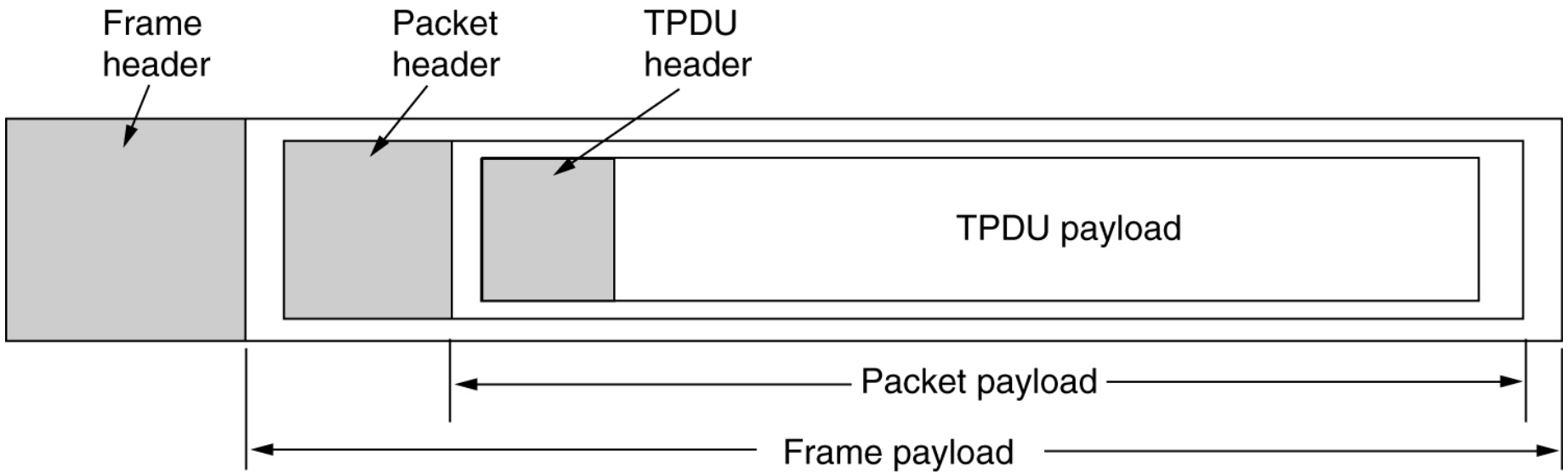
The Transport Layer

UDP & TCP

http://en.wikipedia.org/wiki/User_Datagram_Protocol

http://en.wikipedia.org/wiki/Transmission_Control_Protocol

Transport Service Primitives

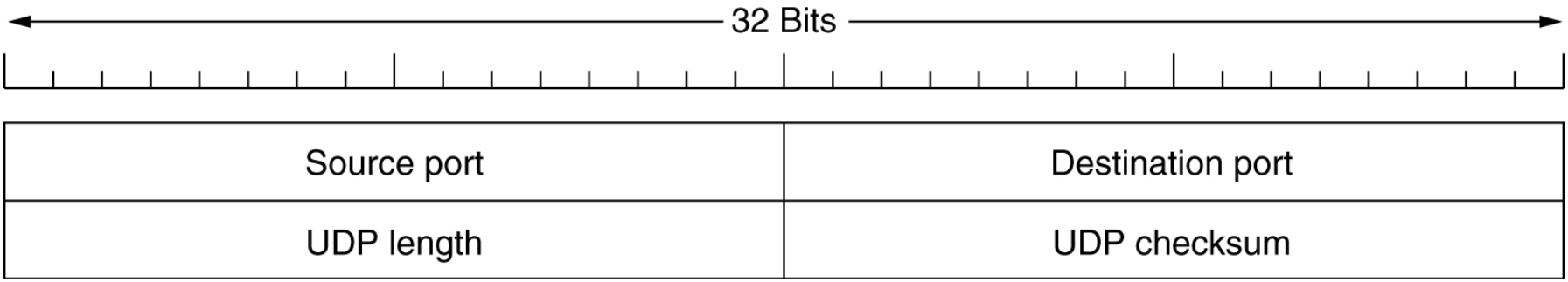


The nesting of TPDUs, packets, and frames.

Introduction to UDP

- UDP (“User Datagram Protocol”)
  - Is essentially IP with a short header to allow addressing in the transport layer
  - It does not do flow control, error control or retransmission; higher layers must deal with problems.
  - Important for applications that want to control packet flow, error control, timming..., like:
    - Idempotent Client / Server communication
    - Remote Procedure Call (RPC)
    - Multimedia (video, voice,...)

Introduction to UDP



The UDP header.

Ver exemplo de UDP: no GNS3 (wireshark), DHCP e DNS; netstat -p udp

Introduction to TCP

- TCP main goals
  - ***Reliable delivery***: IP only provides best effort delivery which is not enough for many applications; TCP handles retransmissions.
  - ***Exactly-once delivery***: Delayed duplicates can show up in the receiver due to retransmissions; TCP discards duplicates.
  - ***In-order delivery***: Due to different network paths, packets may arrive out of order at the receiver; TCP feed applications with ordered segments.
  - ***Stream based transfer***: for an application, data sent and received via TCP is only a stream of bytes, no notion of segments, datagrams, packets or frames is present. All of this is made transparent to the application by TCP.
  - ***Full duplex transfer***: TCP allows data transfer in both directions at the same time.
  - ***Flow and congestion control***: TCP controls sender data rate based on the speed of the receiver and on the capacity of the network.
  - ***Connection orientation***: There is no notion of connection in IP. TCP introduces the concept of connection that must be splicitely started and ended.

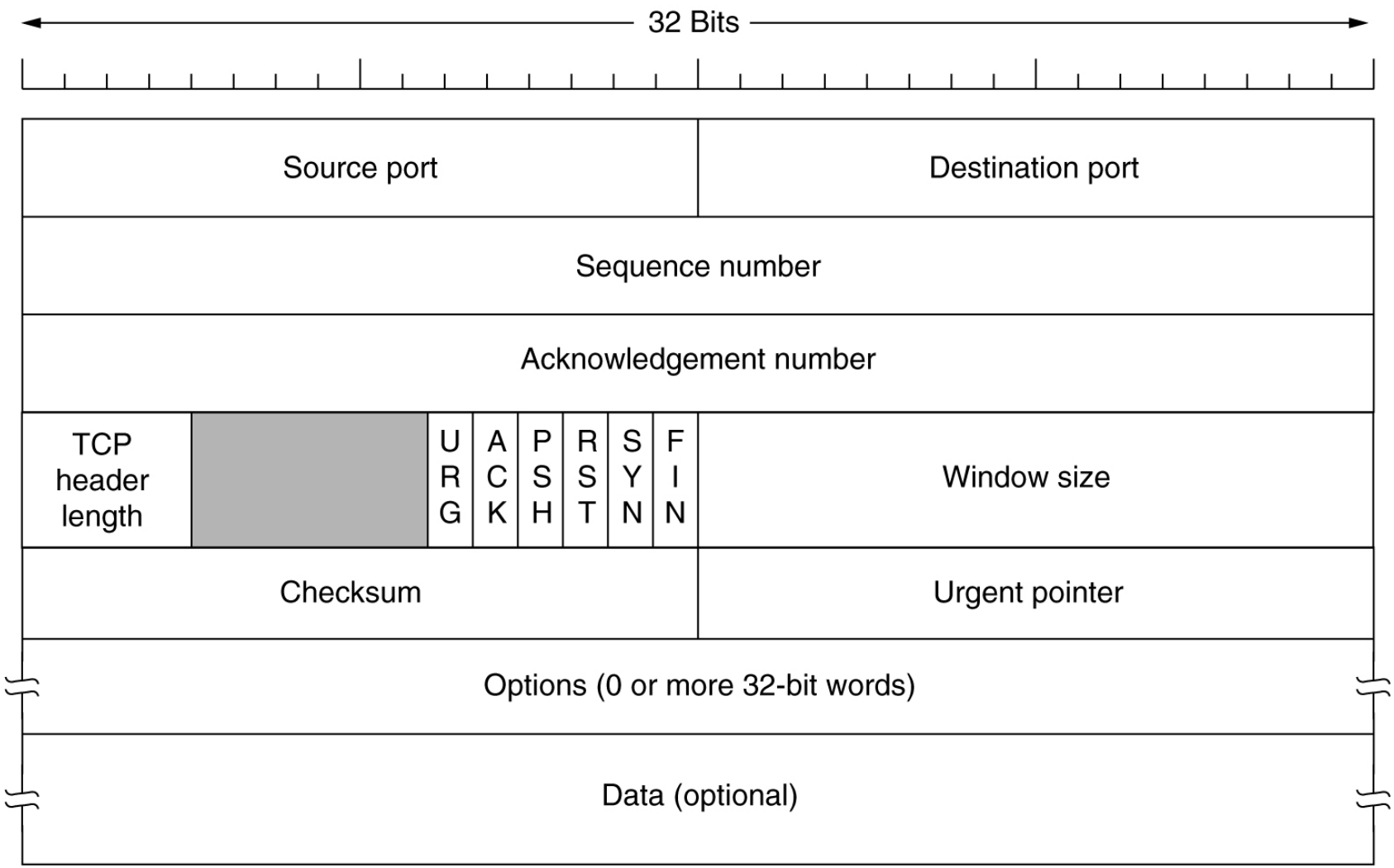
The TCP Service Model

Port numbers below 1024 are called “well known ports” and are reserved for standard services.

Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial File Transfer Protocol
79	Finger	Lookup info about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

Some assigned ports.

The TCP Segment Header



TCP Header.

The TCP Segment Header

- *Source port and destination port*
- *Sequence number* and *acknowledgement number* behave as usual (note that the latter reports the next byte expected...)
- *TCP header length* – tells how many 32 bits words are present in the header
- 6 bit field not used.
- *URG* – indicates a byte offset from the current sequence where urgent data are to be found.
- *ACK*– if it is 1 indicates that the acknowledgment number is valid; if it is 0 no ack is present in this segment.
- *PSH* – the receiver is asked to deliver data to the application upon arrival and not to buffer it.
- *RST* – Used to reset a connection that has become confused due to a host crash or some other reason. Also used to reject an invalid segment or refuse an attempt to open a connection.

The TCP Segment Header

- *SYN* – Used to establish connection. The connection request has SYN=1 and ACK=0 the reply has SYN=1 and ACK=1
- *FIN* – Used to end a connection.
- *Window size* – TCP uses a variable-sized slided window protocol. The window size tells how many bytes may be sended started at the byte acknowledged (a 0 size window is prefectly legal).
- *Checksum* – It checksums the header, the data and the conceptual pseudo-header presented next.

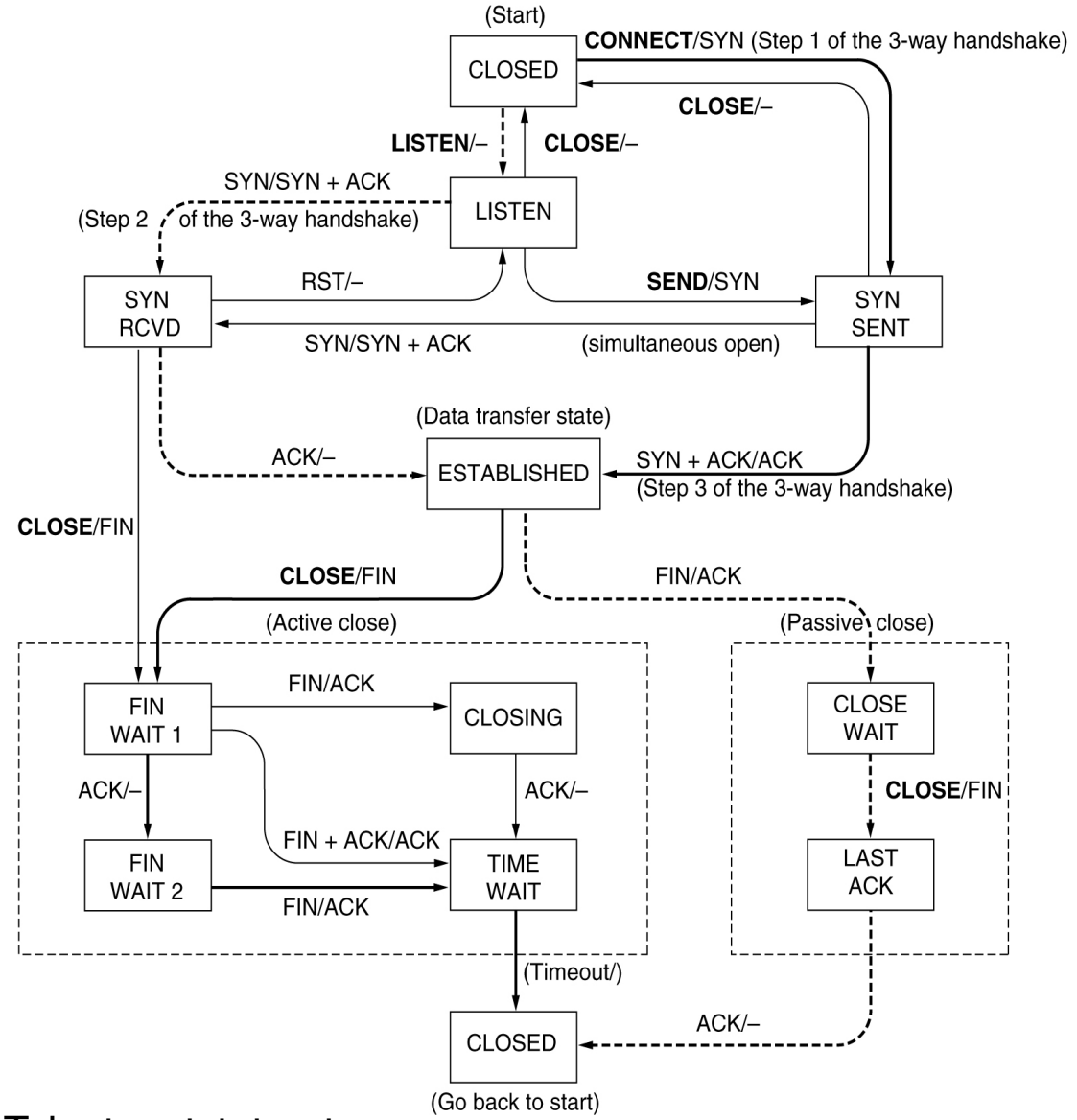
TCP Connection Management Modeling

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

The states used in the TCP connection management finite state machine.

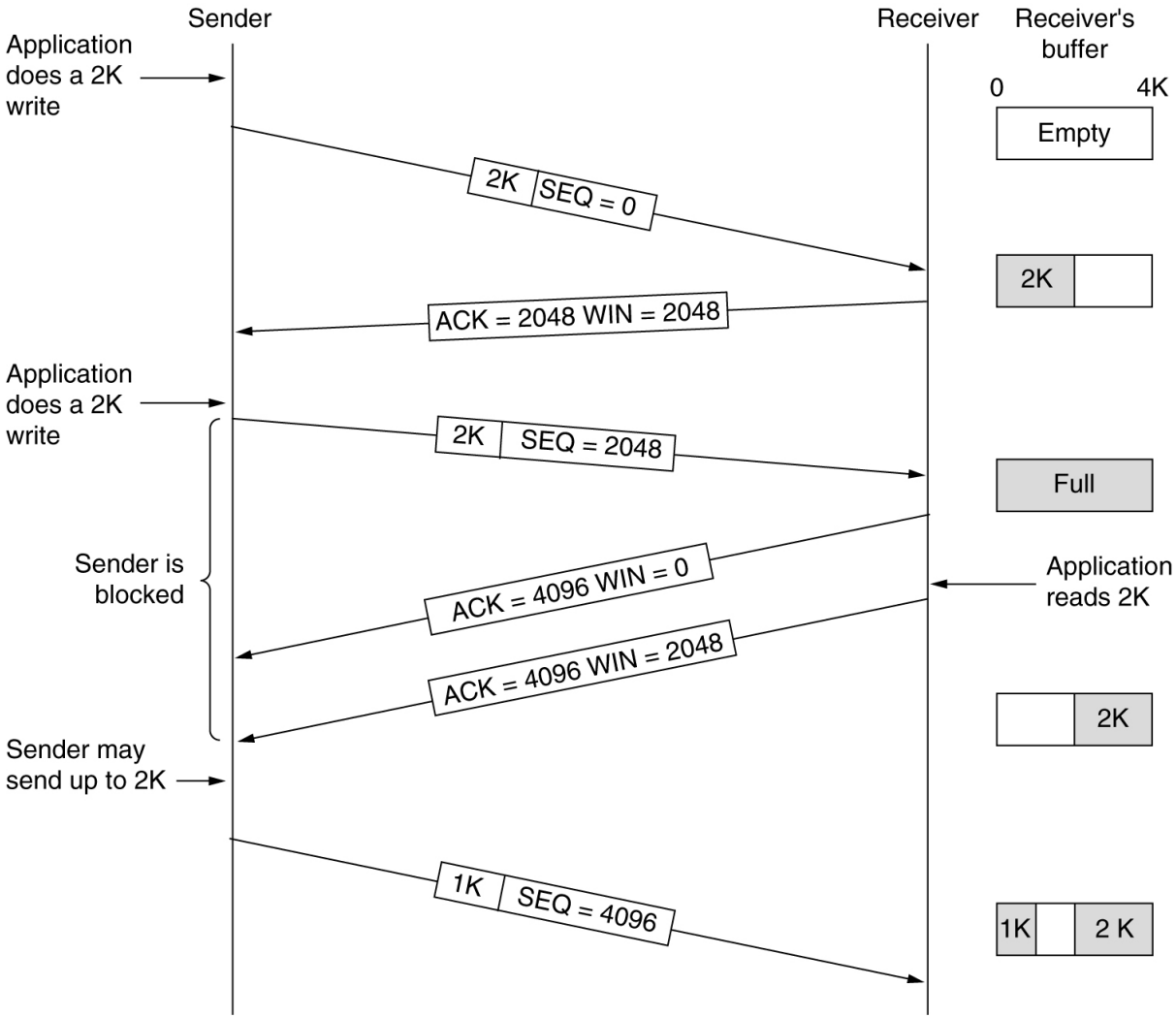
TCP Connection Management Modeling (2)

TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. Each transition is labeled by the event causing it and the action resulting from it, separated by a slash.



Ver exemplo de TCP: no GNS3 (wireshark), Telnet; netstat -p tcp

TCP Transmission Policy

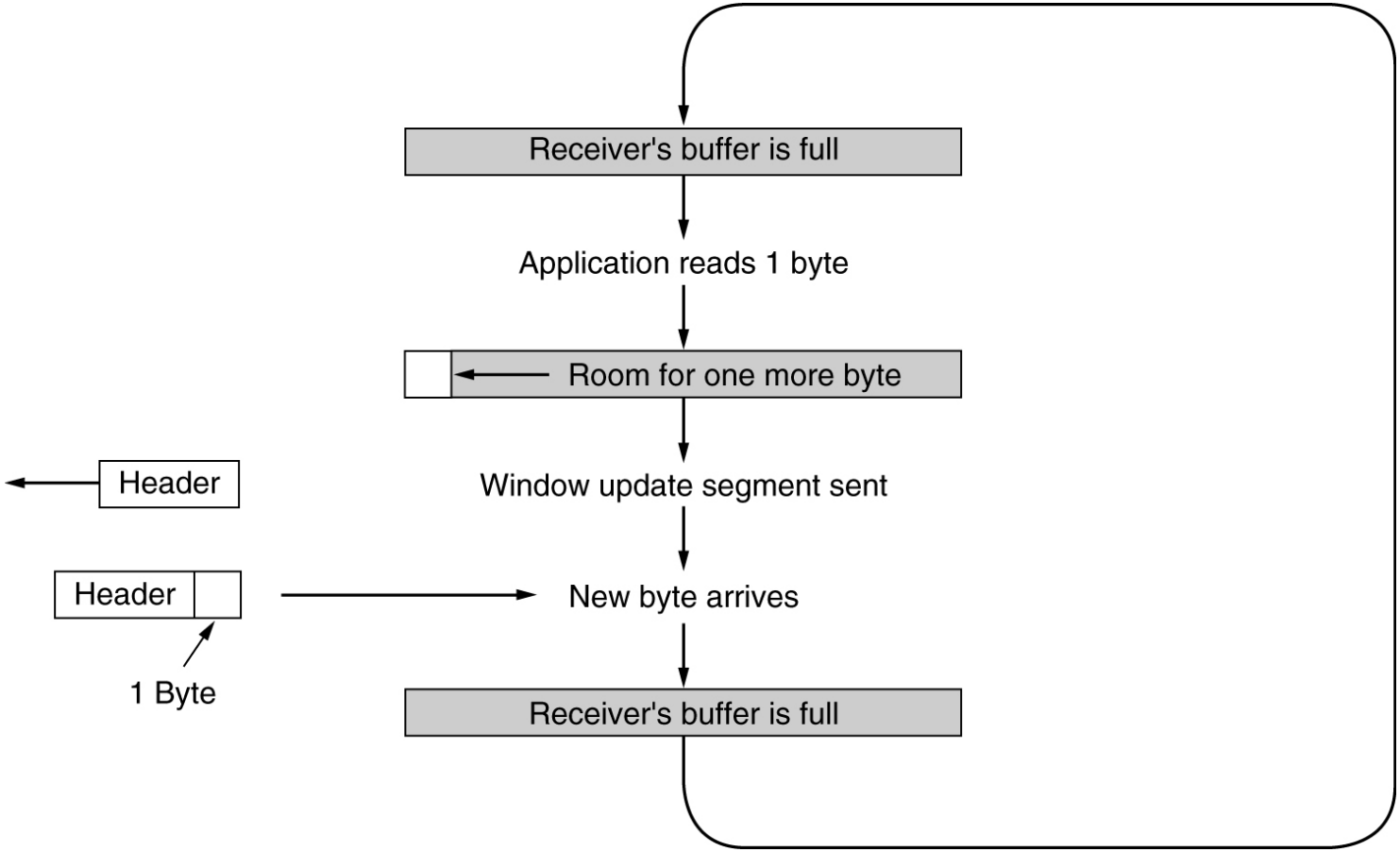


Window management in TCP.

TCP Transmission Policy

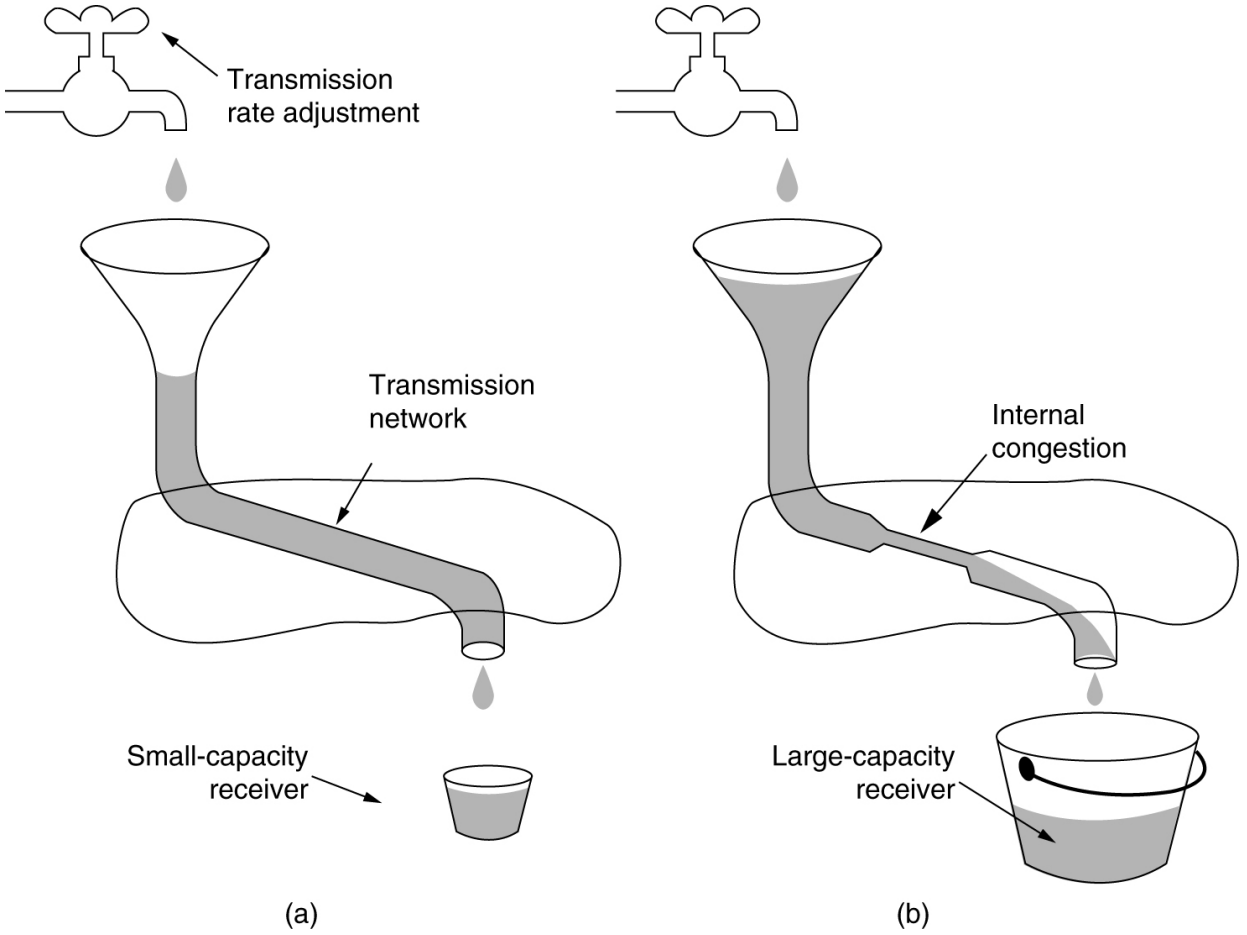
- In order to improve TCP performance several optimizations are possible:
  - Whenever possible delay acknowledgments and window updates in the hope of acquiring some data to hinch a free ride.
  - ***Nagle's algorithm***: solve the problem caused by the sending application delivering data to TCP one byte at a time. When data come into the sender one byte at a time, just send the first byte and buffer all the rest until the outstanding byte is acknowledged.Then send all the buffered characters in a TCP segment and start buffering again until they are acknowledged.
  - ***Clark's solution*** to the silly window sindrome: solve the problem caused by the receiving application getting the data up from TCP one byte at a time. Just wait until a decent amount of space may be advertised.
  - Both solutions needed: Sender does not send small fragments and the receiver does not ask for them.

TCP Transmission Policy



Silly window syndrome.

TCP Congestion Control



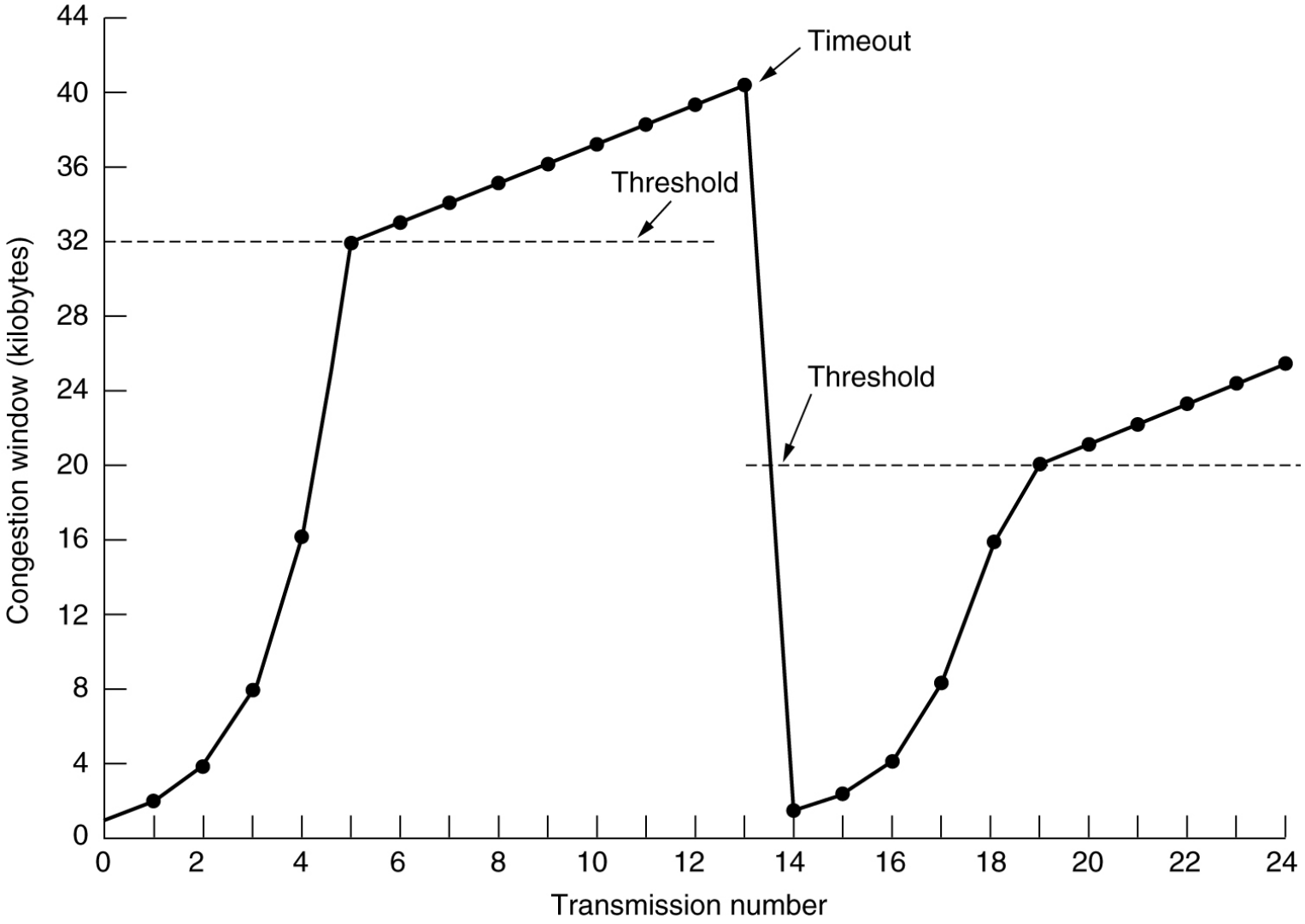
(a) A fast network feeding a low capacity receiver.

(b) A slow network feeding a high-capacity receiver.

TCP Congestion Control

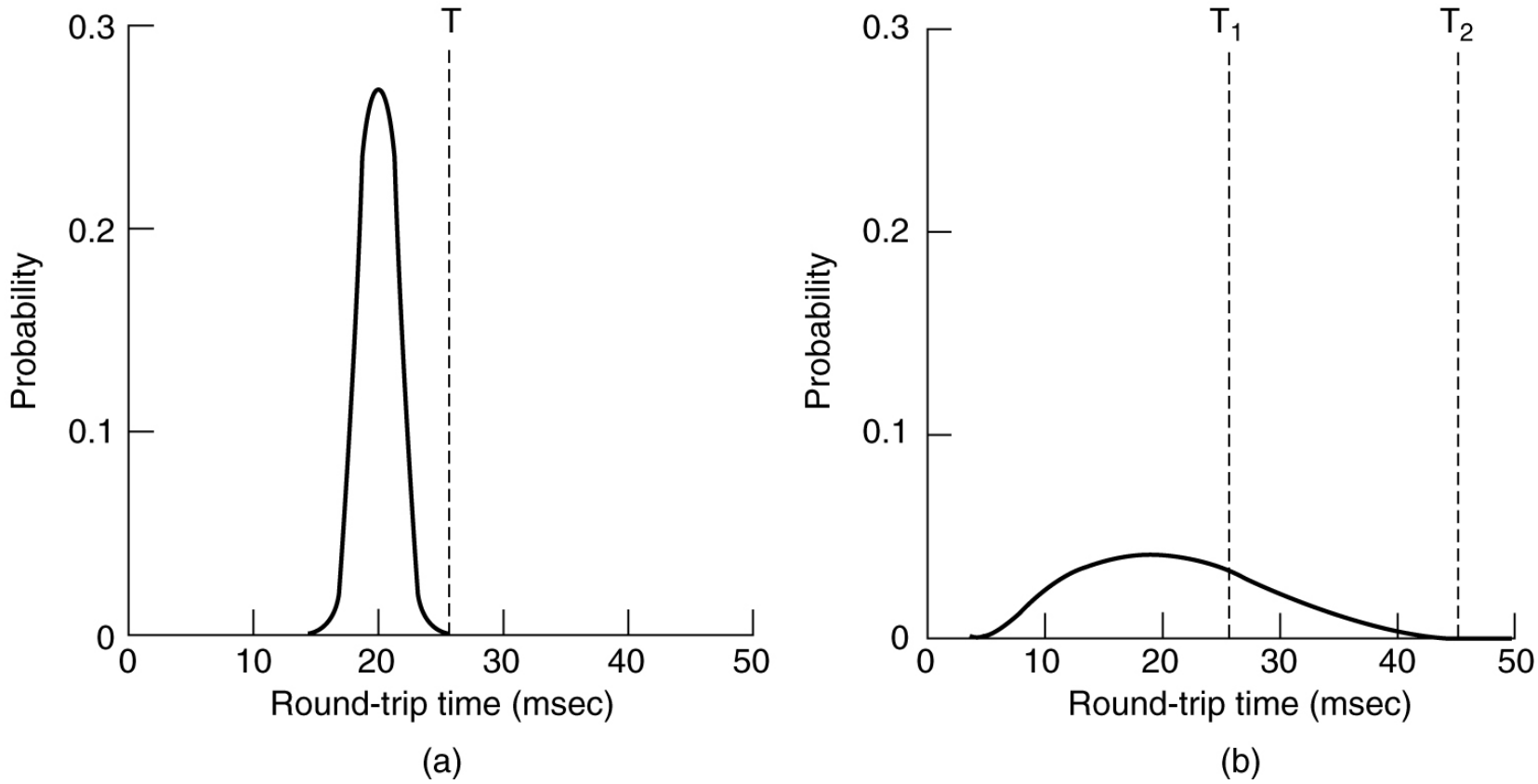
- TCP has a congestion window independent from the receiver window.
- TCP uses an algorithm for controlling the congestion window called: slow start – it is not slow at all it is exponential!
  - When the congestion window is *n* segments and all of them are acknowledged in time, the congestion window is increased by a byte count corresponding to *n* segments, doubling the congestion window;
  - The congestion window keeps growing exponentially until the receiver window is reached, **slow start threshold** is reached or a timeout occurs. If this last situation happens the congestion window size is reset to 1 and the slow start threshold is set to half of the current congestion window.
  - If no timeout happens, after hitting the slow start threshold the congestion window will increase linearly until the receiver window is reached.
  - See example: initial threshold=32k;

TCP Congestion Control



An example of the Internet congestion algorithm.

TCP Timer Management



(a) Probability density of ACK arrival times in the data link layer.

(b) Probability density of ACK arrival times for TCP.

TCP Timer Management

- How long should the sender wait before retransmit unacknowledged data ?
  - *Too soon* : duplicates; bad network performance
  - *Too late*: advertised window will not advance causing the sending host to limit its transmission rate unnecessarily.
  - TCP tries to solve this problem adapting the timeout used to trigger retransmissions depending on previously observed round trip time (RTT) and RTT variance between the two hosts.
  - Timeout = RTT + 4*(RTT variance)

TCP Timer Management

- Karn's Algorithm
  - The round trip time is estimated as the difference between the time that a segment was sent and the time that its acknowledgment was returned to the sender
  - When packets are re-transmitted there is an ambiguity: the acknowledgment may be a response to the first transmission of the segment or to a subsequent re-transmission.
  - Karn's Algorithm ignores retransmitted segments when updating the round trip time estimate.
  - ***Problem:***
    - If TCP never acknowledges retransmitted packets, the round trip estimate time will never be updated, and TCP will continue retransmitting segments without adjusting to the increased delay.
  - ***Solution:***
    - If the timer expires and causes a retransmission, TCP increases the timeout generally by a factor of 2. **new_timeout = 2 * timeout**

TCP Retransmission Policy

- Fast Retransmit
  - Receiver
    - TCP may generate an immediate acknowledgment (a duplicate ACK) when an out- of-order segment is received
    - This duplicate ACK should not be delayed
    - The purpose of this duplicate ACK is to let the other end know that a segment was received out of order, and to tell it what sequence number is expected
  - Sender
    - Since TCP does not know whether a duplicate ACK is caused by a lost segment or just a reordering of segments, it waits for a small number of duplicate ACKs to be received
    - If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost
    - TCP then performs a retransmission of what appears to be the missing segment, without waiting for a retransmission timer to expire.

TCP Retransmission Policy

- Fast Recovery
  - Sender
    - After fast retransmit sends what appears to be the missing segment, congestion avoidance, but not slow start is performed
    - This is the fast recovery algorithm. It is an improvement that allows high throughput under moderate congestion, especially for large windows.
    - Since the receiver can only generate the duplicate ACK when another segment is received, that segment has left the network and is in the receiver's buffer.
    - That is, there is still data flowing between the two ends, and TCP does not want to reduce the flow abruptly by going into slow start.

TCP Retransmission Policy

- The fast retransmit and fast recovery algorithms are usually implemented together as follows:
  1. When the third duplicate ACK in a row is received, set *ssthresh* to one-half the current congestion window, *cwnd*, but no less than two segments. Retransmit the missing segment. Set *cwnd* to *ssthresh* plus 3 times the segment size. This inflates the congestion window by the number of segments that have left the network and which the other end has cached.
  2. Each time another duplicate ACK arrives, increment *cwnd* by the segment size. This inflates the congestion window for the additional segment that has left the network. Transmit a packet, if allowed by the new value of *cwnd*.
  3. When the next ACK arrives that acknowledges new data, set *cwnd* to *ssthresh* (the value set in step 1). This ACK should be the acknowledgment of the retransmission from step 1, one round-trip time after the retransmission. Additionally, this ACK should acknowledge all the intermediate segments sent between the lost packet and the receipt of the first duplicate ACK. This step is congestion avoidance, since TCP is down to one-half the rate it was at when the packet was lost.

Berkeley Sockets

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Transport connections are made using the Operating System sockets interface (Berkeley Sockets) – see tcp-server.c /tcp-client.c

Berkeley Sockets