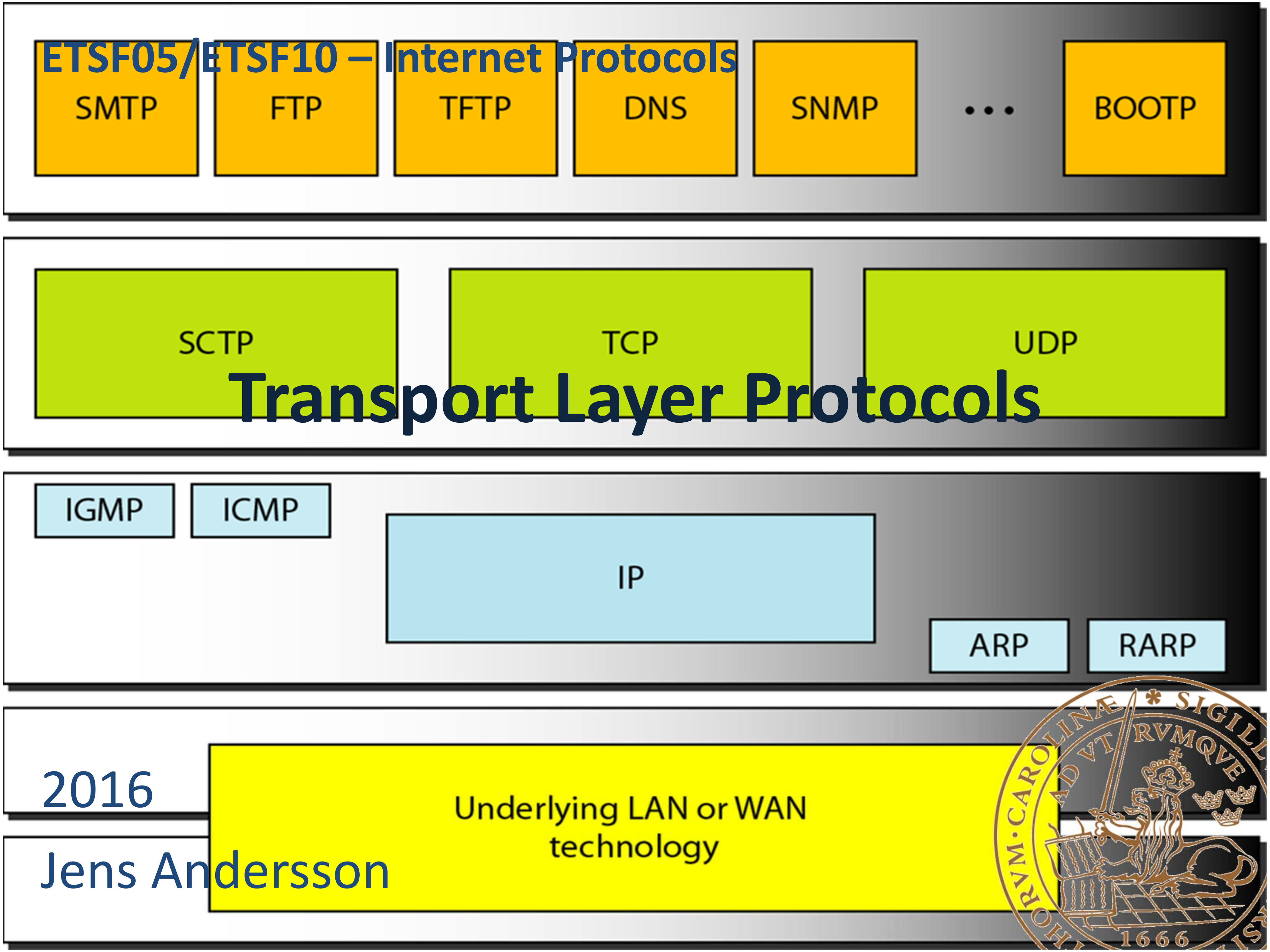
"Hi, I'd like to hear a TCP joke."
"Hello, would you like to hear a TCP joke?"
"Yes, I'd like to hear a TCP joke."
"OK, I'll tell you a TCP joke."
"Ok, I will hear a TCP joke."
"Are you ready to hear a TCP joke?"
"Yes, I am ready to hear a TCP joke."
"Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline."
"Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline."
"I'm sorry, your connection has timed out.
...Hello, would you like to hear a TCP joke?"

ETSF05/ETSF10 – Internet Protocols

Transport Layer Protocols



2016

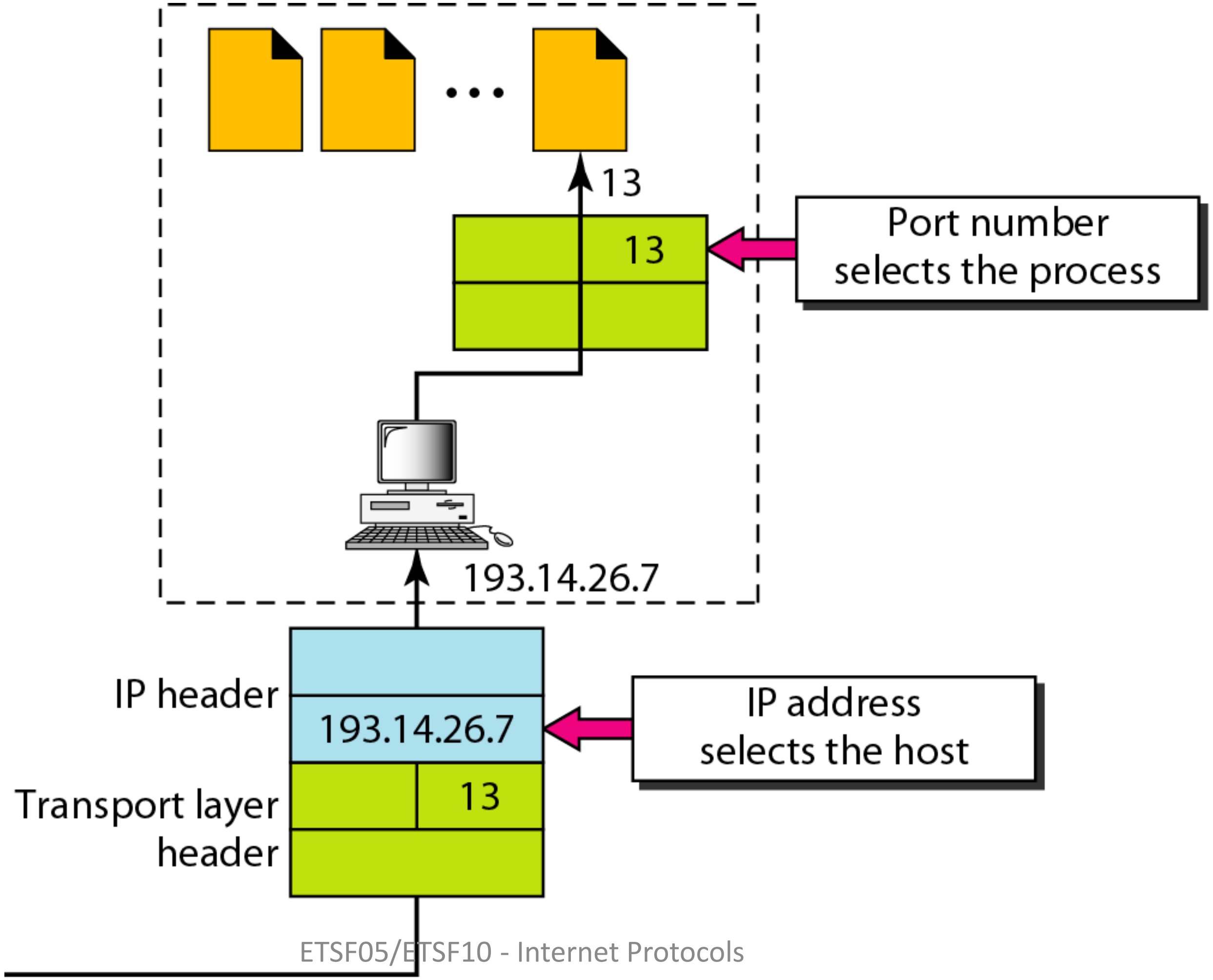
Jens Andersson

Transport Layer

Communication between applications

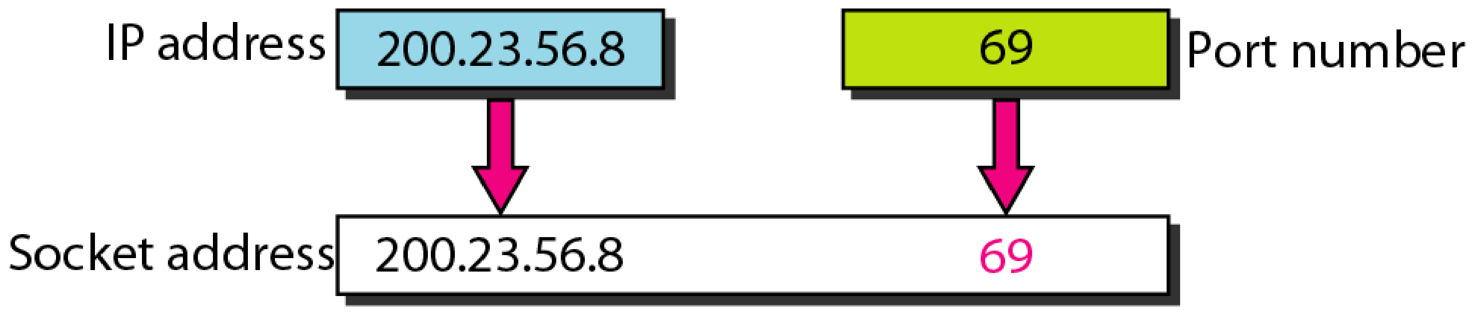
- Process-to-process delivery
- Client/server concept
  - Local host
    - Normally initialiser
  - Remote host
    - Normally always on server

IP addresses and port numbers



Socket addresses

- Combination of IP address & port number
  - Unique for each process on the host



Two Transport Mechanisms

Connectionless or Datagram Service

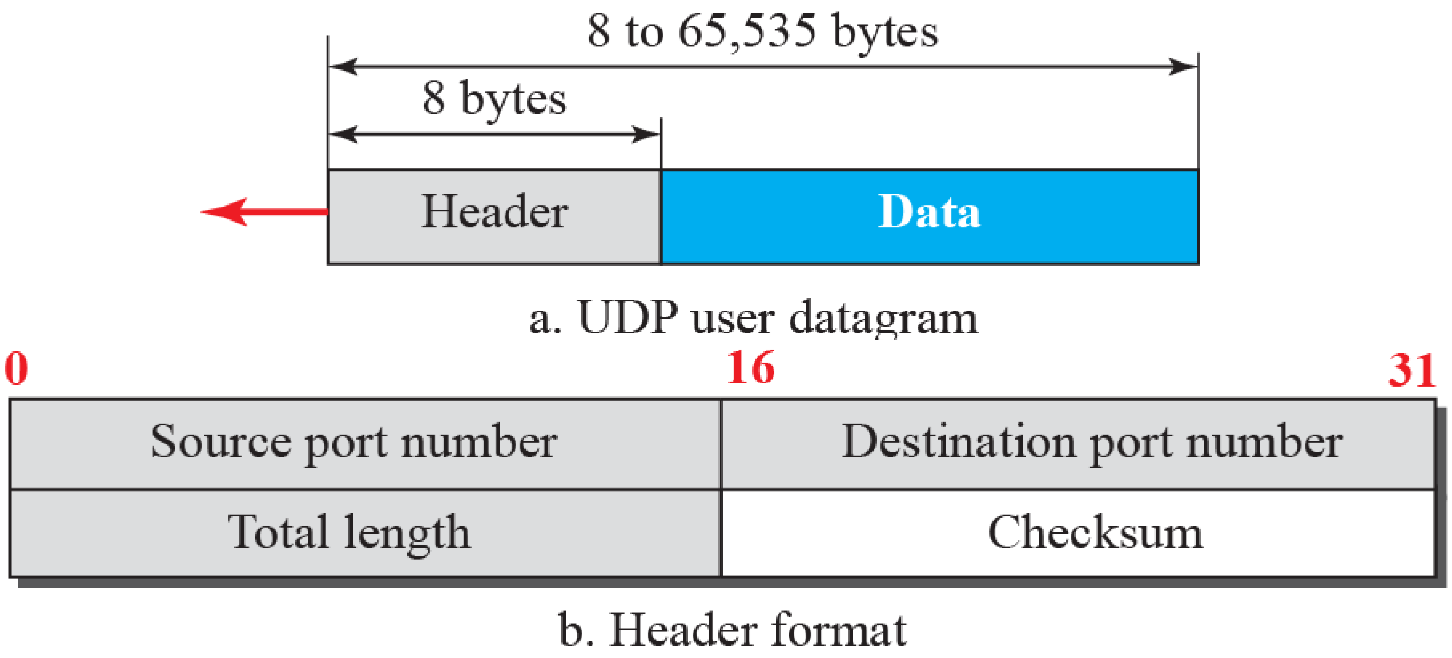
Connection-oriented

- Establishment, maintenance and termination of a logical connection between TS users
- Has a wide variety of applications
- Most common
- Implies service is reliable

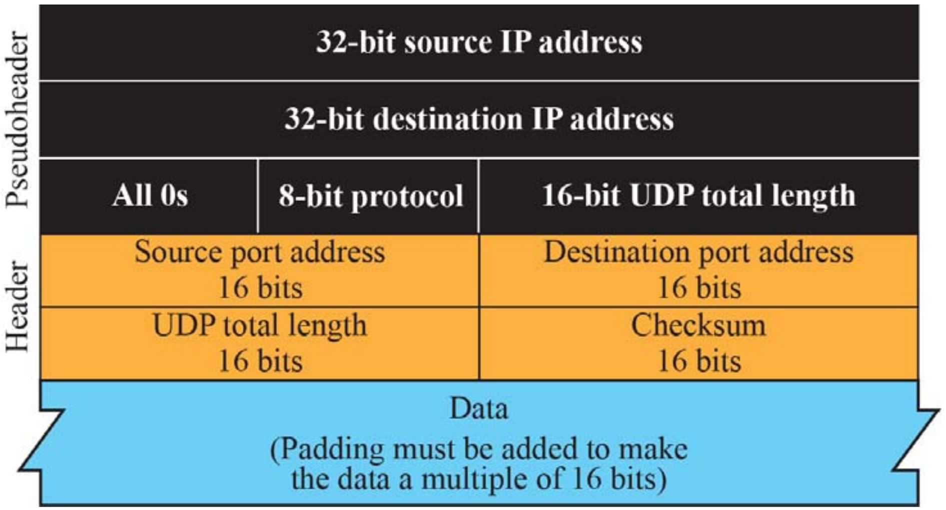
User Datagram Protocol (UDP)

- Transport-level protocol that is commonly used as part of the TCP/IP protocol suite
- RFC 768
- Provides a **connectionless** service for application-level procedures
- Unreliable service; delivery and duplicate protection are not guaranteed
- Reduces overhead and may be adequate in many cases

User Datagram Packet format



IPv4 Pseudoheader for Checksum Calculation



- Optional for IPv4, mandatory for IPv6
- Data not included in IPv6
- Used also for TCP
- Cross Layer!

How to Deal with Unreliable Network Service

Examples:

- Internetwork using IP
- IEEE 802.3 & 802.11 LAN using the unacknowledged connectionless LLC service

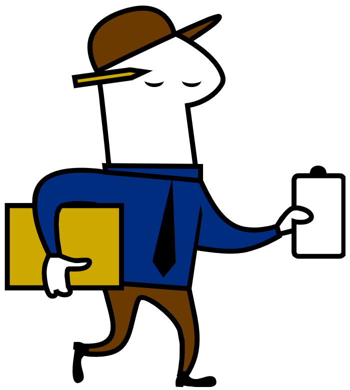
➢ Segments are occasionally **lost** and may arrive **out of sequence** due to variable transit delays

Issues to Address

- Ordered delivery
- Duplicate detection
- Retransmission strategy
- Flow control (sender/receiver)
- Connection establishment
- Connection termination
- Failure recovery

Ordered Delivery

- With an unreliable network service it is possible that segments may arrive out of order
- Solution: number segments sequentially
  - *TCP uses scheme where each data octet is implicitly numbered*



Duplicate Detection

- Receiver must be able to recognize duplicates
- Segment sequence numbers help
- Complications arise if:
  - A duplicate is received prior to the close of the connection
    - Sender must not get confused if it receives multiple acknowledgments to the same segment
    - Sequence number space must be long enough
  - A duplicate is received after the close of the connection

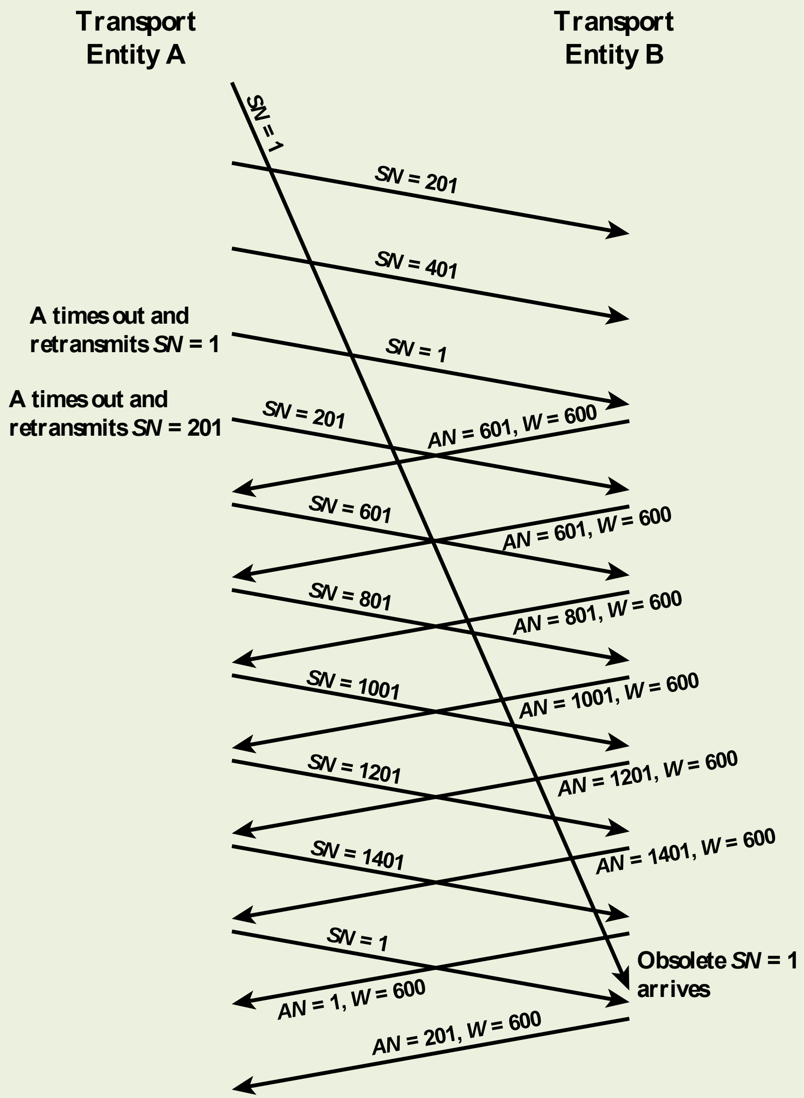


Figure 15.5 Example of Incorrect Duplicate Detection

Max window size and number of bits for sequence number are dependent!

Retransmission Strategy

- Events necessitating retransmission:

Segment may be damaged in transit but still arrives at its destination	Segment fails to arrive

- Sending entity does not know transmission was unsuccessful
- Receiver acknowledges successful receipt by returning a segment containing an acknowledgment number

Cont.

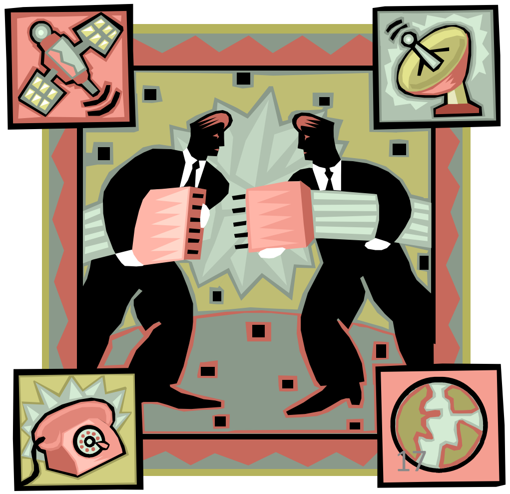
Retransmission Strategy

- No acknowledgment if a segment does not arrive successfully
- A timer needs to be associated with each segment as it is sent
- If timer expires before acknowledgment is received, sender must **retransmit**
- *See Table 15.1 for Transport Protocol Timers*

Connection Establishment and Termination

- Serves three main purposes:
  - Allows each end to assure that the other exists
  - Allows exchange or negotiation of optional parameters
  - Triggers allocation of transport entity resources

➢ Is by mutual agreement



Remarks on Connection Establishment

- Must take into account the unreliability of a network service
- Calls for the exchange of SYNs (two way handshake minimum)
  - Could result in:
    - Duplicate SYNs
    - Duplicate data segments
- *Check Figures 15.4, 15.6—15.9 for details*

Remarks on Connection Termination

- Two-way handshake was found to be inadequate for an unreliable network service
- Out of order segments could cause the FIN segment to arrive before the last data segment
    - To avoid this problem the next sequence number after the last octet of data can be assigned to FIN
    - Each side must explicitly acknowledge the FIN of the other using an ACK with the sequence number of the FIN to be acknowledged

Remarks on Failure Recovery

- When the system that the transport entity is running on fails and subsequently restarts, the state information of all active connections is lost
  - Affected connections become half open because the side that did not fail does not realize the problem
    - Still active side of a half-open connection can close the connection using a ***keepalive*** timer

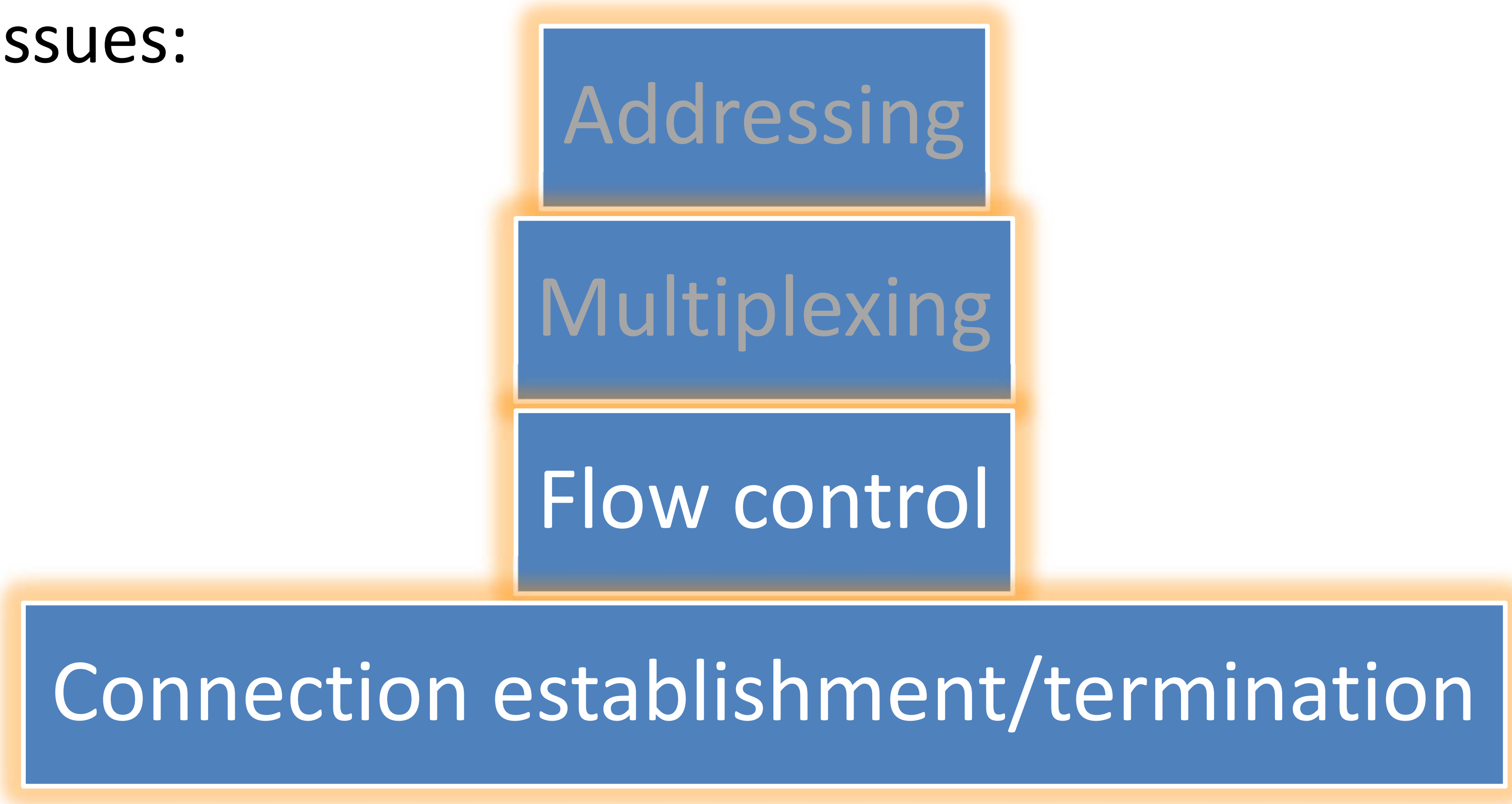
Cont…

Failure Recovery (cont.)

- In the event that a transport entity fails and quickly restarts, half-open connections can be terminated more quickly by the the use of the RST segment (RST = RESET)
  - Failed side returns an RST *i* to every segment *i* that it receives
  - RST *i* must be checked for validity on the other side
    - If valid an abnormal termination occurs
- There is still the chance that some user data will be lost or duplicated

Reliable Sequencing Network Service

Issues:



Flow Control

Reasons for control:

User of the receiving transport entity cannot keep up with the flow	Receiving transport entity itself cannot keep up with the flow of segments

- Complex at the transport layer:
  - Considerable delay in the communication of flow control information
  - Amount of the transmission delay may be highly variable, making it difficult to effectively use a timeout mechanism for retransmission of lost data

Alternatives to Flow Control Requirements

Receiving transport entity can:

Do nothing

- Segments that overflow the buffer are discarded
- Sending transport entity will retransmit

Refuse to accept further segments from the network service

- Relies on network service to do the work (backpressure)

Use a fixed sliding window protocol

- (Window size never changes)
- With a reliable network service this works quite well

Use a credit scheme

- Receiver controls senders window size
- A more effective scheme to use with an unreliable network

Sliding Windows: *Reminder*

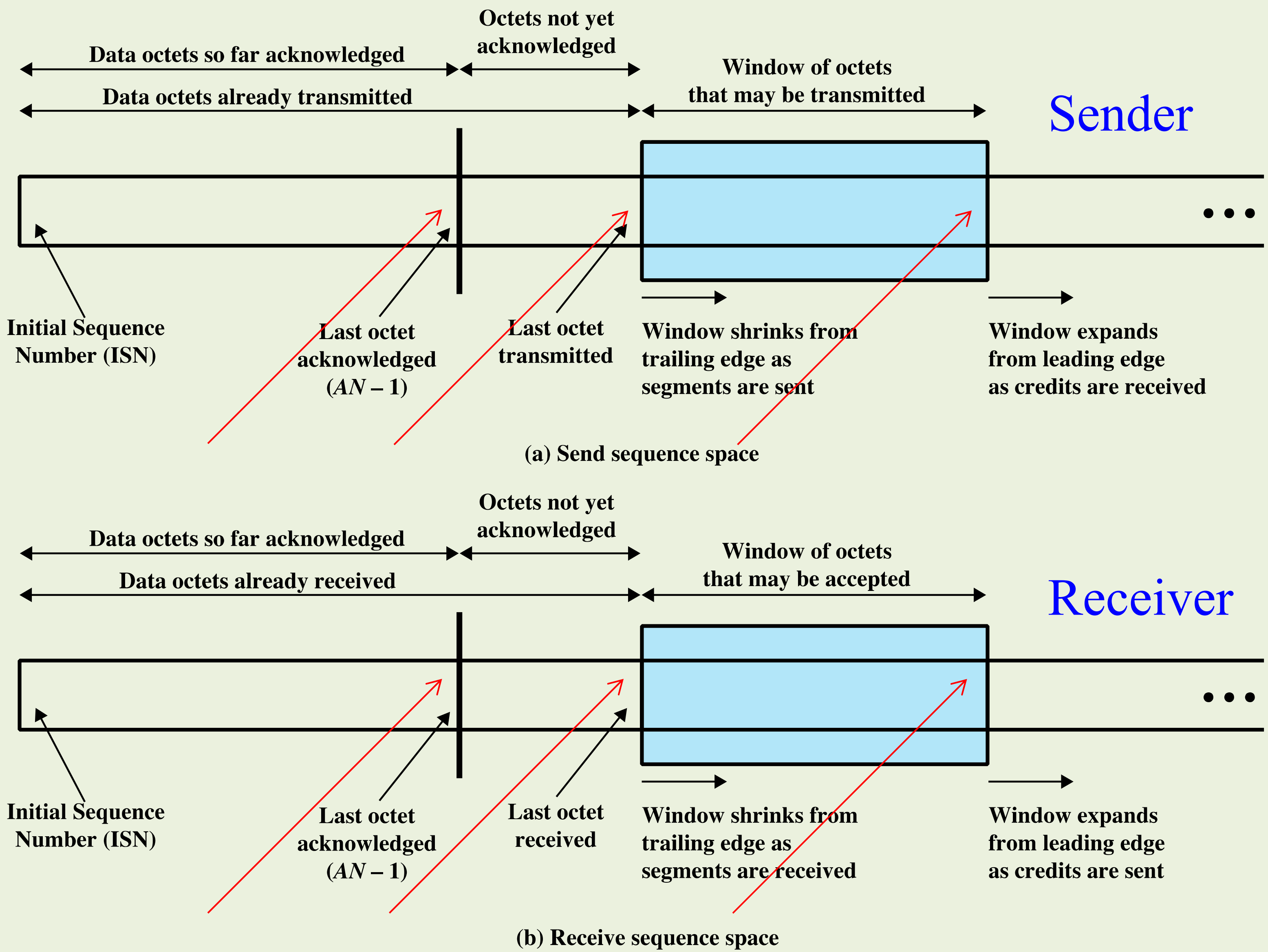


Figure 15.2 Sending and Receiving Flow Control Perspectives

ETSF05/ETSF10 - Internet Protocols 28

Congestion Control in Packet-Switching Networks

Send control packet to some or all source nodes

- Requires additional traffic during congestion

Rely on routing information

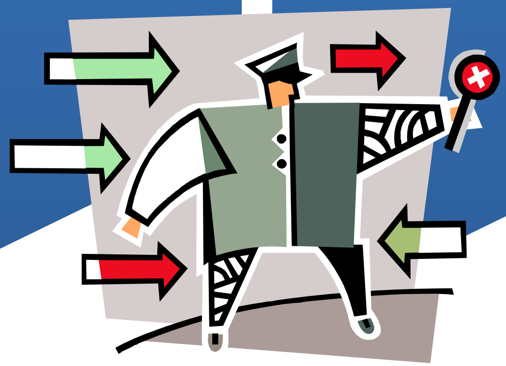
- May vary too quickly

End to end probe packets

- Adds to overhead

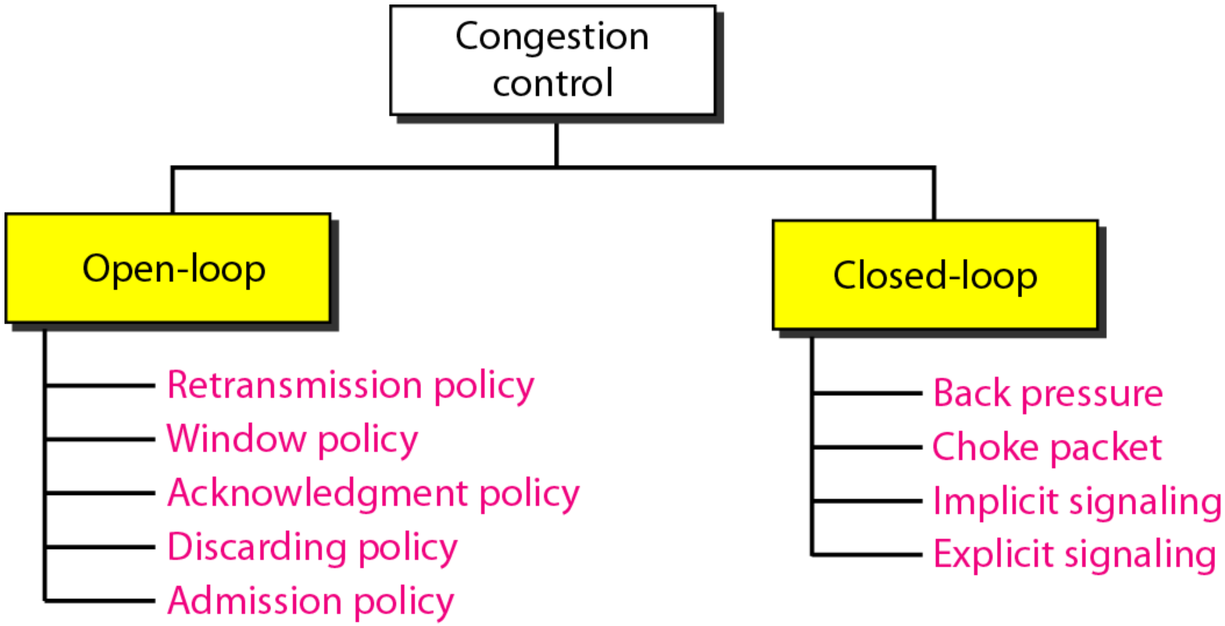
Add congestion information to packets in transit

- Either backwards or forwards



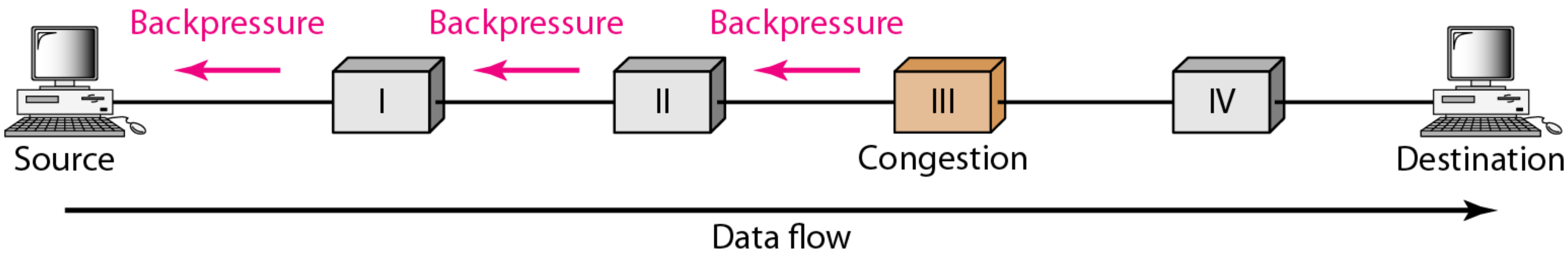
Congestion control methods

- Avoiding and eliminating congestion
  – Open-loop = proactive
  – Closed-loop = reactive



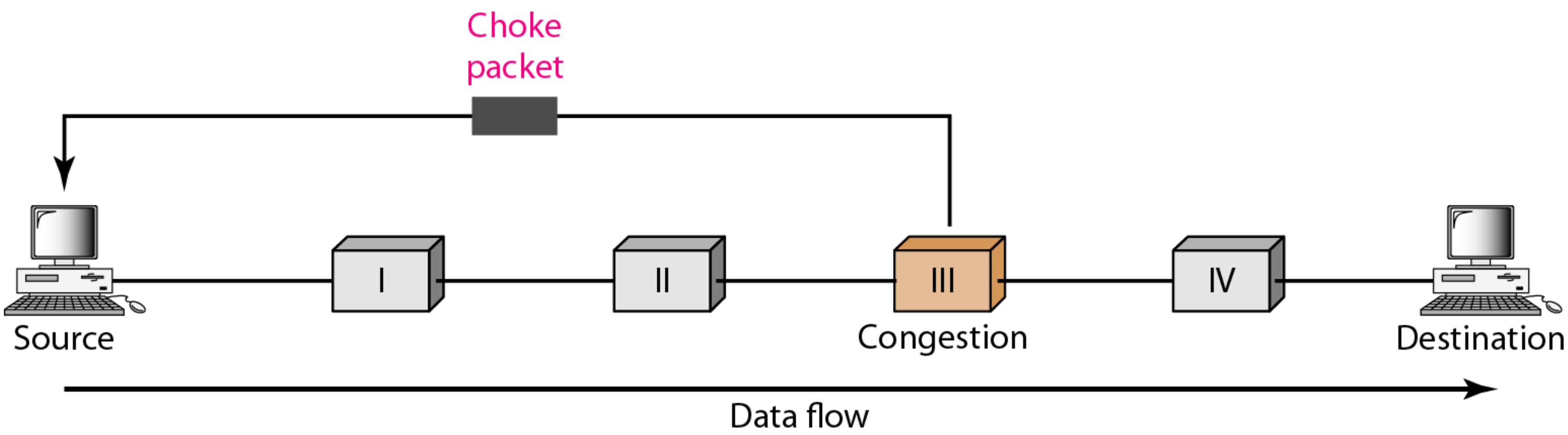
Closed-loop congestion control (1)

- Backpressure



Closed-loop congestion control (2)

- Choke packet



TCP Services

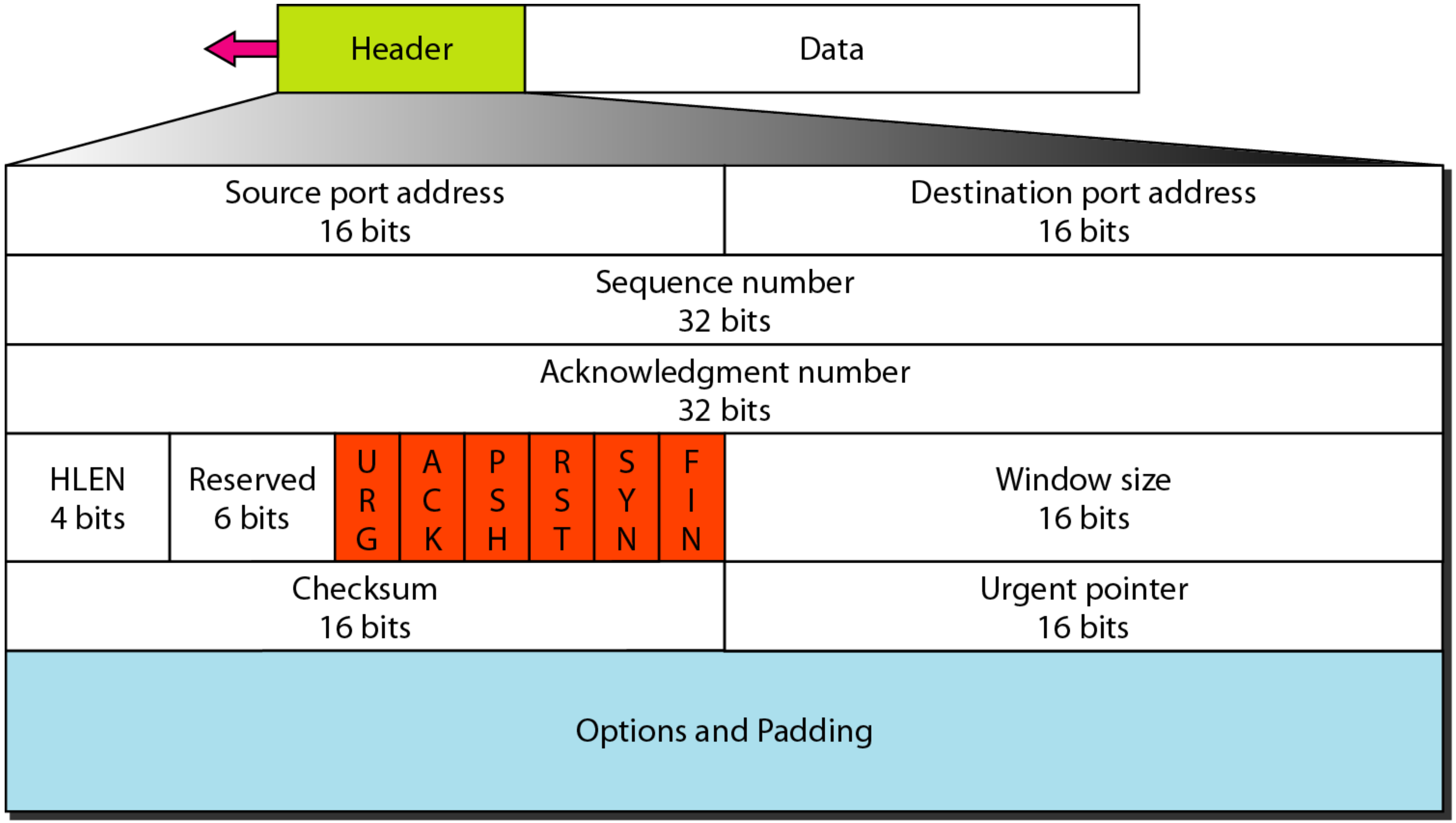
- RFC 793

TCP labels data as:

- Data stream Push
- Urgent data signaling

- Defined in terms of primitives and parameters (see Tables 15.2, 15.3 & 15.4 for details)

TCP header format



TCP Mechanisms

- Can be grouped into:

Connection establishment

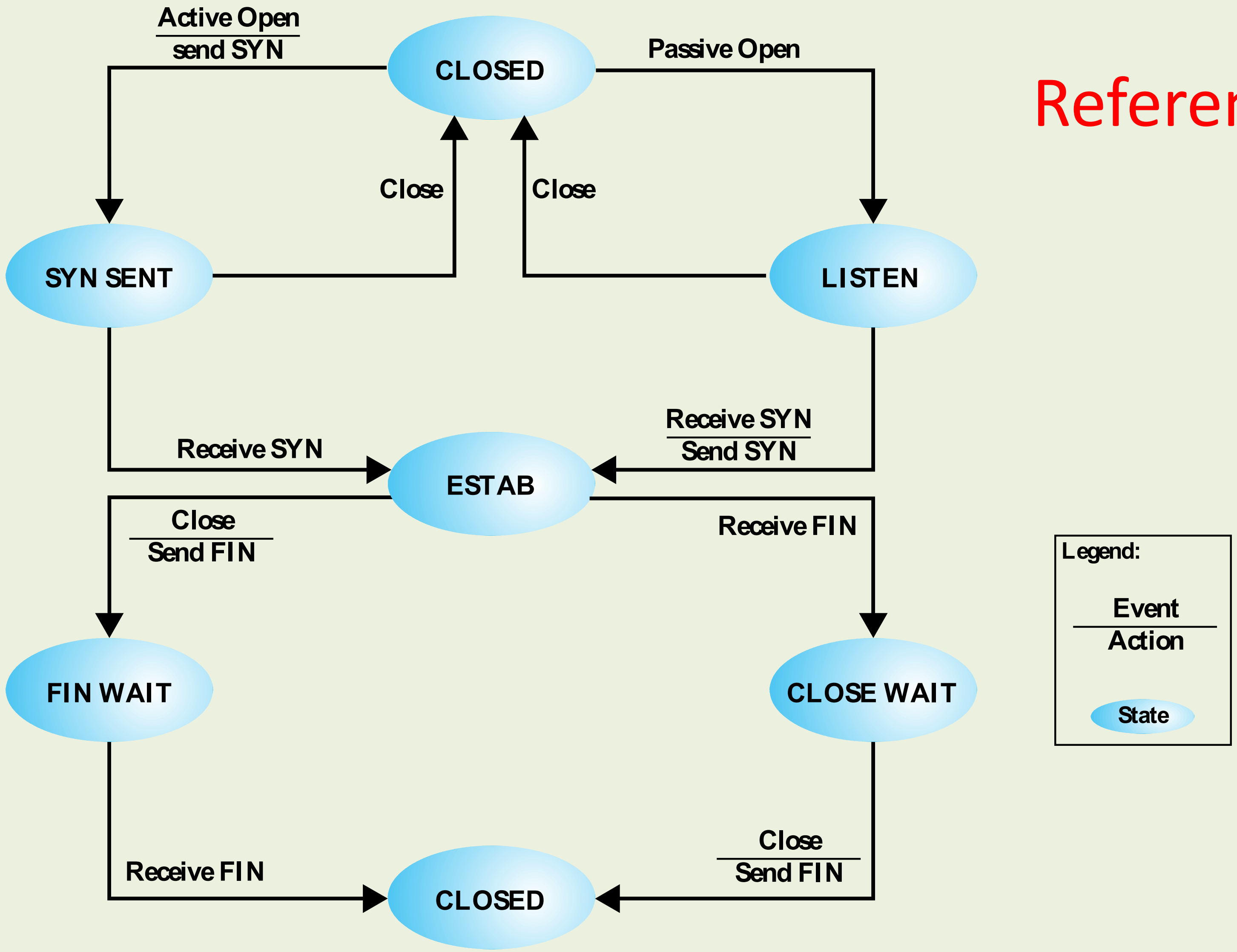
- Always uses a three-way handshake
- Connection is determined by host and port

Data transfer

- Viewed logically as consisting of a stream of octets
- Flow control is exercised using credit allocation (Received ACKs open the sender window)

Connection termination

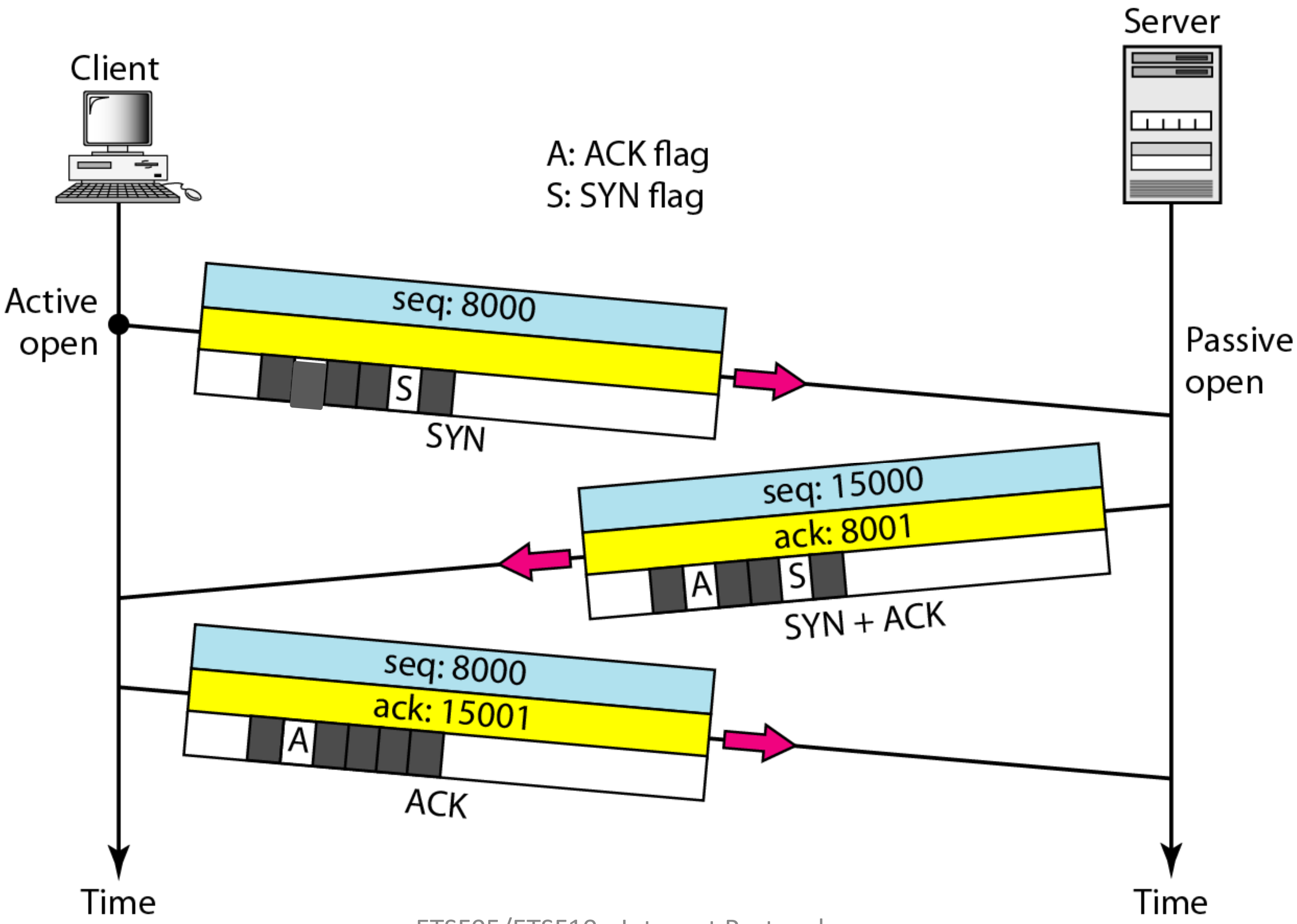
- Each TCP user must issue a CLOSE primitive
- An abrupt termination occurs if the user issues an ABORT primitive



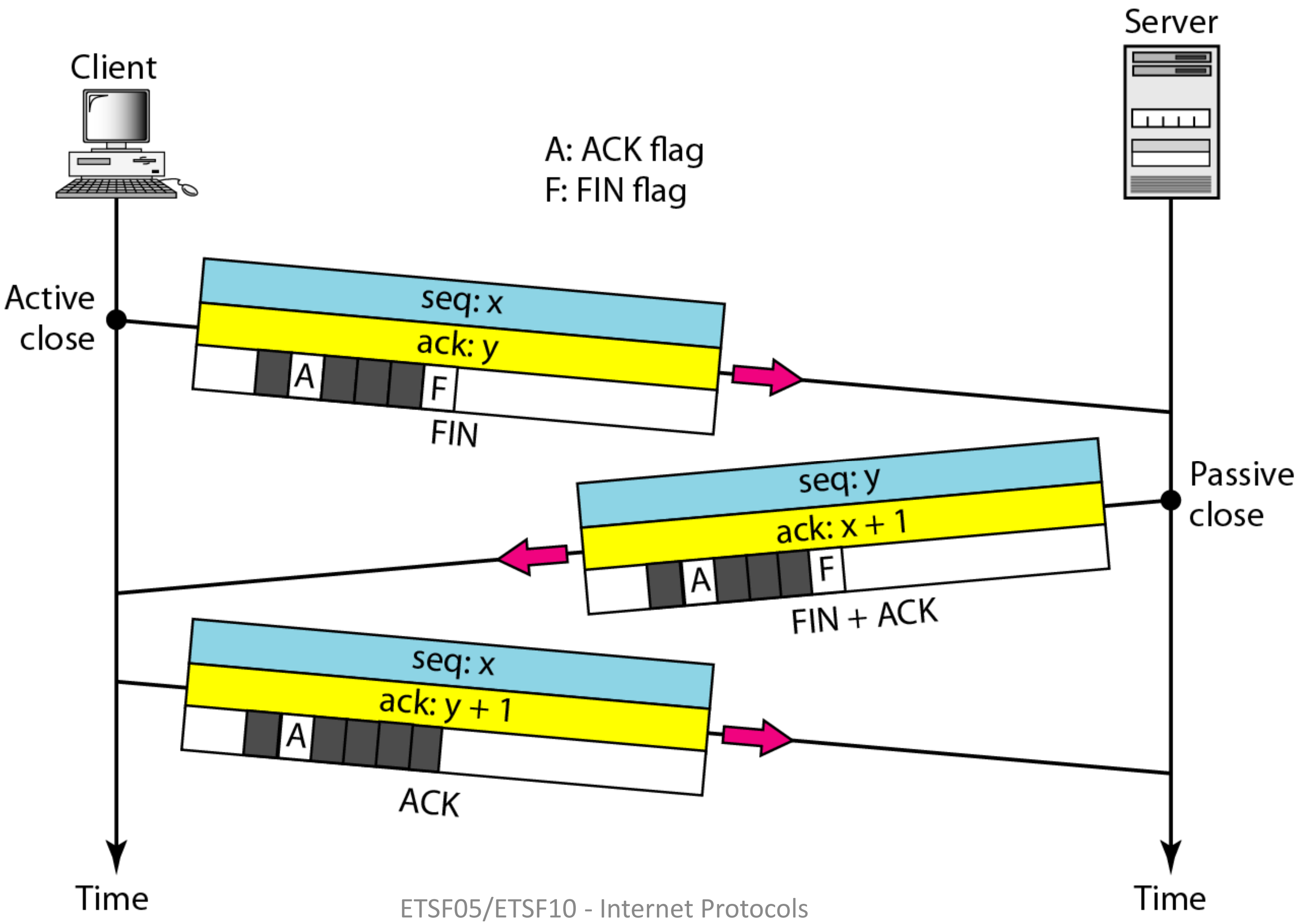
Reference

Figure 15.3 Simple Connection State Diagram

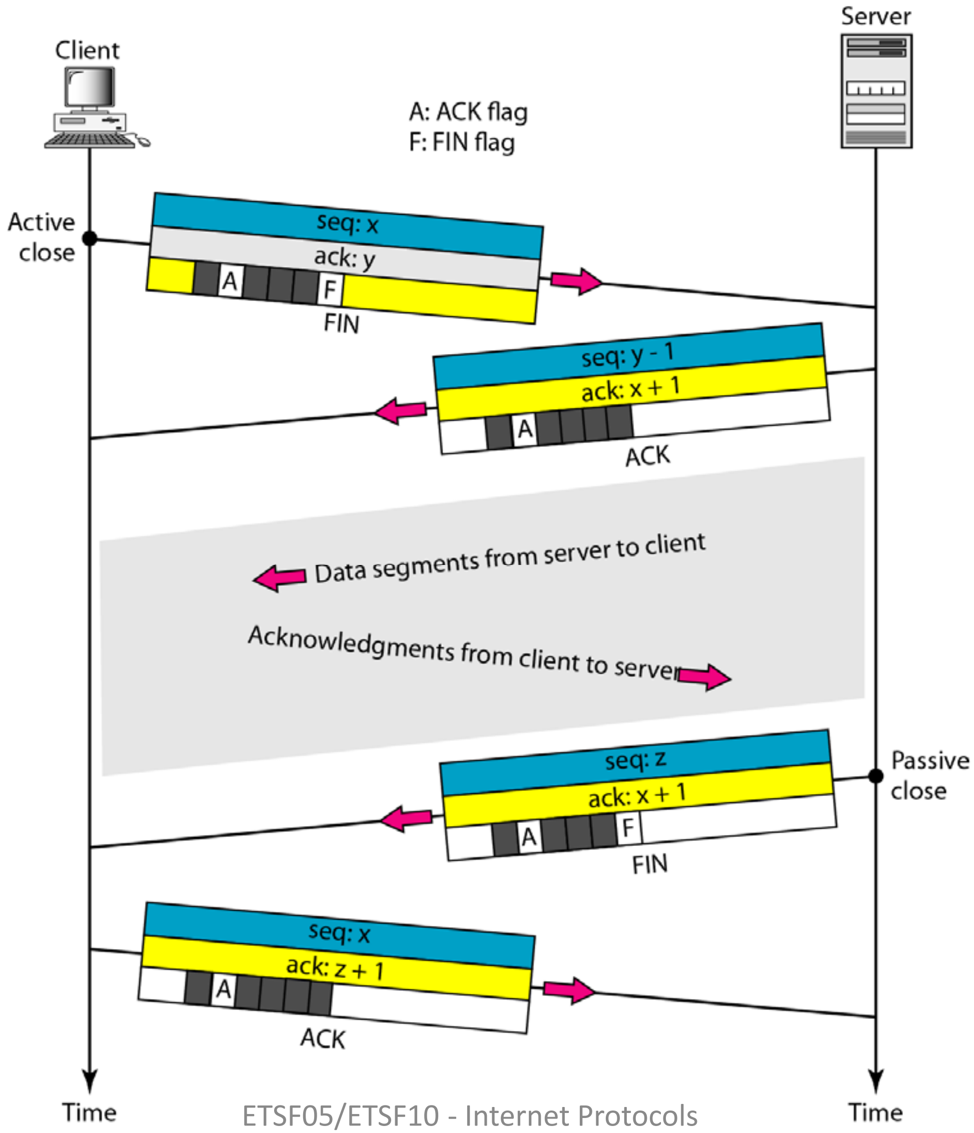
TCP Three Way Handshake



TCP Connection termination



TCP Half-close



TCP **Flow** Control

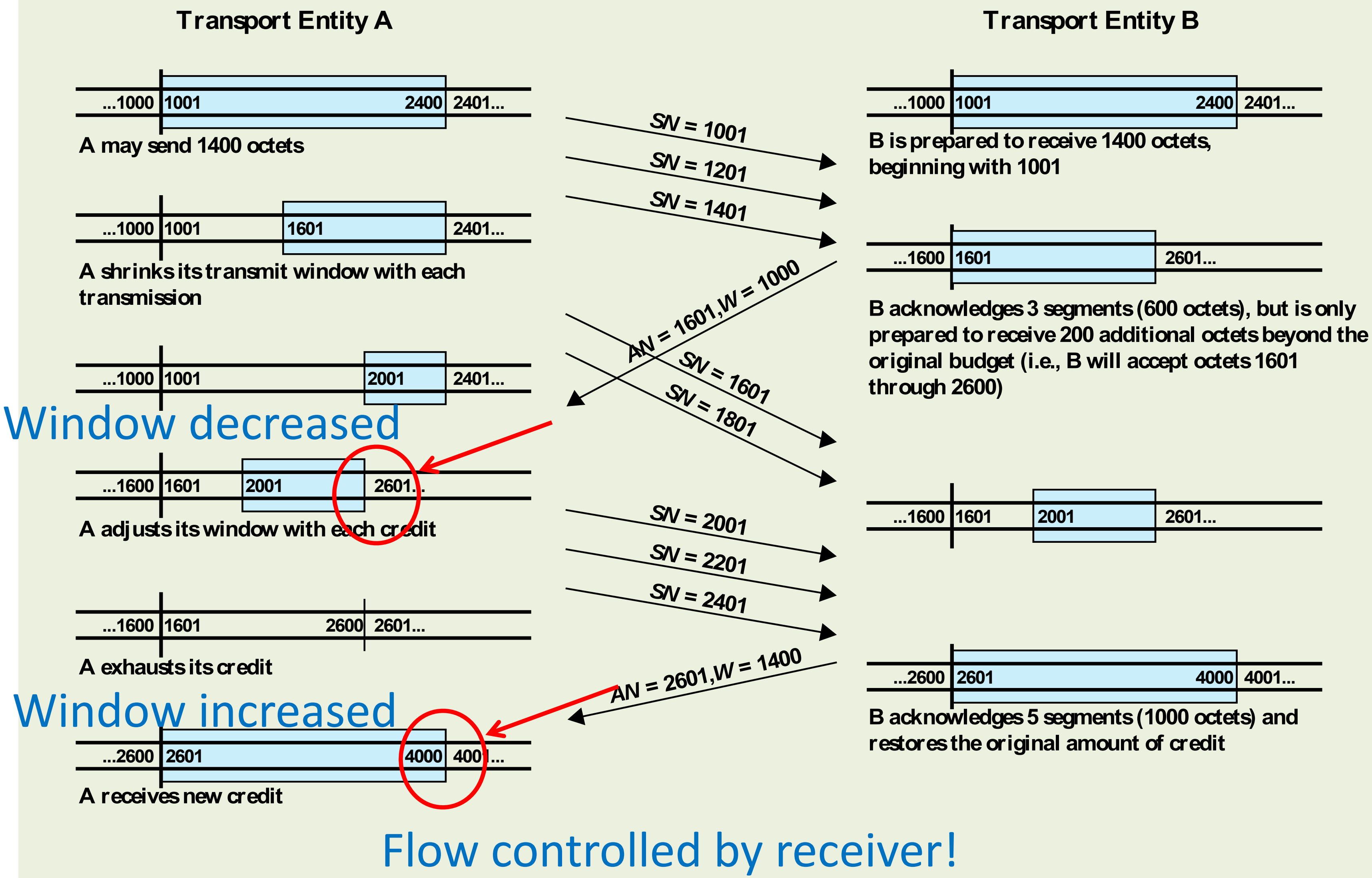


Figure 15.1 Example of TCP Credit Allocation Mechanism

TCP Implementation Policy Options

- Implementation opportunities:

Send policy

Deliver policy

Accept policy

Retransmit policy

Acknowledge policy

Send Policy

- In the absence of both pushed data and a closed transmission window a sending TCP entity is free to transmit data at its own convenience
- TCP may construct a segment for each batch of data provided or it may wait until a certain amount of data accumulates before constructing and sending a segment
- Infrequent and large transmissions have low overhead in terms of segment generation and processing
- If transmissions are frequent and small, the system is providing quick response

Deliver Policy

- In the absence of a Push, a receiving TCP entity is free to deliver data to the user at its own convenience
- May deliver as each in-order segment is received, or may buffer data before delivery
- *If deliveries are infrequent and large, the user is not receiving data as promptly as may be desirable*
- *If deliveries are frequent and small, there may be unnecessary processing, as well as operating system interrupts*

Accept Policy

- If segments arrive out of order the receiving TCP entity has two options:

In-order

- Accepts only segments that arrive in order; any segment that arrives out of order is discarded
- Makes for simple implementation but places a burden on the networking facility
- If a single segment is lost in transit, then all subsequent segments must be retransmitted

In-window

- Accepts all segments that are within the receive window
- Requires a more complex acceptance test and a more sophisticated data storage scheme

Retransmit Policy

- Retransmission strategies:

Retransmit First-only	• Maintain **one retransmission timer for entire queue** • Efficient in terms of traffic generated • Can have **considerable delays**
Retransmit Batch/All	• Maintain **one retransmission timer for entire queue** • Reduces the likelihood of long delays • May result in unnecessary retransmissions
Retransmit Individual	• Maintain **one timer for each segment in the queue** • More complex implementation

Acknowledge Policy

- Timing of acknowledgment:

Immediate	Cumulative
• **Immediately transmit ACK (empty segment containing the appropriate acknowledgement number)** • Simple and keeps the remote TCP fully informed • Limits unnecessary retransmissions • Results in extra segment transmissions • Can cause a further load on the network	• Wait for an outbound segment with data on which to **piggyback the acknowledgement** • Typically used • Requires more processing at the receiving end and complicates the task of estimating round-trip time

TCP **Congestion** Control

- Parallel to but separate from Flow Control
- Congestion window
  - Sliding window (byte-oriented)
  - Variable size
  - Hybrid impl. (Go-back-N & Selective repeat)
- Slow start (state)
- Congestion avoidance (state)
- Congestion detection (event to act upon)

Table 20.1
Implementation of TCP Congestion Control Measures

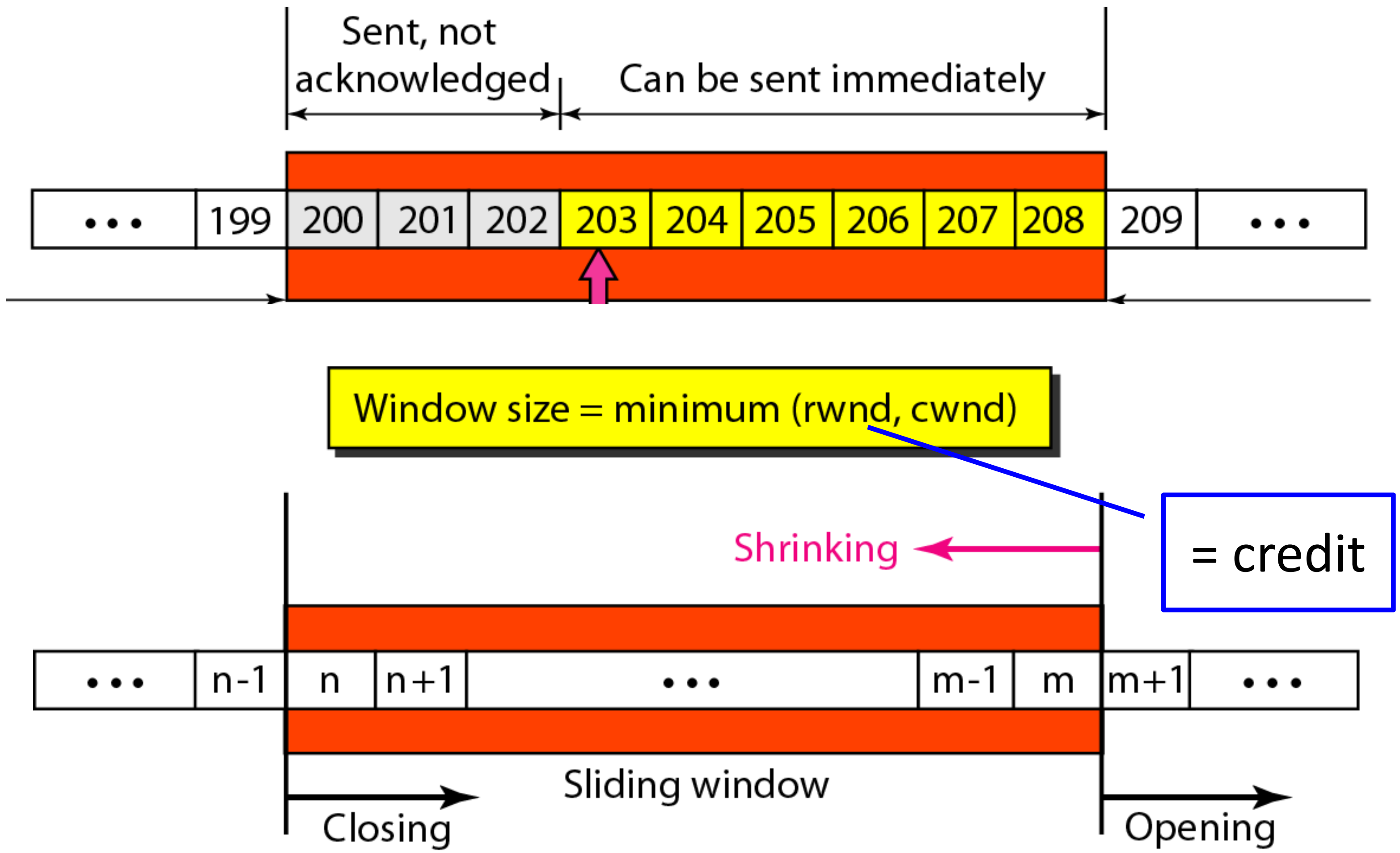
Measure	RFC 1122	TCP Tahoe	TCP Reno	NewReno
RTT Variance Estimation	✓	✓	✓	✓
Exponential RTO Backoff	✓	✓	✓	✓
Karn's Algorithm	✓	✓	✓	✓
Slow Start	✓	✓	✓	✓
Dynamic Window Sizing on Congestion	✓	✓	✓	✓
Fast Retransmit		✓	✓	✓
Fast Recovery			✓	✓
Modified Fast Recovery				✓

Retransmission Timer Management

Essential! Virtually all TCP implementations estimates RTT and sets timer to a somewhat higher value.

- Static RTT
  - Cannot adapt to network conditions
- Simple average RTT
  - Over a number of segments
  - Works well if average is a good predictor
- Exponential average RTT
  - predicting the next value on the basis of a time series of past values (RFC 793)

Congestion window



Window Management

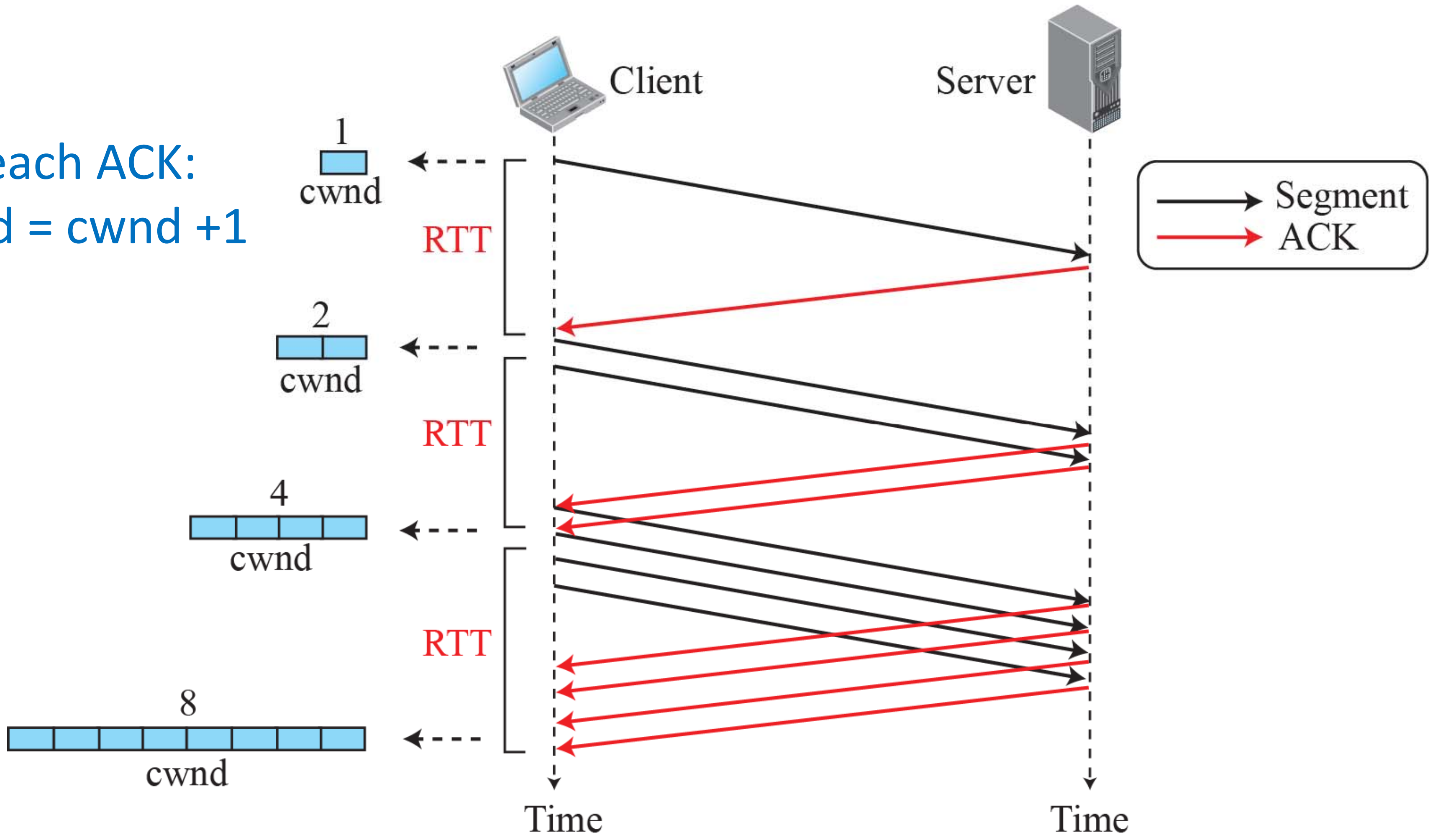
- The size of TCP's send window can have a critical effect on whether TCP can be used efficiently without causing congestion
- Two techniques found in virtually all modern implementation of TCP are:
  - **Slow start**
  - **Dynamic window sizing on congestion**
- Combined with flow control (credit)

  $$awnd = \mathrm{MIN}[\boldsymbol{rwnd},\ \boldsymbol{cwnd}]$$

  - **Credit = rwnd**

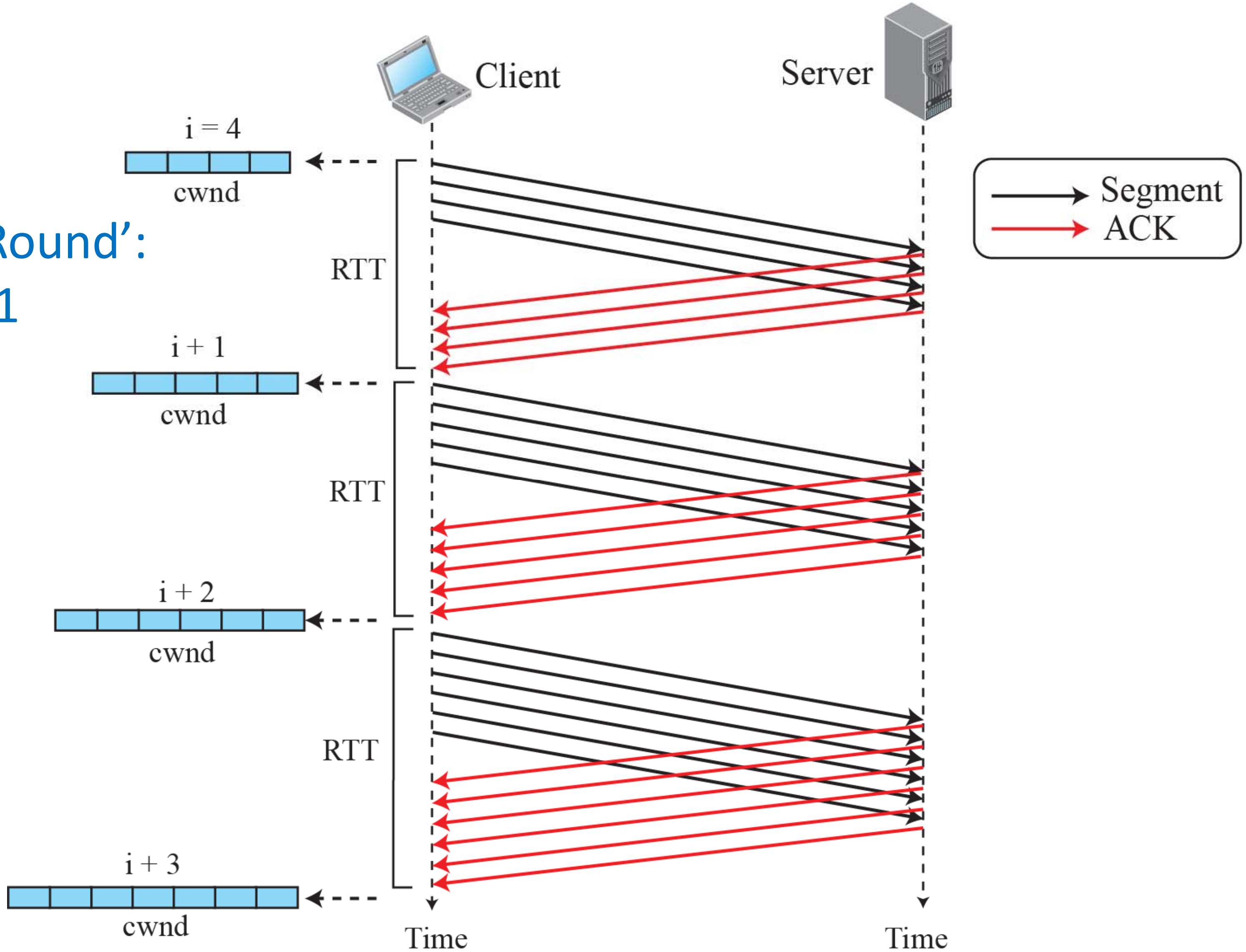
Slow start: *Exponential increase*

For each ACK:
cwnd = cwnd +1



Congestion avoidance: *Additive increase*

For each RTT/'Round':
cwnd = cwnd +1

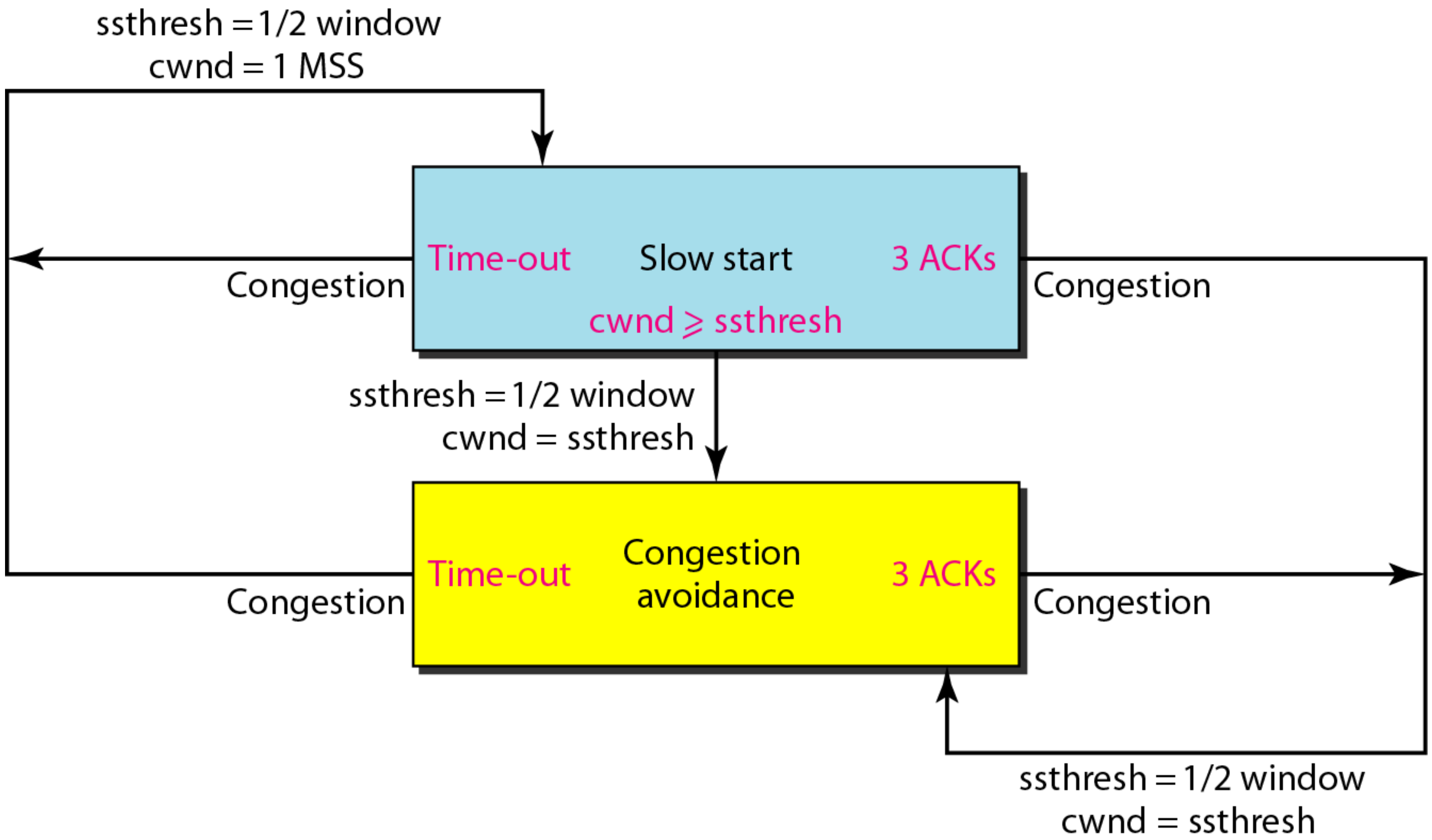


Reaction to **Congestion Detection**

- Detection by time-out (RTO)
  - Probably both channels congested
  - New slow start phase
- Detection by three ACK of same segment
  - Indicates lost segment
    (= hole in segment sequence)
  - Probably sending channel congested only
  - New congestion avoidance phase

RTO Retransmission Timer Overflow

TCP congestion policy: *Summary*



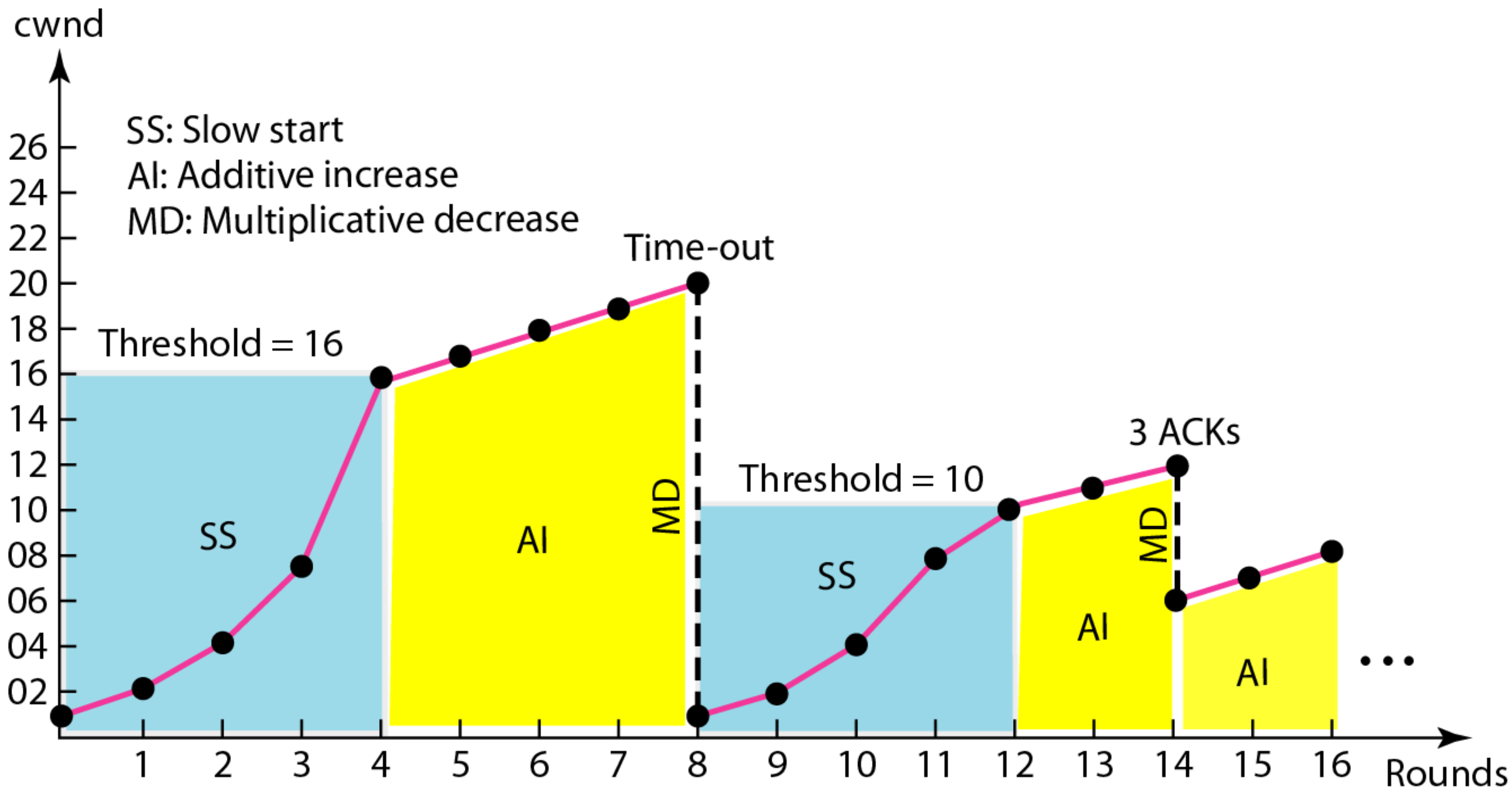
Fast Retransmit

- After three duplicate ACKs
- Under some circumstances improve on the performance provided by RTO
- Takes advantage of the rule that if a TCP entity receives a segment out of order, it must immediately issue an ACK for the last in-order segment that was received

Fast Recover

- Retransmit the lost segment, cut *cwnd* in half, and then proceed with the linear increase of *cwnd*
- RFC 3782 modifies the fast recovery algorithm to improve the response when two segments are lost within a single window

TCP congestion policy: *Example*



Random Early Discard (RED)

- Buffers in routers can detect congestion (buffer overflow)
- Buffer overflow impact on TCP:
  - Problably RTO
  - All TCP connections affected
  - Will return to slowstart; synkronised
- RED: Start discarding packets randomly before buffer overflow
  - Single pkt loss = Fast Restransmitt,
  - cwnd = cwnd/2
- Compare with ECN flag in IP header

TCP operation: *Summary*

- Connection establishment
  - Three-way handshake
- Data transfer
  - Flow control (→ congestion control)
  - Error control
- Connection termination
  - Three-way handshake
  - Half-close