Computer Nerworks. Unit 3: TCP

Notes of the subject *Xarxes de Computadors, Facultat Informàtica de Barcelona, FIB*

Llorenç Cerdà-Alabern

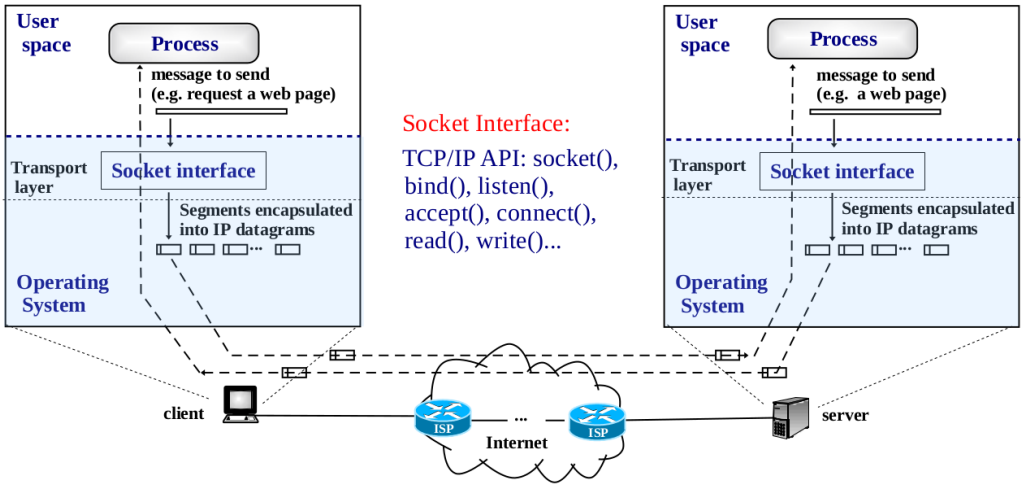
November 7, 2017

Contents

3 Unit 3: TCP

3.1 Transport layer: UDP/TCP

- UDP *User Datagram Protocol*: Connectionless, no reliable.
- TCP *Transmission Control Protocol*: Connection oriented, reliable.

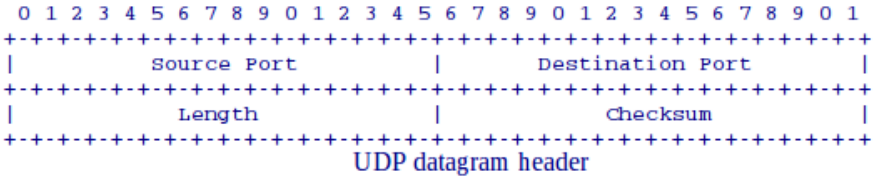


3.2 UPD Protocol RFC768

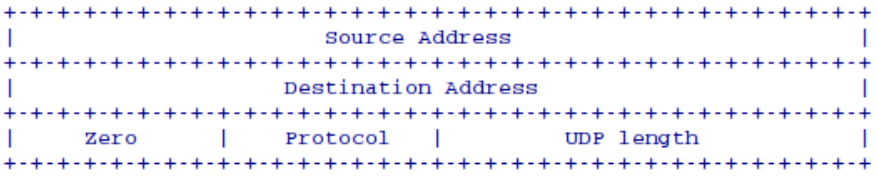
- **Service**: same as IP:
  - Non reliable
  - No error recovery
  - No ack
  - Connectionless
- **Applications** that use UDP
  - short messages e.g. DHCP, DNS, RIP
  - Real time e.g. Voice over IP

3.2.1 UDP Header RFC768

- Fixed size of **8 bytes**
- **checksum**: computed using header, pseudo-header, payload
- Drawback: **NAT-PAT** must update the checksum



UDP datagram header



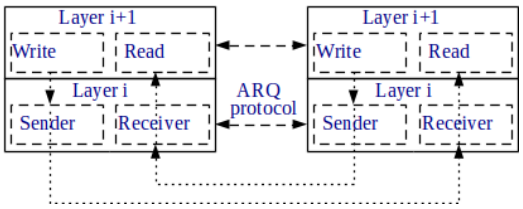
UDP pseudo-header

3.3 Automatic Repeat reQuest (ARQ) RFC3366

3.3.1 What is ARQ?

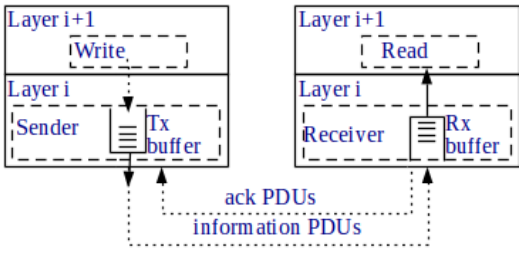
Communication channel between endpoints design for **reliability** and **efficiency**. Typically involves:

- error detection
- error recovery
- flow control



3.3.2 ARQ Ingredients

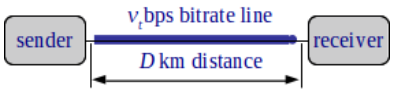
- **Connection oriented**
- Tx/Rx **buffers**
- Acknowledgments (**ack**)
- Acks can be piggybacked
- Retransmission Timeout, **RTO**
- **Sequence Numbers**



ARQ Protocol Implementation (one way)

3.3.3 ARQ evaluation

- evaluate one direction
- there is always information ready to send
- line of distance **D** and bitrate **vt**
- propagation speed of **vp**: propagation **delay = D/vp**
- Information PDUs (**Ik**) / ack PDUs (**Ak**)
- Ik,Ak of **LI**, **LA** bits: Tx times **tt=LI/vt**, **ta=LA/vt**

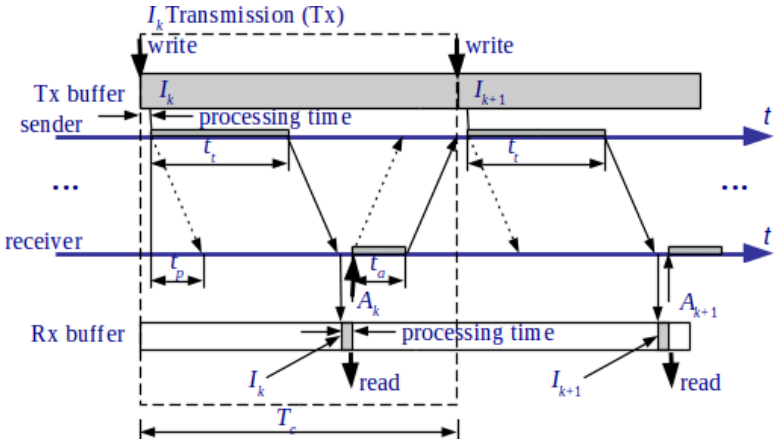


3.3.4 Basic ARQ Protocols:

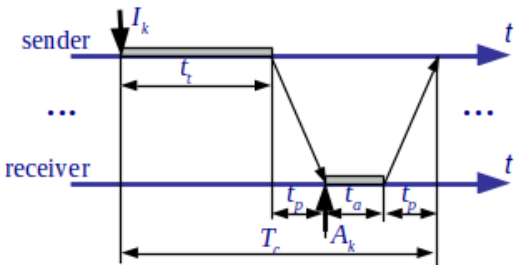
- Stop & Wait
- Go Back N
- Selective Retransmission

3.3.5 Stop & Wait

1. When the **sender** is ready: (i) allows writing from upper layer, (ii) build Ik and pass it down for Tx.
2. When Ik arrives to the **receiver**: (i) pass Ik to upper layer, (ii) generate Ak and pass it down for Tx.
3. When Ak arrives to the **sender**, goto 1.

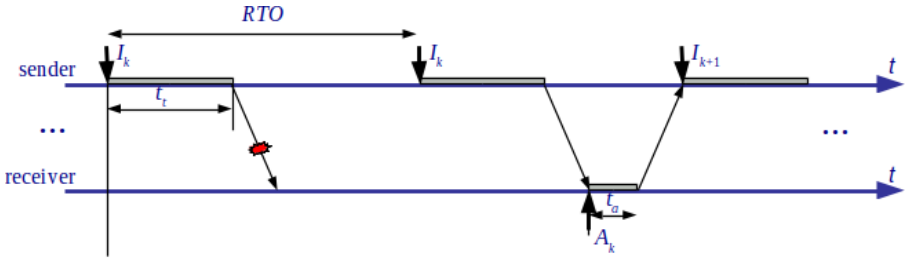


3.3.6 Stop & Wait simplified diagram

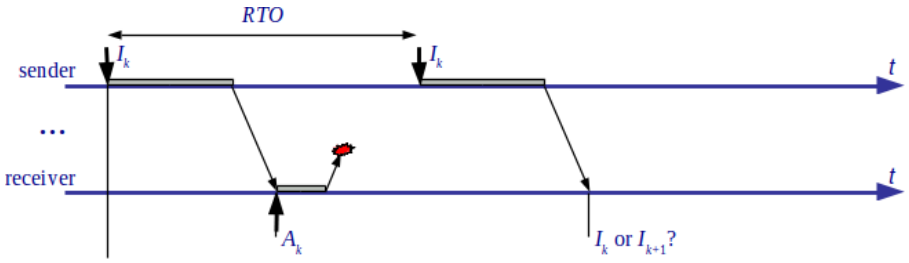


3.3.7 Stop & Wait Retransmission

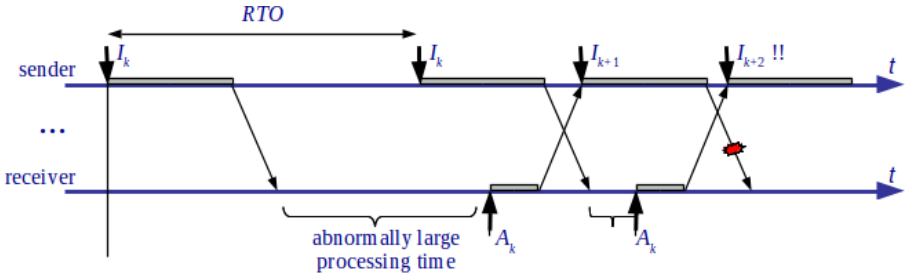
- Retransmission timeout (**RTO**) is started upon each Tx
- If Ik does not arrive, or arrives with errors, **no ack** is sent
- When RTO expires, the sender **ReTx** (retransmits) Ik



3.3.8 Why sequence numbers are needed?



Need to number information PDUs



Need to number ack PDUs

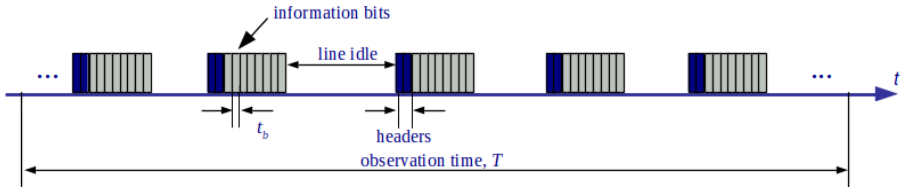
3.3.9 Evaluation

- Given a line with bitrate vt [bps]:
- **Throughput** (velocidad efectiva)

$$v_{ef}[\text{bps}] = \frac{\text{number of information bits}}{\text{observation time}}$$

- **Efficiency** or channel utilization

$$E[\%] = \frac{v_{ef}}{v_t} \times 100$$



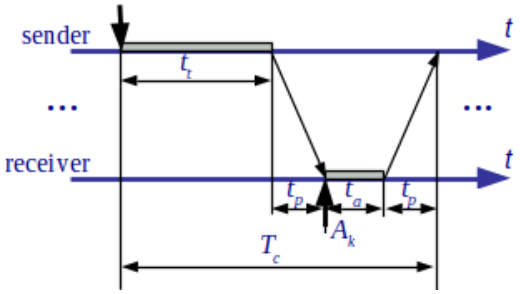
Practical example: throughput with speedtest

```
tcpdump -ni wlan0
```

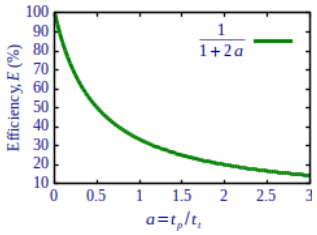
3.3.10 Efficiency in terms of time and bits

$$E = \frac{v_{ef}}{v_t} = \frac{\#\text{info bits}/T}{1/t_b} = \begin{cases} \dfrac{\#\text{info bits} \times t_b}{T} = \dfrac{\text{time Tx information}}{T} \\ \dfrac{\#\text{info bits}}{T/t_b} = \dfrac{\#\text{info bits}}{\#\text{bits at line bitrate}} \end{cases}$$

3.3.11 Stop & Wait efficiency without Tx errors

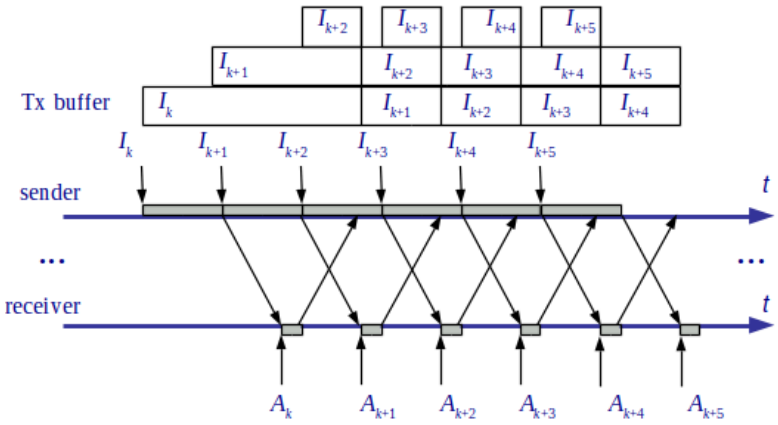


$$E_{protocol} = \frac{t_t}{T_C} = \frac{t_t}{t_t + t_a + 2t_p} = \frac{t_t}{t_t + 2t_p} \simeq \frac{1}{1+2a}, \text{ where } a = \frac{t_p}{t_t}$$



3.3.12 Continuous Tx Protocols

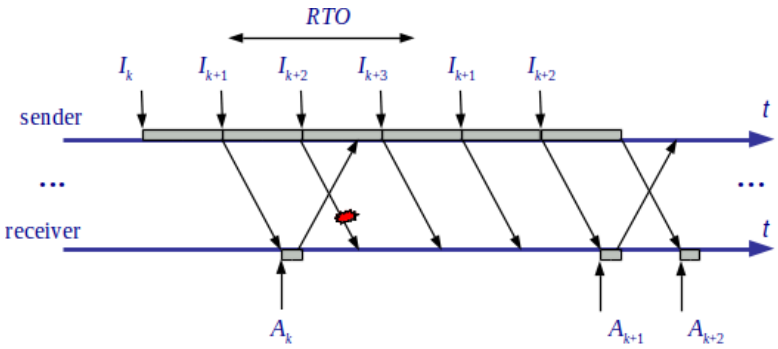
- Without errors: E = 100%



- In case of **errors**:
  - Go Back N
  - Selective ReTx

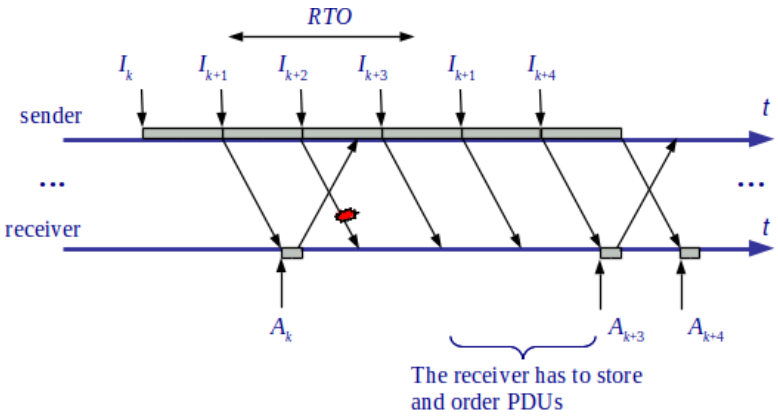
3.3.13 Go Back N

- **Cumulative acks**: Ak confirm Ii, i <= k
- If error or out of order PDU: **Do not send acks**, discards all PDU until the expected PDU arrives. The receiver does not store out of order PDUs.
- Upon **RTO**: go back and starts Tx from that PDU.



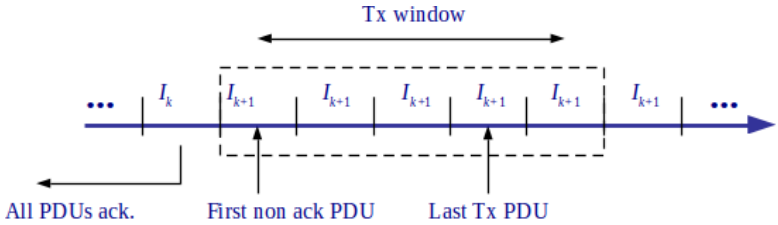
3.3.14 Selective ReTx

- Same as Go Back N, but:
  - The sender only ReTx a PDU when a **RTO** occurs.
  - The **receiver** stores out of order PDUs, and ack all stored PDUs when missing PDUs arrive.



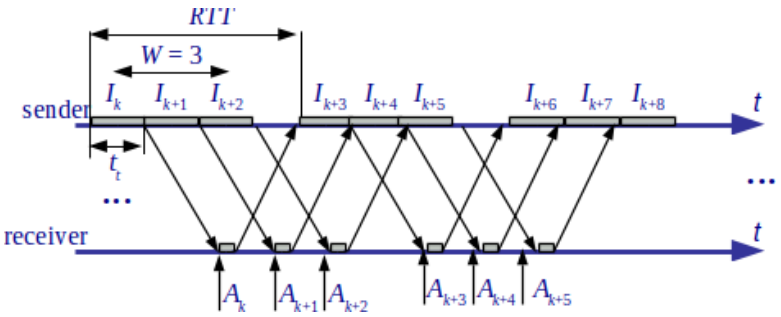
3.3.15 Flow Control and Window Protocols

- **Flow control**: adapt Tx to Rx rate.
- **Stop & Wait**: automatic Flow control.
- **Continuous Tx** protocols: Use a **Tx window**
- **Tx window** maximum number of **non-ack PDUs that can be Tx**. If the Tx window is exhausted, the sender stales.
- Stop & Wait is a window protocol with Tx window = 1 PDU.
- Tx window allows dimension the Tx and Rx buffers.



3.3.16 Optimal Tx window

Optimal window: Minimum window that allows the maximum throughput.



$$W_{opt}[\text{PDU}] = \left\lceil \frac{\text{RTT}}{t_t} \right\rceil$$

In bytes (**bandwidth delay product**):

$$W_{opt}[\text{B}] = v_{ef}[\text{bps}] \times \text{RTT}[\text{s}]/8[\text{bits/B}]$$

Example: for $v_{ef} = 4$ Mbps and RTT $= 200$ ms we need

$$W_{opt} = 4 \times 10^6 \text{ bps} \times 200 \times 10^{-3} \text{ s}/8[\text{bits/B}] = 100 \text{ kB}$$

3.4 TCP Protocol RFC793

3.4.1 TCP Service

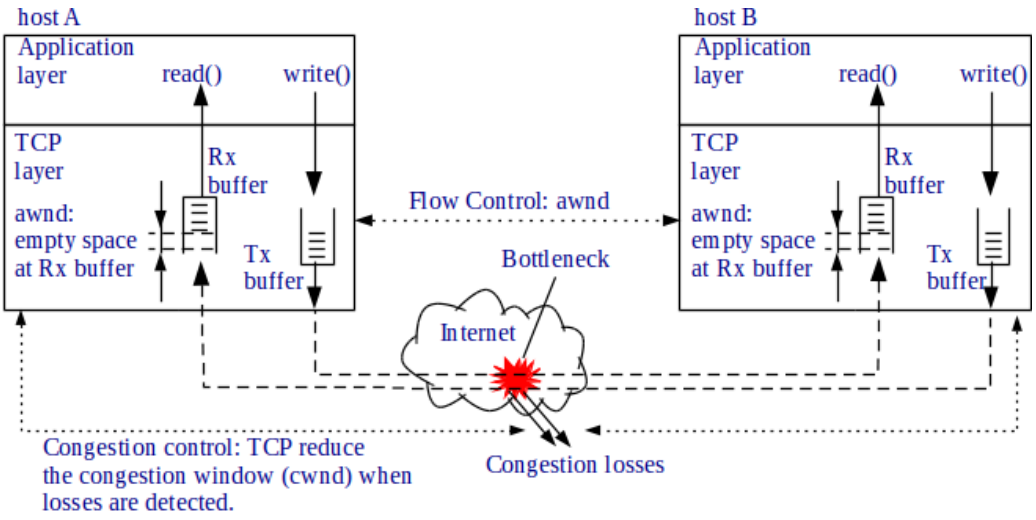
- **Service**:
  - Reliable service (ARQ):
    - Connection oriented
    - Error recovery
    - Congestion control: Adapt throughput to network
    - Flow control: Adapt throughput to receiver
- **Usage**
  - Applications requiring reliability: Web, ftp, ssh, telnet, mail, ...

3.4.2 TCP Basis

- Segments of optimal size: Maximum Segment Size (**MSS**)
  - MSS adjusted using **MTU path discovery**
- **ARQ** window protocol, with **variable window**
- Each time a segment arrives, TCP sends an **ack**

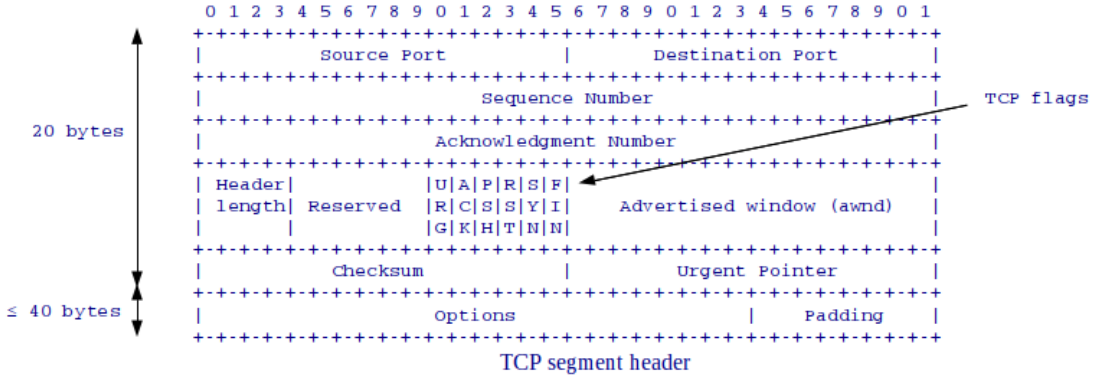
3.4.3 TCP window

- **wnd = min(awnd, cwnd)**
  - **awnd**, advertised window: used for **flow control**
  - **cwnd**, congestion window: used for **congestion control**



3.4.4 TCP header

- Fixed **20** bytes + **options** 15x4 = **60** bytes max
- Like UDP, the checksum is computed using header + **pseudo-header** + payload

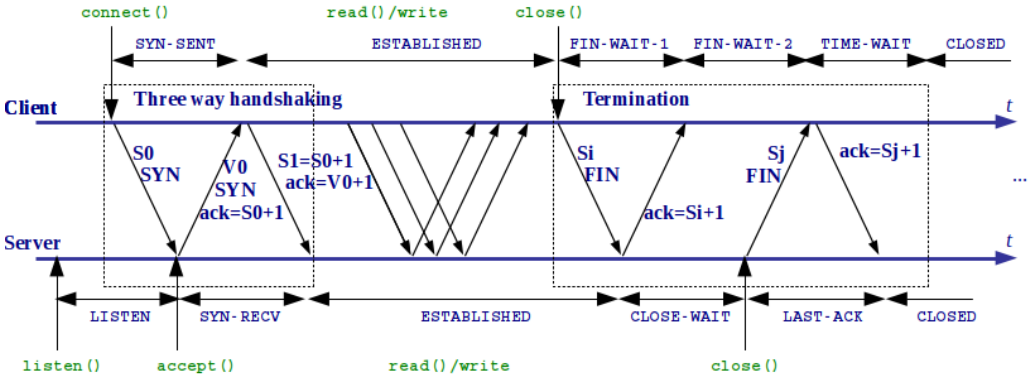


3.4.5 TCP Flags

- **URG** (Urgent): Urgent Pointer points to the first urgent byte. Example: ^C in a telnet session.
- **ACK**: Always set except for the first segment.
- **PSH** (Push): "push" all data to the receiving buffer.
- **RST** (Reset): Abort the connection.
- **SYN**: Used in the connection setup (three-way-handshaking, TWH).
- **FIN**: Used in the connection termination.

3.4.6 Connection Setup and Termination

- The client always send the 1st segment
- Three-way handshaking segments have payload = 0
- SYN and FIN segments consume 1 sequence number
- Initial sequence number is random



Practical example

capture a TCP connection with tcpdump and observe the connection setup and termination (bash)

```
tcpdump -ni lo
```

Minimal TCP server (perl)

```
#!/usr/bin/perl -w
use IO::Socket::INET; use Term::ANSIColor;

print "Sart TCP server.\n" ;
my $s_sock = IO::Socket::INET->new(
    LocalHost => '127.0.0.1',
    LocalPort => 5000,
    Proto     => 'tcp',
    Listen    => 5
) or die "Could not create socket!\n";

while(1) {
  my $c_sock = $s_sock->accept() ;
  printf colored("Accepted: ", 'green')."%s, %s\n",
    $c_sock->peerhost(), $c_sock->peerport() ;
  while(<$c_sock>) {
    print "Received from Client : $_";
  }
  printf colored("Closed: ", 'red'),"%s, %s\n",
    $c_sock->peerhost(), $c_sock->peerport() ;
}
```

Minimal TCP client (perl)

```
#!/usr/bin/perl -w
use IO::Socket::INET;

print "Sart TCP client.\n" ;
my $socket = IO::Socket::INET->new(
    PeerHost => '127.0.0.1',
    PeerPort  => 5000,
    Proto     => 'tcp'
) or die "Could not create socket: $!\n";

print "TCP Connected.\n" ;
while (<>) {
  print "sending $_" ;
  $socket->send($_);
}
```

3.4.7 TCP Options

- **Maximum Segment Size** (MSS): Used in the TWH: **MTU-40** (IPv4+TCP headers without options).

- **Window Scale factor**: Used in the TWH: **awnd** is multiplied by $2^{\text{WindowScale}}$ (number of bits to left-shift awnd). Allows using awnd larger than $\mathbf{2^{16}}$ bytes.
- **Timestamp**: Used to compute the Round Trip Time (**RTT**). Is a **10 bytes** option. Clock of the TCP sender + echo of the timestamp of the segment being ack.
- **SACK**: In case of errors, blocks of consecutive correctly received segments for Selective ReTx.

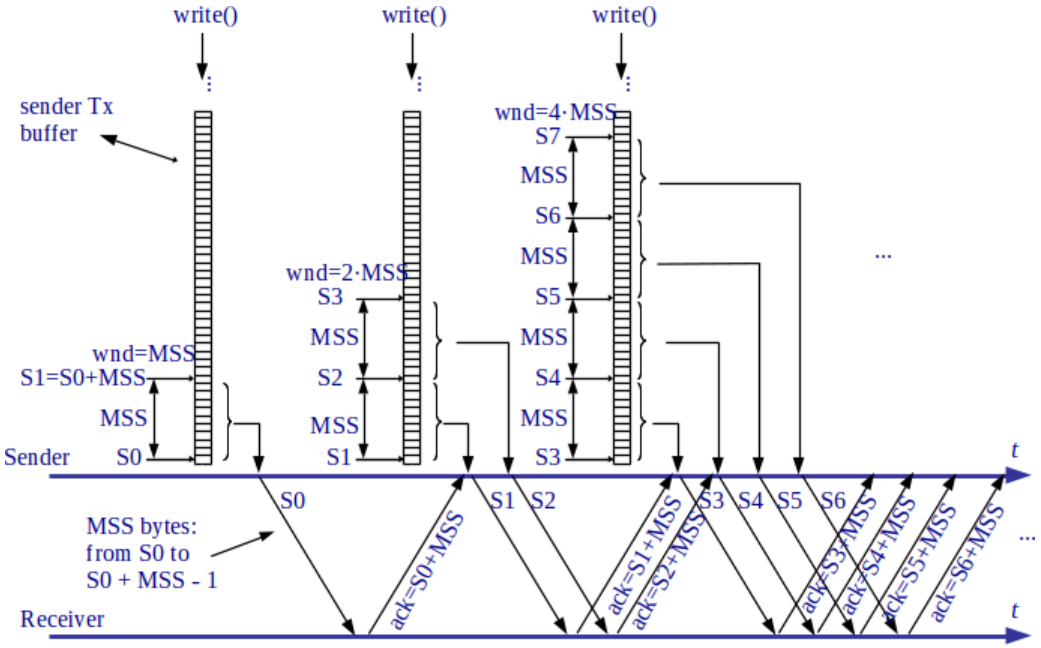
Practical example

capture a TCP connection with tcpdump and observe the TCP options (bash)

```
tcpdump -ni lo
```

3.4.8 TCP Sequence Numbers

- **Sequence number**: points the first payload byte.
- **Ack number**: points the next missing byte

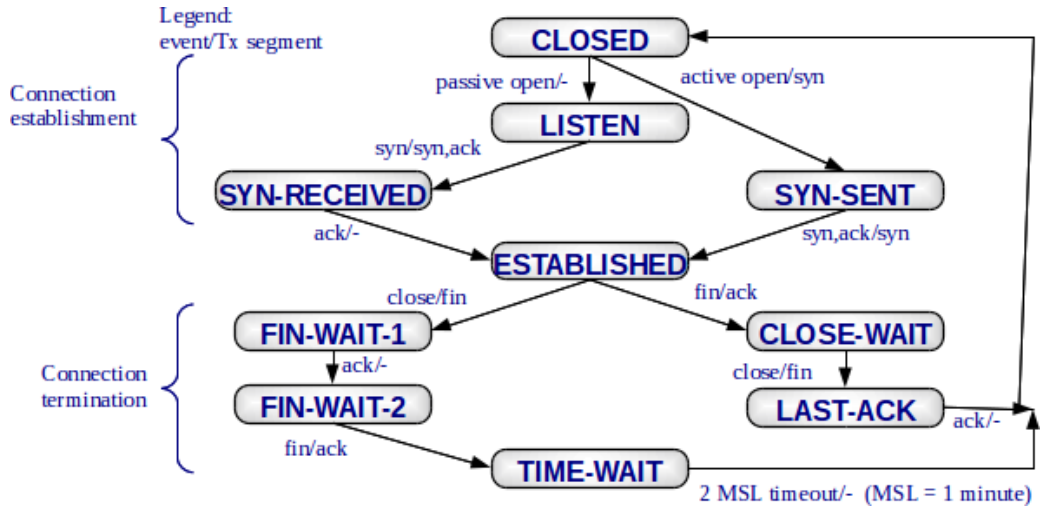


Practical example

Capture a TCP connection with tcpdump and observe the sequence numbers (bash)

```
tcpdump -ni lo
```

3.4.9 TCP State diagram



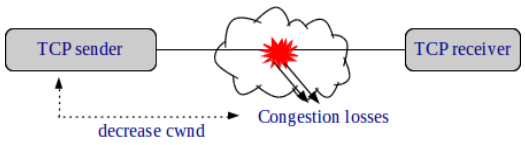
Practical example

capture a TCP connection with tcpdump and observe the connection states (bash)

```
tcpdump -ni lo
netstat -nat
```

3.4.10 TCP Congestion Control RFC2581

- **wnd = min(awnd, cwnd)**
  - **awnd**, advertised window: used for **flow control**
  - **cwnd**, congestion window: used for **congestion control**
- TCP interprets losses as congestion:
  - Basic Congestion Control Algorithm:
  - **Slow Start / Congestion Avoidance** (SS/CA)



3.4.11 Slow Start / Congestion Avoidance (SS/CA)

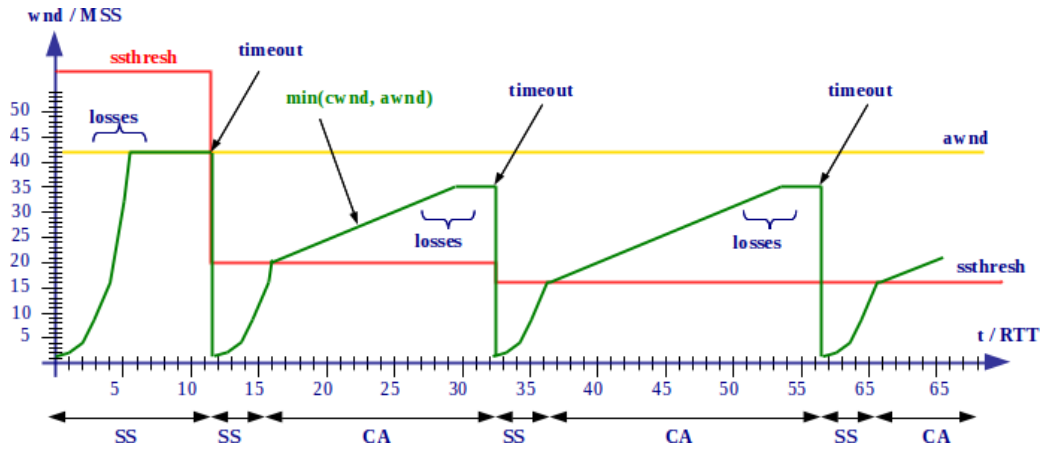
- **ssthresh**: Threshold between SS and CA.

Slow Start / Congestion Avoidance (SS/CA) (c)

```
Initialization:
 cwnd = MSS ; (NOTE: RFC 2581 allows an initial window of 2 segments)
 ssthresh = infinity ;
// Each time an ack confirming new data is received:
 if(cwnd < ssthresh) { /* Slow Start */
   cwnd += MSS ;               // add 1 segment
 } else {              /* Congestion Avoidance */
   cwnd += MSS * MSS / cwnd ; // add 1/cwnd segments
  }
// When RTO expires:
 Retransmit ;
 ssthresh = max(min(awnd, cwnd)/2, 2*MSS) ;
 cwnd = MSS ;
```

- During SS cwnd is rapidly increased to the "operational point"

- During CA cwnd is slowly increased looking for more "bandwidth"



3.4.12 Practical example

enabling chargen (bash)

```
$ cat /etc/xinetd.d/chargen
service chargen
{
        disable         = no
        type            = INTERNAL
        id              = chargen-stream
        socket_type     = stream
        protocol        = tcp
        user            = root
        wait            = no
}
```

add a queue of size 10kB and rate 100kbps to eth0 (bash)

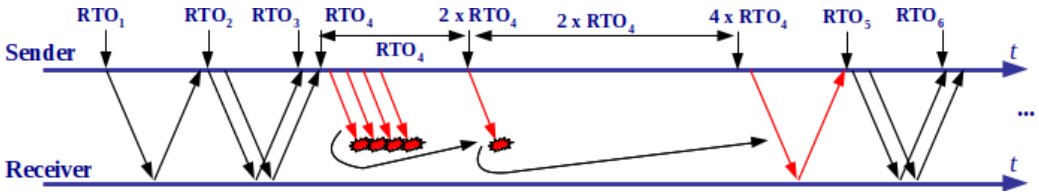
```
sudo tc qdisc add dev eth0 root tbf burst 5000 rate 100kbit limit 10000
sudo tc qdisc show
```

capture the traffic generated by the chargen server (bash)

```
sudo tcpdump -ni wlan0
telnet localhost chargen
netstat -nat
```

3.4.13 Retransmission time-out (RTO)

- Activation:
  - Active whenever there are pending acks
  - Continuously decreased, **ReTx** occurs when RTO reaches zero
- Each time an ack **confirming new data** arrives:
  - RTO is computed
  - RTO is restarted if pending acks
- Computation:
  - TCP sender measures RTT mean (**srtt**) and variance (**rttvar**)
  - **RTO = srtt + 4 * rttvar**
  - RTO is duplicated each retransmitted segment
- **RTT** measurements:
  - Using "slow-timer tics" (coarse)
  - Using the TCP **timestamp** option


RTO1
RTO2
RTO3
RTO4
2 x RTO4
2 x RTO4
4 x RTO4
RTO5
RTO6
RTO4
Sender
Receiver
t
t
...