LØSNINGSFORSLAG; INF-240
Våren 2002

OPPGAVE 1

1. Transmisjonstiden T for en pakke er tiden det tar å klokke alle bit i en pakke ut på linken. Med en nominell datarate C = 10^{-6} Sek og en pakkelengde på L bit får vi følgende:

   Tdata = L/C Sek = $10^4/10^6 = 10^{-2}$ Sek
   Tack = $10^2/10^6 = 10^{-4}$ Sek

2. Propagasjonstiden over linken er Tp = 10^{-2} Sek.
   Tiden T fra A starter å sende første bit av en pakke til siste bit er mottatt av B blir:

   T = Tdata + Tp = 10^{-2} Sek + 10^{-2} Sek = 2* 10^{-2} Sek

3. Prosesseringstiden Tpros for en pakke, Data eller Ack, er satt lik 10^{-6} Sek.

A kan ikke sende en ny pakke før kvitteringen for foregående pakke er mottatt og prosessert.
Tiden som medgår her, Ttot, blir:

Ttot = Tdata + Tp + Tpros + Tack + Tp + Tpros = 10^{-2} +10^{-2} +10^{-6} +10^{-4} +10^{-2} +10^{-6}

Ttot ~ 3 *10^{-2} Sek

4. Utnyttelsesgraden for linken er den fraksjonen av linktiden som medgår til overføringen av data pakker. Eventuelt kan vi korrigere for den overhead pakkehodet representerer. Vi ser bort fra det her. Ttot er tiden mellom starten av to påfølgende datapakke transmisjoner. Tdata er transmisjonstiden for en datapakke.

   Utnyttelsesgrad U = (10^{-2} /3*10^{-2}) * 100 % = 33 %

5. I en reell situasjon vil vi oppleve transmisjonsfeil, på grunn av støy og liknende, med det resultat at både Data-pakker og Ack-pakker kan bli forkastet på grunn av feil. I begge tilfeller vil Sendersiden ikke motta noen kvittering, og bli blokkert om vi ikke benytter en klokke (timer). Første tiltak er derfor å innføre en klokke på sendersiden, som settes i det en pakke blir sendt. Dersom kvittering uteblir, vil tiden utløpe, og sendersiden retransmitterer pakken. Anta nå at det er kvitteringen som er blitt ødelagt på grunn av transmisjonsfeil. Når sendersiden retransmitterer pakken, vil mottakersiden ha fått både originalen og retransmisjonen, og vil kunne oppfatte den retransmitterte pakken som en ny og ikke en kopi av originalen. Vi må derfor utstyre hver datapakke med et sekvensno, slik at mottakersiden kan skille mellom originaler og kopier. Sender og mottakersidne må derfor holde greie på hvilket sekvensno som er kvittert og hvilket som er i bruk.

6. Flere applikasjoner skal kunne benytte linktjenesten samtidig, og linklaget må derfor utstyres med Service Aksess Punkter (SAP). Videre forutsetter vi 2 pakketyper, Data og Ack, men disse kan selvsagt kombineres.

Datapakke

Data	Sekv.no.	SAP-Mot	SAP-Send	Data	CRC

Ack

Ack	Sekv.no.	SAP-Mot	SAP-Send

Siden vi bare kan ha en utestående pakke pr SAP, skulle et bit for sekvensno klare seg.
SAP-feltene kan være fra 3 til 8 bit, avhengig av behovet.
CRC-feltet kan mest hensiktsmessig være på 32 bit.

Datafeltet må romme maks pakke størrelse, for eks 10.000 bit.

7. Det er meste hensiktsmessig å starte klokke i det siste bit av pakke er klokket ut på linken. Vi kan da benytte en time-out verdi som er uavhengig av pakkelengden. En fornuftig time-out verdi må være slik at for eks. 95 % av kvitteringene mottas og prosesseres før tiden utløper. Antar vi at denne tiden er rimelig konstant, får vi at:
   Ttimer > Tp + Tpros + Tack + Tp + Tpros = $2 * 10^{-2}$ Sek

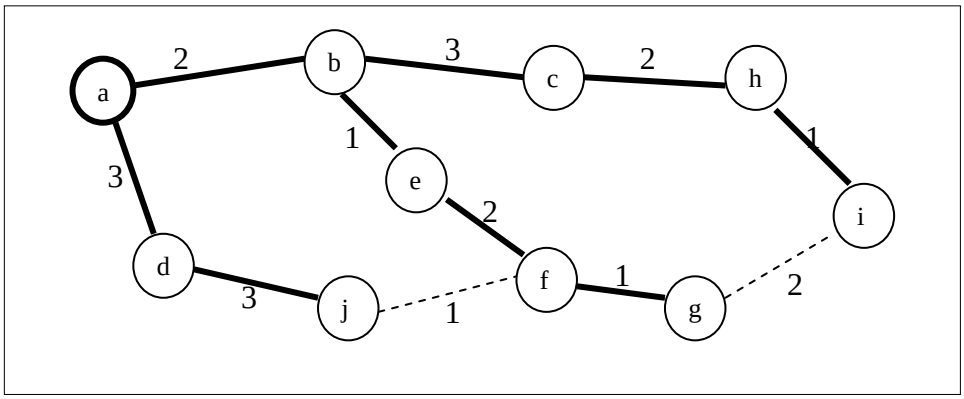
8. Sannsynligheten Pdata for feilfri overføring av en datapakke med lengde L og med en bitfeilsannsynlighet p er:
   Pdata = $(1 - p)^L \sim 1 - L*p = 1 - 10^4 * 10^{-5} = 1 - 0.1 = 0.9$
   Pack ~$1 - L*p$ $1 - 10^2 * 10^{-5} = 1 - 10^{-3} \sim 1$

9. Siden sendersiden bare utnytter 1/3 av kanaltiden, vil få en betydelig forbedring i ytelsen ved å innføre glidende vindu. Sendersiden må som et minimum kunne ha tre utestående pakker for å kunne sende kontinuerlig. Sender og mottakervinduene bør være tilnærmet like store, og summen av dem lik eller mindre enn størrelsen på sekvensnummer rommet.

Benytter vi binære verdier, bør sender og mottaker som minimum ha en vindu- størrelse på 4, og sekvens-nummer rommet være på minimum 4 + 4 = 8, altså 3 bit for sekvensnummerering.

OPPGAVE 2

1. En linktilstandpakke fra node a kringkastes til alle andre noder i nettet. Den vil inneholde hvem som er naboer til a og en kostfaktor for hver link ut fra a. Kostfaktoren for en link kan være basert på forsinkelsen over linken, kombinasjonen av nominell kapasitet og belastning, etc.
2. Minimum kost treet beregnet ut fra node a gir flere alternative veier for å nå flere noder. En strategi kan da være å spre trafikken, slik at vi utnytter den samlede kapasitet i nettet best mulig. Vi kan også kombinere dette med last-deling, slik at belastningen blir fordelt ut over nettet



Vi har markert den delen av korteste-vei treet som gir god spredning i trafikkbildet. Avhengig av fordelingen av trafikken over disse løypene, kan vi i tilfelle den midterste løypa er lite belastet, benytte den for å nå node i og/eller node j.

3. Fremsendingstabellen for node a vil se slik ut:

Node	Neste node	Link
a	0	
b	b	1
c	b	1
d	d	2
e	b	1
f	b	1
g	b	1
h	b	1
i	b	1
j	d	2
j	b	1

4. Dette er alt svart på i deloppgave 1 og 2.

5. Node a vil sende sin distansevektor bare til nabonodene b og d. Den vi se slik ut:

Node	Kost
a	0
b	2
c	5
d	3
e	3
f	5
g	6
h	7
i	8
j	6

6. Jeg ville ha valgt linktilstands metoden. Da har alle noder full oversikt over topologien i nettet til enhver tid. Fordi linktilstandsinformasjonen kringkastes til alle nodene, vil endringer i nettet oppdages raskere av alle nodene, og derved oppdatere rutetabellene raskere, enn distansevektor metoden. I distansevektormetoden vil det ta tid før endringer i den ene enden av nettet har forplantet seg ut til noder i den andre enden av nettet. I dette tidsrommet vil nodene ha ufullstendig ruteinformasjon, og vi vil risikere at det kan oppstå midlertidige ruteløkker i nettet, med de uheldige konsekvenser det kan få for ytelesen.

OPPGAVE 3: TCP mekanismer.

a) Mottakeren inkluderer størrelsen på det annonserte vinduet (”advertised window”) i ACK-meldingen til senderen (som i dette tilfellet er denne lik 0). Selv om senderen i dette tilfelle ikke kan sende ordinære data-pakker, vil den kunne sende små ”sonderingsmeldinger” til mottakeren med jevne mellomrom (dvs. etter en timeout) for å finne ut når det annonserte vinduet igjen blir større enn 0. Hvis mottakers ACK-melding med annonsering av et større vindu blir borte, vil en senere sonderingsmelding fra sender resultere i en duplikat ACK-melding.

Hvis ansvaret for å holde orden på hvorvidt vindusstørrelsen har forandret seg fra 0 til noe større blir skiftet fra sender til mottaker, vil mottakeren trenge en timer for å håndtere retransmisjon av ACK-meldinger (forhåpentligvis etter hvert med annonsert vindu større

enn 0). Dette må foregå inntil mottakeren er i stand til å verifisere at ACK-meldingen er blitt mottatt av senderen.

Det som her er en mulig ulempe, er at mottageren bare får bekreftelse på at senderen har mottatt ACK-meldingen ved at nye data ankommer. Dette betyr at hvis forbindelsen blir ledig (idle), vil mottageren sløse med tida.

b) Sekvensnummeret som hver side vil starte sin byte-nummerering fra, utveksles i etableringsfasen gjennom et tre-veis håndtrykk. Det initielle sekvensno (ISN) er vanligvis avledet av en hardware klokke, og vil ha en startverdi mellom 0 og $2^{32}-1$. Avhengig av initialverdien og antall bytes som overføres, vil det alltid være en sjanse for at "wrap-around".

c) Det annonserte vinduet bør være stort nok til å "holde røret fullt":
Forsinkelsen (RTT) * båndbredde er her: 100ms * 100Mb/s = 10Mb = 1.25MB med data. Dette krever minimum 21 bits (2**21 = 2.097.152) for det annonserte vinduet. Med like store sender og mottaker vinduer, må sekvensnummer rommet være større enn summen av disse to vinduene. I TCP sammenheng er annonsert vindu beskrevet av 16 bit, mens sekvens numrne er beskrevet av 32 bit. Så her er denne betingelsen alltid oppfylt.
Sekvensnummer-feltet må ikke "wrappe" rundt i løpet av maksimum levetid for segmentet. Da er det en viss sannsynlighet for at mottaker kan motta to pakker med overlappende sekvensnummere. I morgendagens nett med ekstremt høye båndbredder, vil dette kunne bli et problem. TCP må da modifiseres for å tillate "wrap-around".
Usikkerheter:
Båndbredden er fast definert av nettverks-komponentene. RTT er mindre presis, vil bli påvirket av trafikkbelastning, størrelsen på nettverket, og lengden av ruten gjennom nettet. Den mest usikre faktoren er antageligvis den maksimale levetiden for et segment (MSG: "Maximum Segment Lifetime"). Denne størrelsen avhenger bl.a. av slike ting som størrelsen og kompleksiteten av nettverket, samt hvor lang tid det tar å løse opp løkker mellom rutere og lignende.

d) Hvis en SYN pakke bare er et duplikat, så vil dets ISN (Initial Sequence Number) være lik den forrige ISN. Hvis SYN-pakken ikke er et duplikat, og ISN verdier er klokke-generert, så vil den andre SYN-pakkens ISN være forskjellig fra den forrige.

Vi antar i det følgende at mottageren har en entydig IP-adresse. (dvs. ikke er "multi-homed"). La <raddr, rport> være den fjerne avsenderen, og la lport være den lokale porten. Vi antar at det eksisterer en tabell T som indekseres av <lport, raddr, rport>, og som (blant annet) inneholder datafeltene lISN og rISN svarende til lokal og fjern ISN. En mulig speudokode ved mottak av en SYN-pakke, vil da kunne være som følger:

```
if  (tilkoplinger til lport ikke aksepteres)
    Send RESET  pakke
else if (det ikke er noe registrering i T for <lport, raddr, rport>) //ny SYN
    Put <lport, raddr, rport> inn i tabellen,
    Set rISN til å være den mottatte pakkens ISN,
    Set lISN til å være vår egen ISN,
    Send ACK svar,
    Marker at forbindelsen er i tilstanden SYN_RECD,
else if (T[<lport, raddr, rport>] allerede eksisterer)
```

```
If (ISN i innkommende pakke er lik rISDN i tabellen)
    // SYN er duplikat, ignorer pakken
else
     Send RESET pakke til <raddr, rport>
```

e) Timeout-er indikerer at nettet er i ferd med å gå i metning og at man bør heller sende færre pakker i stedet for flere. Exponential backoff" gir umiddelbart nettet dobbelt så lang tid til å levere pakker (selv om en enkelt lineær backoff ville gjort det samme), og teknikken medfører også rask tilpasning til enda lengre forsinkelser. Denne teknikken vil mao. i teorien lett kunne tillempes markante økninger i RTT uten å belaste allerede hardt kjørte rutere ytterligere. Exponential backoff har størst effekt når RTT har økt radikalt mye; enten pga. metning eller rekonfigurering av nettet eller når metoden benyttes for "polling" av nettet for å finne den initielle RTT-en..

f) Når TCP mottar en ACK, betyr dette at en tidligere sendt pakke har forlatt nettet, hvilket i sin tur innebærer at det er "trygt" å sende en ny pakke ut på nettet. Ved å benytte ACK-er for å kontrollere hastigheten som pakker sendes ut med, oppnås det vi kaller selv-klokking. Uteblivelse av ACK-er tolkes som signal om at nettet kan begynne å gå i metning.

TCP sin strategi er mao.:

- å kontroller metning når det skjer
- å øke belastningen helt til metning detekteres, trekke seg tilbake igjen, for så å øke på nytt

For å bestemme inisielt tilgjengelig kapasitet og for å avføle og justere endringer i kapasiteten på nettet, benytter TCP tre forskjellige teknikker. Disse kalles hhv.: "Additive Increase/Multiplicative Decrease", "Slow Start" og "Fast Retransmit and Fast Recovery". Alle teknikkene benyttes som et hele.

Her er det opp til kandidatene å utbrodere…

g.1) RED og DECbit forsøker å forutsi når nettet holder på å gå i metning, og så redusere senderaten akkurat før pakker går tapt. Dette blir altså å *unngå* metningsproblemer i stedet for metnings*kontroll* slik dette er beskrevet i punkt f) ovenfor. Disse teknikkene er begge "ruter-sentriske". DECbit mekanismen går i hovedsak ut på å merke pakker når ruteren begynner å merke at nettet går i metning. Avsender vil motta opplysninger om denne merkingen via ACK-meldingene fra mottagende endesystem, og vil redusere senderaten hvis et predefinert antall "merkinger" observeres. RED gatewayer, derimot, dropper selv pakker med en viss sannsynlighet etter å ha observert tendenser til metning ved å se på ankomstraten av pakker til gatewayen. DECbit og RED er detaljert beskrevet på s.475 – 482 i boka, og utdypes ikke videre her.

g.2) Merking av en pakke gjør at endesystemene kan tilpasse seg mer effektivt til metnings-situasjoner – de kan være i stand til å unngå tap (og timeouts) fullstendig ved å minske sine sende-rater. Imidlertid må transport-protokollene i dette tilfelle modifiseres for å kunne forstå og ta høyde for de metnings-bitene som settes. Dropping av pakker leder til timeouts på sendersiden, og vil derfor være mindre effektivt, men eksisterende protokoller (slik som TCP) trenger ikke å modifiseres for å benytte RED. Dessuten er dropping en måte å tøyle en sender som oppfører seg dårlig.

OPPGAVE 4

1. IP laget leverer en ”best-effort” tjeneste til transportlaget. Det vil si at pakker kan komme frem i en annen rekkefølge enn den de ble sendt i, og kanskje kan bli dublerte. Vi har antatt at alle pakker kommer frem, men at de kan ha variabel gangtid gjennom nettet. Transportlaget vil da kunne ha tre oppgaver: skille dubletter fra originaler, ordne pakkene i riktig sekvens, og bufre opp en del pakker før de blir levert til utspilling – for å ta hånd om variasjonen i gangtiden gjennom nettet. Pakker skal ikke kvitteres.
2. Når utspillingsprosessen starter, må den ”mates” med en pakke med digitale tale hvert 20. mSek, for ikke å få ”hull” i talestrømmen. Den variable gangtiden gjennom nettet må kompenseres ved å bufre opp et antall talepakker før utspillingen starter. Antall bufrede pakker må stå i forhold til variasjonen i gangtiden gjennom nettet. Er variasjonen i gangtiden 50 mSek, må vi bufre opp minimum 3 pakker a’ 20 mSek tale hver, før utspillingen starter.
3. Det viktigste feltet i transportlagshodet vil være:
   Et sekvens no. For nummerering av pakker; antall sekvensno må være vesentlig større enn 3, slik at gamle og nye sekvensno ikke kan forveksles.
   Dersom vi har flere applikasjonsprosess som bruker, så er **SAP** nødvendig.
   Siden hver pakke har konstant lengde og tilsvarer 20 mSek tale, er det unødvendig med tidsstempling av pakker. Sekvensnummeret i pakken er godt nok.
   Lengdeangivelse er heller ikke nødvendig.

T-PDU

T-port-dest	T-port-src	Seq-no	T-Data

4. Sendersidens prosedyrer:

Det jeg hadde i tankene her, var i første rekke at studentene skulle spesifisere parametrene i prosedyrekallene, deretter litt om hva hver prosedyre gjør. Vi må her gi rom for mange ulike løsninger!!
Jeg antar her et socket-liknende grensesnitt mellom applikasjonsprosessen og transportlaget, og at det er satt opp en assosiasjon mellom T-Port-src og T-Port-dest, og at alle assosiasjonsattributter ligger i en struktur a’ la TCB (Transport Control Block)

T-Send(sock, *Data, Data-len; return = OK)
IP-Send(IP-addr-dest, IP-SAP, *TPDU, TPDU-len)

```
T-Send (( sock,  *Data, Data-len);
          (sequence-no := sequence-no + 1;
           Format-TPDU (dest-port, src-port, sequence-no, *Data, Data-len);
           IP-Send (IP-addr-dest, IP-SAP, *TPDU, TPDU-len);
           );
         );

IP-Send (IP-addr-dest, IP-SAP, *TPDU, TPDU-len;
          Format-IP (IP-addr-dest, IP-addr-src, IP-SAP, *TPDU, TPDU-len);
          Send packet;
          );
```

Mottakersidens prosedyrer:
Vi lar transportlaget ansvaret for å legge pakker i riktig sekvens inn i utspillingsbufferet, og vi lar applikasjonsprosessen hente ut en pakke hvert 20. mSek.

T-Ipop (*TPDU, TPDU-len)
T-Apop (*Data, Data-len)
T-Ipop ((*TPDU, TPDU-len;
Les ut sekvens-no
IF har jeg sett dette før THEN kast pakken
ELSE (lagr nytt sekvens-no
sett pakken inn i utspillingsbufferet i riktig rekkefølge
);
);
);

T-Apop (Data, Data-len);

Denne prosedyren invokeres av applikasjonsprosessen hvert 20. mSek, og henter ut en blokk med digitale data for direkte utspilling.