Labbuppgift 2

Telecom ET1447

Olof Jönsson, oljn22

19901115-0712

A1

Programkod för UDP sändare – se *sender.py* och *udp_sender.py*.

A2

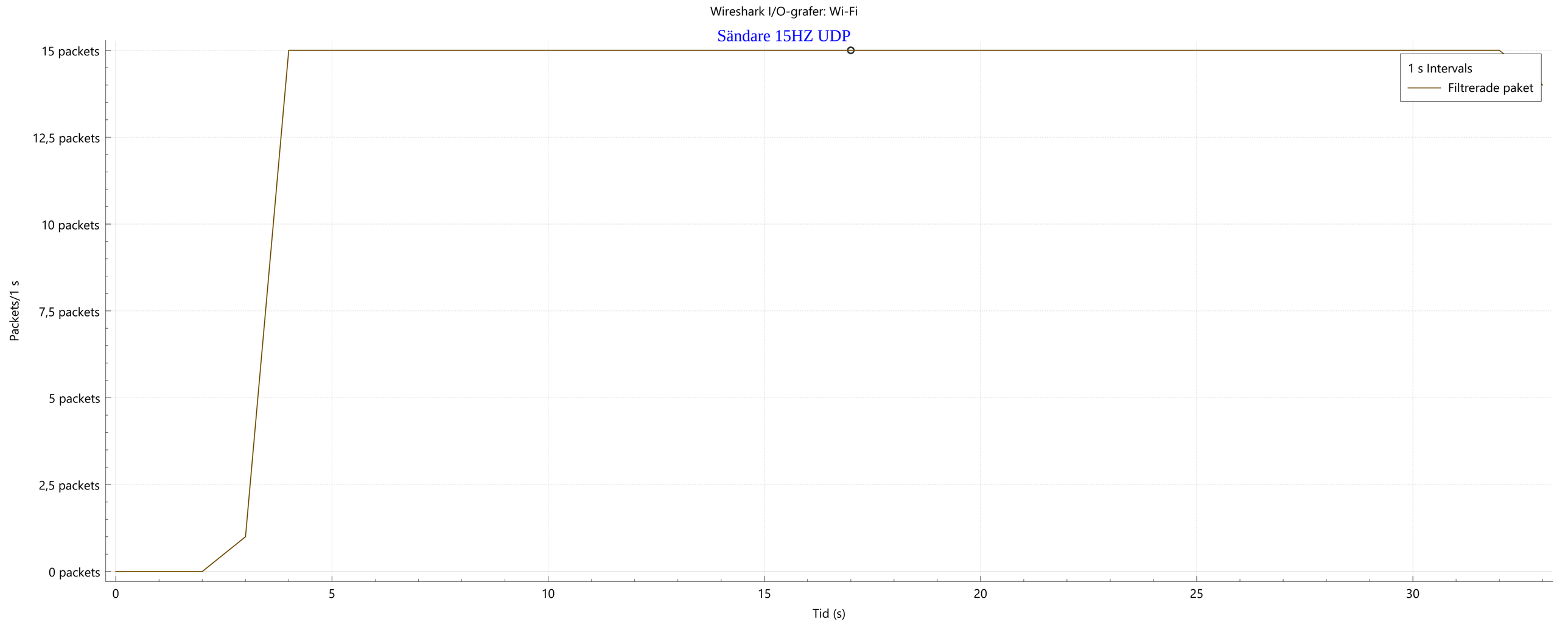
Programkod för UDP mottagare – *se receiver.py* och *udp_reciever.py*.

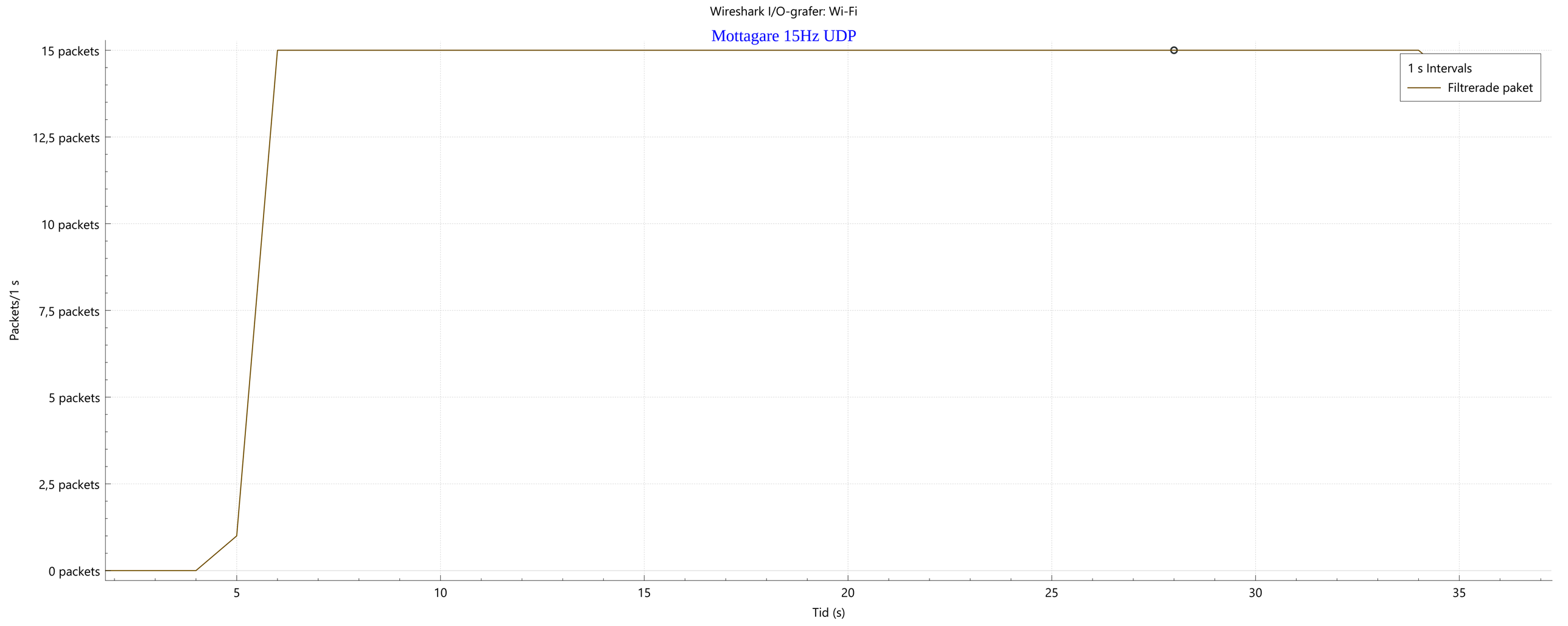
A3

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. Jag behövde inte modifiera koden för att klara olika frekvenser.

B1

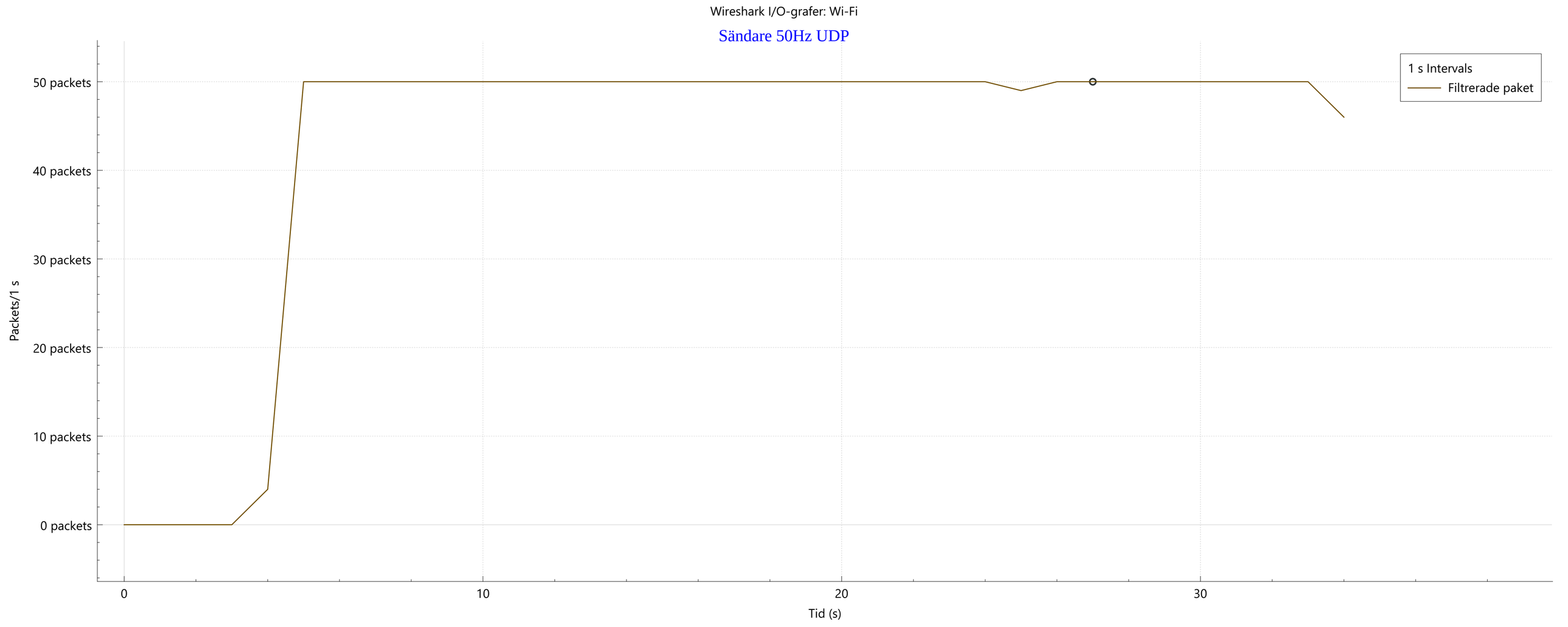
a) Här kunde jag sända 450 pkt på 30 sekunder utan att justera sleep-funktionen (se A3).
b) Sändningen ligger stabilt på 15Hz, vilket syns i I/O-grafen.
c) I/O-graf för sändaren, 15Hz UDP nedan.
d) I/O graf mottagare, 15Hz UDP nedan. Mottagarens I/O-graf ligger stabilt på 15Hz.
e) Inga fel upptäcktes hos mottagaren, trots att nätverket var stressat med 8 olika 4k-video-streams.

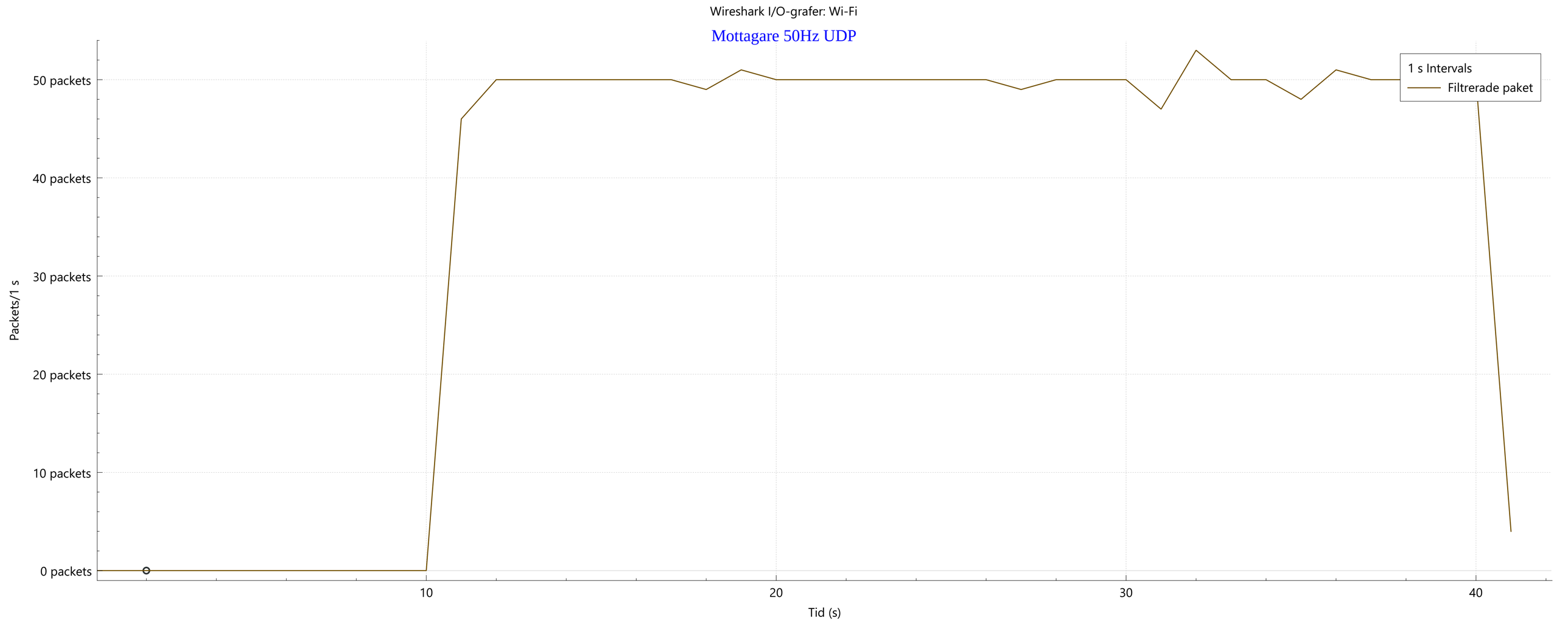

Wireshark I/O-grafer: Wi-Fi
Sändare 15HZ UDP
15 packets
12,5 packets
10 packets
7,5 packets
5 packets
2,5 packets
0 packets
Packets/1 s
0
5
10
15
20
25
30
Tid (s)
1 s Intervals
Filtrerade paket


Wireshark I/O-grafer: Wi-Fi
Mottagare 15Hz UDP
15 packets
12,5 packets
10 packets
7,5 packets
5 packets
2,5 packets
0 packets
Packets/1 s
5
10
15
20
25
30
35
Tid (s)
1 s Intervals
Filtrerade paket

B2

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


Wireshark I/O-grafer: Wi-Fi
Sändare 50Hz UDP
1 s Intervals
Filtrerade paket
50 packets
40 packets
30 packets
20 packets
10 packets
0 packets
Packets/1 s
0
10
20
30
Tid (s)


Wireshark I/O-grafer: Wi-Fi
Mottagare 50Hz UDP
1 s Intervals
Filtrerade paket
50 packets
40 packets
30 packets
20 packets
10 packets
0 packets
Packets/1 s
10
20
30
40
Tid (s)

B3

a) Med *sleep(0)* sändes 125 951 pkt på 30s, vilket motsvarar 4198Hz i medelvärde. I/O-grafen visar på en ganska kraftig variation kring 4kHz.
b) Utan *sleep()* sändes 163 278 pkt på 30s, vilket motsvarar 5443Hz i medelvärde. I/O-grafen visar på en ganska kraftig variation kring 5.5kHz.
c) *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.

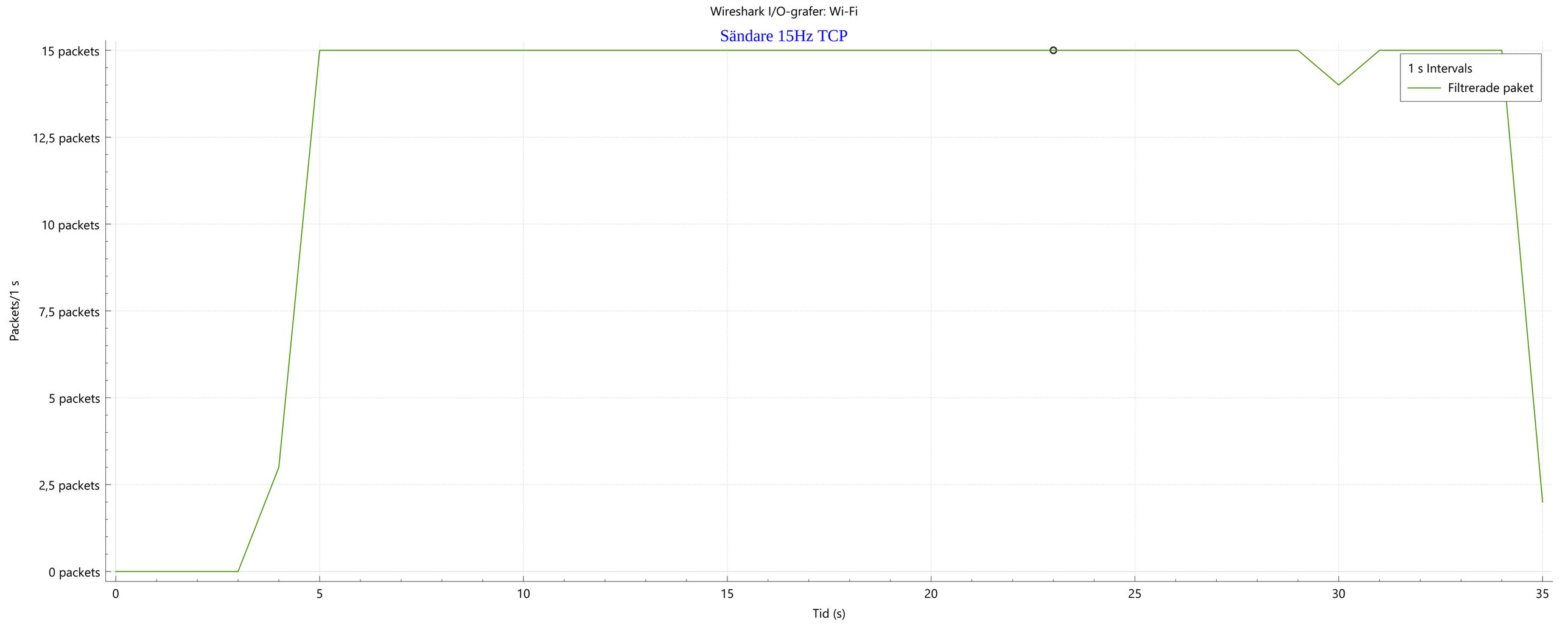
C1

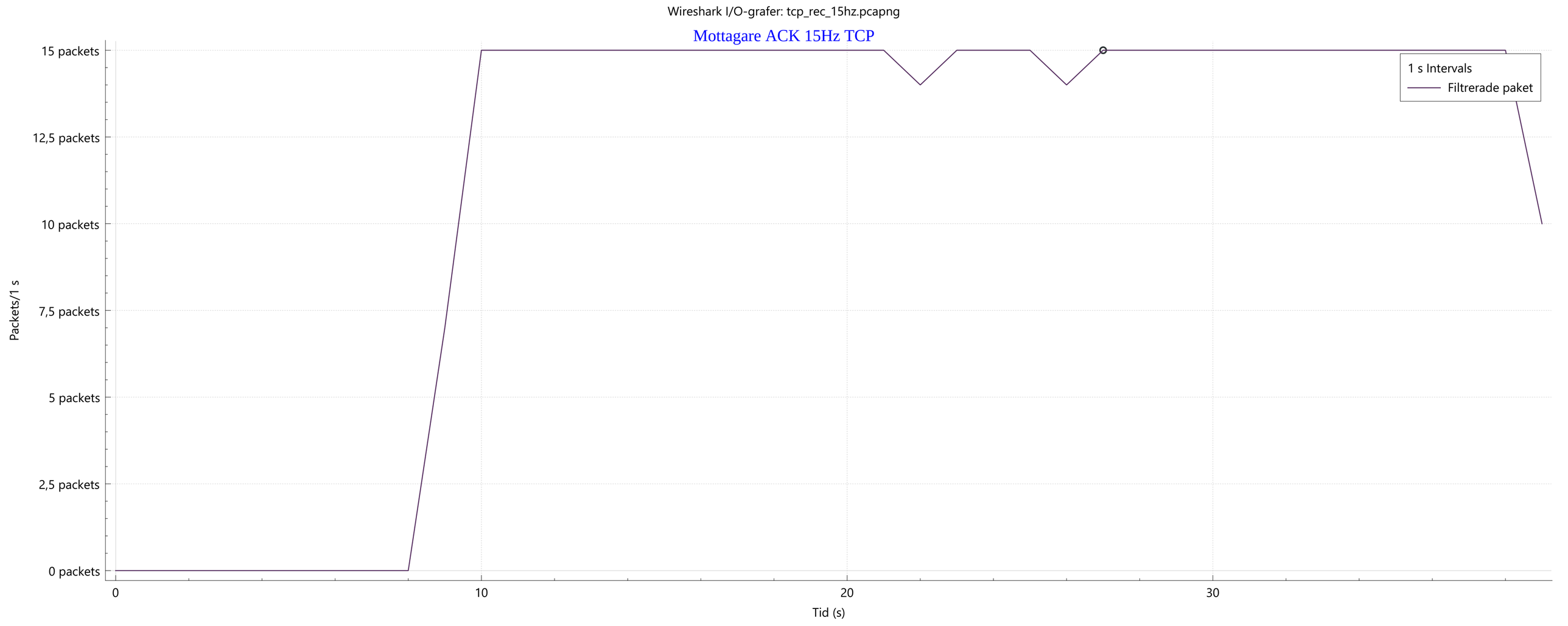
a) Programkod för TCP-sändare – se *sender.py* och *tcp_sender.py*.
b) Programkod för TCP-mottagare – se *receiver.py* och *tcp_receiver.py*.

C2

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

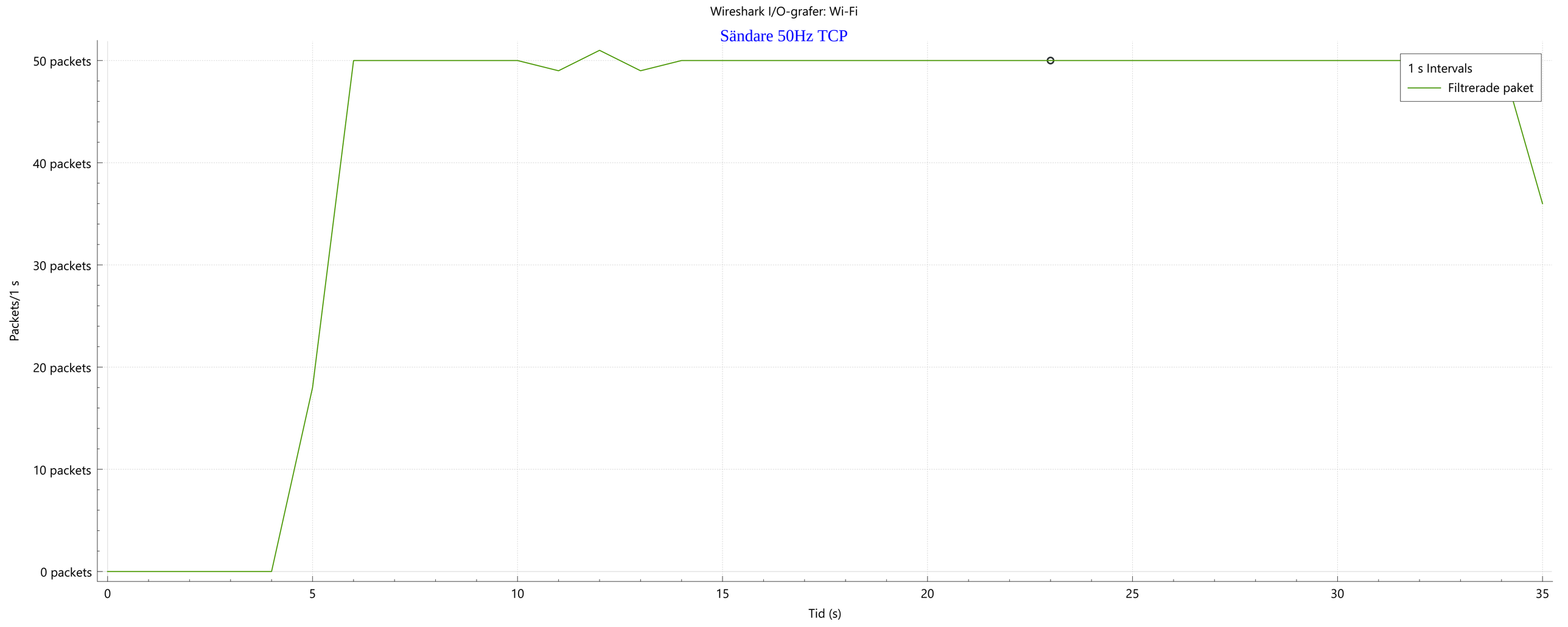
a) Sändningsfrekvensen var stabil på 15Hz.
b) I/O-graf sändare, 15Hz TCP nedan. Det verkar inte förekomma några omsändningar eller ordningsfel. Jag använde *tcp.analysis.retransmission* och *tcp.analysis.duplicate_ack i Wireshark* för att leta efter tecken på omsändningar och ordningsfel.
c) I/O graf för ACK-paket från mottagaren nedan. I/O-grafen visar en frekvens på 15hz för ACK-paketen från mottagaren till sändaren, vilket alltså är en ack per mottaget paket.

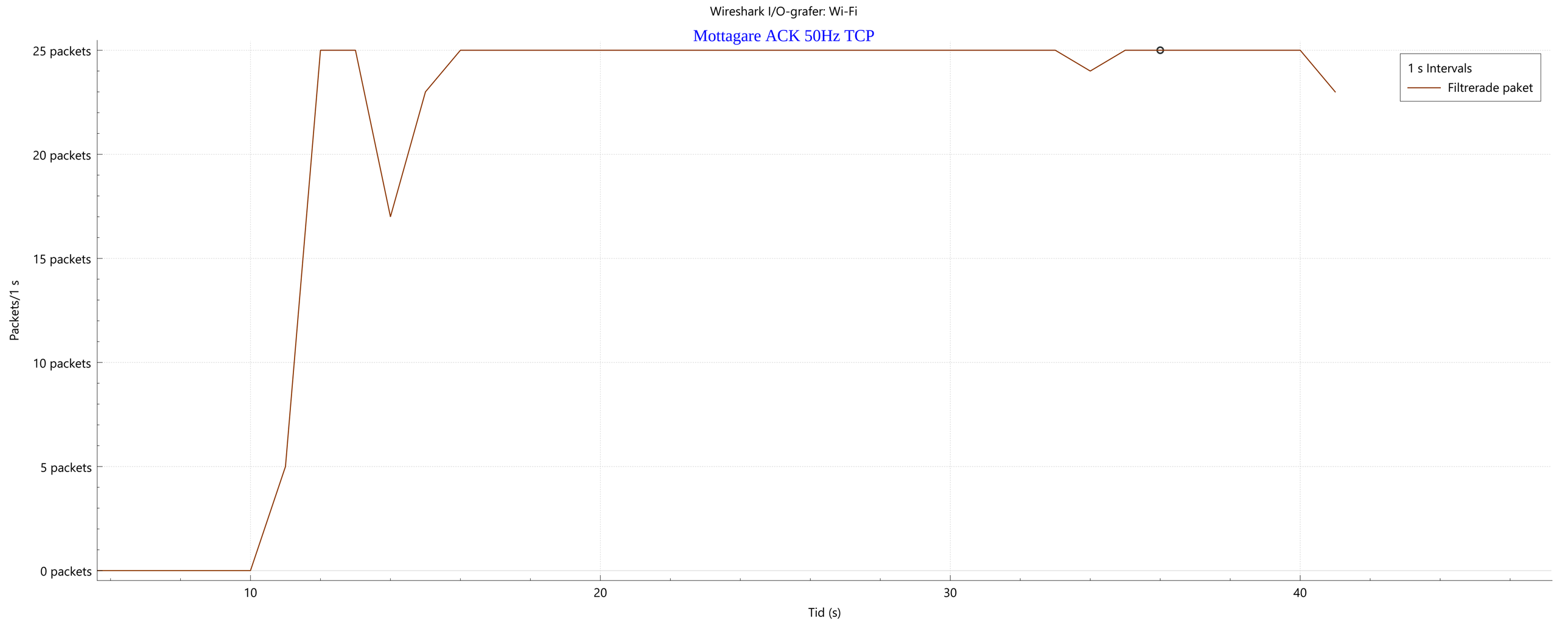

Wireshark I/O-grafer: Wi-Fi
Sändare 15Hz TCP
15 packets
12,5 packets
10 packets
7,5 packets
5 packets
2,5 packets
0 packets
Packets/1 s
0
5
10
15
20
25
30
35
Tid (s)
1 s Intervals
Filtrerade paket


Wireshark I/O-grafer: tcp_rec_15hz.pcapng
Mottagare ACK 15Hz TCP
15 packets
12,5 packets
10 packets
7,5 packets
5 packets
2,5 packets
0 packets
Packets/1 s
0
10
20
30
Tid (s)
1 s Intervals
Filtrerade paket

C3

a) Sändarens frekvens är stabil på 50Hz, med någon liten variation.
b) I/O-graf sändare, 50Hz TCP nedan. Resultatet blev samma som vid 15Hz – Inga ordningsfel eller omsändningar. Dock verkar *TCP segment coalescing* ha tillämpats hos mottagaren, där 2-5 originalpaket har klumpats ihop vid några tillfällen (44 paket blev till 11). All data levereras i ordning till applikation, men den anländer 1460-bytes i taget i de fall där flera paket har klumpats ihop, vilket gör att programmet inte tycker sekvensnumren ser ut att stämma.
c) I/O-graf för ACK-paket från mottagaren nedan. Grafen visar 25hz. Det beror på att mottagaren väntar och ackar två paket åt gången.


Wireshark I/O-grafer: Wi-Fi
Sändare 50Hz TCP
50 packets
40 packets
30 packets
20 packets
10 packets
0 packets
Packets/1 s
0
5
10
15
20
25
30
35
Tid (s)
1 s Intervals
Filtrerade paket


Wireshark I/O-grafer: Wi-Fi
Mottagare ACK 50Hz TCP
25 packets
20 packets
15 packets
10 packets
5 packets
0 packets
Packets/1 s
10
20
30
40
Tid (s)
1 s Intervals
Filtrerade paket

D1

User Datagram Protocol (UDP) är ett enkelt transportprotokoll, med fokus på liten overhead (kompakt header), och snabb överföring. Den erbjuder inga leveransgarantier och den hjälper inte till att anpassa hastighet efter mottagaren eller nätverket. Den erbjuder bitfelsdetektering med checksum i headern. UDP ger inga garantier för att alla paket levereras till applikationslagret.

Transmission Control Protocol (TCP) är ett mer komplext transportprotokoll, med fokus på pålitliga leveranser och anpassning till mottagare och nätverk. Den tillämpar en initiering (trevägs-handskakning) mellan två hostar, paketbekräftelser (ACK) och omsändning vid behov, anpassning av hastighet beroende av mottagarens och nätverkets aktuella status, och checksum. TCP garanterar att alla paket levereras i ordning till applikationslagret.

Fördelar med UDP är mindre tidsfördröjningar – ingen initiering, och ingen blockering vid omsändningar, hastigheten begränsas inte, och att headern tar liten plats. Fördelar med TCP är att överföringen är pålitlig, och att nätverket inte överutnyttjas. Beroende på ändamål lämpar sig den ena bättre än den andra. Vid live-stream av ljud/bild är små förluster försumbara, medan tidsfördröjning och hastighet är viktigt, vilket gör UDP till ett lämpligt val. I andra fall där dataförluster är kritiska, exepelvis e-post, lämpar sig TCP bättre.

I labben var den enda konkreta skillnaden en enda paketförlust (UDP 50Hz), men troligen skulle skillnaderna bli större om paketen skickades en längre väg över internet, istället för över WiFi på ett lokalt näverk.

D2

Det var kul att skriva python-koden, och det var intressant att bekanta sig lite mer med Wireshark.
Jag hade förväntat mig att det skulle bli lite större paketförluster vid 50Hz UDP.
Det tog en stund att lista ut varför vissa paket hade slagits ihop (*TCP segment coalescing*), men jag lärde mig en del om Wireshark på vägen.
Jag behövde låna en dator för att genomföra uppgiften, vilket var lite krångligt.
Labben kändes ganska relevant för kursen.