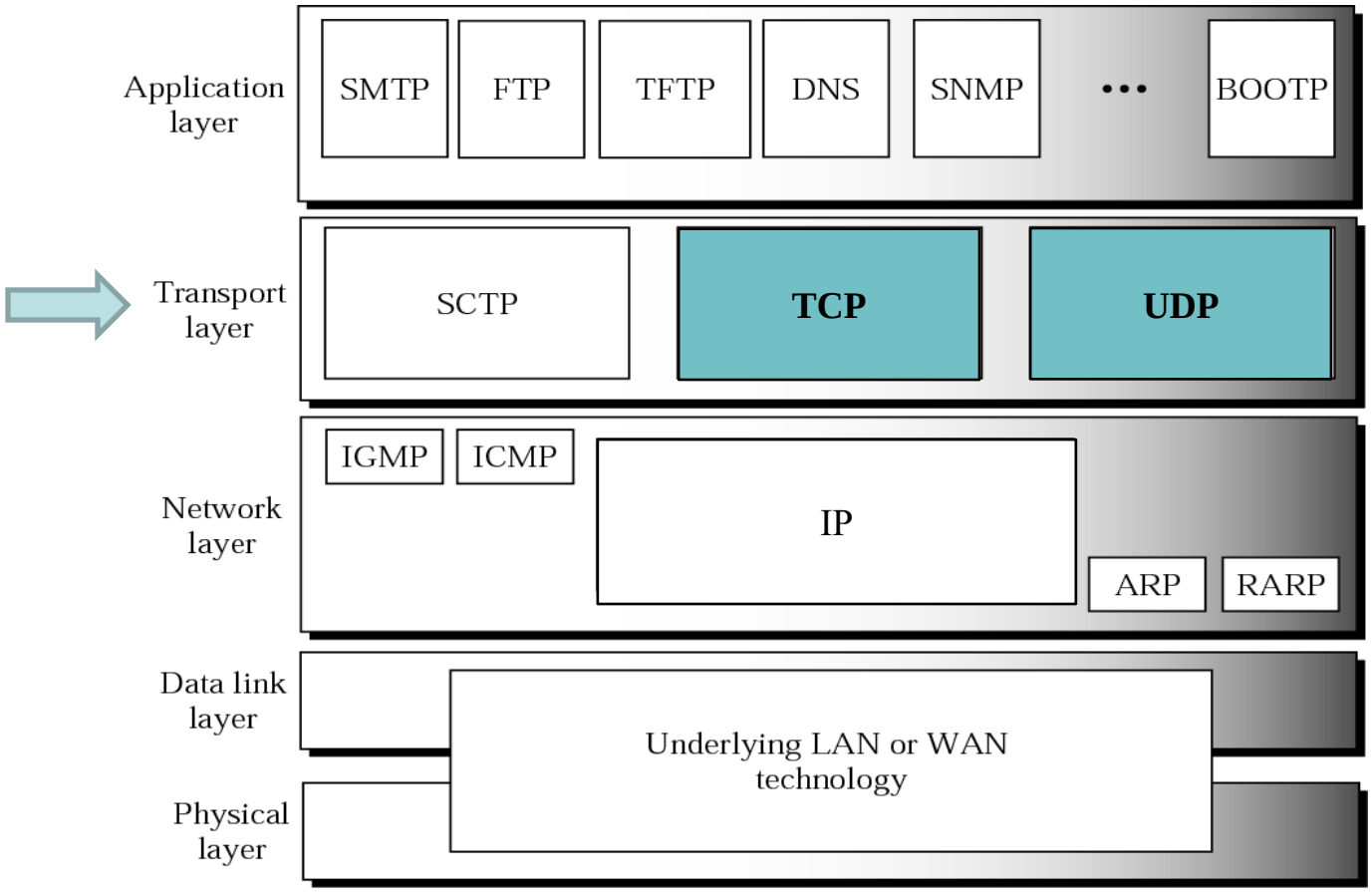
UDP & TCP

Transport Layer



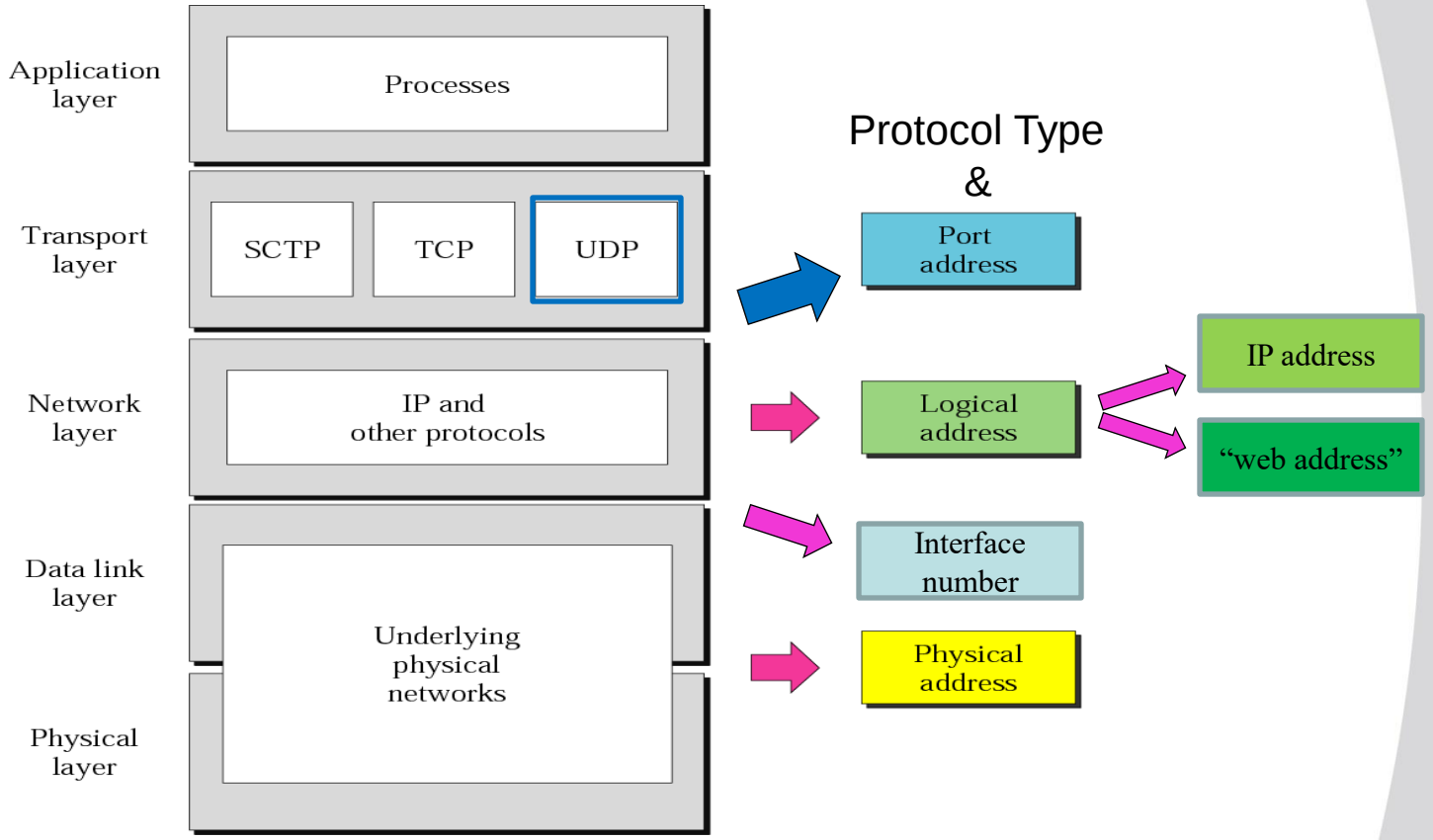
UDP - User Datagram Protocol

- IP – Transfer data from one host to another
- **UDP** – Transfer data from one process to another

  No extra reliability (duplicates, out of order, lost, delayed)
- TCP – Transfer data from one process to another

  Adds reliability

Port Numbers

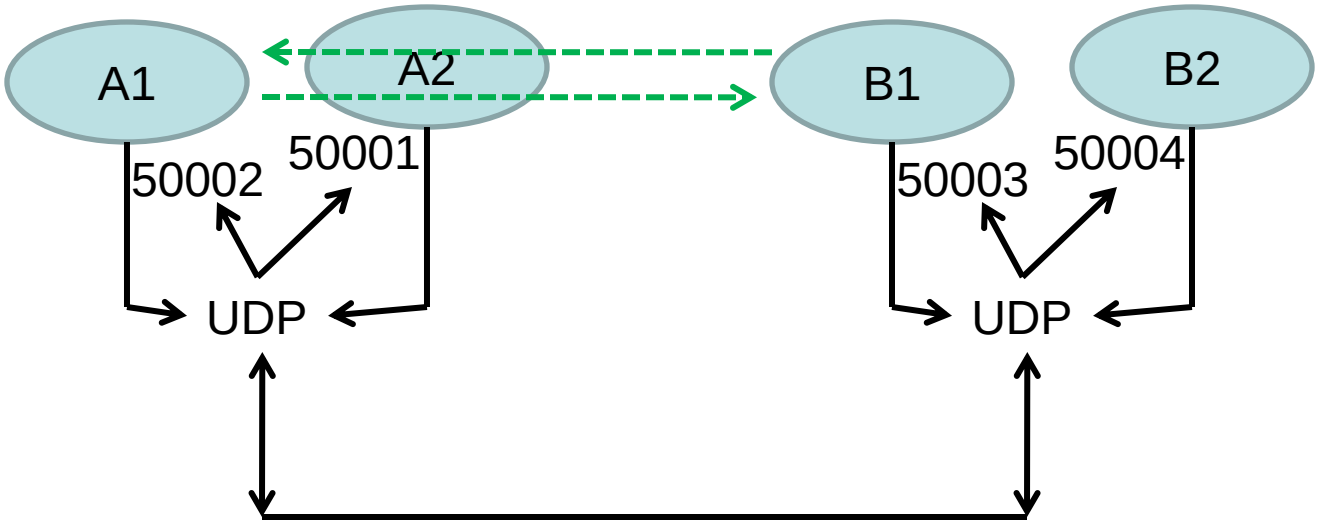


Relationship of layers and addresses in TCP/IP

Modified Fig 2.17 – Forouzan 3rd edition

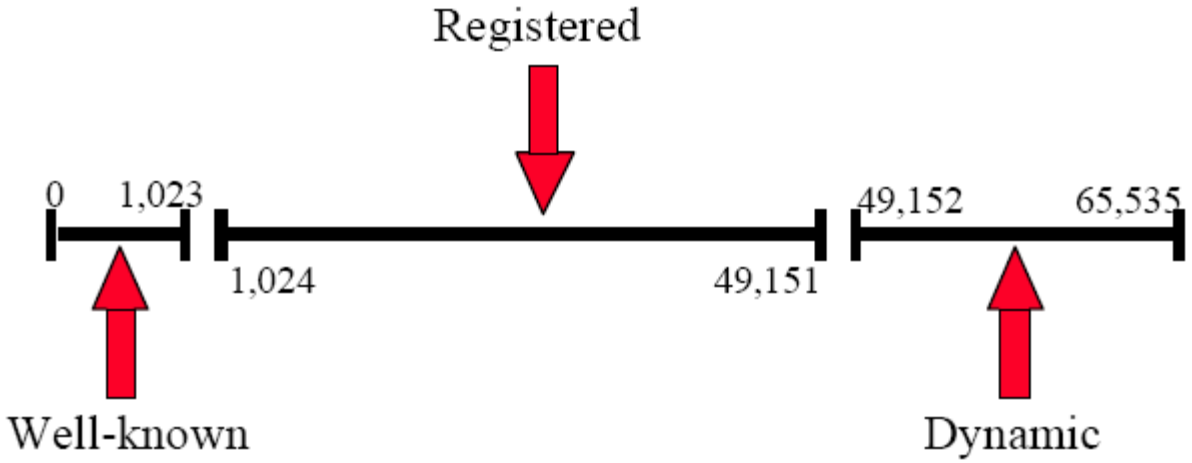
UDP Communication

- The B1 process on host B sends to port 5002 on host A. Process A1 handles the data and can send back data by using the source IP & source port # (5003)

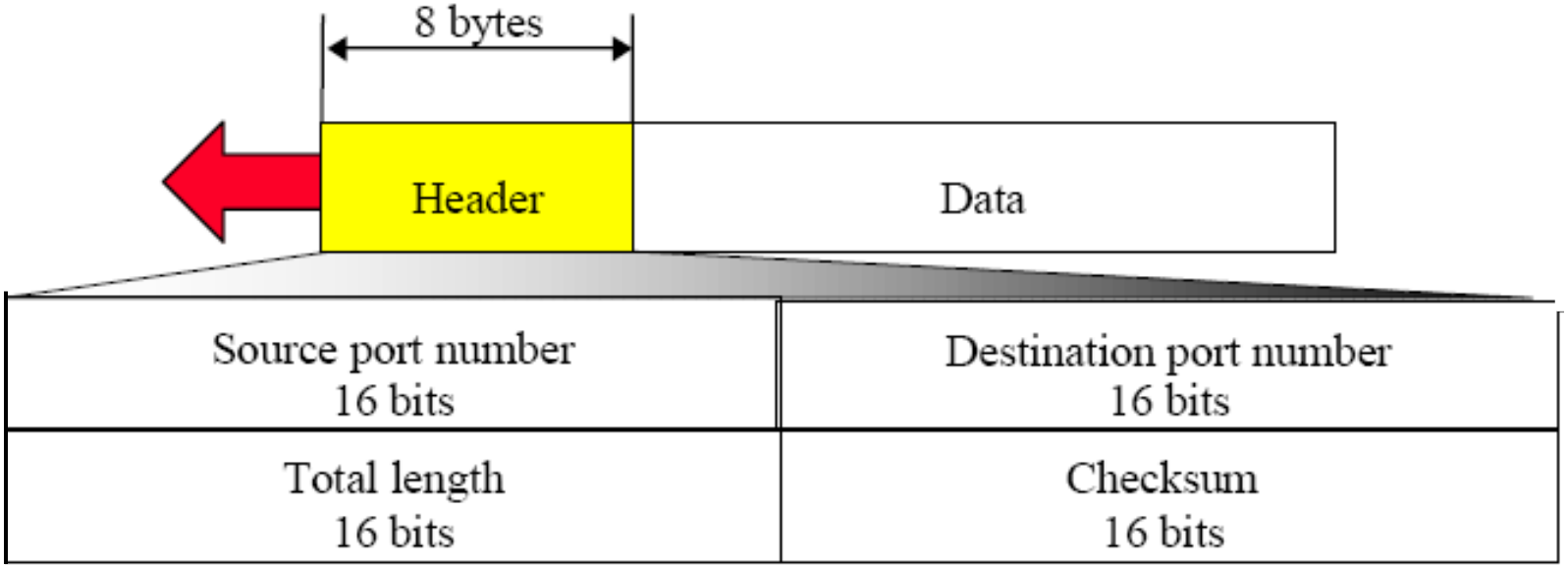


Port Numbers

- The port # has to be agreed upon before the communication
- “Well-known port numbers” (Almost the same for UDP & TCP)
- IANA - Internet Assigned Numbers Authority
- Dynamic numbers are selected locally



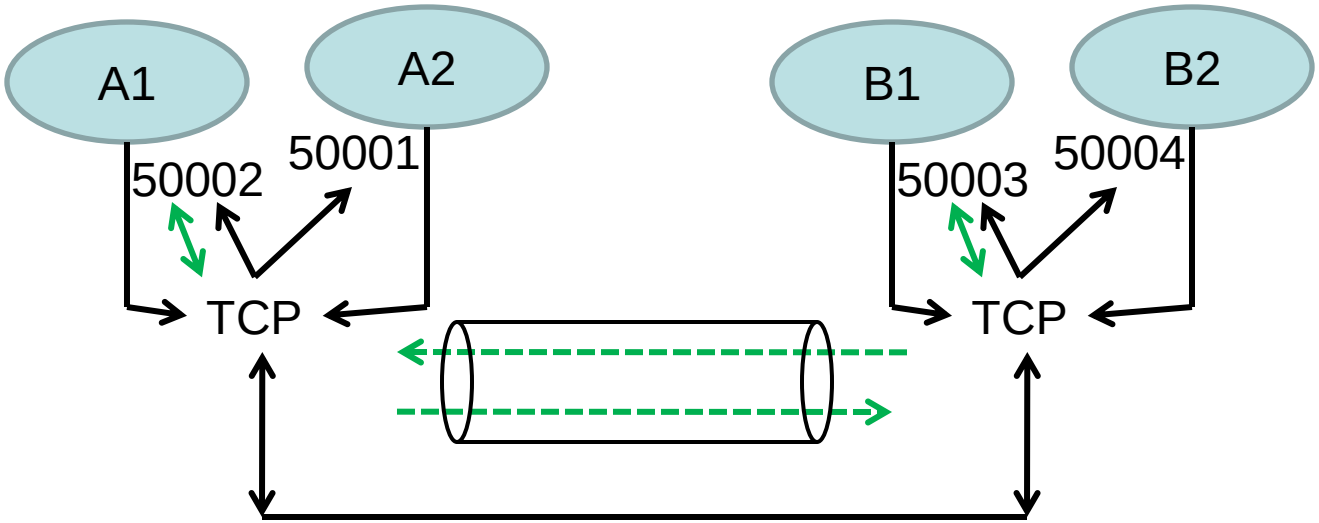
UDP Header



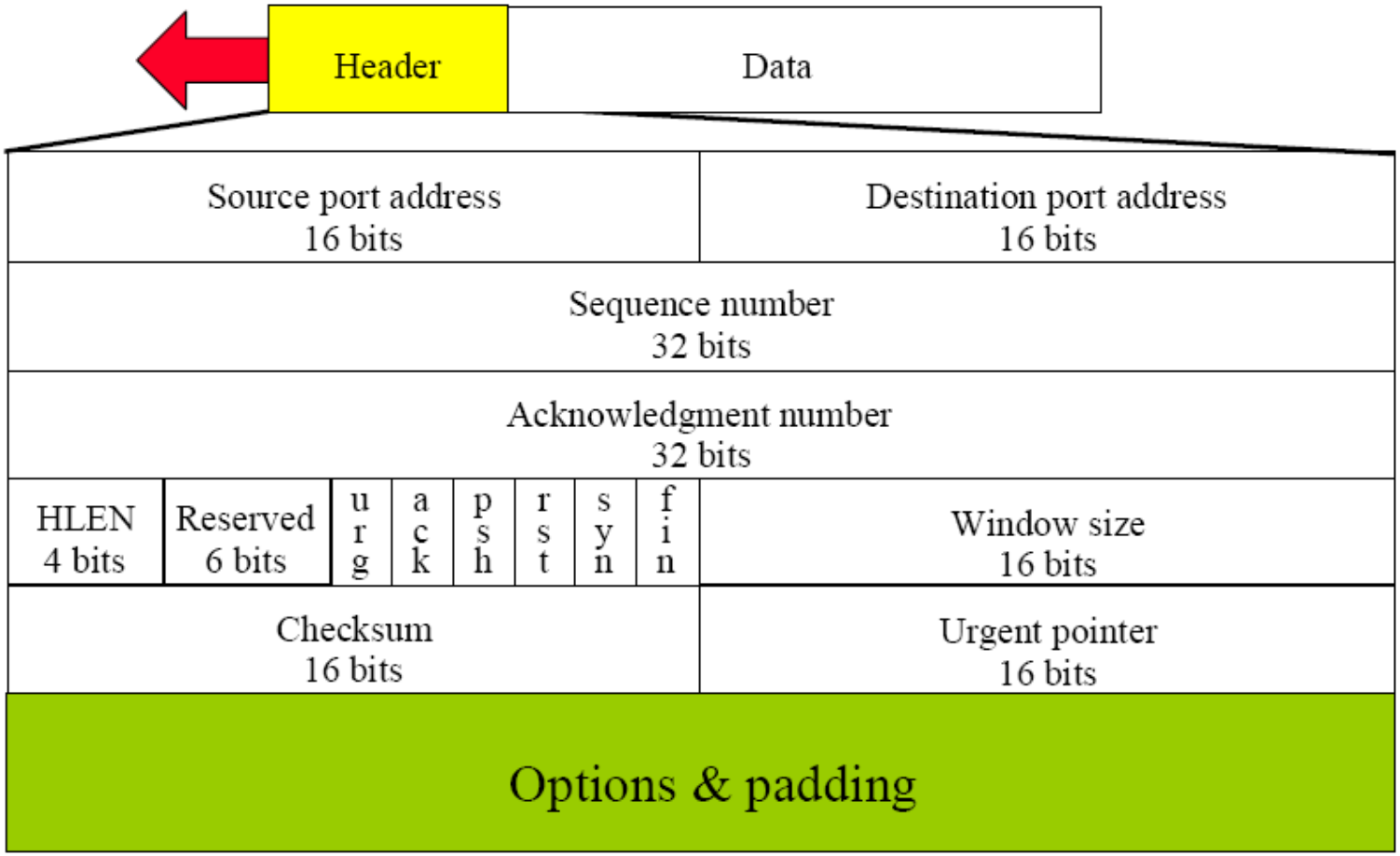
Checksum is optional

TCP – Transport Control Protocol

- A **virtual connection** is established
  - Full duplex. Concurrent transfers in both dir. Identified by source & dest. port #
- This end-to-end communication ensures reliability
  - Managed by the TCP



TCP Header


Header
Data
Source port address
16 bits
Destination port address
16 bits
Sequence number
32 bits
Acknowledgment number
32 bits
HLEN
4 bits
Reserved
6 bits
urg
ack
psh
rst
syn
fin
Window size
16 bits
Checksum
16 bits
Urgent pointer
16 bits
Options & padding

TCP Segment Format

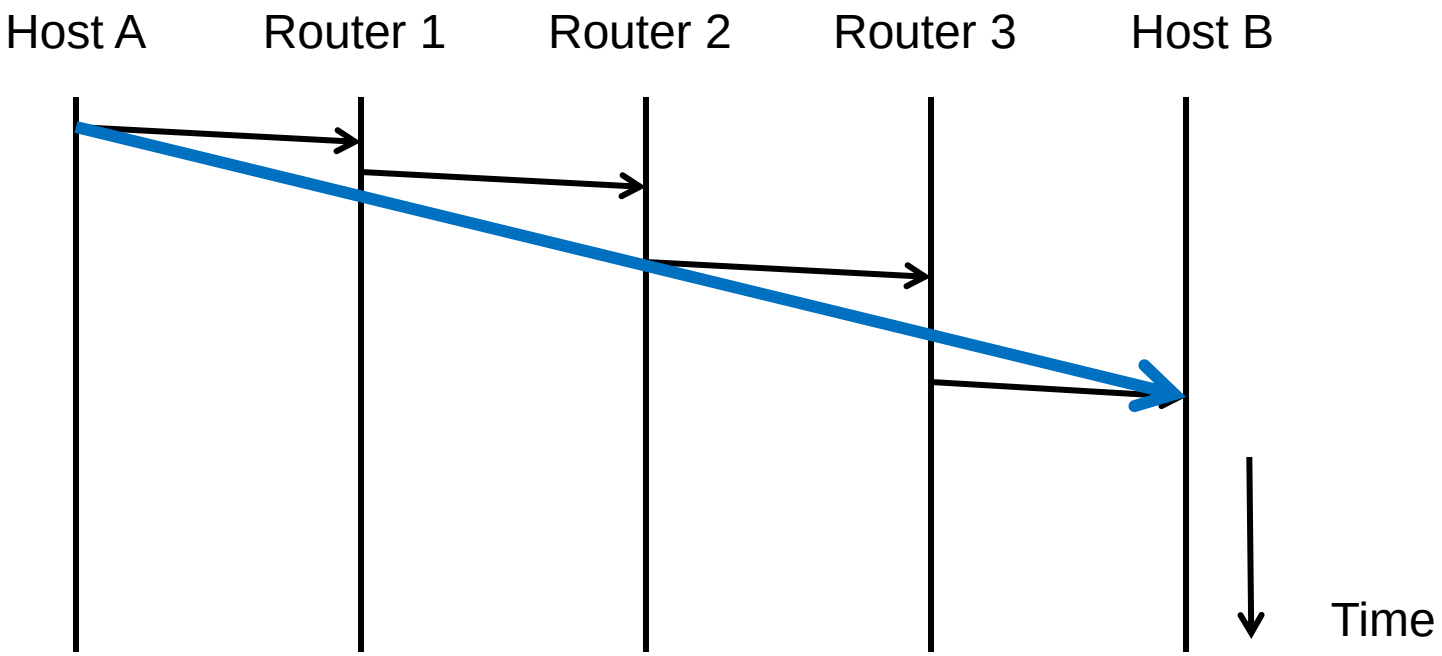
- **Port #** (2 bytes): Used the same way as in UDP
- **Sequence #** (4 bytes): Position of senders byte stream
- **Acknowledgement #** (4 bytes): Byte # that receiver expects to get next
  - All previous bytes has been received and are contiguous
- **HLEN** (4 bits): Header length in multiples of 32 bit
- **Code bits** (6 bits):
  - URG: Urgent pointer is valid
  - ACK: Ack field is valid
  - PSH: Push request
  - RST: Reset connection
  - SYN: Synchronize sequence numbers
  - FIN: Sender closes the connection

TCP Segment Format (cont)

- **Window** (2 bytes): Input buffer size (WIN or *rwnd* – receiver window size)
- **Checksum** (2 bytes): Used on pseudo-header + header + data
  - Same as in UDP but with the protocol # = 6 (instead of 17)
- **Urgent pointer** (2 bytes): Points to where urgent data ends
  - E.g. used for aborting a connection (URG bit is set)
- **Options**:
  - One byte: NO OP (No Operation), End of Option
  - Multiple bytes: MSS (Maximum Segment Size ≈MTU-40), Window Scale Factor*, Timestamp

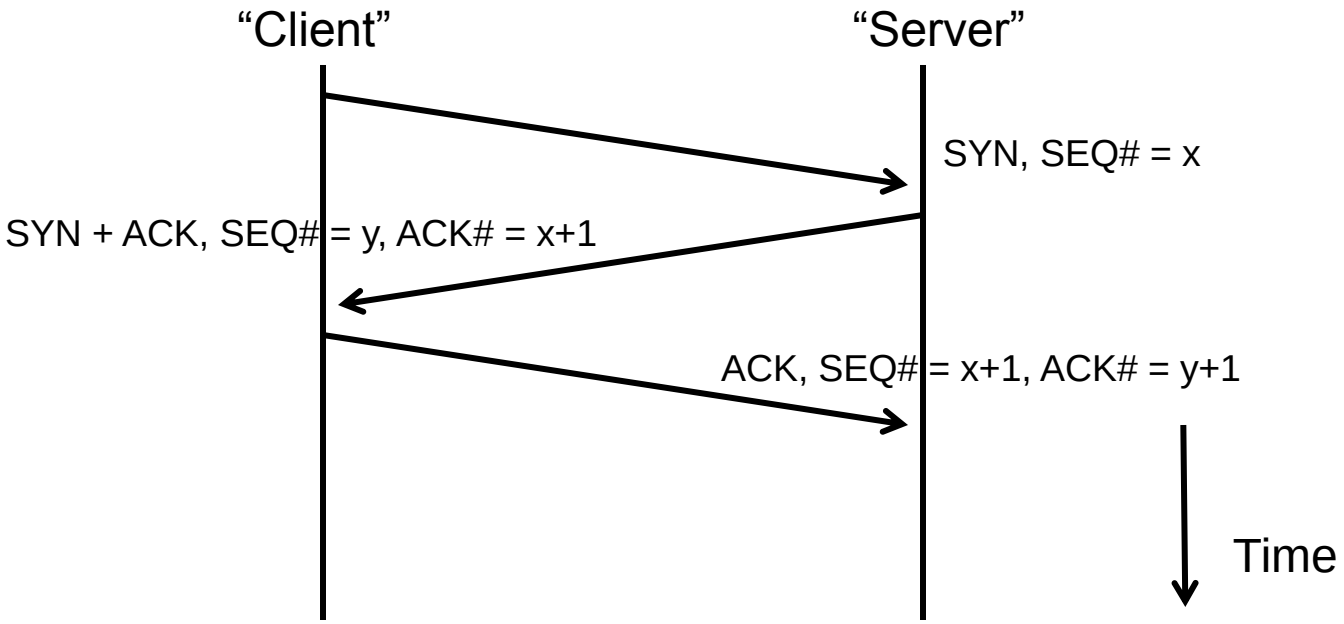
*) Used when window size gets big, e.g. high speed networks

“Signal Diagram”



Connection Establishment

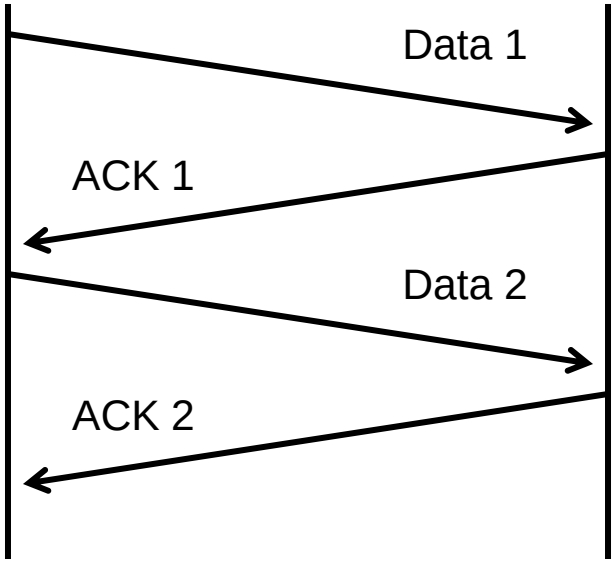
Three-way handshake



NB) Can be established from both sides at the same time
Informs about the initial random sequence number

Data Transfer

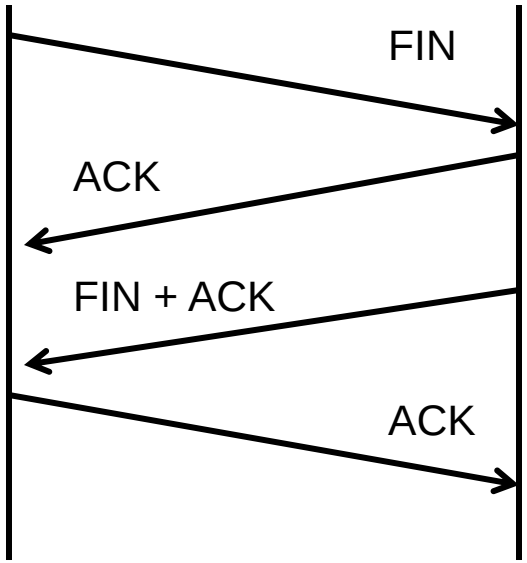
Positive ACKs with retransmissions



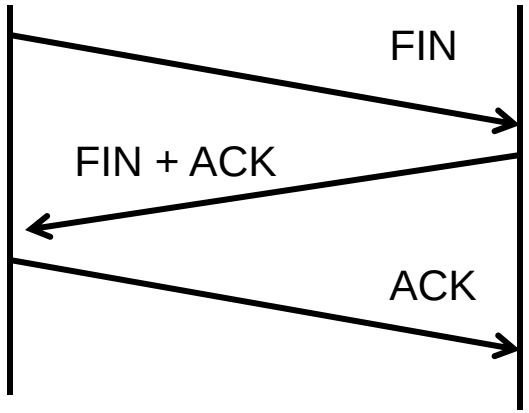
- Retransmission if no ACK (timer)
- Sequence number detects duplicates
- ACKs can be “piggy backed” on data in other direction

Closing a Connection

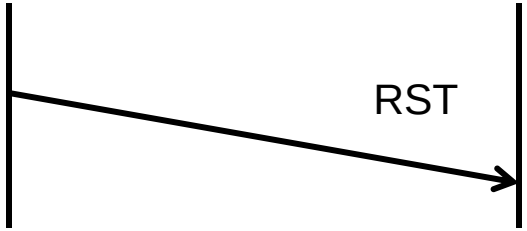
Only 1 direction is closed at a time



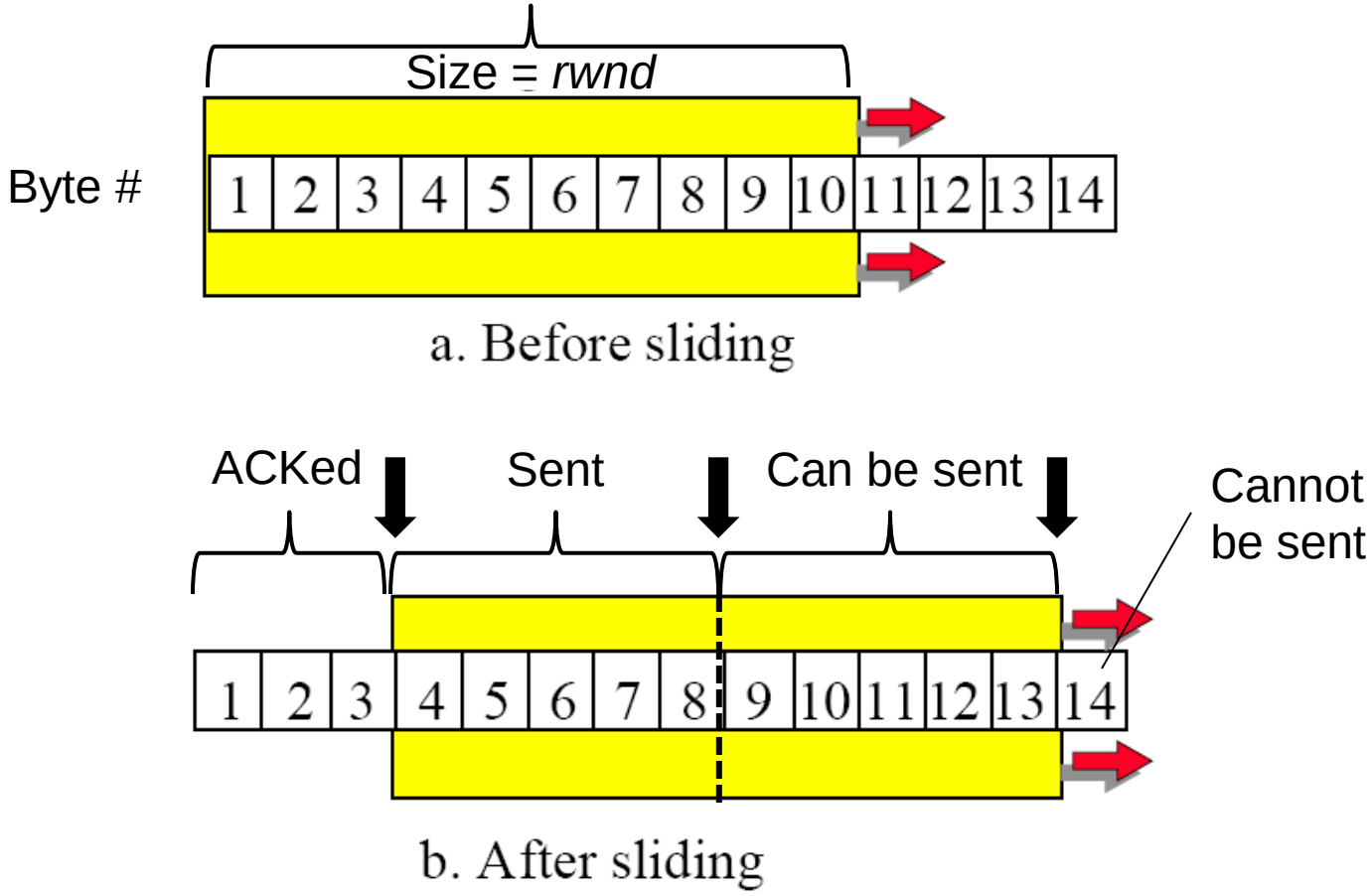
The other side also close



Reset means immediate close
(Initiated from any end)



Sliding Window



NB) Makes transmission more efficient

Congestion

High load →

Buffers get full →

Packets dropped →

Higher RTT estimates & retransmissions →

Even higher load →

Congestion collapse

Versions of TCP

- **TCP Tahoe** (1988, FreeBSD 4.3 Tahoe)
  - Slow Start
  - Congestion Avoidance
- **TCP Reno** (1990, FreeBSD 4.3 Reno)
  - Fast Retransmit
- **NewReno** (1996)
- **SACK** (1996)

Slow Start

Use a congestion window, *cwnd* at the sender side and limit the sending to no more than **min**[*rwnd*, *cwnd*]

Exponential increase:

- *cwnd* = 1 MSS (Maximum Segment Size)
- Increase *cwnd* each time an ACK is received

Congestion Avoidance

Use a slow start threshold, *ssthresh* to change the exponential increase of the *cwnd*, to an additive increase.

Additive increase:

- When *cwnd* has reached *ssthresh*, increment by one after each RTT (all segments in the window have been ACKed)

Congestion Detection

After a timeout: (strong possibility of congestion)

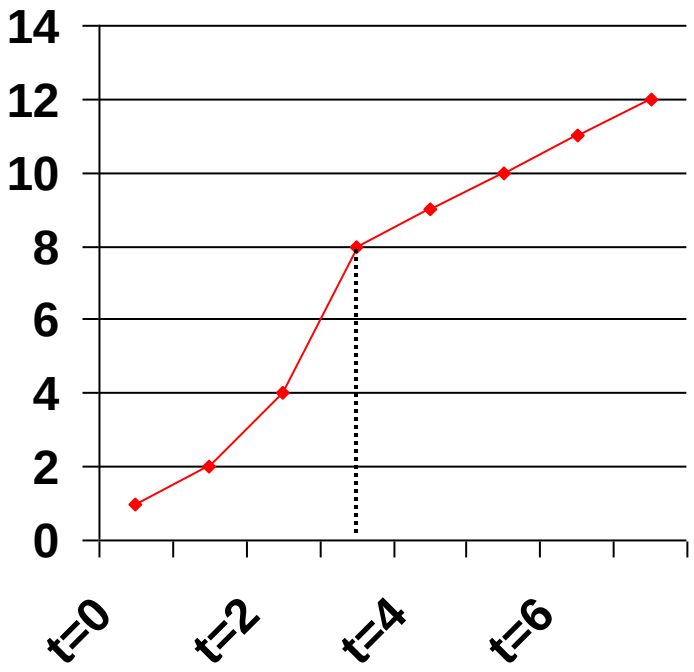
- Set the *ssthresh* to half of *cwnd* (“multiplicative decrease”)
- Set *cwnd* = 1 and use slow start
- For segments in allowed window (**min**[*rwnd*, *cwnd*]), back-off retransmission timer exponentially

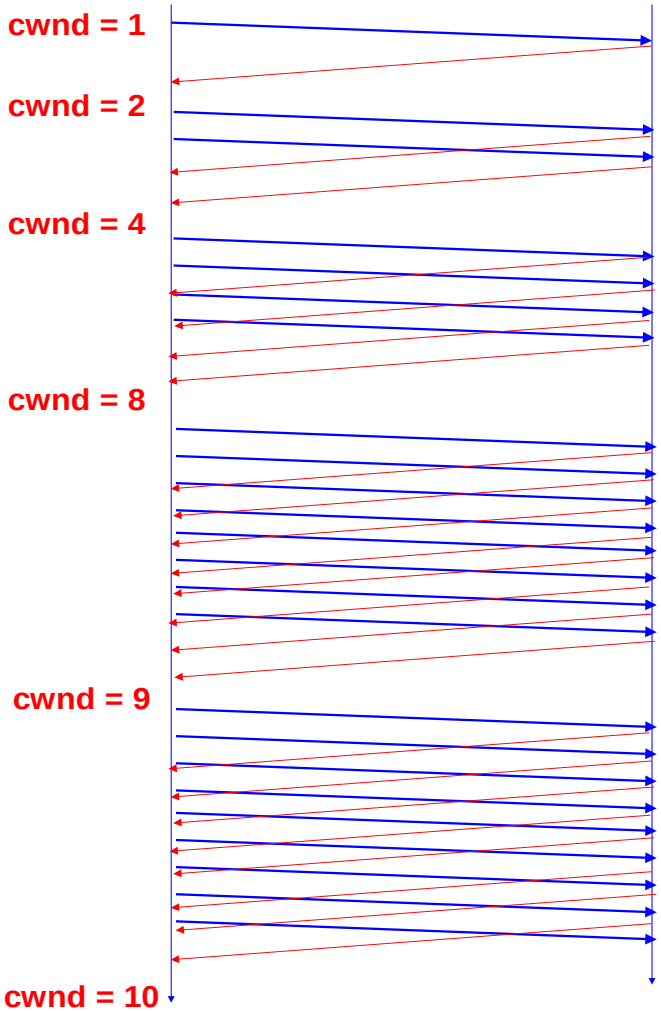
(weaker possibility of cong.)

- Set the *ssthresh* to half of *cwnd*
- Set *cwnd* = *ssthresh* and use additive increase

Example

Assume that *ssthresh = 8*





t≈3.5

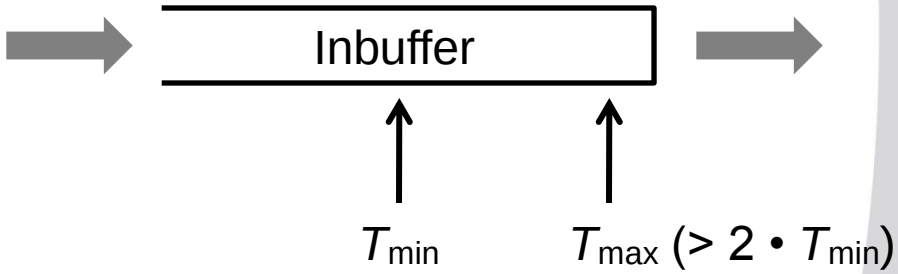
Tail-drop

Problem:

- When a router's inbuffer is full, datagrams are dropped
- Multiplexing means that if *N* packets are dropped, then it will affect up to *N* different TCP connections

Solution:

- Random Early Discard (RED)

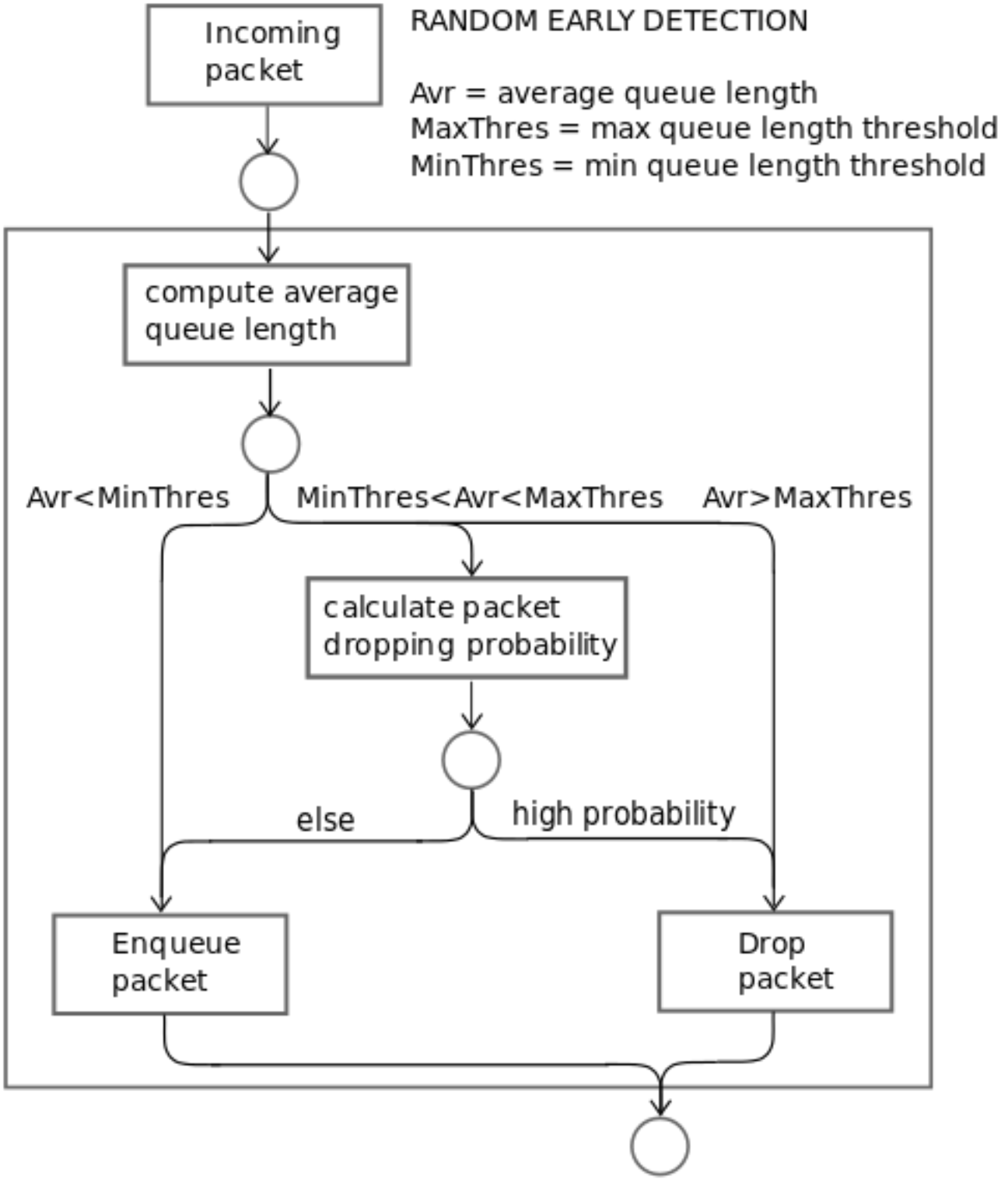


Randomly discard with probability *p*,

p=0 at *T*min and 1 at *T*max

p can also be set according to a weighted average of the queue size

(to better cope for bursty traffic)


Incoming packet
RANDOM EARLY DETECTION
Avr = average queue length
MaxThres = max queue length threshold
MinThres = min queue length threshold
compute average queue length
Avr<MinThres
MinThres<Avr<MaxThres
Avr>MaxThres
calculate packet dropping probability
else
high probability
Enqueue packet
Drop packet