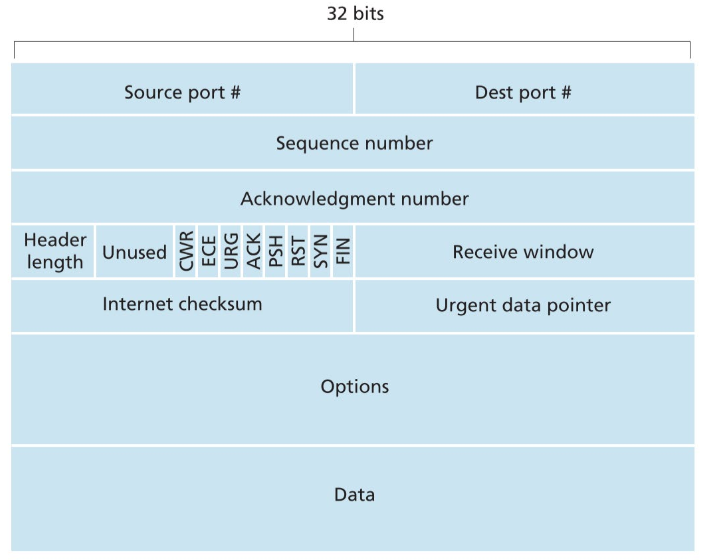
# TCP and socketprogramming

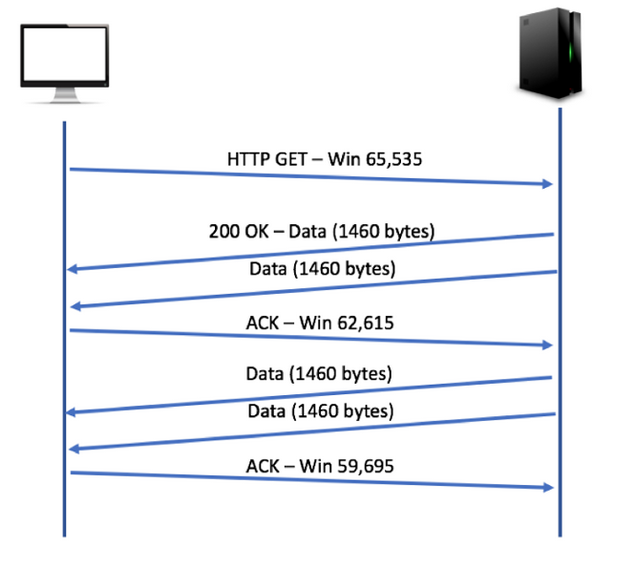
TCP:

|  |  |  |
| --- | --- | --- |
| Point-to-point:   * En sender & en modtager | Reliable, in-order byte stream:   * Man kan sende alt, da det er i den rigtige rækkefølge. Intet bliver tabt. | Pipelined:   * TCP og flow control sætter window size. |
| Full duplex data:   * Data flyder begge veje i samme connection. * Har en max størrelse for segmenterne. (data + header) | Connection-oriented:   * Bruger three way handshake før der bliver sendt data. | Flow controlled:   * Senderen sender ikke for meget data til modtageren. |



Flags:

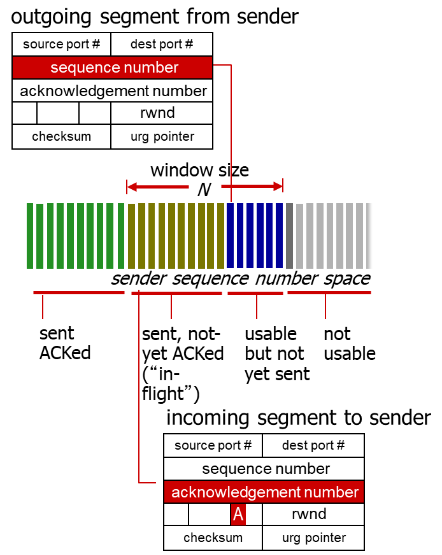
* ECE sendes tilbage til afsender for at sætte farten ned for data.
* CWR sendes tilbage for a acknowledge ECE.
* PSH fortæller at dataen skal sendes til applikationslaget omgående.
* URG fortæller at specifik data i et segment. Der medhører en pointer, så modtageren ved hvor meget data der er vigtigt. Den starter med at tælle fra den første byte. Pointeren lægger i Urgent data pointer.
* RST bruges til at lukke en connection. Hvis modtageren får en pakke fra en sender den ikke regner med, kan den sende RST tilbage for at lukke forbindelsen.
* Header length bruges til at fortælle hvor stor headeren er. Det er nødvendigt da headeren har en skiftende størrelse.
* Receive window bruges til flow control for at fortælle hvor meget plads der er i bufferen. Den bliver mindre og mindre, des mere data der er i den. Hvis den når 0 er bufferen fuld. Når den får tømt bufferen fortæller den igen afsenderen at den kan modtage data.



* Internet checksum bruges bare til at tjekke at dataen er intakt. Ligesom i UDP.
* Options kan f.eks bruges til at sætte maximum størrelse for segmenterne.

**TCP sequence numbers:**

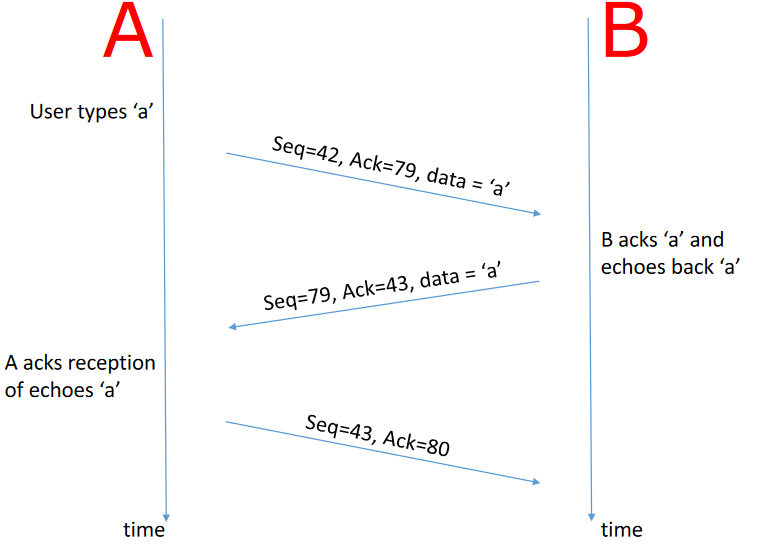
* Sequence number(seq) er nummeret på den første byte i segmentets data. Modtageren ved så hvor de skal starte med at kigge henne.



* Acknowledgement(ack) betyder at alt op til ack – 1 er modtaget korrekt. Den sendes tilbage til afsender.

Det er ikke TCP der sørger for hvordan segmenter ude af orden skal klares. Det er implementation dependent.

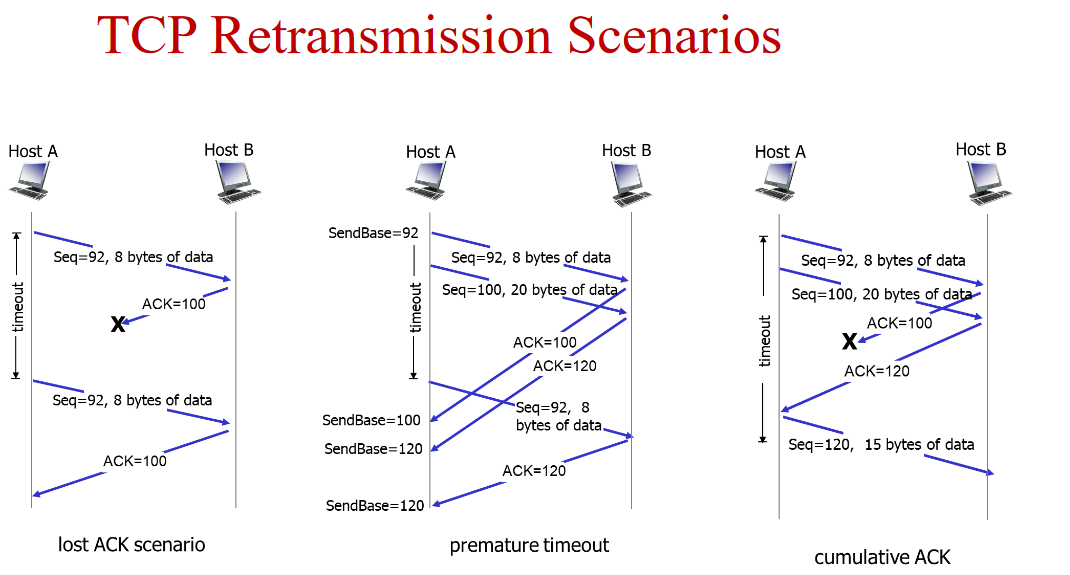
Her sender A data til B og fortæller at segmentet starter på byte 42. B sender så en ACK = 43 for at sige det er modtaget korrekt op til 43 - 1. A kan så verificere dataen og sende tilbage til B at det var rigtigt.



## Reliable data transfer

TCP er reliable da det har:

* Pipelined segmenter.
* Akkumulerede acks.
* Har et retransmission queue, hvor alle segmenter bliver lagt over i når de er sendt afsted. Derefter starter der en timer og hvis den løber ud, sendes segmentet igen. Ellers slettes de når ACK modtages.



|  |  |
| --- | --- |
| Event hos modtager | TCP modtager handling |
| Modtager et segment i rigtig rækkefølge med forventet seq-tal. Al data op til det forventede seq-tal er dog allerede ACK’ed. | Det er et forsinket ACK. Vent op til 500ms for næste segment. Hvis det ikke kommer send ACK. |
| Modtager et segment i rigtig rækkefølge med forventet seq-tal. Et andet segment venter på ACK. | Send med det samme et enkelt akkumuleret ACK der ACK’er begge segmenter. |
| Modtager et segment i forkert rækkefølge med højere end forventet seq-tal. Der er altså et hul i segmenterne. | Send med det samme **duplikeret** ACK der indikerer seq-tal for næste **forventet** byte. |
| Modtager et segmenter der lukker noget eller lukker helt et hul i segmenterne. | Send ACK med det samme, hvis det modtaget segment er i den nederste del af hullet. |

**TCP hurtig Retransmission**

Time-out perioden er oftest relativt lang.

Hvis et segment går tabt, kommer der ofte flere ACKs tilbage, som indikerer at noget data skal sendes igen.

Hvis der modtages 3 ACKs for samme data sendes segmentet med det laveste seq-tal tilbage.