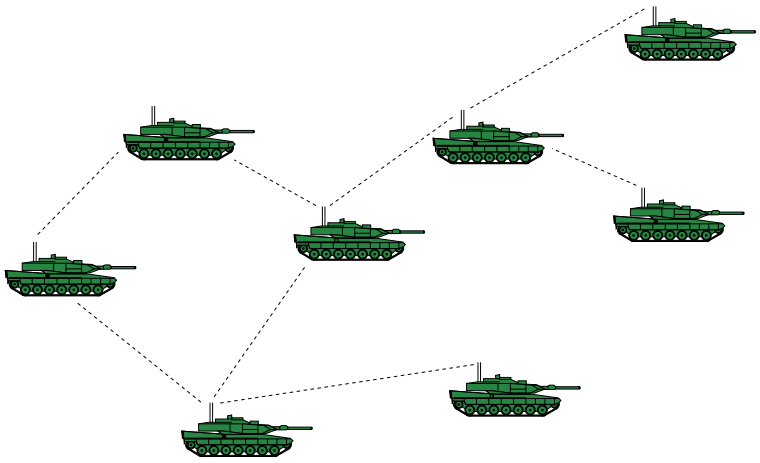
FOI-R--0527--SE
Juni 2002
ISSN 1650-1942
Teknisk rapport

Katarina Persson

TCP/IP i taktiska ad hoc-nät



Ledningssystem
Box 1165
581 11 LINKÖPING

TOTALFÖRSVAREST FORSKNINGSINSTITUT-FOI
Ledningssystem
Box 1165
581 11 LINKÖPING

FOI-R- -0527- -SE
Juni 2002
ISSN 1650-1942

Teknisk rapport

Katarina Persson

TCP/IP i taktiska ad hoc-nät

Utgivare Totalförsvarest Forskningsinstitut - FOI Ledningssystem Box 1165 581 11 LINKÖPING	**Rapportnummer, ISRN** FOI-R--0527--SE	**Klassificering** Teknisk rapport
	Forskningsområde 4. Spaning och ledning	
	Månad, år Juni 2002	**Projektnummer** E7035
	Verksamhetsgren 5. Uppdragsfinansierad forskning	
	Delområde 41. Ledning med samband, telekom och it-system	
Författare Katarina Persson	**Projektledare** Jan Nilsson	
	Godkänd av Christian Jönsson	
	Uppdragsgivare/kundbeteckning FMV- Försvarets materielverk	
	Teknisk och/eller vetenskapligt ansvarig Jimmi Grönkvist	
Rapportens titel TCP/IP i taktiska ad hoc-nät		

Sammanfattning

TCP (Transmission Control Protocol) är ett transportprotokoll och används för Internet. TCP är utvecklat för fasta nät och i radionät blir det problem bl.a. p.g.a. hög bitfelshalt. Tappade paket kan feltolkas av TCP som att de beror på överbelastning, och därmed sänks datatakten i onödan.

Mobila ad hoc-nät består av rörliga enheter som reläar information mellan varandra. Avbrott kan uppstå då enheterna separeras eller omorganiseras. TCP sänker då datatakten vilket leder till minskad kapacitet då förbindelsen återupptas. Vi vill således modifiera TCP för att kunna skilja mellan olika fel, och anpassa datatakten därefter.

En förbindelse i ett ad hoc-nät har modellerats, och avbrottsfel och tappade paket har simulerats. Resultat visar att vår modell inte uppvisar så stora problem som förväntats, men det beror på att vi bl.a. buffrar paket under avbrotten och har låga fördröjningar i nätet.

Tidigare förslag till modifieringar av TCP presenteras och en enkel modifiering simuleras, då vi ökar datatakten hastigt efter ett avbrott. Det visar sig att även en relativt enkel modifiering kan påverka kapaciteten positivt och slutsatsen är att effektiviseringar bör göras.

Nyckelord TCP, ad hoc-nät, kapacitet	
Övriga bibliografiska uppgifter	**Språk Svenska**
Rapporten har även getts ut av Linköpings Universitet, LiTH-ISY-EX-3206-2002	
ISSN 1650-1942	**Antal sidor 68 s.**
Distribution enligt missiv	**Pris Enligt Prislista**

Issuing organization	Report number, ISRN	Report type
FOI- Swedish Defence Research Agency Command and Control Systems P.O. Box 1165 SE-581 11 LINKÖPING SWEDEN	FOI-R- -0527- -SE	Teknisk rapport
	Reasearch area code 4. C^4ISR	
	Month year June 2002	**Project No.** E7035
	Customers code 5. Commissioned Research	
	Sub area code 41. C^4I	

Author/s	Project manager
Katarina Persson	Jan Nilsson
	Approved by Christian Jönsson
	Sponsoring Agency FMV- Defence Material Administration
	Scientifically and technically responsible Jimmi Grönkvist

Report title

TCP/IP in tactical ad hoc networks

Abstract

TCP (Transmission Control Protocol) is a transport protocol designed for the wired Internet. In wireless networks packet losses occur more frequently due to the unreliability of the physical link. The main problem is that TCP treats the losses as congestion, which may lead to an unnecessary low throughput.

Ad hoc networks are multihop wireless networks of mobile nodes, where each node can allow other packets to pass through it. Topology changes often occur and may lead to packet losses and delays, which TCP misinterprets as congestion. We want to modify TCP to recognize the differences between link failure and congestion to improve the capacity.

In our model we have built a connection in an ad hoc network where packet losses and partitions can be made. Simulation experiments show that we didn't get the problem to the extent we expected. This can be explained by low delays and because we buffered the packets during link failure.

A simple modification of TCP was made and simulated, and showed that an improvement of performance is possible. More research should be done to make a modification of TCP that would further improve the throughput.

Keywords

TCP, ad hoc networks, capacity

Further bibliographic information	Language Swedish
The report has also been published at Linköping University, LiTH-ISY-EX-3206-2002	
ISSN 1650-1942	**Pages 68 p.**
	Price Acc. to pricelist

Notation

Förkortningar

ACK	Acknowledgement, bekräftelse.
ATCP	Ad Hoc TCP.
cwnd	Congestion Window.
DUPACK	Duplicate Acknowledge, multipel bekräftelse.
ECN	Explicit Congestion Notification.
FTP	File Transfer Protocol.
HTTP	Hyper-Text Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol.
RFC	Request For Comment.
RTO	Retransmission Timeout.
RTT	Roundtrip Time.
swnd	Sliding window
TCP	Transmission Control Protocol.
TCP-BuS	TCP Buffering capability & Sequence information.
TCP-F	TCP-feedback.
UDP	User Datagram Protocol

Innehåll

Kapitel 1

Inledning

1.1 Bakgrund

Framtidens militära operationer är beroende av ett fungerande och säkert kommunikationssystem. Man ska kunna kommunicera mellan olika enheter, t.ex. för att överföra positions- och sensorinformation. Ett robust och säkert radionät behövs därför för att kunna kommunicera. För att undvika svaga punkter vill man dessutom inte ha ett alltför starkt beroende av utbyggd infrastruktur och central styrning.

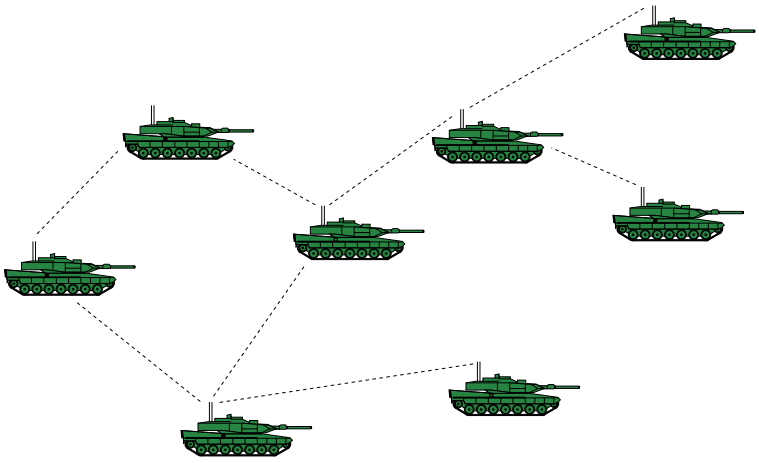
Radionätet ska vara mobilt och snabbt kunna etableras. Det måste vara självläkande då förbindelser bryts eller förändras och ska klara av att enheterna rör sig snabbt genom terrängen. Nätets funktioner bör ej vara beroende av enskilda noder, utan ska klara av att vissa delar slås ut. För att kunna kommunicera med enheter utanför radionätet krävs också kompatibilitet med andra nät.

Dagens svenska militära radionät uppfyller inte dessa krav. En möjlig lösning som är under utveckling är ad hoc-nät.

1.1.1 Ad hoc-nät

Ad hoc är latin och betyder ”för detta ändamål” och används ofta för att ange att något är improviserat. Tolkningen av ett ad hoc-nät är ett nät som förändras kontinuerligt och anpassas till terrängen och situationen.

Ett mobilt ad hoc-nät är uppbyggt av rörliga noder. Någon central enhet behövs inte utan noderna kan fungera både som relästationer där information



Figur 1.1: Ad hoc-nät.

skickas vidare och som sändare/mottagare. Man kan således sända paket via noderna och samtidigt använda dem till applikationer.

I militära sammanhang kan ad hoc-nät användas på slagfält där man snabbt ska etablera ett radionät, se figur 1.1. Man kan även tänka sig att använda ad hoc-nät inom civila nödsituationer då ett radionät måste användas men infrastrukturen är utslagen eller inte finns utbyggd [5]. Forskning pågår även kring ad hoc-nät inom telekomindustrin.

Problemen med ett ad hoc-nät är att nätets struktur ändras kontinuerligt då enheterna rör sig vilket leder till avbrott och problem med kontakt mellan enheterna. Nya vägar för informationen måste hela tiden hittas och ibland kan det hända att ett nät delas så att kontakten bryts helt mellan vissa enheter.

1.1.2 TCP i ad hoc-nät

För att möjliggöra informationsöverföring i ett nät krävs det regler för hur data ska tas emot och hur det ska behandlas. Protokoll är en samling sådana regler och hos alla mottagare och sändare i ett nät finns det flera protokoll med olika uppgifter.

TCP (*Transmission Control Protocol*) är idag ett av de dominerande proto-

kollen för datakommunikation och utgör en del av grunden för Internet. TCP ansvarar för att information som skickas från sändaren tas emot korrekt av mottagaren, och sänder om information som tappas bort. Kravet på att det militära radionätet ska vara kompatibelt med andra nät gör att TCP måste kunna användas även här.

TCP tillhandahåller pålitlig överföring och fungerar bra över fasta nät. Dagens version av TCP utvecklades framförallt under 80-talet och sedan dess har endast mindre förändringar gjorts, grunderna och egenskaperna är desamma [13]. Det innebär tyvärr att TCP är anpassat för fasta uppkopplingar och har vissa problem i trådlösa nät, t.ex. ad hoc-nät.

1.1.3 Problem som kan uppstå

Den stora skillnaden mellan fasta och trådlösa nät är, förutom mobiliteten, förekomsten av fel i överföringen. Fasta nät har låg bitfelsannolikhet och när paketförluster uppkommer beror endast en bråkdel på att paket tappats bort eller förbindelsen avbrutits. Större delen av felen är istället relaterade till överbelastningar och trafikstockningar.

Trådlösa nät är känsliga för yttre störningar och har relativt hög bitfelsannolikhet. Det är vanligt med överföringsfel och att paket inte når mottagaren alls. I ad hoc-nät beror felen dessutom ofta på att länkar försvinner och nya upprättas. Ibland bryts kontakten helt. Detta innebär också att kapaciteten på en ad hoc-länk varierar och att trafiklasten därför måste anpassas hela tiden för att inte överbelasta nätet.

Tyvärr skiljer inte TCP på fel som beror på överbelastning och fel som uppkommer p.g.a. hög bitfelssannolikhet eller avbrott utan reagerar genom på antagandet att felen i de flesta fall beror på överbelastning. TCP reagerar därför genom att sänka datatakten [13]. Detta är mycket olämpligt eftersom det innebär att om vägar ofta ändras blir överföringen över kanalen mycket liten. De krav som ställs på kapaciteten hos radionät uppfylls inte.

1.2 Syfte

Syftet med den här rapporten är att sammanställa, beskriva och undersöka problemen med TCP i taktiska ad hoc-nät. Kapacitet ska studeras över ad hoc-nät

med olika egenskaper. En litteraturstudie över möjliga förbättringar och modifieringar av TCP ska också sammanställas och principen för en modifiering av TCP ska simuleras.

Målet är att undersöka och utvärdera hur TCP fungerar i taktiska ad hoc-nät för att kunna ge en vägledning om vilket protokoll som i framtiden bör användas.

1.3 Rapporten i sammandrag

För att kunna förstå problemet med TCP i taktiska ad hoc-nät krävs en inblick i hur protokollstacken och TCP fungerar. I kapitel 2 beskrivs därför detta översiktligt, för att vi sedan ska kunna presentera problemet i detalj. I detta kapitel avgränsar vi också problemet.

Nästa kapitel behandlar TCP-protokollet i detalj, dess funktioner och egenskaper. I kapitel 4 beskrivs tidigare arbeten inom området med att modifiera TCP-protokollet, och flera modeller beskrivs. I avsnitt 4.5 görs en jämförelse mellan de olika modifieringarna.

En redogörelse av hur ett ad hoc-nät kan implementeras samt antaganden och begränsningar görs i kapitel 5 och i kapitel 6 beskrivs simuleringen av en variant av TCP i ett ad hoc-nät samt principen för en enkel modifiering av TCP och resultatet av denna.

I kapitel 7 summeras resultaten och en diskussion kring slutsatser görs. Slutligen avslutas rapporten med en beskrivning av vilka utvidgningar inom området som bör göras.

Kapitel 2

Teoretisk bakgrund och problemdefinition

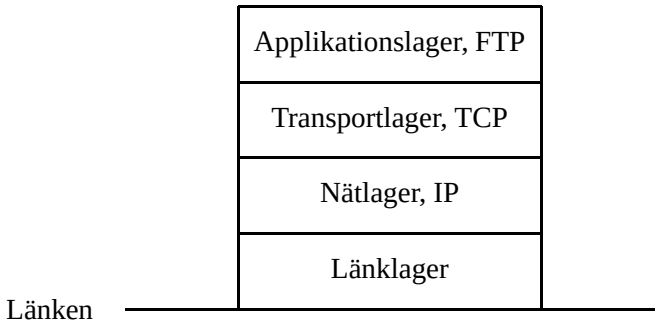
En av grundförutsättningarna för att kunna kommunicera är naturligtvis att kunna förstå informationen som överförs. På samma sätt som man lyssnar efter morsesignaler och tolkar dessa måste informationen som sänds över ett radionät behandlas och tolkas.

Idag använder man sig av ett antal protokoll som tar hand om data för att få fram den information vi är ute efter. Man kan jämföra ett protokoll med en samling regler för hur data ska tas om hand. Protokollen har olika ansvarsområden och egenskaper, t.ex. adressering och kodning, och delas upp i olika lager. Två av de vanligaste protokollen är TCP *(Transmission Control Protocol)* och IP *(Internet Protocol)*, och de verkar i två skilda lager.

Vid mottagande och sändande av information passerar data olika lager, och varje lager har ett protokoll. En protokollstack, t.ex. TCP/IP-stacken, är en kombination av fyra olika protokoll, bland annat TCP och IP, från de olika lagren.

2.1 Protokollstacken

TCP/IP-stacken finns på nästan alla maskinvaru- och programplattformar idag och utgör stommen i Internet. Protokollstacken har blivit standard för att bygga leverantörsblandade Internet eller intranät. Utvecklingen av TCP/IP-stacken började i slutet av 1960-talet och har idag nått långt över sina förväntningar [13].



Figur 2.1: TCP/IP-stacken

En viss utveckling och förnyelse har skett under årens lopp men grunderna med de mest påtagliga egenskaperna är desamma som för 20 år sedan. Genom att förstå de grundläggande funktionerna och egenskaperna hos protokollstacken kan vi undersöka hur de passar in i nya nätsarkitekturer som trådlösa nät och ad hoc-nät.

TCP/IP-arkitekturen kan delas upp i fyra lager, eller nivåer, där varje nivå har ett eget ansvarsområde, se figur 2.1. Varje lager kommunicerar med angränsande lager uppåt och nedåt i stacken, samt kommunicerar med motsvarande lager hos andra enheter [13].

1. Applikationslagret kontrollerar tillämpningar, t.ex. filöverföring, surfande och e-post. Här finns ett antal olika protokoll, som används beroende på tillämpningen. HTTP (*Hyper-Text Transfer Protocol*) hjälper oss när vi surfar. FTP (*File Transfer Protocol*) används för att skicka filer.

2. Transportlagret ser till att applikationsdata från sändaren som nått sin destination även når avsedd applikation hos mottagaren. TCP (*Transmission Control Protocol*) och UDP (*User Datagram Protocol*) är två vanliga nätprotokoll, som skiljer sig åt på några punkter. TCP tillhandahåller pålitlig överföring av data i korrekt ordning till mottagaren. Detta sker genom att TCP tar emot applikationsdata och delar upp i lagom stora paket som numreras, så att de kan sättas ihop igen i rätt ordning hos mottagaren. Därefter tar TCP emot bekräftelser från mottagaren på att paket nått fram.

UDP däremot kontrollerar inte att informationen kommer fram, och är därför enklare. UDP delar inte heller upp data i mindre block, utan skickar paketen som de är.

3. Nätlagret sköter adresseringen av de olika datapaketen, så att de skickas till rätt mottagare. IP (*Internet Protocol*) är det vanligaste nätprotokollet och krävs för att tillhöra Internet. Protokollet avgör också vilken väg paketet ska skickas, s.k. routning. Protokollet kontrollerar dock varken om paket som sänds kommer fram till mottagaren eller i vilken ordning paketen tas emot.

4. Länklagret består av hårdvaran som förbinder sändaren och mottagaren, och specificerar hur data skall skickas genom nätet eller över länken. I radionät sker även kodning, modulation och avkodning i länklagret.

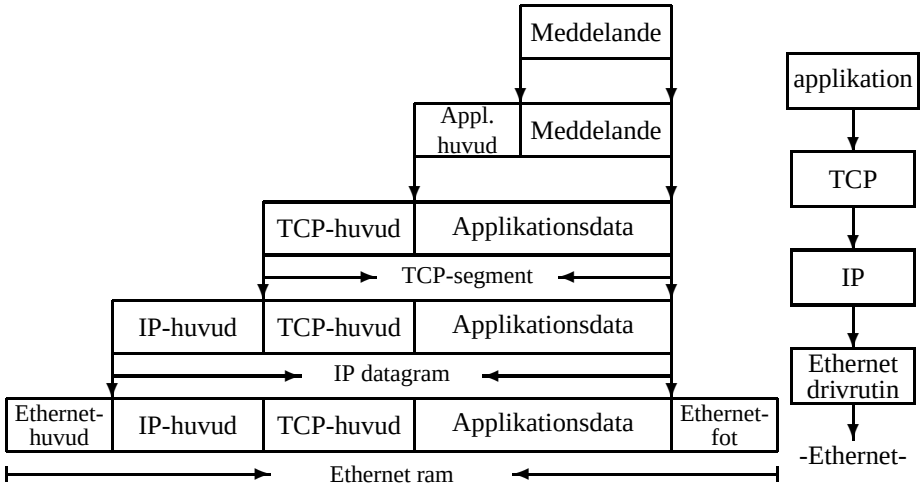
När en applikation sänder data med hjälp av TCP, skickas datapaketet ner genom stacken och ut på kanalen i en ström, se figur 2.2. Varje lager lägger till lite information till applikationsdatat, s.k. *huvuden*. IP-lagret lägger bl.a. till destinationsadress och källadress för att kunna skicka paketet. TCP lägger till ett sekvensnummer m.m. för att kunna kontrollera att paketen kan sättas ihop i rätt ordning hos mottagaren. Hos mottagaren skickas sedan segmentet nerifrån och upp i stacken och avkapslas i sin tur på varje nivå. I den här studien ska vi fokusera på TCP.

2.2 Egenskaper och funktioner hos TCP

TCP ser till så att information överförs mellan två applikationer. Innan de två applikationerna kan överföra data till varandra måste en kontakt mellan de två först skapas, och därefter kan överföringen starta.

TCP delar upp data som ska överföras i lagom stora segment och kapslar in dessa i TCP-huvudet. Den innehåller bl.a. information om i vilken ordning segmenten ska sättas ihop igen hos mottagaren.

TCP bekräftar på mottagarsidan då ett paket har mottagits. Detta görs genom att skicka med ett meddelande om vilket paket som man förväntas ta emot nästa gång. Man märker då om ett paket försvunnit eller kommer fram i fel ordning. Bekräftelsen som skickas kallas ACK (*acknowledgement number*), och finns med i TCP-huvudet.



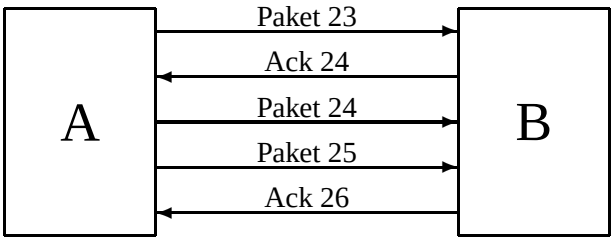
Figur 2.2: Inkapsling av data på väg ner genom stacken.

Exempel: (se figur 2.3) A skickar data till B med sekvensnummer 23. B svarar med ett ACK innehållande nummer 24. A vet då att B har mottagit paket nummer 23 och nu väntar på paket 24. A skickar nu två paket, nummer 24 och nummer 25. B väljer då att endast svara med ett ACK, innehållandes nummer 26. På så sätt vet nu A att både paket 24 och 25 mottagits korrekt.

När dataöverföringen sedan startar anpassas datahastigheten så att inte länken överbelastas. Om paket försvinner och inte kommer fram har TCP flera funktioner som ser till så att informationen som förlorats sänds om.

Om ett enstaka paket försvinner på väg till mottagaren märks detta. Om paketet sedan inte kommer fram som ett av de närmast efterföljande sänds paketet om och datatakten sänks en del.

Exempel: (se figur 2.4), ett exempel på enstaka paket som försvinner: A skickar här paket 35, 36, 37 och 38 till B. B tar först emot paket 35 och skickar ett ACK innehållandes numret på nästa paket som B förväntas ta emot, nämligen 36. Därefter tar B emot paket nummer 37. Paket 36 har således tappats bort eller försenats. B skickar då åter ACK 36, ef-



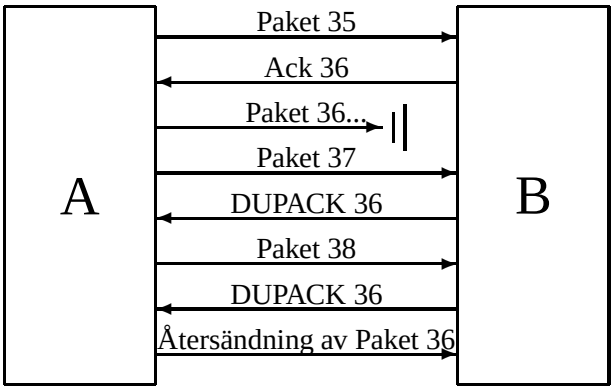
Figur 2.3: Exempel: numrering av paket och bekräftelser

tersom det är paketet som man väntar på. Flera ACK som innehåller samma nummer brukar man kalla DUPACK. När paket 38 därefter mottages skickas återigen ett ACK med nummer 36. A har nu förstått att något är fel eftersom man har mottagit **tre DUPACK**, d.v.s. tre ACK med samma nummer, och sänder därför om paket 36. Datatakten sänks inte så mycket eftersom man vet att en del paket fortfarande kommer fram och att det således inte är avbrott på länken. Om nästa paket till B istället hade varit nummer 36, hade B svarat med ACK 39. Därmed hade alla sända paket bekräftats och överföringen hade fortsatt som förut.

Om fler paket försvinner och inga ACK når sändaren eller om det tar alltför lång tid innan tre DUPACK tas emot kan en **timeout** inträffa. Sändaren har en klocka som mäter hur lång tid det tar tills dess bekräftelsen, (ACK:et), tas emot. Tiden det normalt tar för en bekräftelse att nå sändaren efter att ett paket sänds kallas RTT, *round-trip time*. Om ett ACK inte når sändaren inom tiden RTO, *retransmission timeout*, tar man timeout i överföringen. RTO beräknas utifrån RTT och är ofta

$$RTO = 2 \cdot RTT. \tag{2.1}$$

När en timeout inträffar sänks datatakten betydligt, och man sänder om de paket som inte har bekräftats.



Figur 2.4: Exempel: Enstaka paket som försvinner, mottagande av tre DUPACK samt till sist återsändning av paketet.

2.2.1 Beräkning av RTO

Eftersom tiden det tar för en bekräftelse att normalt nå sändaren varierar ändras också RTO kontinuerligt. Är det en långsam förbindelse är RTO stort, och litet då förbindelsen är snabb. Vi ska nu studera hur tiden RTO bestäms.

Genom att mäta tiden det normalt tar, RTT, (*Round-trip time*), kan RTO beräknas. RTT kan variera mycket beroende på att man t.ex. många gånger fördröjer ACK:ar för att kunna skicka flera samtidigt och tillsammans med data. För att motverka att RTT ändras alltför hastigt uppdateras värdet på RTT *mjukt* med ett lågpassfilter, se ekvation 2.2, där M är den nya mätningen och α är en glömskefaktor och har det rekommenderade värdet $0,9$.

$$newRTT \leftarrow \alpha \cdot RTT + (1 - \alpha)M \tag{2.2}$$

RTT uppdateras varje gång en ny mätning görs, och det nya värdet består till 90% av det gamla värdet och 10% av den nya mätningen.

Därefter kan vi uppskatta den maximala tid vi tillåts vänta på en bekräftelse, RTO.

$$RTO = \beta \cdot RTT \tag{2.3}$$

β är en fördröjningsfaktor med det rekommenderade värdet 2, se ekvation 2.1. Ett problem med den här modellen är att den inte alltid kan följa RTT:s fluktuationer, och därigenom leda till onödiga återsändningar.

Ett annat problem uppstår då vi sänder om ett paket p.g.a. att inte bekräftelsen nått oss. När sedan en bekräftelse når oss kan vi inte veta om det här är bekräftelsen på det första eller andra paketet vi sänt. Det är således högst osäkert att räkna fram ett nytt RTT med dessa värden, och detta undviks därför för paket som sänds om.

2.3 Pålitlig överföring

Att TCP tillhandahåller pålitlig överföring av data bygger på ett antal funktioner [13]. Vi sammanfattar dessa här och beskriver några av dem i detalj i andra delar av rapporten.

- TCP delar upp applikationsdata i lagom stora segment att sända, så att paketen på länkarna inte ska behöva delas upp igen längs vägen. Varje länk har en gräns för hur stora paket som kan hanteras. Ska man sända över en förbindelse, flera länkar, anpassas storleken på paketen efter länken med högst krav, d.v.s. minst paketstorlek, se kapitel 3.1.
- För att kunna veta om ett paket kommit fram korrekt sänds en bekräftelse från mottagaren till sändaren då paketet mottagits. Om paketet tappats sänds det om, se kapitel 2.2.
- TCP kodar informationen i segmentet så att man kan upptäcka om det har smugit sig in ett fel under överföringen. Om det har det slängs segmentet hos mottagaren och sänds sedan om av sändaren.
- TCP-segment kan komma fram i fel ordning. Mottagaren flyttar då om segmenten så att de kommer i rätt ordning igen.
- Ibland genererar IP två paket med samma data till mottagaren. TCP kastar då det ena.

- TCP ser till så att mottagaren inte överbelastas genom att skicka för många paket alltför snabbt. Man anpassar hela tiden trafiklasten på förbindelsen efter kapaciteten på länken, se kapitel 3.4.

2.4 Problemen med TCP i ad hoc-nät

TCP är utvecklat för fasta nät och i radionät kan det uppstå problem. Tidigare studier [4, 6, 8, 11] har visat detta.

Det grundläggande problemet med TCP är att det inte kan skilja mellan olika fel på länken, utan alltid antar att felen beror på överbelastningar i nätet, och därmed sänks datatakten. Detta gör att man är intresserad av att hitta en modell av TCP som kan separera felen och inte minska överföringshastigheten i onödan, och därmed öka kapaciteten hos ad hoc-nät.

- Ett radionät har mycket högre bitfelssannolikhet än ett fast nät, och paket tappas lätt bort. Om paket ofta tappas och timeout inträffar sänds paket om och datatakten sänks markant, vilket innebär en minskning av mängden överförd data.

- I ett ad hoc-nät med mobila noder ändras vägen mellan mottagare och sändare ibland, då noderna rör sig. Om det då tar längre tid än RTO att hitta en ny väg inträffar timeout och datahastigheten sänks ordentligt. När en ny väg sedan hittas går överföringen i början mycket långsamt.

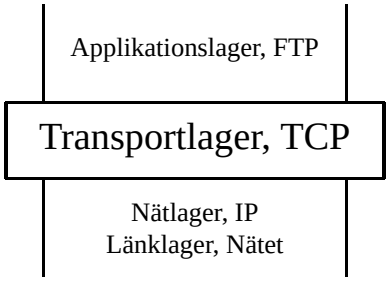
- Ibland kan även nätet splittras under en längre tid utan att kontakt kan upprättas. Sändaren försöker då sända på nytt med ett visst tidsmellanrum. Denna tid ökar då ingen kontakt upprättas men en gräns går dock vid två minuter. Detta kan innebära att då länken väl hittats kan det dröja ytterliggare två minuter innan sändaren upptäcker detta och kan börja sända.

- En del routingprotokoll kan ibland upprätta flera olika vägar mellan två destinationer, för att minimera antalet omräkningar av vägen. Detta kan tyvärr ibland ge upphov till att paket anländer hos mottagaren i fel ordning och genererar flera ACK. Detta kan i sin tur leda till att paket sänds om i onödan.

2.5 Inriktning av studien

Under de senaste åren har man gjort flertalet studier och försök kring att utveckla och modifiera TCP, se [4, 6, 8, 11]. Vi ska i den här studien försöka sammanfatta forskningen inom området och beskriva metoder att minska de eventuella problemen med TCP i taktiska ad hoc-nät.

För att studera hur TCP fungerar i ett ad hoc-nät behöver vi en modell. Vi väljer att studera TCP:s egenskaper över en förbindelse i ett ad hoc-nät. Att studera TCP i ett helt nät skulle bli för komplext och svårt att tyda.



Figur 2.5: Avgränsning

Tjänsten vi använder är filöverföring och applikationsprotokollet är därmed FTP. Vi väljer att skärma av TCP-lagret från de överliggande och underliggande protokollen, se figur 2.5. Detta görs för att få en så ren bild av hur TCP fungerar som möjligt. För att kunna analysera resultaten korrekt och kunna dra slutsatser vill vi ha så få inparametrar som möjligt. Ad hoc-nätets modell beskrivs utförligare i kapitel 5.

Vi vill skapa oss en bild av vad det är som påverkar förbindelsens kapacitet. Hur påverkar tappade paket och länkfel TCP som i sin tur gör så att överföringskapaciteten ändras? Slutligen vill vi genom att modifiera TCP se om det är möjligt att öka kapaciteten hos förbindelsen.

Kapitel 3

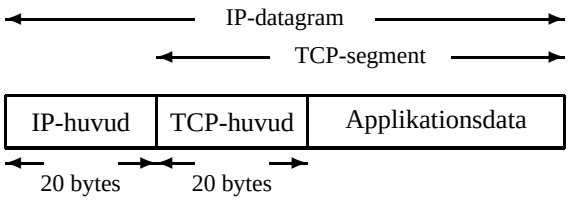
TCP - *Transmission Control Protocol*

3.1 TCP:s huvud

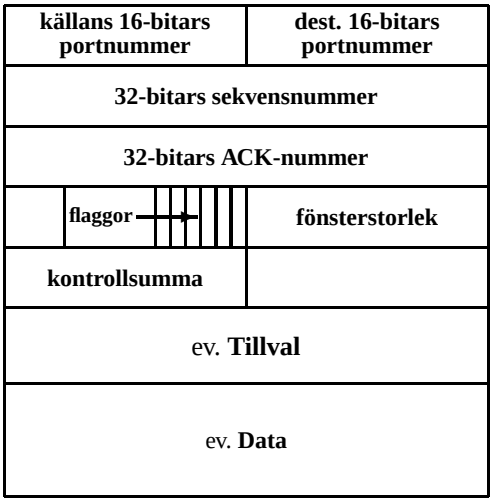
I kapitel 2 beskrevs hur applikationsdata då det ska sändas kapslas in på väg ner genom stacken och ut på länken, se figur 3.1. När paketet sedan tas emot avkapslas informationen istället och vandrar uppåt i stacken genom länklagret, nätlagret, transportlagret och upp till applikationen.

Storleken på IP-huvudet är 20 bytes, och likaså för TCP-huvudet. Vi ska nu studera TCP-huvudet, som innehåller den information som läggs till i transportlagret, se figur 3.2.

TCP-huvudet innehåller källans och destinationens **portnummer** som till-



Figur 3.1: TCP-segmentet inkapslat i IP-datagramet



Figur 3.2: TCP-segmentet

sammans med IP-adresserna i IP-huvudet ger tillräckligt mycket information för att kunna skapa en unik förbindelse. Kombinationen av en IP-adress och ett portnummer kallas ofta *socket*.

Sekvensnumret i TCP-huvudet används för att kunna sätta ihop segmenten i rätt ordning hos mottagaren. **ACK-numret** innehåller nästa pakets sekvensnummer, som sändaren av ACK:et förväntas ta emot. Det innebär att ACK-numret är det *senast korrekt mottagna paketets sekvensnummer plus ett*.

Att sända en bekräftelse, ett ACK, behöver inte kosta något eftersom det alltid finns med i huvudet, och genom att sätta en flagga visar man om ett ACK skickas med eller ej. Huvudet innehåller ett flertal **flaggor** med olika betydelser.

- ACK - *(acknowledgement number)* Flagga som visar om ett ACK-nummer, en bekräftelse av mottaget paket sänds med.
- SYN - *(synchronize sequence number)* Visar att ett SYN-segment sänds, som används för att synkronisera numreringen av segmenten som ska skickas. SYN-segment sänds då en uppkoppling ska påbörjas.
- FIN - *(sender is finished sending data)* Sänds då överföringen är klar från det ena hållet, och den ena parten vill avsluta uppkopplingen.

En **kontrollsumma** skickas också med. Det är för att kunna kontrollera att informationen som överförs är korrekt. Över radiokanaler krävs i allmänhet ytterligare kodning eftersom kanalen är brusigare och därför kodas också paketet i länklagret innan det sänds.

För att veta hur stora segment som kan skickas över en förbindelse används MSS, (*Maximum Segment Size*), som är det största hela segmentet data som kan skickas över en länk. MSS skickas med i fältet **tillval** i TCP-huvudet då kontakt ska upprättas. Generellt sett är det bättre ju större segment man kan skicka, men ytterligare uppdelningar av paketet sänker å andra sidan kapaciteten. När en kontakt har etablerats väljer man således MSS till den största segmentstorleken som kan transporteras hela vägen utan att delas upp ytterligare.

3.2 Upprättande av kommunikation

Innan informationsutbyte mellan två stationer kan ske måste kontakt upprättas. Det hela kan liknas med ett vanligt samtal. Innan information börjar överföras krävs ett par korta hälsningsfraser.

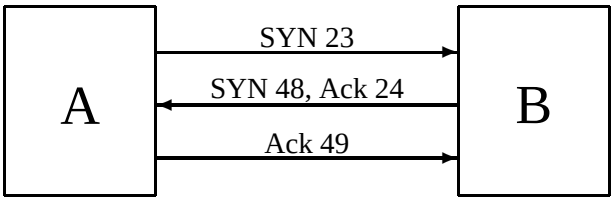
TCP:s väg för att upprätta kontakt brukar kallas *tre-vägs handskakning*, eftersom den sker i tre steg, se figur 3.3.

1. Den del som vill upprätta en förbindelse, t.ex. klienten, börjar med att sända ett segment som specificerar vilken port som skall användas hos mottagaren, här kallad servern. Klienten sänder också med det initiala sekvensnumret, och har SYN-flaggan satt, för att kunna numrera paketen som ska skickas till servern.

2. Servern svarar då med att skicka ett eget SYN-segment, innehållandes serverns initiala sekvensnummer. Servern svarar också på det förra meddelandet genom att skicka med en bekräftelse, ett ACK.

3. Klienten bekräftar meddelandet genom att skicka ett ACK innehållandes serverns initiala sekvensnummer plus ett.

Därefter kan information börja överföras. Datasegment skickas och bekräftas. Eftersom bekräftelserna som skickas är mycket små och mest innehåller huvud brukar man fördröja sändningen av ACK:et och vänta på att det ska sändas något data som man kan skicka ACK:et med. Man kan också vänta och sända flera ACK samtidigt.



Figur 3.3: Upprättande av kommunikation via TCP

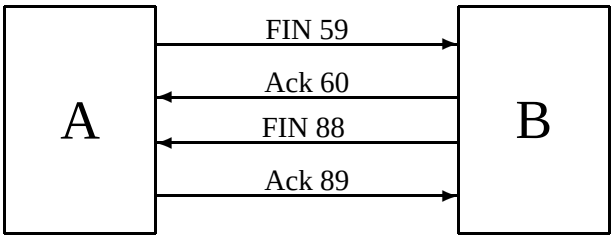
Då en kontakt ska avslutas kan det också jämföras med ett vanligt samtal. Ett par korta avskedsfraser visar då att samtalet är över. Avslutandet sker i fyra steg, som ses i figur 3.4.

1. Avslutningen initieras genom att den ena stationen, t.ex. klienten, skickar ett FIN-meddelande.

2. Servern svarar med ett ACK, (senaste mottagna sekvensnumret plus ett).

3. Servern avslutar sin kontakt genom att skicka ett FIN-meddelande.

4. Klienten svarar på FIN-meddelandet genom ett ACK och därmed är kontakten avslutad.



Figur 3.4: Avslutande av kommunikation via TCP

3.3 Glidande fönster

TCP använder sig av vad man brukar kalla glidande fönster, swnd, *(sliding window protocol)*, för att styra dataflödet. Om TCP ska sända en mängd paket kan inte alla sändas på en gång utan man måste vänta på bekräftelser på att paketen kommer fram.

Mottagarsidan meddelar till sändaren hur mycket data som kan tas emot. Det hela jämförs med fönster, och antalen bytes som mottagarsidan kan ta emot kallas erbjudet fönster, (*advertised window*). Det är det här fönstret som skickas med i TCP-huvudet och uppdateras därför hela tiden.

Det är fönstrets storlek som styr hur många paket som kan skickas, och därmed hur hög belastning vi har på länken. Då överbelastning inträffar vill vi minska fönstret. Vi inför då ett nytt fönster som vi kallar cwnd, (*congestion window*). Det är ett fönster som anpassar sig efter belastningen på nätet. Om paket försvinner eller om det blir avbrott på länken minskar cwnd, och om vi kan öka belastningen ökar cwnd.

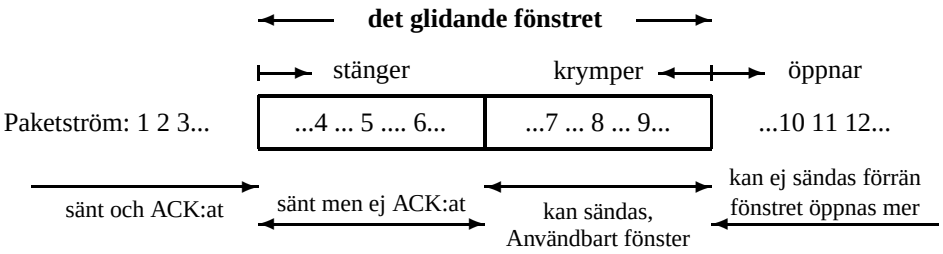
Det glidande fönstret, swnd, anpassar sig både efter vad mottagarsidan kan ta emot och om nätet är hårt belastat, se ekvation 3.1.

$$swnd = min[cwnd, advertised\ window] \tag{3.1}$$

Advertised window styrs av mottagaren och cwnd styrs av sändarsidan. Vi ska nu studera egenskaperna hos det glidande fönstret.

Sändaren beräknar utifrån det glidande fönstret hur mycket som omedelbart kan sändas, det så kallade användbara fönstret, (*usable window*), se figur 3.5. Den andra delen av fönstret innehåller paket som sänts men som inte ACK:ats.

Fönstret kan öppnas, stängas och krympa. Det stängs då data som skickats bekräftas, och innebär att den vänstra kanten flyttas åt höger. Fönstret öppnas då den högra kanten avancerar åt höger, och innebär att mer data kan skickas. Detta sker då mottagarsidan bekräftar meddelanden och därigenom ökar sin buffert för att ta emot meddelanden. Slutligen krymper fönstret när den högra kanten flyttas åt vänster. Detta behövs t.ex. då något problem uppstår och man vill minska överföringshastigheten.



Figur 3.5: Förflyttningar av det glidande fönstrets kanter

3.4 Överbelastningskontroll

TCP är inte designat för trådlös kommunikation utan är anpassat till låg bitfelssannolikhet och antar att de flesta paketförluster beror på överbelastning i nätet. TCP har därför utvecklat en mekanism som undviker överbelastningar och trafikstockningar i nätet.

När en förbindelse upprättats och dataöverföring ska inledas börjar man med att sända med låg datatakt för att sedan öka denna till lämplig nivå. Detta görs genom att reglera fönsterstorleken, cwnd.

Om ett paket tappas eller en *timeout* inträffar minskas cwnd för att sedan långsamt ökas igen. Det finns två olika sätt att öka cwnd på. De två algoritmerna som används är oberoende av varandra men används tillsammans.

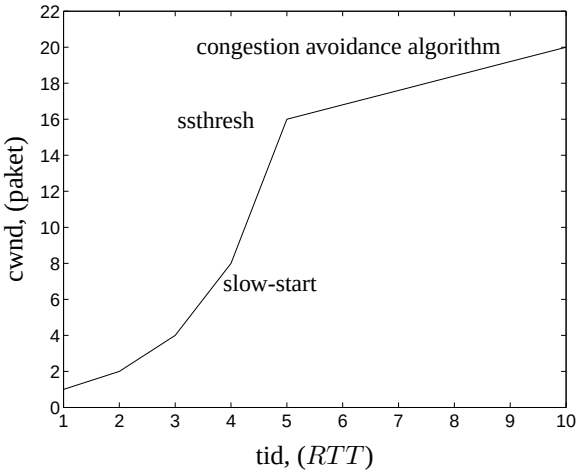
- Slow-start, långsam start.
  I algoritmen för långsam start ökar cwnd exponentiellt. Den reglerar hur ofta nya paket kan sändas beroende på hur ofta den andra sidan svarar med bekräftelser på sända paket. När en kontakt upprättas låter man cwnd vara satt till storleken som motsvarar ett paket. Därefter ökar cwnd med ett pakets storlek varje gång en bekräftelse på ett sänt paket mottages av sändaren. Om sändaren startar med att sända ett paket och tar emot ett ACK ökar cwnd från ett till två. Två paket kan därefter sändas och då dessa har bekräftats kan fyra paket skickas. Det innebär att cwnd ökar exponentiellt.

- Congestion avoidance algorithm - ”Trafikstockningsundvikande”.
  Ökningen av cwnd sker långsamt och kontrollerat enligt formeln nedan där cwnd uppdateras då ett ACK mottagits.

$$nya\ cwnd = cwnd + \frac{1}{cwnd} \tag{3.2}$$

Ökningen är maximalt ett segment per RTT.

Som nämnts ovan är ökningen av cwnd den stora skillnaden mellan de två algoritmerna. Till skillnad från vad namnet antyder är långsam start den snabba algoritmen, och används i början när man ska öka cwnd. När man sedan närmar sig maximal belastning används istället den långsammare *Congestion avoidance algorithm*. Figur 3.6 visar skillnaden mellan de två algoritmerna.



Figur 3.6: Ökning av congestion window, cwnd.

Vad är det då som avgör när man byter algoritm? Detta sker inte slumpartat, utan vid ett tröskelvärde, ssthresh, (*slow-start threshold*). Värdet på ssthresh varierar beroende på hur hög risken är för överbelastning.

Exempel: Hur mekanismen för att undvika överbelastning fungerar:

1. Kommunikation upprättas, $cwnd = 1\ paket$ och $ssthresh = 65535$ bytes t.ex. Dataöverföring startar.

2. Data bekräftas från den andra änden, och vi ökar då cwnd, men beroende på om vi tillämpar långsam start eller trafikstockningsalgoritmen ökar cwnd olika snabbt. Om cwnd är mindre än eller lika med ssthresh används långsam start, annars trafikstockningsalgoritmen.

$$\begin{aligned} cwnd &\leq ssthresh \Rightarrow \textit{Slow-start} \\ cwnd &> ssthresh \Rightarrow \textit{Congestion avoidance algorithm} \end{aligned}$$

   Det glidande fönstret som används är det minsta av cwnd och det erbjudna fönstret från mottagarsidan, se ekvation 3.1.

3. Överbelastning inträffar, indikeras av:

   a. En bekräftelse ankommer inte inom tiden RTO, och timeout inträffar. Man sätter ssthresh till hälften av de nuvarande fönstren, dock minst 2 pakets storlekar, och cwnd sätts till 1 paket för att sätta igång långsam start när paket börjar sändas igen, se ekvationerna nedan.

$$\begin{aligned} ssthresh &= max[\frac{\textit{nuvarande fönster}}{2}, 2] \\ cwnd &= 1 \quad \text{vid timeout} \end{aligned}$$

   b. Sändaren tar emot ett tredje DUPACK, en indikation på att paket tappats. Även här halveras fönstren, men cwnd sätts inte till ett paket utan bestäms av nedanstående ekvation.

$$\begin{aligned} ssthresh &= max[\frac{\textit{nuvarande fönster}}{2}, 2] \\ cwnd &= min[\textit{advertised window, cwnd}] \end{aligned}$$

4. Data bekräftas åter från den andra änden, och vi ökar då cwnd. Som förut gäller att om cwnd är mindre än eller lika med ssthresh används långsam start, annars trafikstockningsalgoritmen, se steg 2. Vi fortsätter med långsam start ända tills vi är halvvägs där vi var när överbelastningen inträffade, d.v.s. det värde vi sparade i ssthresh.

3.4.1 Snabb återsändning och snabb återhämtning

En del versioner av TCP använder sig av en funktion som kallas ”snabb återsändning - snabb återhämtning". I exemplet ovan förändras steg 3 och 4 då denna funktion används.

När paket kommer fram till mottagaren i fel ordning, t.ex. då ett paket har tappats, genereras omedelbart en bekräftelse, ett s.k. *DUPACK*. DUPACK får inte fördröjas utan sänds omedelbart. Eftersom vi inte vet om paketet tappats eller bara försenats väntar man på ytterligare några ACK för att se om det kommer fram. Man räknar med att om det bara handlar om att paketen har kommit fram i fel ordning, bör det inte skilja mer än ett eller två paket emellan dem. Om tre DUPACK tas emot antar man att paketet försvunnit, och sänder om det. Detta kallas algoritmen för snabb återsändning, (*fast retransmit algorithm*). Man väntar inte på att tiden RTO ska ta slut så att man kan sätta igång långsam start, eftersom förbindelsen fortfarande fungerar då flera DUPACK nått sändaren inom tiden RTO.

TCP startar då algoritmen för snabb återhämtning, (*fast recovery algorithm*), som beskrivs nedan.

1. När det tredje DUPACK:et når sändaren sätts ssthresh till hälften av det nuvarande fönstret och det saknade paketet sänds om. Därefter sätts cwnd till ssthresh plus tre gånger paketstorleken.

2. Varje gång ett nytt DUPACK tas emot ökas cwnd med paketets storlek och sänder ett paket om det går.

3. När nästa ACK kommer som bekräftar sänt data sätts cwnd till ssthresh och trafikstockningsalgoritmen startas. Detta ACK bör vara bekräftelsen på det först saknade paketet och bekräftar därmed alla mellanliggande paket som sänts mellan det tappade paketet och mottagningen av det tredje ACK:et.

Kapitel 4

Olika modifieringar av TCP

Trådlösa nät har problem med en ibland hög bitfelsannolikhet som påverkar prestandan negativt. Paketförluster leder till att TCP sänker takten då paketförlusterna antas bero på överbelastning. I ad hoc-nät får vi ytterligare problem enligt [8], då länkar ibland försvinner och ändras.

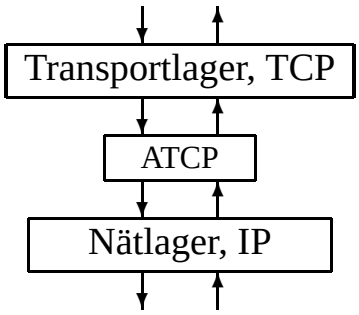
Problemen med TCP har den senaste tiden uppmärksammats mer och mer och flera förslag till modifieringar av protokollet har gjorts. Det är dock osäkert om någon förändring verkligen kommer att ske, och i så fall när och hur. Vi ska nu studera några av de förslag som har presenterats.

4.1 ATCP

ATCP [8], (ad hoc TCP) är ett förslag på en modifikation som är specifikt framtaget för ad hoc-nät. Eftersom man vill behålla kompatibiliteten med andra nät har man valt att inte förändra TCP direkt utan istället implementera ett tunt lager mellan IP och TCP, se figur 4.1.

ATCP lyssnar på nätets status och påverkar TCP genom att välja vad som ska skickas vidare från IP till TCP. Ett av de största problemen som måste övervinnas för att kunna använda ATCP är att få information om vad som sker i nätet. Detta är nödvändigt om man ska veta vad en paketförlust beror på.

ATCP använder sig av ECN-meddelanden (Explicit Congestion Notification) och ICMP-meddelanden (Internet Control Message Protocol) för att få information om nätets status. ECN meddelar om överbelastning har inträffat och



Figur 4.1: ATCP implementerat som ett protokoll mellan IP och TCP hos sändaren.

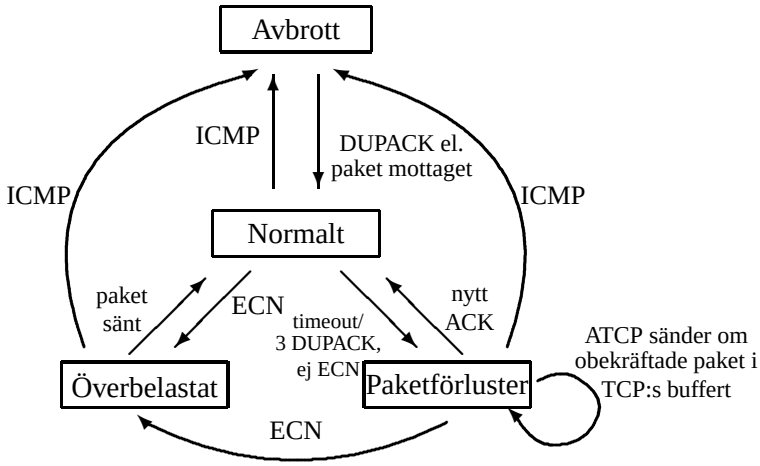
sänds med i TCP-huvudet som två nya flaggor, men befinner sig än så länge endast i försöksstadiet. En implementering av ECN i nästa version av TCP har föreslagits men det är oklart om vad som kommer att ske. I beskrivningen av ATCP [8] förutsätts ECN vara implementerad. ICMP-meddelanden finns i IP-datagramet och kan t.ex. tala om när destinationen ej kan nås, då länken är bruten.

ATCP har fyra olika tillstånd: *normal, överbelastat, paketförlust* och *avbrott*, se figur 4.2. ATCP är bara aktiv hos sändaren. Initialt befinner sig ATCP i tillståndet *normal*. I det tillståndet gör inte ATCP-lagret någonting och påverkar inte TCP.

Nätets status ger information om överbelastning har inträffat eller ej. ATCP kan därför enligt [8] välja att inte minska cwnd om paketförlusterna t.ex. beror på en brusig kanal. ATCP räknar mottagna ACK och när det märker att tre DUPACK har mottagits och cwnd ska minskas kontrollerar ATCP om trafikstockning verkligen har inträffat genom ECN-flaggan.

Om paketförlusten beror på något annat, t.ex. hög bitfelshalt eller avbrott, skickar ATCP inte vidare nästa DUPACK till TCP. ATCP sänder istället om de segment från TCP:s buffert som inte bekräftats ha mottagits. När en bekräftelse, ett ACK, till sist når ATCP skickas detta vidare till TCP som då börjar sända som vanligt igen. ATCP återvänder till normalt tillstånd.

Om nätet verkligen blir överbelastat och trafikstockning inträffar sätts ECN-



Figur 4.2: Tillståndsmodell över ATCP [8].

flaggan i ACK-meddelanden och datapaket. Om ATCP tar emot ett sådant ACK då det befinner sig i *normalt* tillstånd skickar ATCP vidare ACK:et till TCP och ATCP går in i sitt *överbelastade* tillstånd och är passiv. ATCP väntar tills TCP lyckas anpassa belastningen och sända ett nytt paket som når fram, och går då tillbaka till sitt *normala* tillstånd [8].

I ett ad hoc-nät med rörliga noder måste man ibland räkna om ändrade vägar och ibland försvinner en länk helt och hållet. När det här händer genererar nätet ett ICMP-meddelande som visar att destinationen inte går att nå. I ett sådant skede vill man absolut inte sätta igång med att sänka datatakten genom att minska cwnd utan ATCP sätter sig själv i tillståndet *avbrott* och låter TCP med jämna mellanrum sända ut ett slags testpaket för att se om kontakten återupptagits. När mottagaren till sist skickar en bekräftelse på testpaketet genom ett DUPACK går ATCP tillbaka till sitt *normaltillstånd* [8].

För att inte den nya länken ska råka använda sig av det gamla cwnd som är anpassat efter kapaciteten på den gamla länken sätts TCP:s cwnd till ett segment av ATCP då förbindelsen återupptas. Detta tvingar TCP att hitta det rätta värdet på cwnd för den nya vägen.

Om ATCP är i tillståndet *paketförlust* och tar emot ett ECN eller ICMP-

meddelande om överbelastning övergår ATCP enligt [8] till det *överbelastade* tillståndet och låter TCP fungera som vanligt vid trafikstockning.

På samma sätt betyder ett mottagande av ett ICMP-meddelande om avbrott på förbindelsen att ATCP alltid flyttas till tillståndet *avbrott* även om ATCP befinner sig i *paketförlust* eller *överbelastning*, se figur 4.2.

Notera att om vi sänder över en dålig kanal kan även ECN eller ICMP-meddelanden försvinna utan att nå sändaren, vilket kan vara ett problem. Det betyder att sändaren kan fortsätta sända en stund trots att en länk kan ha försvunnit eller trafikstockning inträffat. Eftersom ATCP endast finns hos sändaren kan det dessutom bli problem då man tar emot information från en sändare utan ATCP till ett ad hoc-nät.

4.2 TCP-F

Paketförluster i ett ad hoc-nät beror till stor del på att bitfelsannolikheten ibland är hög. Detta kan reduceras genom att öka kodningen ytterligare [4]. Det TCP-F (feedback) inriktar sig på är fel som beror på att länkar bryts. Då en länk är bruten är det onödigt att sända om paket eftersom de inte kan nå fram, utan bara kräver energi. Man vill dessutom att då kontakten återupptagits ska överföringen fortsätta med samma hastighet som innan avbrottet.

TCP-F använder sig av återkoppling mellan de mellanliggande noderna och sändaren. Antag att en sändare skickar paket mot en mottagare då kontakten bryts i någon mellanliggande nod. Så snart som nätlagret i noden upptäcker att vägen till nästa nod är bruten skickar den ett meddelande om detta, ett RFN *”route failure notification”*, till sändaren. Alla noder längs vägen tar också emot meddelandet och stänger för all passage. Om en mellanliggande nod känner till en annan väg till mottagaren kan den istället skicka paketet den vägen och strunta i att skicka vidare RFN, vilket visas i figur 4.3.

När sändaren nås av ett RFN går den över i ett *passivt tillstånd*. Den slutar sända paket, ignorerar RTO och fryser värden på cwnd och RTT. Den startar istället en ”route failure timer”. Värdet på denna beror på de underliggande routingprotokollen.

När en mellanliggande nod sedan upptäcker en ny väg för överföring skickar denna omedelbart ett meddelande till sändaren, ett RRN, *”route reestablished notification”*. Om noden tar emot fler RRN ignoreras dessa. När sändaren nås av meddelandet övergår den till sitt aktiva tillstånd och sänder sedan iväg alla

obekräftade paket i sitt fönster. Kommunikation återupptas med samma datahastighet som förut.

”Route failure timer” används i de fall som RRN tappas bort så att inte sändaren väntar för länge i onödan, utan istället skickar paket i alla fall.

Problemet med TCP-F är framförallt att införa RFN och RRN-meddelanden. Ett krav är att varje mellanliggande nod ska kunna avgöra om en länk är nere eller inte. Därefter ska RFN/RRN-medddelanden kunna sändas mellan noderna.

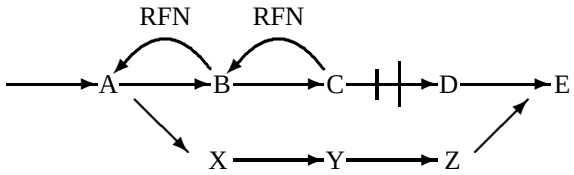
4.3 TCP-BuS

TCP-BuS (Buffering capability & Sequense information) påminner en hel del om TCP-F men har utvecklats genom att de mellanliggande noderna buffrar paket medan vägen är bruten [6].

Om en väg förloras skickas ett meddelande (ett RFN), som beskriver att länken är bruten. Sändaren slutar då sända paket. När sedan sändaren tar emot ett RRN-meddelande återupptar sändaren överföringen.

Paket buffras under tiden för avbrottet längs vägen i mellanliggande noder tills en ny väg upprättats. Noderna sänder då vidare dessa paket och sändaren informeras om vilka paket som tappats och endast dessa sänds om.

Principiellt påminner TCP-BuS mycket om TCP-F. Man koncentrerar sig på fel som beror på avbrott och antar att hög bitfelshalt i paketen kan åtgärdas med bättre kodning. Samma problem som med TCP-F kvarstår gällande överföring och implementeringen av RFN/RRN-meddelanden.



Figur 4.3: TCP-F: hur en mellanliggande nod, A, hittar en ny väg efter att mottagit ett meddelande om avbrott mellan nod C och D.

4.4 Internets standarder, RFC

Vi kan konstatera att flera förslag på modifikationer av TCP har gjorts. Frågan är när TCP tidigast kommer att modifieras för att bättre passa trådlösa nät, och i så fall hur? Vad är det som bestämmer vilka protokoll som gäller över Internet?

RFC, *Request For Comments* [1], är en publikationsserie som startades 1969, samma år som Internet. RFC kan ses som tekniska PM som är av intresse för alla de som arbetar med Internet. De flesta RFC:er är tekniska rapporter som behandlar ämnen som t.ex. protokoll, datornät, hård- och mjukvaruteknik.

Namnet ger intryck av att dokumenten inte är speciellt viktiga men sanningen är den att de mest betydelsefulla dokumenten med de facto standarder för Internets utveckling presenteras som RFC-dokument.

Här kan man hitta utförliga beskrivningar av TCP:s funktioner [2], förslag till införande av ECN-meddelanden [12] m.m. Varje ny RFC får ett nummer, och idag finns det fler än 3000.

Som namnet antyder var meningen från början att RFC skulle vara ett forum där alla som vill ska kunna skriva om vad de vill. Idén lever kvar men idag är det betydligt hårdare krav om man vill få något publicerat. När en förändring i ett protokoll skall annonseras sker detta bl.a. genom RFC.

4.5 Jämförelse

För att kunna få en översikt av vad som skiljer de olika modellerna åt har en sammanställning över skillnader och likheter mellan de olika modifieringsmodellerna av TCP gjorts, se tabell 4.1.

ATCP	TCP-F	TCP-BuS
Implementeras som ett lager mellan TCP och IP. Angriper fel som beror både på avbrott samt p.g.a. hög bitfelshalt. Har enligt tidigare arbeten [8] visat sig kunna öka kapaciteten 2-3 gånger.	Minskar problemen med fel beroende på avbrott i ad hoc-nät genom att låta de mellanliggande noderna meddela om avbrott sker. På så vis kan sändaren snabbt sluta sända paket och när förbindelsen återupptas sänds paket med samma takt som innan avbrottet. [4] visar att effekten av TCP-F ökar då avbrotten blir längre.	En vidareutveckling av TCP-F, genom att man i de mellanliggande noderna buffrar paket. När man sedan kan sända igen behöver inte alla paket sändas om utan skickas från de mellanliggande nodernas buffrar istället.

Tabell 4.1: Jämförelse mellan tre olika modifikationer av TCP.

Kapitel 5

Ad hoc-modellen

5.1 Modellen

För att kunna undersöka hur TCP fungerar i ett ad hoc-nät skapar vi en modell att använda för simuleringar. Vi är intresserade av att se hur en förbindelses kapacitet påverkas av att paket tappas och att kontakten ibland bryts. Vi väljer att studera en förbindelse istället för ett helt nät, för att få en bättre kontroll.

Modellen byggs i ett simuleringsprogram för kommunikationer och nät. TCP/IP-stacken finns implementerad, men inte ad hoc-nätet, så vi bygger en länk med ad hoc-nätets egenskaper och funktioner i modellen.

Ett mobilt ad hoc-nät [10, 5] består av rörliga noder som kommunicerar i ett trådlöst nät. Skillnaden från vanliga cellulära nät, t.ex. mobiltelefonnät, är att en central enhet skall undvikas. Man vill inte vara beroende av t.ex. basstationer. Detta beror framför allt på att det i militära sammanhang är särskilt viktigt att undvika svaga punkter som lätt kan slås ut. Mobiliteten hos nätet är en förutsättning för att ad hoc-nätet ska kunna användas på slagfältet.

För att undvika problemet med central styrning kan alla noder i nätet fungera som mellanliggande noder och skicka paketet vidare eller fungera som sändare och mottagare. Man har behov av att kunna kommunicera med enheter på andra platser och därför krävs också kompatibilitet med andra nät.

5.2 Antaganden

Några antaganden för modellen måste göras:

1. Förbindelsen.
   Eftersom ett helt nät skulle vara för komplext att studera väljer vi att enbart titta på en förbindelse. Vad sker mellan sändaren och mottagaren i ett ad hoc-nät? Vi intresserar oss framför allt för datatakten på förbindelsen under olika förutsättningar.

   En länk sammanbinder två noder i ett nät. En förbindelse i ett nät leder i verkligheten över ett antal länkar och mellanliggande noder. Antalet varierar i ett mobilt ad hoc-nät beroende på var sändaren och mottagaren befinner sig, samt vilka mellanliggande noder som finns att tillgå som kan reläa information. Vi har i vår modell endast en mellanliggande nod, men försöker applicera de egenskaper som finns i ett nät på den.

2. Kapaciteten.
   Kapaciteten på förbindelsen låter vi vara konstant eftersom det är skillnaden mellan maxvärdet av överföringshastigheten som fås på en ostörd kanal, och uppnådd kapacitet som fås då kanalen störs på något sätt som är intressant.

   Kapaciteten varierar i verkligheten eftersom en förbindelse används av olika antal användare samt p.g.a. att vägen mellan sändaren och mottagaren kan ändras då de mellanliggande noderna är mobila. Vi väljer i den här studien att inte titta på hur TCP klarar av kapacitetsvariationer. En motivering till detta är att man även i fasta nät har variationer i kapaciteten p.g.a. att antalet användare varierar och TCP inte har några problem med det. Vi intresserar oss således inte för detta och håller kapaciteten konstant för att inte försvåra tolkningarna av resultatet.

   Maximal överföring mäts då inga paket tappas och överföringen sker utan avbrott. Vi har en kapacitet på 1 MBit/s men vi når maximalt upp till ca. 0,8 MBit/s kapacitet på applikationsnivåerna. Förlusten beror på flera saker. Först och främst skickas det i varje paket en viss del data i huvudena som inte är applikationsdata. IP- och TCP-huvudet är tillsammans 40 bytes och sedan tillkommer 18 bytes på länknivån. Hela paketet kan vara på 1500 bytes så totalt sett är det ofta en liten del av kapaciteten som går åt

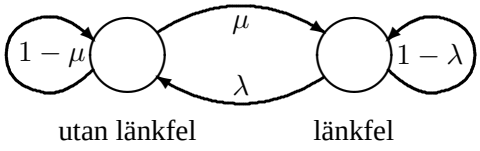
till huvuden, minst 4%. Dessutom är vår förbindelse dubbelriktad, d.v.s. vi sänder åt båda håll på den. I den ena riktningen går det mest ACK:ar och i den andra riktningen sänds data, och det är det vi studerar. Vi kommer fortsättningsvis att räkna med en maximal kapacitet på 0,8 MBit/s.

3. TCP-protokollet.
   Det finns flera olika versioner av TCP, och de vanligaste är TCP Reno, TCP Vegas och TCP Tahoe. Skillnaderna mellan de olika versionerna är små, och berör t.ex. hastigheten som cwnd kan ökas med. Vi har valt att använda TCP Reno för simuleringarna eftersom det är det vanligaste protokollet. TCP Reno finns fullständigt implementerat i simuleringsprogrammet som vi valt att använda. TCP Reno innehåller dessutom funktionen som ger snabb återsändning. En fullständig beskrivning av TCP Reno och de olika protokollen kan fås från RFC [1].

4. Överliggande och underliggande protokoll.
   Vi fördjupar oss inte i de överliggande och de underliggande protokollen. I simuleringarna använder vi applikationsprotokollet FTP, för filöverföring, och nätprotokollet IP som vi antar utför routing. Vi väljer filöverföring som tjänst eftersom vi inte vill vara beroende av fördröjningar utan bara studera hur man kan få så hög datatakt som möjligt. IP som nätprotokoll väljs p.g.a. att det är så dominerande och används över Internet.

5. Mellanliggande nod.
   Länkarna i ett ad hoc-nät har en viss felsannolikhet som varierar och gör så att en andel paket tappas p.g.a. bitfel. Förbindelsen bryts och ändras också i ett ad hoc-nät när noderna rör sig. Vi har därför implementerat en nod i nätet för att kunna stanna upp trafiken och tappa en viss andel paket. Andelen bestäms av oss och paketen tappas helt oberoende av varandra. I noden låter vi också trafiken stanna upp under en viss slumpartad tid för att simulera hur en länk bryts och en ny hittas. Det viktiga med simuleringen är att få fördröjd trafik p.g.a. länkfel och inte överbelastning, som TCP tror. Länkfelen modelleras med en markovmodell, se figur 5.1, där λ är sannolikheten att övergå från tillståndet med länkfel till det felfria tillståndet och μ betecknar sannolikheten att övergå till tillståndet med länkfel.

   Vi väljer markovmodellen eftersom länkfelen uppträder som skurfel [3].



Figur 5.1: Markovmodell länkfel

När ett länkfel inträffar söker man under en tid efter en ny väg till mottagaren tills förbindelsen återupptas igen.

6. Buffringskapacitet
   Vi förutsätter att den mellanliggande noden kan buffra paket då avbrott uppstår utan vare sig tids- eller utrymmesbegränsningar. En diskussion angående detta antagande kommer att göras senare, se kapitel 6.1.4.

7. Fördröjningar
   Vi väljer att inte studera hur fördröjningar i nätet påverkar kapaciteten. Eftersom vår väg mellan sändare och mottagare hålls konstant och vi endast har en mellannod kan vi dra slutsatsen att fördröjningen i vårt nät är liten och inte varierar så mycket. I senare arbeten skulle fördröjning kunna vara en intressant parameter att undersöka.

5.3 Simuleringsprogrammet OPNET och validering

Vi använder oss av ett simuleringsprogram som heter OPNET [9]. Hela TCP/IP-stacken finns implementerad och det underlättar naturligtvis arbetet. Ad hoc-länken implementeras på en vanlig Ethernetlänk genom en nod som placeras mitt i länken. Där tappas en andel paket och länken går upp och ner. Då det är avbrott på länken sparas paketen i mellannoden och skickas då förbindelsen fungerar igen.

Att validera OPNET och diskutera hur säkra värden som fås kan vara svårt. Man ska naturligtvis alltid vara kritisk mot resultat och kontrollera att de är rimliga. Möjligheten att kunna kontrollera hur väl OPNET motsvarar verkligheten undersöks men det är naturligtvis en hel del arbete och kostnader förenade med ett sådant projekt.

Man kan även spekulera i vad som är intressant att studera i ett ad hoc-nät. Idag är ad hoc-nät ännu under utveckling och få resultat annat än simuleringar finns att tillgå. Vilken kapacitet och prestanda som kommer att krävas vet man ännu inte. Det gör att man bör studera resultatens trender och principer snarare än exakta värden.

Kapitel 6

Simuleringar och Resultat

6.1 Simulering av TCP Reno i ett ad hoc-nät

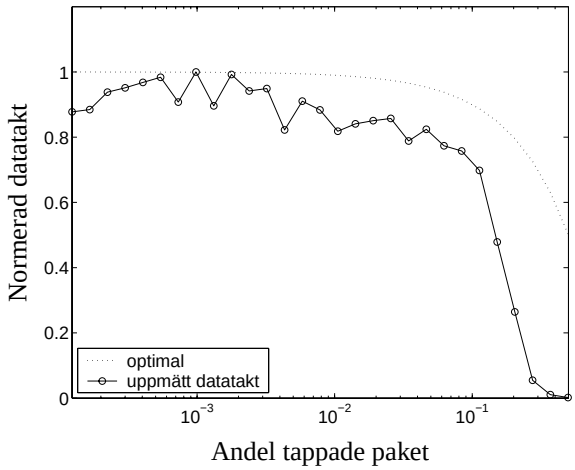
Vi startar med att simulera en filöverföring i ett ad hoc-nät mellan två noder. Länkens maximala kapacitet är $0,8 Mbit/sekund$. Tre variabler kan vi variera;

- tiden för länkbrott
- tiden mellan länkbrotten
- andelen tappade paket

6.1.1 Avbrottsfri länk, simulering 1

Vi startar med att låta länken vara avbrottsfri och studerar hur överföringen påverkas av att enbart paket tappas. Andelen tappade paket varieras mellan 0 och 0.5, se bilaga B. Paketen tappas oberoende av varandra enligt sannolikheten som ges, och vi simulerar vid 30 olika sannolikheter. Tyvärr tar simuleringarna så lång tid att simuleringarnas varianser hos de uppmätta värdena fortfarande är relativt höga. Vid längre simuleringar hade vi troligen fått jämnare kurvor med mindre varians.

I figur 6.1 ser vi resultatet av vår första simulering. Vi har mätt den genomsnittliga datatakten på filöverföringen och plottat den mot andelen tappade paket. Den prickade linjen motsvarar en optimal överföring där inga förluster utöver en tappad andel paket är medräknade. Den heldragna linjen är dragen



Figur 6.1: Kapacitet hos länken, mätt i datatakt, beroende på andel tappade paket.

mellan våra uppmätta värden. Optimal överföring, (den prickade linjen) beräknas enligt;

$$y = \text{maximalt uppnådd kapacitet } \cdot (1 - (\text{andelen tappade paket})) \qquad (6.1)$$

Vi ser att vår uppmätta kurva följer den prickade linjen väl under större delen av mätningen. Kapaciteten minskar under den vänstra delen inte mycket mer än den normalt skulle göra om det enbart berodde på tappade paket. Vi är intresserade av att se när datatakten minskar mer än vad som är motiverat p.g.a. paketförluster. Det skulle i så fall kunna bero på att TCP sänker datatakten då paket tappas, och därmed i onödan minskar vår överföringskapacitet. I figuren ser vi att kapaciteten i vår modell inte ändras förrän andelen tappade paket överskrider 10%.

Diskussion Våra resultat visar att andelen tappade paket inte påverkar kapaciteten i vår modell så mycket som vi skulle ha trott. Vid jämförelser med tidigare mätningar [7] ser man att andra modeller påverkats betydligt mer.

Att kapaciteten inte påverkas förrän andelen paket överstiger 10% kan diskuteras. Paket tappas beroende på hög bitfelshalt som uppkommer på radiokana-

len. 10% är en relativt hög andel och bör kunna undvikas genom bättre kodning av informationen på länknivå.

Vad beror det då på att vår modell inte får så stora problem? Några teorier om vad som skulle kunna vara orsaker kan presenteras:

- Paketen i vår modell tappas oberoende av varandra. Det innebär att vi sällan tappar flera paket i rad. För att cwnd hos sändaren ska minskas krävs det antingen att det inträffar en timeout eller att flera paket tappas och tre DUPACK tas emot. När ett paket tappas händer inte så mycket. Tack vare att snabb återsändning används kan hastigheten på länken snabbt ökas igen då paketet sänts om. En timeout har betydligt större effekt och minskar cwnd betydligt, se 3.4. Timeout inträffar dock inte förrän efter relativt lång tid, och skulle i det här fallet framförallt kunna bero på att flera paket i rad tappas.

- Ingen fördröjning finns i vår modell. I ett ad hoc-nät har antalet hopp som paketen måste göra mellan olika mellanliggande noder betydelse. I vår enkla modell har vi konstant väglängd över en enda mellanliggande nod. Det innebär att det i princip inte blir några fördröjningar och när paket tappas upptäcks det nästan omedelbart, och det går snabbt att sända om paketet.

6.1.2 Bitfelsfri överföring, simulering 2

Nästa steg i studien av TCP i ad hoc-nät är att studera avbrott på förbindelsen. Avbrott beror på att de rörliga noderna i ad hoc-nätet kommit för långt ifrån varandra och kontakten bryts helt. Det kan också vara en nod som försvunnit och en ny väg som måste hittas, vilket leder till ett kort avbrott. Eftersom vi i tidigare kapitel 6.1.1 har kunnat konstatera att felen beroende på paket som tappas inte påverkar kapaciteten i vår modell nämnvärt, väljer vi då vi studerar avbrottsfel att sätta andelen tappade paket till noll.

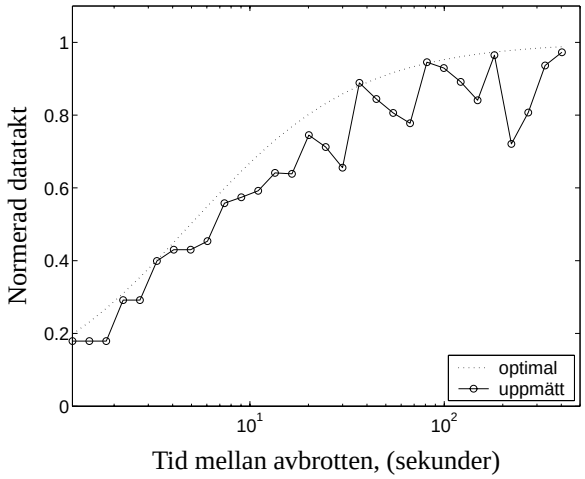
Vi kan då studera vilken inverkan endast avbrott kan ha på kapaciteten. Vi undersöker en simulering där tiden för avbrott samt tiden mellan avbrott är representerade som exponentialfördelningar av 30 olika värden. På så sätt fås en simulering av 900 punkter. I simuleringarna exponentialfördelar vi tiderna runt värden som har valts i intervallet 1 sekund till 400 sekunder, se bilaga B.

I vår figur, se 6.2, har vi ritat in en prickade kurva som motsvarar den maximala kapacitet som skulle kunna fås om det bara var under tiderna för avbrott som kapaciteten minskar. Den prickade funktionen motsvarar således

$$y = \text{maxkapacitet} \cdot \frac{\text{tid mellan avbrott}}{\text{tid mellan avbrott} + \text{avbrottets tid}} \tag{6.2}$$

Vi antar i det optimala fallet att man sänder på maximal hastighet omedelbart när förbindelsen tas upp igen. Det är naturligtvis inte möjligt men det är en intressant jämförelse.

En kurva har valts att visas bland de 60 kurvor som vi har studerat med datatakten plottad mot avbrottstid respektive tid mellan avbrott. Vi har valt att visa hur kapaciteten förändras då avbrottets tid är exponentialfördelad kring 5 sekunder, och tiden mellan avbrotten varieras mellan 1 och 400 sekunder, se figur 6.2. Valet av just denna kurva motiveras med att den tydligt visar tendensen hos alla de figurer som studerats. Vi ser här att våra uppritade resultat följer den prickade kurvan relativt väl.



Figur 6.2: Kapacitet hos länken mätt i datatakt vid avbrott på ca 5 sekunder, med olika tid mellan avbrotten längs x-axeln.

Diskussion Tidigare undersökningar och studier har visat att avbrott på länken ger ytterligare sämre dataöverföring än normalt, beroende på att TCP startar

överföringen efter ett avbrott långsamt och ibland inte direkt när förbindelsen är klar. I vår studie visar resultaten istället att problemet inte är så stort och att det i vår modell inte behöver åtgärdas.

Förklaringen till resultaten kan man spekulera kring. Några teorier som vore intressanta att undersöka ytterligare är följande:

- Hur påverkar buffrandet av paket situationen? Då avbrott inträffar i vår modell buffras alla paket och då avbrottet är slut skickas de igen. Detta förutsätter dessutom att samma väg upprättats eller att alla buffrade paket i alla fall kan användas. Det innebär att små avbrott inte har så stor påverkan i vår modell, utan snarare innebär en fördröjning på förbindelsen. Vi ska senare studera hur buffrandet av paketen i mellannoden kan påverka överföringen, se stycke 6.1.4.

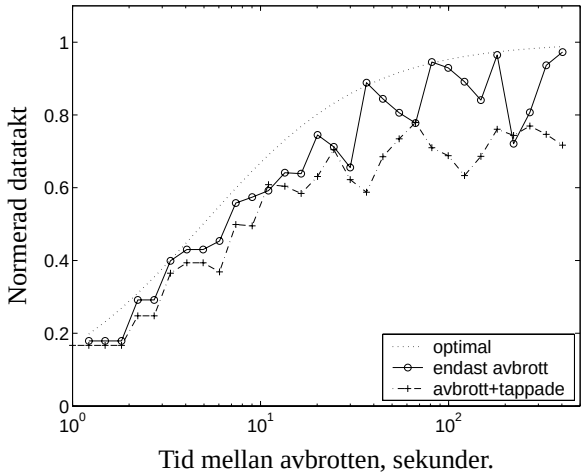
- Då ett avbrott inträffar i ett ad hoc-nät kommer sändaren inte att meddelas omedelbart, eftersom det är en viss fördröjning i nätet. Hur lång tid det kan ta beror på var i nätet avbrottet sker samt kapaciteten på länken vid detta tillfälle.

  I vårt nät har vi en mycket kort förbindelse och sändaren meddelas nästan omedelbart, och sänder inte ut så många paket i onödan. På samma sätt meddelas vi snabbt om kontakt återupptas eftersom vår mellannod genast sänder ut paket från bufferten. I ett ad hoc-nät utan buffring kan det ta upp till 2 minuter innan man upptäcker att en förbindelse är återupptagen.

6.1.3 Avbrott samt tappade paket, simulering 3

För att ta reda på hur vår modell reagerar då vi har fel beroende på både tappade paket och avbrott lät vi simulera detta. Det är svårt att dra en slutsats kring hur felen egentligen påverkar varandra men vi kan klarlägga att kapaciteten minskar ytterligare utöver vad som normalt borde ha skett, se figur 6.3.

Den prickade linjen motsvarar vad vi optimalt skulle kunna uppnå, den heldragna är kapaciteten då vi endast har avbrott, (se figur 6.2) och slutligen är den streckade kurvan med kors i mätpunkterna kurvan som fås då vi har både fel p.g.a. tappade paket och avbrott. Vi ska senare diskutera resultatet i angränsning till modifieringar av TCP.



Figur 6.3: Kapacitet hos länken då ca 9% av paketen tappas och avbrott på ca 5 sekunder uppträder med varierande intervall, plottad längs x-axeln.

6.1.4 Buffrandet av paket, simulering 4

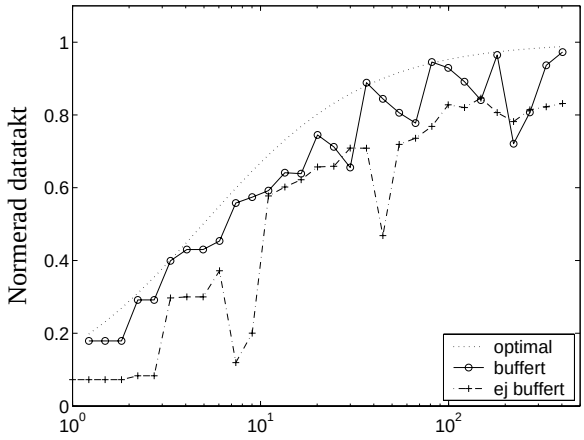
Vi har tidigare diskuterat vårt val av en buffert i den mellanliggande noden. Vi ska nu studera vilken inverkan en buffert kan ha på kapaciteten i vår modell.

Vi jämför vår kurva från 6.2, inga tappade paket men avbrott på ca 5 sekunder med varierande mellanrum, med motsvarande modell utan buffert vid avbrott. Paketen slängs istället av den mellanliggande noden vid avbrott.

I vår figur 6.4 kan vi studera resultatet. Den prickade kurvan är den optimala datatakten som vi vill uppnå. Den heldragna linjen är vår modell med buffring och den streckade kurvan är den utan buffring. Vi ser tydligt att buffring påverkar datatakten positivt, den heldragna kurvan ligger över den streckade.

Diskussion Vad påverkar egentligen vårt antagande om huruvida buffring bör förekomma eller ej? Vi utreder här ett antal faktorer som man bör ta i hänsyn till vid ett sådant beslut.

- Applikation
  Beroende på vilken tjänst som nätet ska leverera har vi olika krav på fördröjningskänslighet. Vid en filöverföring kan vi buffra paket, eftersom det



Figur 6.4: Kapacitet hos länken då ca 9% av paketen tappas och avbrott på ca 5 sekunder uppträder med varierande intervall, plottad längs x-axeln.

inte gör något om en viss fördröjning sker. Om det däremot är ett mobiltelefonnät kan vi naturligtvis inte buffra paketen särskilt länge. Undersökningar har visat att fördröjningar på ca 0,25 sekunder uppfattas av oss och då fördröjningar på 1 sekunder uppträder är telefonsamtal i princip omöjliggjorda.

- Routingprotokoll
  Buffringen av paket kan inte ske i transportlagret och knappast heller i nätverkslagret om vi inte är beredd att göra några större förändringar där. Det naturligaste borde vara att implementera buffring i routingprotokollet där problemen med att hitta nya vägar uppstår. Någon slags buffring är på den här nivån nödvändig.

- Minnesutrymme
  Det är orimligt att man ska kunna klara av att buffra oändligt mycket paket. En gräns för hur många paket som ska kunna buffras måste väljas.

- Fördröjning
  Buffring av paket kan innebära olika slag av fördröjningar. När en förbin-

delse återupptas måste bufferten tömmas först innan nya paket kan komma igenom. Det kan i sin tur innebära att viktig information som snabbt måste fram fördröjs. Ett problem inträffar då vi blandar applikationerna i ett nät, där vissa är mer känsliga för fördröjning.

- Nätets topologi
  Beroende på mobiliteten i nätet kommer buffringen ge olika god effekt. För att buffringen skall kunna vara effektiv krävs att samma noder används då förbindelsen återupptas.

6.2 Modifierat TCP - Simulering

I det tidigare avsnittet beskrevs en simulering av TCP Reno i ett ad hoc-nät. Resultatet kunde tolkas som att bitfelshalten och fel p.g.a. avbrott har mindre påverkan på kapaciteten i vår modell än vad vi trott. Vi ska nu se om en modifikation av TCP ändå kan ge en ökad kapacitet hos förbindelsen. Vi gör inte någon fullständig implementering av en modifierad modell av TCP utan det är principen som ska studeras. Kan en modifiering av TCP påverka förbindelsens kapacitet, och i så fall hur?

6.2.1 Modifiering: snabbare ökning av cwnd efter avbrott

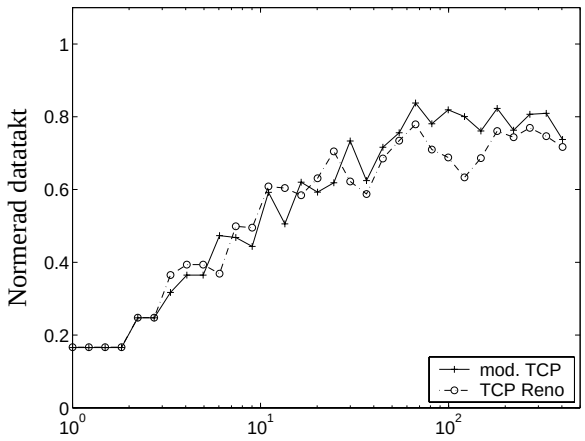
Vi inriktar oss på att studera hur kapaciteten förändras då vi ökar överföringshastigheten, cwnd, kraftigare efter avbrott. Paket som tappas p.g.a. hög bitfelshalt kan lösas med hjälp av ökad kodning, dock på bekostnad av bandbredd. I tidigare studier [4, 6] har man också konstaterat att det är avbrottsfelen som är de viktigaste.

Problemet med avbrott är att då det inträffar tolkas det som en överbelastning och cwnd krymper. Om inte förbindelsen återupptagits inom tiden RTO sätts cwnd till ett pakets storlek och ssthresh halveras. Ökningen av cwnd sker därefter långsamt. En förbättring av modellen skulle kunna vara att då ett avbrott inträffar minskas varken ssthresh eller cwnd.

Detta kräver naturligtvis att man ska kunna skilja mellan avbrott, tappade paket och trafikstockning. Vid trafikstockning ska de vanliga åtgärderna vidtas för att minska sannolikheten för överbelastning igen.

Efter långa avbrott väljer vi i vår modifiering att sätta cwnd och ssthresh till de värden de hade före avbrottet, till skillnad från TCP Reno som vid långa avbrott får timeout och sätter cwnd till ett segment och minskar ssthresh till hälften. Vi använder samma simuleringsmodell som vid studien av både bitfel och avbrott.

En jämförelse görs med en motsvarande simulering med TCP Reno, se figur 6.3. Kurvorna är simulerade med både bitfel och avbrott, men kurvan med ringar i mätpunkterna använder TCP Reno medan den andra med kors i mätpunkterna har modifierat TCP. Andelen tappade paket är 9% och avbrotten är exponentialfördelade runt 5 sekunder. På x-axeln varieras tiden mellan avbrotten från 1 sekund till 400 sekunder.



Figur 6.5: Kapacitet hos länken med TCP Reno jämfört med en modifierad modell av TCP. 9% av paketen tappas och avbrott på ca 5 sekunder uppträder med varierande intervall, plottad längs x-axeln.

Diskussion Vi ser att även en relativt enkel modifiering av TCP kan påverka kapaciteten. Vår modifiering har lett till en något högre överföringstakt i vår modell i den högra delen av vår figur. Det intressanta och som man bör fokusera på är att se hur mycket TCP faktiskt kan påverka kapaciteten på en förbindelse.

Man kan därmed konstatera att en modifiering av TCP skulle kunna ha en

betydelse för kapaciteten hos ett ad hoc-nät, och därför bör vara viktig att ha i åtanke då ad hoc-nät ska modelleras. Ytterligare forskning och effektivisering av modifieringarna bör kunna leda till ytterligare förbättringar av kapaciteten, vilket också tidigare arbeten [8, 4, 6] visar.

6.3 Antalet avbrott relativt andelen tid i avbrott

En intressant del att undersöka är att studera om datatakten varierar mycket beroende på längden hos avbrotten. Ett sätt att studera detta är att låta proportionen mellan tiden för avbrotten och tiden mellan avbrotten vara konstant, men variera längden på avbrotten, se ekvation 6.3. Vi studerar hur kapaciteten kan förändras då vi 50% av tiden råkar ut för avbrott, och hoppas på att kunna dra någon slutsats kring vilka avbrott som ger störst problem i vår modell. Om överföringen då var optimal skulle vi kunna sända 50% av vad vi brukar. Vi vet dock att eftersom det tar ett tag att komma upp i maximal datatakt så kommer vi inte att lyckas med detta.

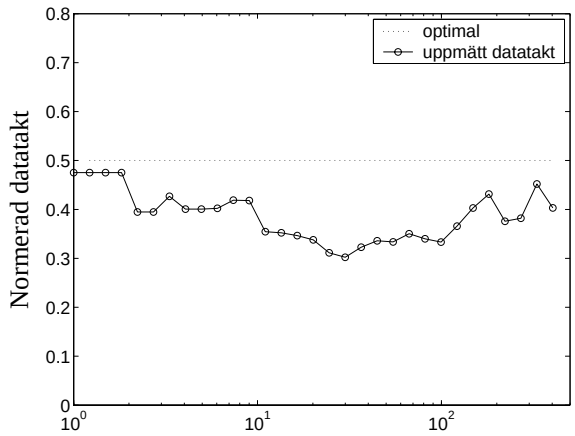
$$\delta = \frac{\text{tid mellan avbrotten}}{\text{avbrottets tid}} \tag{6.3}$$

En teori är att om vi låter avbrotten vara relativt korta men frekventa hinner inte TCP öka cwnd tillräckligt mycket mellan varje avbrott och resultatet blir då sämre datatakt. Man kan också tänka sig att om avbrottstiderna är mycket små hinner inte någon timeout göras och eftersom vi buffrar paket innebär det endast att paketen fördröjs något. Det skulle kunna innebära att små avbrott inte minskar kapaciteten så mycket.

Vid mycket långa avbrott som dock inte inträffar så ofta kommer timeout garanterat att inträffa vilket innebär en mycket långsam start efter avbrottet. Detta kommer troligen att vägas upp av att avbrotten sker så sällan att man lyckas överföra mycket information under tiden mellan avbrotten.

Vi lät proportionen mellan avbrott och förbindelse vara ett, d.v.s. de är lika stora. Kapaciteten studerades och vi varierade avbrottstiden och därmed också tiden mellan avbrotten mellan 1 och 400 sekunder, se figur 6.6

Diskussion I figuren 6.6 ovan ser vi en något hackig kurva, men vi kan ändå urskilja en viss trend. En hypotes skulle kunna vara att vi vid mycket korta



Figur 6.6: Kapacitet mätt i datatakt vid olika avbrottstider men med konstant proportion mellan avbrottsfri tid och tid i avbrott. I detta fall är tiden i avbrott lika lång som den avbrottsfria tiden och plottad längs x-axeln.

avbrott tack vare vår buffert enbart får en viss fördröjning i systemet och därmed en medioker datatakt.

Då avbrottstiden överskrider 2 sekunder minskar överföringen en del och det skulle kunna bero på att det krävs ett avbrott på ca 2 sekunder för att timeout ska inträffa. När sedan avbrottstiden ökar ytterligare ökar också tiden mellan avbrotten och vi lyckas öka fönsterstorleken tillräckligt under den här tiden för att överföra relativt mycket paket.

Kapitel 7

Slutsatser och fortsatt arbete

7.1 Slutsatser

Vi har i den här rapporten studerat problemen med TCP (Transmission Control Protocol) i ad hoc-nät. Som bakgrund har en sammanställning av protokollstacken och en fördjupning av TCP:s egenskaper och funktioner gjorts samt en litteraturstudie över tidigare arbeten för att se vilka problem med TCP i ad hoc-nät som framkommit samt vilka åtgärder som föreslagits.

I trådlösa nät fås en hög bitfelshalt vilket innebär att TCP sänker takten om fler paket tappas. I ad hoc-nät förvärras situationen ytterligare eftersom information skickas mellan de rörliga noderna, och det händer att avbrott inträffar då enheterna splittras eller då vägar i kommunikationen ändras. Tidigare arbeten visar att en modifiering av TCP skulle kunna vara effektiv [4, 6, 8, 11] och öka kapaciteten.

Våra resultat har dock visat att problemen i vår modell, byggd i simuleringsprogrammet OPNET, inte är så stora som man kunnat förvänta sig. I vår modell kan paket tappas, avbrott uppstå och då detta sker buffras paket i en mellanliggande nod. Sträckan mellan sändaren och mottagaren hölls konstant och fördröjningen studerades ej.

Då paket tappades krävdes det att vi förlorade över 10% av paketen för att allvarligt påverka kapaciteten, en siffra som man med felrättande kodning i länklagret lätt kan klara. Avbrott visade sig inte heller ge alltför stora problem i vår modell.

Vissa förklaringar till detta har getts. Bland annat kan vi konstatera att buff-

randet i ad hoc-nätet har betydelse. Kapaciteten ökade då paketen buffrades i vår modell. Med buffring får t.ex. korta avbrott endast en fördröjande effekt. Detta berodde dock på att alla buffrade paket kunde användas, d.v.s. vi använde samma noder igen efter avbrottet.

En enkel modifiering av TCP gjordes också genom att låta datatakten öka hastigt efter avbrott och visade att en ökning av kapaciteten var möjlig. Vi kan slutligen konstatera att en modifiering av TCP är effektiv men om vi vill uppnå full effekt krävs det en förändring av TCP-huvudet för att kunna sända med information om överbelastning inträffar eller ej.

7.2 Utvidgningar

I den här rapporten har TCP studerats och en del resultat och slutsatser har kunnat dras. Det skulle dock vara intressant att ytterligare undersöka TCP och göra en fullständig modell av en modifierad variant. Förhoppningsvis kan ytterligare kapacitetsvinst göras.

Ett annat område som vi inte har behandlat överhuvudtaget är routing- och accessprotokollen. Dessa påverkar kapaciteten på så sätt att det kan ta olika lång tid att hitta nya vägar och effektiva vägar. Säkerheten bör också diskuteras. De mellanliggande noderna som reläar paketen ska inte kunna komma åt informationen, och det är en mycket viktig fråga då det gäller ett militärt radionät.

Buffring av paket vid avbrott i ad hoc-nät bör studeras. I vår modell hade buffringen en positiv effekt. En intressant fråga är under vilka förutsättningar detta är fallet.

Vår modell bör utvecklas genom att noggrant definiera hur buffring bör ske, och implementera detta. En fördröjning bör dessutom läggas till. Den bör dessutom kunna varieras, för att simulera hur förbindelserna varieras.

Litteraturförteckning

[1] RFC. http://www.ietf.org, 2001-01-22. Tekniska PM för internetutvecklare.

[2] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. http://www.ietf.org/rfc/rfc2581.txt, April 1999. RFC 2581.

[3] D. Bertsekas and R. Gallager. *Data networks*. Prentice-Hall, 2 edition, 1992.

[4] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. A Feedback based Scheme for Improving TCP Performance in Ad Hoc Wireless Networks. *IEEE Personal Communications*, pages 34–39, February 2001.

[5] A. Hansson. Ad Hoc presentation. augusti 2001.

[6] D. Kim, C-K. Toh, and Y. Choi. TCP-BuS: Improving TCP Performance in Wireless Ad Hoc Networks. *IEEE International Conference on Communications*, pages 1707–1713, June 2000.

[7] F. Lefevre and G. Vivier. Understanding TCP:s behaviour over wireless links. pages 123–130.

[8] J. Liu and S. Singh. ATCP: TCP for Mobile Ad Hoc Networks. *IEEE journal on selected areas in communications*, 19(7):1300–1315, July 2001.

[9] Opnet. www.opnet.com, 2002-01-28.

[10] C. Perkins, E. Belding-Royer, and S. Das. IP Flooding in Ad Hoc Mobile Networks. *Mobile Ad Hoc Networking Working Group Memo*, November 2001.

[11] R. Prakash and M. Sahasrabudhe. Modifications to TCP for Improved Performance and Reliable end-to-end Communication in Wireless Networks. *IEEE*, pages 938–942, 1999.

[12] K. Ramakrischnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. http://www.ietf.org/rfc/rfc2481.txt, January 1999. RFC 2481.

[13] W. R. Stevens. *TCP/IP Illustrated, Volume 1*, volume 1. Addison-Wesley, 1989.

Appendix A

OPNET

I OPNET (*Optimum Network Performance*) kan modeller byggas samman av olika redan implementerade byggstenar, moduler. Egna moduler kan också konstrueras, och programmeringsspråket som används är C, [9].

För att kunna variera olika inparametrar under en simulering skrivs lämpligen ett perlscript som sköter detta. En tabell över de olika inparametrar och värden hittas i bilaga B.

OPNET samlar in statistik, men det kan vara svårt att bearbeta data i OPNET. Därför väljer vi att exportera resultatfilerna till MATLAB där man utan problem kan visualisera de resultat som är intressanta.

Det grafiska gränssnittet i OPNET gör modellerna lätta att visualisera och förstå. Tyvärr är det inte helt enkelt att arbeta i OPNET och det krävs en relativt lång inläringstid. Detta beror bl.a. på att man i OPNET använder många egna kommandon och att all kod inte finns tillgänglig för kunden.

Det finns stora valmöjligheter vid simuleringar i de olika modellerna i OPNET, när det gäller att sätta parametrar. I tabell A.1 följer en beskrivning av de variabler i OPNET som har satts och till vilka värden. En viss förklaring ges också. Detta är framförallt intressant för andra som arbetar i OPNET eller liknande program.

Projekt som bearbetats:	LAN i OPNET
Filstorlekar som överförs:	5.000.000 bytes. (Detta för att maximera användningen av kanalen)
Nodmodell för TCP/IP-stacken:	ethernet_wkstn_adv samt ethernet_server_adv
Processmodell för TCP:	tcp_manager_v3, tcp_conn_v3
Mottagarens buffert:	8760 bytes.
Maximal ACK-fördröjning:	0.2 sekunder
Initialt RTO:	1 sekund
Minsta RTO:	0.5 sekunder
Maximalt RTO:	64 sekunder

Tabell A.1: Tabell över olika inställningar av variabla parametrar i OPNET.

Appendix B

Mätpunkter

Vid simuleringarna har vi mätt kapacieten i nätet under olika förutsättningar. Här ges de exakta värden som vi använt oss av, och de kan också jämföras med figurerna där mätvärdena plottats mot datatakter. Tiderna anges i sekunder och vi har använt exponentialfördelning runt dessa. Andel tappade paket är likformigt fördelade.

Nr	Andel tappade paket	Länkbrottstid/Tid mellan avbrott (sek)
1	0,49	1,22
2	0,37	1,49
3	0,28	1,82
4	0,21	2,22
5	0,16	2,72
6	0,12	3,32
7	$8,9 \cdot 10^{-}2$	4,06
8	$6,7 \cdot 10^{-}2$	4,95
9	$5,0 \cdot 10^{-}2$	6,04
10	$3,8 \cdot 10^{-}2$	7,38
11	$2,8 \cdot 10^{-}2$	9,02
12	$2,1 \cdot 10^{-}2$	11,02
13	$1,6 \cdot 10^{-}2$	13,46
14	$1,2 \cdot 10^{-}2$	16,44
15	$9,0 \cdot 10^{-}3$	20,08
16	$6,8 \cdot 10^{-}3$	24,53
17	$5,1 \cdot 10^{-}3$	29,96
18	$3,8 \cdot 10^{-}3$	36,60
19	$2,9 \cdot 10^{-}3$	44,70
20	$2,1 \cdot 10^{-}3$	54,60
21	$1,6 \cdot 10^{-}3$	66,69
22	$1,2 \cdot 10^{-}3$	81,45
23	$9,1 \cdot 10^{-}4$	99,48
24	$6,9 \cdot 10^{-}4$	121,51
25	$5,2 \cdot 10^{-}4$	148,41
26	$3,9 \cdot 10^{-}4$	181,27
27	$2,9 \cdot 10^{-}4$	221,40
28	$2,1 \cdot 10^{-}4$	270,43
29	$1,6 \cdot 10^{-}4$	330,30
30	$0,0$	403,43

Tabell B.1: Tabell över simuleringarnas mätpunkter