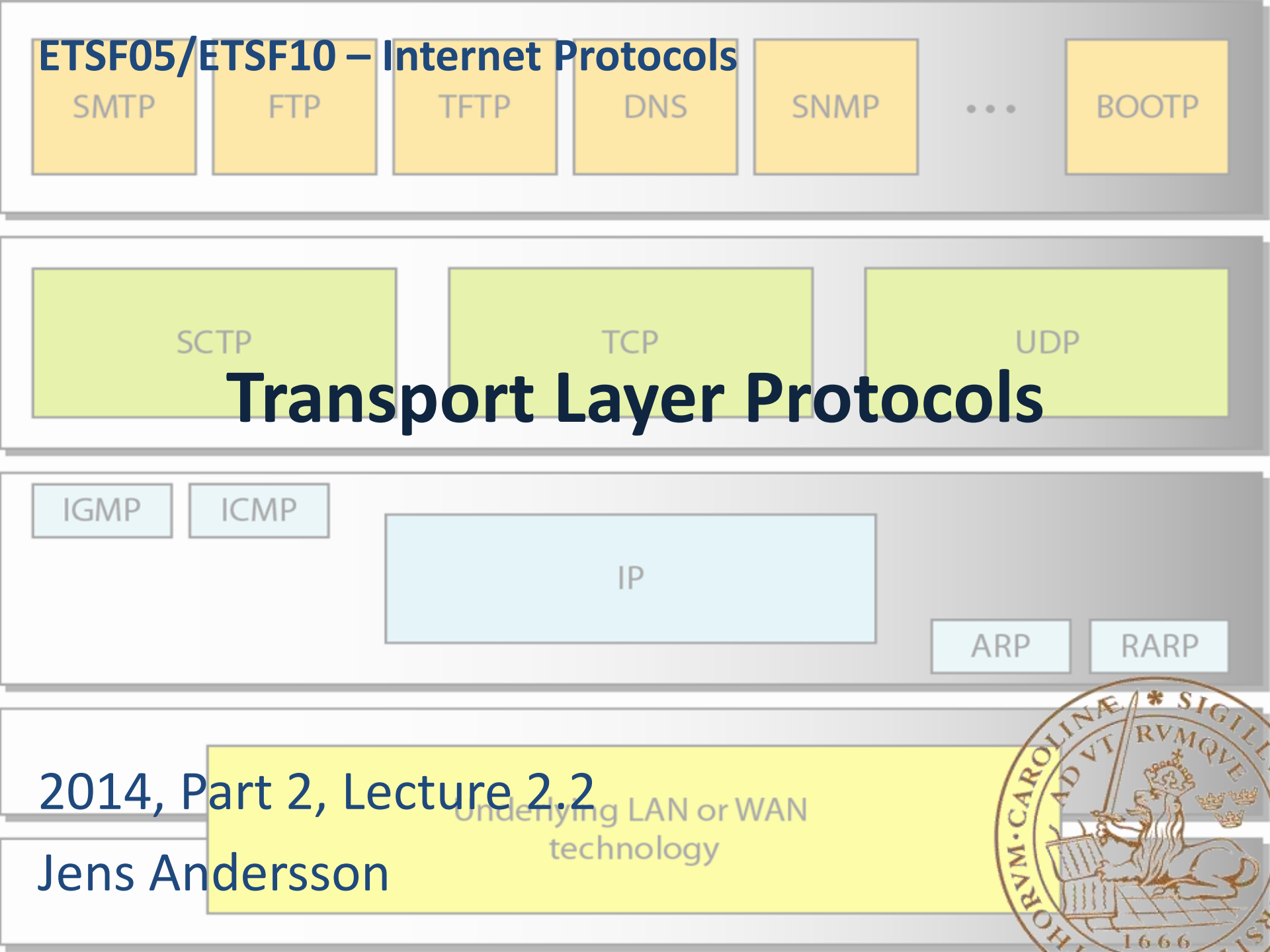
ETSF05/ETSF10 – Internet Protocols



Transport Layer Protocols

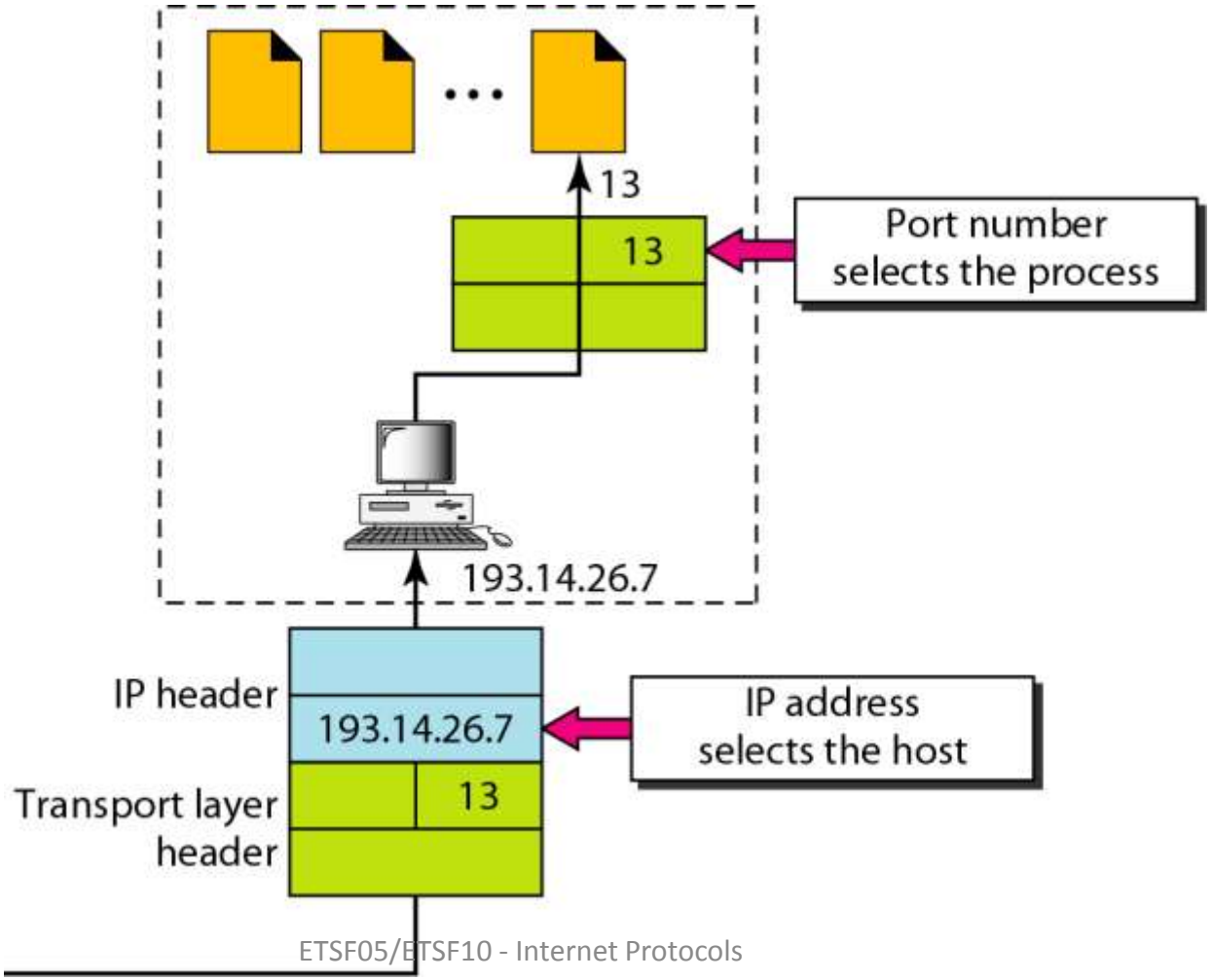
2014, Part 2, Lecture 2.2

Jens Andersson

Transport Layer

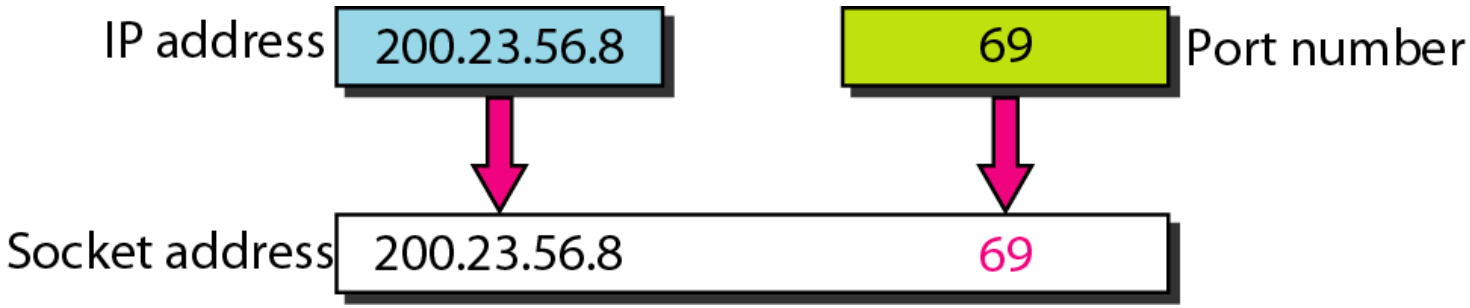
- Communication between applications
- Process-to-process delivery
- Client/server concept
  - Local host
  - Remote host

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

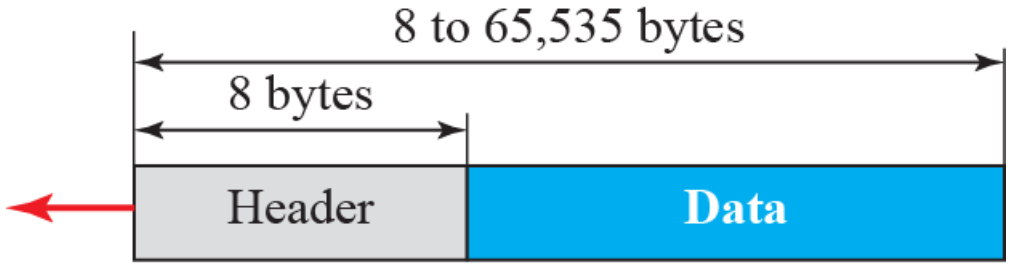
Connection-oriented

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

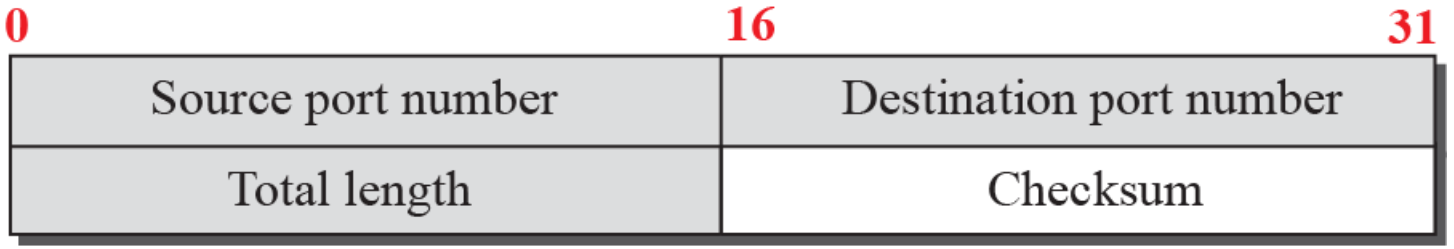
User Datagram Protocol (UDP)

- 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

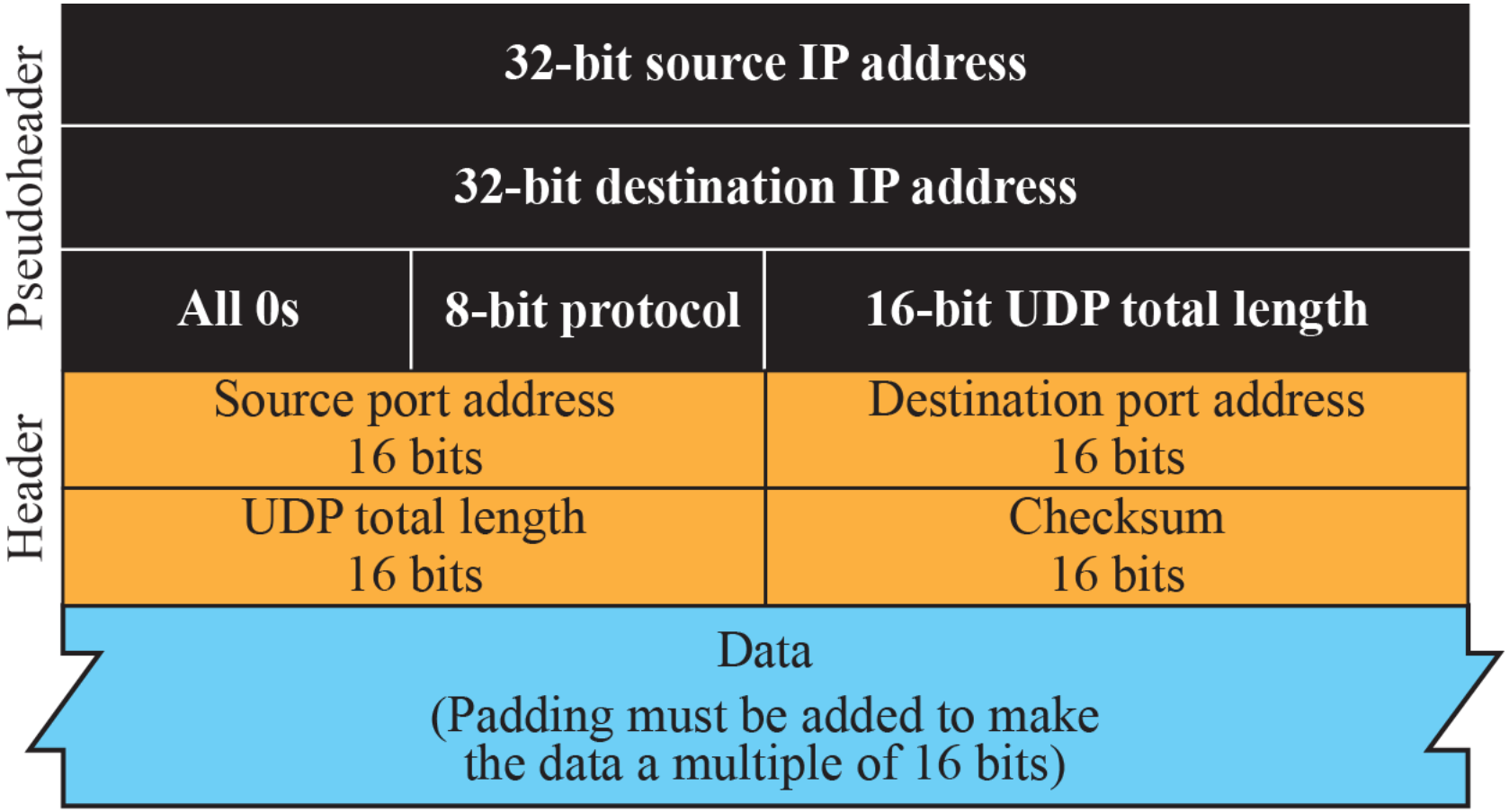


a. UDP user datagram



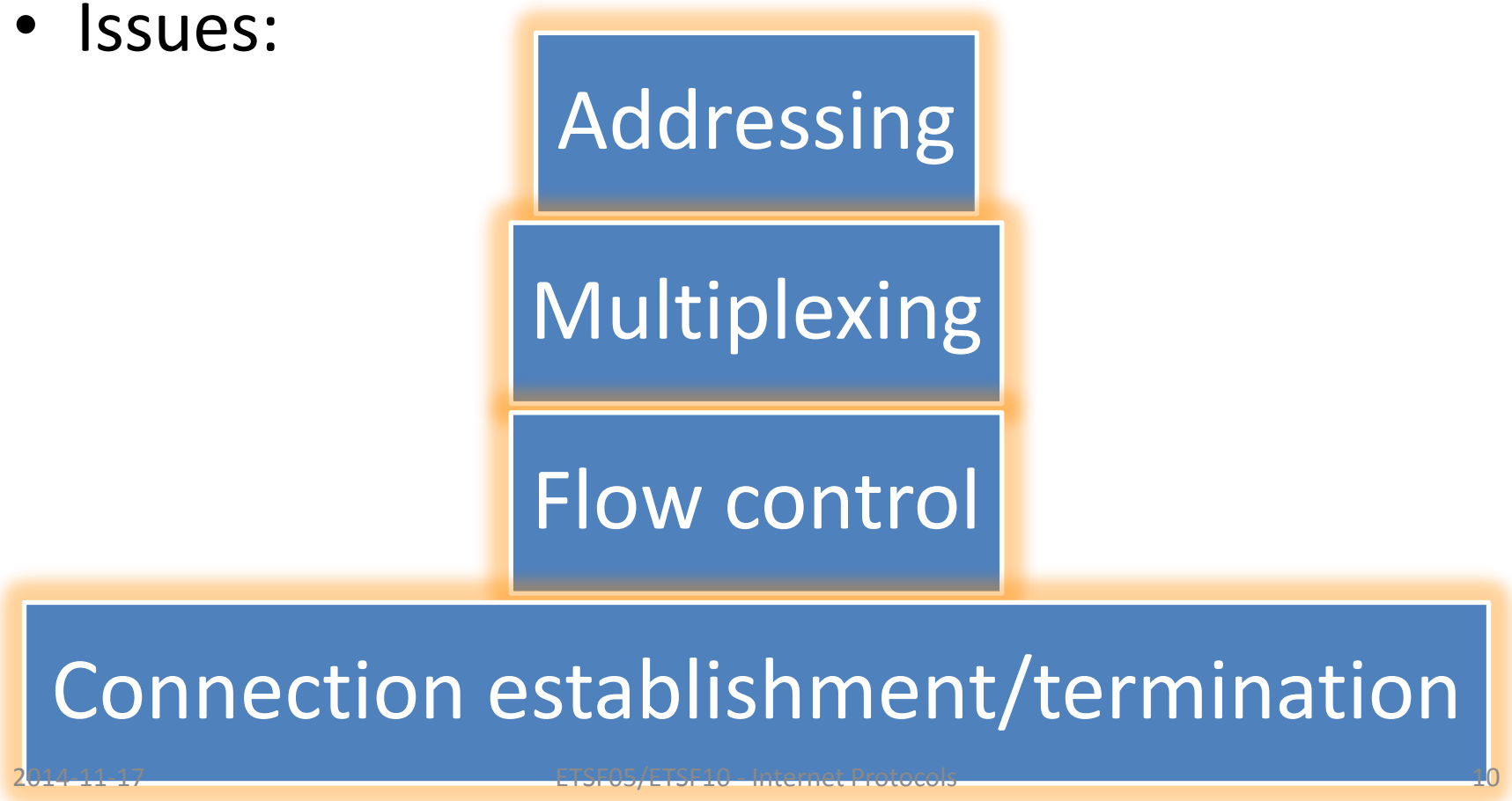
b. Header format

Pseudoheader for checksum calculation



Reliable Sequencing Network Service

- Issues:



Flow Control

- 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

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

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

Use a fixed sliding window protocol

- With a reliable network service this works quite well

Use a credit scheme

- A more effective scheme to use with an unreliable network service
- Compare with ACK

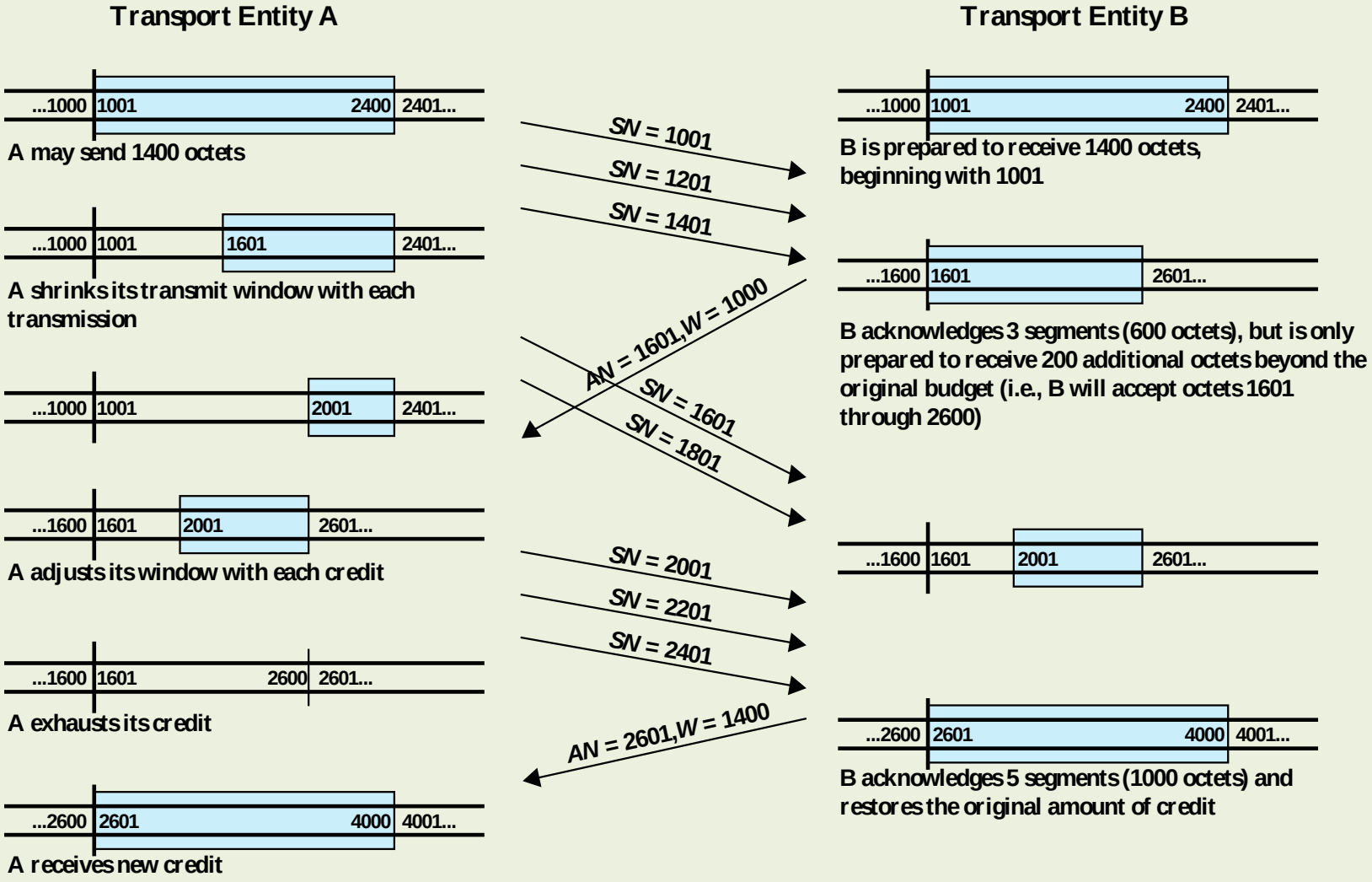


Figure 15.1 Example of TCP Credit Allocation Mechanism

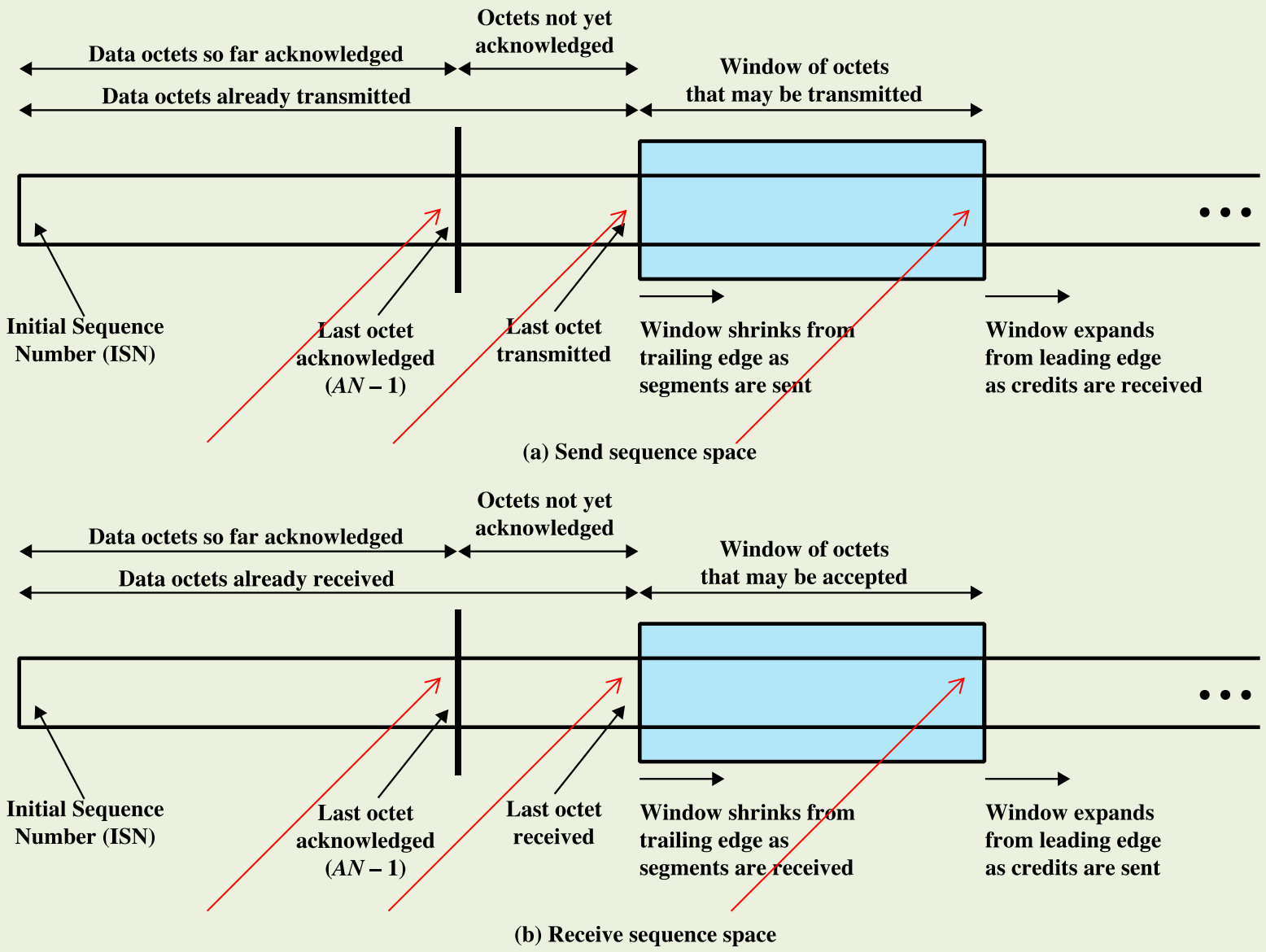
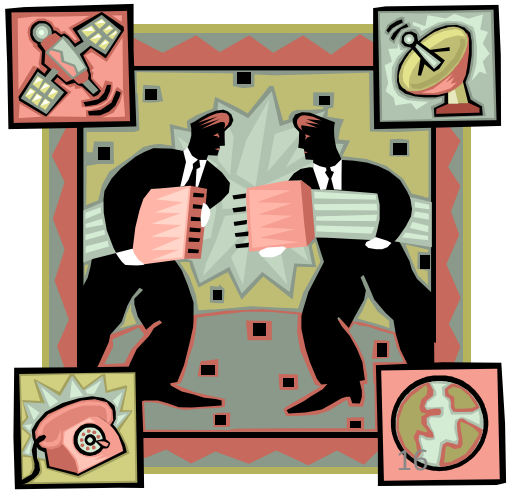


Figure 15.2 Sending and Receiving Flow Control Perspectives

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



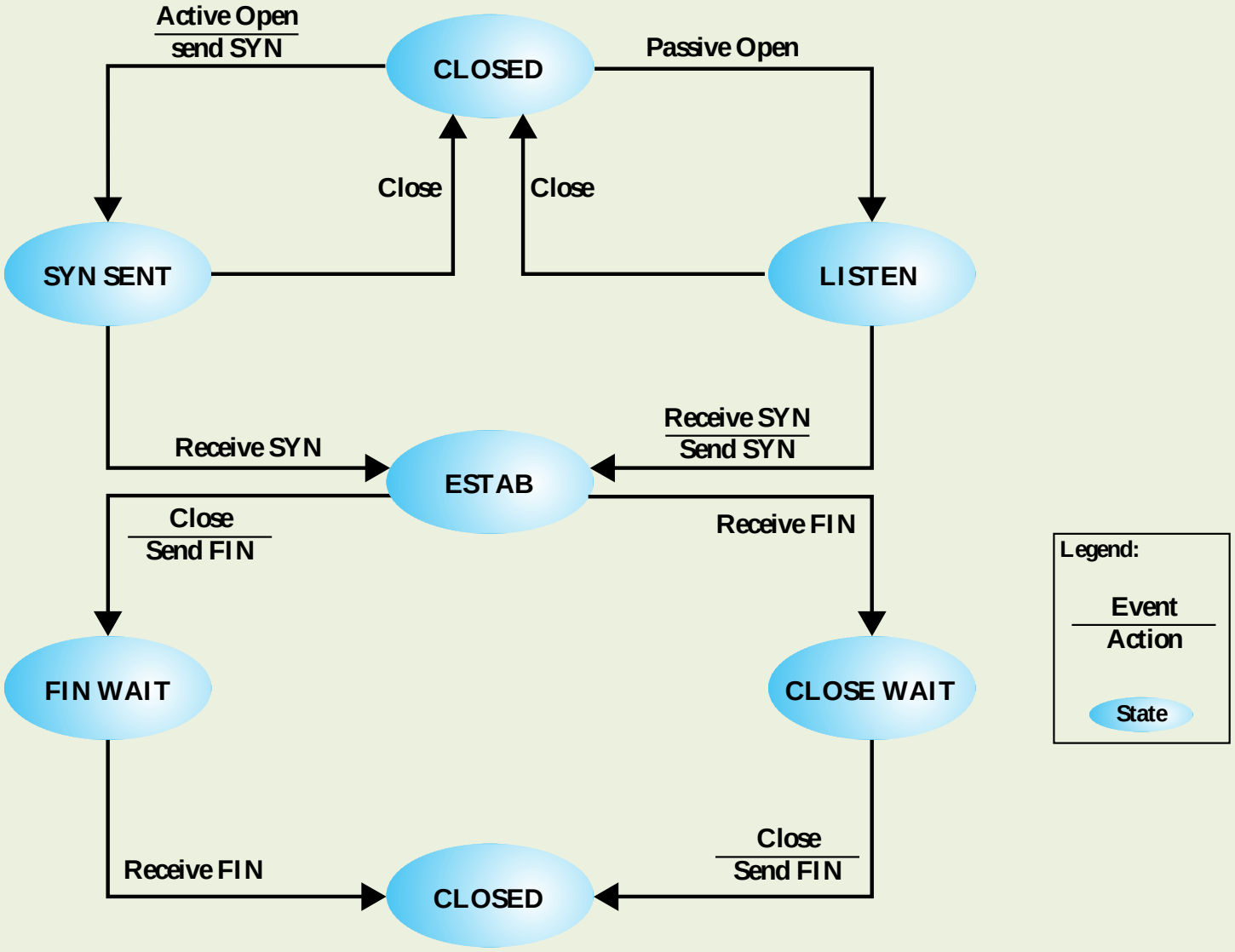


Figure 15.3 Simple Connection State Diagram

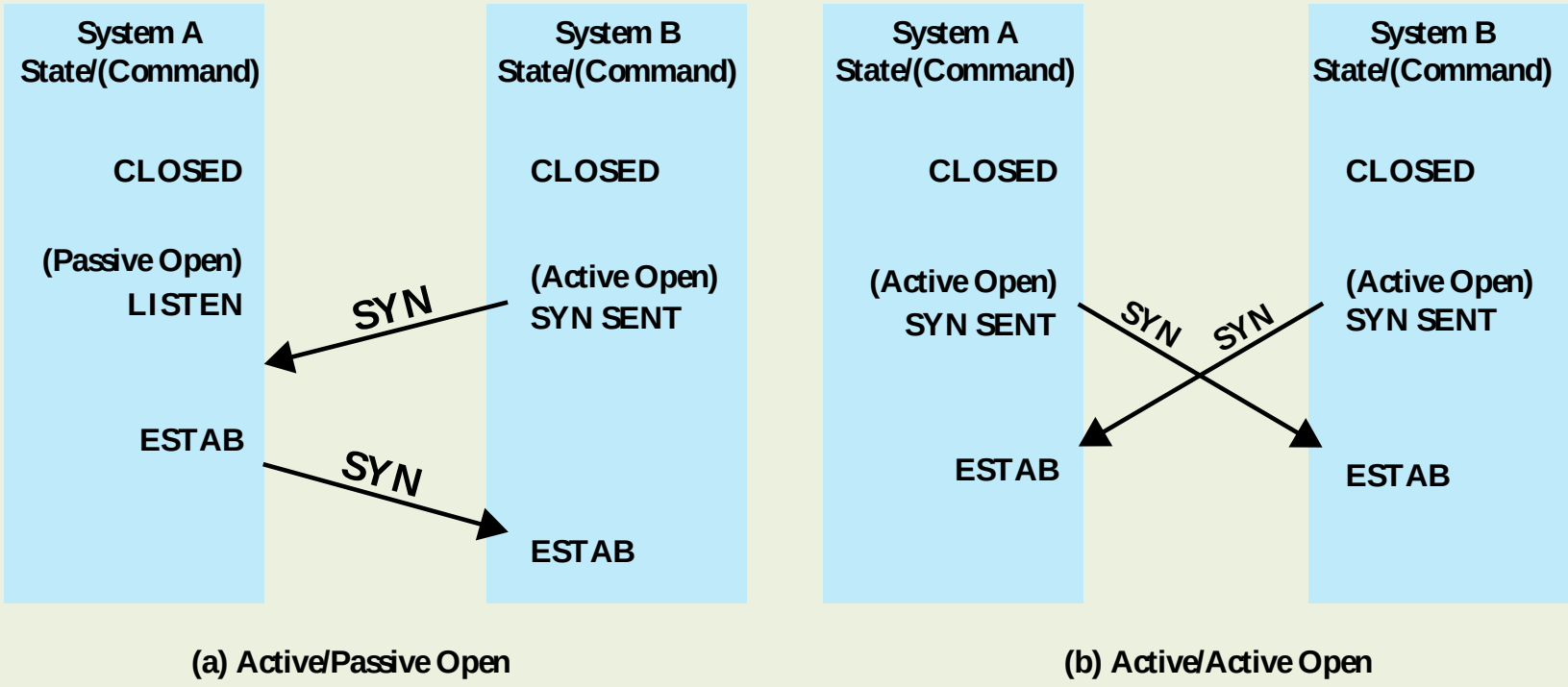


Figure 15.4 Connection Establishment Scenarios

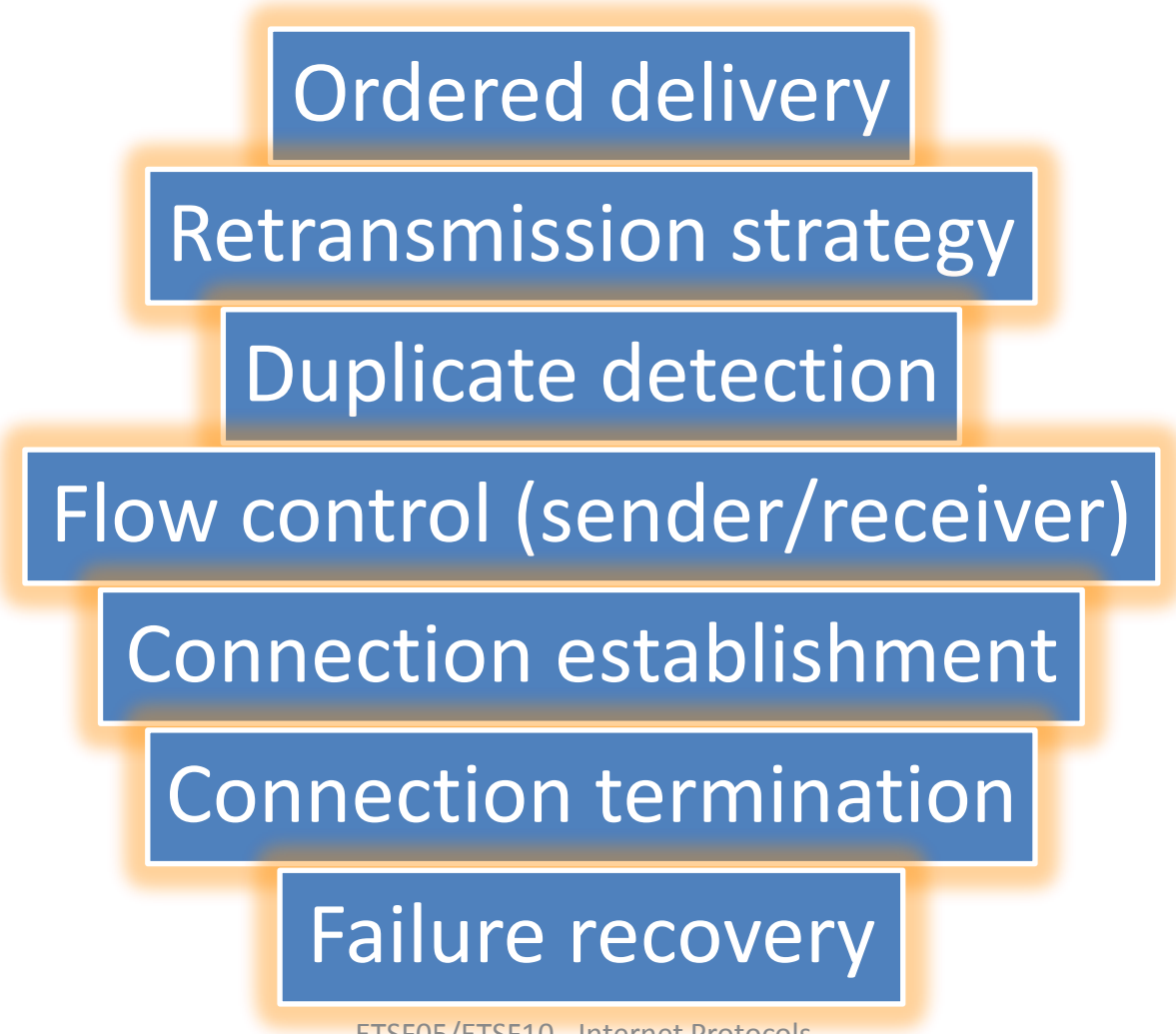
Unreliable Network Service

Examples:

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

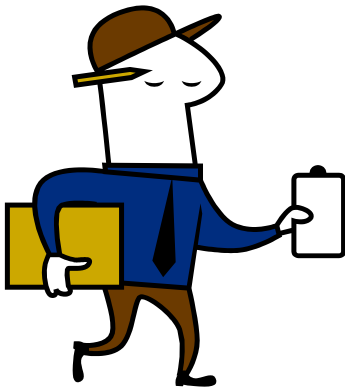
- Segments are occasionally **lost** and may arrive **out of sequence** due to variable transit delays

Issues to Address



Ordered Delivery

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



Retransmission Strategy

- Events necessitating retransmission:

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

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

Cont.

Retransmission Strategy

- No acknowledgment will be issued if a segment does not arrive successfully
  - Resulting in a retransmit
- 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*

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

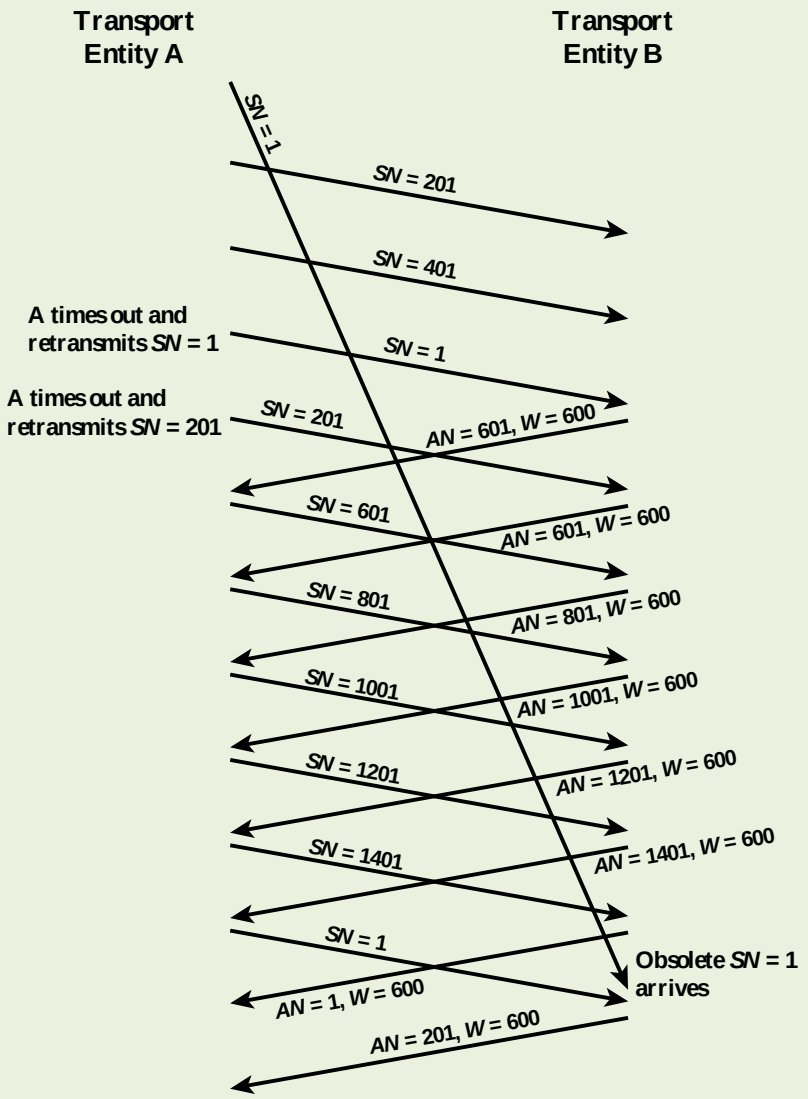


Figure 15.5 Example of Incorrect Duplicate Detection

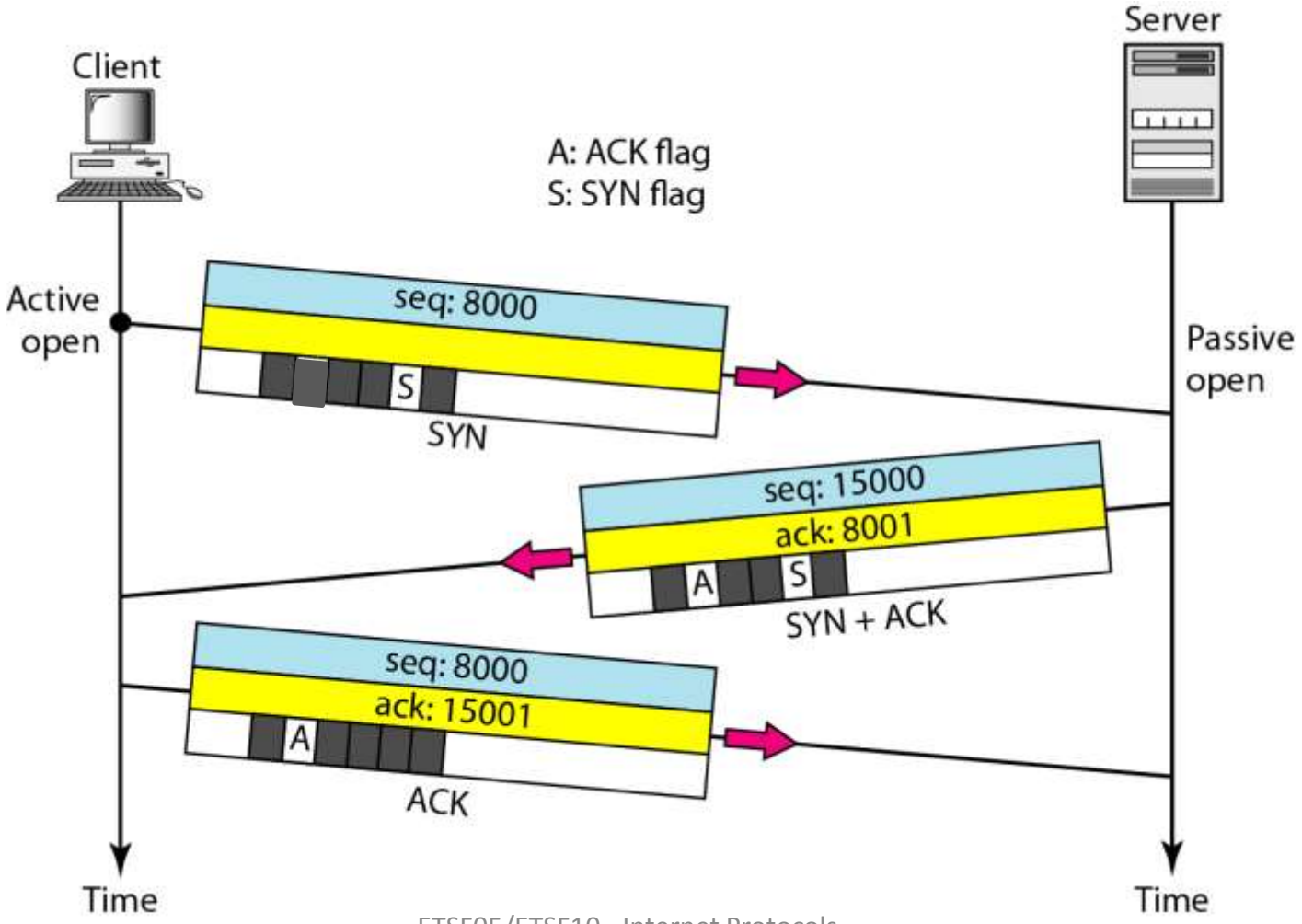
Remarks on Flow Control

- Future acknowledgments will resynchronize the protocol if an ACK/CREDIT segment is lost
- If no new acknowledgments are forthcoming the sender times out and retransmits a data segment which triggers a new acknowledgment
- Still possible for deadlock to occur

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 15.6—15.9 for details

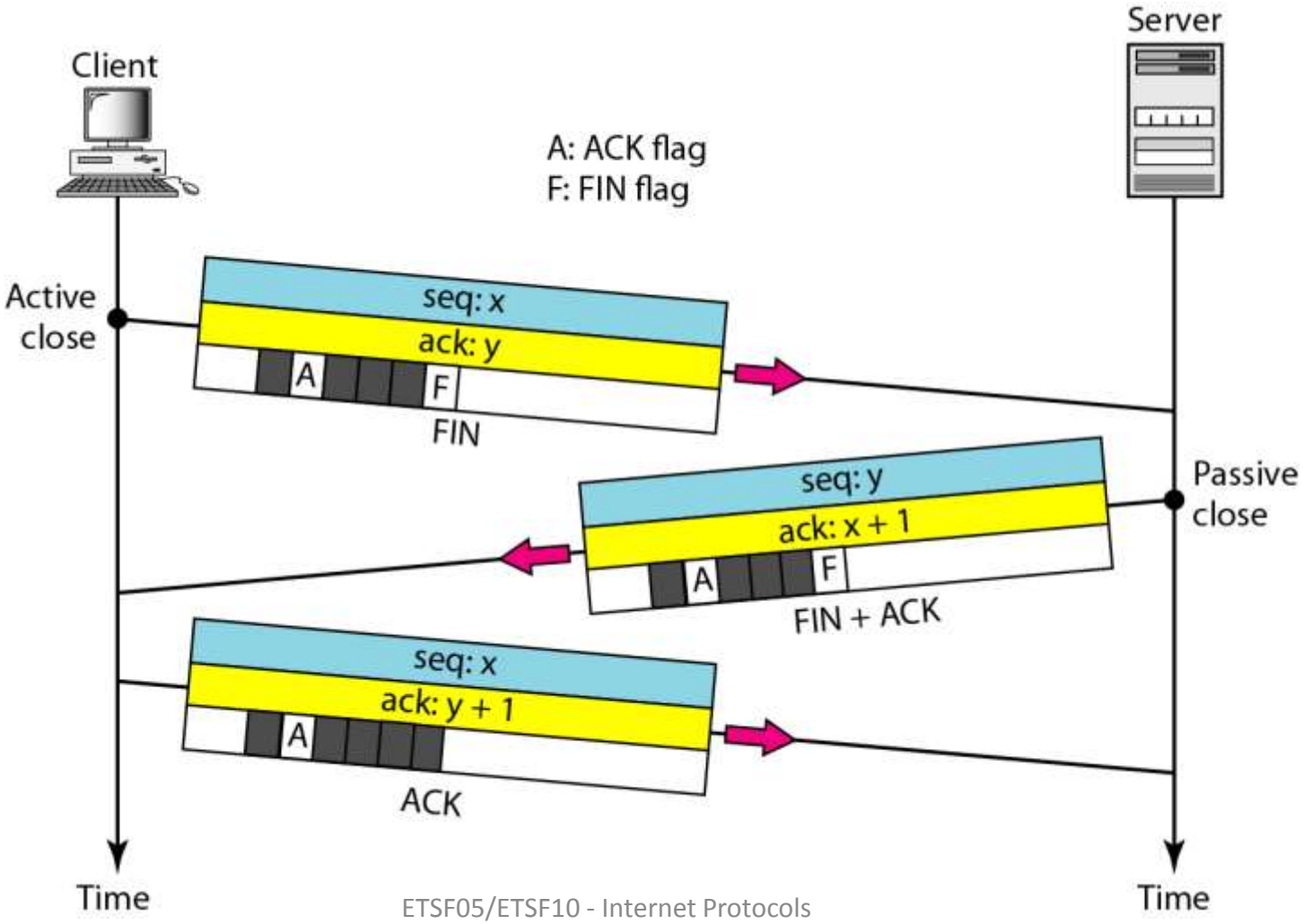
TCP Three Way Handshake



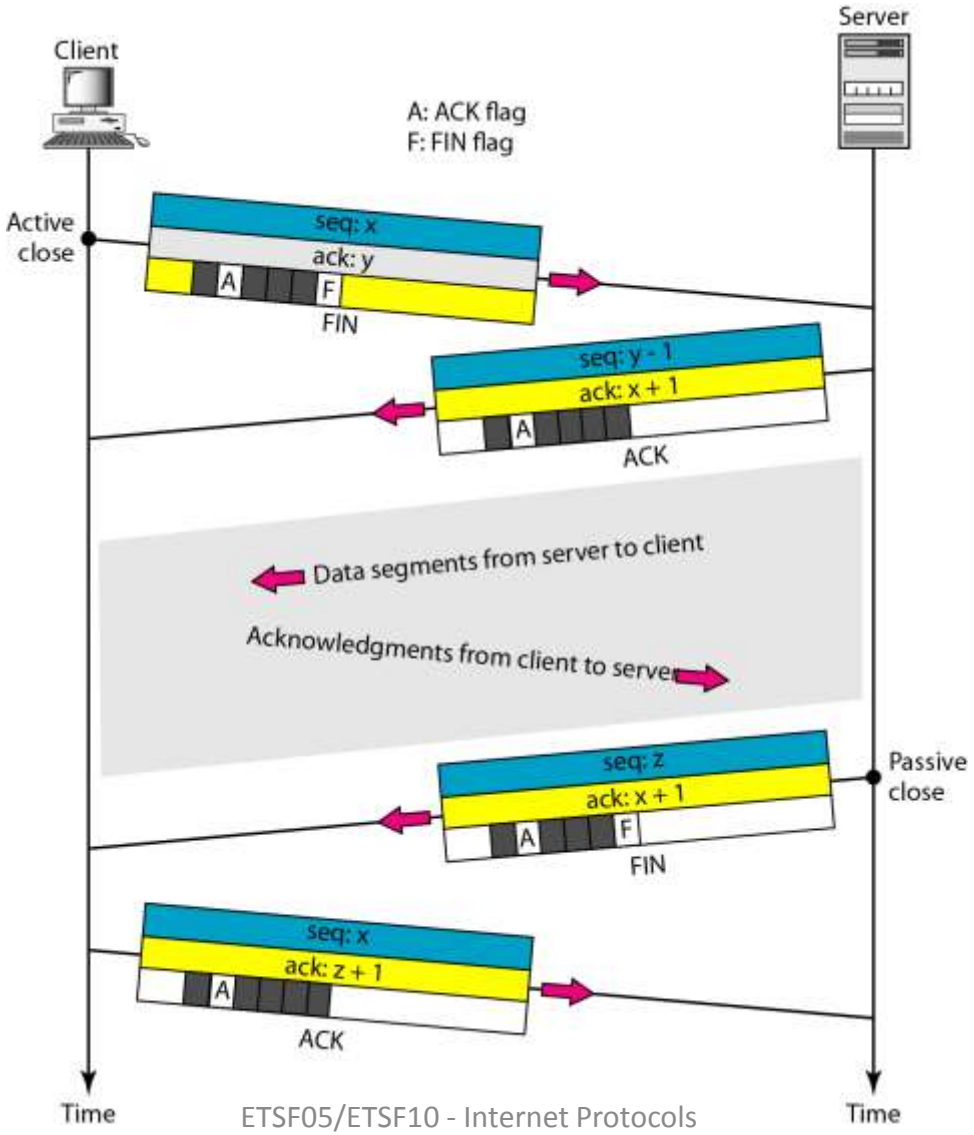
Remarks on Connection Termination

- 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

TCP Connection termination

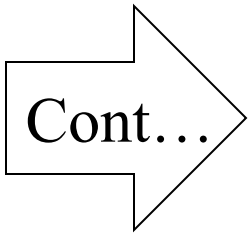


TCP Half-close



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



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

TCP Services

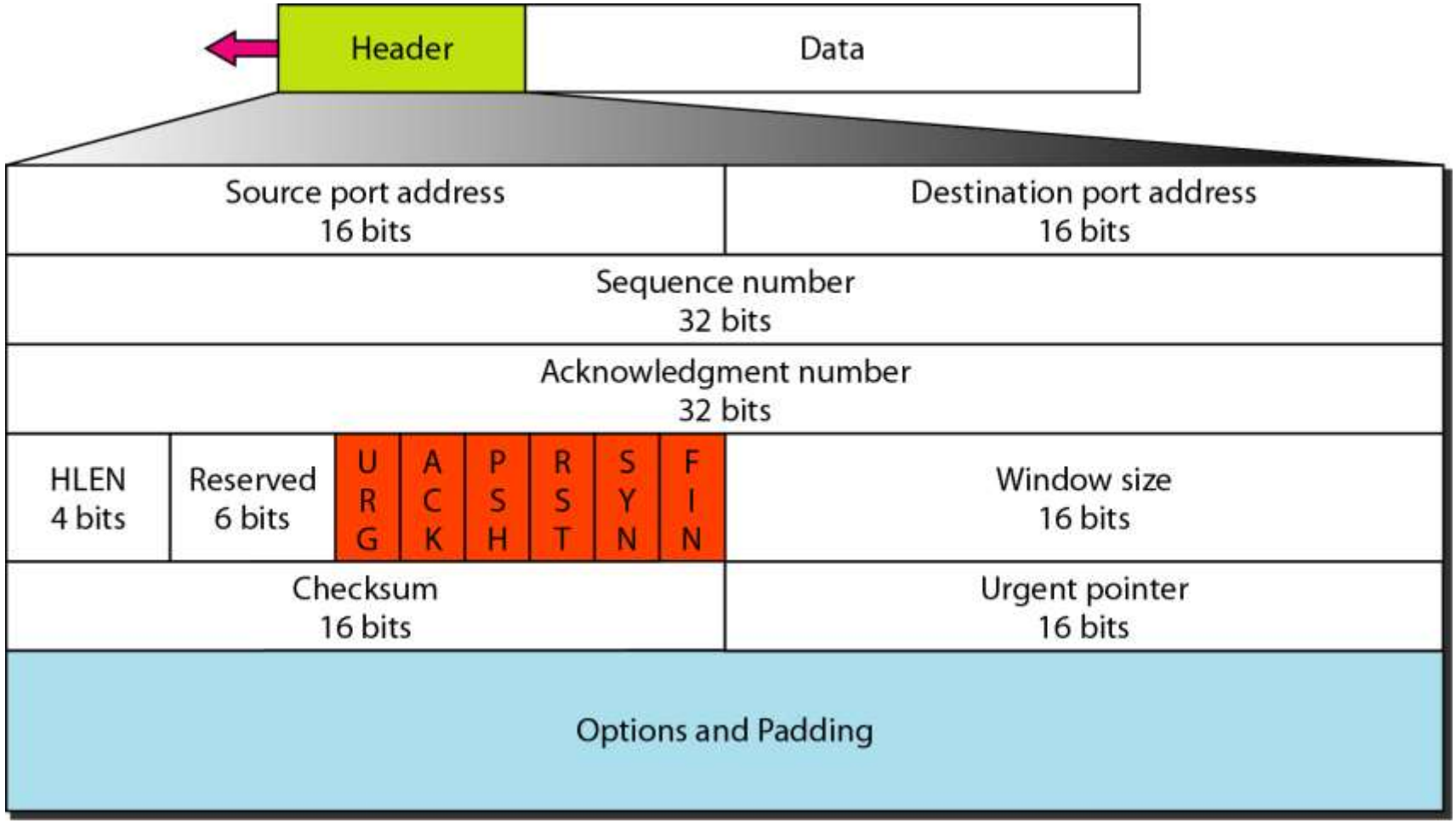
- RFC 793

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

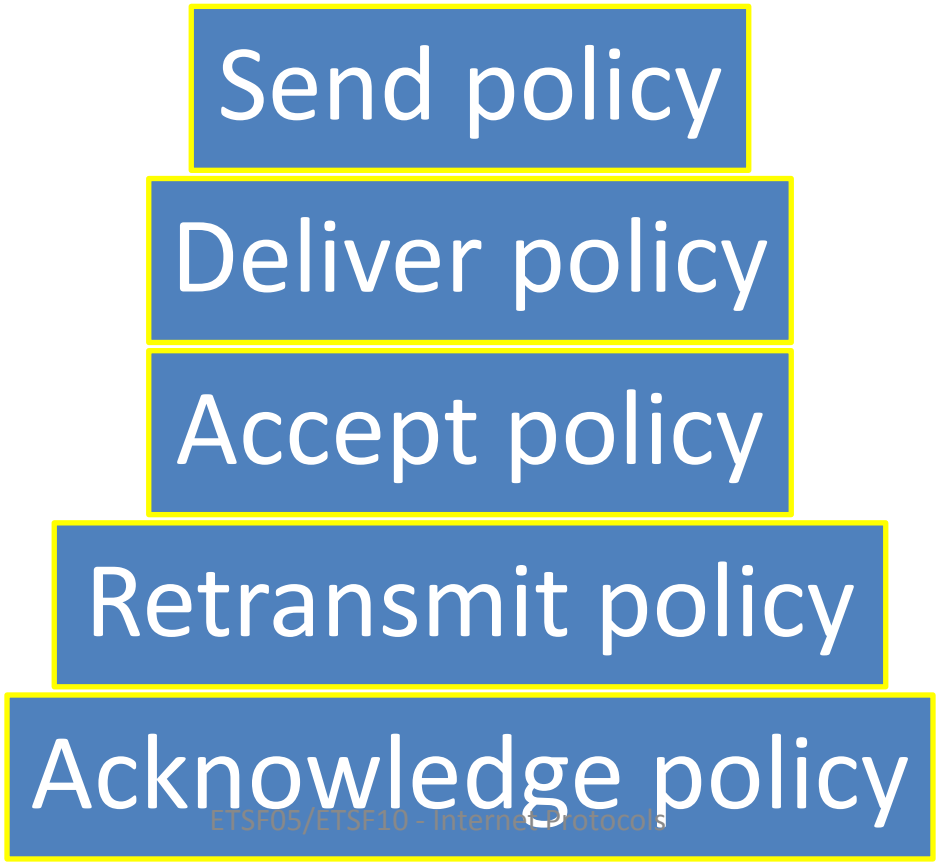
- Viewed logically as consisting of a stream of octets
- Flow control is exercised using credit allocation (Received ACKs open the sender window)

Connection termination

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

TCP Implementation
Policy Options

- Implementation opportunities:



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

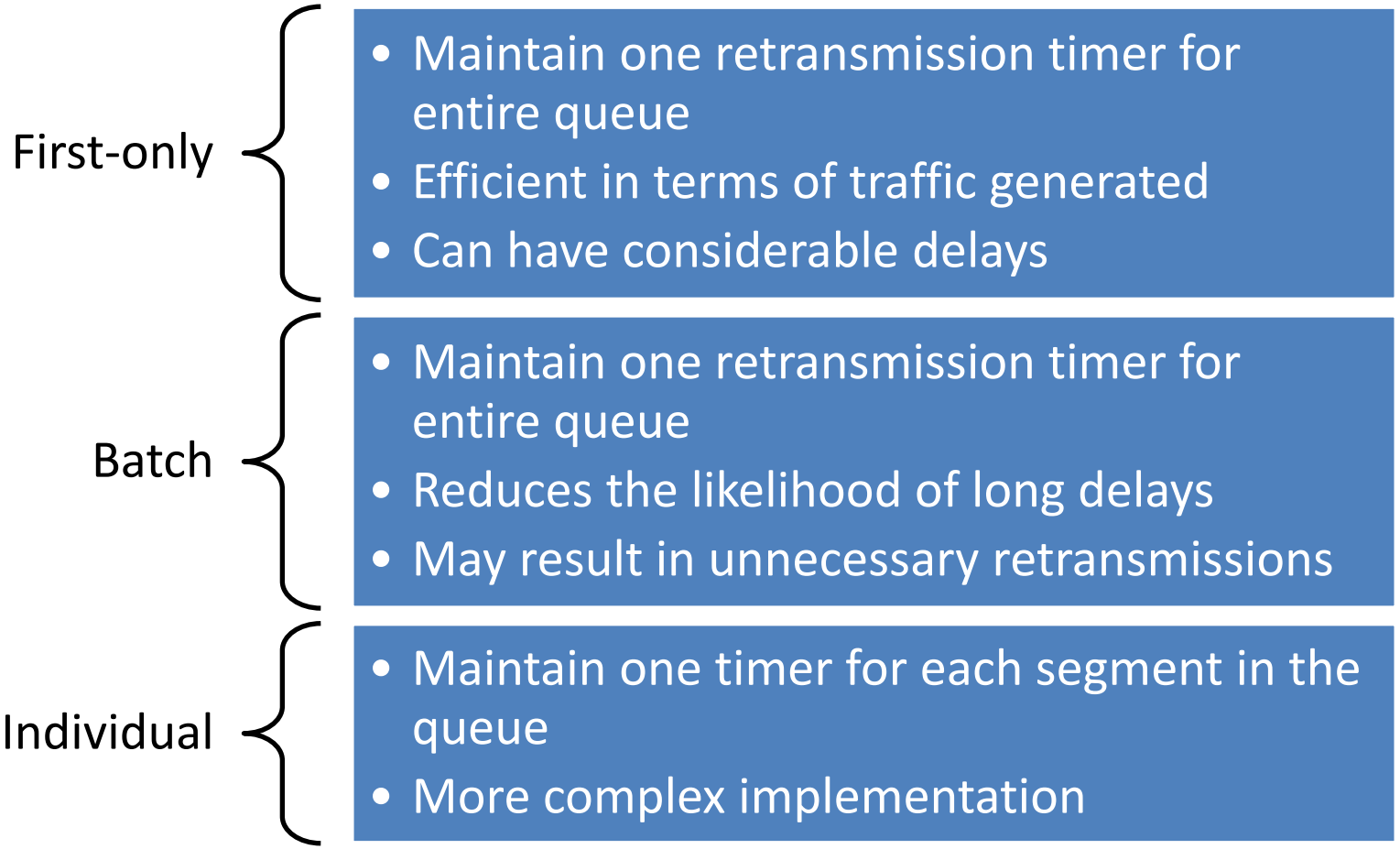
- Accepts only segments that arrive in order; any segment that arrives out of order is discarded
- Makes for simple implementation but places a burden on the networking facility
- If a single segment is lost in transit, then all subsequent segments must be retransmitted

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:



Acknowledge Policy

- Timing of acknowledgment:

Immediate	Cumulative
• Immediately transmit an 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

Congestion Control in Packet-Switching Networks

Send control packet to some or all source nodes

- Requires additional traffic during congestion

Rely on routing information

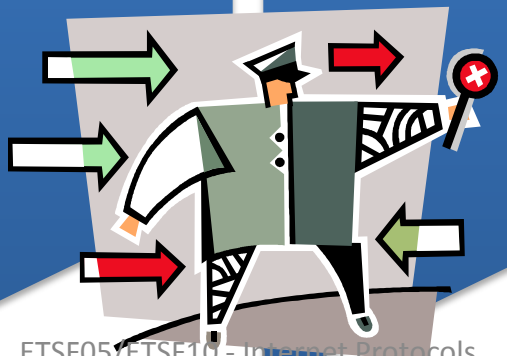
- May react too quickly

End to end probe packets

- Adds to overhead

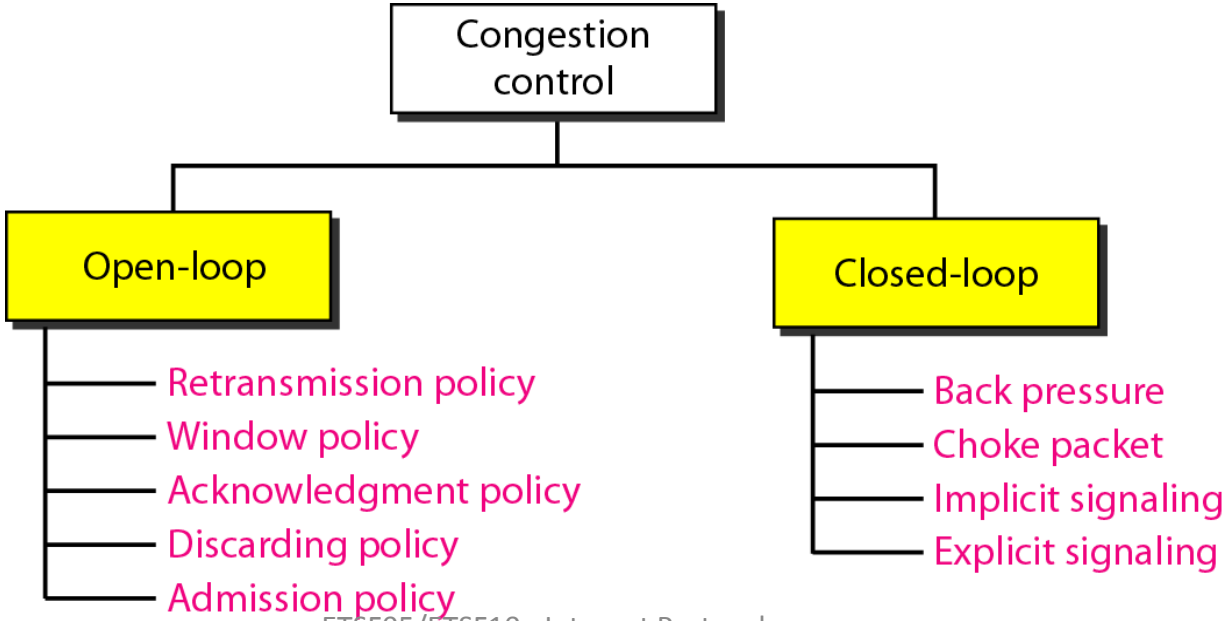
Add congestion information to packets in transit

- Either backwards or forwards



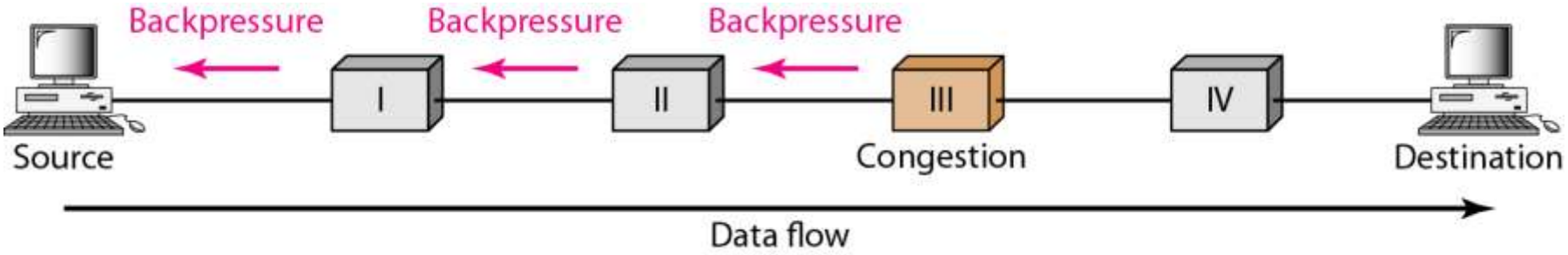
Congestion control

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



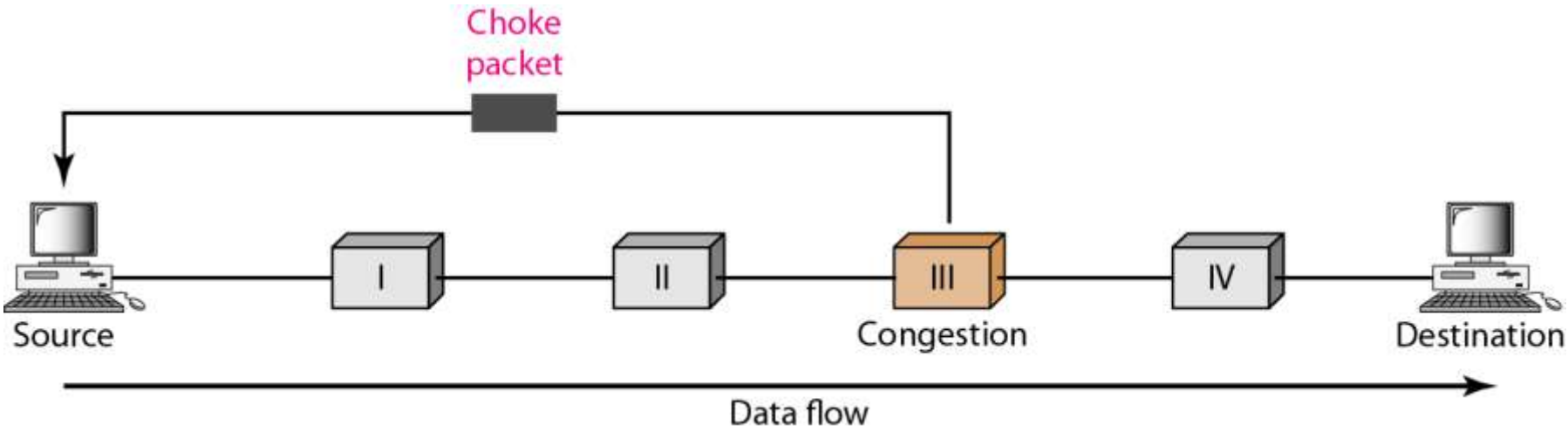
Closed-loop congestion control (1)

- Backpressure



Closed-loop congestion control (2)

- Choke packet



Congestion Control in TCP

- 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

As network or internet conditions change a static retransmission timer is likely to be either too long or too short

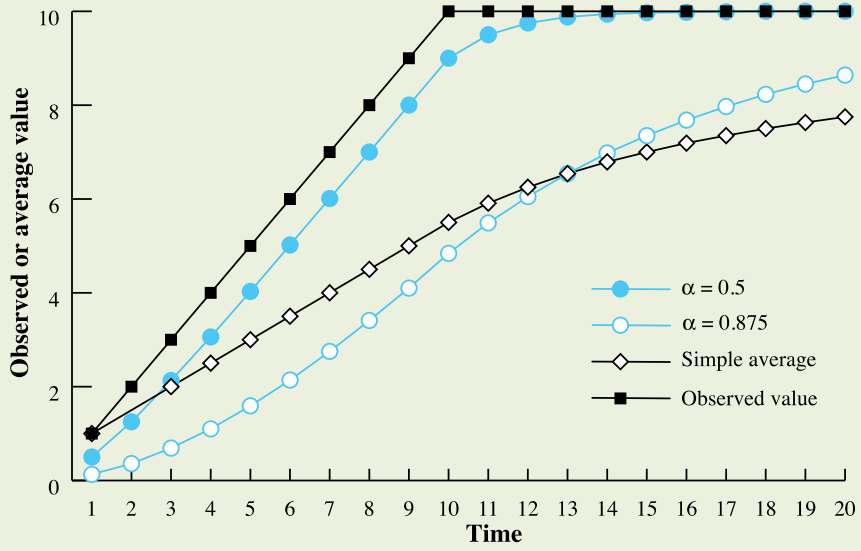
Virtually all TCP implementations attempt to estimate the current round-trip time and then set the timer to a value somewhat greater than the estimated time

Simple average:

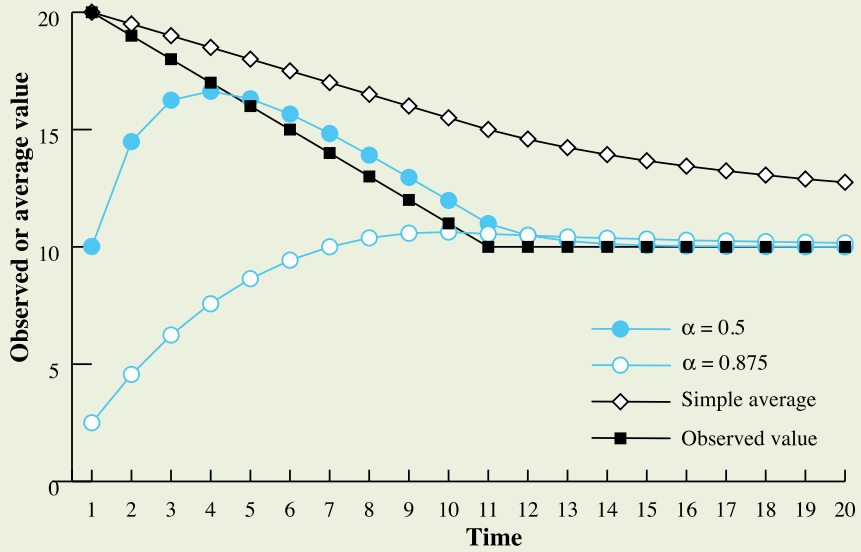
- Take the average of observed round-trip times over a number of segments
- If the average accurately predicts future round-trip times, then the resulting retransmission timer will yield good performance

Exponential average:

- Technique for predicting the next value on the basis of a time series of past values
- Specified in RFC 793

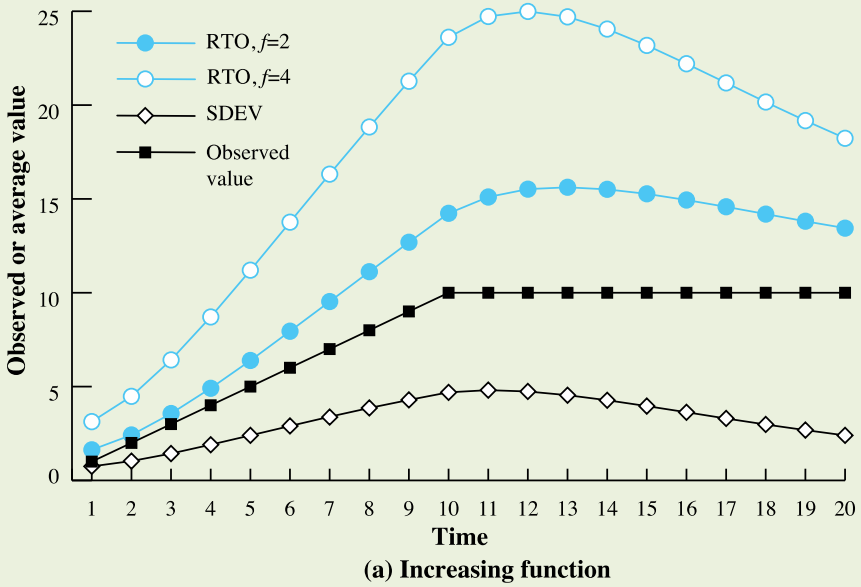


(a) Increasing function

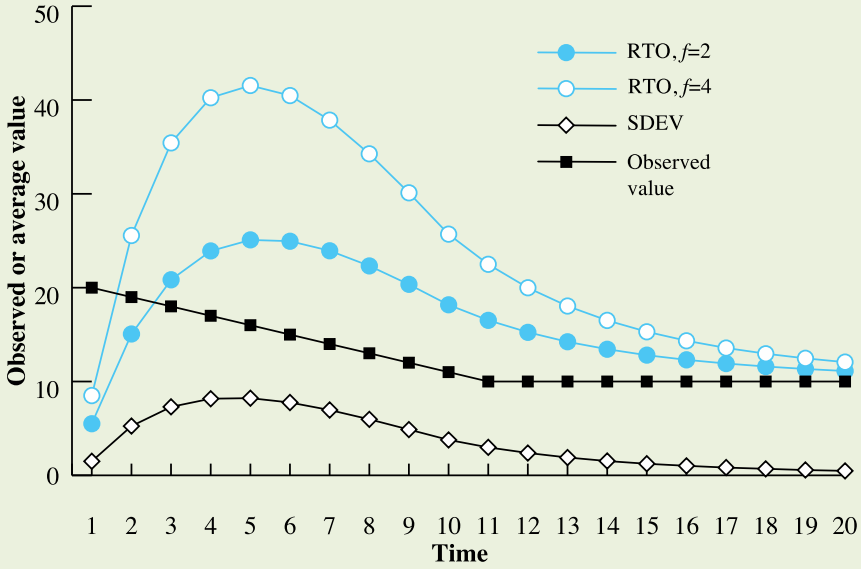


(b) Decreasing function

Figure 20.8 Use of Exponential Averaging



(a) Increasing function



(b) Decreasing function

Figure 20.9 Jacobson's RTO Calculation

Exponential RTO Backoff

- When a TCP sender times out on a segment, it must retransmit that segment
  - RFC 793 assumes that the same RTO value will be used
  - Because the time-out is probably due to network congestion, maintaining the same RTO value is ill advised
- A more sensible policy dictates that a sending TCP entity increase its RTO each time a segment is retransmitted

Karn's Algorithm

Do not use the measured RTT for a retransmitted segment to update SRTT and SDEV

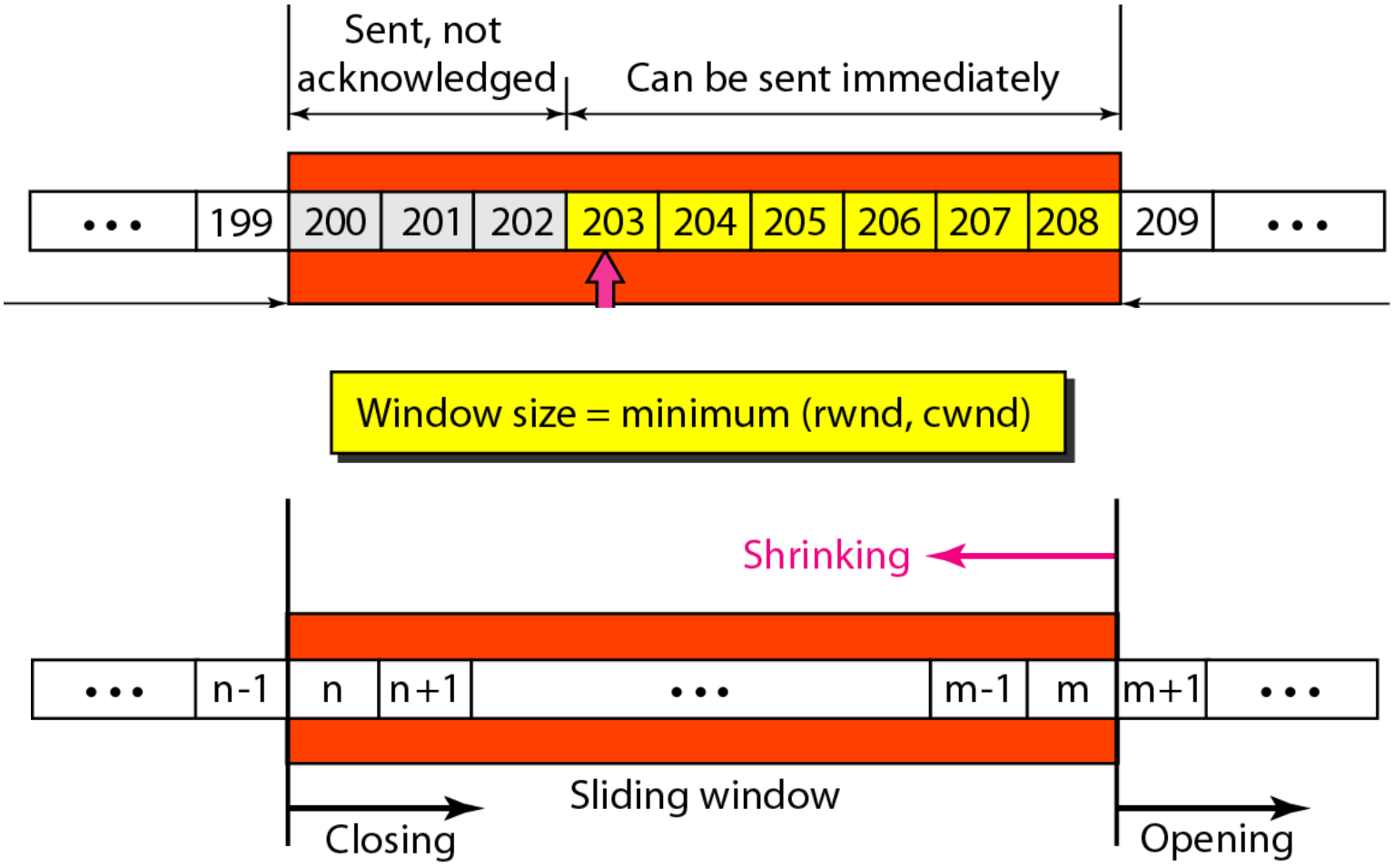
Calculate the backoff RTO using the equation RTO = q * RTO when a retransmission occurs

Use the backoff RTO value for succeeding segments until an acknowledgment arrives for a segment that has not been retransmitted

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**

Congestion window



Slow Start

- Gradually expand the window until acknowledgments are received
- TCP transmission is constrained by

$$awnd = \text{MIN}[credit, cwnd]$$

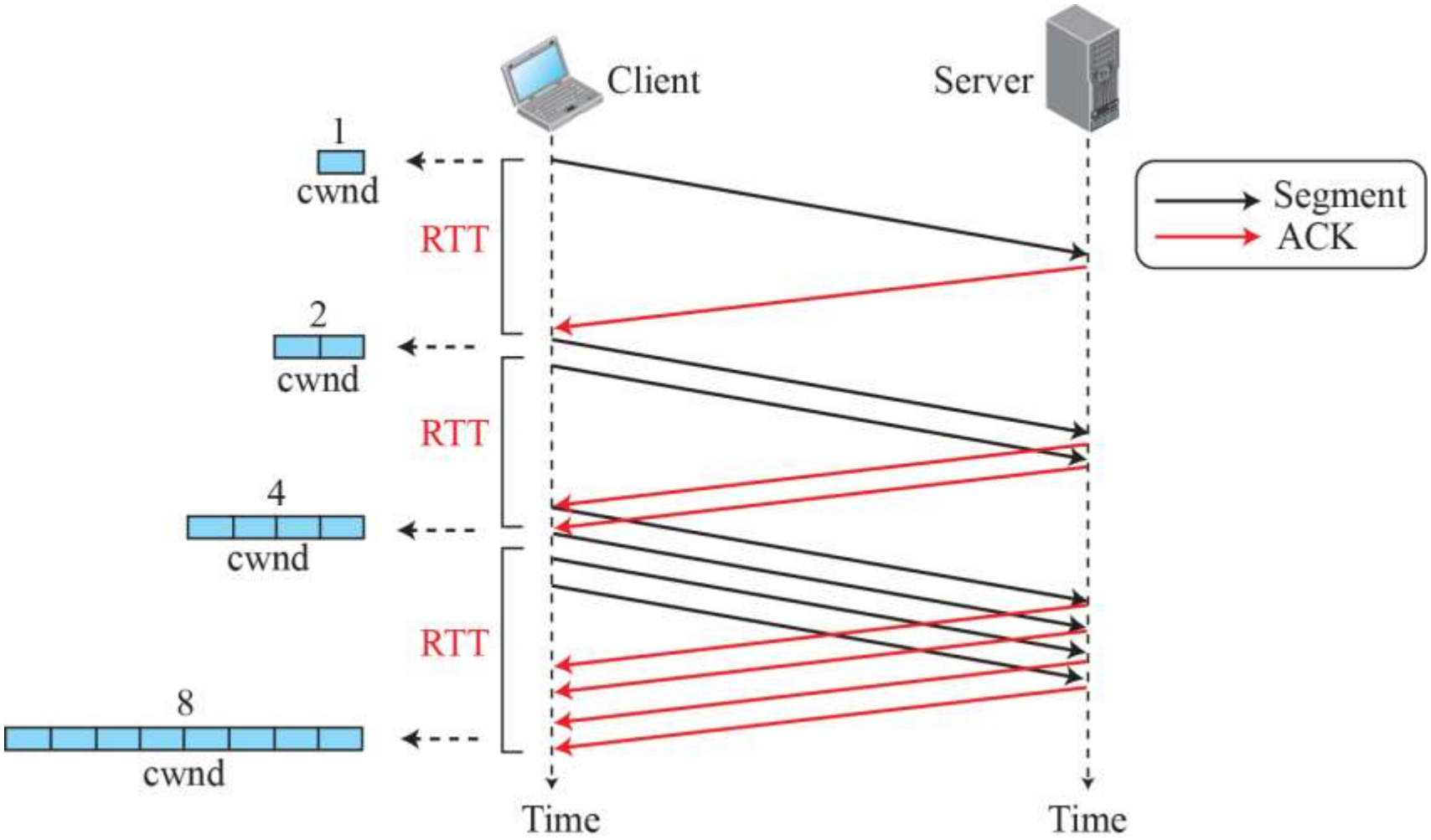
where

- awnd = allowed window, in segment
  - This is the number of segments that TCP is currently allowed to send without receiving further acknowledgments

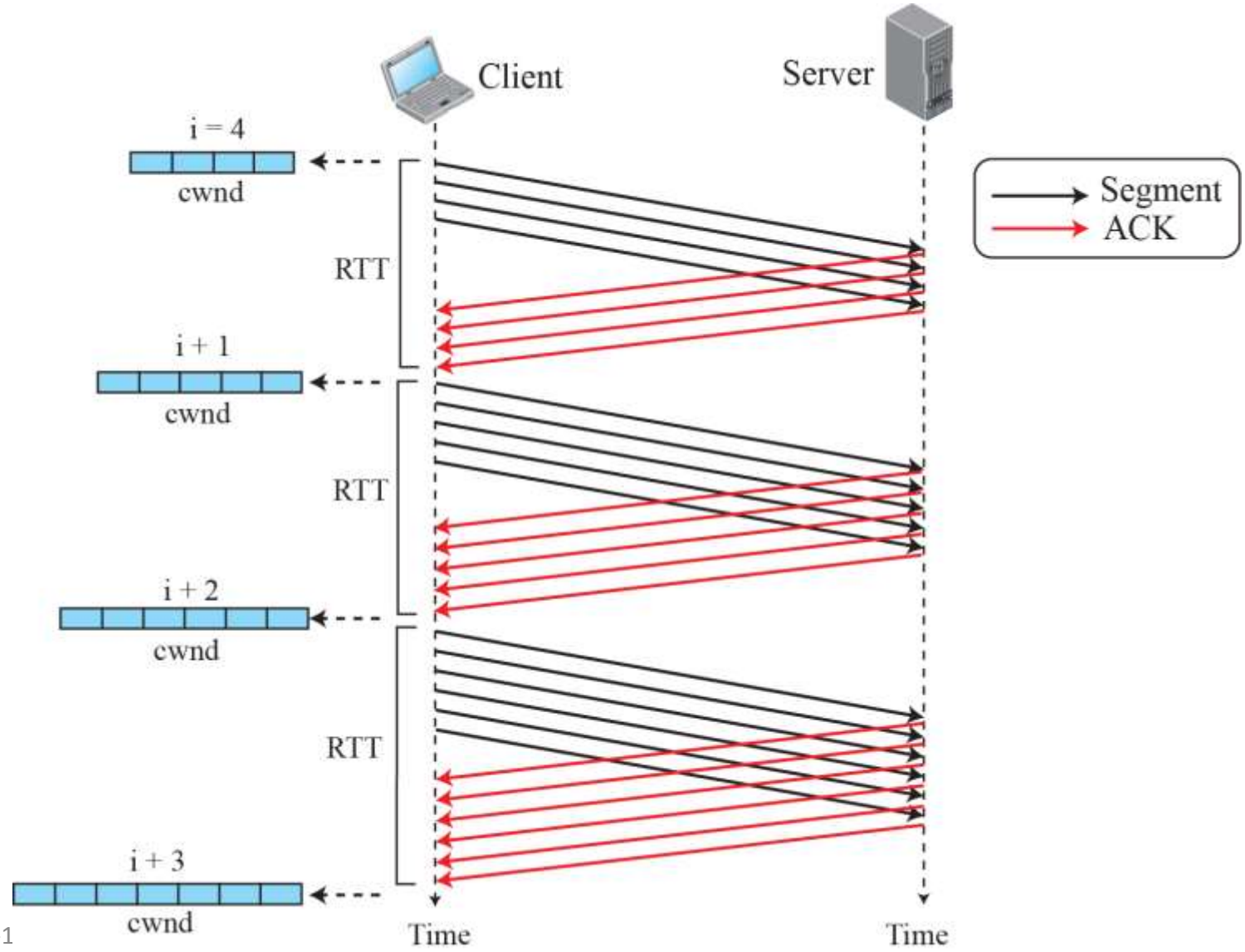
- *cwnd*= congestion window, in segments
  - A window used by TCP during startup and to reduce flow during periods of congestion

- *credit* = the amount of unused credit granted in the most recent acknowledgment, in segments
  - When an acknowledgment is received, this value is calculated as window/segment_size, where window is a field in the incoming TCP segment (the amount of data the peer TCP entity is willing to accept)

Slow start: *Exponential increase*



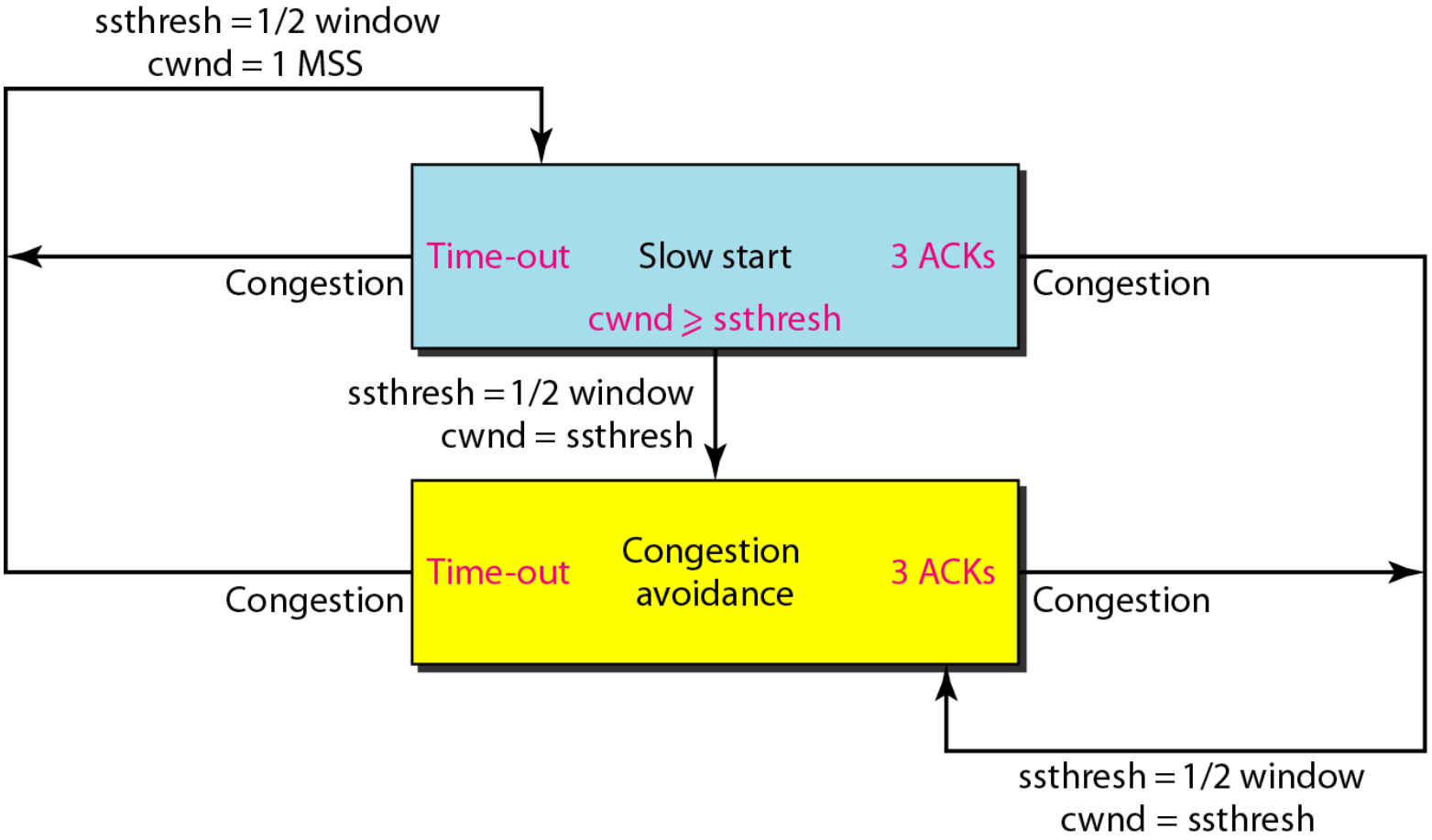
Congestion avoidance: *Additive increase*



Reaction to congestion detection

- Detection by time-out
  - Probably both channels congested
  - New slow start phase
- Detection by three ACK
  - Probably sending channel congested only
  - New congestion avoidance phase

TCP congestion policy: *Summary*



Fast Retransmit

- 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
- After three duplicate ACKs

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*