TCP och UDP-nivån

- I det här kapitlet går vi igenom hur UDP och TCP använder portnummer. TCP:s trevägs handskakning förklaras. Headers för TCP och UDP gås igenom. Flödeshanteringen i TCP inklusive sliding windows förklaras kortfattat.
- Kapitlet är tänkt att kunna läsas fristående.

TCP-nivån på din dator kan du inte göra så mycket åt. Det finns inte så mycket att ställa in, och programmen väljer själva om de ska använda TCP eller UDP. Men det betyder inte att du inte har nytta av att känna till hur TCP och UDP fungerar. Det är TCP och UDP som ser till så att flera program kan använda en och samma IP-nivå. Det är flödeshanteringen i TCP som måste fungera effektivt så att kanalen mellan sändare och mottagare utnyttjas så effektivt som möjligt.

I kapitlet IP-grunder tittade vi på de funktioner som TCP lägger till:

- TCP gör kommunikationen förbindelseorienterad
- TCP lägger till portnummer
- TCP delar upp dataflödet i lagom stora paket
- TCP lägger till sekvensnummer på paketen så mottagaren kan kvittera vilka paket som kommit fram
- TCP hanterar omsändning av paket

Hur paket ska sändas om och kvitteras är relativt komplicerat om det ska ske på ett effektivt sätt. Och vad som är effektivt beror på sammanhanget. Vi lägger in hela den här hanteringen under begreppet flödeshantering nedan.

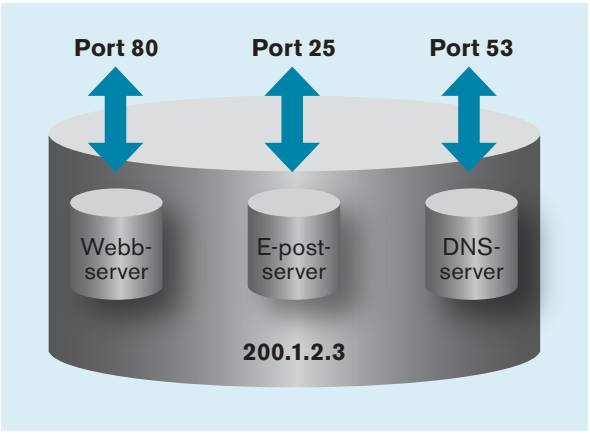
Portnummer

Applikationer får inte kommunicera med IP direkt.

I arkitekturen kring TCP/IP finns inte så många fundament men ett av dem är att en applikation inte pratar direkt med IP. TCP eller UDP ska finnas i mellan och lägga till portnummer så att flera applikationer delar på IP:s funktioner. Man kan också se det som olika nivåer. Med hjälp av IP adresserar vi själva datorn, så på IP-nivå handlar det om kommunikation mellan datorer. Med

hjälp av TCP och UDP adresserar vi applikationer (som vi också kan kalla program eller processer). TCP och UDP möjliggör kommunikation mellan applikationer.

Med hjälp av portnummer blir det bland annat möjligt att starta fler samtidiga filöverföringar och hämta filer från fler servrar på en gång eller att öppna flera fönster i en webbläsare och öppna fler sessioner på en gång. Med hjälp av portnummer kan en server vara webbserver och e-postserver på en gång.



En server kan lyssna på fler olika portnummer.

Det finns möjlighet att använda 65 536 olika portnummer (2^{16}). De 1 023 lägsta numren kallas för ”wellknown ports” och används av välkända kommunikationsprotokoll för filöverföring, e-post, namnuppslagningar, surfning och mycket mer. Vilket nummer ett protokoll använder finns bland annat fastslaget i RFC 1700 (Assigned Numbers). RFC 1700 är sakligt sett inte aktuell längre (den är inte ”standards track”) men i stort sett följs den fortfarande. En uppdaterad lista kan hämtas på:

http://www.iana.org/assignments/port-numbers

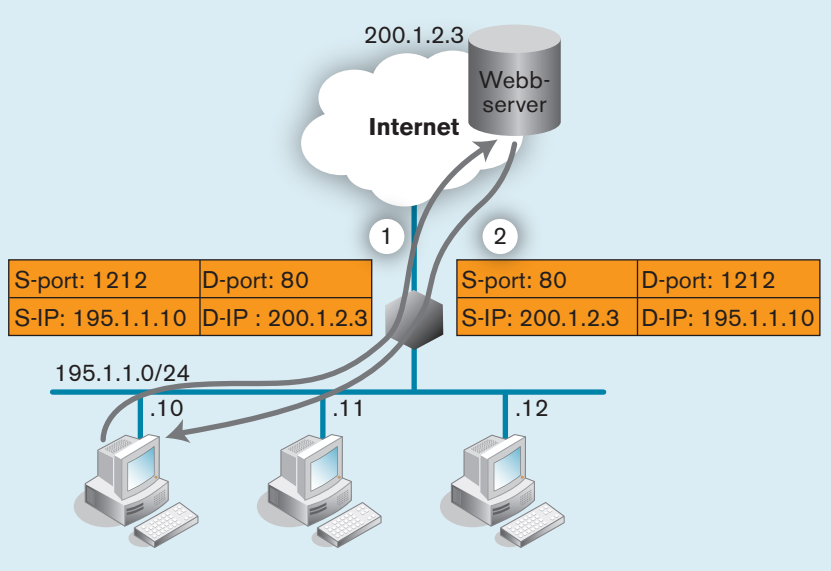
Portnummer	Protokoll	Anm
20	FTP	FTP Kontroll
21	FTP Data	
22	SSH	Secure Shell
23	Telnet	
25	SMTP	
53	DNS	
67	BOOTP	BOOTP och DHCP servers
68	BOOTP	
80	HTTP	
88	KERBEROS	
110	POP	
111	RPC	Sun Remote Procedure Call
123	NTP	Network Time Protocol
137	NetBIOS	
143	IMAP	För e-post
161 och 162	SNMP	Nätverksövervakning
443	HTTPS	Secure HTTP
1080	SOCKS	Proxy Server
3389	Remote Desktop	

Några populära protokoll och de portnummer de använder.

Flera av dessa protokoll använder både TCP och UDP. Om möjligt behåller man i så fall samma portnummer. Ett exempel är DNS för namnuppslagningar. När enkla namnfrågor ska göras används UDP. Men när större överföringar ska göras mellan två namnservrar så använder de TCP. Allt detta med port 53 som serverport.

TCP/IP ger en möjlighet för alla slags program att kommunicera över alla slags WAN och LAN.

Servrar använder väldefinierade portnummer. Klienter däremot använder slumpartade portnummer med nummer över 1 023. Om vi startar en webbläsare och skriver in webbadressen till en webbsida så etableras en session från port 1 212 till mottagarport 80 i fallet nedan:



Hur portnummer används.

Webbklienten vet således att den ska etablera kontakt med port 80. När servern ska skicka tillbaka ett paket håller den bara reda på vem (IP-adress och portnummer) som skickat frågan. En session identifieras inte bara av den egna IP-adressen och portnumret utan även av motsvarande information på andra sidan. Alltså kan servern returnera svaret till klientens portnummer.

Ska man vara noga så delas portnummer in i tre kategorier: Well Known Ports, Registered Ports och Dynamic Ports.

Du kan enkelt kontrollera vilka portar din egen maskin lyssnar på med hjälp av kommandot ”netstat -an”. Kommandot returnerar en tabell med de portar som är öppna för kommunikation (135, 455, 5225, 5226 och 8008 nedan). Adressen 0.0.0.0 innebär att datorn tar emot anrop från vilken annan adress om helst. Adress 127.0.0.1 innebär att programmet bara accepterar anrop från den egna maskinen (127.0.0.1 kallas även localhost). TCP/IP gör det alltså möjligt att bygga tjänster som bara kan användas av den egna maskinen eller från vilken maskin som helst på Internet.

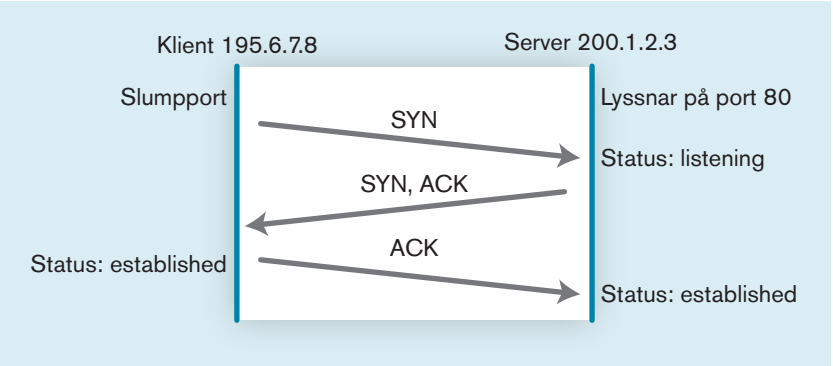
```
C:\>netstat -an

Aktiva anslutningar

  Prot.  Lokal adress            Extern adress          Status
  TCP    0.0.0.0:135             0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445             0.0.0.0:0              LISTENING
  TCP    0.0.0.0:5225            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:5226            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:8008            0.0.0.0:0              LISTENING
  TCP    127.0.0.1:1025          0.0.0.0:0              LISTENING
  TCP    127.0.0.1:1031          127.0.0.1:5225         CLOSE_WAIT
  TCP    127.0.0.1:1032          127.0.0.1:5225         CLOSE_WAIT
  TCP    127.0.0.1:1033          127.0.0.1:5225         CLOSE_WAIT
  TCP    127.0.0.1:1034          127.0.0.1:5225         CLOSE_WAIT
  TCP    127.0.0.1:1035          127.0.0.1:5226         ESTABLISHED
  TCP    127.0.0.1:1039          127.0.0.1:5225         CLOSE_WAIT
  TCP    127.0.0.1:1043          127.0.0.1:5225         CLOSE_WAIT
  TCP    127.0.0.1:1654          127.0.0.1:5225         TIME_WAIT
  TCP    127.0.0.1:1655          127.0.0.1:5225         TIME_WAIT
  TCP    127.0.0.1:5226          127.0.0.1:1035         ESTABLISHED
  TCP    127.0.0.1:5679          0.0.0.0:0              LISTENING
  TCP    127.0.0.1:8005          0.0.0.0:0              LISTENING
  TCP    127.0.0.1:10110         0.0.0.0:0              LISTENING
  UDP    0.0.0.0:445             *:*
  UDP    0.0.0.0:500             *:*
  UDP    0.0.0.0:1054            *:*
  UDP    0.0.0.0:4500            *:*
  UDP    127.0.0.1:123           *:*
  UDP    127.0.0.1:1900          *:*
```

Tabellen, som du får fram via kommandot "netstat -an", visar de portar som är öppna för kommunikation.

Kommandot netstat visar även vilken status varje session har. I läget "listening" agerar processen som server och ligger och väntar på anrop. I läge "established" har den etablerat en session med andra änden och överföring av data är möjlig. I RFC 793, som beskriver TCP, finns de tillstånd som TCP kan ha beskrivna i en så kallad tillståndsgraf. I den bästa av världar skulle established och listening vara vanligast, men eftersom TCP inte kan ta för givet att alla paket kommer fram blir nedkoppling komplicerat och ett flertal tillstånd kan inträffa. Några exempel finns med ovan i form av TIME_WAIT och CLOSE_WAIT.

Handskakning

I TCP etableras en session med hjälp av handskakning i tre steg. Eftersom det är två jämlikar (peers) som ska etablera en session och kanalen under inte är tillförlitlig blir det lite omständligare än ett enkelt open/accept förfarande. De tre stegen motsvarar helt enkelt en begäran om att starta en session, en kvittens av detta följt av ett meddelande om att även andra sidan är intresserad och slutligen en kvittens.



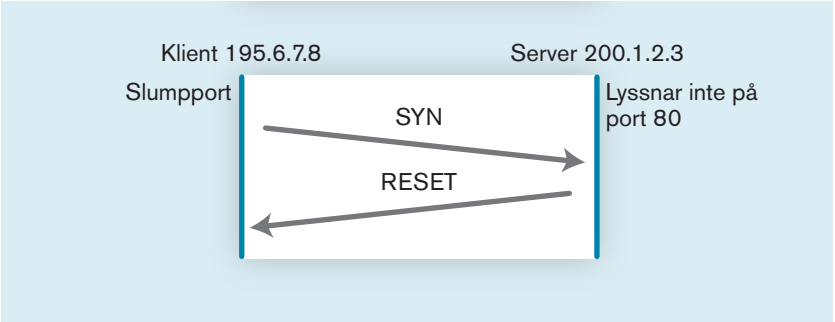
Handskakning i TCP.

TCP är förbindelseorienterat (connection-oriented är den engelska benämningen till skillnad från connectionless). Man handskakar och etablerar sessionen mellan två punkter innan överföring av information startar. När kommunikationen är etablerad kan den hanteras som en så kallad socket som identifieras av IP-adresser och portnummer i bägge ändar.

På motsvarande sätt avslutas en session med tre paket: FIN, FIN-ACK samt ACK. FIN står givetvis för finish, och denna begäran kan komma från server eller klient. Vid avslutning av en session kan det dröja mellan de två FIN-paketen. Ena sidan kanske vill avsluta men den andra sidan anser sig ha mer data att sända. Först när bägge sidor skickat FIN och det sista paketet kvitterats är sessionen avslutad. Nedkoppling i TCP är komplicerad då TCP baseras på en otillförlitlig underliggande nivå. Avsändaren kan inte vara säker på att FIN-paketet kommer fram. Jämför med ett telefonsamtal över en mycket brusig förbindelse. Hur kan man vara säker på att få fram sitt ”klart slut” och är det mitt meddelande eller andra sidans kvittens som inte kommer fram? Man kan faktiskt bevisa att det inte går att göra tillförlitlig nedkoppling över en otillförlitlig kanal. TCP:s lösning är att starta olika timers så man tillslut betraktar förbindelsen som nedkopplad.

För att påskynda nedkoppling så finns även en möjlighet att använda RESET. Kommunikation bryts då omedelbart, inga kvittenser skickas och buffrar etc kastas. RESET används av TCP i vissa fall då ett oväntat paket mottages. RESET används även om

Om en server inte lyssnar på en port kan den besvara uppkopplingsbegäran (SYN) med RESET. Klienten ska då förstå att denna port inte är aktiv. Metoden används bland annat av hackers vid en så kallad portskanning. Angriparen skickar SYN-paket till portar som kan vara öppna. Normalt får man inget svar eller RESET, om man istället erhåller SYN+ACK så vet man att porten är i lyssnande läge.



en klient försöker anropa en port som inte används på en server som ett sätt att meddela att ingen process svarar på den porten eller att servern inte kan hantera fler inkommande sessioner.

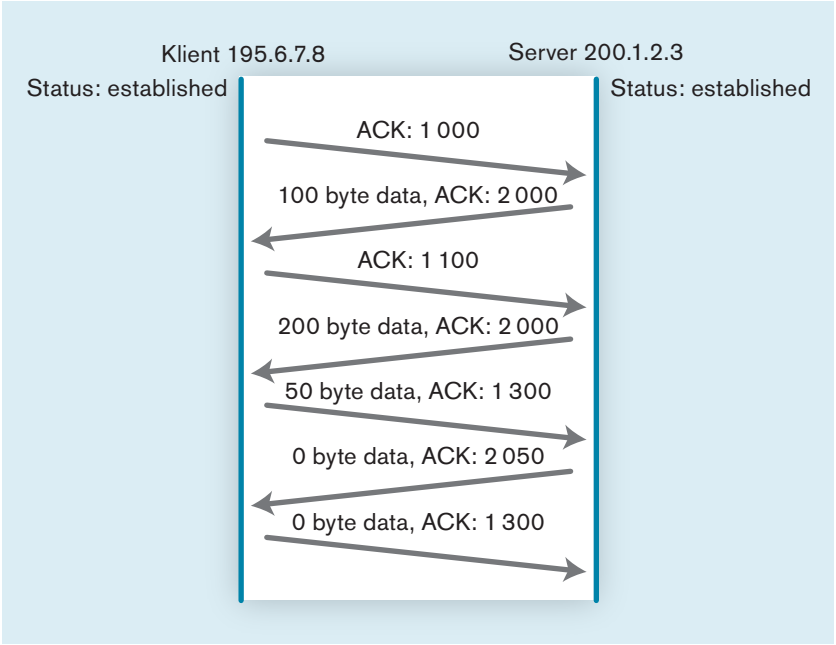
Dela upp dataflödet i paket

En av TCP:s viktigaste uppgifter är att upp mot applikationen leverera tjänsten ”ett kontinuerligt dataflöde” samtidigt som underliggande IP-nivå förväntar sig paket av en viss maximal storlek. TCP tar därför en liten genväg och kontrollerar vad länknivån har för MTU (Maximum Transmision Unit) och sedan delas dataflödet upp i paket. Paketen numreras och kvitteras på ett sofistikerat sätt.

Ett enkelt protokoll är att man numrerar varje paket med start på ett och sedan ökar med ett. TCP är mer komplicerat och hämtar första värden från en räknare som går hela tiden. Om vi tänker oss att en användare eller en process skulle starta fler sessioner inom en kort tid så hade det varit en nackdel om alla startar på ett. Sannolikheten ökar då att man felaktigt kvitterar ett paket. Och det är just sekvensnummer man synkroniserar med SYN-paketet, det första sekvensnummer som används i sessionen (SYN står för synchronize). Att sekvensnumren startar på ett nummer som är unikt och jobbar med höga siffror gör det dessutom svårare för en angripare att ta över eller koppla ner en session. (En angripare kan ju enkelt krångla till det för en användare genom att skicka nedkopplingspaket (RES-flaggan) till den server som användaren kommunicerar med.)

För det andra så kvitteras inte varje enskilt paket. Istället kvitterar mottagaren hur många byte som tagits emot i form av nästa förväntade sekvensnummer. Avsändaren räknar på motsvarande sätt ut vad mottagaren borde kunna kvittera (senast kvitterade nummer ökat med antal skickade byte). Fördelen med detta är lätt att förstå:

Paket nummer tre, det med ACK = 1 100, behöver inte skickas i sekvensen utan servern kan från paket fem se att jämfört med



Klienten behöver egentligen inte sända paket nummer tre (se text).

paket ett har mottagaren tagit emot 1 300–1 000 = 300 byte vilket stämmer med antal skickade byte.

Att inte varje paket behöver kvitteras har stor betydelse för prestanda. Om TCP ska fungera över kontinenter så skulle kvittens av varje paket dra ner prestanda då ”roundtrip delay” kan vara över hundra millisekunder. Istället kan TCP kvittera fler paket på en gång. I Windows XP skickas kvittenser tidsstyrt var 500 millisekund. Man utnyttjar i TCP även möjligheten att skicka kvittenser samtidigt som man skickar data (det är ju bara att sätta giltiga värden i ACK-fältet). Tekniken att om möjligt skicka kvittenser och annan kontrollinformation samtidigt som man skickar data brukar kallas piggybacking.

För att dessutom öka prestanda och tillförlitlighet använd en teknik som kallas sliding windows, men mer om den senare.

TCP Headern

De funktioner som TCP behöver hantera lägger vi in i headern. I bilden på nästa sida visas headern med fyra byte per rad. Totalt är headern 20 byte lång plus eventuella optioner. Maximal längd på headern är 60 byte.

De fyra första byten anger avsändande respektive mottagande portnummer. Fyra byte används för sekvensnummer, vilket gör att sekvensnumret kan räknas upp till drygt fyra miljarder, vilket normalt är tillräckligt högt för att inte sekvensnummer ska upp-

Avsändande portnr			Destination portnr
Sekvensnummer			
Kvittensnummer			
HLEN	Res.	Flaggor	Window
Checksumma			"Urgent pointer"
Optioner			

TCP-header.

repas under en session. Kvittensnumret används för att kvittera hur många byte som kommit fram.

HLEN anger längden på TCP-headern och räknas precis som med IP i 32-bitars ord. Typiskt värde är alltså fem motsvarande 20 byte. HLEN består av fyra bitar och därför blir maximal längd på headern 60 byte. Sedan följer fyra till sex bitar som är reserverade för framtida användning.

För att hantera handskakning med mera används ett antal flaggor:

URG Urgent. Används ej i modern TCP
ACK Anger att kvittensfältet har ett giltigt värde
PSH Push
RST Reset. Kan användas vid nedkoppling av en session
SYN Synchronize. Används vid etablering av en session
FIN Finish. Användas vid normal nedkoppling av en session.

ACK-flaggan har i stort sett alltid värdet ett eftersom man passar på att kvittera med hur mycket data som mottagits. RST, SYN och FIN används vid upp- och nedkoppling av en förbindelse.

PSH-flaggan används mycket och hänger samman med en funktion inom TCP som heter push. Det är TCP som avgör när data ska sändas, därför kan TCP samla ihop data innan det skickas iväg. En ovanpåliggande applikation kan dock skicka kommandot "push" vilket meddelar TCP att detta data ska skickas och inte får buffras. Dessutom sätts PSH-flaggan i motsvarande paket. Mottagande TCP-stack ska på motsvarande sätt inte buffra inkommande data utan skicka upp det till mottagande applikation omgående.

Dessa är de klassiska flaggorna från RFC793, men tillägg finns i RFC2481, A Proposal to add Explicit Congestion Notification (ECN) to IP. Före Urgent-flaggan finns då två flaggor: Congestion Window Reduced (CWR) och ECN-Echo. Dessa används för att förbättra flödeshanteringen i TCP.

Fältet Window används för att hela tiden meddela den andra sidan om hur stor buffert man har. Det vore ju ett slöseri att skicka en massa data som sedan inte mottagaren har möjlighet att bearbeta.

Checksumman används för att kontrollera riktigheten i headern samt lite mer. Se avsnittet om UDP.

Fältet "Urgent Pointer" har som man kan misstänka att göra med flaggan Urgent. Varken flaggan eller fältet används i modern TCP.

TCP optioner

Eftersom flödeshantering och felhantering är ett område som utvecklats och fortfarande utvecklas används optioner till TCP relativt ofta. När en session etableras kan sändare och mottagare handskaka om vilket stöd för flödeshantering de använder.

En typisk option kan ha värdet 0x0204 05b4 0101 0402. Detta är en option om åtta byte som Windows XP kan annonsera.

- 02 anger typ av option, här ”Maximum Segment Size”
- 04 anger optionens längd i byte
- 05b4 är det hexadecimala talet 1460, dvs maxlängden när man tar emot paket. Annat relativt vanligt värde är 1 380 decimalt.
- 0101 är NOOP (no operation)
- 0402 är flaggor som meddelar att man har stöd för SACK enligt RFC 2018.

Hur stora paket som ska användas under sessionen är en svår fråga. Det optimala om man ska skicka mycket data är så stora som möjligt men utan risk för att fragmentering inträffar. Detta är svårare än det låter eftersom IP-nivåns routrar kan ändra den väg som paketen routas, så MTU kan variera över tiden. Ett annat problem är att IP kan välja att lägga till optioner eller tunnla IP i IP eller i IPsec, detta påverkar hur stor last vi kan skicka med. Det här därför blivit allt vanligare att man inte använder 1 500 byte (typiskt värde för Ethernet v II) utan går ner lite till 1 460 för att ha marginal.

Windows Scaling Option är en annan relativt vanlig option. I grunden har TCP stöd för en buffert om 64 Kbyte. Om kanalen har hög bandbredd men lång fördröjning är detta en begränsning, detta kan gälla en kanal över mobila nät, via satellit eller mellan kontinenter.

Om vi tänker oss att vi har en förbindelse om 8 Mbit/s och en fördröjning (RTT Round Trip Time, det vill säga fram och tillbaka) om 100 ms så behöver man en buffert om $8\,000\,000 \times 0{,}1 = 800$ kbit vilket motsvarar 100 kbyte. (Avsändaren bör kunna hålla hela kanalen full med data tills paketen har kvitterats. Dubblar vi fördröjningen behövs en dubbel så stor buffert.) 64 kbyte är ett gammalt värde. Om vi tänker oss en satellitkanal om 500 ms fördröjning (RTT) så skulle detta ge en kapacitet om $64 \times 8 \times 1\,024/0{,}5$ vilket ger cirka 1 Mbit/s. Snabbare kanaler än så skulle inte ge någon prestandaförbättring på grund av TCP:s begränsningar. Idag med Gigabithastigheter uppträder problemet redan vid 0,5 mikro-

sekunder. För att kunna använda större buffrar finns optionen Windows Scaling Option. Man meddelar vilken faktor fönsterstorleken ska multipliceras med under handskakningen vilket gör att mångdubbelt större buffert kan användas.

Windows Scaling är relativt grundläggande kommunikationsteori men tyvärr relativt okänd bland systemarkitekter och designers. Organisationer köper snabba internationella förbindelser men kombinerar dessa med gamla TCP/IP-implementationer från 1990-talet som ska göra backup över nätet och det kan vara en dålig kombination. En snabb analys kan visa om Windows Scaling används, annars måste man byta TCP/IP-stack. Problemet finns även med enkla handdatorer och långa fördröjningar över mobila tjänster.

Ytterligare en option är tidsstämpling vilket kan användas för att beräkna kanalens fördröjning vilket bland annat kan användas för att beräkna när en kvittens borde tagits emot.

Flödeshantering

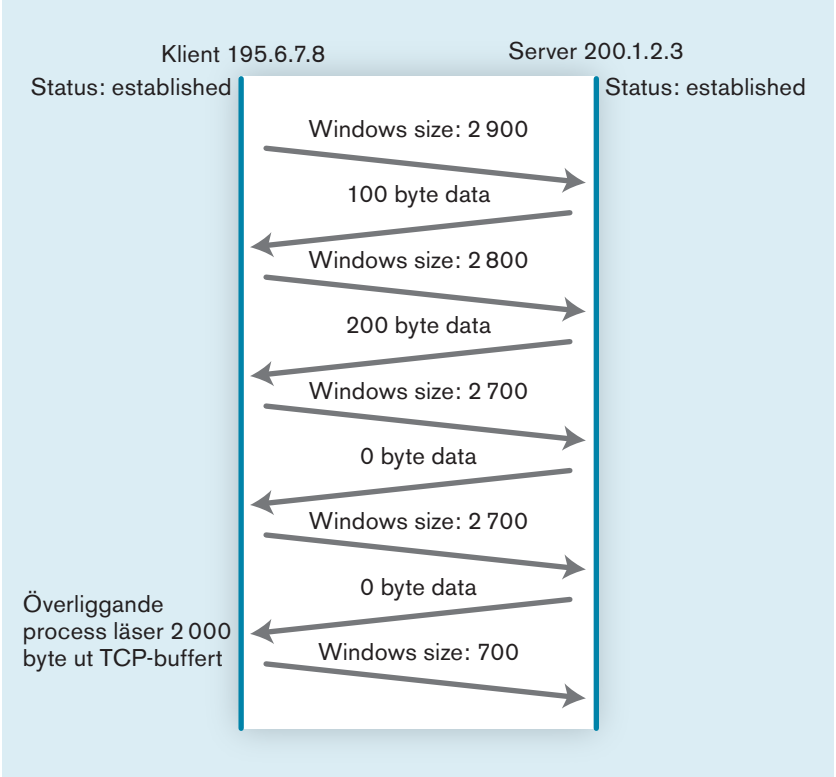
TCP är ett exempel på hur Internet designats. Jämfört med telefonnätet så ligger många funktioner i TCP hos avsändare och mottagare och inte i själva nätet. Routrar skickar paket vidare så snabbt de kan, det är i TCP som funktioner för att hitta borttappade och duplicerade paket ligger. På engelska kallas denna design för edge centric. I själva verket är det ett komplext samspel mellan hur TCP försöker utnyttja nätet och hur routrar hanterar dataströmmar.

För att uppskatta hur effektiv och tillförlitlig flödeshantering TCP använder så bör man ha använt ett tidigare protokoll som Kermit eller X-modem. Det kändes ofta som att det räckte med att någon stängde en dörr ovarsamt i närheten så avbröts kommunikationen och man fick börja från början. Äldre protokoll var känsliga för störningar och utnyttjade inte kanalen så effektivt. Det är lätt att se hur effektivt TCP är. Om man sätter sig på en långsam förbindelse på exempelvis 256 kbit/s och ska överföra en fil om 8 Mbyte så borde det minst ta $8 \times 8 \times 10^6 / 256 \times 10^3 = 0{,}25 \times 10^3 =$ 250 sekunder. Ofta når man 70–90 procent av detta teoretiska värde. Använder man en applikation som visar momentana värden kan man också se hur överföringshastigheten ökar successivt för att hitta kanalens maximum. Man kan också prova med att dra ut sladden några sekunder och sätta tillbaka den. TCP kommer oftast att klara störningar. Vi ska titta översiktligt på de metoder TCP använder.


Testa själv

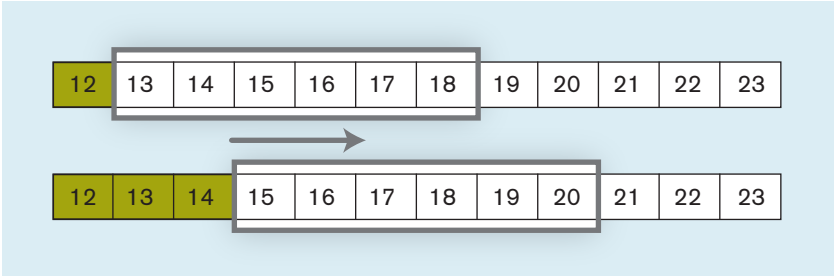
För att inte skicka över data som inte kan tas emot meddelar varje nod hela tiden hur stor mottagande buffert den har. Det är till

detta som fältet ”Windows” används. I början av en session skickas denna information men buffertens storlek varierar med tiden. TCP skickar ju upp informationen till mottagande process men det varierar i vilken hastighet applikationen tar emot data.



Under sessionen meddelar TCP den andra ändpunkten hur stor buffert den har för mottagning.

För att kunna skicka data effektivt använder TCP en metod som kallas sliding windows. Sliding windows används även av andra program som Z-modem. Sliding windows gör processen mer komplex men prestanda, speciellt vid borttappade paket, blir avsevärt bättre. Vi tittar på en lite förenklad modell jämfört med vad som används i TCP. TCP använder som bekant antal byte i sin sekvensräknare och för att kvittera.



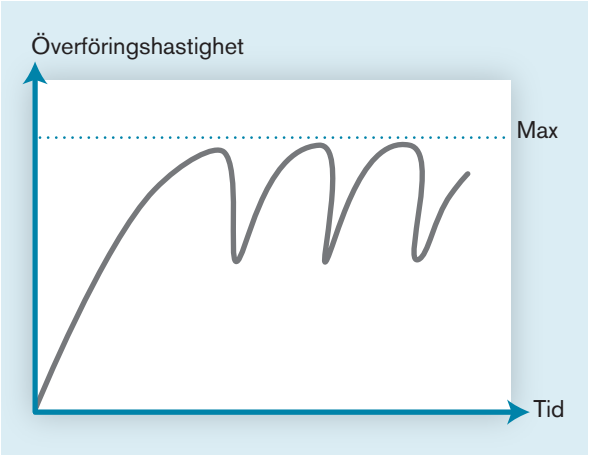
Sliding windows.

Sliding windows ökar prestanda betydligt jämfört med om varje paket ska kvitteras för sig.

Om vi har en fönsterstorlek om sex byte ovan och tänker oss att byte 12 är kvitterad så kan byte 13–18 sändas iväg. Om sedan byte 13–14 kvitteras kan fönstret glida fram och skicka byte 19 och 20. Om något händer och till exempel 13 inte kvitteras men däremot 14–17 så får vi sända om byte 13. När sedan 13 kvitterats kan fönstret glida fram till 18 och skicka fem nya byte.

Hur snabbt TCP skickar ut paket får en avgörande betydelse på prestanda. Modern TCP använder flera tekniker. Ett grundläggande antagande är att kanalen mellan två ändpunkter har en maximal kapacitet och när paket försvinner beror det på trafikstockning (congestion). Mottagaren eller en switch eller router på vägen har inte hunnit hantera paketet riktigt. Om paket inte blir kvitterat så minskar TCP överföringshastigheten genom att öka tiden innan den sänder nästa paket. På detta sätt undviker TCP att skicka ut en massa paket på ett nätverk som inte hinner med. Om det går bra så minskar den tiden mellan paketen igen. Vi får alltså en kurva som i princip ser ut som nedan.

TCP har funktioner som gör att flera sessioner delar på befintlig bandbredd och TCP undviker att belasta nätverk i onödan.



Schematisk bild av hur TCP utnyttjar en kanal.

När TCP startar en ny session tar det tid innan kanalen utnyttjas till max, detta kallas för slow start och har visat sig vara en bra metod. Flödeshantering och omsändning i TCP är komplext och vi kommer att återkomma till det i ett senare kapitel. Men redan med denna enkla modell ser vi en fördel med TCP. Om vi startar en ny överföring som konkurrerar om samma kanal så kommer TCP att backa och den första sessionen ger plats även för den andra. TCP har alltså en slags automatik som gör att flera sessioner delar på befintlig bandbredd och TCP undviker att belasta nätverk i onödan.

Hur fungerar UDP?

User Datagram Protocol, UDP, innehåller minsta möjliga funktion, i princip lägger vi bara till portnummer för avsändare och mottagare. UDP-headern består av åtta byte.

Avsändande portnr	Destination portnr
Längd	Checksumma

UDP-header.

Längdfältet anger hur långt UDP-paketet är inklusive last (mäts i byte). Vidare finns ett fält för att beräkna checksumma för att verifiera headern och lite mer. Avsändaren har en möjlighet att be-

stämma om checksumman ska användas eller inte. Oftast används funktionen men, eftersom IP inte beräknar checksumma inklusive last så gör funktionen nytta på UDP-nivå, den kontrollerar riktigheten i data. Avsändaren får också sätta fältet till noll vilket betyder att checksumma inte har beräknats. Ett protokoll kan ju utvecklas för att bara fungera lokalt över ett tillförlitligt LAN. Ifall beräkningen av checksumman skulle ge värdet noll, så betecknar man istället detta med värdet oxFF.

Både TCP och UDP använder samma princip för att beräkna checksumma. Checksumman beräknas på:

1. Själva TCP/UDP-headern (där man före beräkningen sätter fältet för checksumman till noll).
2. En pseudo-header vars innehåll främst kommer från IP-nivån. Som innehåll använder man avsändarens och mottagarens IP-adress, protokollfältet samt längdfältet (från TCP respektive UDP).
3. Nyttolasten.

Poängen men en pseudo-header är att vi tillsammans med TCP/UDP-headern kan kontrollera att informationen har nått rätt mottagare avseende både adresser, protokoll och portnummer. Är inte denna information rätt behöver vi inte behandla paketet.

UDP ger alltså i stort sett bara multiplexeringsfunktion så att flera processer/program kan dela på IP-nivån. Inga kvittenser görs, så vi har ingen kontroll på att om paketen kom fram. På samma sätt som i IP får vi ett nät som använder ”best effort”. Kommunikationen är förbindelselös, vi har alltså ingen kontroll på om motparten vill eller kan ta emot paket. Eller om mottagaren ens existerar.

Men i sin enkelhet används UDP flitigt:

- protokollet är enkelt och kräver lite resurser
- ibland har vi inte tid att vänta på omsändning
- vi kan få överföringsfel (lokalt nät)
- vi slipper etablera session, det är effektivt om vi ska göra något enkelt som att bara skicka en fråga och få ett svar
- när vi vill utveckla en egen flödeshantering
- -vid tidsstyrda protokoll där man skickar om data regelbundet, exempelvis vissa routingprotokoll.

Både TCP och UDP använder samma princip för att beräkna checksumma. I beräkningen ingår TCP/UDP-headern, en pseudoheader samt lasten.

Anmärkning:
Pseudo-headern har främst haft historisk betydelse eftersom varken TCP eller IP kunnat lita på att data från nivå två är korrekt. I modern datakommunikation används bättre checksummekontroller som CRC-32 på nivå två. Pseudoheadern påverkas även vid adressöversättning (NAT).

Nivå fyra och framtiden

TCP och UDP har bägge förtjänster men i takt med att IP används även inom avancerade tjänster för telekom så ökar kraven. Ett protokoll som passar bättre för bland annat signalering har utvecklats inom IETF. Protokollet heter SCTP (Stream Control Transmission Protocol).

Även inom ramen för hur TCP fungerar går det att utveckla flödeshanteringen så att den anpassar sig bättre för exempelvis trådlösa förbindelser. Samverkan mellan TCP, UDP och hur IP tappar paket är komplext. Här görs fortfarande arbete och nya tekniker tas fram.

Referenser

RFC 793	Transmission Control Protocol
RFC 768	User Datagram Protocol
RFC 2001	Handlar om TCP Slow Start, fast retransmit med mera
RFC 2018	TCP Selective Acknowledgment Options
RFC 2481	A Proposal to add Explicit Congestion Notification (ECN) to IP