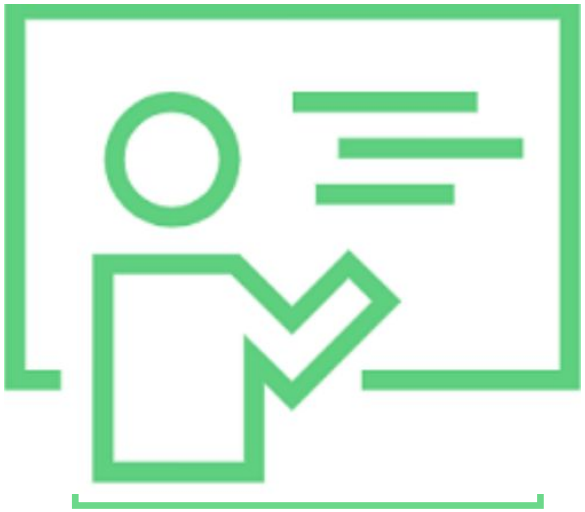


Portar, sockets och klientkommunikation

Joakim Englund, Systementor AB

Repetition från dag 1

Vad minns vi från tisdagen?

- En publisher skickar meddelanden till en broker
- En subscriber tar emot meddelanden från brokern
- Topics styr vilka meddelanden en subscriber får
- JSON kan användas för att strukturera sensordata
- IP-adress och port behövs för att nå brokern

Från MQTT till sockets

Vad händer under applikationsprotokollet?

Dagens mål

- Förstå IP-adress, port och socket
- Jämföra TCP och UDP
- Jämföra client-server, peer-to-peer och MQTT
- Observera trafik i Wireshark
- Felsöka ett kommunikationsflöde systematiskt

Vad är en socket?

- Ett programmeringsgränssnitt för nätverkskommunikation
- En ändpunkt som programmet använder för att sända och ta emot
- Skapas med val av adressfamilj och sockettyp

IPv4 + TCP → AF_INET + SOCK_STREAM
IPv4 + UDP → AF_INET + SOCK_DGRAM

Tänk typ en kontakt/uttag - fast digitalt

MQTT och sockets

Publisher → TCP-socket → MQTT-broker → TCP-socket → Subscriber

- MQTT ger topics och meddelanderegler
- TCP transporterar MQTT-trafiken
- IP och portar leder trafiken till rätt plats

IP-adress och port

192.168.1.50:5000

- IP-adressen hittar ett nätverksgränssnitt
- Porten leder trafiken till rätt process
- TCP och UDP har separata portutrymmen (men överlappar ofta)

TCP: en tillförlitlig byte-ström

- Anslutning etableras
- Data levereras i ordning
- Förlorad data skickas om
- Applikationen måste definiera meddelandegränser

connect → send/receive → close

Byte-ström - mellan connect och close kan i princip hur många paket som helst strömma utan att behöva “koppla upp” igen.

UDP: fristående datagram

- Ingen anslutning behöver etableras
- Ett datagram är ett avgränsat meddelandepaket
- Ett `sendto` motsvarar normalt ett mottaget datagram
- Ingen garanti för leverans eller ordning
- Låg protokolloverhead

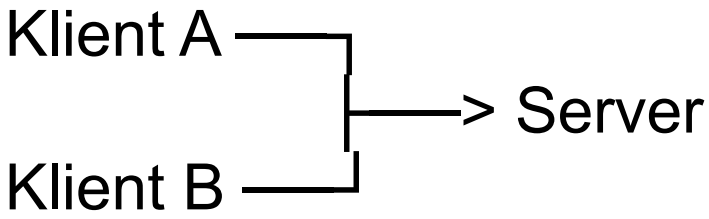
sendto → datagram → recvfrom

Varje paket går igenom “sendto → datagram → recvfrom”. De har *inte* den öppna strömmen som TCP har.
https://speedtesthq.com/guides/fundamentals/frame-vs-packet-vs-datagram

TCP eller UDP?

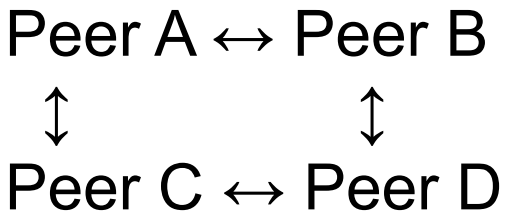
TCP	***UDP***
Tillförlitlig och ordnad	Leverans kan utebli
Byte-ström	Bevarade datagramgränser
Mer protokollmekanik	Mindre protokollmekanik
Bra när korrekt data prioriteras	Bra när viss förlust kan tolereras

Client-server



- Servern lyssnar på en känd port
- Klienten initierar kontakten
- Central plats för data och regler
- Servern blir ett centralt beroende

Peer-to-peer



- Jämbördiga noder kommunicerar direkt
- En nod kan vara både sändare och mottagare
- Mindre centralisering
- Upptäckt, säkerhet och felsökning blir ofta svårare

Tre kommunikationsmodeller

Modell	***Styrka***	***Utmaning***
Client-server	Tydlig central tjänst	Centralt beroende
Peer-to-peer	Direkt kommunikation	Komplex samordning
MQTT med broker	Producent och konsument frikopplas	Beroende av broker

URL: från socket till webbtjänst

http://192.168.1.50:8080/api/readings

- Schema: `http`
- Värd: `192.168.1.50`
- Port: `8080`
- Sökväg: `/api/readings`

Wireshark som observationsverktyg

tcp.port == 5000
udp.port == 5001

Kan visa:

- käll- och destinationsadress,
- käll- och destinationsport,
- protokoll och flaggor,
- innehåll i okrypterad testtrafik.

Dagens laboration

1. Bygg och testa ett TCP-flöde
2. Fånga TCP-trafiken i Wireshark
3. Bygg och testa motsvarande UDP-flöde
4. Inför och analysera kommunikationsfel
5. Jämför TCP, UDP, MQTT och peer-to-peer
6. Dokumentera och committa resultatet

Vad behöver vara rätt för att ett program ska nå rätt tjänst?
I princip allt: Protokoll + IP/värdnamn + port + fungerande process + tillåten trafik