Labbuppgift 2

Telecom ET1447

Olof Jönsson, oljn22

19901115-0712

# A1

Programkod för UDP sändare

# A2

Programkod för UDP mottagare

# A3

För att anpassa sändningstakten i strömmningen till angiven frekvens, använde jag *time.sleep()* och *time.perf\_counter().* Jag lät tiden för *sleep()* vara 1/frekvensen, minus exekveringstiden för sändningen, minus 570 mikrosekunder. På så sätt sänds paket nära den angivna frekvensen.

# B1

1. Här kunde jag sända 450 pkt på 30 sekunder utan att justera sleep-funktionen (se A3).
2. Sändningen ligger stabilt på 15Hz, vilket syns i I/O-grafen.
3. I/O graf sändare
4. I/O graf mottagare, Mottagarens I/O-graf visar stabil 15Hz.
5. Inga fel upptäcktes hos mottagaren, trots att nätverket var stressat med 8 olika 4k-video-streams.

# B2

1. Här kunde jag sända 1499pkt pkt på 30 sekunder utan att justera sleep-funktionen (se A3).
2. Sändningen ligger stabilt på 50Hz, vilket syns i I/O-grafen.
3. I/O graf sändare
4. I/O graf mottagare, Mottagarens I/O-graf visar ganska stabil 50Hz, med vissa variationer.
5. Ett paket (nr 818) nådde aldrig mottagaren. Nätverket var även här under stress med åtta 4-k-streams.
6. Skillnaden mellan låg och hög last (15 Hz respektive 50 Hz) blev i praktiken att ett paket inte kom fram vid hög last.

# B3

1. Med *sleep(0)* sändes 125951 pkt på 30s, vilket motsvarar 4198Hz i medelvärde. I/O-grafen visar på en ganska kraftig variation kring 4kHz.
2. Utan *sleep()* sändes 163278 pkt på 30s, vilket motsvarar 5443Hz i medelvärde. I/O-grafen visar på en ganska kraftig variation kring 5.5kHz.
3. *Sleep(0)* avbryter den pågående tråden, och ger andra processer en chans att köras, vilket kan innebära en kort paus för den pågående processen istället för att börja om loopen direkt. Andra saker som kan begränsa frekvensen är hur hög belastningen är på det trådlösa nätverket, prestandan hos CPU och nätverkskort, samt vilket programmeringsspråk som används, och hur väl optimerad koden är. Jag kan även tänka mig att bufferstorlek hos socket spelar in.

# C1

Programkod för TCP, sändare och mottagare

# C2

TCP 15p/s – 30s

Jag fick sänka meddelandestorleken till 1450 tecken, för att undvika fragmentering.

1. Sändnings frekvensen var stabil på 15Hz.
2. I/O graf sändare. Det verkar inte förekomma några omsändningar eller *duplicate acks*. Jag använde *tcp.analysis.retransmission* och *tcp.analysis.duplicate\_ack* för att leta efter tecken på omsändningar. Eftersom det inte förekom några *Dup ACKS* borde även paketen ha kommit fram i rätt ordning.
3. I/O graf mottagares ack-paket. I/O-grafen visar en frekvens på 15hz för ACK-paketen från mottagaren till sändaren.

# C3

TCP 50p/s – 30s

1. Sändarens frekvens är stabil på 50Hz, med någon liten variation.
2. I/O graf sändare. Resultatet blev samma som vid 15Hz – Inga ordningsfel eller omsändningar. Dock verkar *TCP segment coalescing* ha tillämpats hos mottagaren, där två till fem originalpaket har klumpats ihop till ett vid några tillfällen (44 paket). All data levereras i ordning till applikation, men den kommer inte portioner om originaldata, vilket gör att sekvensnumren inte ser ut att stämma.
3. I/O graf ackpaket från mottagaren. I/O-grafen för ACK-paketen visar 25hz. Det beror på att mottagaren väntar och ackar två paket åt gången.

# D1

Resonera kring UDP / TCP. För- och nackdelar med protokollen. Egna observationer

# D2

Reflektion kring uppgiften. Problem etc. relevant?