IDATT2104 – Oblig DKO 2

Oppgave 1 - Adressering

a)

Delstrekk / Adresse	A Til Svitsj1	B Til Ruter 1	C Til Ruter 2	D Til Svitsj 2	E Til PC
Mottaker IP	177.177.1.1	177.177.1.1	177.177.1.1	155.155.1.55	155.155.1.55
Mottaker MAC	22:BB	22:BB	N/A (33:CC?)	44:DD	44:DD

Til ruter 2 vil antagelig mac-adresse være nærmeste proxy server, evt. 33:CC hvis ruterne er koblet direkte som illustrert i tegningen. Vi antar også at alle nodene har en viss kjennskap til nettstrukturen fra før av. Eventuelt vil muligens MAC-adresser til svitsjene også bli involvert.

b)

155-nettet:

155 nettet har flere enn 32 adresser 55-2 = 53. Dermed er det ikke et /27 nett. Samtidig er det fullt mulig at det har færre enn 64 adresser. Størst mulig nettmaske blir /26 som har 64 adresser hvorav 62 er host-adresser. Nettverksadressen er **155.155.1.0.**

166-nettet:

166 nettet har flere enn 165 adresser da 166-1 = 165. For å få størst mulig nettverksmaske må derfor dette være et /24 nett som har 256 adresser, hvorav 254 er host-adresser. Nettverksadressen blir da **166.166.1.0.**

177-nettet:

Ettersom dette nettet kun dukker opp en gang er det vanskelig å si noe om størrelse. Ettersom masken skulle være størst mulig settes det opp som et /30 nett som har 2 host adresser. Nettverksadressen blir da **177.177.1.0.**

Oppgave 2 - MAC-adresser og ARP

a)

Hensikten med ARP (Address Resolution Protocol) er å gjøre det mulig å knytte nettverksadresser fra nettverkslaget (lag 3) til hardware-adresser i datalinklaget (lag 2) i OSI-modellen.

b)

ARP forespørsler sendes til MAC-adressen ff-ff-ff-ff-ff-ff. Denne adressen er også kjent som Broadcast MAC-adressen. Alt som blir sendt hit blir videresendt til alle porter på nettverksswitchen hvor det blir mottatt av enheten med adressen avsender spesifiserte, og forkastet av de som ikke er den riktige mottakeren. Mottakeren sender deretter en bekreftelse på at den er den riktige mottakeren via unicast tilbake til avsenderen. Det som kjennetegner enheter som er mottaker av disse pakkene blir da at de er alle koblet til samme LAN eller er del av det samme broadcast domenet. c)

Resultatet av kjøring av kommandoen “arp -a”

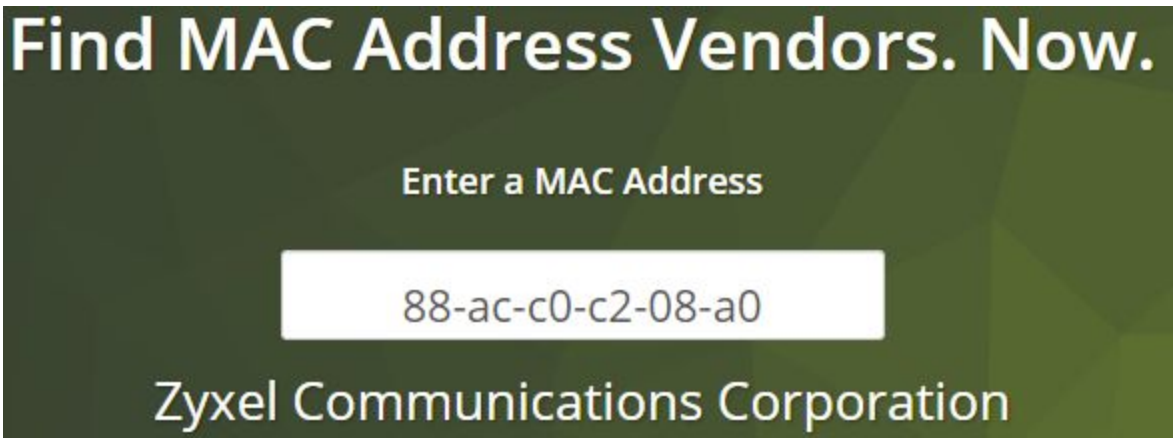
```
C:\Users\Lars>arp -a

Interface: 192.168.1.42 --- 0xb
  Internet Address      Physical Address      Type
  192.168.1.1           88-ac-c0-c2-08-a0     dynamic
  192.168.1.37          54-60-09-d6-80-32     dynamic
  192.168.1.163         d4-f5-47-17-e8-23     dynamic
  192.168.1.168         44-09-b8-45-f3-76     dynamic
  192.168.1.199         d0-03-df-82-ab-e6     dynamic
  192.168.1.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

Som bildet klart viser, ligger ikke min egen IP-adresse, 192.168.1.42 og dermed heller ikke min MAC-adresse, 74-c6-3b-ed-68-20, i min egen ARP table. Det man derimot kan gjøre for å finne sin egen MAC er å kjøre ipconfig /all, kjøre getmac kommandoen, sjekke routerens ARP table eller kjøre arp -a fra en annen maskin på det lokale nettet.

For å finne produsenten av nettverkskortet kan man kopiere MAC-adressen og søke den opp på for eksempel macvendors.com. Dette fungerer fordi IEEE sine lister med produsenter og de MAC-adresser de har blitt tildelt er offentlig informasjon. For å bruke informasjon fra arp tabellen for å svare på oppgaven har vi bestemt oss for å bruke routeren(192.168.1.1) istedenfor maskinens nettverkskort. Routerens MAC-adresse er ifølge tabellen 88-ac-c0-c2-08-a0.

En sjekk på macvendors.com forteller meg at produsenten av routeren er Zyxel. Dette stemmer også med virkeligheten.
Om man vil sjekke opp produsenten selv kan man gå til



https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries og søke opp enten produsentnavn eller de tre første bytes i MAC-adressen.

SEARCH RESULTS

88-ac-c0 | Filter | Reset

View 10 rows

1

Showing 1 - 1 of 1

ALL MAC (MA-L, MA-M, MA-S) SEARCH RESULTS — EXPORT

Assignment	Assignment Type	Company Name	Company Address
88-AC-C0 (hex) 88ACC0	MA-L	Zyxel Communications Corporation	No. 6 Innovation Road II, Science Park Hsichu Taiwan 300 TW

Oppgave 3 - IPv6

a)

<table>
<tr><td>Routing prefix</td><td>Subnet ID</td><td>Interface ID (IID)</td></tr>
<tr><td>48 or more bits</td><td>16 or less bits</td><td>64 bits</td></tr>
<tr><td colspan="2">Network prefix = Routing prefix + Subnet ID = 64 bits</td><td></td></tr>
<tr><td colspan="3">IPv6 Address = Network prefix + Interface ID = 128 bits.</td></tr>
</table>

IPv6 adresser er 128 bit. 64 bit blir brukt til routing, og 64 bit til interface identifikasjon. Routing prefix kan ha 48 eller fler bits, mens Subnet ID kan ha 16 eller færre. Network prefix er kombinasjonen av routing prefix og subnet ID. Interface identifier for unicast adresser er alltid 64 bit. Unntaket er, ifølge RFC 7421, de som starter med binærverdien 000. Figuren ble tegnet i draw.io.

b)

Et /60 IPv6 nett er ekvivalent i størrelse med 16 stk /64 nett. Man bør aldri splitte IPv6 nett slik at routing prefix blir mer enn 64 bit. Dette er fordi IPv6-adresser er 128 bit brede, og de siste 64 bit er alltid interface identifier. Om routing prefix skulle vært større enn 64 bit ville det gått utover interface identifieren, og funksjoner som SLAAC (Stateless address autoconfiguration) ville ikke fungert da den er avhengig av en 64 bit interface identifier. Antall hosts i et IPv6 nett er egentlig irrelevant da det har utrolig mange adresser uansett. Det ville vært ekstremt lite hensiktsmessig å splitte nettet helt ned /120 For å komme i nærheten av bedriftens “krav” til antall adresser.

Samtidig mener vi det også blir feil å splitte det til 4 /62 nett da man kaster bort veldig mange adresser og mulige nye subnett til senere bruk.

2001:db8::/60 blir derfor i vår løsning splittet ned til 16 /64 nett. Hvert nett har 2^{64} tilgjengelige adresser. På denne måten er nettet fortsatt innenfor alle krav satt i oppgaven, og det frigjør store deler av nettet som kan brukes senere. Bedriften vil etter splittingen sitte igjen med 12 ledige subnett. En annen løsning kunne vært å splitte til to /62 nett og åtte /64 nett.

Subnet	Subnetadresse	Address Range	Skrivemåte	I bruk
1	2001:db8::	2001:db8:: - 2001:db8::ffff:ffff:ffff:ffff	2001:db8::/64	Ja
2	2001:db8:0:1::	2001:db8:0:1:: - 2001:db8:0:1:ffff:ffff:ffff:ffff	2001:db8:0:1::/64	Ja
3	2001:db8:0:2::	2001:db8:0:2:: - 2001:db8:0:2:ffff:ffff:ffff:ffff	2001:db8:0:2::/64	Ja
4	2001:db8:0:3::	2001:db8:0:3:: - 2001:db8:0:3:ffff:ffff:ffff:ffff	2001:db8:0:3::/64	Ja
5	2001:db8:0:4::	2001:db8:0:4:: - 2001:db8:0:4:ffff:ffff:ffff:ffff	2001:db8:0:4::/64	Nei
6	2001:db8:0:5::	2001:db8:0:5:: - 2001:db8:0:5:ffff:ffff:ffff:ffff	2001:db8:0:5::/64	Nei
7	2001:db8:0:6::	2001:db8:0:6:: - 2001:db8:0:6:ffff:ffff:ffff:ffff	2001:db8:0:6::/64	Nei
8	2001:db8:0:7::	2001:db8:0:7:: - 2001:db8:0:7:ffff:ffff:ffff:ffff	2001:db8:0:7::/64	Nei
9	2001:db8:0:8::	2001:db8:0:8:: - 2001:db8:0:8:ffff:ffff:ffff:ffff	2001:db8:0:8::/64	Nei
10	2001:db8:0:9::	2001:db8:0:9:: - 2001:db8:0:9:ffff:ffff:ffff:ffff	2001:db8:0:9::/64	Nei
11	2001:db8:0:a::	2001:db8:0:a:: - 2001:db8:0:a:ffff:ffff:ffff:ffff	2001:db8:0:a::/64	Nei
12	2001:db8:0:b::	2001:db8:0:b:: - 2001:db8:0:b:ffff:ffff:ffff:ffff	2001:db8:0:b::/64	Nei
13	2001:db8:0:c::	2001:db8:0:c:: - 2001:db8:0:c:ffff:ffff:ffff:ffff	2001:db8:0:c::/64	Nei
14	2001:db8:0:d::	2001:db8:0:d:: - 2001:db8:0:d:ffff:ffff:ffff:ffff	2001:db8:0:d::/64	Nei
15	2001:db8:0:e::	2001:db8:0:e:: - 2001:db8:0:e:ffff:ffff:ffff:ffff	2001:db8:0:e::/64	Nei
16	2001:db8:0:f::	2001:db8:0:f:: - 2001:db8:0:f:ffff:ffff:ffff:ffff	2001:db8:0:f::/64	Nei

Oppgave 4 – DHCP

a)

Kjøring av ipconfig /all etter kjøring av ipconfig /release:

```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Broadcom 802.11ac Network Adapter
   Physical Address. . . . . . . . . : 24-0A-64-F5-50-CC
   DHCP Enabled. . . . . . . . . . . : Yes
1  Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::9c43:11f2:6bc1:97fc%12(Preferred)
2  Autoconfiguration IPv4 Address. . : 169.254.151.252(Preferred)
3  Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Default Gateway . . . . . . . . . :
   DHCPv6 IAID . . . . . . . . . . . : 153356900
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-25-A3-5F-47-BC-EE-7B-76-8C-E1
   DNS Servers . . . . . . . . . . . : fec0:0:0:ffff::1%1
                                       fec0:0:0:ffff::2%1
                                       fec0:0:0:ffff::3%1
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

Ettersom autoconfiguration (1) er skrudd på, vil man ved (2) se at PC-en har fått den nye midlertidige adressen 169.254.151.252 etter at ipconfig /release er kjørt. Ved Subnet Mask (3) kan man også se at den har fått /16 masken 255.255.0.0 Alt dette betyr egentlig bare at du ikke når DHCP serveren, så maskinen deler ut en midlertidig adresse ved å velge et tilfeldig 16 bits nummer og legger det til etter 169.254. Deretter utføres en Gratuitous ARP broadcast og maskinen assigner adressen til seg selv.

Etter kjøring av ipconfig /renew:

```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : home
   Description . . . . . . . . . . . : Broadcom 802.11ac Network Adapter
   Physical Address. . . . . . . . . : 24-0A-64-F5-50-CC
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::9c43:11f2:6bc1:97fc%12(Preferred)
1  IPv4 Address. . . . . . . . . . . : 192.168.0.7(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
3  Lease Obtained. . . . . . . . . . : fredag 19. mars 2021 15:09:01
4  Lease Expires . . . . . . . . . . : fredag 19. mars 2021 16:09:01
   Default Gateway . . . . . . . . . : 192.168.0.1
2  DHCP Server . . . . . . . . . . . : 192.168.0.1
   DHCPv6 IAID . . . . . . . . . . . : 153356900
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-25-A3-5F-47-BC-EE-7B-76-8C-E1
   DNS Servers . . . . . . . . . . . : 84.208.20.110
                                       84.208.20.111
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

Her kan man se ved (1) at PCen har fått tildelt IP-adressen 192.168.0.7 av DHCP serveren (2). Man kan også se på Lease Obtained (3) og Lease Expires (4) for å få informasjon om når IP-adressen ble reservert og hvor lenge reservasjonen varer.

De fire utvekslede pakkene i WireShark:

```
1 0.000000      0.0.0.0           255.255.255.255     DHCP     344 DHCP Discover - Transaction ID 0x2ba8cbcc
2 0.532225      192.168.0.1       192.168.0.7         DHCP     342 DHCP Offer    - Transaction ID 0x2ba8cbcc
3 0.534136      0.0.0.0           255.255.255.255     DHCP     370 DHCP Request  - Transaction ID 0x2ba8cbcc
4 0.550053      192.168.0.1       192.168.0.7         DHCP     342 DHCP ACK      - Transaction ID 0x2ba8cbcc
```

Fra Klient, Option: (53) Message type:

MAC (3 første byte)		IP		UDP Portnummer	
DST	SRC	SRC	DST (2 byte)	SRC	DST
ff:ff:ff	24:0a:64	0.0.0.0	255.255	68	67

Fra Tjener, Option: (53) Message type:

MAC (3 første byte)		IP		UDP Portnummer	
DST	SRC	SRC	DST (2 byte)	SRC	DST
24:0a:64	98:1e:19	192.168.0.1	0.7	67	68

Fra Klient, Option: (53) Message type:

MAC (3 første byte)		IP		UDP Portnummer	
DST	SRC	SRC	DST (2 byte)	SRC	DST
ff:ff:ff	24:0a:64	0.0.0.0	255.255	68	67

Fra Tjener, Option: (53) Message type:

MAC (3 første byte)		IP		UDP Portnummer	
DST	SRC	SRC	DST (2 byte)	SRC	DST
24:0a:64	98:1e:19	192.168.01	0.7	67	68

Oppsummering fra tabell

1. Hvilke MAC-adresser sendes alle pakker til:

Klient bruker kringkastings-adressen på lenkelaget (ff:ff:ff:ff:ff:ff), så pakker blir videresendt til alle noder i nettverket. Pakker i retur fra DHCP-tjener blir unicast adressert til på klient.

2. Hvilken IP-adresse sendes alle pakker til:

I likhet med lenkelaget blir DHCP spørringene også kringkastet på nettverkslaget vha. den reserverte kringkastings-adressen 255.255.255.255. Pakker i retur har unicast adressering med den leasede adressen.

3. Hvilken IP-adresse sender klienten *fra*:

Man kan se i skjermbildet fra WireShark på forrige side at klienten bruker IP-adressen 0.0.0.0. Dette er fordi den enda ikke har fått tildelt en IP-adresse på nettet, og er helt forventet når klienten sender DHCP Discover pakken.

b)

Option (1): Her deles subnettet ut. I dette tilfellet deles 255.255.255.0

```
Option: (1) Subnet Mask (255.255.255.0)
    Length: 4
    Subnet Mask: 255.255.255.0
```

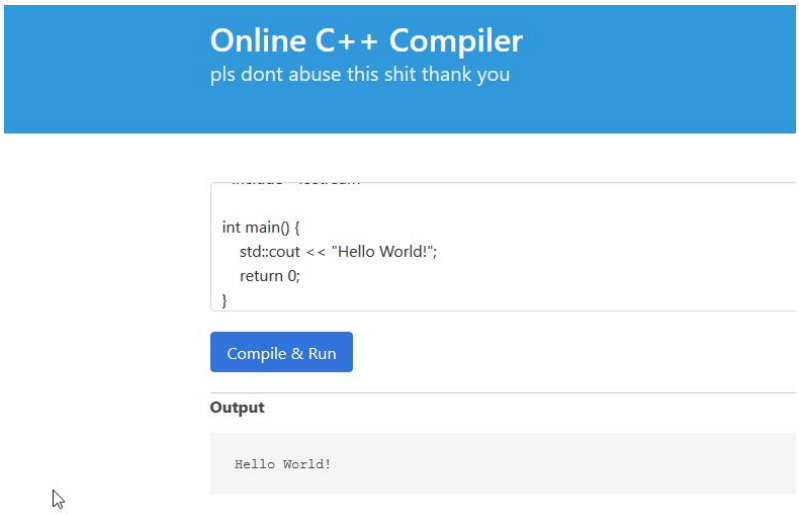
Option (3): Her deles router-adressen/default gateway ut. I dette tilfellet 192.168.0.1

```
Option: (3) Router
    Length: 4
    Router: 192.168.0.1
```

Option (6): Brukes til å dele routerens primære og sekundære DNS-server.

```
Option: (6) Domain Name Server
    Length: 8
    Domain Name Server: 84.208.20.110
    Domain Name Server: 84.208.20.111
```

Oppgave 5 – Dokumentasjon øving 5



1. *Øvingens hensikt*

Øvingen gikk ut på å lage en nettside som lot brukeren skrive inn kode, og la webserveren vise resultatet av kompileringen. Kompileringen skulle foregå i Docker.

2. *Ordforklaringer*

Docker: OS-level virtualiseringsverktøy. Bruker mindre ressurser enn virtuelle maskiner da alle containers i docker deler services.

Node.js: Open source runtime-system som kjører på tvers av plattformer for server- og nettverksapplikasjoner.

Express.js: rammeverk for webapplikasjoner for Node.js.

jQuery: populært javascript-rammeverk for nettleseren

3. *Program og datakommunikasjon*

Øvingen ble gjennomført ved bruk av teknologiene Docker, Node.js med Express.js som rammeverk for webapplikasjonen. I tillegg ble det brukt jQuery på nettsiden for å gjøre det enklere. Docker i denne sammenhengen brukes for å kjøre programkode som sendes inn av brukeren i et virtualisert miljø.

Kommunikasjon med webapplikasjonen er veldig enkel, og består av to steg. Det første steget er å sende inn programkode, andre steget er webapplikasjonen som svarer med output fra kjøring av koden.

788	9.942639	127.0.0.1	127.0.0.1	TCP	56	64702 → 3000 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=25
791	9.942672	127.0.0.1	127.0.0.1	TCP	56	3000 → 64702 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=
792	9.942694	127.0.0.1	127.0.0.1	TCP	44	64702 → 3000 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
795	9.942800	127.0.0.1	127.0.0.1	HTTP	754	POST /compile HTTP/1.1 (application/x-www-form-urlencod
796	9.942808	127.0.0.1	127.0.0.1	TCP	44	3000 → 64702 [ACK] Seq=1 Ack=711 Win=2619648 Len=0
1038	12.813639	127.0.0.1	127.0.0.1	HTTP	283	HTTP/1.1 200 OK (text/html)
1039	12.813665	127.0.0.1	127.0.0.1	TCP	44	64702 → 3000 [ACK] Seq=711 Ack=240 Win=2619392 Len=0
2000	17.826600	127.0.0.1	127.0.0.1	TCP	44	3000 → 64702 [FIN, ACK] Seq=240 Ack=711 Win=2619648 Len=
2002	17.826614	127.0.0.1	127.0.0.1	TCP	44	64702 → 3000 [ACK] Seq=711 Ack=241 Win=2619392 Len=0
2004	17.826698	127.0.0.1	127.0.0.1	TCP	44	64702 → 3000 [FIN, ACK] Seq=711 Ack=241 Win=2619392 Len=
2005	17.826709	127.0.0.1	127.0.0.1	TCP	44	3000 → 64702 [ACK] Seq=241 Ack=712 Win=2619648 Len=0

Skjermbildet ovenfor viser kommunikasjonen fra innsending av kode til respons med output.

Første steg:

Nettleseren kjører et javascript-snutt som utfører en POST forespørsel der den sender med innholdet (koden) i tekstboksen. Dersom forespørselen får et svar med kode 200 OK, vil den plassere innholdet i svaret på nettsiden.

```
$.post('/compile', { code: codebox.val() }, (data) => {
    output.html(data);
}).fail((data) => {
    output.html("Error occured lol");
}).always(() => {
    button.removeClass('is-loading');
})
```

```
> Hypertext Transfer Protocol
v HTML Form URL Encoded: application/x-www-form-urlencoded
    Form item: "code" = "// Your First C++ Program

    #include <iostream>

    int main() {
        std::cout << "Hello World!";
        return 0;
    }"
```

Andre steg:

Webserveren mottar forespørselen med kode. Koden blir skrevet til en fil slik at den kan kjøres senere. Koden blir kompilert gjennom en prosess som kjøres via Docker. Etter kompilering blir den så kjørt, der output blir fanget og videresendt som et HTTP 200 OK svar med innhold.

```
exec('docker build "./cppc/" -t gcc', (stderr : ExecException | null ) => {
    exec('docker run --rm gcc', (err : ExecException | null , stdout : string , stderr : string ) => {
        console.log('Err: ' + err);
        console.log('Stdout: ' + stdout);
        console.log('Stderr: ' + stderr);
        if (err) {
            console.log(err);
            return res.status( code: 500).send( body: 'Internal Server Error');
        }
        else {
            return res.status( code: 200).send(
                stdout
            );
        }
    })
});
```

```
> Transmission Control Protocol, Src Port: 3000, Dst Port: 64702, Se
> Hypertext Transfer Protocol
v Line-based text data: text/html (1 lines)
    Hello World!
```

4. <u>Sammendrag</u>

Gikk greit å komme i gang med den grunnleggende webtjenesten, der ligger det flere tilgjengelige rammeverk i IDE-en (IntelliJ). For noen som ikke er så kjent med Docker, kan det bli mye feilsøking dersom man gjør feil i for eksempel installering og oppsett av WSL. Videre progresjon i utviklingen gikk greit etter man fikk Docker riktig satt opp.

Oppgave 6 – Dokumentasjon øving 6

Øvingen gikk ut på å bruke rfc6455 til å programmere et websocket server library som man gjennom socketprogrammering kunne bruke til å sende og motta meldinger fra klienter, samt utføre handshakes.

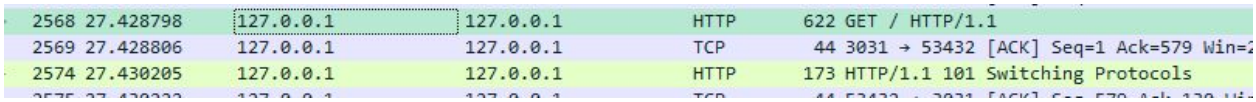
2. <u>Ordforklaringer</u>

RFC6455: Websocket-protokoll standardisert av Internet Engineering Task Force (IETF)

Websocket: En persistent TCP tilkobling mellom en klient og en server. Bruker HTTP til kommunikasjon.

3. <u>Program og datakommunikasjon</u>

Øvingen ble gjennomført ved å lage en enkel webserver i JavaScript med Node.js, som sender ut en HTML-fil til nettleseren. HTML-filen inneholder JavaScript-kode som oppretter en WebSocket-tilkobling over TCP til WebSocket-serveren som lytter på en annen port enn webserveren.



Skjermbildet viser pakkeoverføring ved opprettelse av ny WebSocket-tilkobling. I den først pakken inneholder den “upgrade”-verdi i connection-attributten.

```
connection.on( event: 'data', listener: (data : Buffer ) => {
    if (!connection.upgraded) {
        let key = getHeaderLine(data, filter: "Sec-WebSocket-Key");
        connection.write(getHandshakeResponse(getAcceptKey(key)));
        connection.upgraded = true;

        console.log('Upgraded client connection');
    } else {
```

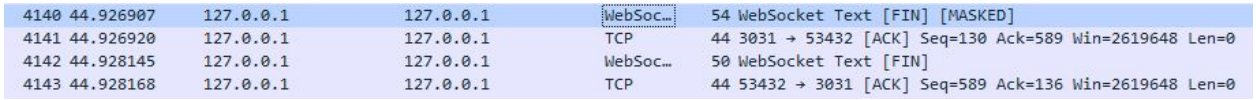
På serversiden venter vi på oppkoblinger fra nettlesere, og sjekker om tilkoblingen er flagget som upgraded. Dersom den ikke er det, vil den sende et svar med HTTP 101 Switching Protocols, sammen med en Sec-WebSocket-Accept attributt som genereres av kode i skjermbilde nedenfor:

```
const getHandshakeResponse = (acceptKey) => {
    return "HTTP/1.1 101 Switching Protocols\r\n" +
        "Upgrade: websocket\r\n" +
        "Connection: Upgrade\r\n" +
        "Sec-WebSocket-Accept: " + acceptKey + "\r\n\r\n"
}
```

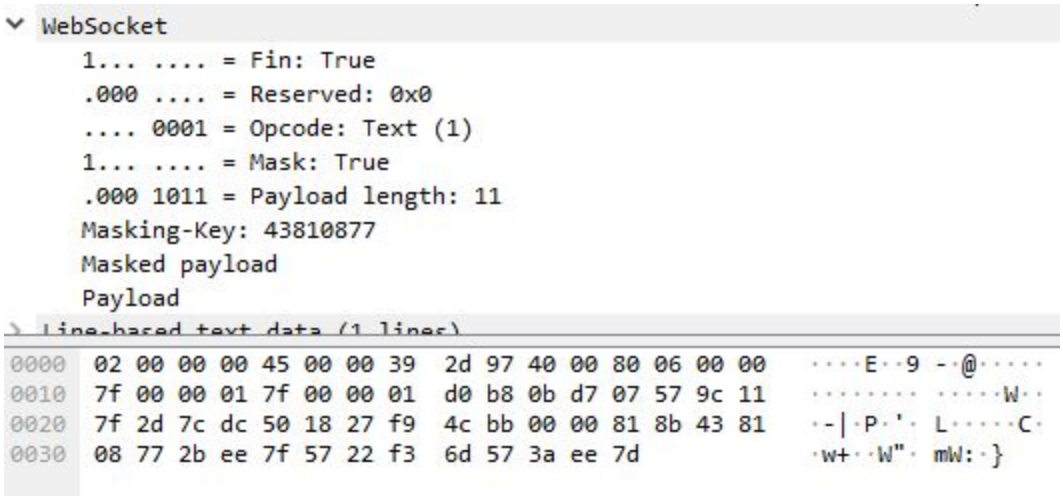
Inne i metoden getHandshakeResponse henter vi acceptKey fra metoden der “key” er maskeringsnøkkelen til klientsiden.

```
const getAcceptKey = (key) => {
    return base64encode.stringify((sha1(key.concat('258EAFA5-E914-47DA-95CA-C5AB0DC85B11'))));
}
```

Etter denne utvekslingen vil nettleseren sende maskerte WebSocket-meldinger til serveren. Dette kan man se med pakkefangst i Wireshark, for eksempel i den første meldingen i chat:



```
4140 44.926907    127.0.0.1    127.0.0.1    WebSoc…    54 WebSocket Text [FIN] [MASKED]
4141 44.926920    127.0.0.1    127.0.0.1    TCP        44 3031 → 53432 [ACK] Seq=130 Ack=589 Win=2619648 Len=0
4142 44.928145    127.0.0.1    127.0.0.1    WebSoc…    50 WebSocket Text [FIN]
4143 44.928168    127.0.0.1    127.0.0.1    TCP        44 53432 → 3031 [ACK] Seq=589 Ack=136 Win=2619648 Len=0
```



```
WebSocket
    1... .... = Fin: True
    .000 .... = Reserved: 0x0
    .... 0001 = Opcode: Text (1)
    1... .... = Mask: True
    .000 1011 = Payload length: 11
    Masking-Key: 43810877
    Masked payload
    Payload
Line-based text data (1 lines)
0000  02 00 00 00 45 00 00 39  2d 97 40 00 80 06 00 00   ····E··9 -·@·····
0010  7f 00 00 01 7f 00 00 01  d0 b8 0b d7 07 57 9c 11   ········ ·····W··
0020  7f 2d 7c dc 50 18 27 f9  4c bb 00 00 81 8b 43 81   ·-|·P·'· L·····C·
0030  08 77 2b ee 7f 57 22 f3  6d 57 3a ee 7d            ·w+··W"· mW:·}
```

Når nettleseren sender nye meldinger, vil serveren motta meldingene og videresende til registrerte WebSocket-tilkoblinger, slik at alle som er tilkoblet mottar meldingen. Serveren demaskerer og sender meldingen umaskert til alle som er tilkoblet.

```
} else {
    console.log(data);
    console.log(connection.remoteAddress);

    if (data.readUInt8( offset: 0) === 136) {
        connection.end();
    }
    else {
        sendToAllConnections(decryptClientMessage(data));
    }
}
```

Den maskerte meldingen demaskeres med metoden decryptClientMessage:

```
const decryptClientMessage = (encrypted) => {
    let bytes = Buffer.from(encrypted);

    let length = bytes[1] & 127;
    let maskStart = 2;
    let dataStart = maskStart + 4;

    let string = "";

    for (let i = dataStart; i < dataStart + length; i++) {
        let byte = bytes[i] ^ bytes[maskStart + ((i - dataStart) % 4)];
        string += String.fromCharCode(byte);
    }

    return string;
}
```

Før metoden convertToWebSocketMessage konverterer meldingen til ett Buffer-objekt.

```
const convertToWebSocketMessage = (message) => {
    const b = Buffer.alloc( size: message.length + 2);
    b.writeUInt8( value: 129,  offset: 0);
    b.writeUInt8(message.length,  offset: 1);
    b.write(message,  offset: 2);

    return b;
}
```

4. Sammendrag

Oppgaven var grei å løse. Kan i begynnelsen virke litt utfordrende når man begynner med bits og bytes, men ellers overkommelig. Gruppen fikk god utbytte fra øvingen da vi fikk mer kjennskap til hvordan WebSockets og maskering fungerer.