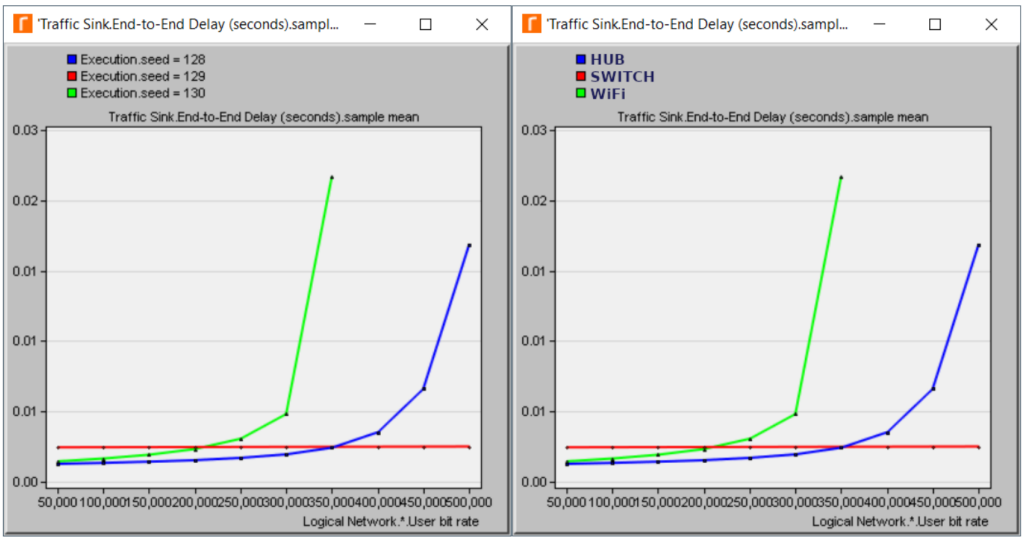


Figur 15: *End-to-end* forsinkelse med hub, switch og WiFi

Det kan ses på figur 15, at WiFi bedst kan sammenlignes med et hub-baseret netværk, da det er næsten lige så hurtigt. WiFi-netværket har en bitrate på $11\frac{bit}{s}$, hvor hub- og switchnetværkene kun har en bitrate på $10\frac{bit}{s}$. Normalt ville dette betyde, at WiFi hurtigere sender pakker; dette er dog ikke tilfældet. Dette skyldes WiFi's MAC-algoritme, der indeholder et DIFS-interval. Dette er ikke et problem i kabelnetværk, hvorfor WiFi bliver langsommere.

Del 5.3: Delay ved belastning

Dette går ud på at undersøge, hvad der sker, når belastningen stiger i de tre forskellige net; hub-, switch- og WiFi-netværk. Simuleringen gav følgende resultat:



Figur 16: *End-to-end* forsinkelse med belastning på netværket

På figur 16 kan man se effekterne af at tilføje belastning på netværket. Belastningen opstår, når der sendes for mange pakker, hvilket gør, at det tager længere tid, når alle pakkerne skal sendes tilbage ud. Dog kan man se, at de 3 netværkstyper blev påvirket forskelligt:

Hub

Grunden til, at det tager længere tid for hubben, er fordi netværket kan detektere kollisioner. Det sker, når hubben modtager data på to porte på samme tid. Når kollisioner sker, stoppes transmissionen, hvorefter afsenderne venter et tilfældigt stykke tid, før de prøver at sende igen. Dermed vil der være mere ventetid, når der sendes mange pakker.

Switch

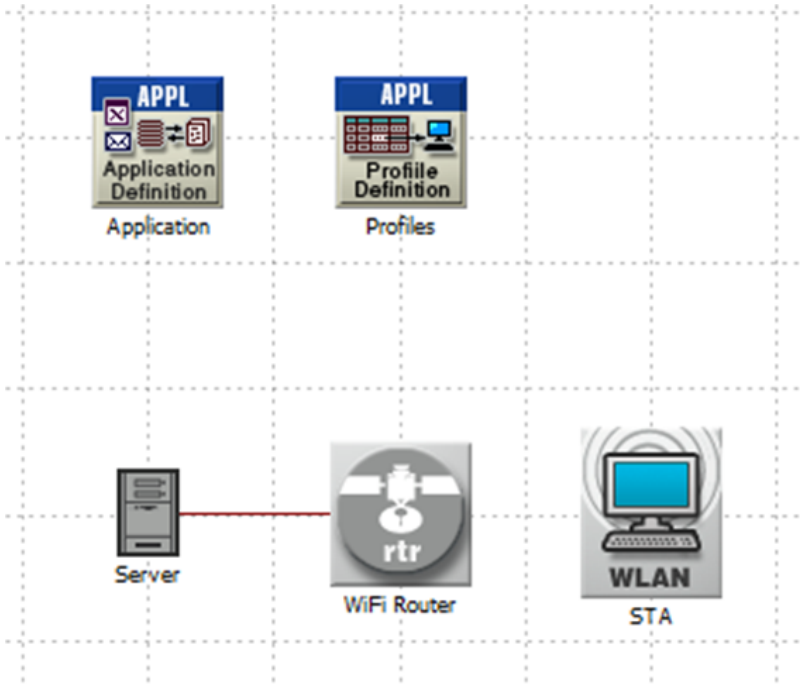
Grafen for switchen er stort set konstant. Grunden til dette er, at switchen i denne simulering er en full-duplex switch. Dette vil sige, at switchen kan håndtere transmissioner både fra klienter til switch og fra switch til klienter på samme tid, da de to retninger kører på forskellige ledere i kablet. Da vil der ikke opstå nogen kollisioner, hvilket gør *end-to-end* forsinkelsen konstant og næsten uafhængig af belastningen på netværket.

WiFi

Som man kan se på grafen, opfører WiFi-netværket ligesom hub-netværket, bortset fra, at der skal færre pakker til for at forsinkelsen bliver højere. Grunden til dette er, at der også kan opstå kollisioner på WiFi-netværket ligesom der kan på hub-netværket; men forskellen er, at kollisionerne varer i længere tid på WiFi-netværket. Når en kollision bliver detekteret på hub-netværket, stoppes transmissionen med det samme og der ventes et tilfældigt stykke tid. På WiFi-netværket bliver man også nødt til at medregne DIFS- og SIFS-intervaller. Altså er WiFi's evne til at håndtere belastning værre end hub-netværkets.

Del 6: Fysisk og aktuel bitrate

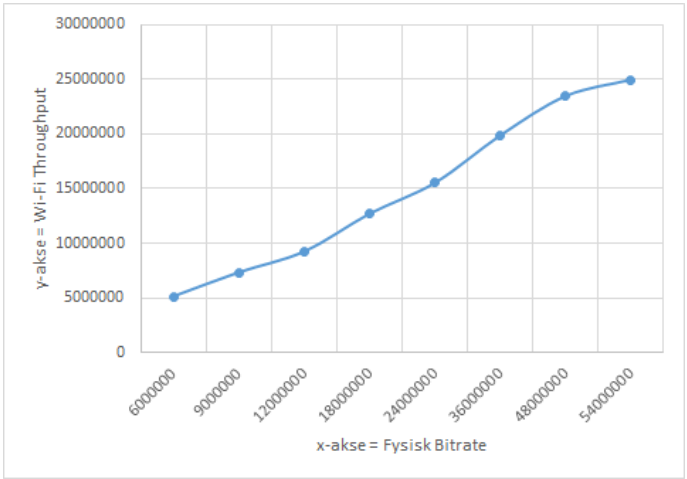
Denne undersøgelse går ud på at simulere indflydelsen af et trådløst lokalnets fysiske bitrate på hvor meget information, der kan sendes. Simuleringen bygger på et netværk bestående af en WiFi Router, som en stationær PC er forbundet til. Routeren har også en kabelforbindelse til en server, som PC'en vil forsøge at uploade en fil til via FTP-protokollen. Netværket er opbygget som vist på figur 17.



Figur 17: Opbygning af netværk til del 6

Del 6.1: WiFi Throughput

Til at starte md, skal det undersøges, hvordan WiFi throughput afhænger af den fysiske bitrate på den trådløse forbindelse. For at gøre dette, er upload-hastigheden målt ved de mulige bitrater i 802.11*g*. Dette har givet følgende resultater:



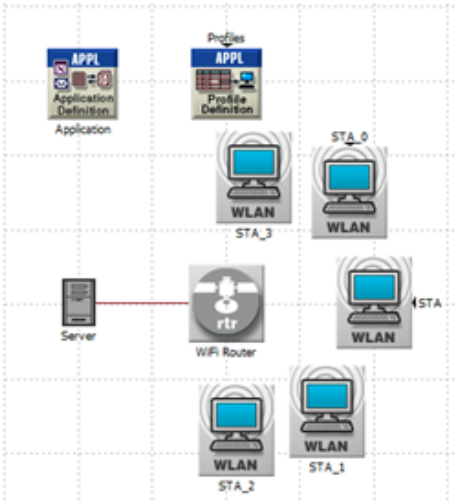
Figur 18: WiFi Throughput ved forskellige fysiske bitrater

Grunden til forskellen mellem den fysiske bitrate og throughput i WiFi, som man kan se på figur 18, er, at DIFS- og SIFS-intervallerne ikke er afhængige af bitraten. Disse er intervaller, hvor der intet bliver sendt. Det vil sige, at mængden af bits, der kunne have været sendt i den tidsramme, er højere, hvis den netværkets fysiske bitrate er højere; dette vil sænke det aktuelle

throughput mere, des højere den fysiske bitrate er. Kurven er ikke lineær, da der ikke er lige stor afstand mellem hvert punkt og at kollisionerne vil lave noget tilfældighed.

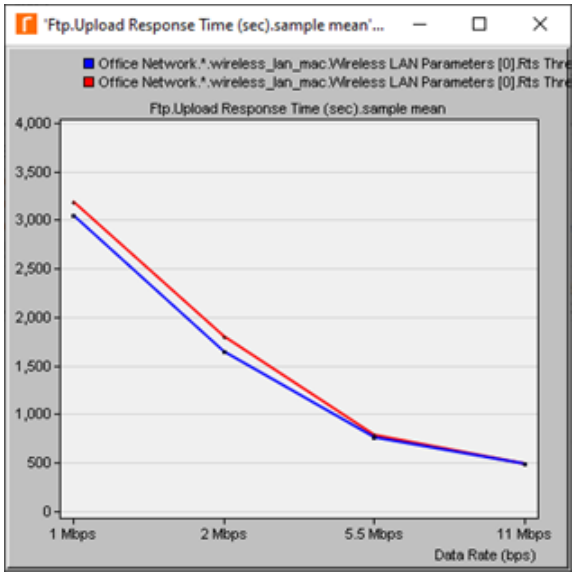
Del 6.2: RTS/CTS Handshake

Til denne øvelse skulle det undersøges, hvordan et RTS/CTS-handshake påvirker Upload Response Time på netværket som funktion af den fysiske bitrate. Til dette blev der brugt en modificeret version af netværket vist på figur 17:



Figur 19: Netværk til del 6.2; fem trådløse stationer vil samtidigt uploade en fil til serveren

Her vil 5 trådløse stationer samtidigt forsøge at uploade en stor fil til serveren. Simuleringen køres både med og uden et RTS/CTS-handshake, hvilket giver følgende resultater:

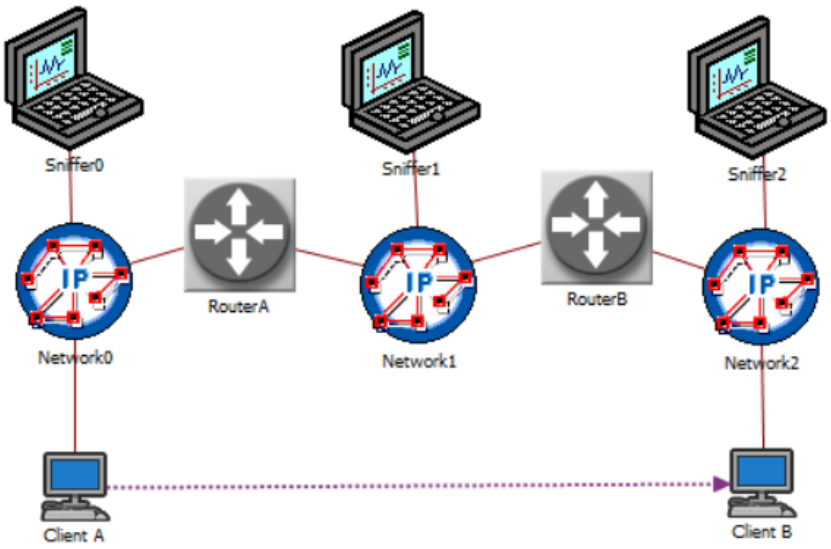


Figur 20: Upload response time som funktion af transmissionrate. Blå med RTS/CTS, rød uden

Det kan ses på figur 20, at det ikke altid gavner at anvende et RTS/CTS-handshake. Grunden til dette er, at et handshake kræver en vis mængde tid, som vil være konstant. Den tid kunne man da have brugt på at transmittere data. Jo højere transmissionsraten er, des kortere vil en eventuel kollision også være; derfor betyder kollisionen mindre for den samlede upload-tid. Ved lavere bitrater får man mere ud af et RTS/CTS-handshake, da en eventuel kollision vil betyde mere.

Del 7: Statisk routing

På nedenstående figur (21) kan netværkskonfigurationen for denne simulering ses. Netværket består af 3 subnet, der er forbundet med 2 routere. På hvert netværk er der en snifferenhed og på både netværk 0 og 2 er der en klient forbundet.



	Name	Status	Operational Status	Address	Subnet Mask	Secondary Address Information	Subinterface Information	Routing Protocol(s)	MTU (bytes)	Protoco MTUs
IF0	IF0	Active	Infer	192.168.0.1	255.255.255.0	Not Used	None	None	Ethernet	(...)
IF1	IF1	Active	Infer	192.168.1.1	255.255.255.0	Not Used	None	None	Ethernet	(...)
IF2	IF2	Active	Infer	Auto Assigned	Auto Assigned	Not Used	None	None	Ethernet	(...)
IF3	IF3	Active	Infer	Auto Assigned	Auto Assigned	Not Used	None	None	Ethernet	(...)

	Name	Status	Operational Status	Address	Subnet Mask	Secondary Address Information	Subinterface Information	Routing Protocol(s)	MTU (bytes)	Protoco MTUs
IF0	IF0	Active	Infer	192.168.1.2	255.255.255.0	Not Used	None	None	Ethernet	(...)
IF1	IF1	Active	Infer	192.168.2.1	255.255.255.0	Not Used	None	None	Ethernet	(...)
IF2	IF2	Active	Infer	Auto Assigned	Auto Assigned	Not Used	None	None	Ethernet	(...)
IF3	IF3	Active	Infer	Auto Assigned	Auto Assigned	Not Used	None	None	Ethernet	(...)

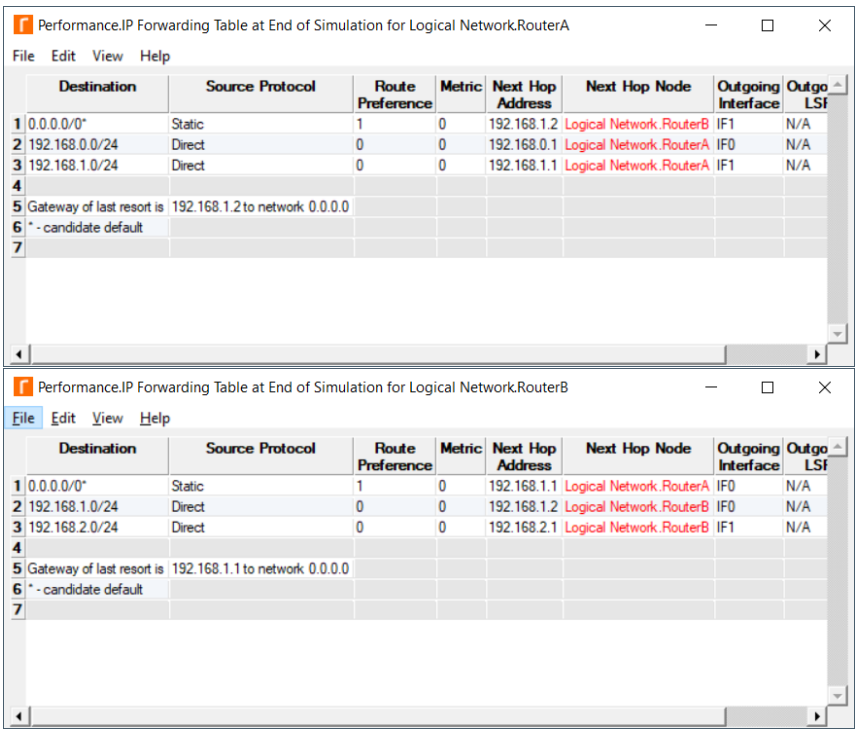
Figur 21: Opstilling af netværk til del 7 med interface-indstillinger til henholdsvis router A og B

Efter 100 sekunder vil Klient A forsøge at sende en ICMP Ping request til klient B. Dette gøres 5 gange med 1 sekund mellem hver Ping Request. Modtager klient B denne, vil den forsøge at besvare med en ICMP Ping Reply. Er netværket sat korrekt op, vil denne nå tilbage til klient A. Dette vil Sniffer 0 kunne observere. Kun denne sniffer vil kunne vidne om routingen er sat korrekt op, da den både kan opsnappe requesten, når den bliver sendt, og reply, når dette har bevæget sig hele vejen tilbage til Klient A.

No.	Time	Source	Destination	Protoc	Len	Info
1	0.000000	0.0.0.0	0.0.0.0	IPv4	20	
2	100.000009	192.168.0.2	192.168.2.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=32 (reply in 3)
3	100.000137	192.168.2.2	192.168.0.2	ICMP	92	Echo (ping) reply id=0x0000, seq=0/0, ttl=30 (request in 2)
4	101.000009	192.168.0.2	192.168.2.2	ICMP	92	Echo (ping) request id=0x0000, seq=1/256, ttl=32 (reply in 5)
5	101.000137	192.168.2.2	192.168.0.2	ICMP	92	Echo (ping) reply id=0x0000, seq=1/256, ttl=30 (request in 4)
6	102.000009	192.168.0.2	192.168.2.2	ICMP	92	Echo (ping) request id=0x0000, seq=2/512, ttl=32 (reply in 7)
7	102.000137	192.168.2.2	192.168.0.2	ICMP	92	Echo (ping) reply id=0x0000, seq=2/512, ttl=30 (request in 6)
8	103.000009	192.168.0.2	192.168.2.2	ICMP	92	Echo (ping) request id=0x0000, seq=3/768, ttl=32 (reply in 9)
9	103.000137	192.168.2.2	192.168.0.2	ICMP	92	Echo (ping) reply id=0x0000, seq=3/768, ttl=30 (request in 8)
10	104.000009	192.168.0.2	192.168.2.2	ICMP	92	Echo (ping) request id=0x0000, seq=4/1024, ttl=32 (reply in 11)
11	104.000137	192.168.2.2	192.168.0.2	ICMP	92	Echo (ping) reply id=0x0000, seq=4/1024, ttl=30 (request in 10)

Figur 22: Udskrift af observerede pakker fra Sniffer0

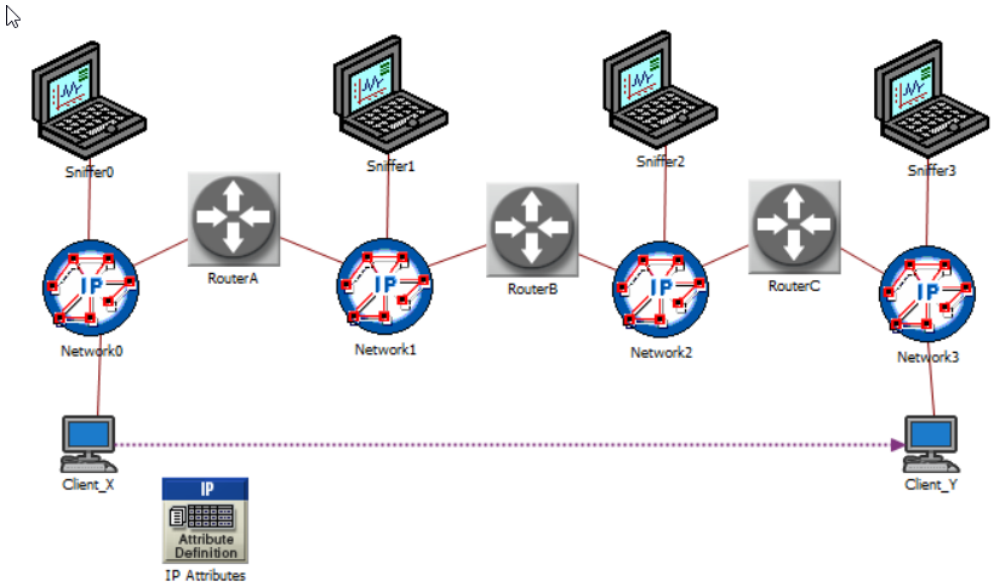
På figur 22 kan et udprint af de pakker, Sniffer0 har opdaget på netværket. Som det kan ses, må netværket være konfigureret korrekt, da snifferen både observerer Ping Requests og Ping Replies. Disse bliver sendt efter 100 sekunder som forventet. Svaret når tilbage til Network0 under et milisekund efter afsendelsen. For at få routerne til at sende trafikken det rigtige sted hen, er følgende routingtabeller anvendt:



Figur 23: Router A og B routing tables genereret af Modeler

Del 8: Statisk routing og routing loops

I dette scenarie er netværket forstørret med et enkelt subnet og en router. Igen er der en sniffer på hvert subnet, som lytter efter den ping-trafik, der bliver sendt fra klient A til klient B. Netværket er altså det samme som i del 7 med et ekstra subnet indsat, hvilket er vist på nedenstående figur 24. For at gøre dette, er routing-tabellerne naturligvis også ændret.



Figur 24: Netværksopbygning til del 8

Del 8.1: Uden routing loop

Til at starte med testes det, om trafikken stadig går igennem, som den skal. Til dette bruges loggen fra Sniffer0 igen, da denne både kan observere ping requests og ping replies, hvis pakkerne løber korrekt gennem netværket. På nedenstående figur 22 kan et udskrift af denne log ses.

No.	Time	Source	Destination	Protocol	Lenç	Info
1	0.000000	0.0.0.0	0.0.0.0	IPv4	20	
2	100.000009	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=32 (reply in 3)
3	100.000196	192.168.3.2	192.168.0.2	ICMP	92	Echo (ping) reply id=0x0000, seq=0/0, ttl=29 (request in 2)
4	101.000009	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=1/256, ttl=32 (reply in 5)
5	101.000196	192.168.3.2	192.168.0.2	ICMP	92	Echo (ping) reply id=0x0000, seq=1/256, ttl=29 (request in 4)
6	102.000009	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=2/512, ttl=32 (reply in 7)
7	102.000196	192.168.3.2	192.168.0.2	ICMP	92	Echo (ping) reply id=0x0000, seq=2/512, ttl=29 (request in 6)
8	103.000009	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=3/768, ttl=32 (reply in 9)
9	103.000196	192.168.3.2	192.168.0.2	ICMP	92	Echo (ping) reply id=0x0000, seq=3/768, ttl=29 (request in 8)
10	104.000009	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=4/1024, ttl=32 (reply in 11)
11	104.000196	192.168.3.2	192.168.0.2	ICMP	92	Echo (ping) reply id=0x0000, seq=4/1024, ttl=29 (request in 10)

Figur 25: Log fra Sniffer0

Det kan ses i udskriftet, at pakkerne løber korrekt gennem netværket og at klient A får svar på de ping requests, den sender. Som i del 7 sker dette også indenfor et milisekund af afsendelsen.

Del 8.2: Med routing loop

Nu introduceres et routing loop til netværket. Dette gøres ved at ændre router B's routing table, således at denne sender pakker, der skal til en klient på netværk 3, tilbage gennem netværk 1 og dermed tilbage til router A. Router A vil modtage pakken og sende den gennem sit højre interface til router B, og et routing loop opstår. Dette reflekteres i loggen fra Sniffer1, som er vist på nedenstående figur.

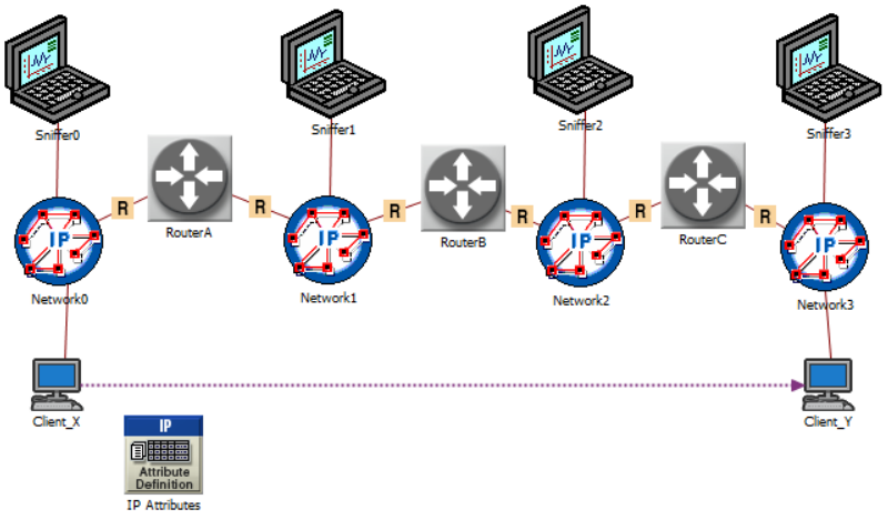
No.	Time	Source	Destination	Protocol	Len	Info
1	0.000000	0.0.0.0	0.0.0.0	IPv4	20	
2	100.000038	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=31 (no response found!)
3	100.000068	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=30 (no response found!)
4	100.000097	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=29 (no response found!)
5	100.000127	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=28 (no response found!)
6	100.000156	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=27 (no response found!)
7	100.000186	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=26 (no response found!)
8	100.000215	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=25 (no response found!)
9	100.000244	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=24 (no response found!)
10	100.000274	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=23 (no response found!)
11	100.000303	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=22 (no response found!)
12	100.000333	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=21 (no response found!)
13	100.000362	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=20 (no response found!)
14	100.000392	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=19 (no response found!)
15	100.000421	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=18 (no response found!)
16	100.000451	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=17 (no response found!)
17	100.000480	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=16 (no response found!)
18	100.000509	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=15 (no response found!)
19	100.000539	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=14 (no response found!)
20	100.000568	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=13 (no response found!)
21	100.000598	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=12 (no response found!)
22	100.000627	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=11 (no response found!)
23	100.000657	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=10 (no response found!)
24	100.000686	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=9 (no response found!)
25	100.000716	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=8 (no response found!)
26	100.000745	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=7 (no response found!)
27	100.000774	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=6 (no response found!)
28	100.000804	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=5 (no response found!)
29	100.000833	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=4 (no response found!)
30	100.000863	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=3 (no response found!)
31	100.000892	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=2 (no response found!)
32	100.000922	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=0/0, ttl=1 (no response found!)
33	101.000038	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=1/256, ttl=31 (no response found!)
34	101.000068	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=1/256, ttl=30 (no response found!)
35	101.000097	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=1/256, ttl=29 (no response found!)
36	101.000127	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=1/256, ttl=28 (no response found!)
37	101.000156	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request id=0x0000, seq=1/256, ttl=27 (no response found!)

Figur 26: Udsnit af log fra sniffer 1. Ping-request-pakkerne hopper mellem router A og B

Det kan ses på figur 26, at ping-request-pakkerne hopper mellem router A og B på netværk 1. Dog gør de ikke dette for altid, da pakkerne også indeholder en Time-To-Live Header, som tælles ned, hver gang en enhed håndterer pakken. Da vil pakkerne blive slettet efter 32 hop. Dette kan ses i loggen som et TTL-felt, der tælles 1 ned for hver pakke, snifferen observerer. Når pakken `ttl=1`, bliver den slettet. Grunden til, at `ttl`-feltet starter på 31, er, at pakken allerede har passeret igennem router A og derfor er blevet talt 1 ned. Den næste pakke, snifferen observerer, er så den næste ping-request, som klient A sender ud på netværket. Eftersom hver ping-request bliver observeret 31 gange, er der $31 \cdot 5 + 1 = 156$ linjer i loggen; da mange af disse er ens, er kun et udsnit vist. Da pakkerne bliver sendt tilbage ind i netværk 1, vil de aldrig nå frem til klient B, hvorfor loggen ikke indeholder ICMP ping replies. Denne simulering viser vigtigheden af, at Time-To-Live eksisterer. Var dette ikke tilfældet, ville netværkene hurtigt blive fyldt op af pakker, der ikke længere ville have en funktion.

Del 9: Dynamisk Routing

Følgende simuleringer udføres for at demonstrere og analysere, hvordan routere dynamisk lærer et netværks topologi og optimerer routingen, så pakkerne kan nå frem med så få hop som muligt. Dette gør det også muligt at sætte et netværk op uden manuelt at konfigurere routing-tabeller. Til denne øvelse genanvendes netværksstrukturen fra del 8, dog med den ændring, at routerne nu kan kommunikere via RIPv2-protokollen, som muliggør kortlægningen af netværket.



Configuration.IP Router Address

File Edit View Help

	Router Name	Interface Name	Is Connected?	IP Address	Secondary Address
1	RouterA	IF0	Yes	192.168.0.1/24	None
2		IF1	Yes	192.168.1.1/24	None
3	RouterB	IF0	Yes	192.168.1.2/24	None
4		IF1	Yes	192.168.2.1/24	None
5	RouterC	IF0	Yes	192.168.2.2/24	None
6		IF1	Yes	192.168.3.1/24	None

Figur 27: Netværksopsætning til del 9

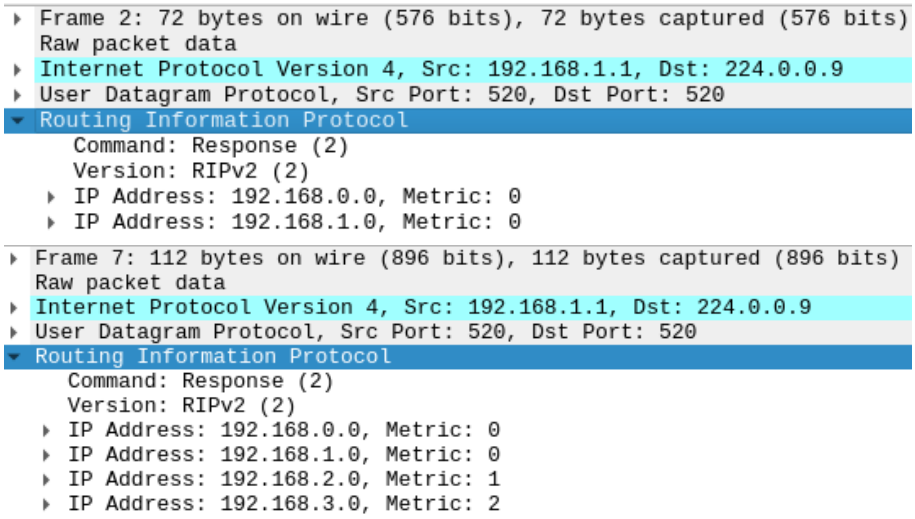
Del 9.1: Dynamisk routing uden netværksfejl

Til at starte med køres en simulering med netværket, som det ser ud på figur 27. Igen bliver aktiviteten logget på hver sniffer. Et udklip af aktiviteten under simuleringen fra Sniffer1 kan her ses:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	0.0.0.0	0.0.0.0	IPv4	20	
2	5.251377	192.168.1.1	224.0.0.9	RIPv2	72	Response
3	6.404455	192.168.1.2	224.0.0.9	RIPv2	92	Response
4	11.381008	192.168.1.1	224.0.0.9	RIPv2	52	Response
5	12.995118	192.168.1.2	224.0.0.9	RIPv2	52	Response
6	16.123665	192.168.1.1	224.0.0.9	RIPv2	52	Response
7	35.251380	192.168.1.1	224.0.0.9	RIPv2	112	Response
8	36.404457	192.168.1.2	224.0.0.9	RIPv2	112	Response
9	65.251380	192.168.1.1	224.0.0.9	RIPv2	112	Response
10	66.404457	192.168.1.2	224.0.0.9	RIPv2	112	Response
11	95.251380	192.168.1.1	224.0.0.9	RIPv2	112	Response
12	96.404457	192.168.1.2	224.0.0.9	RIPv2	112	Response
13	100.000038	192.168.0.2	192.168.3.2	ICMP	92	Echo (ping) request
14	100.000167	192.168.3.2	192.168.0.2	ICMP	92	Echo (ping) reply

Figur 28: RIP-beskeder fra Sniffer1

Det kan ses på figur 28, at routerne til at starte med vil broadcaste pakker cirka hvert 5. sekund, men at de efter ca. 16 sekunder begynder kun at sende statusmeddelelser. Nedenfor er et udklip af indholdet af Frame 2 og Frame 7 vist. Frame 2 er den første besked, router A sender, og Frame 7 er den første periodiske statusmeddelelse, routeren sender:

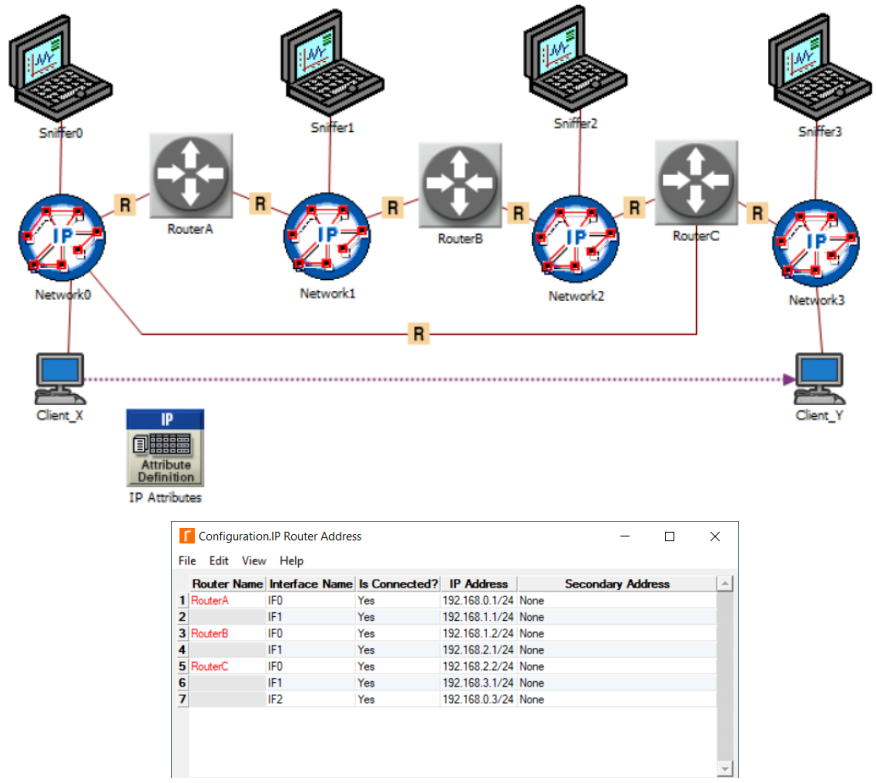


Figur 29: Indhold af RIP-beskeder fra sniffer 1. Udkip fra frame 2 og 7

På figur 29 kan man se, at router A starter med at melde, at den har direkte forbindelse til netværk 0 og netværk 1. Man kan se, at det er router A, da afsenderadressen er `192.168.1.1`, hvilket er adressen på router A's højre interface, hvilket kan ses på figur 27. Man kan se, at router A har direkte forbindelse, da den melder en metric på 0 til disse netværk, hvilket betyder, at routeren ikke skal sende pakken igennem andre hop for at levere den til de netværk. Ved Frame 7 har router A dog fået besked fra router B om, at router B også kan tilgå både netværk 2 og 3; netværk 2 kan for router B tilgås med en metric på 0. Da router A har forbindelse til router B gennem netværk 1, må router A derfor kunne sende til netværk 2 via router B og derfor med en metric på 1. Ligeledes kan router B sende til netværk 3 gennem router C, altså med en metric fra B på 1; dette vil sige, at router A kan sende til netværk 3 med en metric på 2 (de to hop værende router B efterfulgt af router C). Da har routerne formået at kortlægge hele netværket, og A kan selv genkende den korteste rute til klient B. Derfor er det også muligt at sende ping requests og replies gennem netværket, hvilket også fremgår af figur 28.

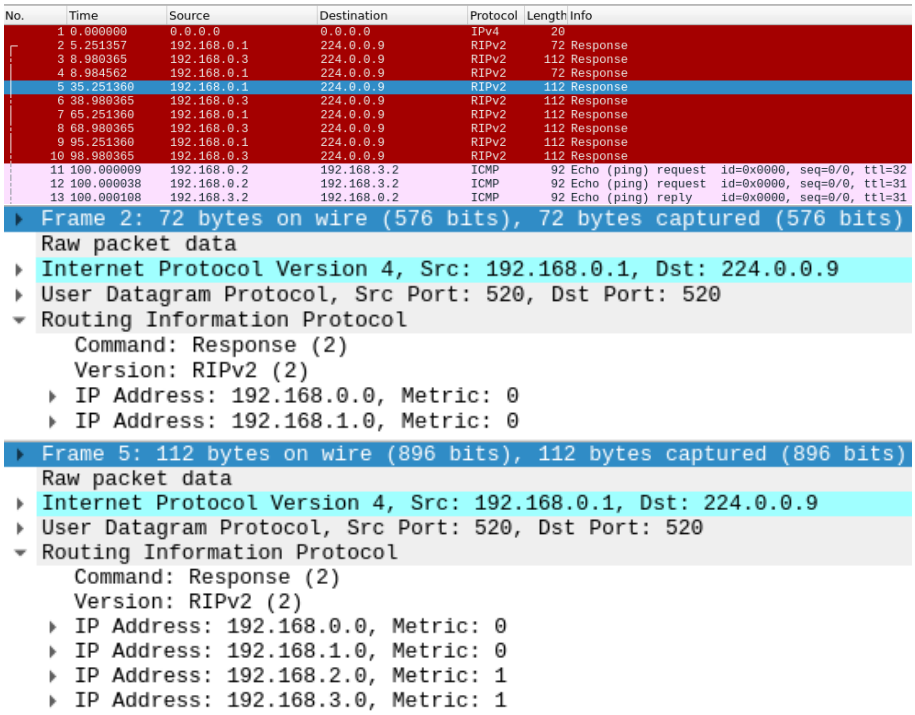
Del 9.2: Dynamisk routing med netværksfejl

Til denne simulering ændres der lidt ved netværket vist på figur 27; der introduceres først en direkte forbindelse mellem `Network0` og router C. Netværket kommer da til at se således ud:



Figur 30: Netværk til simulering 9.2. Ny forbindelse introduceres mellem network0 og router C

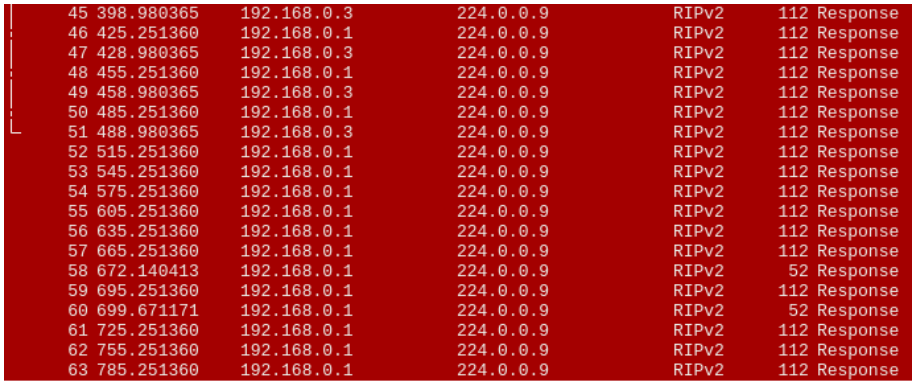
Først udføres en simulering af, hvorvidt netværket opdager den optimale rute i netværket igen. Følgende viser et udklip fra packet capture af Sniffer0:



Figur 31: Indhold af RIP-beskeder fra Sniffer 0. Udklip fra frame 2 og 5

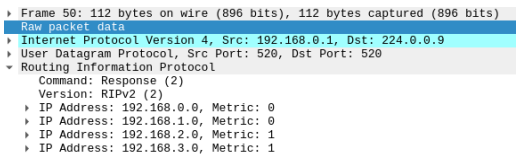
Det fremgår af figur 31, at router A starter med at have viden om netværk 0 og 1. Dette er

igen eftersom routeren har direkte forbindelse til disse to. Ved frame 5, hvilket svarer til router A's første periodiske statusmeddelelse, har routeren dog fået information om, at router C har direkte forbindelse til netværk 2 og 3. Derfor har router A nu en vej til netværk 3 med en metric på kun 1 - som tidligere nævnt, kunne router A, før forbindelsen mellem router C og netværk 0 blev oprettet, kun nå netværk 3 med en metric på 2 (figur 29). Det fremgår også af figur 31, at ping-request pakkerne bliver sendt to gange over netværk 0; dog med en TTL-værdi på henholdsvist 32 og 31. Dette er eftersom sniffer0 fanger pakken, så snart klient X sender den. Router A vil så læse pakken, hvorefter den vil sende pakken til klient y så hurtigt som muligt. Før blev pakken sendt gennem netværk 1, men denne gang er der en direkte forbindelse fra netværk 0 til router C, hvorfor router A vil sende pakken gennem den vej. Dette betyder, at pakken passerer igennem netværk 0 igen. Dog vil klient y kun besvare pakken 1 gang, da klient x kun sender pakken til router A og ikke router C; kun en kopi af pakken vil da nå frem til klient y. Som den sidste simulering testes det, hvordan netværket reagerer, når der introduceres en fejl på en af forbindelserne. Dette gøres ved at lave et planlagt nedbrud af forbindelsen mellem netværk 0 og router C efter $500s$. Dette giver følgende log fra sniffer0:



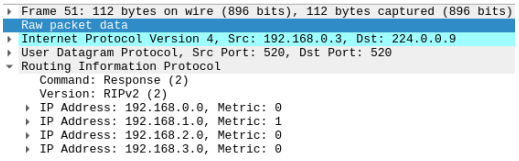
45	398.980365	192.168.0.3	224.0.0.9	RIPv2	112 Response
46	425.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response
47	428.980365	192.168.0.3	224.0.0.9	RIPv2	112 Response
48	455.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response
49	458.980365	192.168.0.3	224.0.0.9	RIPv2	112 Response
50	485.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response
51	488.980365	192.168.0.3	224.0.0.9	RIPv2	112 Response
52	515.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response
53	545.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response
54	575.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response
55	605.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response
56	635.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response
57	665.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response
58	672.140413	192.168.0.1	224.0.0.9	RIPv2	52 Response
59	695.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response
60	699.671171	192.168.0.1	224.0.0.9	RIPv2	52 Response
61	725.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response
62	755.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response
63	785.251360	192.168.0.1	224.0.0.9	RIPv2	112 Response

(a) RIPv2-beskeder omkring tidspunktet, hvor fejlen introduceres



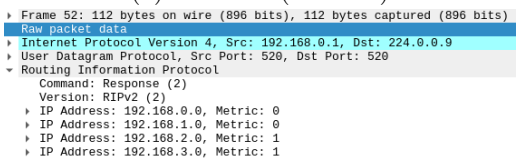
```
Frame 50: 112 bytes on wire (896 bits), 112 bytes captured (896 bits)
Raw packet data
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
    Command: Response (2)
    Version: RIPv2 (2)
  IP Address: 192.168.0.0, Metric: 0
  IP Address: 192.168.1.0, Metric: 0
  IP Address: 192.168.2.0, Metric: 1
  IP Address: 192.168.3.0, Metric: 1
```

(b) Frame 50 ($t = 485s$)



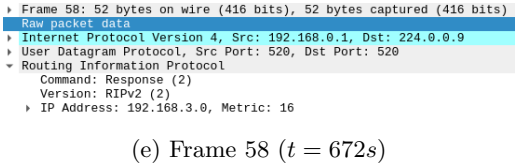
```
Frame 51: 112 bytes on wire (896 bits), 112 bytes captured (896 bits)
Raw packet data
Internet Protocol Version 4, Src: 192.168.0.3, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
    Command: Response (2)
    Version: RIPv2 (2)
  IP Address: 192.168.0.0, Metric: 0
  IP Address: 192.168.1.0, Metric: 1
  IP Address: 192.168.2.0, Metric: 0
  IP Address: 192.168.3.0, Metric: 0
```

(c) Frame 51 ($t = 488s$)



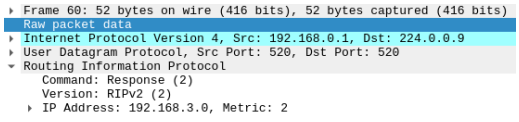
```
Frame 52: 112 bytes on wire (896 bits), 112 bytes captured (896 bits)
Raw packet data
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
    Command: Response (2)
    Version: RIPv2 (2)
  IP Address: 192.168.0.0, Metric: 0
  IP Address: 192.168.1.0, Metric: 0
  IP Address: 192.168.2.0, Metric: 1
  IP Address: 192.168.3.0, Metric: 1
```

(d) Frame 52 ($t = 515s$)



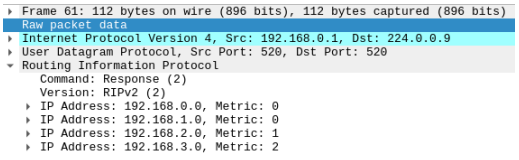
```
Frame 58: 52 bytes on wire (416 bits), 52 bytes captured (416 bits)
Raw packet data
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
    Command: Response (2)
    Version: RIPv2 (2)
  IP Address: 192.168.3.0, Metric: 16
```

(e) Frame 58 ($t = 672s$)



```
Frame 60: 52 bytes on wire (416 bits), 52 bytes captured (416 bits)
Raw packet data
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
    Command: Response (2)
    Version: RIPv2 (2)
  IP Address: 192.168.3.0, Metric: 2
```

(f) Frame 60 ($t = 699s$)



```
Frame 61: 112 bytes on wire (896 bits), 112 bytes captured (896 bits)
Raw packet data
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 224.0.0.9
User Datagram Protocol, Src Port: 520, Dst Port: 520
Routing Information Protocol
    Command: Response (2)
    Version: RIPv2 (2)
  IP Address: 192.168.0.0, Metric: 0
  IP Address: 192.168.1.0, Metric: 0
  IP Address: 192.168.2.0, Metric: 1
  IP Address: 192.168.3.0, Metric: 2
```

(g) Frame 61 ($t = 725s$)

Figur 32: RIP-beskeder omkring tidspunktet, hvor fejlen introduceres samt, indhold i beskederne

Det kan ses på figur 32b og 32c, at RIP-beskederne før fejlen introduceres ved $t = 500s$ er korrekte; både router A og B har styr på de optimale ruter til de 4 subnet. Dog er beskederne efter dette tidspunkt ikke korrekte, som det kan ses på figur 32d; her tror router A fejlagtigt, at den stadig har forbindelse til netværk 3 med en metric på 1. Forbindelsen er dog forsvundet her, hvilket kan ses på manglen af statusmeddelelser fra router C på figur 32a. Indholdet af router A's RIP-beskeder er først helt korrekt igen ved $t = 699s$, hvilket fremgår af figur 32f. Her har A genfundet forbindelsen til netværk 3, denne gang igennem router B. Ved $t = 725s$ sender router A for første gang siden bruddet på forbindelsen en korrekt statusmeddelelse, som det kan ses på figur 32g; denne meddelelse er præcis den samme som den, der vist på figur 29.

Det fremgår af figur 32e, at router A har en rute til netværk 3 med en metric på 16. Dette betyder dog i RIP-protokollen, at der er uendeligt langt derhen. Router A ved altså, at netværket findes, men kender endnu ikke den optimale rute derhen. Denne meddelse kommer cirka $672s - 489s = 183s$ efter router A fik et statusmeddelelse fra router C. Fra $t = 489s$ til $t = 672s$ har router A naturligvis fået opdateringer fra router B om, at der findes en vej hen til netværk 3; dog ignorerer router A dette, da den stadig tror, at den kan komme hen til det med en lavere metric. De cirka $180s$ er bare en værdi, der er høj nok til, at router A ikke fejlagtigt vælger en for lang vej hen til netværk 3. Eftersom RIP-beskederne sendes over UDP, hvilket fremgår af indholdet af beskederne (fx figur 32b), kan det sagtens være, at en statusopdatering eller to går tabt. Dette vil router A dog ikke reagere på, før der er gået ca. $180s$. Der vil router A så melde, at den ikke længere kan nå netværk 3. Næste gang den får en statusopdatering fra router B over netværk 1, vil den så sende en opdatering til netværk 0 om, at den igen kan nå netværk 3, hvilket kan ses på figur 32f - her melder router A korrekt, at den kan nå netværk 3 med en metric på 2. Router A kunne godt, i stedet for at melde en metric på 16, have en buffer over tidligere ruter og statusmeddelelser fra andre routere; dette ville dog være dyrere at implementere, da router A så skulle have mere hukommelse til at gemme oplysningerne i. Denne ekstra omkostning ville ikke give meget i forhold til prisen, hvorfor det ikke er implementeret i RIPv2-protokollen. Det er dog muligt at implementere ved siden af RIPv2 med andre protokoller.

Konklusion

Overordnet har vores resultater vist følgende:
Udnyttelsen af et netværk kommer meget an på, hvilke protokoller og algoritmer, der anvendes, samt hvilket fysisk udstyr, der håndterer kommunikationen. Del 1 til 3 viste, at udnyttelsen afhænger af både transmissionsdelay og transmissionsrate samt om man bruger *sliding-window* eller *stop-and-wait* flow control. Det blev også bestemt, at netværkets udnyttelse afhænger af transmissionsraten i forhold til pakkestørrelsen. Derudover blev det vist, at bitfejl har en betydelig virkning på netværkets throughput. Del 4 og 5 viste forskellen mellem de forskellige fysiske måder, med hvilke man kan forbinde et netværk. Et hub-baseret netværk havde en bedre *end-to-end* forsinkelse end et switch-baseret netværk, men viste sig at være langsommere, hvis netværket blev belastet. WiFi viste sig at have omtrent de samme fordele og ulemper som hub-baserede netværk; selvom de er hurtigere end switch-netværk, har belastning en betydelig rolle, da belastningen kan føre til for eksempel kollisioner.
I del 6 blev forskellen mellem WiFi's fysiske bitrate og egentlige throughput sammenlignet. Selvom WiFi har en høj teoretisk bitrate, kan netværket ikke udnytte denne på grund af blandt andet kollisioner og aftale mellem enheder om, hvilken enhed, der kan sende hvornår. Det viste sig også, at det ikke altid var gunstigt at lave et RTS/CTS-handshake før en pakkeafsendelse, specielt hvis netværket i forvejen havde en høj transmissionrate.
Del 7 til 9 undersøgte forskellen mellem statisk og dynamisk routing. Statisk routing havde den fordel, at man selv kunne bestemme, hvordan pakkerne skulle sendes rundt i netværket. Dog var det meget nemt at introducere et routing loop, hvorfor statisk routing ikke altid er optimal. Desuden kan netværket så ikke tage højde for ændringer. Dynamisk routing, derimod, afsøger netværkets topologi og finder den hurtigste vej - eller i hvert fald den vej med færrest hop. Dynamisk routing havde dog den ulempe, at det tog relativt lang tid for routerne at opdage fejl i forbindelserne, hvilket ville kunne påvirke et virkeligt netværk.
Alt i alt viste undersøgelserne vigtigheden af at lave simuleringer af et netværk. Simuleringerne har muligheden for at vise den optimale brug af fysisk udstyr og protokoller til kommunikation. Har man for eksempel et netværk uden belastning, giver en hub eller WiFi bedst mening at implementere, men så snart belastning introduceres, bør man forbinde netværket med en switch, da denne ikke har et problem med kollisioner.