Computer Networks - *Xarxes de Computadors*

Outline

- Course Syllabus
- Unit 1: Introduction
- Unit 2. IP Networks
- **Unit 3. Point to Point Protocols -TCP**
- Unit 4. LANs
- Unit 5. Data Transmission

Unit 3. Point to Point Protocols -TCP

Outline

- **Introduction**
- Basic ARQ Protocols
- UDP Protocol
- TCP Protocol

Unit 3. Point to Point Protocols -TCP

Introduction

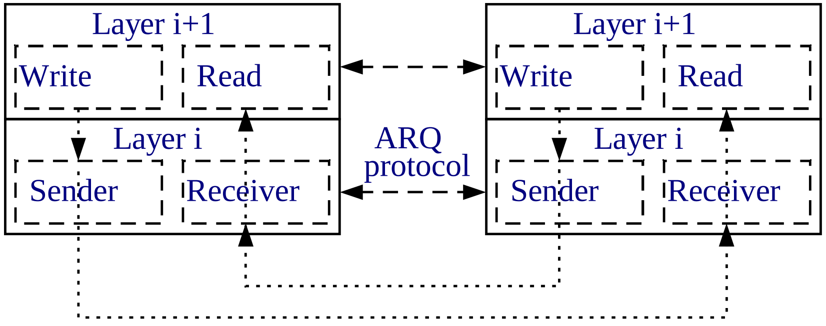
- A Point to Point Protocol (PPP) takes place between exactly two endpoints.
- PPP is usually used to identify protocols that builds up a communication channel between endpoints, adding features of the type:
  - Error detection
  - Error recovery
  - Flow control
- These are typical data-link layer features, although protocols from other layers can be also regarded as PPPs:
  - Physical: RS-232
  - Data-link: The PPP protocol used in TCP/IP
  - Network: X.25
  - Transport: TCP
- Automatic Repeat reQuest (ARQ) protocols are typically used for PPP.

Unit 3. Point to Point Protocols -TCP

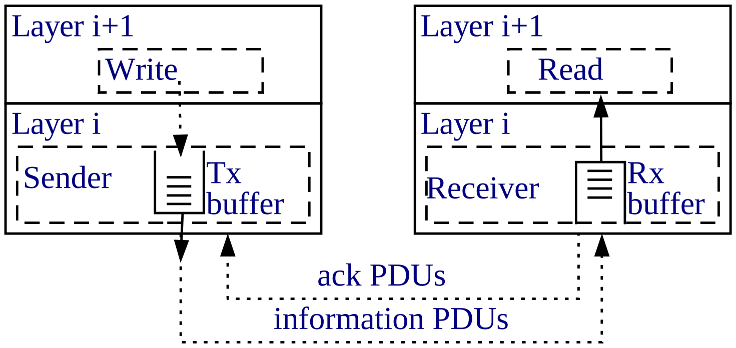
Introduction - Automatic Repeat reQuest, ARQ

ARQ Ingredients

- Connection oriented
- Tx/Rx buffers
- Acknowledgments (ack)
- Acks can be *piggybacked* in information PDUs sent in the opposite direction.
- Retransmission Timeout.
- Sequence Numbers



ARQ Protocol Architecture



ARQ Protocol Implementation (one way)

Unit 3. Point to Point Protocols -TCP

Introduction - Automatic Repeat reQuest, ARQ

Basic ARQ Protocols:

- Stop & Wait
- Go Back N
- Selective Retransmission

Unit 3. Point to Point Protocols -TCP

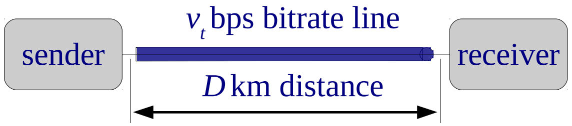
Outline

- Introduction
- **Basic ARQ Protocols**
- UDP Protocol
- TCP Protocol

Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols - Assumptions

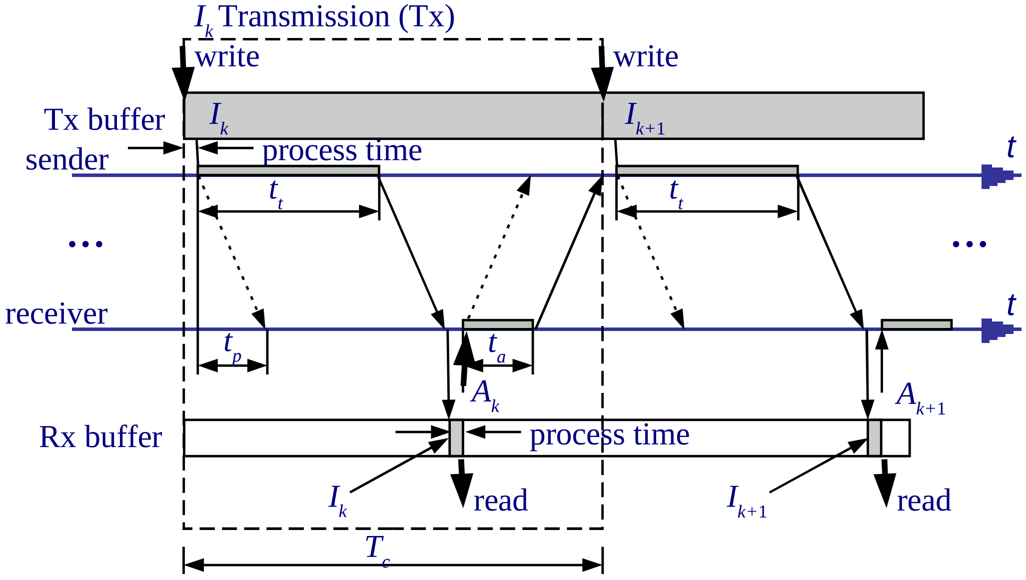
- We shall focus on the the transmission in one direction.
- We shall assume a saturated source: There is always information ready to send.
- We shall assume full duplex links.
- ppp protocol over a line of D m distance and v_t bps bitrate.
- Propagation speed of v_p m/s, thus, propagation delay of D/v_p s.
- We shall refer to a generic layer, where the sender sends Information PDUs (I_k) and the receiver sends ack PDUs (A_k).
- Frames carrying I_k respectively A_k, are Tx using L_I and L_A bits, thus the Tx times are respectively: $t_t = L_I/v_t$ and $t_a = L_A/v_t$ s.



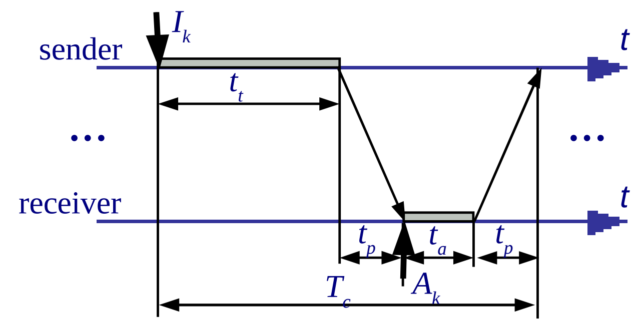
Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols - Stop & Wait

1. When the sender is ready: (i) allows writing from upper layer, (ii) builds I_k, (iii) I_k goes down to data-link layer and Tx starts.
2. When I_k completely arrives to the receiver: (i) it is read by the upper layer, (ii) A_k is generated, A_k goes down to data-link layer and Tx starts.
3. When A_k completely arrives to the sender, goto 1.



Time diagram

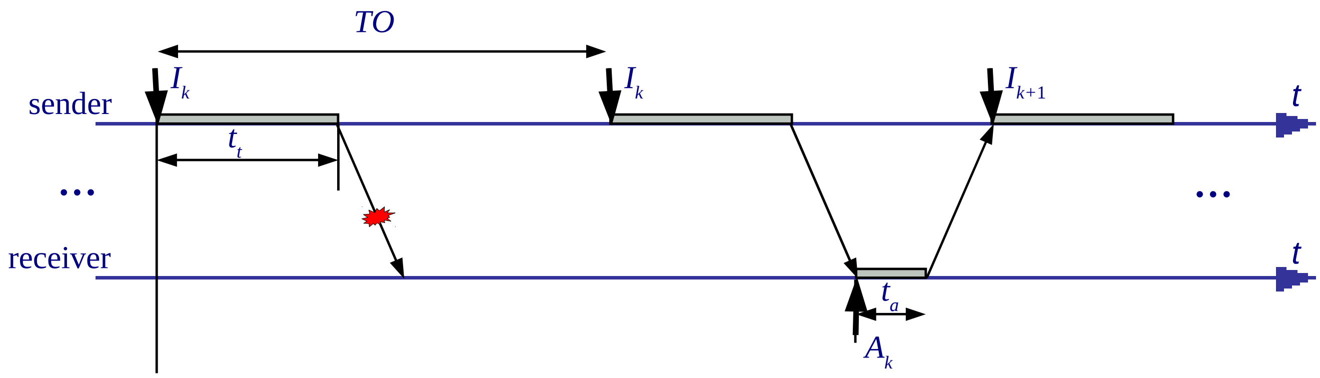


Simplified time diagram

Unit 3. Point to Point Protocols -TCP

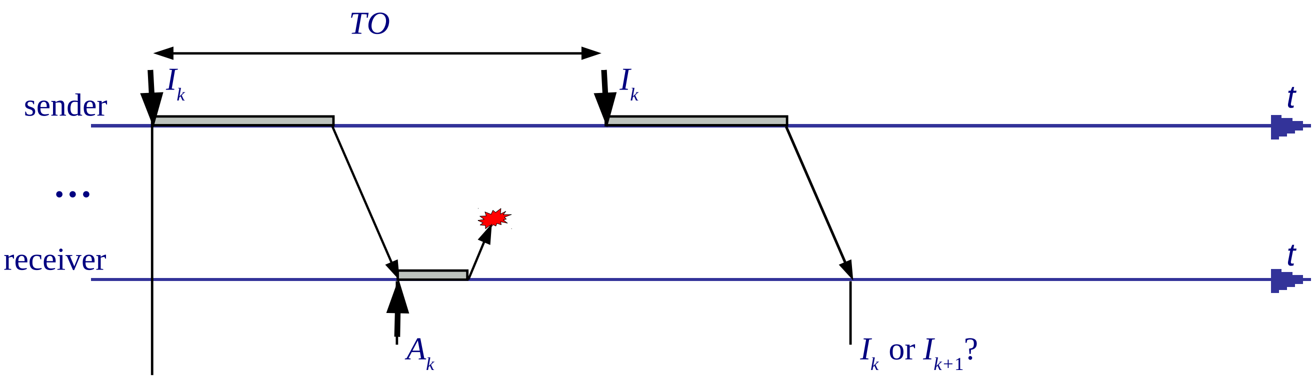
Basic ARQ Protocols - Stop & Wait Retransmission

- Each time the sender Tx a PDU, a retransmission timeout (TO) is started.
- If the information PDU do not arrives, or arrives with errors, no ack is sent.
- When TO expires, the sender ReTx the PDU.

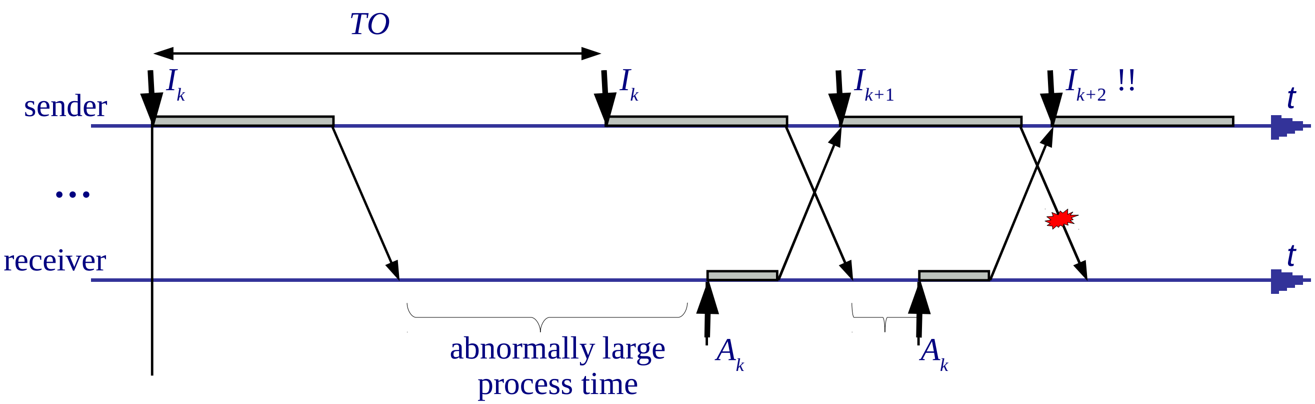


Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Why sequence numbers are needed?



Need to number information PDUs

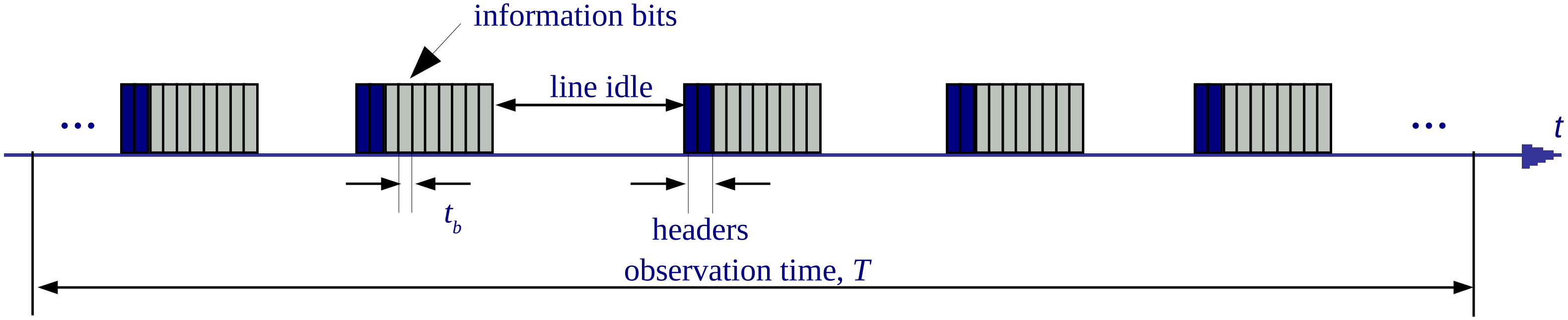


Need to number ack PDUs

Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Notes on computing the efficiency (channel utilization)

- Line bitrate: $v_t = 1/t_b$, bps
- Throughput (*velocidad efectiva*) v_{ef} = number of inf. bits / obs. time, bps
- Efficiency or channel utilization $E = v_{ef} / v_t$ (times 100, in percentage)

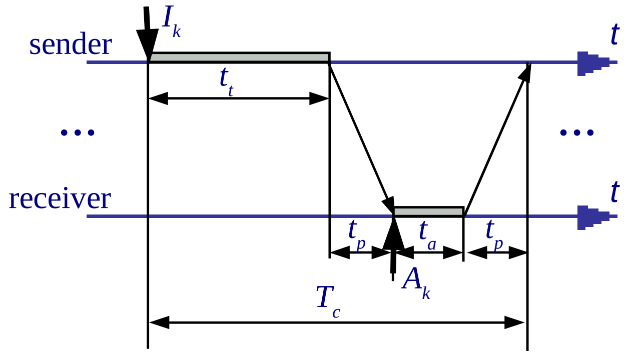


$$E = \frac{v_{ef}}{v_t} = \frac{\#\text{info bits}/T}{1/t_b} = \begin{cases} \dfrac{\#\text{info bits} \times t_b}{T} = \dfrac{\text{time Tx information}}{T} \\[2ex] \dfrac{\#\text{info bits}}{T/t_b} = \dfrac{\#\text{info bits}}{\#\text{bits at line bitrate}} \end{cases}$$

Unit 3. Point to Point Protocols -TCP

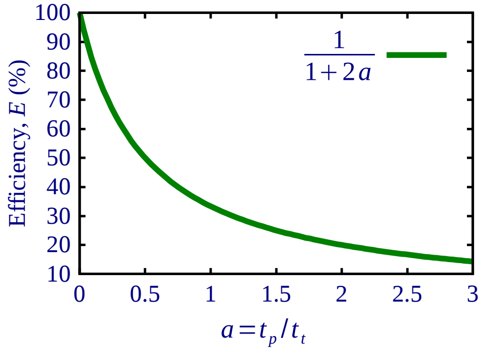
Basic ARQ Protocols – Stop & Wait efficiency

- Assuming no errors (maximum efficiency), the Tx is periodic, with period T_c.
- $E_{protocol}$: We do not take into account headers.



$$E_{protocol} = \frac{t_t}{T_C} = \frac{t_t}{t_t + t_a + 2t_p} =$$

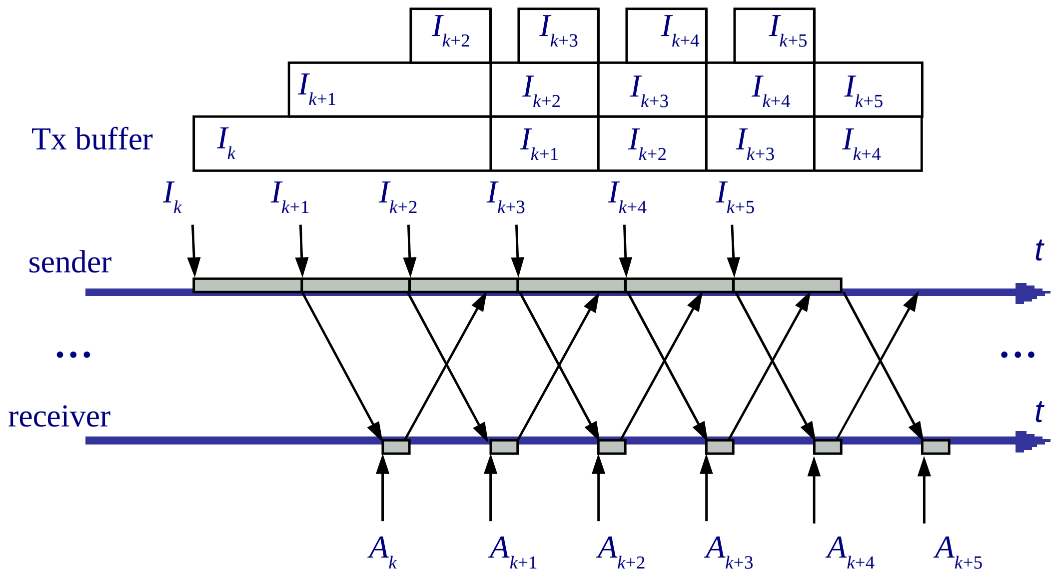
$$\frac{t_t}{t_t + 2t_p} \simeq \frac{1}{1+2a}, \text{ where } a = \frac{t_p}{t_t}$$



Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Continuous Tx Protocols

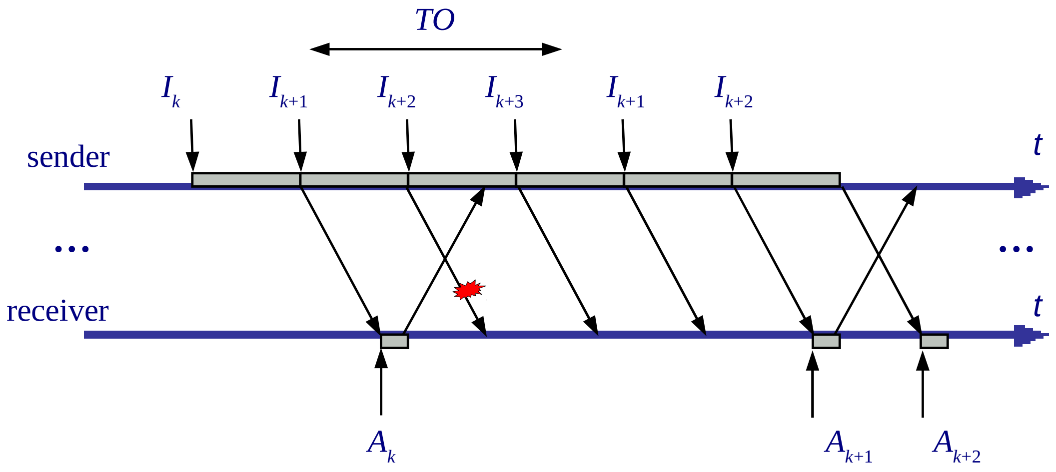
- Goal: Allow high efficiency independently of propagation delay.
- Without errors: E = 100 %



Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Go Back N

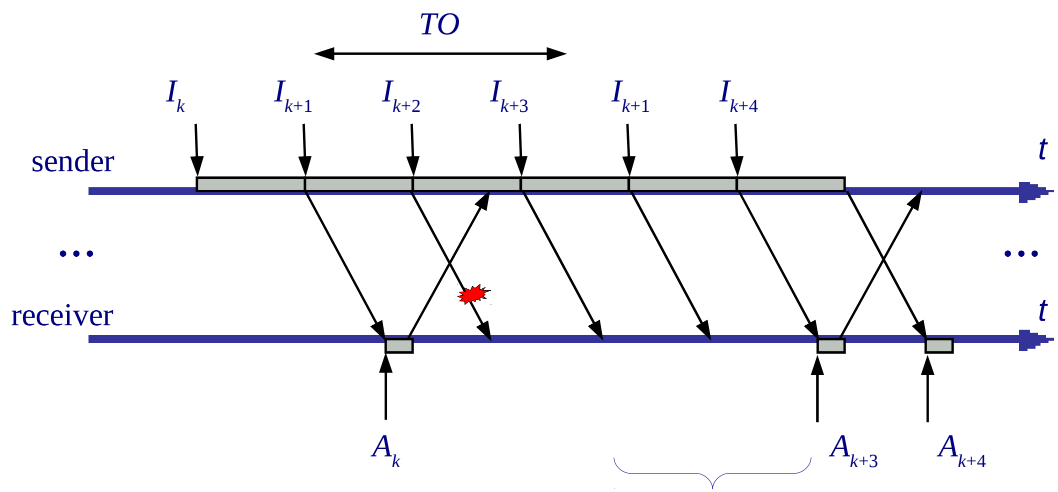
- Cumulative acks: A_k confirm I_i, $i \leq k$
- If the sender receives an error of out of order PDU: Do not send acks, discards all PDU until the expected PDU arrives. Thus, the receiver does not store out of order PDUs.
- When a TO occurs, the sender *goes back* and starts Tx from that PDU.



Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Selective ReTx.

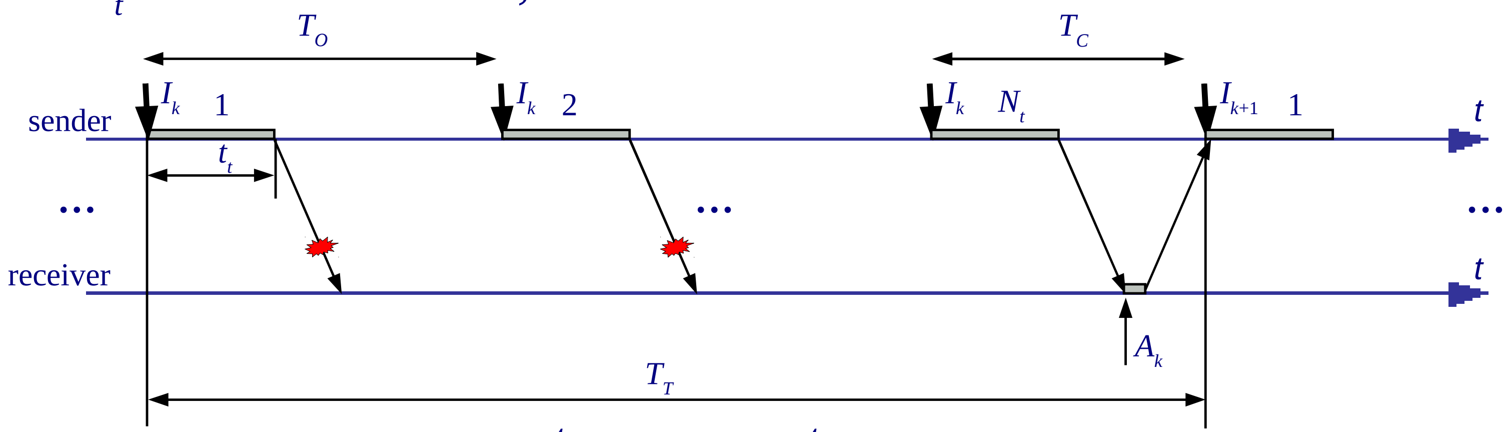
- The same as Go Back N, but:
  - The sender only ReTx a PDU when a TO occurs.
  - The receiver stores out of order PDUs, and ack all stored PDUs when missing PDUs arrive.



Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Efficiency with Tx errors: Stop & Wait

- Assumptions: On average, N_t Tx are needed to successfully send a PDU: $N_t - 1$ with Tx errors, and 1 correct.



$$E_{protocol} = \frac{t_t}{T_T} = \frac{t_t}{(N_t - 1)T_O + T_C}$$

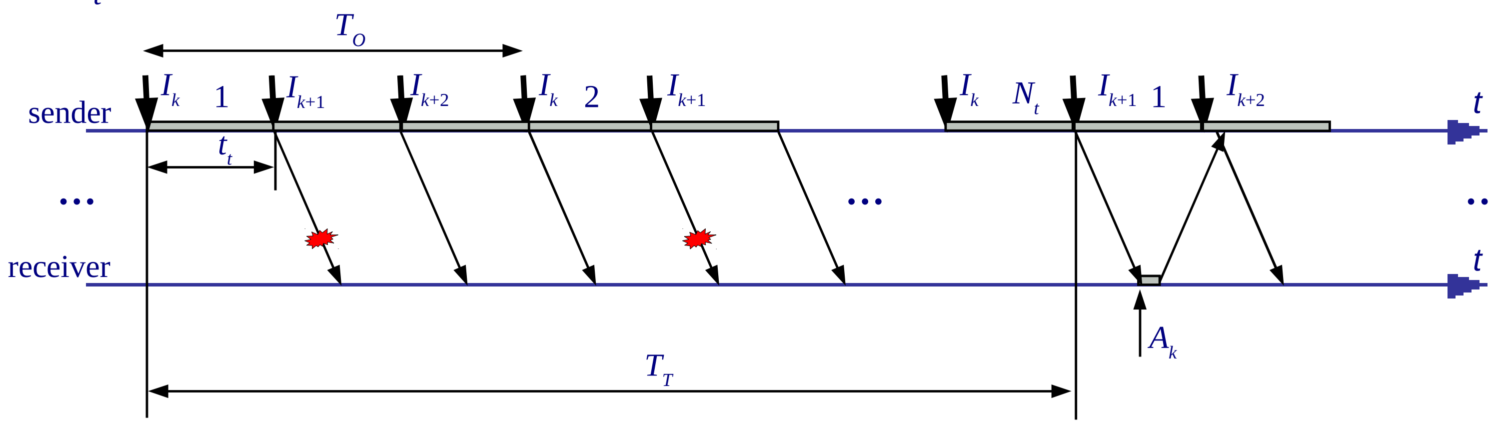
- To avoid unnecessary ReTx $T_O > T_C$. The maximum throughput is when $T_O \approx T_C$:

$$E_{protocol} \simeq \frac{t_t}{N_t T_C} = \frac{1}{N_t(1+2a)}$$

Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Efficiency with Tx errors: Go Back N

- Assumptions: On average, N_t Tx are needed to successfully send a PDU: $N_t - 1$ with Tx errors, and 1 correct.



$$E_{protocol} = \frac{t_t}{T_T} = \frac{t_t}{(N_t - 1) T_O + t_t}$$

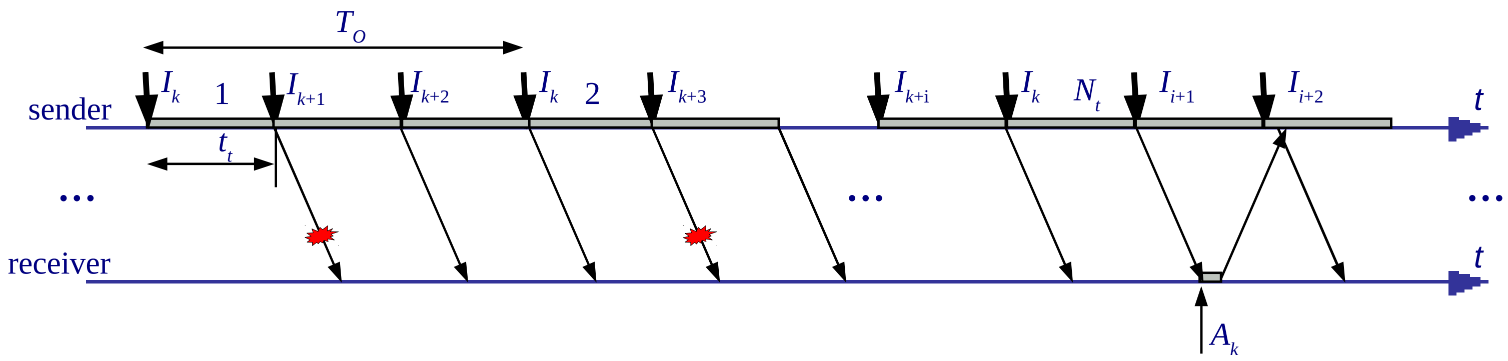
- The maximum throughput is when $T_O \approx T_C$:

$$E_{protocol} \simeq \frac{t_t}{(N_t - 1) T_C + t_t} = \frac{1}{(N_t - 1)(1 + 2a) + 1} = \frac{1}{N_t (1 + 2a) - 2a}$$

Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Efficiency with Tx errors: Select. ReTx

- Assumptions: On average, N_t Tx are needed to successfully send a PDU: $N_t - 1$ with Tx errors, and 1 correct.

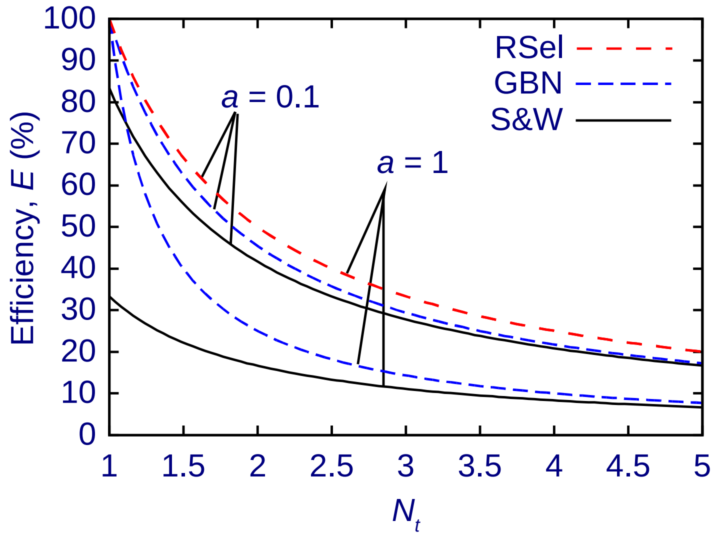


$$E_{protocol} = \frac{t_t}{N_t t_t} = \frac{1}{N_t}$$

Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Efficiency with Tx errors: Comparison

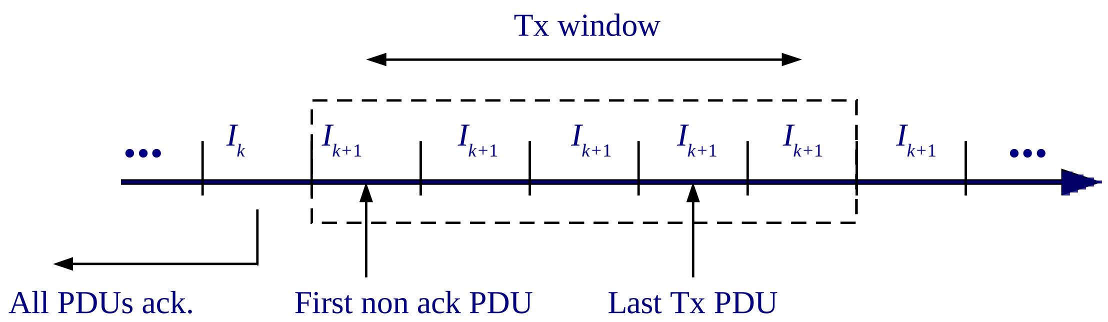
- For $a << 1$, all protocols are similar ($a = t_p/t_t$)
- For $N_t >> 1$ all protocols have low E.
- E in selective ReTx does not depend on a.
- If a << 1 does not hold:
  - E of Stop & Wait is low.
  - For moderate N_t, Sel-ReTx outperforms GoBackN.



Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Flow Control and Window Protocols

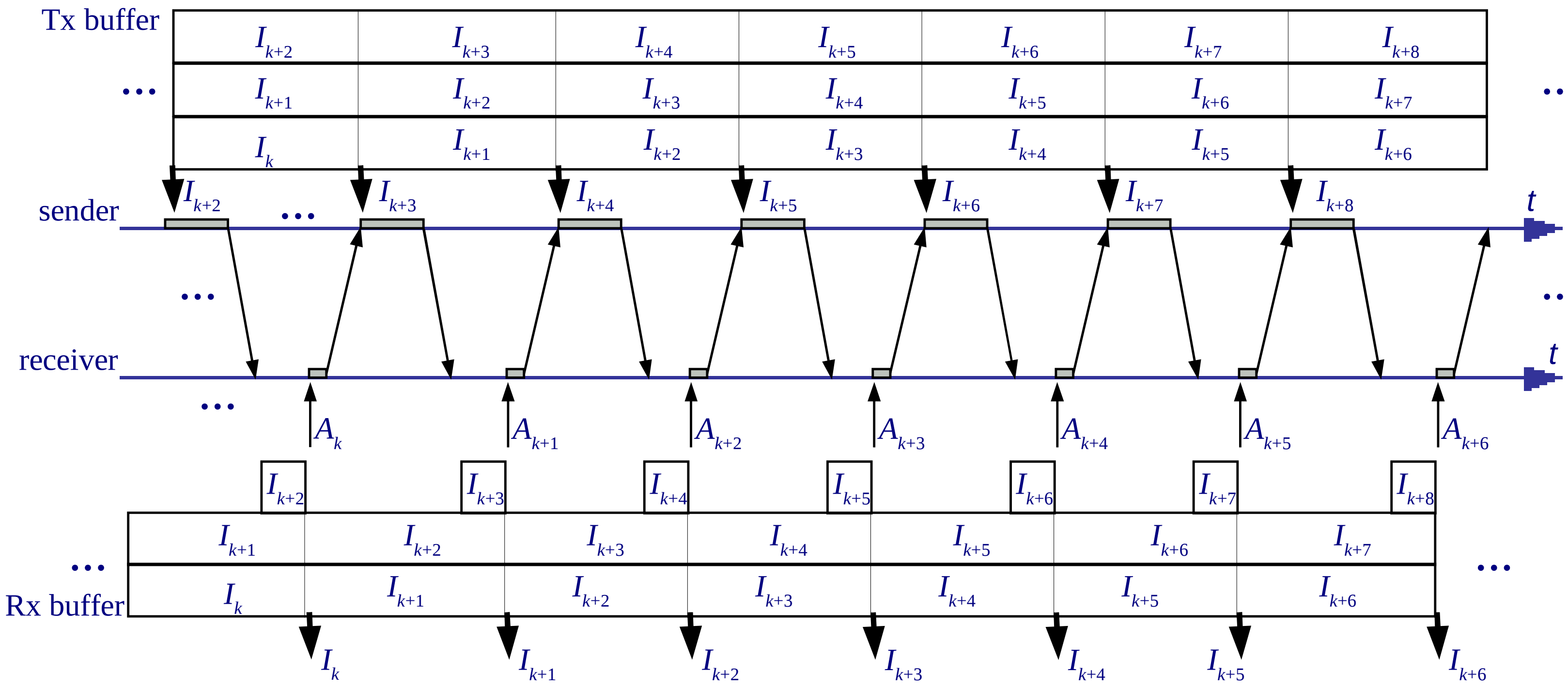
- ARQ are also used for flox control. Flox control consists on avoiding the sender to Tx at higher PDU rate than can be consumed by the receiver.
- With Stop & Wait, if the receiver is slower, acks are delayed and the sender reduces the throughput.
- With continuous Tx protocols: A *Tx window* is used. The window is the 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.
- Furthermore, the Tx window allows dimensioning the Tx buffer, and the Rx buffer for Selective ReTx: No more the Tx window PDUs need to be stored.



Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Window Protocol Flux Control Example

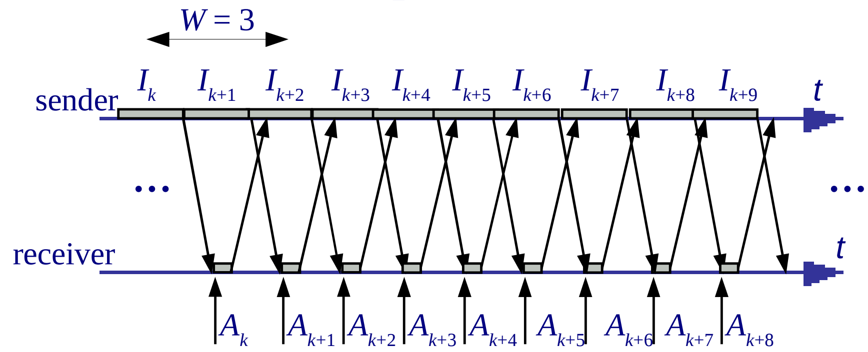
- $W = 3$
- Stationary regime



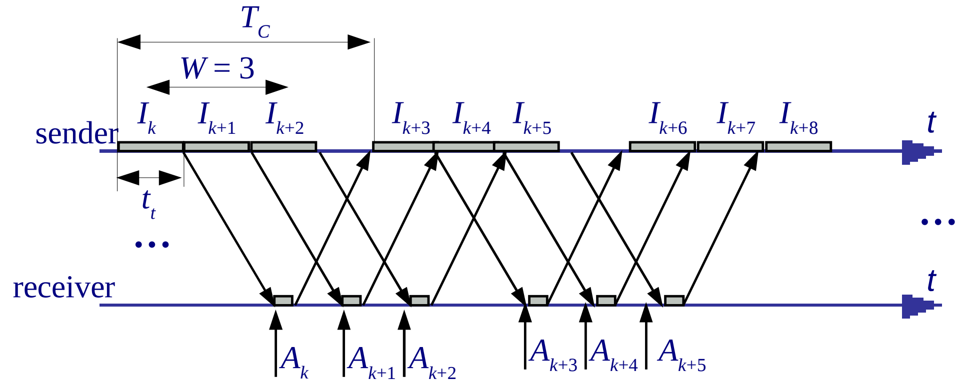
Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Optimal Tx window

- Optimal window: Minimum window that allows the maximum throughput.
- Optimal window example:



- Non optimal window example:



- Clearly, for this example:

$$W_{opt} = \left\lceil \frac{T_C}{t_t} \right\rceil$$

Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Sequence Number Field Dimensioning

- Information and ack PDUs need sequence numbers.
- Using n bits, the sequence number used for PDU k is: $k \bmod 2^n$.
- What value we need for n to avoid ambiguity?
- Without proof: If PDUs arrive in the same order they are Tx, with arbitrary delays, we need to distinguish the PDU that can be in the Tx and Rx buffers of the sender and receiver respectively:

Protocol	#Seq. Numbers (≥)	#Bits (≥)
Stop & Wait	2	1
GoBackN	W+1	$\log_2(W+1)$
Selective ReTx	2 W	$\log_2(2\,W)$

Unit 3. Point to Point Protocols -TCP

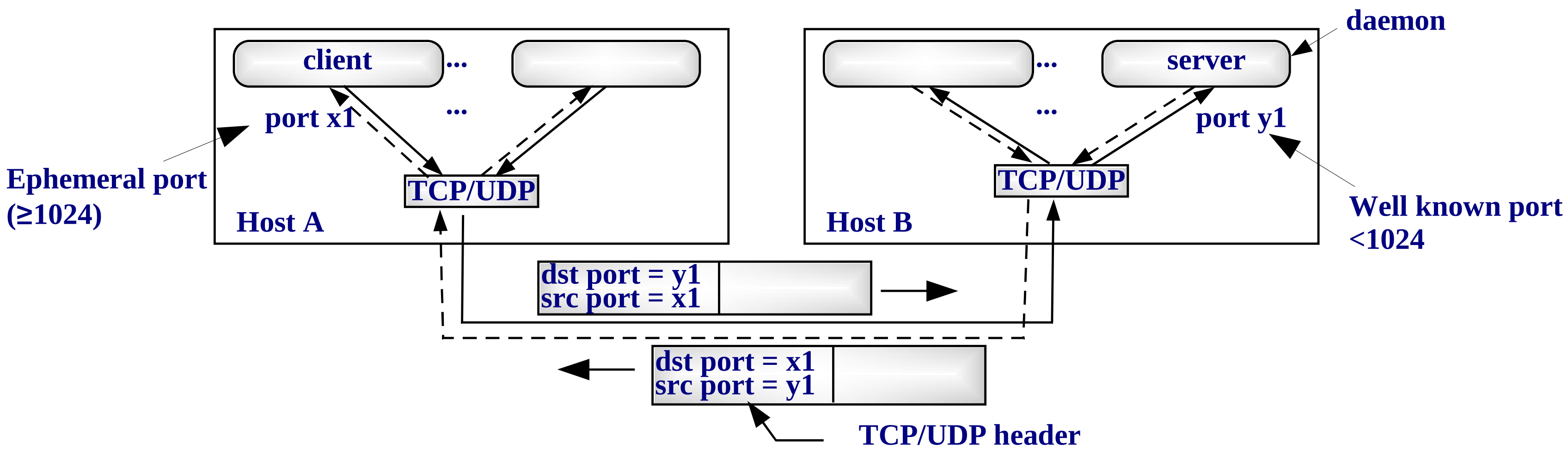
Outline

- Introduction
- Basic ARQ Protocols
- **UDP Protocol**
- TCP Protocol

Unit 3. Point to Point Protocols -TCP

UPD Protocol – Introduction: The Internet Transport Layer

- Two protocols are used at the TCP/IP transport layer: User Datagram Protocol (UDP) and Transmission Control Protocol (TCP).
- UDP offers a *datagram service* (non reliable).
- TCP offers a reliable service.
- Transport layer offers a communication channel between applications.
- Transport layer access points (applications) are identified by a 16 bits port numbers.
- TCP/UDP use the client/server paradigm



Unit 3. Point to Point Protocols -TCP

UPD Protocol – Description (RFC 768)

- UDP Datagram service: same as IP.
  - Non reliable
  - No error recovery
  - No ack
  - Connectionless
  - No flux control
- UDP PDU is referred to as UDP datagram.
- UDP does not have a Tx buffer: each application write operation generates a UDP datagram.
- UDP is typically used:
  - Applications where short messages are exchanged: E.g. DHCP, DNS, RIP.
  - Real time applications: E.g. Voice over IP, videoconferencing, stream audio/video. These applications does not tolerate large delay variations (which would occur using an ARQ).

Unit 3. Point to Point Protocols -TCP

UPD Protocol – C Code example

```
main(int argc, char *argv[])
{
    int sock ;
    struct sockaddr_in clnt_addr, serv_addr ;
    struct hostent *host ;
    int serv_len = sizeof(serv_addr) ;

    if(argc != 2) {
        fprintf(stderr, "usage: %s hostname\n", argv[0]) ;
        exit(1) ;
    }
    host = gethostbyname(argv[1]) ; /* call the resolver for server addr. */
    if(host == NULL) {
        perror("gethostbyname ") ;
        exit(2) ;
    }
    serv_addr.sin_family = AF_INET ;
    memcpy(&serv_addr.sin_addr, host->h_addr, host->h_length) ;
    serv_addr.sin_port = htons(3333) ;

    sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP) ; /* create a socket */
    clnt_addr.sin_family = AF_INET ;
    clnt_addr.sin_addr.s_addr = htonl(INADDR_ANY) ; /* the OS choose any valid address */
    clnt_addr.sin_port = htons(0) ; /* the OS choose an ephemeral port */
    bind(sock, (struct sockaddr *)&clnt_addr, sizeof(clnt_addr)) ;  /* give local addr. to a socket */

    char msg[] = "hello world\n" ;
    sendto(sock, msg, strlen(msg), 0,
           (struct sockaddr )&serv_addr, sizeof(serv_addr)) ; /* send a UDP datagram */
    close(sock) ; /* close the socket */
}
```

Unit 3. Point to Point Protocols -TCP

UPD Protocol – UDP Header

- Fixed size of 8 bytes.
- The checksum is computed using (i) the header, (ii) a pseudo-header, (iii) the payload.
- Because of the pseudo-header, the UDP checksum needs to be updated if PAT is used.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Length             |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

UDP datagram header

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Source Address                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Destination Address                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Zero      |    Protocol   |          UDP length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

UDP pseudo-header

Unit 3. Point to Point Protocols -TCP

Outline

- Introduction
- Basic ARQ Protocols
- UDP Protocol
- **TCP Protocol**

Unit 3. Point to Point Protocols -TCP

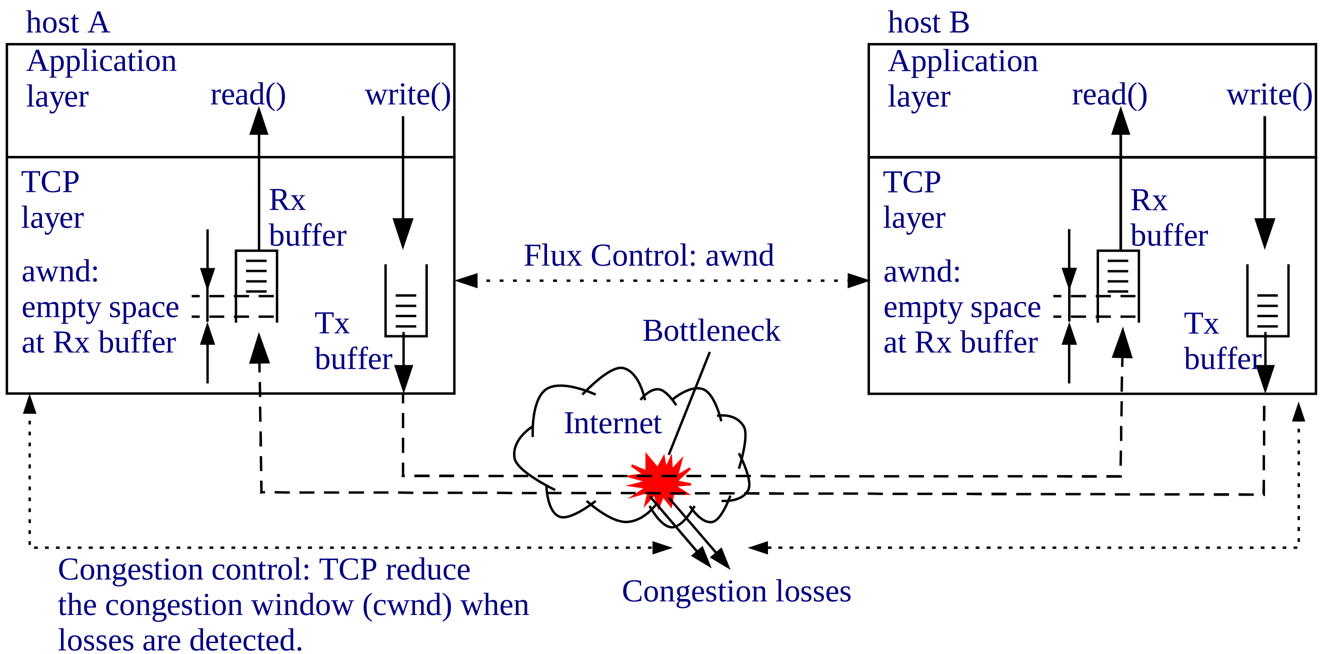
TCP Protocol – Description (RFC 793)

- Reliable service (ARQ).
  - Error recovery
  - Acknowledgments
  - Connection oriented
  - Flux control
- TCP PDU is referred to as TCP segment.
- Congestion control: Adapt the TCP throughput to network conditions.
- Segments of optimal size: Variable Maximum Segment Size (MSS).
- TCP is typically used:
  - Applications requiring reliability: Web, ftp, ssh, telnet, mail, ...

Unit 3. Point to Point Protocols -TCP

TCP Protocol – Basic operation

- ARQ window protocol, with variable window: **wnd = min(awnd, cwnd)**
- Each time a segment arrives, TCP send an ack (unless delayed ack is used) without waiting for the upper layer to read the data.
- The advertised window (**awnd**) is used for flux control.
- The congestion window (**cwnd**) is used for congestion control.



Unit 3. Point to Point Protocols -TCP

TCP Protocol – Delayed acks and Nagle algorithm

- TCP connections can be classified as:
  - Bulk: (e.g. web, ftp) There are always bytes to send. TCP send MSS bytes.
  - Interactive: (eg. telnet, ssh) The user interacts with the remote host.
- In interactive connections small packets are sent: Each keyboard hit may generate a segment, and one ack is sent for each.
- Solutions: Delayed acks, Nagle algorithm.

Unit 3. Point to Point Protocols -TCP

TCP Protocol – Delayed acks and Nagle algorithm

- Delayed ack. It is used to reduce the amount of acks. Consists of sending 1 ack each 2 MSS segments, or 200 ms. Acks are always sent in case of receiving out of order segments.

- tcpdump example:

bulk transfer

```
...
11:27:13.798849 147.83.32.14.ftp > 147.83.35.18.3020: P 9641:11089(1448) ack 1 win 10136 (DF)
11:27:13.800174 147.83.32.14.ftp > 147.83.35.18.3020: P 11089:12537(1448) ack 1 win 10136 (DF)
11:27:13.800191 147.83.35.18.3020 > 147.83.32.14.ftp: . 1:1(0) ack 12537 win 31856 (DF)
11:27:13.801405 147.83.32.14.ftp > 147.83.35.18.3020: P 12537:13985(1448) ack 1 win 10136 (DF)
11:27:13.802771 147.83.32.14.ftp > 147.83.35.18.3020: P 13985:15433(1448) ack 1 win 10136 (DF)
11:27:13.802788 147.83.35.18.3020 > 147.83.32.14.ftp: . 1:1(0) ack 15433 win 31856 (DF)
...
```

TCP flags

DF flag in IP header set.

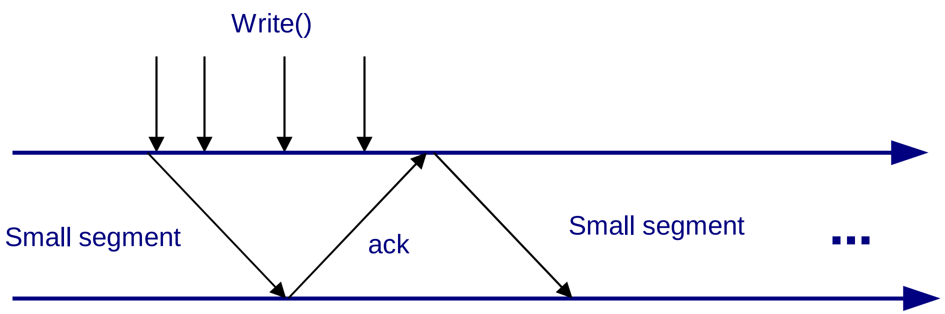
timestamp | src IP addr/port | dst IP addr/port | seq. num:next seq num (bytes) | ack | awnd

```
11:27:13.798849 147.83.32.14.ftp > 147.83.35.18.3020: P 9641:11089(1448) ack 1 win 10136 (DF)
```

Unit 3. Point to Point Protocols -TCP

TCP Protocol – Delayed acks and Nagle algorithm

- Nagle algorithm. It is used to reduce the nunber of small information segments in iterative connections. Consists of TCP sending segments only when:
  - A full MSS segment can be sent.
  - There are no bytes pending to be ack. So, keyboard hits are stored until an ack arrives.



Unit 3. Point to Point Protocols -TCP

TCP Protocol – TCP Header

- Variable size: Fixed fields of 20 bytes + options (15x4 = 60 bytes max.).
- Like UDP, the checksum is computed using (i) the header, (ii) a pseudo-header, (iii) the payload, and needs to be updated if PAT is used.

```
             0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
            +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
            |          Source Port          |       Destination Port        |
            +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
            |                        Sequence Number                        |   TCP flags
            +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  20 bytes  |                    Acknowledgment Number                      |
            +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
            | Header|           |U|A|P|R|S|F|                               |
            | length| Reserved  |R|C|S|S|Y|I|    Advertised window (awnd)   |
            |       |           |G|K|H|T|N|N|                               |
            +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
            |           Checksum            |         Urgent Pointer        |
            +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 ≤ 40 bytes |                    Options                    |    Padding    |
            +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TCP segment header

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Zero      |    Protocol   |          TCP length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TCP pseudo-header

Unit 3. Point to Point Protocols -TCP

TCP Protocol – TCP Flags

- URG (Urgent): The Urgent Pointer is used. It points to the first urgent byte. Rarely used. Example: ^C in a telnet session.
- ACK: The ack field is used. Always set except for the first segment sent by the client.
- PSH (Push): The sender indicates to “push” all buffered data to the receiving application. Most BSD derived TCPs set the PSH flag when the send buffer is emptied.
- RST (Reset): Abort the connection.
- SYN: Used in the connection setup (*three-way-handshaking*, *TWH*).
- FIN: Used in the connection termination.

Unit 3. Point to Point Protocols -TCP

TCP Protocol – TCP Flags

- tcpdump example:

```
TCP flags
  S: SYN
  P: PUSH
  .: No flag (except ack) is set
```

```
09:33:02.556785 IP 147.83.34.125.24374 > 147.83.194.21.80: S 3624662632:3624662632(0) win 5840
                <mss 1460,sackOK,timestamp 531419155 0,nop,wscale 7>
09:33:02.558054 IP 147.83.194.21.80 > 147.83.34.125.24374: S 2204366975:2204366975(0) ack
                3624662633 win 5792 <mss 1460,sackOK,timestamp 3872304344 531419155,nop,wscale 2>
09:33:02.558081 IP 147.83.34.125.24374 > 147.83.194.21.80: . ack 1 win 46 <nop,nop,timestamp
                531419156 3872304344>
09:33:02.558437 IP 147.83.34.125.24374 > 147.83.194.21.80: P 1:627(626) ack 1 win 46
                <nop,nop,timestamp 531419156 3872304344>
09:33:02.559146 IP 147.83.194.21.80 > 147.83.34.125.24374: . ack 627 win 1761 <nop,nop,timestamp
                3872304345 531419156>
09:33:02.559507 IP 147.83.194.21.80 > 147.83.34.125.24374: P 1:271(270) ack 627 win 1761
                <nop,nop,timestamp 3872304345 531419156>
09:33:02.559519 IP 147.83.34.125.24374 > 147.83.194.21.80: . ack 271 win 54 <nop,nop,timestamp
                531419156 3872304345>
09:33:02.560154 IP 147.83.194.21.80 > 147.83.34.125.24374: . 271:1719(1448) ack 627 win 1761
                <nop,nop,timestamp 3872304345 531419156>
09:33:02.560167 IP 147.83.34.125.24374 > 147.83.194.21.80: . ack 1719 win 77 <nop,nop,timestamp
                531419156 3872304345>
09:33:02.560256 IP 147.83.194.21.80 > 147.83.34.125.24374: . 1719:3167(1448) ack 627 win 1761
                <nop,nop,timestamp 3872304345 531419156>
09:33:02.560261 IP 147.83.34.125.24374 > 147.83.194.21.80: . ack 3167 win 100 <nop,nop,timestamp
                531419156 3872304345>
...
```

Unit 3. Point to Point Protocols -TCP

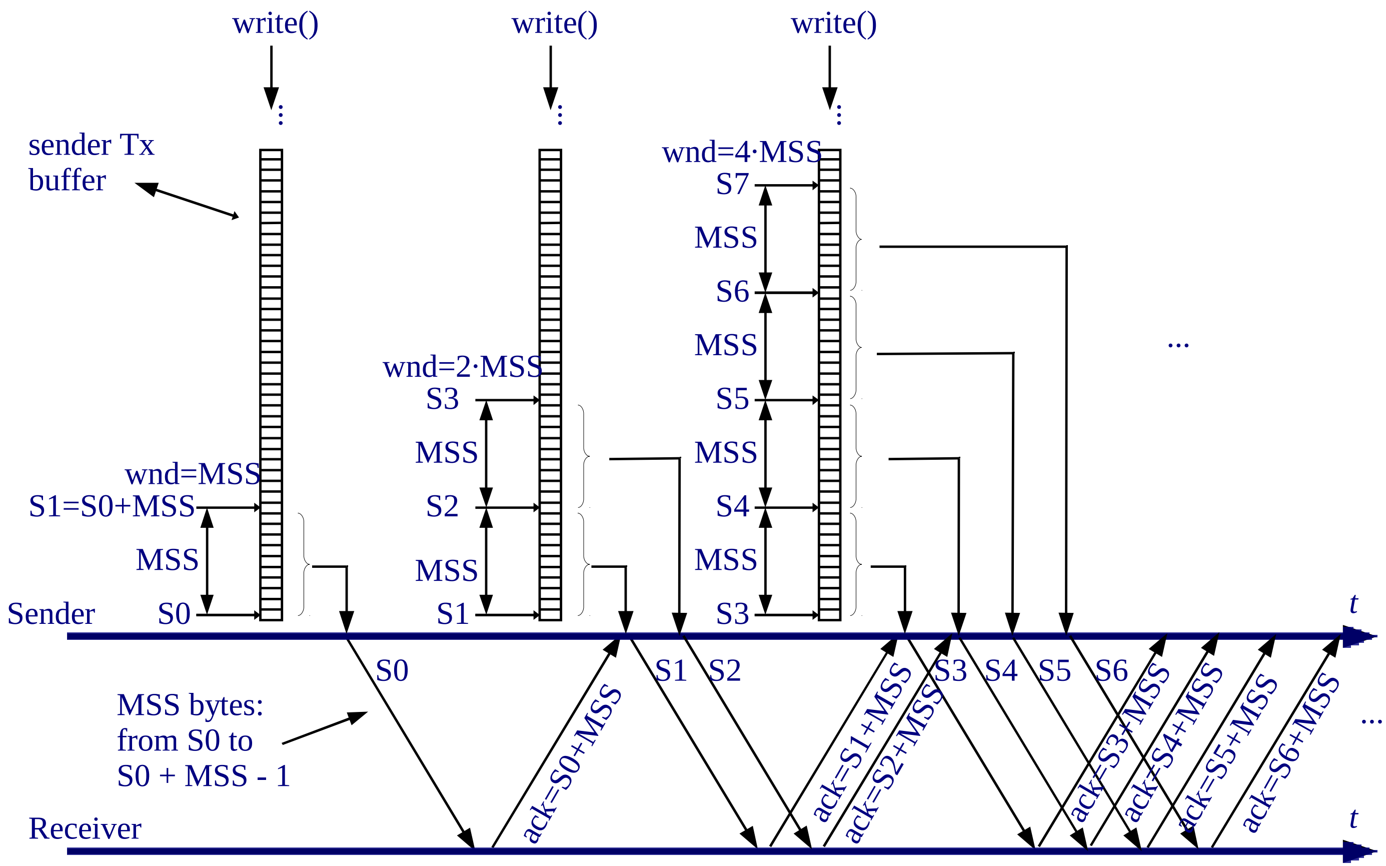
TCP Protocol – TCP Options

- Maximum Segment Size (MSS): Used in the TWH to initialize the MSS.
- Window Scale factor: Used in the TWH. The awnd is multiplied by $2^{\text{Window Scale}}$ (i.e. the window scale indicates the number of bits to left-shift awnd). It allows using awnd larger than 2^{16} bytes.
- Timestamp: Used to compute the Round Trip Time (RTT). Is a 10 bytes option, with the timestamp clock of the TCP sender, and an echo of the timestamp of the TCP segment being ack.
- SACK: In case of errors, indicate blocks of consecutive correctly received segments for Selective ReTx.

Unit 3. Point to Point Protocols -TCP

TCP Protocol – TCP Sequence Numbers

- The sequence number identifies the first payload byte.
- The ack number identifies the next byte the receiver is waiting for.



Unit 3. Point to Point Protocols -TCP

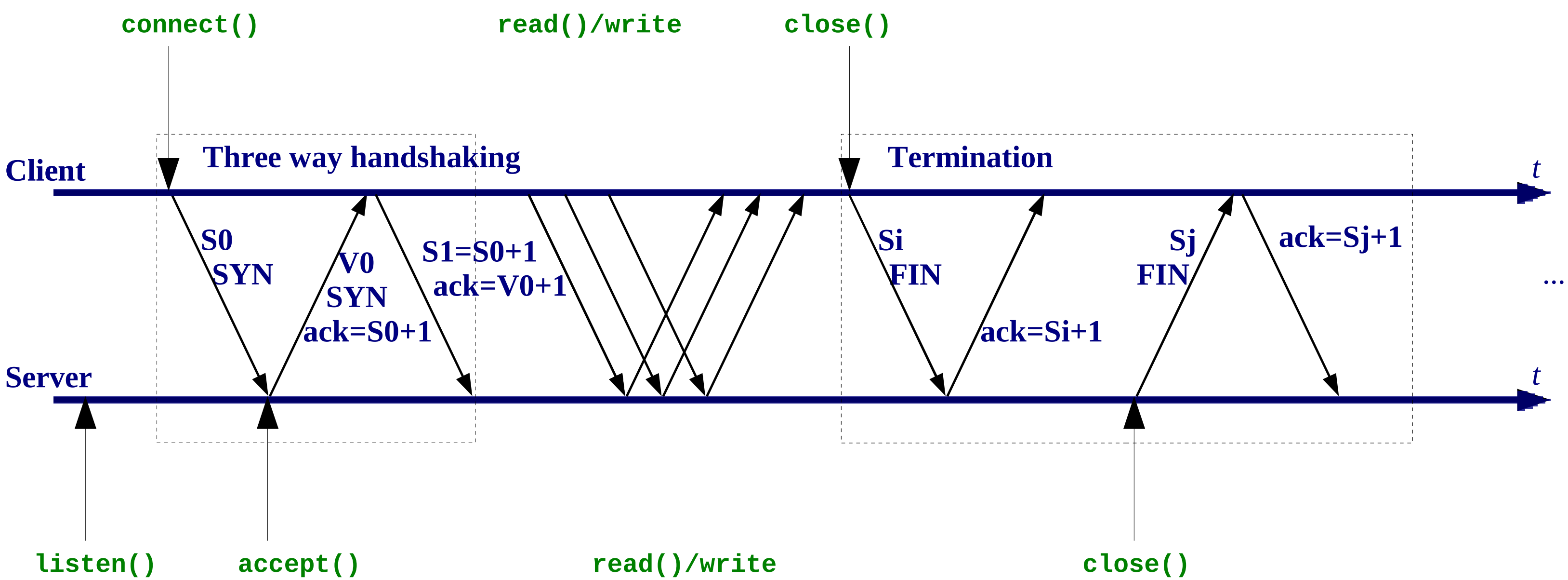
TCP Protocol – C Code example

```
main(int argc, char *argv[])
{
    int sock ;
    struct sockaddr_in serv_addr ;
    struct hostent *host ;

    if(argc != 2) {
        fprintf(stderr, "usage: %s hostname\n", argv[0]) ;
        exit(1) ;
    }
    host = gethostbyname(argv[1]) ; /* call the resolver for server addr. */
    if(host == NULL) {
        perror("gethostbyname ") ;
        exit(2) ;
    }
    bzero(&serv_addr, sizeof(serv_addr)) ;
    serv_addr.sin_family = AF_INET ;
    memcpy(&serv_addr.sin_addr, host->h_addr, host->h_length) ;
    serv_addr.sin_port = htons(3333) ;

    sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) ; /* create a socket */
    connect(sock, (struct sockaddr )&serv_addr, sizeof(serv_addr)) ; /* initiate the connection */
    char msg[] = "hello world\n" ;
    write(sock, msg, strlen(msg)) ; /* write to TCP socket */
    char buf ;
    while(read(sock, &buf, 1) > 0) { /* read from TCP socket */
        printf("%c", buf) ;
    }
    close(sock) ; /* close the socket */
}
```

Unit 3. Point to Point Protocols -TCP

TCP Protocol – Connection Setup and Termination

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



Unit 3. Point to Point Protocols -TCP

TCP Protocol – tcpdump example (web page download)

TWH

```
12:30:37.069541 IP 147.83.34.125.17788 > 147.83.32.82.80: S 3473661146:3473661146(0) win 5840 <mss
                1460,sackOK,timestamp 296476754 0,nop,wscale 7>
12:30:37.070021 IP 147.83.32.82.80 > 147.83.34.125.17788: S 544373216:544373216(0) ack 3473661147 win 5792 <mss
                1460,sackOK,timestamp 1824770623 296476754,nop,wscale 2>
12:30:37.070038 IP 147.83.34.125.17788 > 147.83.32.82.80: . ack 1 win 46 <nop,nop,timestamp 296476754
                1824770623>
12:30:37.072763 IP 147.83.34.125.17788 > 147.83.32.82.80: P 1:602(601) ack 1 win 46 <nop,nop,timestamp 296476754
                1824770623>
12:30:37.073546 IP 147.83.32.82.80 > 147.83.34.125.17788: . ack 602 win 1749 <nop,nop,timestamp 1824770627
                296476754>
12:30:37.075932 IP 147.83.32.82.80 > 147.83.34.125.17788: P 1:526(525) ack 602 win 1749 <nop,nop,timestamp
                1824770629 296476754>
12:30:37.075948 IP 147.83.34.125.17788 > 147.83.32.82.80: . ack 526 win 54 <nop,nop,timestamp 296476755
                1824770629>
```

Termination

```
12:30:53.880704 IP 147.83.32.82.80 > 147.83.34.125.17788: F 526:526(0) ack 602 win 1749 <nop,nop,timestamp
                1824787435 296476755>
12:30:53.920354 IP 147.83.34.125.17788 > 147.83.32.82.80: . ack 527 win 54 <nop,nop,timestamp 296480966
                1824787435>
12:30:56.070200 IP 147.83.34.125.17788 > 147.83.32.82.80: F 602:602(0) ack 527 win 54 <nop,nop,timestamp
                296481504 1824787435>
12:30:56.070486 IP 147.83.32.82.80 > 147.83.34.125.17788: . ack 603 win 1749 <nop,nop,timestamp 1824789625
                296481504>
```

TCP flags

timestamp — src IP addr/port — dst IP addr/port — seq. num:next seq num (bytes) — awnd

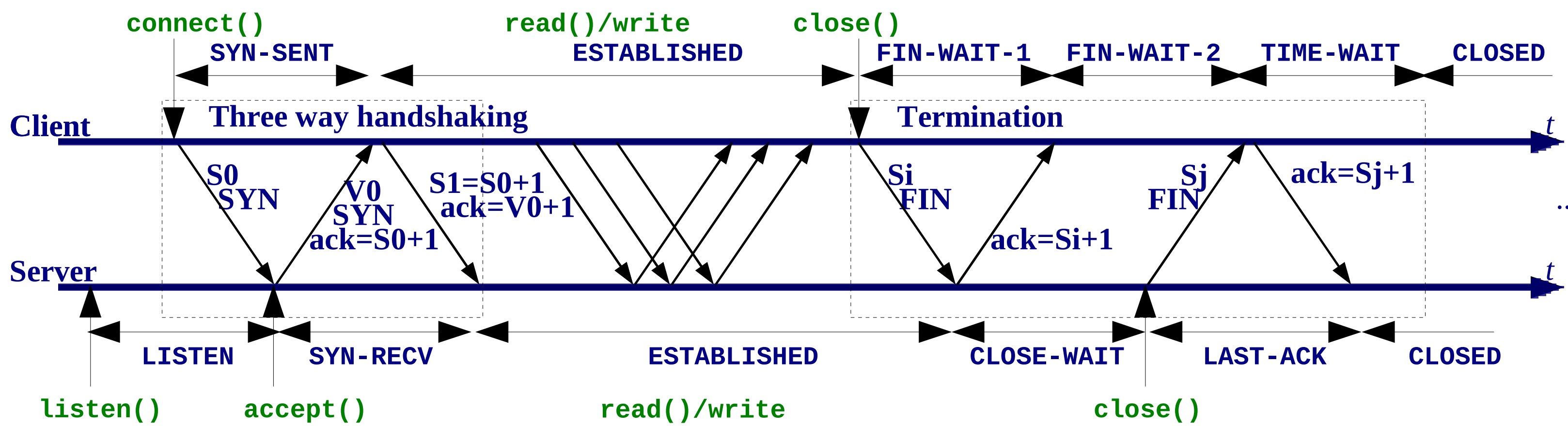
```
12:30:37.069541 IP 147.83.34.125.17788 > 147.83.32.82.80: S 3473661146:3473661146(0) win 5840
            <mss 1460,sackOK,timestamp 296476754 0,nop,wscale 7>
```

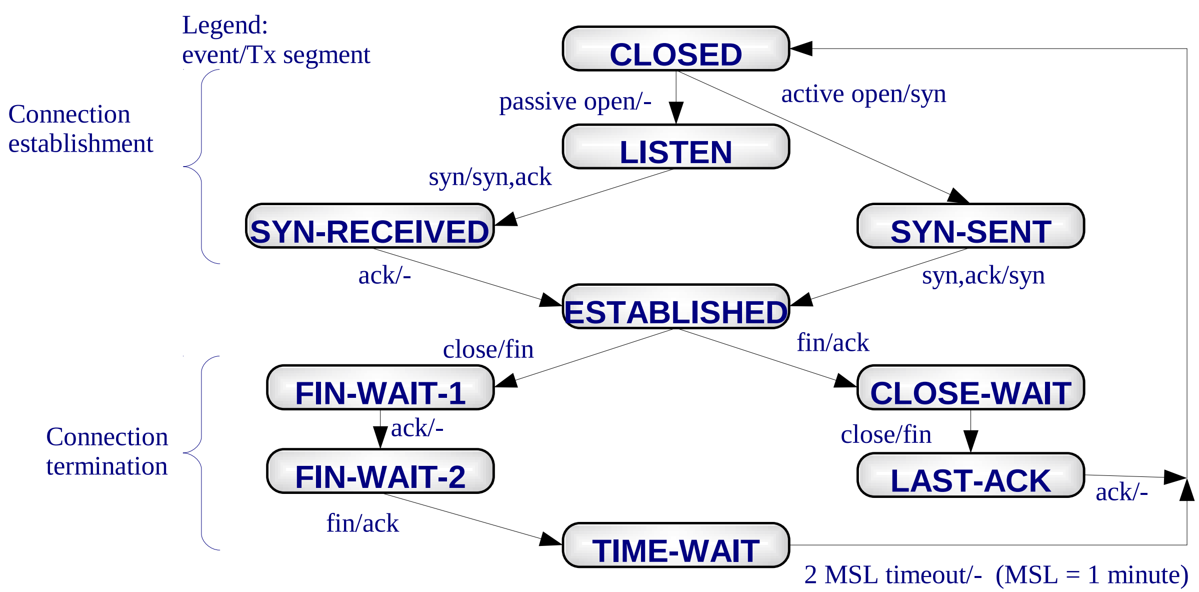
MSS — SACK — timestamp — window scale

padding

Unit 3. Point to Point Protocols -TCP

TCP Protocol – State diagram (simplified)





Unit 3. Point to Point Protocols -TCP

TCP Protocol – netstat dump

- Option -t shows tcp sockets.

```
linux# netstat -nt
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address          Foreign Address         State
tcp        0   1286 192.168.0.128:29537    199.181.77.52:80        ESTABLISHED
tcp        0      0 192.168.0.128:13690    67.19.9.2:80            TIME_WAIT
tcp        0      1 192.168.0.128:12339    64.154.80.132:80        FIN_WAIT1
tcp        0      1 192.168.0.128:29529    199.181.77.52:80        SYN_SENT
tcp        1      0 192.168.0.128:17722    66.98.194.91:80         CLOSE_WAIT
tcp        0      0 192.168.0.128:14875    210.201.136.36:80       ESTABLISHED
tcp        0      0 192.168.0.128:12804    67.18.114.62:80         ESTABLISHED
tcp        0      1 192.168.0.128:25232    66.150.87.2:80          LAST_ACK
tcp        0      0 192.168.0.128:29820    66.102.9.147:80         ESTABLISHED
tcp        0      0 192.168.0.128:29821    66.102.9.147:80         ESTABLISHED
tcp        1      0 127.0.0.1:25911        127.0.0.1:80            CLOSE_WAIT
tcp        0      0 127.0.0.1:25912        127.0.0.1:80            ESTABLISHED
tcp        0      0 127.0.0.1:80           127.0.0.1:25911         FIN_WAIT2
tcp        0      0 127.0.0.1:80           127.0.0.1:25912         ESTABLISHED
```

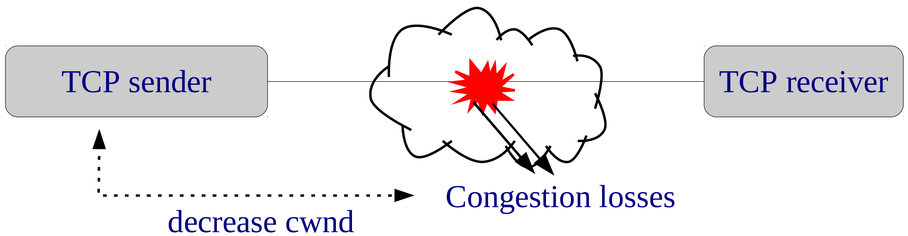
man netstat

```
The count of bytes not acknowledged by the remote host.

The count of bytes not copied by the user program connected to this
socket.
```

Unit 3. Point to Point Protocols -TCP

TCP Protocol – Congestion Control (RFC 2581)

- window = min(awnd, cwnd)
  - The advertised window (awnd) is used for flox control.
  - The congestion window (cwnd) is used for congestion control.
- TCP interprets losses as congestion:



- Basic Congestion Control Algorithms:
  - Slow Start / Congestion Avoidance (SS/CA)
  - Fast Retransmit / Fast Recovery (FR/FR)

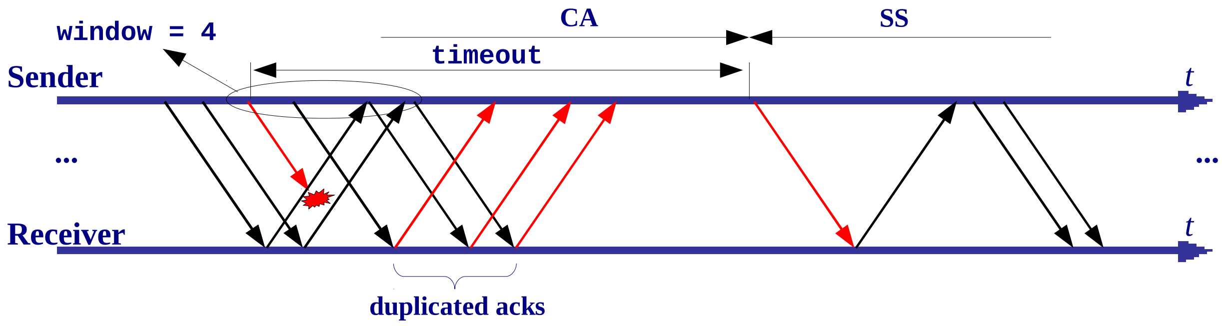
Unit 3. Point to Point Protocols -TCP

TCP Protocol – Slow Start / Congestion Avoidance (SS/CA)

- Variables:
  - snd_una: First non ack segment.
  - ssthresh: Threshold between SS and CA.

```
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) {
                cwnd += MSS ; /* Slow Start */
        } else {
                cwnd += MSS * MSS / cwnd ; /* Congestion Avoidance */
        }

When there is a time-out:
        Retransmit snd_una ;
        cwnd = MSS ;
        ssthresh = max(min(awnd, cwnd) / 2, 2 MSS) ;
```

Time-out Example:

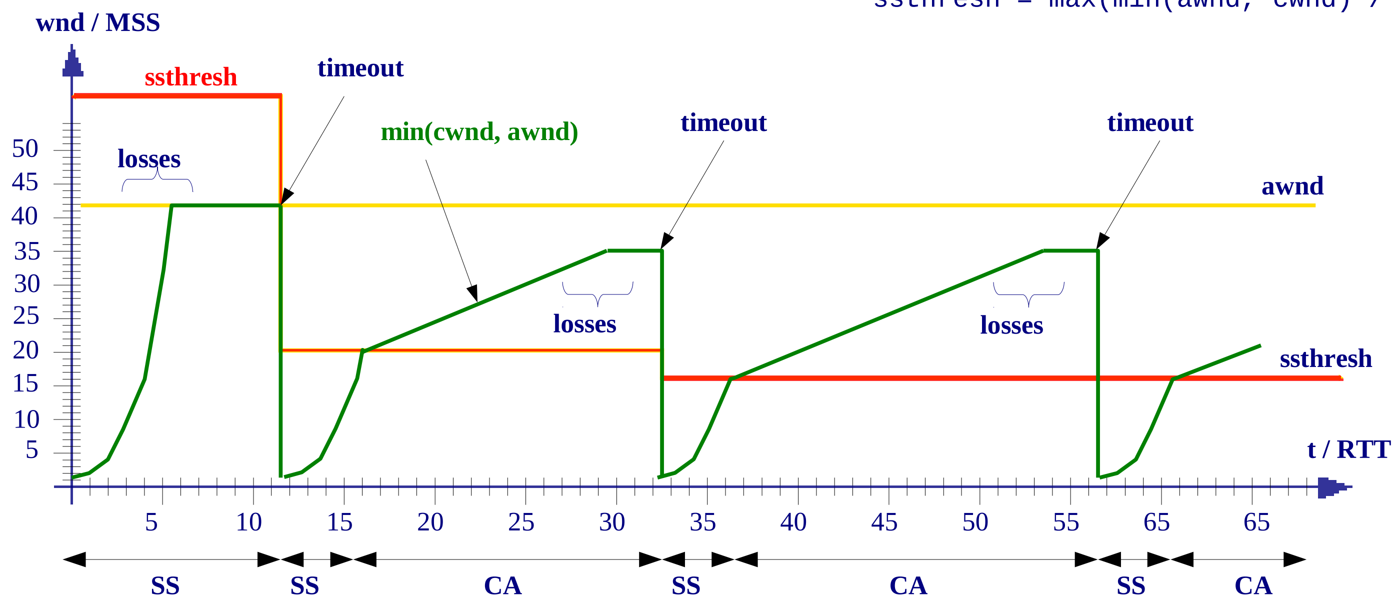


Unit 3. Point to Point Protocols -TCP

TCP Protocol – Slow Start / Congestion Avoidance (SS/CA)

- During SS cwnd is rapidly increased to the "operational point".
- During CA cwnd is slowly increased looking for more available bandwidth.

```
Initialization:
        cwnd = MSS ;
        ssthresh = infinit ;

Each time an ack confirming new data is received:
        if(cwnd < ssthresh) {
                cwnd += MSS ; /* SS */
        } else {
                cwnd += MSS * MSS / cwnd ; /* CA */
        }

When there is a time-out:
        Retransmit snd_una ;
        cwnd = MSS ;
        ssthresh = max(min(awnd, cwnd) / 2, 2 MSS) ;
```



Unit 3. Point to Point Protocols -TCP

TCP Protocol – Fast Retransmit / Fast Recovery (FR/FR)

- Several algorithms are typically used to improve TCP performance.
- A TCP implementation with SS/CA, FR/FR is referred to as TCP-Reno.
- Other improvements include e.g. SACK.

- FR/FR basis: Duplicate acks are indication of losses.

```
Each ack arrival:
        if(it is a duplicated ack) {
                if(it is the 3th duplicated ack) {
                        retransmit snd_una ;
                        ssthresh = max(min(awnd, cwnd) / 2, 2 MSS) ;
                        cwnd = ssthresh + 3 MSS ;
                        fast_recovery = TRUE ;
                } else if(fast_recovery == TRUE) {
                        cwnd += MSS ;
                }
        } else {  /* if new data is ack */
                if(fast_recovery == TRUE) {
                        cwnd = ssthresh ;
                        fast_recovery = FALSE ;
                } else {
                        /* Slow Start / Congestion Avoidance */
                }
        }
```

3th dup-ack arrival

cwnd is set to the wnd estimated previous to the loss + 3 (for the dup acks)

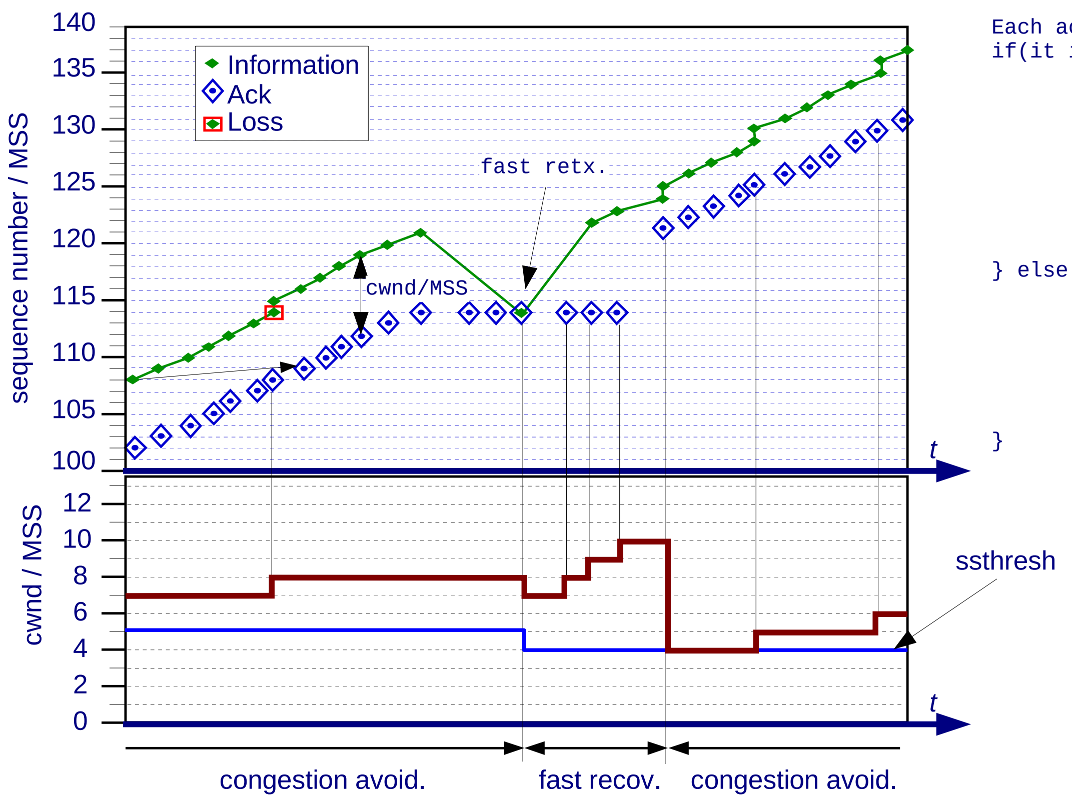
allows sending a new segment for each segment arriving to the TCP receiver

exit FR when new data is ack.

Unit 3. Point to Point Protocols -TCP

TCP Protocol – Fast Retransmit / Fast Recovery (FR/FR)

Time axis at the sender side




```
Each ack arrival:
if(it is a duplicated ack) {
        if(it is the 3th duplicated ack) {
                retransmit snd_una ;
                ssthresh = max(min(awnd, cwnd) / 2, 2 MSS) ;
                cwnd = ssthresh + 3 MSS ;
                fast_recovery = TRUE ;
        } else if(fast_recovery == TRUE) {
                cwnd += MSS ;
        }
} else {  /* if new data is ack */
        if(fast_recovery == TRUE) {
                cwnd = ssthresh ;
                fast_recovery = FALSE ;
        } else {
                /* Slow Start / Congestion Avoidance */
        }
}
```

Unit 3. Point to Point Protocols -TCP

TCP Protocol – Retransmission time-out (RTO)

- Activation:
  - RTO is active whenever there are pending acks.
  - When RTO is active, it is continuously decreased, and a ReTx occurs when RTO reaches zero.
  - Each time an ack confirming new data arrives:
    - RTO is computed.
    - RTO is restarted if there are pending acks, otherwise, RTO is stopped.
- Computation:
  - The TCP sender measures the RTT mean (srtt) and variance (rttvar).
  - The retransmission time-out is given by: RTO = srtt + 4 x rttvar.
  - RTO is duplicated each retransmitted segment (exponential backoff).
- RTT measurements:
  - Using “slow-timer tics” (coarse).
  - Using the TCP timestamp option.

Unit 3. Point to Point Protocols -TCP

TCP Protocol – Retransmission time-out (RTO)