

DATANETTVERK EKSAMEN

Kandidatnummer:
228



Reliable Transport Protocol (DRTP)

Innholdsfortegnelse

Introduksjon	2
Kodesnippets.....	3
<i>Header:</i>	3
<i>Go-Back-N:</i>	3
<i>Three-Way Handshake:</i>	4
<i>Discard:</i>	4
Discussion.....	5
<i>Throughput sammenlikning:</i>	5
<i>Discarding</i>	6
<i>Simulert pakketap</i>	7
<i>100ms delay, window size 5, 2% packet loss:</i>	7
<i>100ms delay, window size 5, 5% packet loss:</i>	7
<i>100ms delay, window size 5, 50% packet loss:</i>	7
Konklusjon.....	8
Bibliografi	8

Introduksjon

Pålitelig overføring av data er avgjørende i moderne datanettverk for å sikre at informasjon sendes korrekt, selv over forbindelser som kan være ustabile eller upålitelige. I dette prosjektet har jeg implementert og testet en egendefinert pålitelig transportprotokoll (DRTP) for å overføre filer mellom to verter i et simulert nettverksmiljø ved hjelp av Mininet.

Eksperimentene ble gjennomført med en enkel topologi bestående av to verter og en ruter. Målet med prosjektet er å undersøke hvordan ulike nettverksforhold som vindusstørrelse, round-trip time (RTT) og pakketap påvirker gjennomstrømningen og ytelsen til filoverføringsapplikasjonen basert på DRTP.

Det er blant annet testet hvordan økende vindusstørrelse påvirker throughput, hvordan endringer i RTT påvirker effektiviteten, og hvordan protokollen oppfører seg ved pakketap eller dropp. Disse testene demonstrerer hvordan DRTP håndterer retransmisjoner, bekreftelser (ACKs), og avslutning av forbindelsen.

Hensikten med prosjektet er å evaluere hvor sterk og effektiv den egendefinerte protokollen er under ulike forhold, og samtidig få innsikt i hvordan transportprotokoller bygges for å sikre pålitelig kommunikasjon over IP-baserte nettverk.

Kodesnippets

Header:

Headeren i en datapakke er den delen av pakken som inneholder kontrollinformasjon som er nødvendig for at sender og mottaker skal kunne håndtere kommunikasjonen riktig. Headeren består av metadata som forteller hvordan dataene skal behandles, og må ikke forveksles med dataen som skal sendes.

«!HHHH» beskriver headerens struktur med fire 16-bits heltall (seq, ack, flags, window) i nettverksrekkefølge. Dette er viktig for korrekt inn- og utpakking av pakker i Go-Back-N-protokollen.

```
9 # Define constants
10 HEADER_SIZE = 8
11 MAX_DATA_SIZE = 992
12 TIMEOUT = 0.4
13 DEFAULT_PORT = 8080
14
15 # Define flag bits
16 SYN = 0b0010
17 ACK = 0b0001
18 FIN = 0b0100
19 RESET = 0b1000
20
21 def log(message):
22     # Print a message with timestamp
23     print(f"{datetime.now().strftime('%H:%M:%S.%f')} -- {message}")
24
25 def create_packet(seq=0, ack=0, flags=0, window=0, data=b''):
26     # Create a packet by combining header and data
27     header = struct.pack('!HHHH', seq, ack, flags, window)
28     return header + data
29
30 def parse_packet(packet):
31     # Extract components from a packet
32     seq, ack, flags, window = struct.unpack('!HHHH', packet[:HEADER_SIZE])
33     data = packet[HEADER_SIZE:] if len(packet) > HEADER_SIZE else b''
34     return seq, ack, flags, window, data
```

Go-Back-N:

I denne kodebiten ser man implementeringen av Go-Back-N protokollen for å øke påliteligheten i overføringen av filer. Det som skjer er at klienten sender pakker i et vindu (sliding window) og venter på bekreftelser (ACK) fra mottakeren. Når en ACK-pakke mottas, sjekker klienten om flagg og sekvensnummer stemmer (kodelinje 230–234). Hvis ACKen gjelder en pakke i vinduet, fjernes den pakken og alle tidligere pakker fra vinduet (kodelinje 237–240), og vinduets startposisjon oppdateres (linje 243).

Hvis en RTO (retransmission timeout) oppstår (kodelinje 245), betyr det at klienten ikke har mottatt ACK i tide. Da antas det at pakken og eventuelle påfølgende pakker i vinduet er tapt, og de sendes derfor på nytt (kodelinje 248–251). Denne mekanismen sikrer at pakker blir levert i riktig rekkefølge, uten tap, og på en pålitelig måte ved at tapte pakker oppdages og sendes på nytt.

```
219 while base <= len(chunks):
220     # Send packets in window
221     while next_seq <= base + window_size and next_seq <= len(chunks):
222         packet = create_packet(seq=next_seq, data=chunks[next_seq-1])
223         client.sendto(packet, server_addr)
224         window[next_seq] = packet
225         next_seq += 1
226
227     window_str = "{" + ", ".join(str(s) for s in sorted(window.keys())) + "}"
228     log(f"packet with seq = {next_seq} is sent, sliding window = {window_str}")
229
230 # Wait for ACK or timeout
231 try:
232     data, _ = client.recvfrom(HEADER_SIZE + MAX_DATA_SIZE)
233     seq, ack, flags, _ = parse_packet(data)
234
235     if flags & ACK:
236         log(f"ACK for packet = {seq} is received")
237
238         # Remove acknowledged packets
239         for s in list(window.keys()):
240             if s <= seq:
241                 del window[s]
242
243         # Advance base
244         base = seq + 1
245
246 except socket.timeout:
247     # Timeout - retransmit all packets in window
248     log("RTO occurred")
249     for seq in sorted(window.keys()):
250         log(f"retransmitting packet with seq = {seq}")
251         client.sendto(window[seq], server_addr)
```

Three-Way Handshake:

Denne koden viser hvordan klienten etablerer en forbindelse med serveren. Først sendes en SYN-pakke (linje 157-159). Deretter venter klienten i en løkke på SYN-ACK fra serveren (linje 165-173). Når SYN-ACK mottas, tilpasses vindusstørrelsen til det minste av klientens og serverens vindu, og en ACK sendes tilbake for å fullføre håndtrykket (linje 179-181). Hvis klienten ikke mottar svar innen 5 sekunder, sendes SYN-pakken på nytt (linje 188). Prosessen er fullført når `syn_ack_received` settes til True og løkken avsluttes.

```
154 # Send SYN
155 syn = create_packet(flags=SYN, window=window_size)
156 client.sendto(syn, server_addr)
157 print("SYN packet is sent")
158
159 # Wait for SYN-ACK
160 syn_ack_received = False
161 start_time = time.time()
162
163 while time.time() - start_time < 5: # 5-second timeout for connection
164     try:
165         data, addr = client.recvfrom(HEADER_SIZE + MAX_DATA_SIZE)
166         server_addr = addr # Update server_addr
167         _, _, flags, server_window, _ = parse_packet(data)
168
169         # Check if SYN-ACK
170         if (flags & SYN) and (flags & ACK):
171             print("SYN-ACK packet is received")
172
173             # Negotiate window size
174             window_size = min(window_size, server_window)
175
176             # Send ACK
177             ack = create_packet(flags=ACK)
178             client.sendto(ack, server_addr)
179             print("ACK packet is sent")
180             print("Connection established")
181             connection_time = time.time() - connection_start_time
182             print(f"Connection establishment took {connection_time:.2f} seconds")
183
184             syn_ack_received = True
185             break
186
187     except socket.timeout:
188         log("Timeout waiting for SYN-ACK, retrying")
189         client.sendto(syn, server_addr)
```

Discard:

Denne koden viser hvordan serveren håndterer pakketap i overføringen. Hvis en pakke skal forkastes for testing (linje 116-119), markeres dette med `has_discarded = True` og pakken ignoreres med `continue`. For pakker som mottas i riktig rekkefølge (linje 122-123), legges dataen til i bufferen, bytes telles opp, og en bekreftelse (ACK) sendes tilbake (linje 128-130). Pakker som kommer i feil rekkefølge logges bare som "out-of-order" (linje 133-134). Dette simulerer pakketap i et nettverk og tester protokollens evne til å håndtere feil.

```
114 if len(data) > 0:
115     # Test case for packet loss
116     if discard_seq and seq == discard_seq and not has_discarded:
117         has_discarded = True
118         log(f"out-of-order packet {seq} is received")
119         continue
120
121     # Check if packet is in order
122     if seq == expected_seq:
123         log(f"packet {seq} is received")
124         received_data.extend(data)
125         bytes_received += len(data)
126
127         # Send ACK
128         ack_packet = create_packet(seq=seq, flags=ACK, window=15)
129         server.sendto(ack_packet, client_addr)
130         log(f"sending ack for the received {seq}")
131         expected_seq += 1
132     else:
133         log(f"out-of-order packet {seq} is received")
```

Discussion

Throughput sammenlikning:

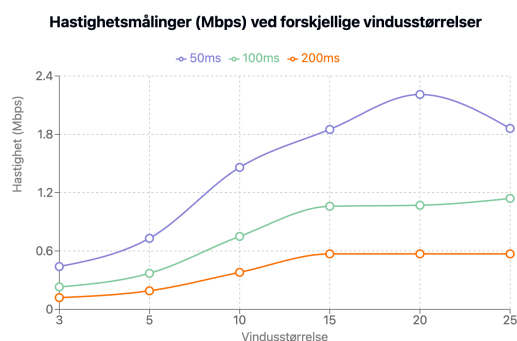
Som tabellen viser, øker gjennomstrømningen jevnt med økende vindusstørrelse opp til et visst punkt. For alle forsinkelsesnivåer ser vi en betydelig ytelsesøkning når vindusstørrelsen økes fra 3 til 20. Dette skyldes at større vinduer tillater flere pakker å være i transitt samtidig, som utnytter nettverkets kapasitet bedre.

Ved 50ms forsinkelse nås optimal gjennomstrømning på 2.21 Mbps ved vindusstørrelse 20, før den reduseres til 1.86 Mbps ved vindusstørrelse 25. Dette representerer et fall på ca. 16%. Interessant nok observeres ikke samme mønster ved 100ms forsinkelse, der ytelsen fortsetter å øke svakt til 1.14 Mbps ved vindusstørrelse 25.

Reduksjonen i ytelse ved vindusstørrelse 25 skyldes økt sannsynlighet for pakketap når for mange pakker sendes samtidig. I Go-Back-N-protokollen medfører ett pakketap retransmisjon av alle etterfølgende ukvitterte pakker, noe som reduserer effektiv gjennomstrømning. Dette er mindre fremtredende ved høyere forsinkelser, da færre pakker rekker å sendes før RTO oppstår.

På 200ms forsinkelse kan man se at protokollen har nådd sitt metningspunkt på vindusstørrelse 15, og at gjennomstrømningen derfor ikke øker videre ved vindusstørrelser 20 og 25. Dette skjer fordi den høye forsinkelsen begrenser hvor mange pakker som kan bekreftes før timeout inntreffer. Med en RTT på ca. 400ms vil pakker sendt utover en viss grense bare øke sannsynligheten for retransmisjoner uten å forbedre ytelsen. Dette viser hvordan forsinkelse, ikke bare vindusstørrelse, setter en øvre grense for praktisk oppnåelig gjennomstrømning.

	Window size: 3	Window size: 5	Window size: 10	Window size: 15	Window size: 20	Window size: 25
50ms	0.44 Mbps	0.73 Mbps	1.46 Mbps	1.85 Mbps	2.21 Mbps	1.86 Mbps
100ms	0.23 Mbps	0.37 Mbps	0.75 Mbps	1.06 Mbps	1.07 Mbps	1.14 Mbps
200ms	0.12 Mbps	0.19 Mbps	0.38 Mbps	0.57 Mbps	0.57 Mbps	0.57 Mbps



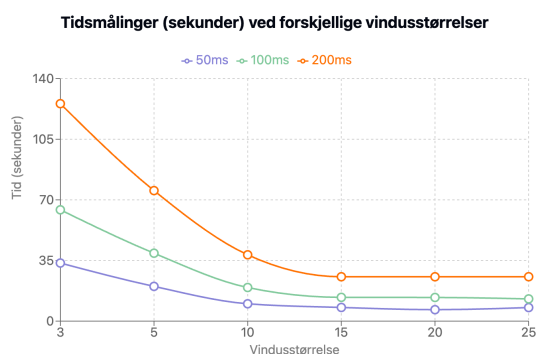
Her er forskjellene på datagjennomstrømming vist i et diagram som bedre visualiserer påvirkningen vindusstørrelse har på datagjennomstrømmingen.

Neste tabell viser forskjellen i tiden overføringene tar. Vi ser en klar reduksjon i overføringstid når vindusstørrelsen øker fra 3 til 20, som korresponderer med den tidligere observerte økningen i gjennomstrømning.

Ved 50ms forsinkelse er beste overføringstid på 6.61 sekunder ved vindusstørrelse 20, mens tiden øker til 7.84 sekunder ved vindusstørrelse 25, noe som bekrefter redusert effektivitet ved for store vinduer. Ved 100ms forsinkelse ser vi en gradvis reduksjon i overføringstid gjennom hele spekteret av vindusstørrelser.

På 200ms forsinkelse flater overføringstiden ut rundt 25.6 sekunder for vindusstørrelser 15, 20 og 25, som da styrker konklusjonen om at vi har nådd et metningspunkt der nettverkets høye forsinkelse setter en grense for hvor raskt overføringen kan fullføres, uavhengig av økning i vindusstørrelsen.

	Window size: 3	Window size: 5	Window size: 10	Window size: 15	Window size: 20	Window size: 25
50ms	33.53 sek	20.06 sek	10.01 sek	7.89 sek	6.61 sek	7.84 sek
100ms	64.26 sek	39.21 sek	19.41 sek	13.72 sek	13.66 sek	12.82 sek
200ms	125.53 sek	75.35 sek	38.28 sek	25.61 sek	25.61 sek	25.63 sek



Her er tidsforskjellene vist i diagram for å bedre visualisere påvirkningen vindusstørrelse har på tiden det tar å overføre.

Discarding

På h2 ser man at pakkene frem til 1799 mottas og bekreftes slik de skal. På pakke 1800 har det skjedd et pakketap ettersom koden ble kjørt med -d 1800. De pakkene som er i samme vindu etter 1800 blir identifisert som «out-of-order» og sender derfor ikke noe ACK. Da starter retransmisjonen som følger av Go-Back-N protokollen.

På h1 ser vi at etter å ha sendt pakke 1803, oppstår en timeout ("RTO occurred"). Dette utløser retransmisjon av alle ubekreftede pakker i vinduet, fra 1800 til 1803. Etter retransmisjonen blir pakkene mottatt i riktig rekkefølge av serveren, som nå sender ACK for dem, og dataoverføringen fortsetter normalt.

Dette demonstrerer effektivt hvordan Go-Back-N protokollen håndterer pakketap ved å sende alle ubekreftede pakker på nytt når en timeout oppstår, og hvordan serveren avviser pakker som kommer ut av rekkefølge.

```
"Node: h2"
06:51:31.787100 -- packet 1796 is received
06:51:31.787670 -- sending ack for the received 1796
06:51:31.787772 -- packet 1797 is received
06:51:31.787864 -- sending ack for the received 1797
06:51:31.787906 -- packet 1798 is received
06:51:31.787956 -- sending ack for the received 1798
06:51:31.787996 -- packet 1799 is received
06:51:31.788041 -- sending ack for the received 1799
06:51:31.788081 -- out-of-order packet 1800 is received but discarded
06:51:31.893664 -- out-of-order packet 1801 is received
06:51:31.893956 -- out-of-order packet 1802 is received
06:51:31.893998 -- out-of-order packet 1803 is received
06:51:31.894076 -- out-of-order packet 1804 is received
06:51:32.301407 -- packet 1800 is received
06:51:32.302009 -- sending ack for the received 1800
06:51:32.302099 -- packet 1801 is received
06:51:32.302195 -- sending ack for the received 1801
06:51:32.302238 -- packet 1802 is received
06:51:32.302286 -- sending ack for the received 1802
06:51:32.302322 -- packet 1803 is received
06:51:32.302355 -- sending ack for the received 1803
06:51:32.302404 -- packet 1804 is received
06:51:32.302448 -- sending ack for the received 1804
06:51:32.406756 -- packet 1805 is received

"Node: h1"
06:51:31.791147 -- packet with seq = 1803 is sent, sliding window = {1799, 1800, 1801, 1802, 1803}
06:51:31.791959 -- ACK for packet = 1799 is received
06:51:31.792129 -- packet with seq = 1804 is sent, sliding window = {1800, 1801, 1802, 1803, 1804}
06:51:32.195746 -- RTT occurred
06:51:32.196147 -- retransmitting packet with seq = 1800
06:51:32.196414 -- retransmitting packet with seq = 1801
06:51:32.196502 -- retransmitting packet with seq = 1802
06:51:32.196548 -- retransmitting packet with seq = 1803
06:51:32.196589 -- retransmitting packet with seq = 1804
06:51:32.302450 -- ACK for packet = 1800 is received
06:51:32.302754 -- packet with seq = 1805 is sent, sliding window = {1801, 1802, 1803, 1804, 1805}
06:51:32.302888 -- ACK for packet = 1801 is received
06:51:32.302947 -- packet with seq = 1806 is sent, sliding window = {1802, 1803, 1804, 1805, 1806}
06:51:32.307485 -- ACK for packet = 1802 is received
06:51:32.307559 -- packet with seq = 1807 is sent, sliding window = {1803, 1804, 1805, 1806, 1807}
06:51:32.307582 -- ACK for packet = 1803 is received
06:51:32.307610 -- packet with seq = 1808 is sent, sliding window = {1804, 1805, 1806, 1807, 1808}
06:51:32.307628 -- ACK for packet = 1804 is received
```

Simulert pakketap

100ms delay, window size 5, 2% packet loss:

Når pakkene ble sendt med 2% pakketap ble det merkbart at det tok lengre tid og det ble dårligere datagjennomstrømming. Dette er fordi mange pakker må sendes på nytt grunnet Go-Back-N protokollen, noe som fører til større tidsbruk og dårligere effektivitet på datagjennomstrømming.

```
"Node: h2"
06:54:59.147682 -- sending ack for the received 1833
06:54:59.249136 -- packet 1834 is received
06:54:59.249738 -- sending ack for the received 1834
06:54:59.249813 -- packet 1835 is received
06:54:59.249864 -- sending ack for the received 1835
06:54:59.249887 -- packet 1836 is received
06:54:59.249922 -- sending ack for the received 1836
06:54:59.249950 -- packet 1837 is received
06:54:59.249986 -- sending ack for the received 1837
06:54:59.250034 -- packet 1838 is received
06:54:59.250137 -- sending ack for the received 1838
06:54:59.354672 -- packet 1839 is received
06:54:59.354953 -- sending ack for the received 1839
06:54:59.355031 -- packet 1840 is received
06:54:59.355060 -- sending ack for the received 1840
06:54:59.355113 -- packet 1841 is received
06:54:59.355137 -- sending ack for the received 1841
FIN packet is received
FIN ACK packet is sent

The throughput is 0.28 Mbps
Data Transfer finished in 52.01 seconds
Connection Closes
root@debian:/home/debian/Desktop#

"Node: h1"
06:54:59.148051 -- ACK for packet = 1833 is received
06:54:59.148082 -- packet with seq = 1838 is sent, sliding window = {1834, 1835, 1836, 1837, 1838}
06:54:59.250124 -- ACK for packet = 1834 is received
06:54:59.250444 -- packet with seq = 1839 is sent, sliding window = {1835, 1836, 1837, 1838, 1839}
06:54:59.250515 -- ACK for packet = 1835 is received
06:54:59.250565 -- packet with seq = 1840 is sent, sliding window = {1836, 1837, 1838, 1839, 1840}
06:54:59.250694 -- ACK for packet = 1836 is received
06:54:59.250643 -- packet with seq = 1841 is sent, sliding window = {1837, 1838, 1839, 1840, 1841}
06:54:59.250671 -- ACK for packet = 1837 is received
06:54:59.250698 -- ACK for packet = 1838 is received
06:54:59.355065 -- ACK for packet = 1839 is received
06:54:59.355142 -- ACK for packet = 1840 is received
06:54:59.355349 -- ACK for packet = 1841 is received

Connection Teardown:
FIN packet is sent
FIN ACK packet is received
Connection Closes
root@debian:/home/debian/Desktop#
```

100ms delay, window size 5, 5% packet loss:

Det samme skjer når man øker prosentandelen på pakketap. Det tar enda lengre tid og det blir enda dårligere datagjennomstrømning fordi det er enda flere pakker som må retransmitteres. Det er igjen grunnet Go-Back-N at alt som er i vinduet til de tapte pakkene må sendes på nytt.

```
"Node: h2"
06:57:36.623687 -- sending ack for the received 1833
06:57:36.627480 -- packet 1834 is received
06:57:36.627968 -- sending ack for the received 1834
06:57:36.628069 -- packet 1835 is received
06:57:36.628114 -- sending ack for the received 1835
06:57:36.628158 -- packet 1836 is received
06:57:36.628205 -- sending ack for the received 1836
06:57:36.628242 -- packet 1837 is received
06:57:36.628284 -- sending ack for the received 1837
06:57:36.628321 -- packet 1838 is received
06:57:36.628373 -- sending ack for the received 1838
06:57:36.731065 -- packet 1839 is received
06:57:36.731404 -- sending ack for the received 1839
06:57:36.731458 -- packet 1840 is received
06:57:36.731484 -- sending ack for the received 1840
06:57:36.731548 -- packet 1841 is received
06:57:36.731572 -- sending ack for the received 1841
FIN packet is received
FIN ACK packet is sent

The throughput is 0.19 Mbps
Data Transfer finished in 76.26 seconds
Connection Closes
root@debian:/home/debian/Desktop#

"Node: h1"
06:57:36.624417 -- ACK for packet = 1833 is received
06:57:36.624554 -- packet with seq = 1838 is sent, sliding window = {1834, 1835, 1836, 1837, 1838}
06:57:36.628422 -- ACK for packet = 1834 is received
06:57:36.628757 -- packet with seq = 1839 is sent, sliding window = {1835, 1836, 1837, 1838, 1839}
06:57:36.628791 -- ACK for packet = 1835 is received
06:57:36.628827 -- packet with seq = 1840 is sent, sliding window = {1836, 1837, 1838, 1839, 1840}
06:57:36.628850 -- ACK for packet = 1836 is received
06:57:36.628897 -- packet with seq = 1841 is sent, sliding window = {1837, 1838, 1839, 1840, 1841}
06:57:36.628908 -- ACK for packet = 1837 is received
06:57:36.628929 -- ACK for packet = 1838 is received
06:57:36.731678 -- ACK for packet = 1839 is received
06:57:36.731781 -- ACK for packet = 1840 is received
06:57:36.731799 -- ACK for packet = 1841 is received

Connection Teardown:
FIN packet is sent
FIN ACK packet is received
Connection Closes
root@debian:/home/debian/Desktop#
```

100ms delay, window size 5, 50% packet loss:

Dette var det som tok desidert lengst tid. 50% pakketap fører til at omtrent halvparten av pakkene går tapt og at de pakkene som er i samme vindu må sendes om igjen, noe som fører til at majoriteten av pakkene må sendes på nytt. Dette tok da 14 minutter for maskinen å gjøre.


```
"Node: h2"
07:13:00.371970 -- packet 1837 is received
07:13:00.372449 -- sending ack for the received 1837
07:13:00.372621 -- packet 1838 is received
07:13:00.372600 -- sending ack for the received 1838
07:13:00.372638 -- out-of-order packet 1841 is received
07:13:00.388986 -- out-of-order packet 1841 is received
07:13:01.288437 -- out-of-order packet 1841 is received
07:13:01.693503 -- out-of-order packet 1840 is received
07:13:01.693656 -- out-of-order packet 1841 is received
07:13:03.322659 -- out-of-order packet 1841 is received
07:13:03.725032 -- packet 1839 is received
07:13:03.725077 -- sending ack for the received 1839
07:13:04.233811 -- out-of-order packet 1841 is received
07:13:05.042217 -- packet 1840 is received
07:13:05.042666 -- sending ack for the received 1840
07:13:05.042737 -- packet 1841 is received
07:13:05.042769 -- sending ack for the received 1841
FIN packet is received
FIN ACK packet is sent

The throughput is 0.02 Mbps
Data Transfer finished in 865.79 seconds
Connection Closes
root@debian:/home/debian/Desktop#

"Node: h1"
07:13:03.213884 -- retransmitting packet with seq = 1840
07:13:03.213968 -- retransmitting packet with seq = 1841
07:13:03.614591 -- RTO occurred
07:13:03.614825 -- retransmitting packet with seq = 1839
07:13:03.614962 -- retransmitting packet with seq = 1840
07:13:03.615010 -- retransmitting packet with seq = 1841
07:13:03.726050 -- ACK for packet = 1839 is received
07:13:04.132044 -- RTO occurred
07:13:04.132438 -- retransmitting packet with seq = 1840
07:13:04.132688 -- retransmitting packet with seq = 1841
07:13:04.535314 -- RTO occurred
07:13:04.535726 -- retransmitting packet with seq = 1840
07:13:04.535958 -- retransmitting packet with seq = 1841
07:13:04.939368 -- RTO occurred
07:13:04.939860 -- retransmitting packet with seq = 1840
07:13:04.940167 -- retransmitting packet with seq = 1841
07:13:05.043749 -- ACK for packet = 1840 is received
07:13:05.043890 -- ACK for packet = 1841 is received

Connection Teardown:
FIN packet is sent
07:13:05.456411 -- Timeout waiting for FIN-ACK, retrying
root@debian:/home/debian/Desktop#
```

Konklusjon

Gjennom dette prosjektet har jeg implementert en egen pålitelig transportprotokoll (DRTP) og testet den under varierende nettverksforhold. Eksperimentene viser tydelig at ytelsen påvirkes betydelig av vindusstørrelse, forsinkelse og pakketap.

Resultatene demonstrerer at for optimal gjennomstrømning bør vindusstørrelsen tilpasses nettverkets forsinkelse. Ved kort forsinkelse (50ms) ga vindusstørrelse 20 best ytelse, mens ved lengre forsinkelser flater ytelsesgevinsten ut. Dette bekrefter at større vinduer ikke alltid gir bedre resultater, særlig når forsinkelsen er høy eller pakketap forekommer.

Pakketapstestene viser hvordan Go-Back-N-protokollens retransmisjonsmekanisme fungerer, men også dens svakhet ved høye pakketap hvor hele vinduer må sendes på nytt. Ved 50% pakketap ble overføringstiden drastisk forlenget, noe som illustrerer protokollens begrensninger under ekstreme forhold.

Implementasjonen har gitt verdifull innsikt i transportprotokollenes virkemåte og utfordringer, samt praktisk erfaring med hvordan pålitelig dataoverføring kan kjøres over upålitelige nettverk. Protokollen fungerer under normale forhold, men kunne potensielt forbedres ved å implementere selektiv retransmisjon for å håndtere hyppig pakketap mer effektivt. (Islam, Datanettverk og Skytjenester, 2025)

Bibliografi

Islam, S. (2025). *Datanettverk og Skytjenester*. Hentet fra OsloMet Instructure :
<https://oslomet.instructure.com/courses/30561/modules>

Islam, S. (2025, Mai). *DRTP-v25*. Hentet fra Github: <https://github.com/safiquel/DRTP-v25>

Islam, S. (u.d.). *Introduction to Mininet*. Hentet fra Github:
<https://github.com/safiquel/2410/blob/main/docs/intro/mininet-intro.md>