Unit 3. Point to Point Protocols -TCP

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

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 functionalities of the type:
- Error detection
- Error recovery
- Flux control

These are typical data-link layer functionalities, 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.

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

Basic ARQ Protocols:
- Stop & Wait
- Go Back N
- Selective Retransmission

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

Basic ARQ Protocols - Assumptions
- We shall focus on 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_p 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_a) and the receiver sends ack PDUs (A_k).
- Frames carrying I_a respectively A_k are Tx using L_i and L_a bits, thus the Tx times are respectively: t_i = L_i/v_i and t_a = L_a/v_a s.

Simplified time diagram

Time diagram

1. When the sender is ready: (i) allows writing from upper layer, (ii) builds I_a, (iii) I_a goes down to data-link layer and Tx starts.
2. When I_a 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.
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.

![Diagram of retransmission process]

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

![Diagram showing sequence number usage]

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) \( v_f = \) number of inf. bits / obs. time, bps
- Efficiency or channel utilization \( E = v_f / v_t \) (times 100, in percentage)

\[
E = \frac{v_f}{v_t} = \frac{\# \text{info bits} \times t_b}{1/t_b} = \frac{\text{time Tx information}}{T} = \frac{\# \text{info bits}}{1/T}
\]

![Diagram showing throughput and efficiency calculations]

Unit 3. Point to Point Protocols - TCP
Basic ARQ Protocols – Stop & Wait efficiency
- Assuming no errors (maximum efficiency), the Tx is periodic, with period \( T_c \).
- \( E_{\text{protocol}} \): We do not take into account headers.

\[
E_{\text{protocol}} = \frac{t_i}{T_c} = \frac{t_i}{t_i + t_u + 2t_p} = \frac{1}{1 + 2\alpha}, \text{ where } \alpha = \frac{t_p}{t_t}
\]

![Diagram showing efficiency calculation for Stop & Wait protocol]
Unit 3. Point to Point Protocols -TCP

Basic ARQ Protocols – Continuous Tx Protocols
- Goal: Allow high efficiency independently of propagation delay.
- Without errors: $E = 100\%$

![Diagram of Continuous Tx Protocols]

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.

![Diagram of Selective ReTx]

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 go back and starts Tx from that PDU.

![Diagram of Go Back N]

Basic ARQ Protocols – Efficiency with Tx errors: Stop & Wait
- Assumptions: On average, $N_i$ Tx are needed to successfully send a PDU: $N_i – 1$ with Tx errors, and 1 correct.

![Diagram of Stop & Wait]

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

- To avoid unnecessary ReTx $T_O > T_C$. The maximum throughput is when $T_O = T_C$: $E_{protocol} = \frac{t_i}{N_i T_C} = \frac{1}{N_i(1+2\alpha)}$
**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.

\[
E_{\text{protocol}} = \frac{t_t}{T_P} = \frac{t_t}{(N_t-1)T_O + t_t}
\]

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

\[
E_{\text{protocol}} \approx \frac{t_t}{(N_t-1)T_C + t_t} \approx \frac{1}{N_t(1+2a)-2a}
\]

---

**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.

\[
E_{\text{protocol}} = \frac{t_t}{N_t t_t} = \frac{1}{N_t}
\]

---

**Unit 3. Point to Point Protocols - TCP**

**Basic ARQ Protocols – Flux Control and Window Protocols**

- ARQ are also used for flux control. Flux 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 – Efficiency with Tx errors: Comparison**

- For $a << 1$, all protocols are similar ($a = t_f / t_t$)
- 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 – Efficiency with Tx errors: Comparison**

- For $a << 1$, all protocols are similar ($a = t_f / t_t$)
- 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 – Efficiency with Tx errors: Comparison**

- For $a << 1$, all protocols are similar ($a = t_f / t_t$)
- 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 – Window Protocol Flux Control

- Stationary regime

```
<table>
<thead>
<tr>
<th>Protocol</th>
<th>Seq. Numbers (≥)</th>
<th>Bits (≥)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop &amp; Wait</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>GoBackN</td>
<td>W+1</td>
<td>(\log_2(W+1))</td>
</tr>
<tr>
<td>Selective ReTx</td>
<td>2W</td>
<td>(\log_2(2W))</td>
</tr>
</tbody>
</table>
```

Optimal window: Minimum window that allows the maximum throughput.

Clearly, for this example:

\[ W_{opt} = \frac{T_C}{I_f} \]
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

![Diagram of TCP/UDP with client and server]

Ephemeral port (21024)
depth known port <1024
TCP/UDP header

---

UDP Protocol – C Code example

```c
main(int argc, char *argv[]) {
    int sock;
    struct sockaddr_in clnt_addr, serv_addr;
    struct hostent *host;
    int sel_len = sizeof(serv_addr);
    if(argc == 2) {
        fprintf(stderr, "usage: %s hostname
", 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;
    serv_addr.sin_port = htons(33333);
    sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP); /* create a socket */
    clnt_addr.sin_family = AF_INET;
    clnt_addr.sin_port = htons(33333);
    bind(sock, (struct sockaddr *)&clnt_addr, sel_len);
    sendto(sock, msg, strlen(msg), 0, (struct sockaddr *)&serv_addr, sizeof(serv_addr)); /* send a UDP datagram */
    close(sock); /* close the socket */
}
```

---

UDP Protocol – Description (RFC 768)

- 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 do not tolerate large delay variations (which would occur using an ARQ).

---

UDP 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Checksum     |     Protocol        |     UDP length   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Data          |                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

---

UDP pseudo-header

---
Unit 3. Point to Point Protocols -TCP

TCP Protocol – Basic operation
- ARQ window protocol, with variable window: \( \text{wnd} = \min(\text{awnd}, \text{cwnd}) \)
- 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 (\( \text{awnd} \)) is used for flux control.
- The congestion window (\( \text{cwnd} \)) is used for congestion control.

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, ...

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.
  - Iterative: (eg. telnet, ssh) The user interacts with the remote host.
- In iterative 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. Ack's are always sent in case of receiving out of order segments.

- tcpdump example:

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

```
timestamp src IP addr/port dst IP addr/port seq. num:next seq TCP flags num (bytes) ack awnd DF flag in IP header set.
11:27:13.798849 147.83.32.14.ftp > 147.83.35.18.3020: P 9641:11089(1448) ack 1 win 10136 (DF)
```

```
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 |
+-----------------+-----------------+-----------------
```

```
Header length Reserved | R | S | E | V | Advertised window (send)
+-----------------+-----------------+-----------------
```

```
TCP segment header
+-----------------+-----------------+-----------------
| Source Address | Destination Address |
+-----------------+-----------------+-----------------
| Port | Protocol | TCP length |
+-----------------+-----------------+-----------------
```

TCP pseudo-header

```
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
```

```
Header length Reserved | R | S | E | V | Advertised window (send)
+-----------------+-----------------+-----------------
```

```
TCP segment header
+-----------------+-----------------+-----------------
| Source Address | Destination Address |
+-----------------+-----------------+-----------------
| Port | Protocol | TCP length |
+-----------------+-----------------+-----------------
```

```
HTTP transfer
```

```
Write()
```

Small segment
ack
Small segment
...

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.
TCP Protocol – TCP Flags

- ToE
- PSYN
- : No Flag (except ack) is set

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.

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^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.

TCP Protocol – C Code example

```c
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);
    }
    if(host == NULL) {
        perror("gethostbyname ");
        exit(2);
    }
    bzero(&serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;

    connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)); /* initiate the connection */

    char msg[] = "hello world\n";
    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 numbers.
- Initial sequence number is random.

```
connect() 
read()/write 
close()
```

Client

Three way handshaking

Server

listen() 
accept() 
read()/write

close() 

TCP flags

- SYN
- ACK
- RST
- PSH
- URG
- FIN
- Window
- SACK
- ECN

tcp        0   1286 192.168.0.128:29537     192.168.0.128:1239300      ESTABLISHED
tcp        0   1286 192.168.0.128:1239300     192.168.0.128:2953700      ESTABLISHED
tcp        1   1286 192.168.0.128:29529     192.168.0.128:2952900      TIME_WAIT
TCP flags

- SYN
- ACK
- RST
- PSH
- URG
- FIN
- Window
- SACK
- ECN

TCP Protocol – State diagram (simplified)

```
connect() 
read()/write 
close()
```

Client

Three way handshaking

Server

listen() 
accept() 
read()/write

close() 

TCP flags

- SYN
- ACK
- RST
- PSH
- URG
- FIN
- Window
- SACK
- ECN

tcp        0   1286 192.168.0.128:29537     192.168.0.128:1239300      ESTABLISHED
tcp        0   1286 192.168.0.128:1239300     192.168.0.128:2953700      ESTABLISHED
tcp        1   1286 192.168.0.128:29529     192.168.0.128:2952900      TIME_WAIT
TCP flags

- SYN
- ACK
- RST
- PSH
- URG
- FIN
- Window
- SACK
- ECN

TCP Protocol – netstat dump example (web page download)

```
12:30:17.069421 IP 147.83.32.82.80 > 147.83.34.125.17788: S 544373216:544373216(0) ack 3473661147 win 5792 <mss
12:30:17.069438 IP 147.83.34.125.17788 > 147.83.32.82.80: . ack 1 win 46 <nop,nop,timestamp 296476754
TCP flags

Feature: sockStat_t
- State
- Ack
- URG
- FIN
- PSH
- RST
- SYN
- TIME_WAIT
- WAIT
- CLOSE_WAIT
- LAST_ACK
- CLOSING
- ESTABLISHED
- LISTEN
- SYN_RCVD
- SYN_SENT
- SYN
- FIN_WAIT1
- FIN
- SYN-SENT
- LISTEN
- FIN_WAIT2
- FIN_WAIT3
- TIME_WAIT
- CLOSE
- CLOSED

The count of bytes not acknowledged by the remote host.

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

Linux: netstat -nt
Active Internet connections (w/ 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 Protocol – netstat dump example (web page download)

```
12:30:17.069421 IP 147.83.32.82.80 > 147.83.34.125.17788: S 544373216:544373216(0) ack 3473661147 win 5792 <mss
12:30:17.069438 IP 147.83.34.125.17788 > 147.83.32.82.80: . ack 1 win 46 <nop,nop,timestamp 296476754
TCP flags

Feature: sockStat_t
- State
- Ack
- URG
- FIN
- PSH
- RST
- SYN
- TIME_WAIT
- WAIT
- CLOSE_WAIT
- LAST_ACK
- CLOSING
- ESTABLISHED
- LISTEN
- SYN_RCVD
- SYN_SENT
- SYN
- FIN_WAIT1
- FIN
- SYN-SENT
- LISTEN
- FIN_WAIT2
- FIN_WAIT3
- TIME_WAIT
- CLOSE
- CLOSED

The count of bytes not acknowledged by the remote host.

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

Linux: netstat -nt
Active Internet connections (w/ 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 Protocol – netstat dump example (web page download)

```
12:30:17.069421 IP 147.83.32.82.80 > 147.83.34.125.17788: S 544373216:544373216(0) ack 3473661147 win 5792 <mss
12:30:17.069438 IP 147.83.34.125.17788 > 147.83.32.82.80: . ack 1 win 46 <nop,nop,timestamp 296476754
TCP flags

Feature: sockStat_t
- State
- Ack
- URG
- FIN
- PSH
- RST
- SYN
- TIME_WAIT
- WAIT
- CLOSE_WAIT
- LAST_ACK
- CLOSING
- ESTABLISHED
- LISTEN
- SYN_RCVD
- SYN_SENT
- SYN
- FIN_WAIT1
- FIN
- SYN-SENT
- LISTEN
- FIN_WAIT2
- FIN_WAIT3
- TIME_WAIT
- CLOSE
- CLOSED

The count of bytes not acknowledged by the remote host.

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

Linux: netstat -nt
Active Internet connections (w/ 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
```
Unit 3. Point to Point Protocols -TCP
TCP Protocol – Congestion Control (RFC 2581)
- window = min(awnd, cwnd)
  - The advertised window (awnd) is used for flux control.
  - The congestion window (cwnd) is used for congestion control.
- TCP interprets losses as congestion:

  ![TCP sender and receiver diagram]

- Basic Congestion Control Algorithm:
  - 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 ; /* 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) ;

- Time-out Example:

  ![TCP sender and receiver diagram with time-out example]

---

Unit 3. Point to Point Protocols -TCP
TCP Protocol – Slow Start / Congestion Avoidance (SS/CA)
- Several algorithms are typically used to improve TCP performance.
- A TCP implementation with SS/CA, FR/FR is referred to as Reno-TCP.
- 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 3rd duplicated ack) {
      - retransmit snd_una ;
      - ssthresh = max(min(awnd, cwnd) / 2, 2 MSS) ;
      - cwnd = ssthresh ;
      - fast_recovery = TRUE ;
    } else {
      - fast_recovery = TRUE ;
      - cwnd = ssthresh ;
    } else {
      - fast_recovery = FALSE ;
      - cwnd = ssthresh ;
    } else {
      - /* Slow Start / Congestion Avoidance */
    }
  } else {
    - allows sending a new segment for each segment arriving to the TCP receiver
    - exit FR when new data is ack.
  }

  3rd dup-ack arrival

  wnd / MSS

  ssthresh

  min(awnd, cwnd)

  timeout

  timeout

  timeout

  awnd

  losses

  losses

  losses

  ssthresh

  1/4 RTT

  SS

  SS

  CA

  CA

  CA

  SS

  CA

---

Unit 3. Point to Point Protocols -TCP
TCP Protocol – Fast Retransmit / Fast Recovery (FR/FR)
- window = min(awnd, cwnd)
- The advertised window (awnd) is used for flux control.
- The congestion window (cwnd) is used for congestion control.
- TCP interprets losses as congestion:

  ![TCP sender and receiver diagram]

- Basic Congestion Control Algorithm:
  - Slow Start / Congestion Avoidance (SS/CA)
  - Fast Retransmit / Fast Recovery (FR/FR)
Unit 3. Point to Point Protocols -TCP
TCP Protocol – Fast Retransmit / Fast Recovery (FR/FR)

Each ack arrival:
  if (it is a duplicated ack) {
    if (it is the 3rd 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 */
    }
  }

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.