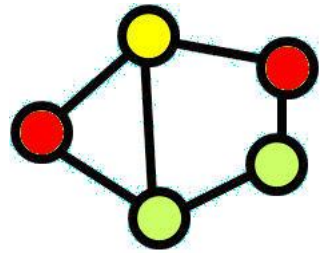
*Slide supporting material*



# Lesson 15: Transport Layer, TCP and UDP

**Giovanni Giambene**

***Queuing Theory and Telecommunications: Networks and Applications***
**2nd edition, Springer**

# Introduction

- We need to consider an intermediate level between network layer (IP protocol layer) and applications; this is the transport layer of the OSI model (layer 4).
- The following 'services' can be optionally provided at the transport level (but not all applications need all these services): connection-orientation, same order delivery, error-free data (reliability), flow control, byte-orientation, and use of ports.

# Introduction (cont'd)

- **Connection-orientation.** Even if the network layer provides a connectionless service, the transport layer often provides a connection-oriented service (in such a case a set up phase is needed).

- **Same order delivery.** The network layer does not generally guarantee that data packets arrive in the same order they were sent. However, the transport layer provides in order delivery.

- **Error-free data.** The underlying network may be noisy and data may be received corrupted. The transport layer deals with this problem by means of a checksum of the received data to detect if errors have occurred. Moreover, IP packets can be lost due to buffer congestion and overflow at the routers. Transport layer may retransmit corrupted packets or lost packets.

# Introduction (cont'd)

- **Flow control & congestion control**. The amount of memory on a computer is limited, and without flow control a powerful computer might flood a computer with so much information that it cannot hold it all before dealing with it. Nowadays, this is not a big issue because memory is cheap, while bandwidth is expensive, but in earlier times of networks this was a more critical issue. **The flow control operated by the transport layer allows the receiver to stop the transmission before it is overwhelmed (layer 4). A similar concept applies to the congestion control, but in this case the control is operated to avoid the congestion of layer 3 buffers at intermediate routers** in the network.

- **Byte orientation**. Rather than dealing with packets, the transport layer views a communication as a stream of bytes.

- **Ports**. Ports are essentially ways to address multiple entities in the same location. Computer applications will each listen to information on their own ports; more than one network-based application can be running at the same time and they are distinguished on the basis of different ports.

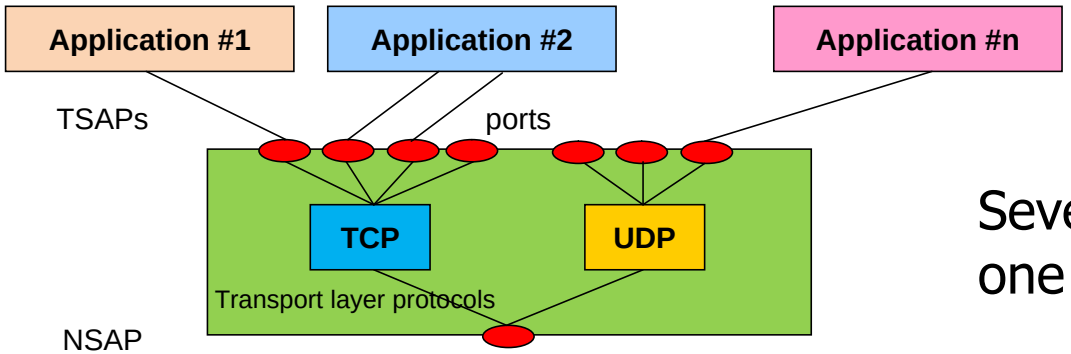
# Introduction (cont'd)

- There are distinct protocols operating at transport layer. These protocols are at least of three different types.
  - **User Datagram Protocol (UDP) defined in RFC 768: it is a connectionless transport layer protocol, which provides a simple and unreliable delivery service for transaction-oriented services**. UDP is basically an interface between IP and upper-layer processes.
  - **Transmission Control Protocol (TCP) introduced with RFC 793: it is a complex protocol, which provides connection-oriented and reliable data transfer to the application layer.**
  - **Multicast protocols** are another important family of transport-layer protocols. These protocols may be reliable (if they guarantee the correct delivery of information to all recipients) or not and require a suitable approach at the network layer. For instance, the **NACK-Oriented Reliable Multicast (NORM) protocol defined in RFC 5740 is an important example of reliable multicast protocol**.

# Ports

- Different applications run on the same device connected to the Internet. **In order to distinguish among them, 16-bit port numbers have been adopted to denote service endpoints.** Source and destination port numbers are specified in both TCP and UDP headers.

- Both TCP and UDP receive requests through transport layer ports (TSAPs) from higher layer protocols, provide a service and send requests through the network layer port (NSAP).




Several TSAPs, one NSAP

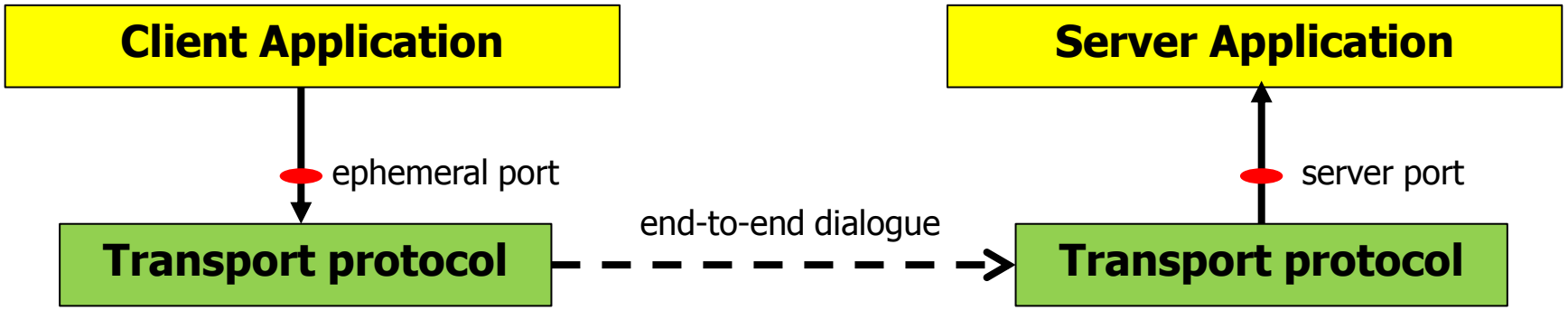
# Ports (cont'd)

- **Port numbers are divided into three ranges (RFC 6335)**:
  - System Ports also called 'well-known' ports (0-1023),
  - User Ports (1024-49151),
  - Dynamic and/or Private Ports (49152-65535).
- System Ports are assigned by IETF. User Ports are assigned by the Internet Assigned Number Authority (IANA). Dynamic Ports are not assigned.
- **TCP and UDP ports are assigned separately, since the services provided by TCP and UDP are different.**

| TCP well-known ports | UDP well-known ports |
|---|---|
| Echo: 7 | DNS: 53 |
| FTP (control): 20 | TFTP: 69 |
| FTP (data): 21 | NTP: 123 |
| Telnet: 23 | SNMP: 161 |
| SMTP: 25 | |
| HTTP: 80 | |

# Ports (cont'd)

- **Well-known port numbers are reserved across different operating systems.**
- User Ports and Dynamic and/or Private Ports can be used for **'ephemeral ports', which are short-lived ports allocated automatically by the TCP/IP software.** They are used on the client in a client-server communication to a well-known port on the server. Moreover, ephemeral ports may also be used on servers.



# Ports and Sockets

- There is some confusion between ports and sockets, since they are closely-related. **A socket is identified by the binding of an NSAP (IP address) and a TSAP (port number).**
- **The socket is part of the operating system (kernel)** of the host and adopts some Application Programming Interfaces (based on Berkeley Software Distribution, BSD, socket).
- Several types of Internet sockets are available. For instance, **connectionless sockets** using UDP and **connection-oriented sockets** using TCP.
- **A FIFO buffer is part of the socket operating in the system kernel between application and network layers at both sender and receiver.** Data received from the network are stored in this buffer, from whence the application can read at its own pace. As the application reads data at the receiver, the receiver buffer space is freed up to accept new data from the network.

# Introduction to TCP

# An Example on How/When TCP is Used

- **TCP is an end-to-end protocol** that is invoked when we open the Web browser (http protocol) and we type an URL (Uniform Resource Locator) address, such as for instance www.wikipedia.org
- The Web browser on the client uses the DNS service to identify the IP address corresponding to the URL and then
  - It **opens a TCP connection** to that IP address using port 80
  - It sends (writes) a request for a reference page, that is 'get index.html'
  - It waits for and receives (reads) the reply in the form of an html page.
- The Web server at the provided IP address
  - It waits for the opening of a TCP connection
  - It waits for and receives (reads) the request in the form of 'get index.html'
  - It sends (writes) the requested html page.

# TCP Basic Characteristics

- **Byte-streams**. TCP data is organized as a stream of bytes: bytes are counted, not datagrams. Nevertheless, TCP information is delivered in blocks (packets), called **segments**.

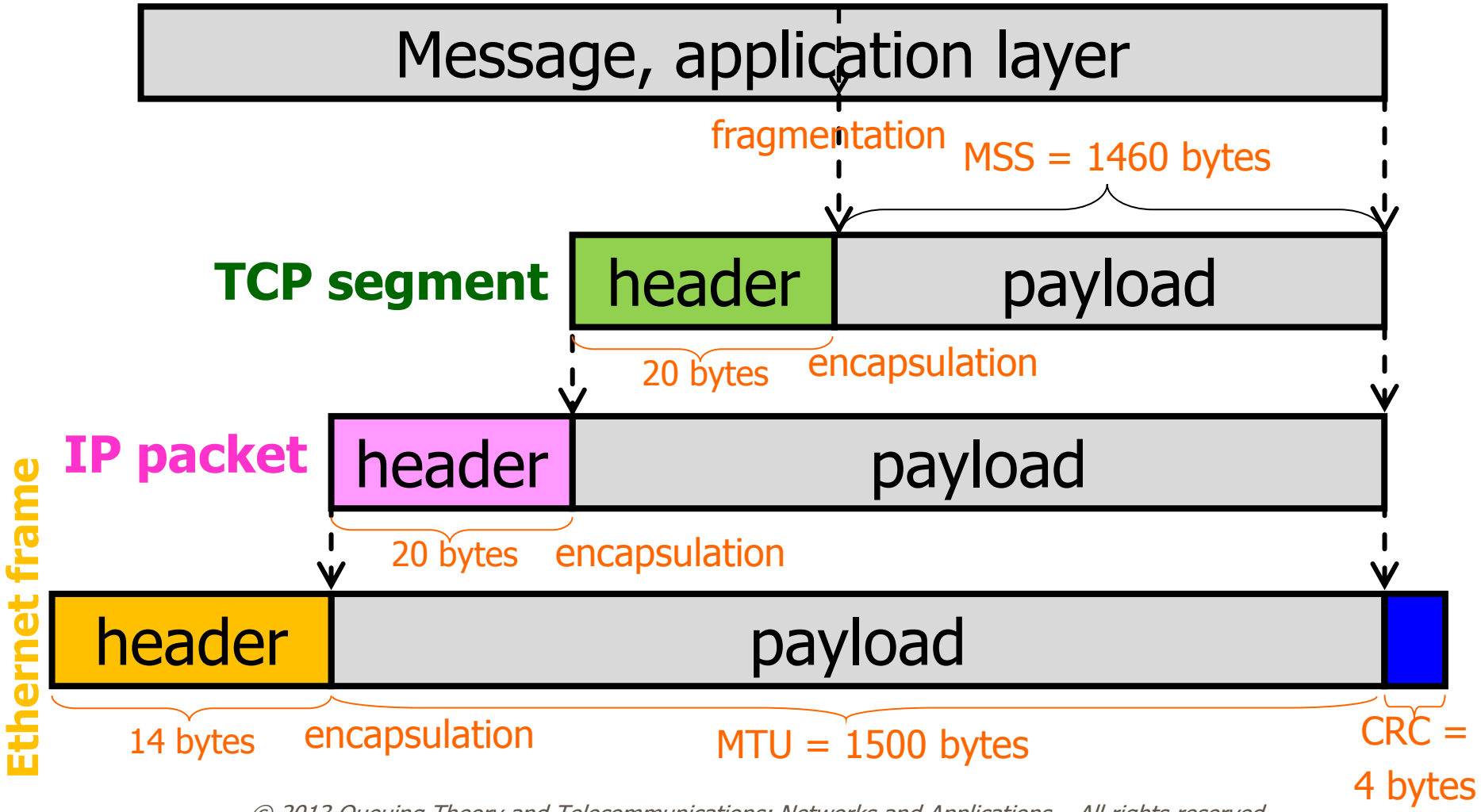
- **Reliable delivery**. Sequence numbers of bytes are used to coordinate which data have been transmitted and received. TCP will arrange for retransmissions if it determines that data have been lost.

- **Network adaptation**. TCP will dynamically infer the status of the network and will adjust its throughput in order to avoid the occurrence of congestion in the network.

# TCP Basic Characteristics (cont'd)

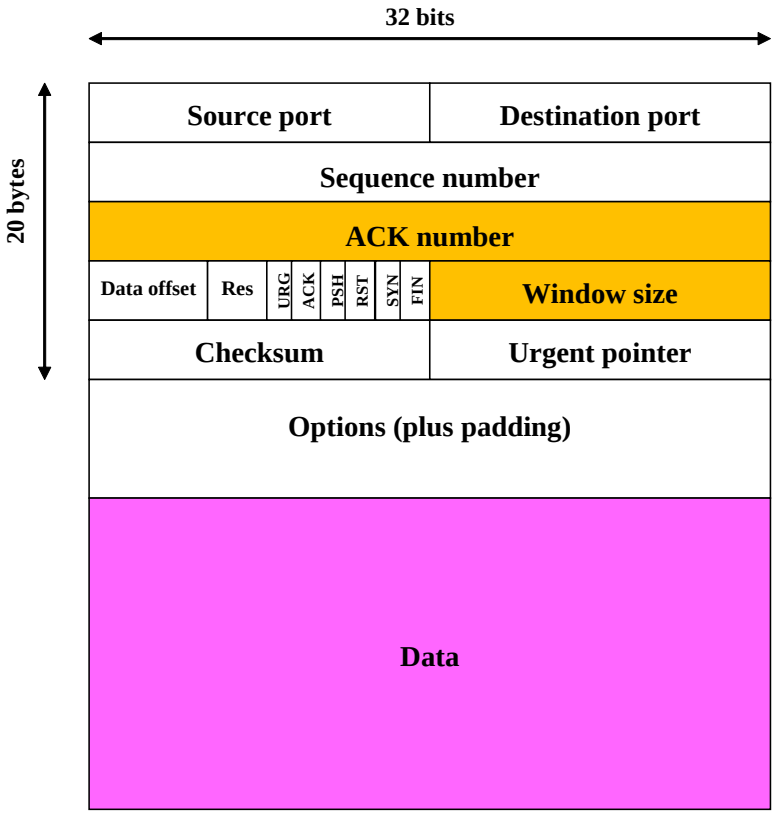
- **Flow control**. TCP controls the congestion of destination buffers in order to avoid overflow events: fast senders will be stopped periodically to keep up with slower receivers.
- **Full-duplex operation**. TCP operates in a full-duplex way: a TCP session entails **two independent byte streams, traveling in opposite directions between the two end-hosts**. During connection start and close phases, TCP can exhibit asymmetric behaviors.
- **TCP is used by mice connections (e.g., HTTP) and by elephant -heavy- ones (e.g., FTP).**
- At the receiver side, IP may receive the datagrams in a wrong order. After IP passes the TCP segments to TCP, **TCP reorganizes these segments according to the correct order**.

# Generation of Packets from Transport to MAC Layer



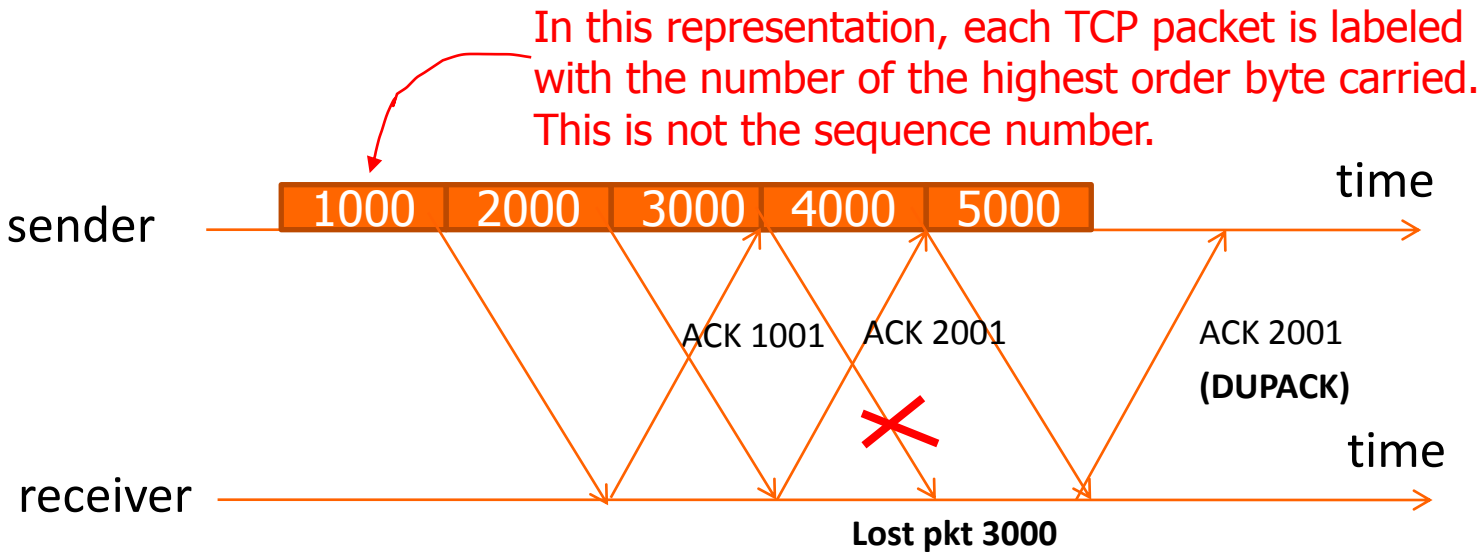
# TCP Segment Format

- The following figure describes the format of TCP segments, organized in words of 32 bits. The size of the **TCP header without options is 20 bytes**.
  - **Source and destination ports** (16 bits each): TCP port numbers of both sender and receiver.
  - **Sequence number** (32 bits): The sequence number of the first byte in the data part of the segment.
  - **Acknowledgment number** (32 bits): If the ACK control bit is set, such field contains the value of the sequence number of the next segment to be transmitted. **This field is intended to acknowledge the last segment received in order (cumulative ACK). ACKs are piggybacked in TCP segments. The mechanism of ACKs is detailed in the next slides.**
  - **Window** (16 bits): The number of bytes beginning with the one indicated in the acknowledgement field that the receiver is able to accept due to the actual occupancy of its buffer (**receiver window for flow control**). If a host/receiver cannot accept more data, it advertises a window equal to zero.
  - **Checksum** (16 bits): it is a parity check for the whole TCP segment that also covers the pseudo-header (source address, destination address, higher layer protocol code, and the TCP segment length).



# Cumulative ACKs for TCP and Packet Losses

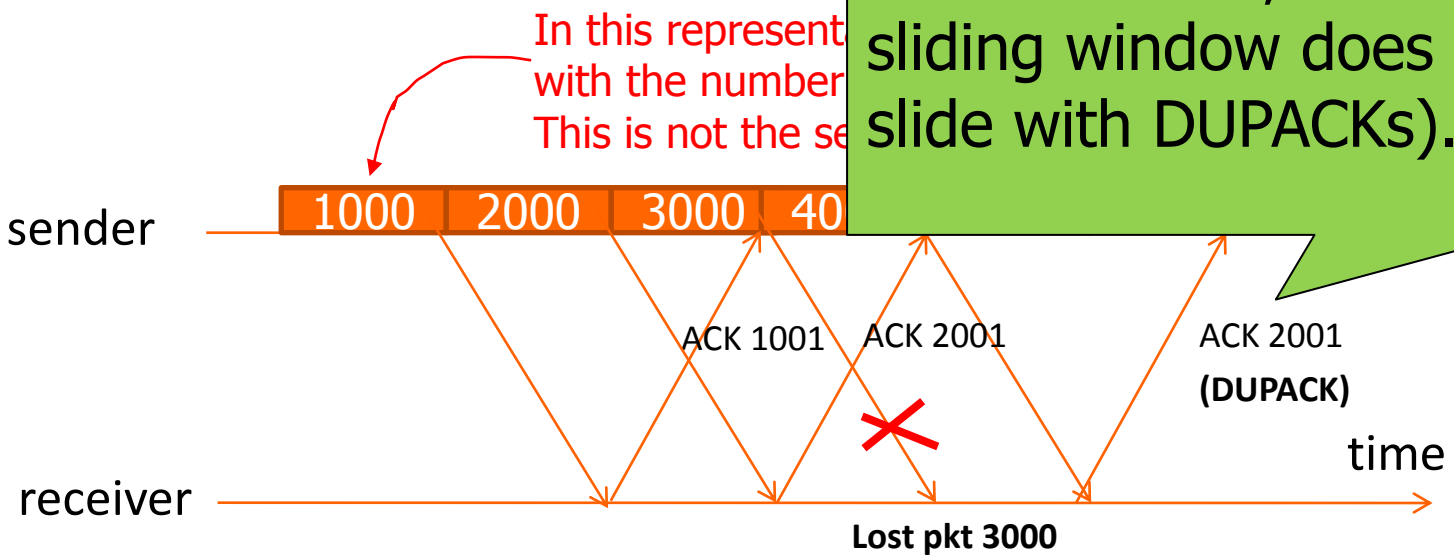
- **Let us remark that TCP adopts a cumulative acknowledgement scheme: an ACK confirms the last segment received in order.**




- An ACK packet is a TCP segment sent in the return direction with respect to the source-destination couple. An ACK packet is at least of 40 bytes (header) at the IP level. **If for some reasons packet 3000 is lost, the next ACK sent after the reception of the segment with number 4000 is still containing the ACK number 2001 (Duplicated ACK, DUPACK).**

# Cumulative ACKs for TCP and Packet Losses

- **Let us remark that TCP adopts a cumu**
  **scheme: an ACK confirms the last seg**

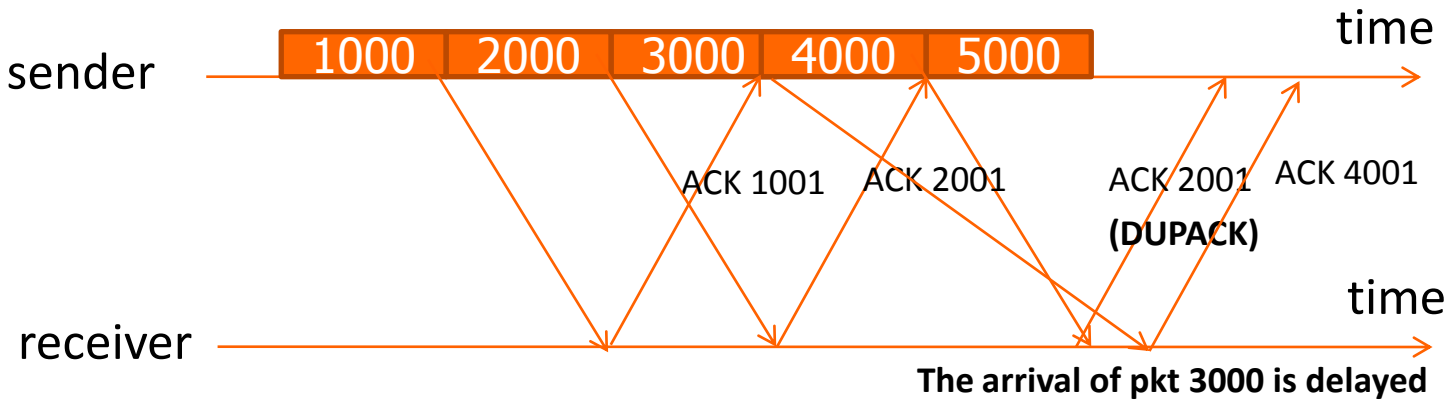



This ACK and the following ones are DUPACKs (as we will show later, the TCP sliding window does not slide with DUPACKs).

- An ACK packet is a TCP segment sent in the return direction with respect to the source-destination couple. An ACK packet is at least of 40 bytes (header) at the IP level. **If for some reasons packet 3000 is lost, the next ACK sent after the reception of the segment with number 4000 is still containing the ACK number 2001 (Duplicated ACK, DUPACK).**

# Cumulative ACKs for TCP and Out-of-Order Packets

- **With respect to the previous example, we have considered here that for some reason packet 3000 is not lost, but just takes another path to reach destination. Hence, this packet arrives out of order. This causes some DUPACKs.**

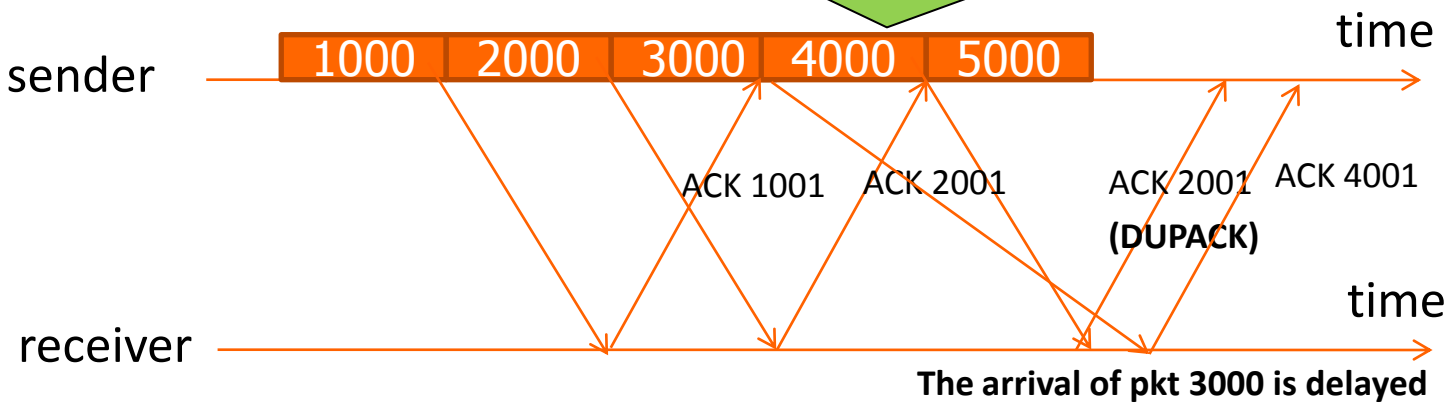



- As soon as packet 3000 is correctly received, the cumulative ACK 4001 is sent, thus recovering the right progress of the ACK numbers: packets 3000 and 4000 in the receiver buffer can be delivered to the higher layer.

# Cumulative ACKs for TCP and O ets

Packet 4000 is received out of order: this packet is stored in the socket of the receiver and not delivered to higher layers until the missing packet 3000 is received.

- **With respe idered here that for some r es another path to reach de of order. This causes son**




- As soon as packet 3000 is correctly received, the cumulative ACK 4001 is sent, thus recovering the right progress of the ACK numbers: packets 3000 and 4000 in the receiver buffer can be delivered to the higher layer.

# Retransmission Time-Out

- **When a packet is transmitted a Retransmission Time Out (RTO) timer is started by the sender.**

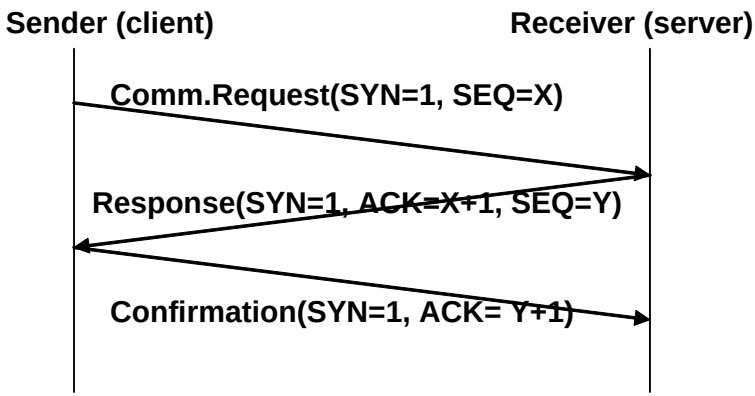
- **The RTO value is continuously updated** on the basis of the measure of the time needed to receive ACKs, called **Round Trip Time (RTT).** RTO represents a filtered version of an estimate of RTT with some margin, proportional to the RTT standard deviation.

- **If the ACK of a packet is not received before RTO expires, it is assumed that a packet loss has occurred due to network congestion** (i.e., overflow in a buffer of a traversed node) and then **retransmissions are performed**, because TCP is a reliable protocol.

**More details on RTT and the RTO algorithm are provided later in this lesson.**

# Start of the TCP end-to-end Transfer: Three-Way Handshake

- TCP uses special segments (with **SYN flag set to 1 in the header**) for establishing a new e2e connection, synchronizing the use of both sequence numbers and ACK numbers on both client and server sides.
  - When a TCP client wants to create a new connection with a remote server, it sends a SYN segment. When the SYN flag is set, it means that synchronization is requested with the remote server. The client also sends **an initial client-side sequence number (random value)**. Moreover, **the TCP header can also contain information on the client-side MSS**; this is achieved by using a suitable TCP option in the header.
  - When the server hears the connection request, the server responds with a **TCP segment with the SYN flag set to 1 containing in the ACK field the sequence number received incremented by 1 (for validation) and containing another initial server-side sequence number (random value).** The TCP header can also contain information on the server-side MSS.
  - The client responds back with a SYN segment where the ACK field contains the sequence number received from the server and incremented of 1**.**



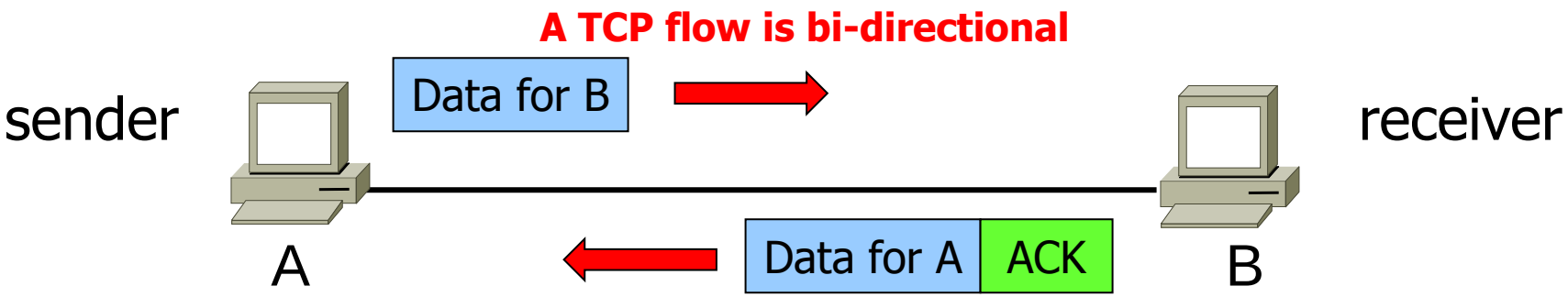
# TCP Flow Control

# TCP Flow Control (RFC 793)

- TCP implements a flow control algorithm based on a **sliding window approach**.
  - We refer to the socket buffer of the operating system between the application program and the network layer. The data received from the network is stored in this buffer. **As the application reads data, the receiver buffer space is freed up to accept new data from the network.**
  - **The window field W (after identified as rwnd) in the TCP header specifies the size of the receiver buffer, less the amount of valid data stored in it**. This is also called the **receiver window** or **advertised window**. Hence, the TCP header permits to inform the sender of the degree of congestion at the receiver.
  - Since the window size field in the TCP header is 16 bits long, the **maximum window size (i.e., room available in the socket buffer at the receiver) is $2^{16}$ = 65536 bytes, corresponding to maximum 44 Ethernet packets of 1500 bytes**.

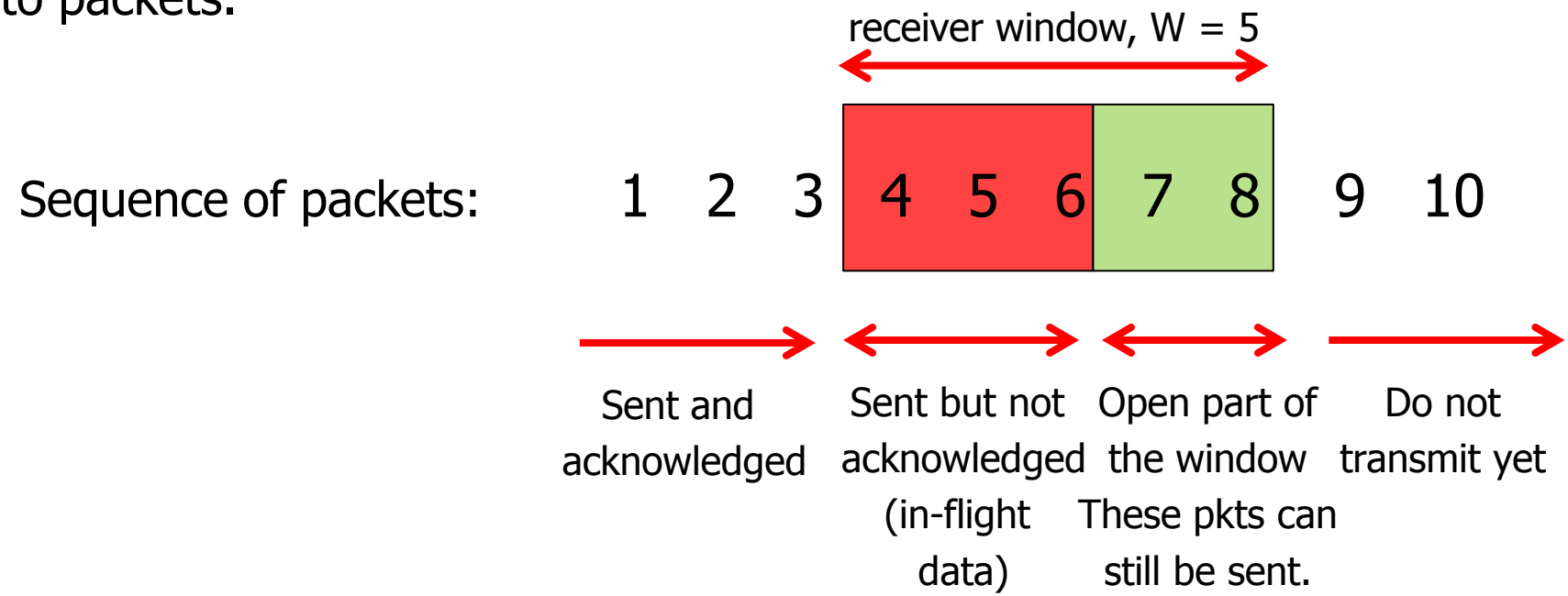
# TCP Flow Control and the Sliding Window

- **The current window value W represents the maximum amount of data that can be sent (in-flight data, outstanding data) without having to wait for ACKs**:
  1. Transmit all the new segments allowed by the current window value W.
  2. Wait for ACKs to arrive; several packets can be acknowledged with the same ACK due to the cumulative ACK scheme allowed by TCP.
  3. **When an ACK arrives, shift the window to the position indicated by the ACK number and set the window size to the value advertised in the ACK window field**; the transmission continues from the packet following the last transmitted one.



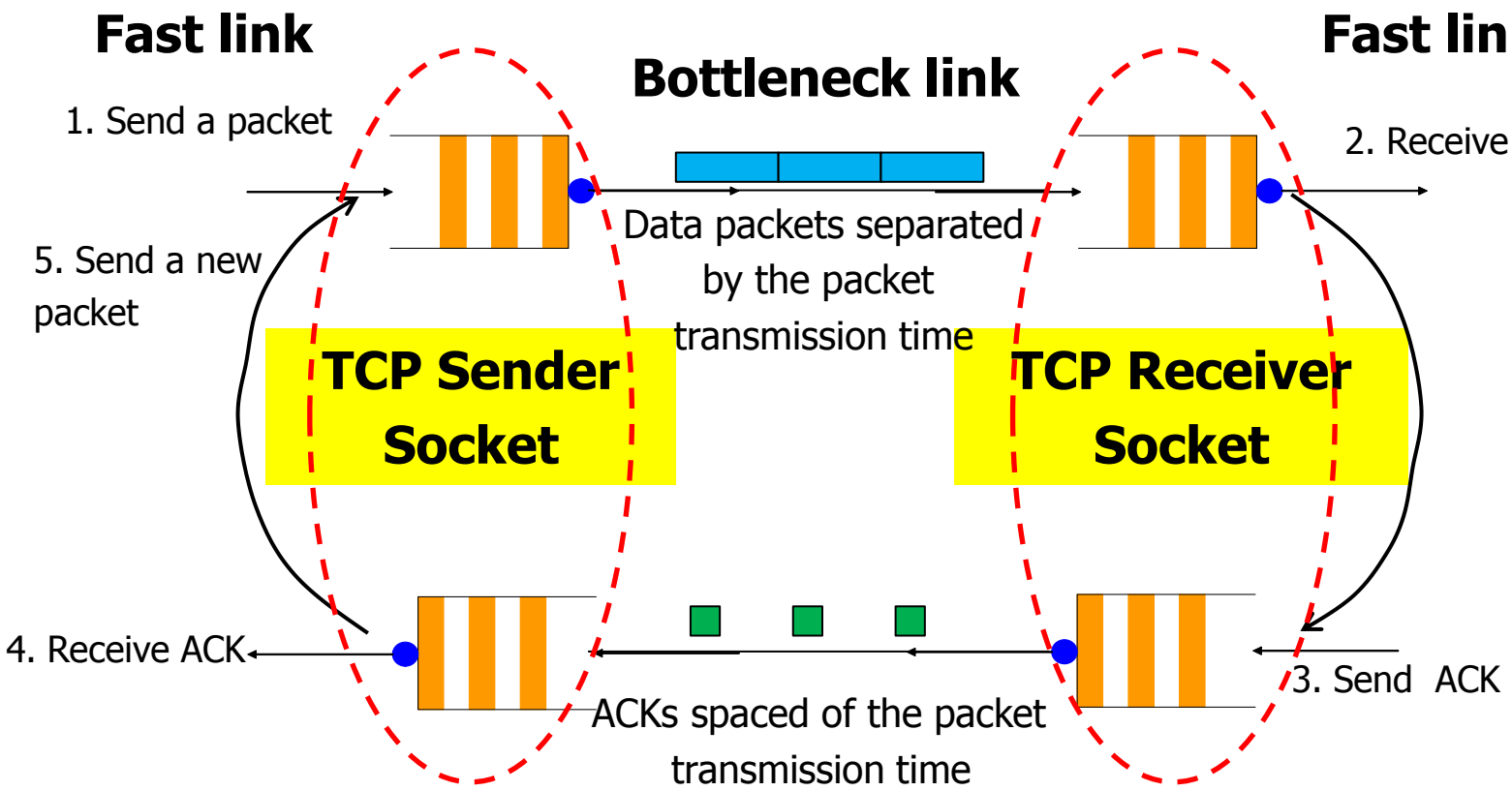
# An Example of the Sliding Window Scheme

- The sliding window scheme is at the byte level, but we refer below to packets.




- In this example, the TCP sender can still transmit packets with sequence numbers 7 and 8 even without receiving new ACKs. The arrival of new ACKs allows the window to slide to the right to transmit more packets.

# Flow Control: ACK Self-Clocking Model

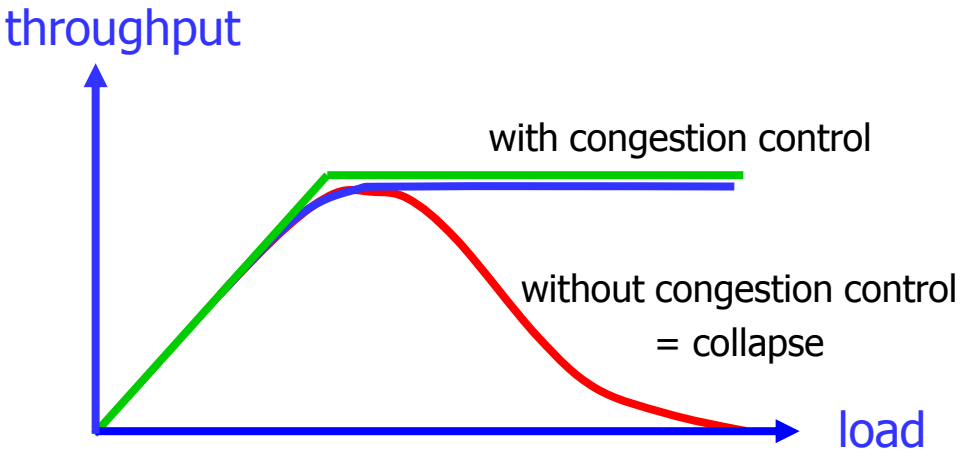



RTT is the time from step #1 to step #4.

# TCP Congestion Control

# Congestion Control

- **On October 1986, Internet had its first congestion collapse event.** The network was totally congested, providing few bit/s of goodput per user. Most users gave up and reconnected later.
- Congestion entails:
  - Packet losses due to buffer overflows
  - Retransmissions to recover packet losses
  - Drastic throughput reduction.




- **In 1986, Van Jacobson proposed a first congestion control and a flow control scheme that are integrated in the same mechanism, based on sliding windows.**

# RTT and RTD

- **Round Trip Time, RTT**: this is the time for a packet to travel from source to destination and the time back to receive its ACK at the source. RTT includes the packet transmission time, queuing delays at the traversed nodes, the ACK packet transmission time, and the e2e physical propagation delay.
- **Round Trip propagation Delay, RTD, the minimum possible RTT**, only accounting for the e2e physical propagation delay in the medium. RTD is much easier to determine than RTT; sometimes RTT is substituted by RTD.
- **The ping command of ICMP provides a measure of RTT**:

```
Execution of Ping to 193.205.7.1 with 32 bytes of data:
Answer from 193.205.7.1: byte=32, RTT duration=129 ms, TTL = 116
……
Answer from 193.205.7.1: byte=32, RTT duration=190 ms, TTL=116
Statistics for the Ping to 193.205.7.1:
    Packets: Transmitted = 4, Received = 4,
    Lost  = 0 (0%),
Approximated RTT in ms:
    Minimum = 129 ms, Maximum =  190 ms, Average =  150 ms
```

# Network Model for TCP Congestion Control Study

In this model, we consider that the size of the socket buffers is sufficiently high that they do not limit the TCP traffic.

**TCP sender**

**TCP receiver**

Arrival of an IP packet with length MTU = MSS+ TCPheader + +IP header = = MSS + 40 bytes.

**Let IBR (Information Bit-Rate) denote the capacity at IP layer of the bottleneck link along the source to destination path**

**IP layer transmission buffer with capacity of B packets**

**RTT ≈ RTD + queuing delays**

Queuing delays are due to buffer congestion on the source-to-destination path.

# Bandwidth-Delay Product

- An important parameter for the TCP behavior and performance is the **Bandwidth-Delay Product (BDP), defined as:**

$$BDP = \frac{IBR \times RTT}{MSS + TCP/IPheader} \; [packets]$$

= MTU

- **BDP represents the maximum number of packets that can be in-flight (outstanding) in the pipe from source to destination.**
- **BDP is computed practically substituting RTT with RTD.**
- **If BDP has a high value (around 10 pkts according to RFC 1062 or around 100 pkts or more according to RFC 1323), the network is said to be a "Long, Fat pipe Network" (LFN).**
- **We consider a new (sliding) window, with size $W_t$, for the flow & congestion control integrated mechanisms**; as before, $W_t$ limits the injection of packets in the network.

# Bandwidth-

RTD is used in this formula (instead of RTT) when doing theoretical investigations.

- An important parameter … mance is the **Bandwidth-Delay Product (…), defined as:**

$$BDP = \frac{IBR \times RTT}{MSS + TCP/IPheader} \; [packets]$$

= MTU

- **BDP represents the maximum number of packets that can be in-flight (outstanding) in the pipe from source to destination.**
- **BDP is computed practically substituting RTT with RTD.**
- **If BDP has a high value (around 10 pkts according to RFC 1062 or around 100 pkts or more according to RFC 1323), the network is said to be a "Long, Fat pipe Network" (LFN).**
- **We consider a new (sliding) window, with size $W_t$, for the flow & congestion control integrated mechanisms**; as before, $W_t$ limits the injection of packets in the network.

# Flow & Congestion Control, Integrated

- **Flow control** to avoid overloading receiver with too much data.
  - **rwnd: receiver (advertised) window**, set in the window size field in the header of the TCP packet sent back as ACK. The maximum (initial) rwnd value is 65535 (= $2^{16}-1$) bytes.
  - **rwnd is updated by the receiver** depending on the occupancy of its transport layer socket buffer: rwnd closes when new data are received and rwnd re-opens when data are read from the socket buffer and delivered to higher layers.
- **Congestion control** to avoid overloading network with too much data.
  - **cwnd: congestion window. It is continuously updated by the sender** on the basis of a congestion control algorithm, which permits to infer the network congestion status on the basis of ACKs received.
- The sender uses the sliding window **$W_t$ = min (cwnd, rwnd) to determine the amount of new data that can be injected into the network (in-flight data) without having to wait for ACKs.**

# The First Congestion Control Algorithm

- The TCP congestion control algorithm is managed by the sender to control/limit the amount of data injected into the network towards a destination (i.e., the receiver) **without any coordination with other hosts, but only on the basis of its perception** of network congestion on the basis of the ACKs received.
- The classical TCP congestion control treats **the network as a black box** and probes network resources by increasing gradually the **amount of injected data ($W_t$)** on the basis of the ACKs received. **The TCP congestion control algorithm conceived by Van Jacobson in 1986 is composed of two phases:**
  - **'slow start'** and
  - **'congestion avoidance'**.
  - This first congestion control scheme **was not included in an RFC**, but it was just implemented under the name of **TCP Berkeley** (UNIX).

# The First Congestion Control Algorithm (cont'd)

- **Cwnd is updated by the sender at each ACK received, which acknowledges new data.**
- The 'slow start' phase and the 'congestion avoidance' one are performed on the basis of the value a cwnd threshold, **ssthresh**, which is dynamically updated.
  - The **initial default ssthresh value** is typically set to the **initial rwnd value,** i.e., 65535 bytes. In the following study, however, we will also consider lower ssthresh values.
- **Note that cwnd, rwnd, $W_t$, and ssthresh have values expressed in bytes, but for the following considerations their values are considered as converted in segment units.** This allows a simpler description even if not totally accurate, since the actual congestion control behavior has a finer granularity.

# The First Congestion Control Algorithm (cont'd)

- Depending on the comparison of cwnd and ssthresh, 'slow start' or 'congestion avoidance' algorithms are used to increase cwnd.
  - **If cwnd < ssthresh, the 'slow start' algorithm is adopted**: the following cwnd update is performed at the receipt of a new ACK:

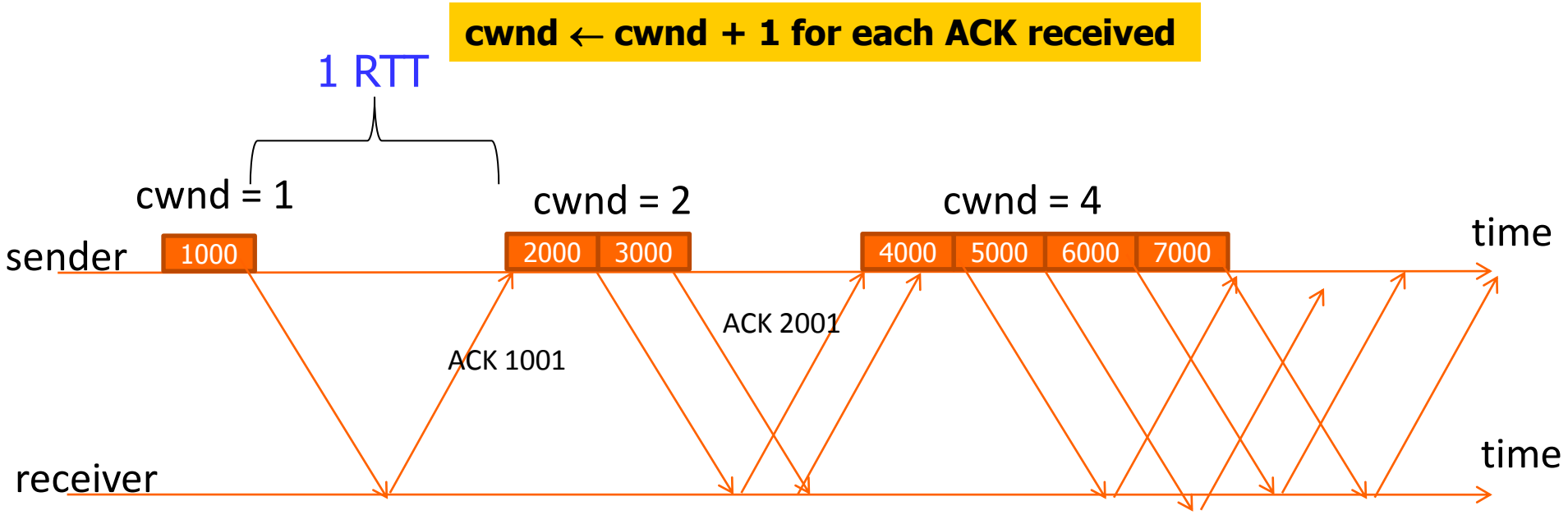
    **cwnd = cwnd + 1 [MSS unit].**

    Correspondingly, **cwnd doubles (exponential increase) on an RTT basis**. In spite of its name, the 'slow start' algorithm tries to enlarge (i.e., to open) the congestion window in a sufficiently-fast (but controlled) way.
  - **As soon as cwnd increases beyond ssthresh, the 'congestion avoidance' algorithm is invoked**: the following cwnd update is performed at the receipt of a new ACK :

    **cwnd = cwnd + 1/cwnd [MSS unit].**

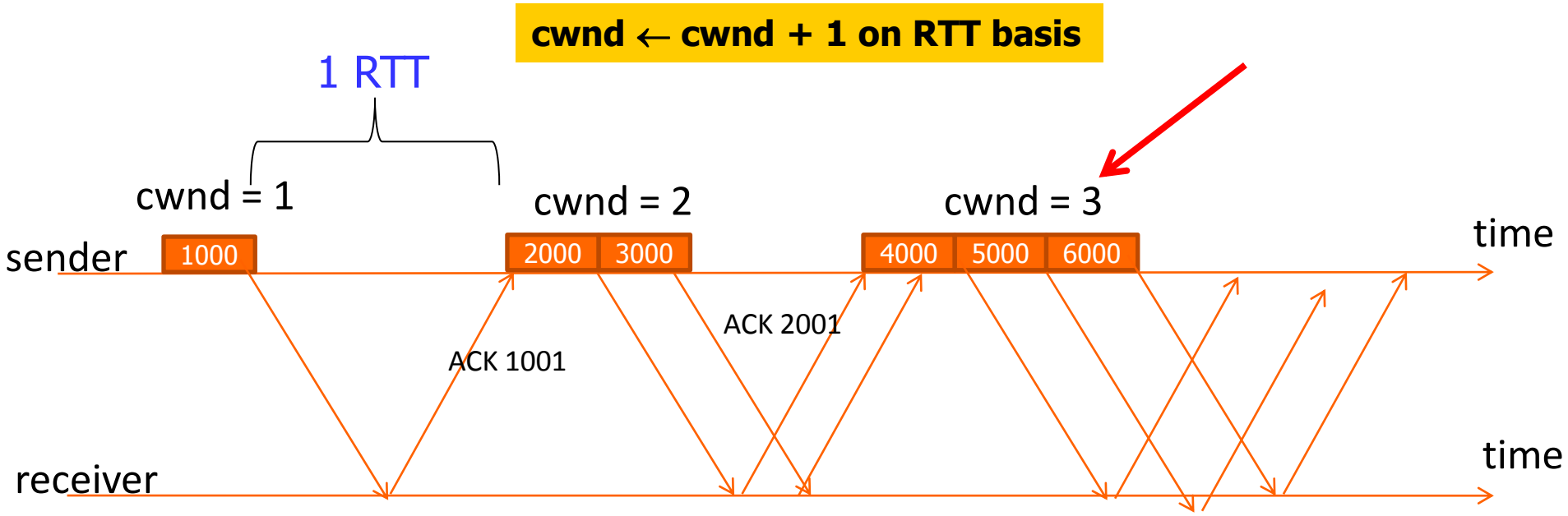
    **For each block of cwnd segments sent/received in an RTT time, cwnd increases of 1: cwnd has a linear increase on an RTT basis.** This solution permits to gently probe the bandwidth still available in the network.

# Slow Start: Exponential cwnd Increase on RTT Basis




Exponential increase of the cwnd size (1, 2, 4, 8, etc.) and of the number of packets sent on an RTT basis.

# Congestion Avoidance: Linear cwnd Increase on RTT Basis




Linear increase of the cwnd size (1, 2, 3, etc.) and of the number of packets sent on an RTT basis.

# Cwnd Behavior

- In the congestion avoidance phase:
  - **If abscissa is expressed in RTT units, the cwnd behavior is linear.**
  - **If abscissa is expressed in seconds, the cwnd behavior is curved (increasing and concave down), especially if B > BDP**.
    - This is because RTT increases with cwnd due to the increase in buffer congestion.

# RTO and Congestion Control

- **When RTO expires for a given packet**:
  1. ssthresh is set to one-half of the current minimum value between cwnd and rwnd;
  2. **cwnd is reset to its initial value (i.e., 1 MSS) to force the 'slow start' algorithm**;
  3. RTO is doubled;
  4. The **sender retransmits all packets starting from the one for which RTO has expired (Go-Back-N approach).**
- The RTO mechanism drastically reduces the TCP traffic injection since it resets cwnd.
- When the overload condition disappears, the normal value of RTO is restored.

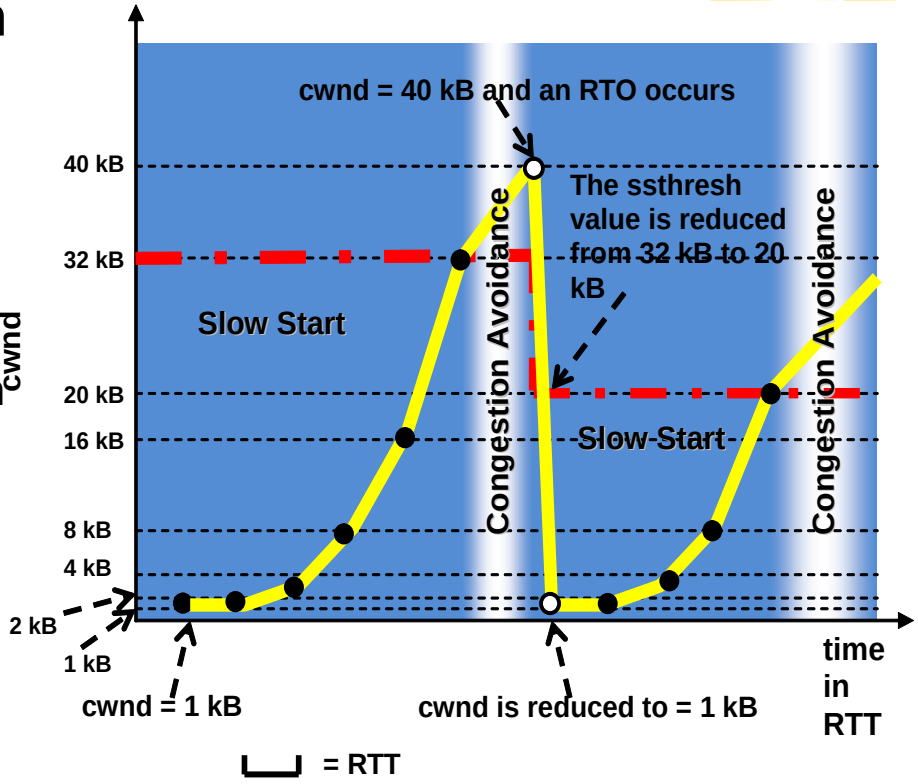
**More details on the RTO algorithm are provided at the end of this lesson.**

# Congestion Control Study Assumption (Model)

- For the study of the TCP congestion control algorithm we make the following simplifying assumptions:
  - **All packets of a cwnd window are sent altogether as in a burst.**
  - **All ACKs corresponding to the packets of a cwnd are received altogether after an RTT.**
  - **The cwnd value is updated on an RTT basis.**
  - **The rwnd value is assumed to be so high (rwnd >> cwnd) that it has no influence in determining the $W_t$ value: $W_t \equiv$ cwnd.**
  - **BDP is computed using RTD instead of RTT. BDP computed with RTT is equivalent to (model) BDP+B, where BDP is computed with RTD.**

# An Example of cwnd Behavior

- An example of cwnd behavior is shown here as a function of time, expressed in RTT units for initial ssthresh = 32 kB, MSS = 1 kB, and rwnd = ∞
  - At the beginning of a TCP connection, during the 'slow start' phase, cwnd has an **exponential increase** ($y = 2^x$, where x is in RTT units);
  - As soon as cwnd reaches the ssthresh value, cwnd **linearly increases** ($y = x+b$) due to the 'congestion avoidance' algorithm.
  - We assume that when cwnd = 40 kB, a **congestion event** occurs in the network (e.g., RTO expires for a given TCP segment): **TCP sets ssthresh to half of last cwnd value (i.e., ssthresh = 20 kB) and resets cwnd to 1 to trigger a 'slow start' phase.**




The TCP behavior described here corresponds to **TCP Berkeley**. A similar cwnd behavior could be obtained by **TCP Tahoe**, but in this case the packet loss is recognized by DUPACKs before RTO expires.

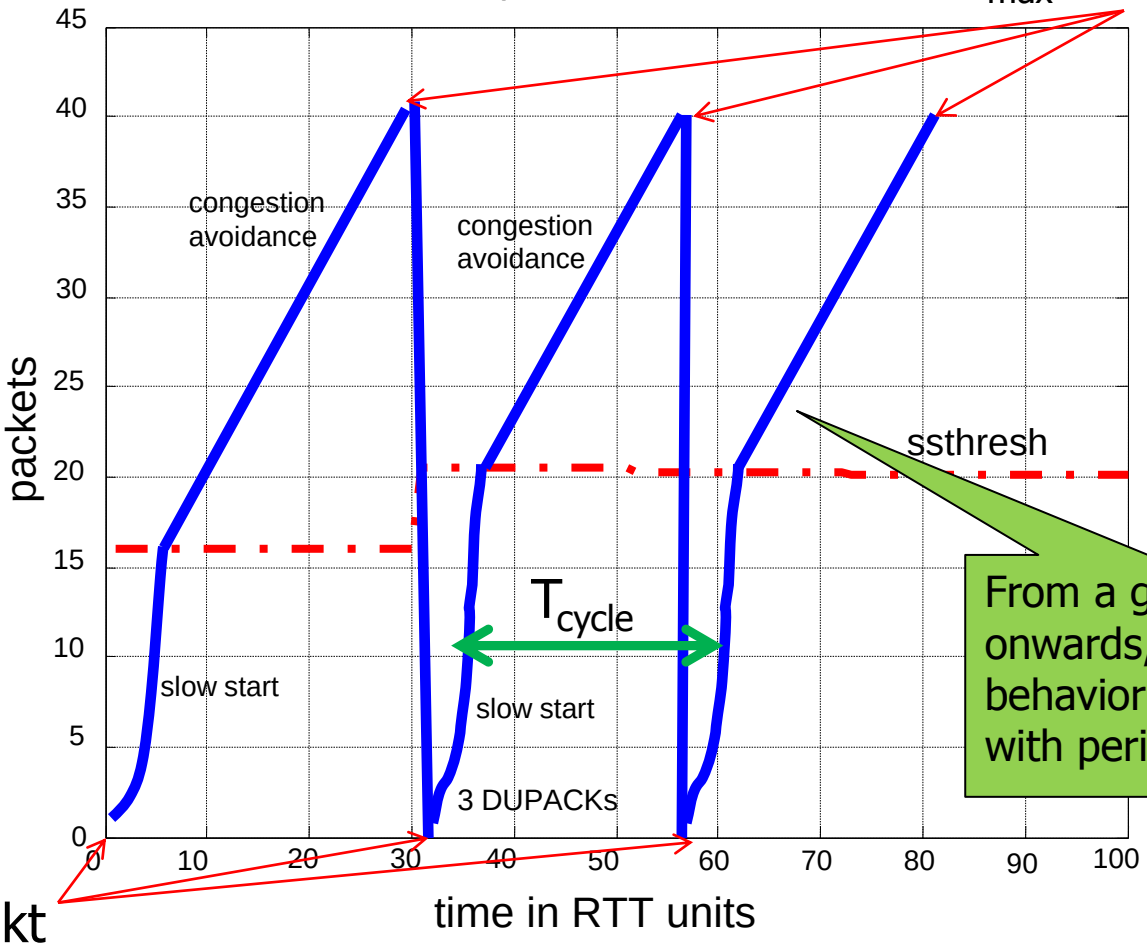
# TCP Tahoe Version (1988)

- TCP Tahoe adopts **slow start** and **congestion avoidance**.
- Moreover, if the sender receives **3 DUPACKs**, Tahoe assumes that there was a packet loss and reacts **as if an RTO expiration occurred**: Tahoe performs a "fast retransmit" phase, ssthresh ← cwnd/2, cwnd ← 1 and restarts from a slow start phase, forgetting everything on the segments sent after the lost one (Go-Back-N). This approach may reduce the throughput too much.
- A packet loss is not decided at the first DUPACK, but at the third one in order not to react too fast, especially because the IP network is connectionless and **out-of-sequence packets could be misinterpreted as packet losses**.

# Cwnd Behavior with TCP Tahoe

We refer to our network model with B, BDP, IBR, etc.

$cwnd_{max}$ = B + BDP

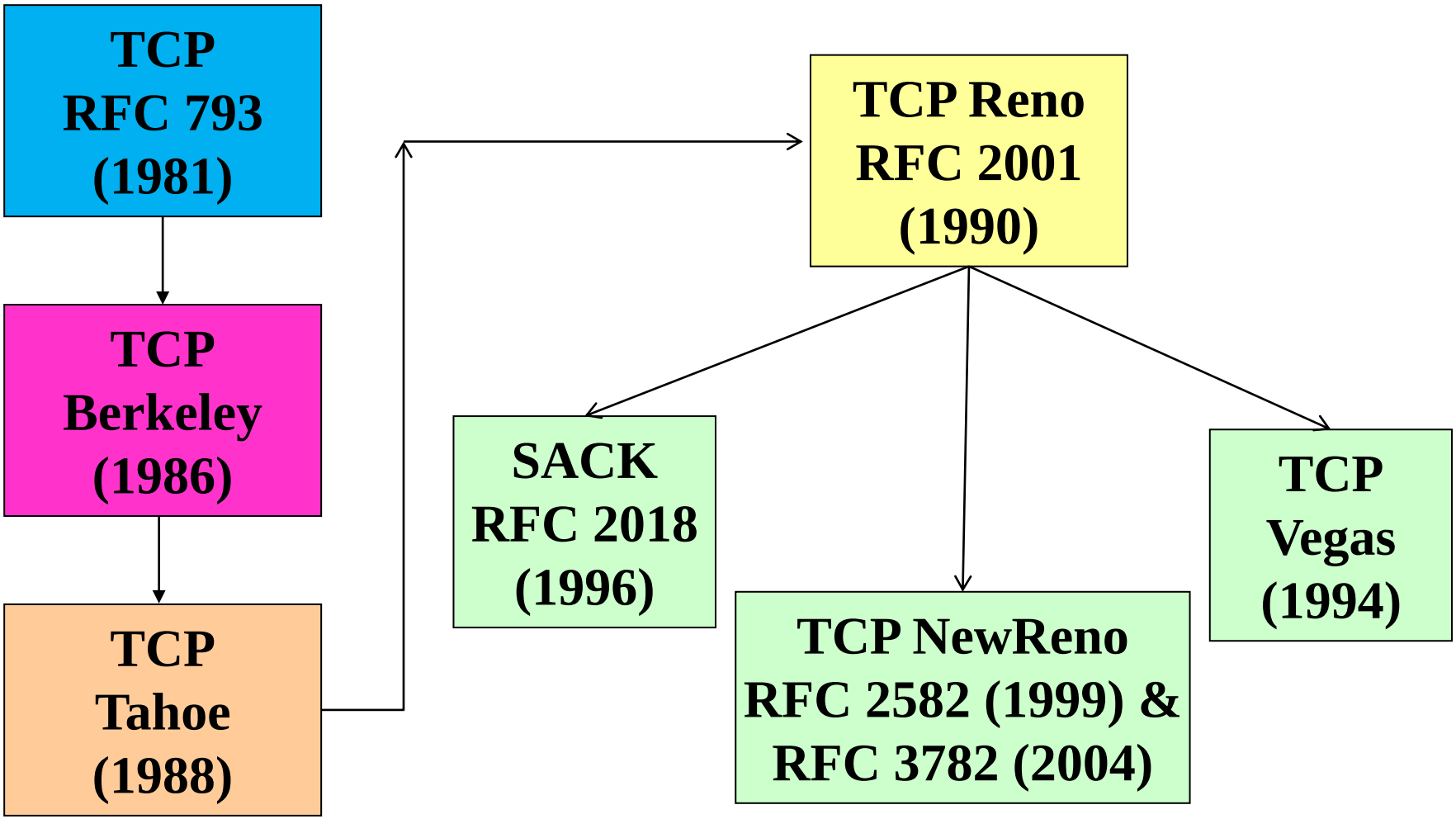



Where this BPD is computed with RTD.

From a given time onwards, the cwnd behavior is periodic, with period $T_{cycle}$.

cwnd = 1 pkt

# TCP Congestion Control: Some Reference Versions



# TCP Performance: Throughput and Goodput

- The TCP performance is measured in two different ways:
  - **Throughput (sender-side): the bit-rate R(t) injected in the network by the TCP sender.**
  - **Goodput (receiver-side): the bit-rate corresponding to the correctly-received segments at the TCP receiver.**
- We can also consider average values of throughput, $\Gamma$ , and goodput, $\gamma$.
- If there are frequent packet losses in the network it may happen that

**Average throughput $\Gamma$ >> Average goodput $\gamma$.**

# Throughput

- The instantaneous throughput R(t) and the average throughput $\Gamma$ can be characterized as:

$$R(t) = \frac{cwnd(t)}{RTT(t)}$$

Instantaneous throughput

$$\Gamma = \frac{1}{T_{cycle}} \int_{T_{cycle}} cwnd(t)\,dt \quad \left[\frac{\text{segments}}{\text{RTT units}}\right]$$

Average throughput on a cwnd cycle

- If cwnd is measured in packets, $\Gamma$ is obtained in TCP segments/s (or segments/RTT unit).
- If cwnd is too small ($\Gamma$ « IBR), the network capacity is underutilized. While, if cwnd is too high, there can be congestion and packet losses.

# Throughput

$t_{cycle}$ depends on BDP where RTT can be approximated by RTD [RTT is almost constant and equal to RTD only if B ≈ 0].

- The instantaneous throu… throughput $\Gamma$ can be characte…:

$$R(t) = \frac{cwnd(t)}{RTT(t)}$$

Instantaneous throughput

$$\Gamma = \frac{1}{T_{cycle}} \int_{T_{cycle}} cwnd(t)\,dt \quad \left[\frac{\text{segments}}{\text{RTT units}}\right]$$

Average throughput on a cwnd cycle

- If cwnd is measured in packets, $\Gamma$ is obtained in TCP segments/s (or segments/RTT unit).
- If cwnd is too small ($\Gamma$ « IBR), the network capacity is underutilized. While, if cwnd is too high, there can be congestion and packet losses.

# Average Goodput (Measurements) and Efficiency

- The average goodput can be measured as follows:

$$\gamma = \frac{\sum \text{packets received (in order) up to time } T}{T}$$

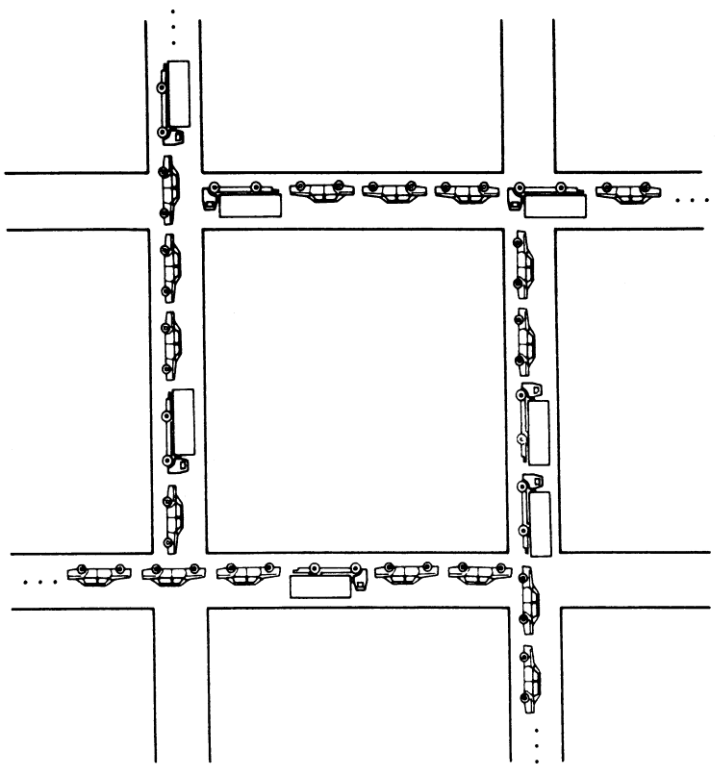
- Efficiency $\eta$ can be measured as average goodput $\gamma$ divided by IBR:

$$\eta = \frac{\gamma}{IBR}$$

If there are no packet losses due to the medium (but just the periodical packet losses due to buffer congestion) we have: $\Gamma \approx \gamma$.

# TCP Deadlock

- **Deadlocks are complex events, which cause a bock in the data transmission**, with a condition that resembles traffic jam in the streets.
- Deadlock events may happen under special circumstances in the TCP case, where **sender and receiver are both waiting for the other to finish**, so that none of them can send new data.
- Some TCP deadlock events are due to implementation (known) problems.



Traffic jam as an illustration of TCP deadlock.

# TCP Deadlock (cont'd)

- A first deadlock case
  - The following case refers to a slow receiver. If the receiver buffer is full of data, then it sends an ACK to the sender containing a window size rwnd = 0. This stops sender transmissions. The receiver sends a window update segment (with rwnd > 0) when it has space available in its buffer. If this window update segment is lost, then a deadlock occurs.

# TCP Deadlock (cont'd)

- A second deadlock case
  - Another deadlock problem could be caused by a **circular-wait condition between sender and receiver** due to the adoption of the Nagle algorithm (RFC 896) jointly with the delayed acknowledgment scheme (RFC 813).
    - **The Nagle algorithm limits the number of outstanding small segments** (segments smaller than MSS) to one in order to avoid inefficiency. The **delayed acknowledgment strategy prevents a receiver from acknowledging small segments by delaying ACKs** until they can be piggybacked onto either a data segment or a window update packet.
  - There is the risk the sender will not send small segments due to the Nagle algorithm and the receiver will not send ACKs because of the delayed ACK algorithm.

# RTO Algorithm, Details

# Retransmission TimeOut (RTO), Calculation

- **RTO should be greater than RTT, but not much bigger than RTT in order not to waste time to react to congestion.** Hence, an accurate dynamic determination of RTO is needed.
- When a packet is sent a timer is started; when the ACK corresponding to the same sequence number is received an **RTT measure** is obtained. Let RTT(i) denote the i-th RTT measure.
- The current RTO value (RFC 6298) is updated by the TCP sender using **two state variables based on RTT measures: SRTT (Smoothed RTT, an average RTT value) and RTTVAR (RTT VARiation, a sort of standard deviation of RTT)**.

$$RTTVAR(i+1) = (1-\beta) \times RTTVAR(i) + \beta \times |SRTT(i) - RTT(i)|$$

$$SRTT(i+1) = (1-\alpha) \times SRTT(i) + \alpha \times RTT(i)$$

Low-pass filters

$$RTO(i+1) = SRTT(i+1) + \max\{G, K \times RTTVAR(i+1)\}$$

where K = 4, $\alpha$ = 1/8, $\beta$ = 1/4, G represents the clock granularity (tick).

# Retransmission Timeout (RTO), Calculation (cont'd)

- In many implementations, RTT is not measured for every segment received, but typically **only for one segment per window of data**.
- RTT is measured as a discrete variable (**granularity**), in multiples of a **"tick"**.
- 1 tick = 500 ms in many implementations.
- According to the **Karn and Partridge algorithm** (RFC 6298), RTT measurements are not taken when an RTO or a packet retransmission occurs, because these RTT measures would be inaccurate.
- **RTO should be at least 2 clock ticks (= 1 s). Instead, the maximum RTO value is 60 s.**

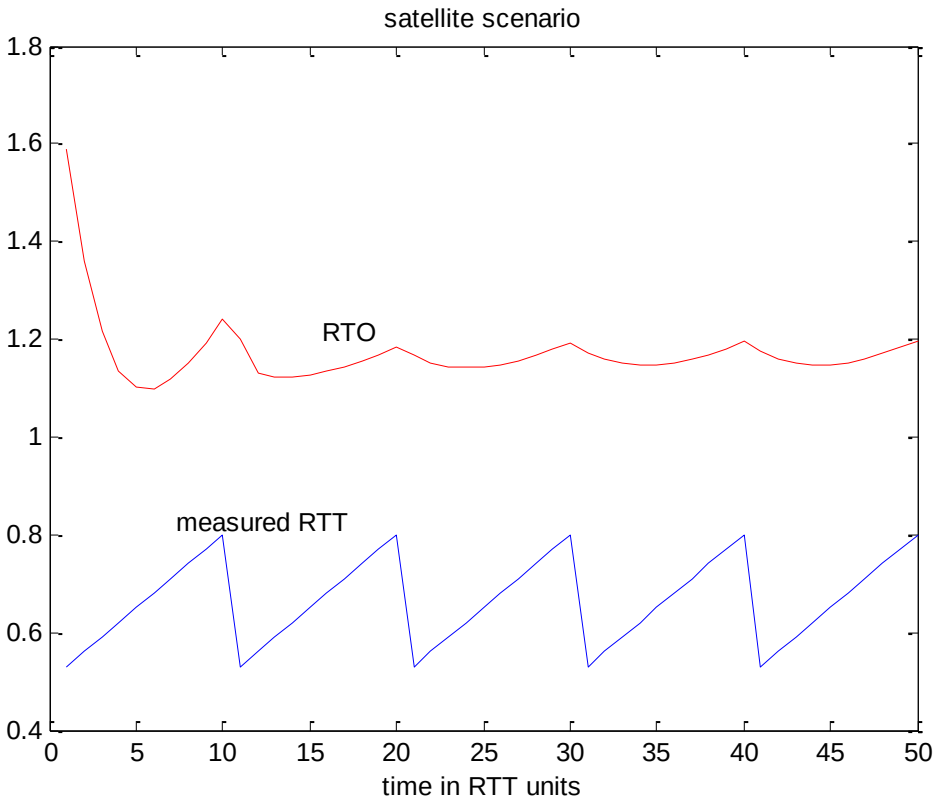
# Retransmission Timeout (RTO), Calculation (cont'd)

- **Large variations in the RTT (typically due to queuing phenomena) increase the deviation RTTVAR(i), leading to a larger RTO value**.

- Whenever an RTO expiration occurs, RTO is increased by some factor before retransmitting the non-ACKed data.

  - **Typically, RTO is doubled at each expiration according to an exponential backoff algorithm. When the overload condition disappears, TCP reduces its RTO to its normal SRTT-based value.**

# RTO Behavior Example (GEO Satellite RTD of 0.5 s)




In doing this graph, continuous values have been considered for both RTT and RTO (no ticks).

# UDP

# UDP Basic Characteristics

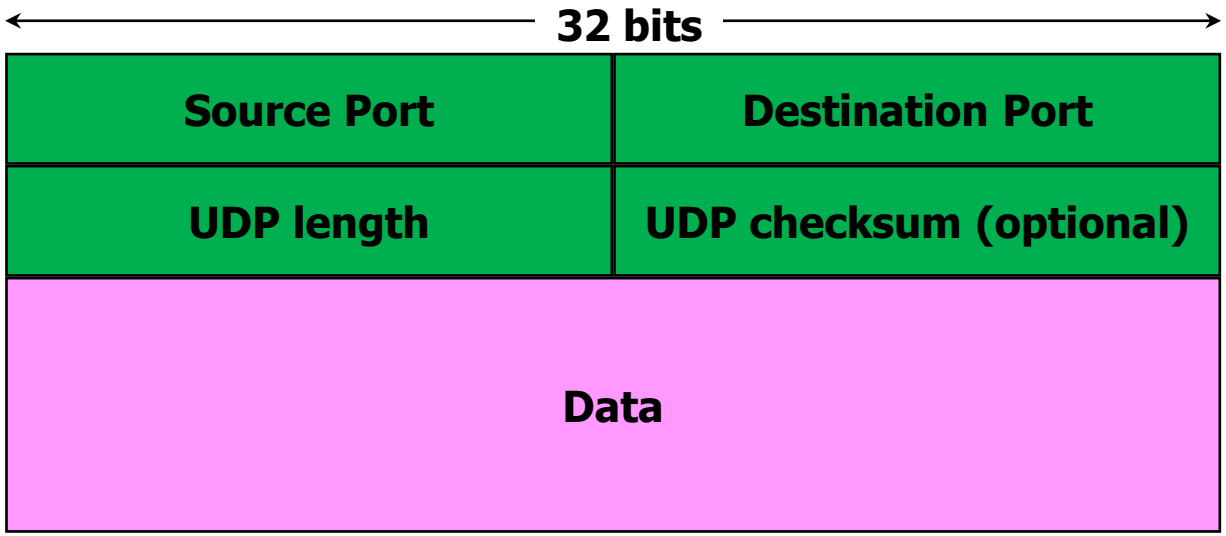
- UDP is a **connectionless** transport protocol.
  - A UDP application sends messages without establishing and then closing a connection.
  - UDP requires a smaller overhead than TCP, especially when the total size of the messages is small.
- UDP **does not guarantee a reliable delivery of data**.
  - UDP messages can be lost or duplicated, or they may arrive out of order. Moreover, UDP messages can arrive faster than the receiver can process them because there is **no flow control mechanism**.
  - Application programmers using UDP have to consider and tackle these issues themselves.

# UDP Basic Characteristics (cont'd)

- The UDP protocol (defined in RFC 768) is extremely simple. Data from the application layer are handed down to the transport layer and encapsulated into a small UDP datagram. The datagram is sent to the host with no mechanisms to guarantee the safe arrival at the destination device. This check is left to the application layer if reliability is needed.

- UDP provides simple functions beyond that of IP, as:
  - **Port Numbers**. UDP uses16-bit port numbers to let multiple processes to use UDP services on the same host.
  - **Checksum**. UDP checksums its data and a pseudo-header in order to verify their integrity. **A packet failing checksum is simply discarded, with no further action taken (i.e., no retransmission is requested).**

# UDP Packet Format



# Thank you!

**giovanni.giambene@gmail.com**