En studie av metoder för att undvika trafikstockning i datanätverk baserade på TCP

John Tran
Åbo Akademi
Kandidatuppsats i datavetenskap 2011

Abstrakt

Internet idag har blivit allt populärare som medium för kommunikation, handel och nöje. Det möjliggör åtkomst av audio- och videomaterial via nätbrowser. De senaste åren har intresse för nätsändning blivit allt större och många radio- och TV-stationer försöker etablera sig på Internet. Internet är ett mycket bekvämt medel för att sända e-mail, läsa e-tidningar och att utföra andra former av elektronisk kommunikation. Men Internet är inte utan problem, den är ursprungligen inte designat för att fungera som ett medium för att transportera stora mängder av data. I takt med att bandbredd har ökat världen över har också kongestion problem blivit mera synligt Jag försöker i detta arbete beskriva hur TCP congestion kontroll är uppbyggd. Men eftersom TCP congestion kontroll mekanism är byggd runt nätverkslagret och transportlagret behövs också en översikt av hur dessa två IP lager fungerar.

1. Inledning

Vad är kongestion?

Internet erbjuder oss med en service som kallas för best effort, vilket betyder att nätverket gör sitt bästa att leverera data från en punkt till en annan så effektivt som möjligt. Det finns ingen garanti för datapaket som sänds över Internet att det kommer fram till sin destination snabbt eller att de överhuvudtaget når fram till sin destination. Att ladda ner en fil kan ena dagen ta dubbelt så länge som dagen före. De flesta av oss är vana vid att internet kan bete sig på detta sätt, men vad beror detta på?

Det finns ett par orsaker till detta till exempel: Internet länkar blir otillgängliga, omberäkning av rutt och congestion. Kongestion är då länk eller nod belastas med för mycket paket att dess genomströmning degraderas som resultat av kö i routers eller förlust av paket.

Låt oss betrakta ett exempel där en internetlevererantör erbjuder sina kunder en maximal bandbredd på 1 Mbps, totala antalet kunder är 1000. Fastän maximal bandbredd är 1 Mbps per kund, använder kunden igenom snitt endast 40% av bandbredden, d.v.s. 400 kbps. En 1 Gbps länk till deras Internet gateway borde vara att naturligt val för internet levererantören, men låt oss anta att de istället väljer att ha endast en 800 Mbps länk till gateway av ekonomiska skäl.

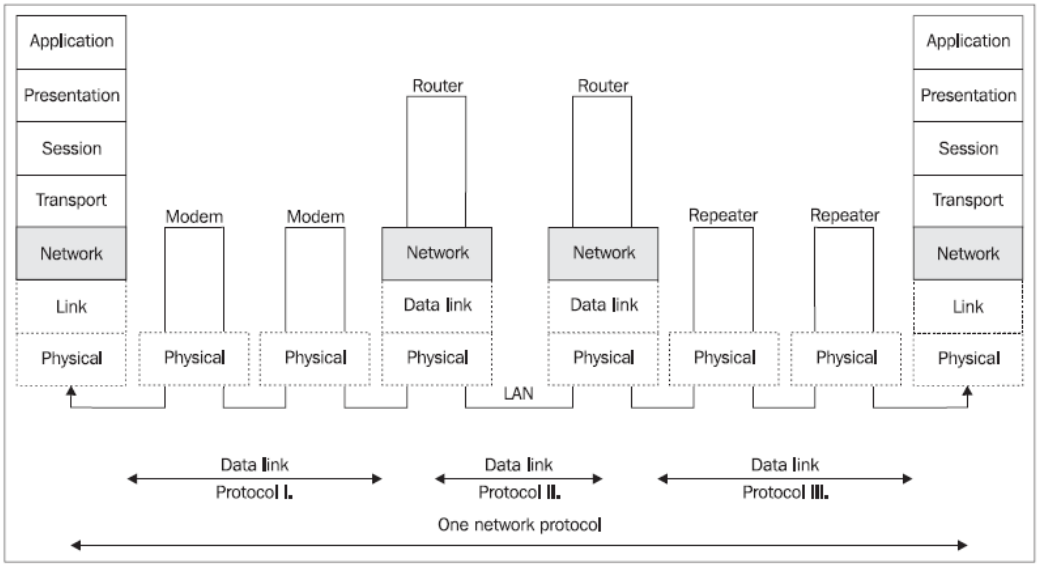
Trafik från gateway lider ibland av enstaka toppar då länkens maximala kapacitet överskrids då ett antal kunder använder sin maximal bandbredd samtidigt. Detta leder till att inte alla paket kan transporteras från gateway. I sådant fall har gateway endast två alternativ: att buffra eller förkasta de överflödiga paketen. Eftersom trafiktoppar händer ganska sällan är det vanligt att internet router har en buffer för paket som överskrider bandbredden. Buffern har oftast schemaläggning av typen FIFO (First In, First Out) och förkastar överflödiga paket endast då buffern är full. Orsak till denna design är att det antas att trafiken så småningom kommer att minska och buffern till slut kommer att tömmas, så buffern är designad för att endast hantera korta toppar i trafiken.

Det kan verka naturligt att ha en så stor buffer som möjligt för att hantera långa köer i trafiken men detta är inte en bra design eftersom paket som sparas i en bufferkö ökar på fördröjning av levereransen. När kön växer sägs det att nätverket av lider stockning och detta är kongestion. Detta kan ha som effekt att levereranstiden blir stor och i värsta fall att paketen förkastas ur kön.

2. Definitioner

För att förstå sig på hur kongestionskontroll fungerar måste vi ha en snabb översikt av nätverkprotokoll och transportprotokoll eftersom congestion oftast uppstå i dessa två nivåer. Många congestion kontroll principer är baserad huvudsakligen på hur dessa två protokoll fungerar.

2. 1 Nätverkslager



Figur1: nätverkslager i OSI:s sjulagersystem. [7]

Nätverkslager är det tredje lagret i OSI:s sjulagersystem. Lagret ansvarar för ruttning av paket som sker i alla mellanliggande router mellan två värdar. Lagrets roll är med andra ord att transportera paket från en värd till en annan. För att nätverkslagret skall kunna utföra detta behövs ett par viktiga funktioner:

Värdadress (IP)

För att kunna leverera paket från en värd till en annan måste alla värdar ha en unik address för att bestämma var värden är belägen. Adressen för värd kallas för IP-adress. Adresseringen är baserad på ett 32-bit nummer, vilket betyder att det maximalt kan finnas 2^{32} adresser.

Bestämning av rutt

Nätverkslagret måste bestämma via vilken rutt transport av paketer skall ske från en värd till en annan. Algoritm för beräkninger av dessa rutter kallas oftast för ruttningsalgoritmer.

Vidarebefordring av meddelande

Eftersom många nätverk är uppdelat i mindre subnätverk måste de vara sammanbundna med gateway eller router. Gateway och router har som uppgift att vidarebefordra paket som sänds mellan olika nätverk [1].

2.1.1 Nätverksservicemodell

Nätverksservicemodell definieras av end-to-end transport av data mellan två värdar. Den viktigaste abstraktionen som nätverkslagret erbjuder till transportlagret är om den använder connection-oriented service (virtual circuits), eller en connectionless service (datagram) modell. Med Virtual circuit lager måste det finnas en service som hanterar setup (handskakning), dataöverföring, teardown och signalering. Datagram fungerar på så sätt att varje gång en värd vill sända ett datapaket, så måste paketet innehålla address för källa och destination [1, 3].

2.1.2 Nätverksmodell

Ruttning av datapaket fungerar på samma sätt som vanlig brevpost, ett paket transporteras till sin destination på basen av dess addressering, mellan liggande switchar använder sig av en ruttningstabell och vidarebefordrar paket enligt denna. Olika paket till samma destination kan färdas via olika rutter. Den nuvarande internetarkitekturen erbjuder för tillfället endast en servicemodell, datagram servicemodell också kallad för “best-effort service”. Det här betyder att det finns ingen garanti för att datapaket levereras eller att paket från en värd har högre prioritet än en annan. Servicen gör sitt bästa för att leverera paket till sin destination[1, 3].

2.1.3 Ruttning

För att transportera paket från en värd till destinationsvärden måste nätverkslagret utföra två funktioner. Den första funktionen är ruttning, med andra ord bestämma den rutt som skall användas för transport av paket. Ruttning använder sig av följande algoritmer (RIP, OSPF, BGP). Den andra funktionen är vidarebefordring [1, 3].

2.2 TCP Transportlager

Transportlagret är beroende av de tjänster som nätverkslagret erbjuder för kommunikation mellan olika värdar. Nätverkslagret förflyttar transportslagrets data segmenter från en värd till en annan. Datasegment som skickas från en värd skickas från transportlagret ner till nätverkslagret. Det har som uppgift att se till att datasegmentet kommer till destinationsvärden och sedan skicka segmentet upp i protokollstacken tillbaka till transportlagret.

Transportlagerprotokoll har som uppgift att förse den logiska kommunikation mellan olika applikationprocesser körda i det olika värdar. Logisk kommunikation innebär inte att applikationsprocesser är fysiskt kopplade till varandra utan de kan vara placerade på olika kontinenter, men om de ändå är sammankopplade via routrar och länkar kan det ses som att de är fysiskt sammankopplade. Applikationsprocesser använder sig av den logiska kommunikationen från transportlagret för att kommunicera med varandra utan att behöva oroa sig om den fysiska infrastruktur som används för att förmedla dessa meddelanden.

Transportlagrets protokoll implementeras endast av värdarna men inte av nätverksroutrar. Nätverksroutrar har endast hand om de tre första lagren i OSI-modellen. När en applikationsprocess i en värd skall skicka data till en annan värd, konverterar transportlagret data från applikationen. Det här processen går till på så sätt att data bryts ner till mindre bitar och till varje bit sätts en transportlagerheader för att bilda ett segment. Transportlagersegmentet skickas sedan ner till nätverkslagret där vaje segment inkapslas med en header från nätverketslagret för att bilda nätverkslagerpaket.

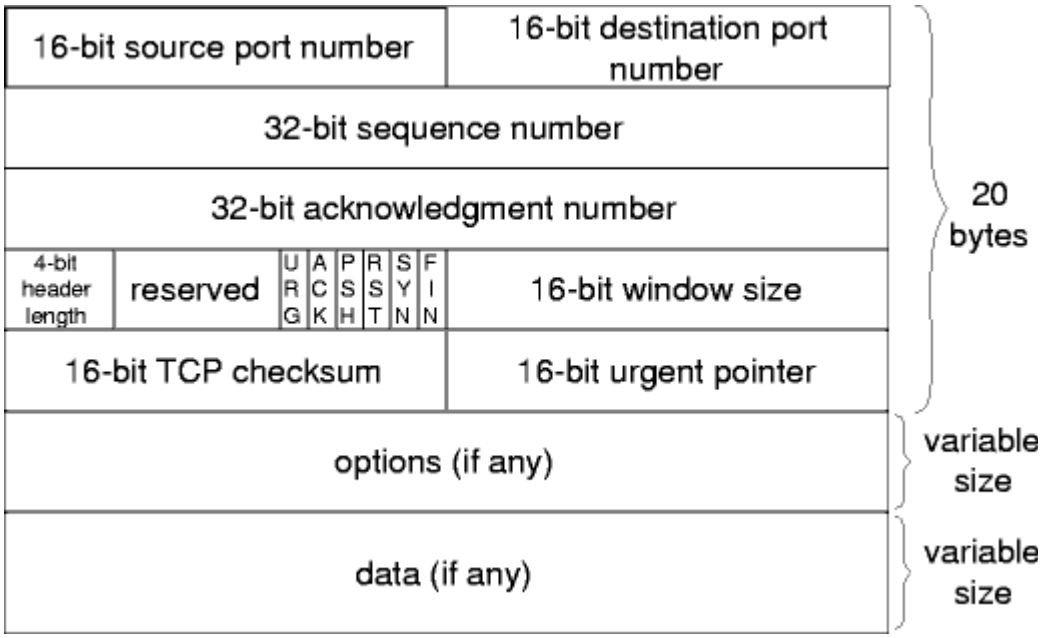
Då nätverkslagerpaket tas emot vid destinationsvärden tas transportlagrets header bort från paket innan de skickas upp till transportlagret, som i sin tur också avlägsnar sin header innan innehållet skickas vidare till applikationsprocessen [1].

2.2.1 TCP

TCP är ett connection-oriented och tillförlitligt transportprotokoll. Detta innebär att innan två processer kan börja sända data till varandra måste de först göra en trevägs handskakning med varandra. En TCP-anslutning är alltid point-to-point, med andra ord finns den endast mellan två värdar, en sändare och en mottagare, dock är fullduplex dataöveföring också möjlig. Ifall det är en TCP-anslutning mellan två processer på två värdar A och B, är det möjligt att data kan flöda i båda riktning.

Då en TCP-anslutning har etableras via trevägs handskakning kan data börja sändas. Data från sändaren skickas först till en sändningsbuffer som satts upp vid handskakningen, TCP tar från denna buffer och skickar vidare till den mottagande värden. Den maximala mängden av data som kan skickas från buffern är begränsad av MSS (maximal segmentstorlek). Då TCP på mottagande värden tar emot ett segment placeras det i en buffer som processen på värden kan använda. TCP skapar en tillförlitlig dataöverföringsservice av typen best-effort service ovanpå IP-lagret. TCP är en tillförlitlig dataöverföringsservice som också ser till att data som finns i buffern är okorrupterad och att det inte finns fördubblingar. Den data som finns i den mottagande buffern är med andra ord exakt samma data som skickades från avsändaren [1].

2.2.2TCP segment



Figur 2: TCP-segment [8]

Ett TCP-segment består av ett headerfält och ett datafält. Storleken på datafältet begränsas av MSS-storleken. Då TCP skall skicka en stor datafil, bryts först filen ner till mindre delar av MSS-storlek. I header delen finns avsändarens port-nummer och mottagarens port nummer, som används för multiplexing eller demultiplexing av data som transporteras upp eller ner mellan de olika OSI-lagren. Andra fält som finns i TCP-headern:

-Checksumfält, för felkontroll

-32-bit sekvens nummer fält och 32-bit acknowledgment nummer fält som används för att implementera tillförlitlig dataöverföring.

-16-bit window-size, används för flödeskontroll

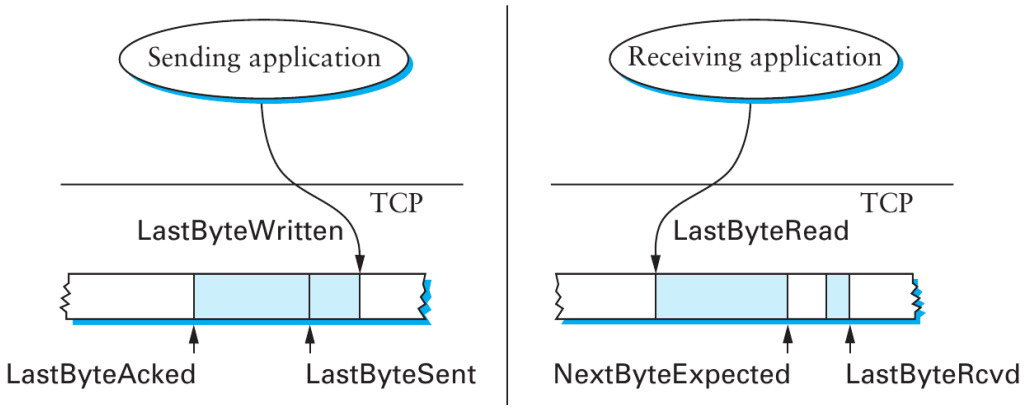
-Optionsfält, används för att bestämma MSS storlek.

-Flagfält med 6 bitar för ACK, RST, SYN, FIN, PSH och URG, för setup och teardown och kvitto [1, 2, 3].

2.2.3 TCP-sekvens och acknowledgment-nummer

Dessa fält är två av de viktigaste fälten i TCP-headern och behövs för att TCP skall kunna implementera tillförlitlig dataöverföring. TCP använder sekvens-nummer för att hålla reda på vilka bitar i byte streamen som har skickats. En byte stream från värd A till B har storleken 10000 bitar och storleken på MSS är 100 bitar, det första segmentet kommer då att ha sekvensnummer 0, det andra segmentet kommer att ha sekvensnummer 100 och så vidare. Sekvensnummer används också för flödeskontroll och felåterhämtning. Eftersom TCP är av typen full-duplex kan värd A ta emot data medan den sänder data till värd B. Acknowledgment-nummer används för att meddela värd B vilket som är nästa byte som värd A väntar att få.

2.2.4 Tillförlitlig och ordnad leverans



Figur 3: sändnings buffer och mottagnings buffer [1]

För att förstå hur TCP upprätthåller en tillförlitlig leverans av data, låt oss betrakta bilden 3 .TCP håller reda på vilka byte som har skickats och tagits emot av källvärden och destinationsvärden med två buffrar, en sändningsbuffer och en mottagnings- buffer. Varje buffer har tre pekare som varierar beroende på om det är en sändningsbuffer eller mottagningsbuffer för att hålla reda på data som skickats och mottagits. Vid sändnings buffer måste värdena på pekarna
LastByteAcked och
LastBytesent vara

LastByteAcked ≤ LastbyteSent

eftersom mottagande värden int kan ha tagit emot bitar som inte har blivit

skickade och

LastbyteSent ≤ LastByteWritten

eftersom TCP inte kan skicka bitar som ännu inte existerar.

I en mottagande buffer med samma antal pekare måste värdena på LastByteRead och NextByteExpected vara

LastByteRead < NextByteExpected

eftersom NextByteExpected kan läsas före LastByteRead och

NextByteExpected ≤ LastByteRcvd + 1

för att data skall komma i rätt ordning [1].

.

2.2.5 RTT

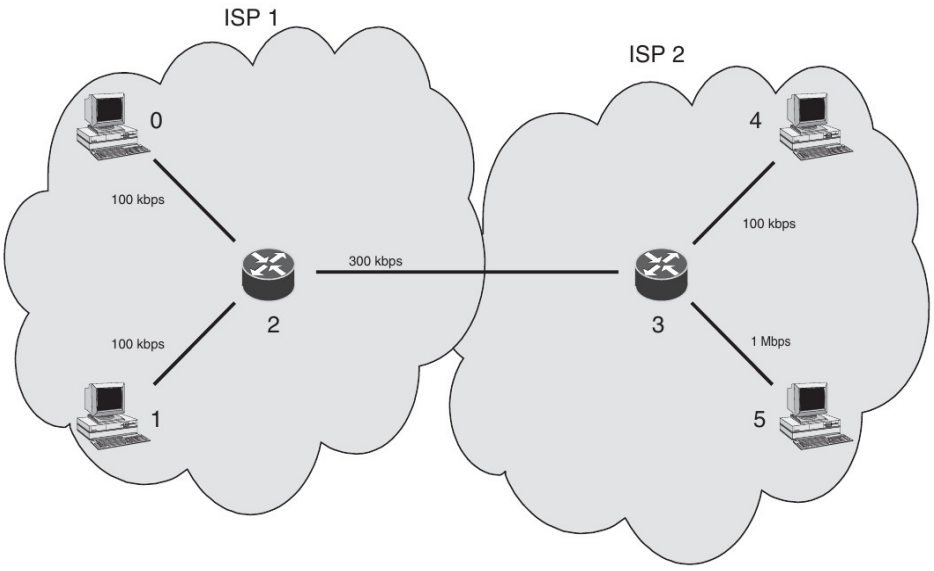
När en värd skickar ett segment med TCP startas en timer. Ifall timern går ut före värden har tagit emot och bekräftat mottagning av segmentet, skickas samma segment igen. Den tid från det att timern startas tills den går ut kallas för timeout. Det är naturligt att timeout-tiden måste vara större än den tid det tar att göra en rundtur i uppkopplingen från att segmentet skickas ut tills det har blivit bekräftat, annars skulle det leda till onödig omsändning. Dock kan inte timeout-tiden vara för stor ty ifall segmentet tappas bort på vägen kan det ta för länge för TCP att skicka segmentet på nytt och detta kan leda till datatransportfördröjning. SampleRTT är den tid det tar för ett segment skickas ut tills en bekräftelse för mottagning av segmentet erhållits. Det är ganska klart att SampleRTT kommer att variera för olika segment av olika skäl t.ex. congestion på router. Ett genomsnitt av SampleRTT måste därmed beräknas, genomsnittsvärdet kallas för EstimatedRTT och beräknas enligt följande:

$$EstimatedRTT = x \cdot EstimatedRTT + (1 - x) \cdot SampleRTT$$

Paratern x är för att jämna ut EstimatedRTT har oftast ett rekommenderat värde på 0.9. Beräkning av timeouttiden görs enligt följande formel:

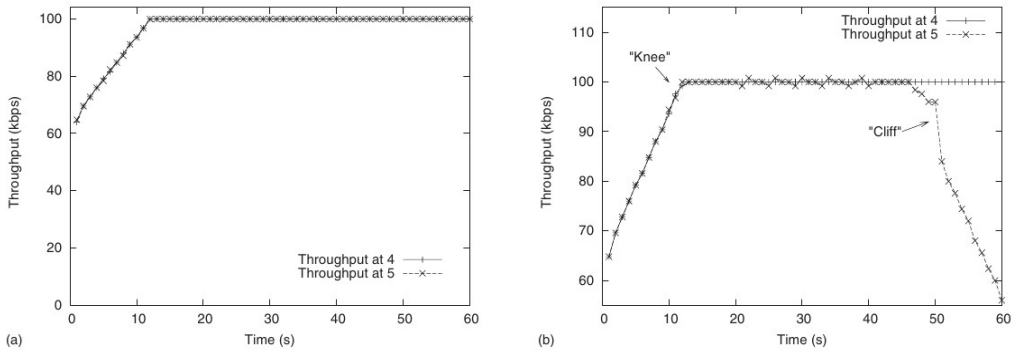
$$Timeout = 2 \cdot EstimatedRTT$$

3. Congestion collapse



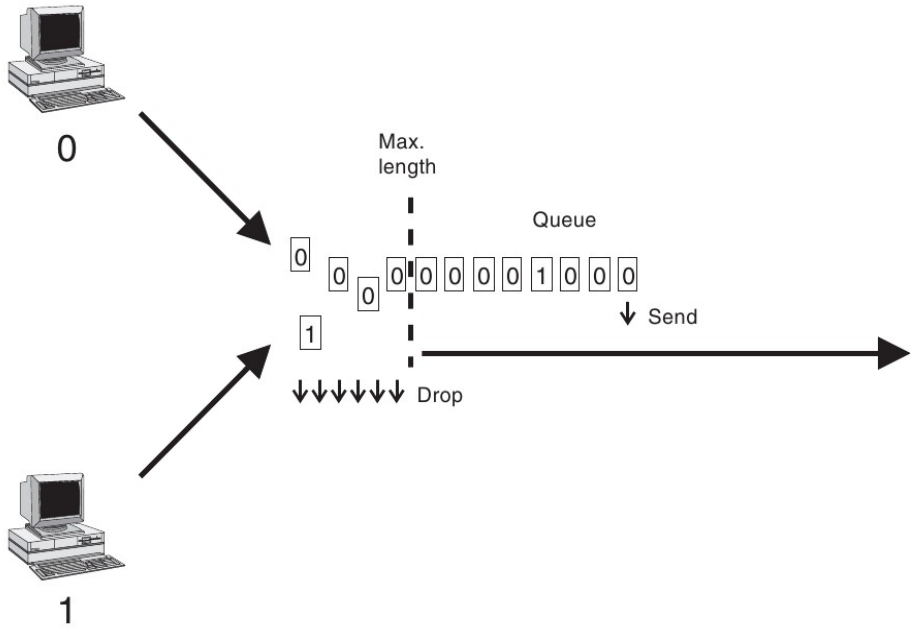
Figur 4: Exempel på congestion collapse [2]

Som exempel på hur congestion collapse kan uppstå skall vi se på ett exempel med två ISP som har två kunder var enlig bilden 4. Båda ISP är sammankopplade med en 300 kbps länk. Kund 0 skickar sin data till kund 4 och kund 1 skickar sin data till kund 5, varje kund har till gång till en 100kbps länk och ingen kongestionskontroll används. ISP 1 märker att utgående länken inte utnyttjas till sitt maximum eftersom dess två kunder endast använder 2/3 av länkens kapacitet. ISP 1 bestämmer sig för att uppgradera länken för en av sina kunder, kund 0 får nu en kapacitet på 1Mpbs. Bilden visar konsekvensen av detta beslut, a) före och efter uppgradering av länken. I bild a) ser vi att genomströmning till kund 5 och kund 4 har samma topp, I bild b) ser vi att kund 5 genomströmning minskas drastiskt efter tidpunkten 45 sekunder och faller till 0 .



figur 5: uppgradering av länk före a) efter b) [2]

Orsaken till detta fenomen är kongestion, kund 0 med den nya kapaciteten på 1Mbps överbelastar nod 2, eftersom den utgående länken till nod 3 endast är av storleken 300kbps. Detta innebär att kön i nod 2 kommer att växa för att den inte hinner vidareförmedla alla paket från kund 0. Bild 2.3 visar ungefär vad som pågår inuti nod 2, för varje paket från kund 1 finns det 10 paket från kund 0. Detta leder till att det blir en flaskhals i nod 2 som resulterar i att fördröjningen för kund 0 ökar eftersom buffern i nod 2 blir full. Alla nya paket som skickas från kund 0 förkastas på grund av full buffer vilket resulterar i att genomströmningen till nod 5 faller till 0. Detta kongestionsproblem skulle ha kunnat undvikas ifall kund 0 vetat att dess maximala genomströmning är 100kbps [2].



Figur 6: data flöde i nod2 [2]

3.1 Design av kongestionskontroll

För att undvika kongestionsproblemet som kund 0 upplevde, borde en automatisks mekanism finnas på plats för att ställa flödeshastigheten för kund 0. På vilket sätt skall då denna kongestionskontroll byggas? Det finns två alternativ för design av kongestionskontroll. Ett är end-to-end kongestionskontroll. Det här innerbär att kongestionskontroll implementeras endast i slutnoderna, med andra ord endast på transportlagret och nätverklagret erbjuder ingen hjälp. Den sändande noden kommer inte att veta om att kongestion har skett på vägen förrän timout har skett. Den andra designen för kongestionskontroll är att ha assistans från nätverkslaget. Detta skulle innebära att routern nu meddelar åt den sändande värden att ksongestion har skett. Eftersom IP-lagret inte stöder något feedback-system måste man använda end-to-end kongestionskontroll med TCP [1, 2].

3.2 TCP Congestion Control

Vi vet nu att TCP-protokollet erbjuder en tillförlitlig transportservice mellan två applikationsprocesser på två olika värdar, och att TCP kongestionskontroll mekanism måste vara av end-to-end typ, eftersom IP lagret inte erbjuder något stöd för feedback till värdarna om congestion i nätverket.

Detta har lett till att design av TCP-kongestionskontroll bygger på att varje sändare justerar sin egen sändningshastighet beroende på om det har skett kongestion i nätverket eller inte. Ifall en TCP-sändare uppfattar att det inte finns kongestion längs transportrutten till destinationsvärden, ökar den sändande värden på sin sändningshastighet. På motsvarande sätt minskar värden på sin sändningshastighet då den märker att det finns kongestion längst transportrutten.

Men för att få den här TCP-kongestionskontrollsdesign att fungera måste vi ha lösningar till följande problem. Det första problemet gäller hur en TCP-sändare skall justera sin sändningshastighet? Det andra problemet gäller hur en TCP-sändare uppfattar att kongestion har skett längs sin transportrutt? Det slutliga problemet handlar om vilken algoritm som sändare skall använda för att justera sändningshastigheten då kongestion uppstår för att maximera användning av bandbredden [1, 2].

3.2.1 Begränsning av sändningshastighet

Vi skall börja med att utforska hur TCP begränsar sin sändningshastighet. I varje TCP-uppkoppling existerar en mottagnings buffer och en sändningsbuffer i båda värdarna. Varje buffer har tre pekare för att hålla reda på hur mycket data som har sänts och tagits emot. Design av TCP-kongestionskontroll bygger på att sändaren ännu håller reda på en extra variabel, cwnd kongestionsfönster. Kongestionsfönstrets uppgift är att påtvinga en restriktion på sändarens

sändningshastighet. Den nya restriktionen är att mängden av obekräftad data från sändaren inte får överskrida minimum av cwnd och rwnd, med andra ord:

Last Byte Sent—Last Byte Acked < min {cwnd, rwnd}

För att lättare förstå sig på mekanismen för cwnd skall rwnd inte tas i beaktande, med andra ord antar vi att den mottagande buffern är så stor att rwnd kan helt ignoreras, begränsning av obekräftad data sker nu endast med cwnd-variabeln. Vi antar också att sändaren nu alltid har data att sända. Den nya restriktionen begränsar mängden av obekräftad data och därmed också sändarens sändningshastighet. Cwnd-variabeln begränsar den maximala mängden av data som får sändas för varje RTT, cwnd-variabeln uppdateras först efter att den mottagande värden har bekräftat mottagning av data. På detta sätt kan sändarens sändningshastighet beräknas enligt cwnd/RTT bitar/sekund och genom att justera variabel cwnd kan sändaren med andra ord styra dess sändningshastighet [1, 2].

3.2.2 Upptäckning av kongestion

Vi skall nu se på hur TCP-sändaren uppfattar att det finns kongestion på dess transportsrutt. En ny variabel “lost event” behövs för att hålla reda på timeout eller en tredubbel ACK från mottagaren. Då kongestion sker längs sändarens länk på grund av köer i routrar eller av andra orsaker resulterar detta i ett lost event (av timeout eller tredubbel ACK). Sändaren uppfattar nu att lost event indikerar att kongestion har skett på dess länk [1, 2].

3.2.3 Maximering av genomströmning

Vi kan nu sjustera sändninghastighet genom att modifiera värdet på variabel cwnd. Dock är det ännu oklart vilken som är den optimala sändningshastigheten vid en viss tidpunkt. Vi såg i tidigare exempel att ifall sändaren sänder med för hög hastighet kan detta resultera i kongestion i nätverket och i värsta fall en congestion collapse. Om sändningshastigheten är för liten utnyttjar inte sändaren sin bandbredd till fullo, med andra ord är det slöseri av bandbredd.

Hur skall TCP-sändaren gå tillväga för att bedöma sin sändninghastighet på så sätt att det inte leder till kongestion i nätverket men samtidigt ändå utnyttjar bandbredden till fullo? För att få svar på detta måste vi ta i beaktande följande:

-Borttappade segment implicerar att kongestion har skett längs länken och att sändaren borde minska på sin sändningshastighet. Sändaren måste med andra ord justera cwnd-värdet då lost event sker.

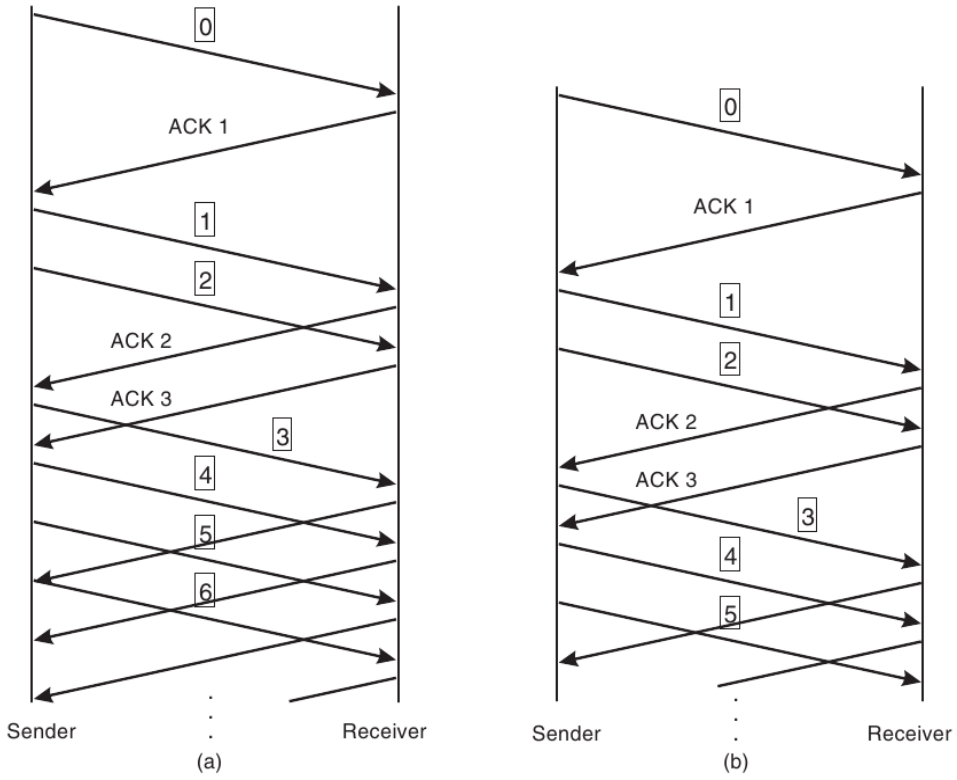
-En bekräftelse att mottagning av ett segment skett indikerar att segmentet har kommit fram till sin destination och ingen kongestion har skett på vägen. Sändaren kan med andra ord öka på sändningshastigheten så länge den får ACK för sina obekräftade segment. Eftersom ACK implicerar att det sända segmentet har kommit fram och att ingen kongestion har skett på vägen kan med andra ord en ökning av värdet ske.

-Vi vet nu att varje ACK är en indikation på att ingen kongetion har skett och lost event att kongestion har skett på vägen. TCP:s kongestionskontrollsmekanism bygger på dessa principer. Genom att undersöka det bästa värdet på cwnd kan sändaren utnytta sin bandbredd till fullo, med andra ord ökar sändaren sändningshastigheten tills ett lost event sker. När detta sker minskar sändaren på sändningshastigheten och undersöker sedan om det ännu uppstår kongestion längs vägen. Om inte så ökas sändningshastigheten igen. Det måste noteras att nätverket inte ger någon specifik indikation på kongestion åt sändaren utan det görs lokalt av sändaren själv genom kontroll av ACK och lost event.

Det är dessa principer som dagens TCP-kongestionskontroll bygger på och har beskrivits för första gången i [Jacobson1988]. Kongestionskontrollsmekanismen har tre viktiga komponenter: slowstart, congestion avoidance och fast recovery, varav slowstart och congestion avoidance är obligatoriska komponenter i TCP. Fast recovery rekommenderas åt sändare men det är inte ett måste att ha [1, 2].

3.2. 4 Slow start

Slow start är designat på följande sätt: då en TCP uppkoppling startas upp är värdet på cwnd initierat med ett så litet värde som möjligt, oftast av storleken 1 MSS. Det här innebär att TCP:s initiella sändningshastighet är 1 MSS/RTT, t.ex. om MSS = 500 bitar och RTT = 200 msec skulle initiell sändningshastighet vara 20 kbps. Eftersom den tillgängliga bandbredden oftast är mycket större än 1 MSS/RTT är det naturligt att sändaren snabbt vill kunna finna gränsen för den maximala bandbredden.



Figur 7: Slow start a) congestion avoidance b)

Slow start startar initiellt med värdet på cwnd = 1 MSS och ökar med 1MSS varje gång ett segment har blivit bekräftat från den mottagande värden, vi kan se detta på bilden 7. TCP börjar med att sända ett segment och väntar sedan på bekräftelse. Då segmentet har blivit bekräftat ökar sändaren på värdet av cwnd med 1 MSS och skickar ut 2 MSS. Då dessa två segment har blivit bekräftade ökas cwnd till 4 MSS och så vidare. Den här strategin innebär att sändningshastigheten fördubblas för varje RTT, så fastän TCP börjar med långsam initiell sändningshastighet växer den mycket snabbt.

När skall tillväxt av sändningshastighet avslutas? När ett lost event sker, med andra ord när kongestion har skett under transporten indikerat av en timeout, ändrar TCP-sändaren på värdet för cwnd till 1 MSS och börjar om med slow start processen på nytt. Den ändrar också på värdet på ssthresh (slow start threshold) till cwnd/2, med andra ord hälften av kongestionsfönstrets värde då en kongestion inträffar. Slow start avslutas också då värdet av ssthresh är lika med nuvarande cwnd värde ty ssthresh är halva värdet av cwnd innan föregående kongestion inträffade. Då cwnd värdet är lika med ssthresh avslutas slow start och TCP övergår till congestion avoidance.

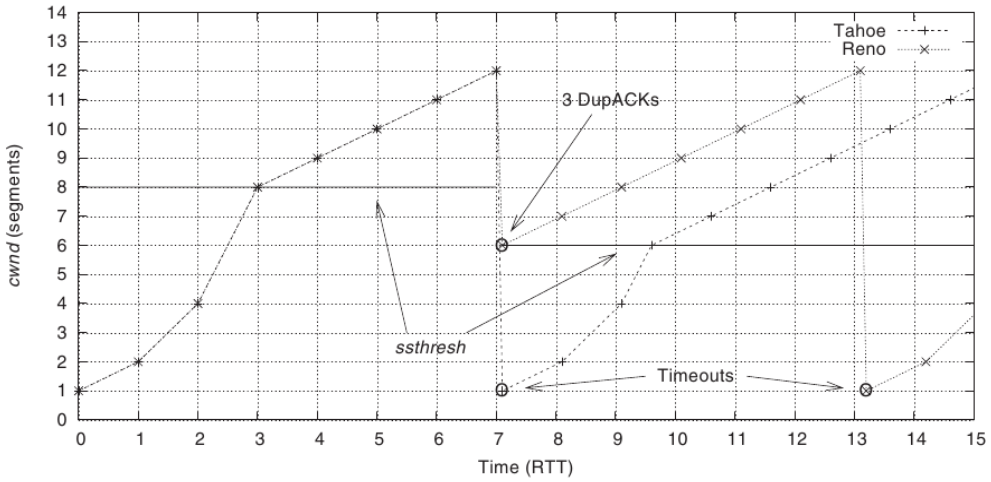
3.2.5 Congestion Avoidance

För att TCP skall övergå från slow start till Congestion avoidance tillstånd måste värdet på cwnd vara hälften av vad det var före föregående kongestion inträffade.Således istället för att dubbla värdet av cwnd för varje RTT övergår TCP till ett mera konservativt läge och ökar värdet av cwnd endast med en 1MMS för varje RTT.

När skall congestion avoidance ökning avslutas? TCP congestion avoidance algoritmen fungerar på samma sätt som slow start. Då en timeout sker sätts värdet av cwnd till 1 MSS och värdet av ssthresh ändras till hälften av föregående värde av cwnd före en kongestion har skett. Dock om lost event är utlöst av tredubbel ACK istället för timeout är ändring på cwnd lite annorlunda. I det här fallet halveras värdet av cwnd och ssthresh ändras till hälften av föregående värde av cwnd före en kongestion har skett och TCP övergår till fast recovery läge.

3.2.6 Fast Recovery

I fast recovery läge ökar värdet på cwnd med 1 MSS för varje dubbel ACK som mottagits före det borttappade segmentet som orsakade att TCP övergår till fast recovery läge. Då ACK för det saknade segmentet mottas går TCP tillbaka till congestion avoidance läge. Ifall en timeout sker under fast recovery läge övergår TCP till slow start. Fast recovery är rekommenderat men är inte ett måste för TCP.

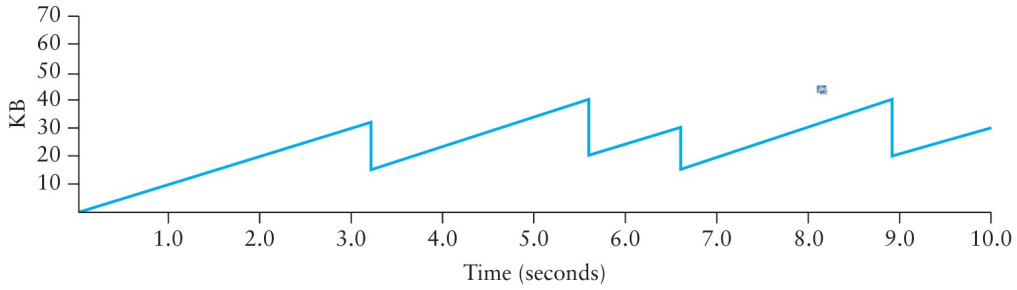


Figur8 : Evolution av cwnd med TCP Tahoe och Reno [2]

Bild 8 illustrerar evolution av TCP kongestionsfönstret för Reno och Tohea. Vi ser att ssthread är initiellt 8 MSS och för de första åtta RTT ger både Reno och Tohea algoritm samma resultat. Cwnd växer linjärt ända tills ett tredubbelt ACK event sker, vid tidpunken 8 RTT. Värdet på ssthread ändras till 6 MSS ty cwnd/2 = 6 MSS. Med TCP Reno är cwnd 6 MSS, och TCP tahoe är 1MSS.

3.2.7Additive-increase, multiplicative decrease (AIMD)

TCP congestion kontroll är också oftast kallad för Additive-increase, multiplicative decrease, det här kommer från att TCP:s kongestionskontroll ökar värdet på cwnd med addition och minskar värdet på cwnd med division. Det här resulterar i ett sågtandsmönster bild9.



Figur 9: AIMD [1]

4. Avslutning

Internets popularitet har vuxit explosionsartat de senaste åren och är för många det naturliga valet av medium för både jobb och nöje. Dock lider internet av ett spädbarnsproblem nämligen kongestion. Det var aldrig byggt för transport av stora mängder av data över stora avstånd. Fastän nya teknologier har tillkommit kan inte uppgradering eller byte ske så lätt ty det skulle leda till stora kostnader. Istället måste förbättring ske genom optimering av existerande system. I den här avhandlingen har jag presenterat vad kongestion är och vilka förbättringar som har gjorts för att lösa kongestionsproblemet. Dock har inte alla lösningar ännu presenterats. Avhandlingen behandlade mest kontroll av kongestion efter att det upptäckts i systemet. Bättre lösning på kongestionskontroll som finns men inte togs upp i avhandlingen är system där man förutspår när kongestion kommer att ske och vidtar åtgärder före den kan påbörja.

Referenser

[1] Larry l. Petterson & Bruce s. Davie 2003 “Computer Networks a system approach”

[2] Michael Welzl, 2005 ”Network congestion control: managing Internet traffic”

[3] Andrew S Tanenbaum Computer Networks —,4th Edition

[4] Sally Floyd in September 2000 “Congestion Control Principle”

[5] Jacobson, V. Congestion avoidance and control. In

Proceedings of SIGCOMM ’1988

[6] Jain, R. Divergence of timeout algorithms for packet

retransmissions. In Proceedings Fifth Annual International Phoenix Conference on Computers and Communications (Scottsdale, AZ, Mar. 1986).

[7] Introduction to Network Protocols hämtad 2.21.2011
http://www.codeguru.com/cpp/sample_chapter/print.php/c12219

[8] TCP in high speed networks hämtad 2.21.2011

http://www.tkn.tu-berlin.de/curricula/ss96/bla/tcp_hs.html