Chapter 5
Transport Layer

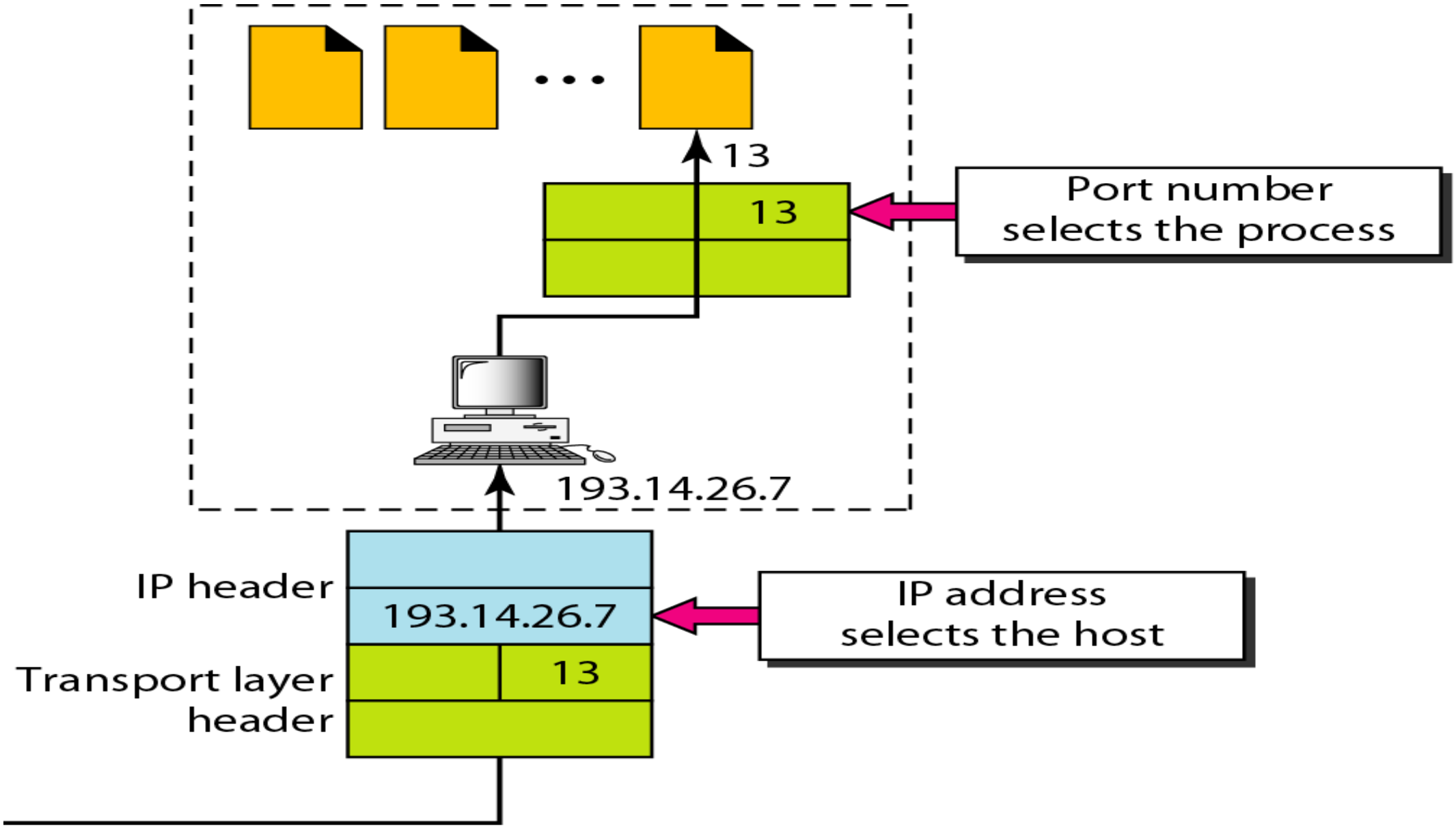
- ***Outline:***

  5.1 The transport service: Services provided to the upper layers
  5.2 Transport protocols: UDP, TCP
  5.3 Port and Socket
  5.4 Connection establishment, Connection release
  5.5 Flow control & buffering
  5.6 Multiplexing & de-multiplexing
  5.7 Congestion control algorithm: Token Bucket and Leaky Bucket

Transport Layer

- The **data link layer is responsible** for delivery of frames between two neighboring nodes over a link. This is called ***node-to-node delivery***.
- The **network layer is responsible** for delivery of datagrams between two hosts. This is called ***host-to-host delivery***.
- The **transport layer is responsible** for ***process-to-process delivery***-the delivery of a packet, part of a message, from one process to another.
- The basic function of the transport layer **is to accept data from above**, split it up **into smaller units** if need be, pass **these to the network layer**, and ensure that the **pieces all arrive correctly at the other end**.
- Furthermore, all this must be done efficiently and in a way that **isolates the upper layers from the inevitable changes** in the hardware technology.

IP addresses versus port numbers

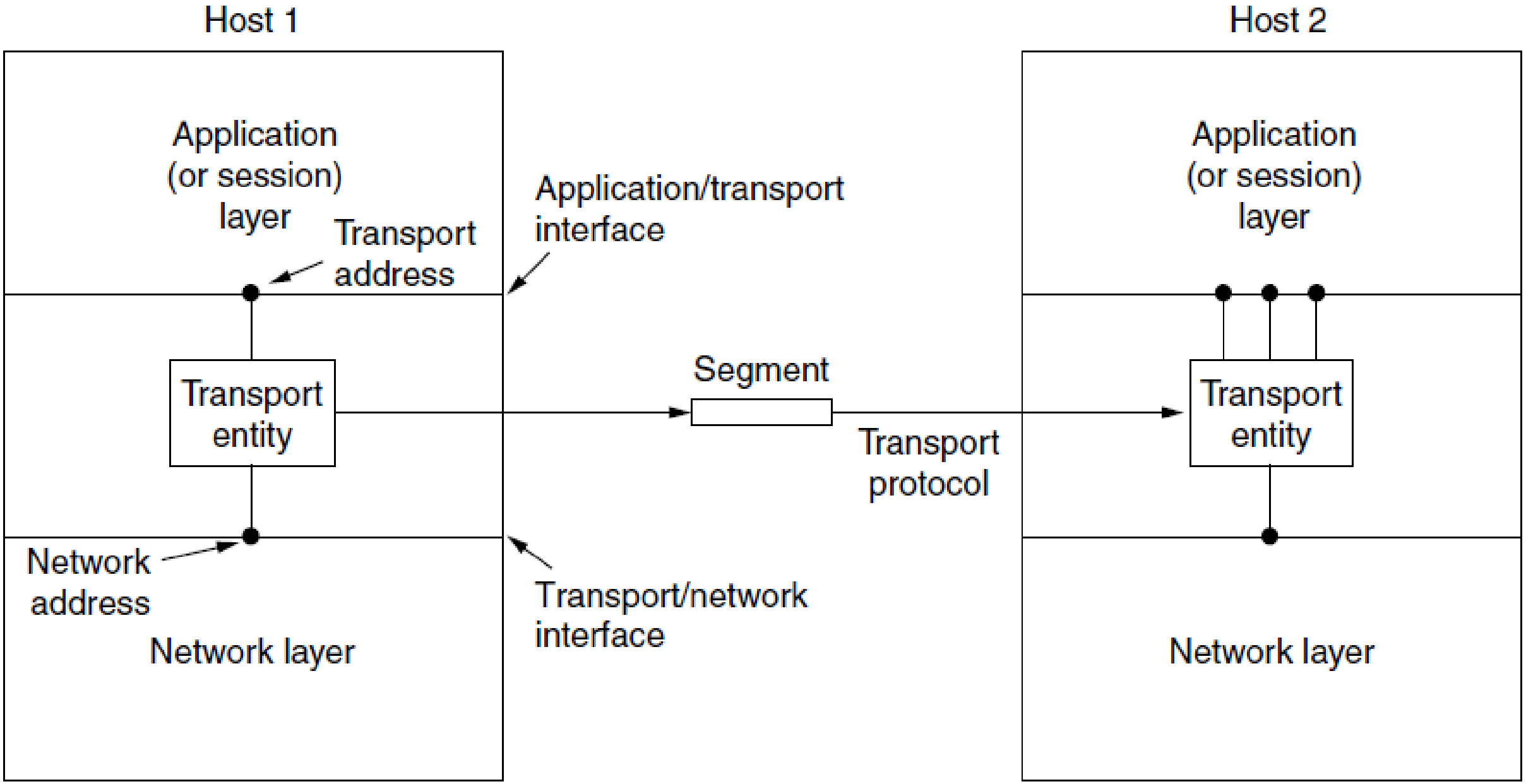


Functions of Transport Protocols

- Error Handling
- Flow Control
- Multiplexing
- Connection Set-up and Release
- Congestion Handling
- Segmentation and Reassembly
- Addressing

Services provided to the upper layers

- The **ultimate goal** of the transport layer is to provide **efficient, reliable, and cost-effective data transmission service** to its users.
- To achieve this, the transport layer **makes use of the services** provided **by the network layer.** The software and/or hardware within the transport layer that does the work is called the **transport entity**.
- There are two types of transport services:
  - ***connection-oriented***
  - ***connectionless***
- This is very similar to the network layer. ***Why bother having two?***
- The answer is subtle: ***network layer code runs on routers while transport layer code is run on user's machines.***



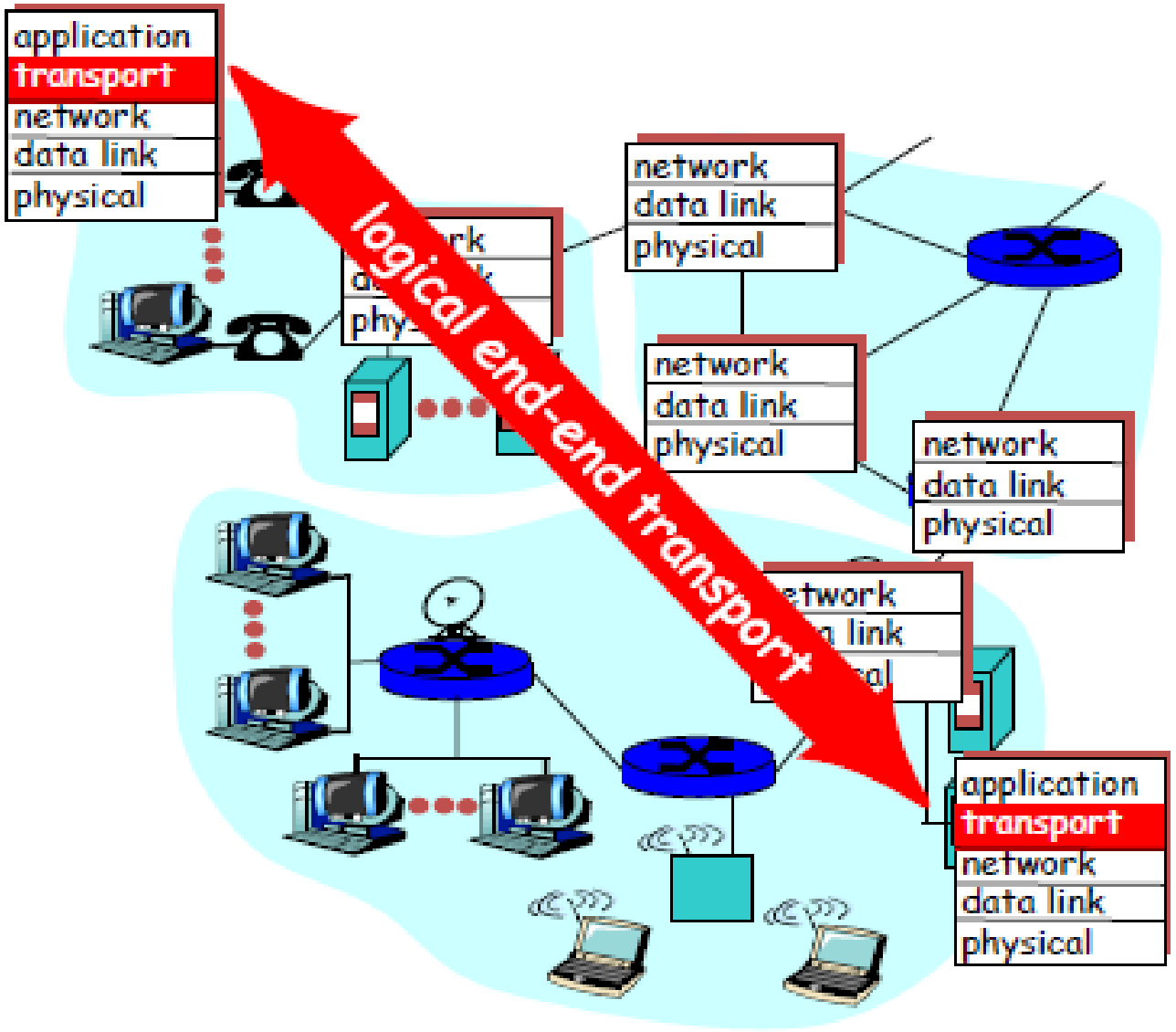
The (logical) relationship of the network, transport, and application layers

Transport Layer Protocol Services

- **Connection-oriented Services**
  - Connection-Oriented means that **when devices communicate**, they **perform handshaking to set up an end-to-end connection**.
  - Connection-Oriented systems can only work **in bi-directional communications** environments. **To negotiate** a connection, **both sides must** be able to **communicate with each other.** This will not work in a unidirectional environment. Since a connection needs to **be established, the service also becomes reliable** one.
  - Example: **TCP**
- **Connection-less Services**
  - A **connection less** transport service **does not require connection establishment** . Since there is **no establishment**, there should be **no termination also**.
  - This happens to be a **unreliable service** but much **more faster than connection oriented.**
  - Example: **UDP**

Transport Layer Protocol Services

- Provide *logical communication* between application processes running on different hosts
- Transport protocols run in end systems
  - **send side:** breaks application messages into segments, passes to network layer
  - **receive side**: reassembles segments into messages, passes to application layer



Transport Control Protocol (TCP)

TCP Protocol Functions

- Multiplexing
- Error Handling
- Flow Control
- Congestion Handling
- Connection Set-up and release

TCP Transport Service

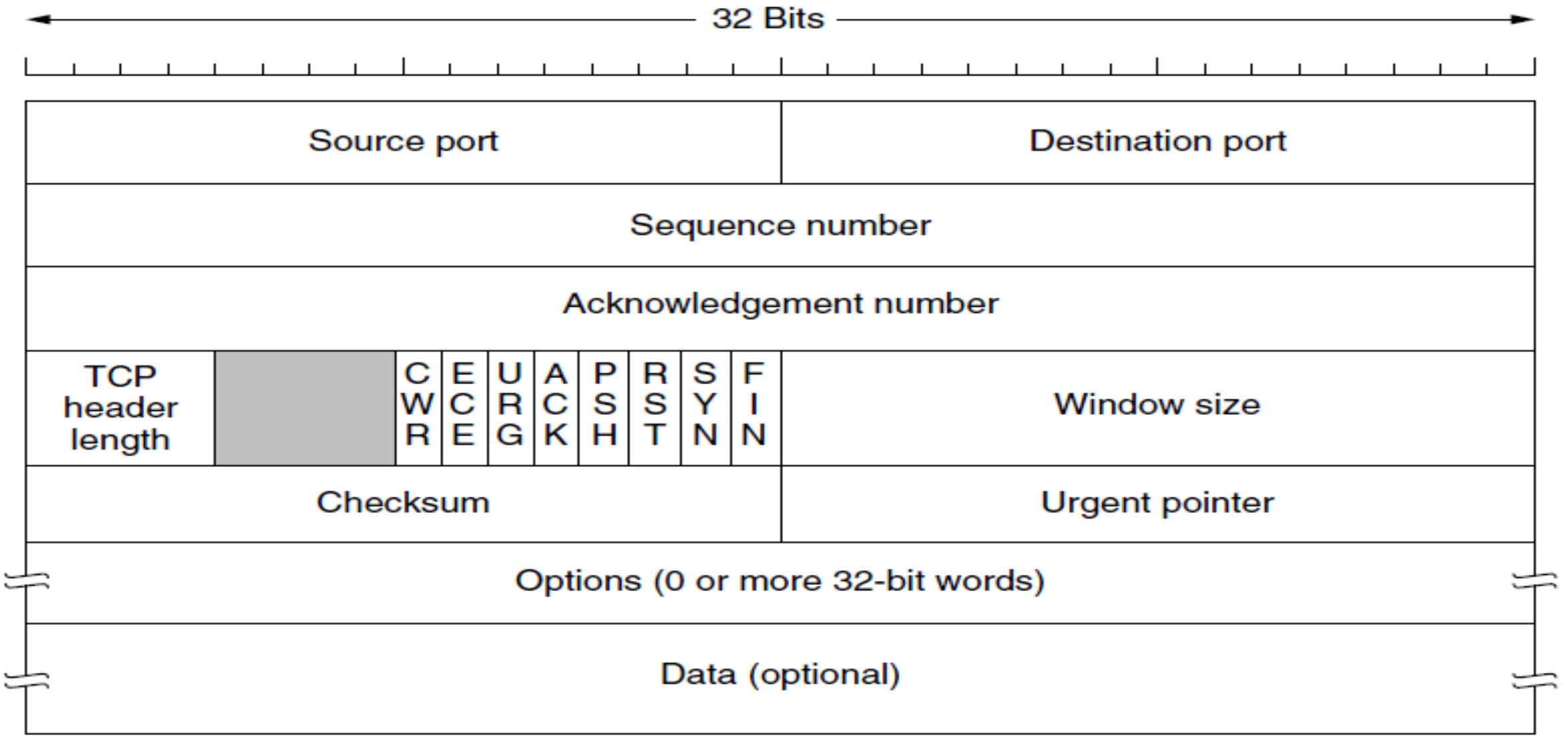
- Connection Oriented (full duplex point-to-point connection between processes).
- Reliable
- In-sequence segment delivery

- In TCP, connection-oriented transmission **requires three phases**: *connection establishment, data transfer, and connection termination*.
- TCP identifies connections on the basis of endpoints:
  - *IP address + port number*
  - ***Often written as**: IP-address : port-number, for instance: 130.89.17.3:80*

TCP

- Prior to data transmission, hosts establish a ***virtual connection*** via a synchronization process. The synch process is a **3-way "handshake"**, which **ensures both sides are ready to transfer data and determines the initial sequence numbers.**
- Sequence numbers are reference numbers between the two devices. Sequence numbers give hosts a way to acknowledge what they have received.
- TCP provides the following **major services to the upper protocol layers:**
  - **Connection-oriented data management** to assure the end-to-end transfer of data across the network(s).
  - **Reliable data transfer** to assure that all data is accurately received, in sequence and with no duplicates.
  - **Stream-oriented data transfer** takes place between the sender application and TCP and the receiving application and TCP.

The TCP Header



The TCP Header

- **Source Port:** This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.
- **Destination Port:** *This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.*
- **Source and Destination port are of each 16 bit. Number of port number = 2^{16} = 65535.**
- **0-1023 are well known port numbers, which is used by system to represent different services. e.g. port 25 for SMTP, port 80 for http, 443 for https 21 for ftp, 23 for telnet etc.**
- The **32 bit Sequence number and 32 bit Acknowledgement** *number fields are used for reliable data transfer between hosts.*

The TCP Header

- **Header length:** The *TCP header length* tells how many 32-bit words are contained in the TCP header. This information is needed because the *Options* field is of variable length, so the header is, too.
- **Reserved:** Six bits reserved for future use; must be zero.
- ***Six bit flags i.e. URG, ACK, PSH, RST, FIN and SYN.***
- The ***ACK*** bit is set to 1 to indicate that the ***Acknowledgement number*** is valid. If ***ACK* is 0**, the segment does not contain an acknowledgement, so the ***Acknowledgement number* field is ignored**.
- The ***SYN* bit is used to establish connections**. The connection request has ***SYN* = 1 and *ACK* = 0** to indicate that the **piggyback acknowledgement field is not in use**.
- In essence, the *SYN* bit is used to denote both *CONNECTION REQUEST and CONNECTION ACCEPTED*, with the *ACK bit* used to distinguish between those two possibilities.

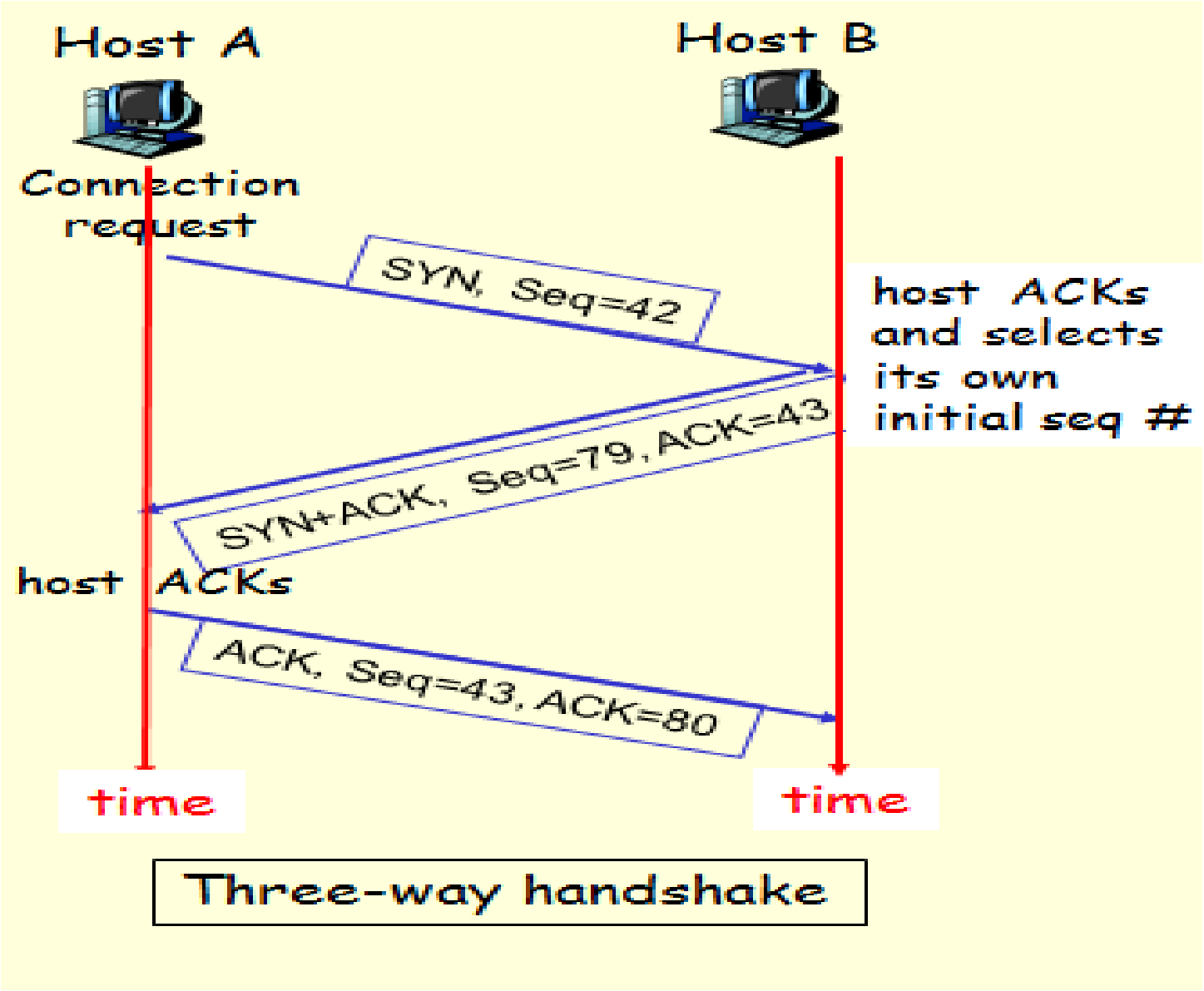
The TCP Header

- **RST:** Reset the connection.
- **FIN: Terminate the connection**. The FIN bit is used to release a connection. It specifies that the sender has no more data to transmit.
- **Window size :** Flow control in TCP is handled using a variable-sized sliding window. The **Window size** field tells how many bytes may be sent starting at the byte acknowledged.
- **URG:** *URG* is set to 1 if the ***Urgent pointer*** is in use. Sender can use URGENT flag to have TCP send data immediately and have the receiver TCP signal the receiver application that there is data to be read.
- **PSH:** The *PSH* bit indicates PUSHed data.

Connection establishment using three-way Handshaking

- **Step 1:** Client host sends TCP SYN segment to server specifies
  - a random initial seq. #
  - no data
- **Step 2:** Server host receives SYN, replies with SYNACK segment
  - server allocates buffers
  - specifies server initial seq. #
- **Step 3:** Client receives SYNACK, replies with ACK segment, which may contain data. *An ACK segment, if carrying no data, consumes no sequence number.*

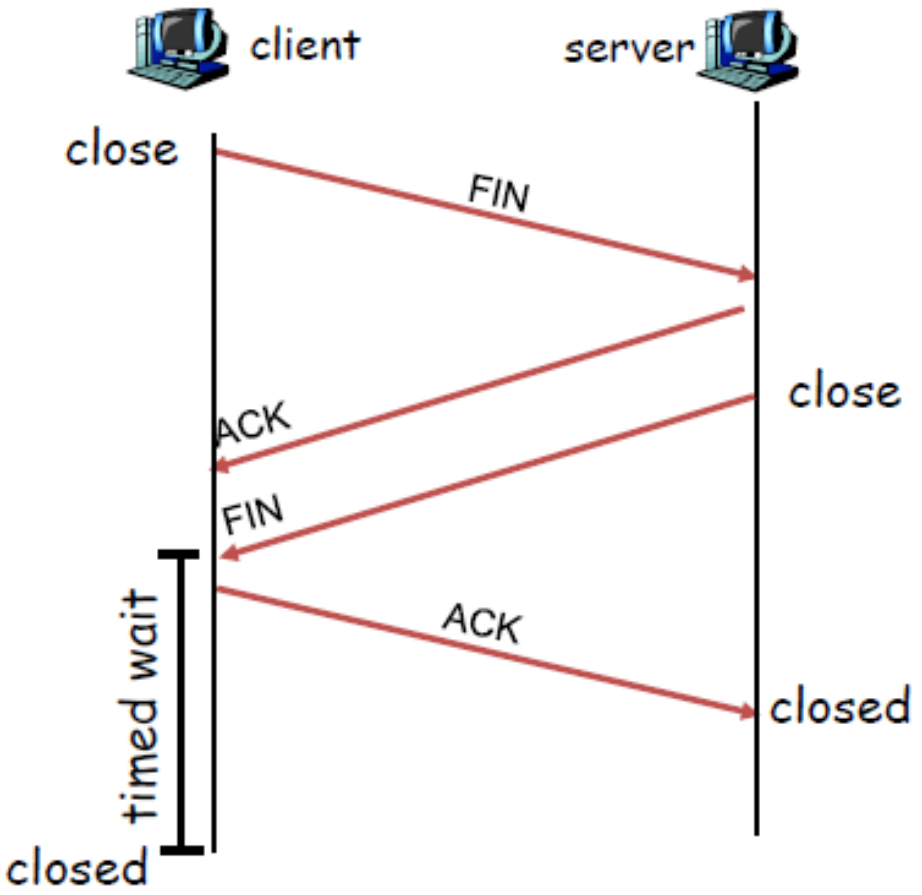
Connection establishment using three-way Handshaking



TCP Connection Termination

Connection Closing

- **Step 1:** client end system sends TCP FIN control segment to server.
- **Step 2:** server receives FIN, replies with ACK. Closes connection, sends FIN.
- **Step 3:** client receives FIN, replies with ACK. Enters “timed wait” - will respond with ACK to received FINs.
- **Step 4:** server, receives ACK. Connection closed.



User Datagram Protocol (UDP)

- UDP is a connectionless, unreliable transport protocol.
- No flow control. No ACK. of received packets. Only error control is if a checksum error is detected.
- ***The UDP protocol functions are:***
  - Multiplexing
  - Error Detection
- ***The UDP Service:***
  - Is a connectionless service
  - Is unreliable
  - Has no in-sequence delivery guarantees

Why Would Anyone Use UDP?

- *No delay for connection establishment*
  - UDP just blasts away without any formal preliminaries
  - avoids introducing any unnecessary delays
- *No connection state*
  - No allocation of buffers, parameters, sequence numbers, etc.
  - easier to handle many active clients at once
- *Small packet header overhead*
  - UDP header is only eight-bytes long

The UTP Header

- **Destination Port:** identifies destination process
- **Source Port:** It is optional – identifies source process for replies, or zero. The source port is primarily needed when a reply must be sent back to the source.
- **Message Length:** length of datagram in bytes, including header and data. The *UDP length* field includes the 8-byte header and the data. The minimum length is 8 bytes, to cover the header. The maximum length is 65,515 bytes.
- **Checksum:** The *UDP checksum* is optional and stored as 0 if not computed.

Source port	***Destination port***
UDP length	***UDP checksum***
Application Data (Message)	

TCP vs. UDP

UDP - User Datagram Protocol

- datagram oriented
- unreliable, connectionless
- simple
- unicast and multicast
- No windows or ACKs
- Smaller header, less overhead
- No Sequencing
- useful only for few applications, e.g., multimedia applications
- network management (SNMP), routing (RIP), naming (DNS), etc.

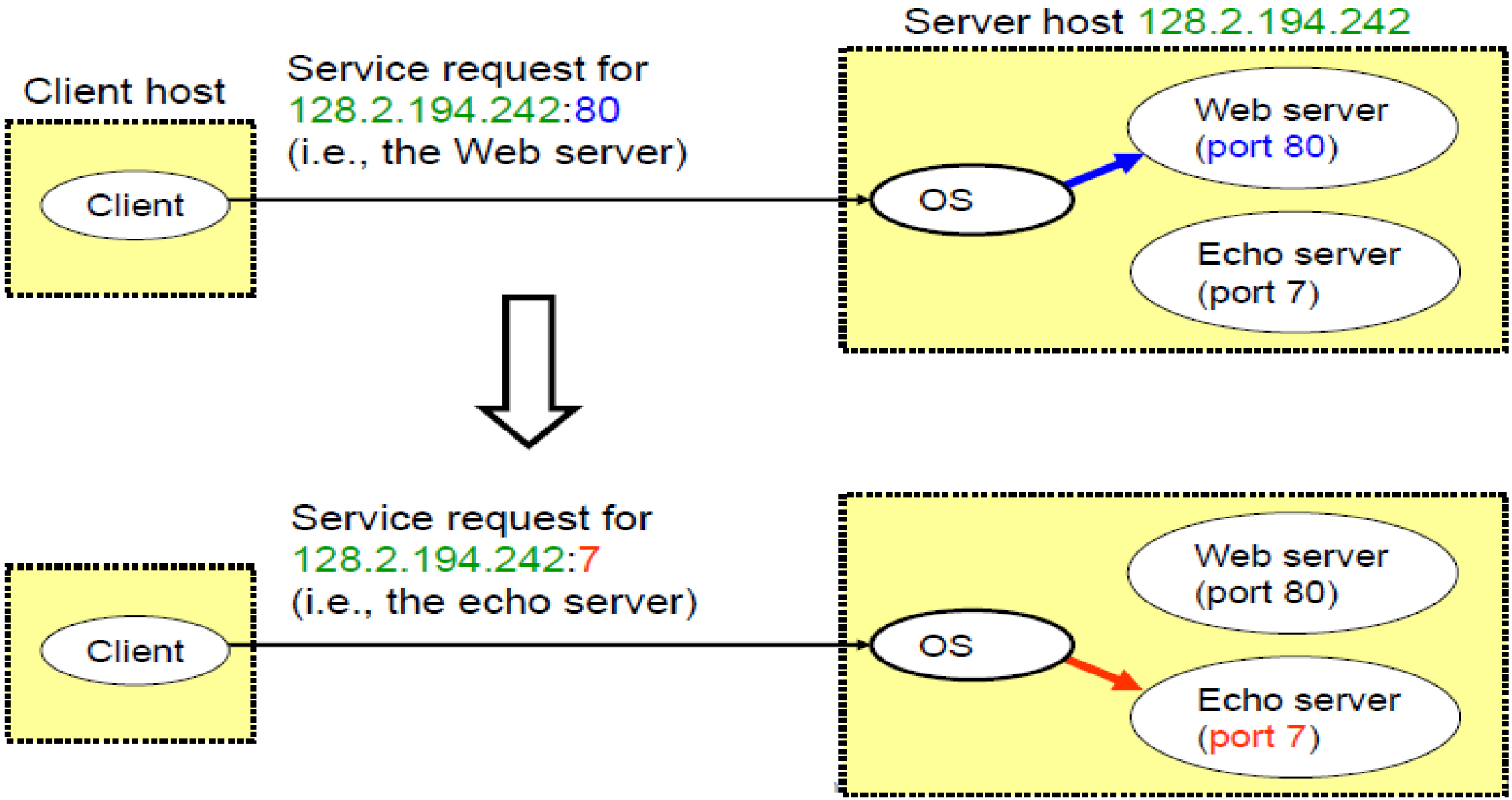
TCP - Transmission Control Protocol

- stream oriented
- reliable, connection-oriented
- complex
- only unicast
- Uses windows or ACKs
- Full header
- Sequencing
- used for most Internet applications
- web (HTTP), email (SMTP), file transfer (FTP), terminal (telnet), etc.

Port

- Port numbers are used to **keep track of different conversations** that cross the network at the same time.
- Port numbers **identify which upper layer service is needed**, and are needed when a host communicates with a server that uses multiple services.
- Ports are conceptual **“points of entry”** into a host computer.
- Servers **“listen on a port”** for connection attempts.
- Ports provide **one level of internet security**.
- **Both TCP and UDP** use port numbers **to pass to the upper layers**.
- Port numbers have the following **ranges**:
  - **0-255 used for public applications**, 0-1023 also called **well-known ports,** regulated by IANA**.**
  - Numbers from 255-1023 are assigned to marketable applications
  - 1024 through 49151 Registered Ports, not regulated.
  - 49152 through 65535 are Dynamic and/or Private Ports .

Using Ports to Identify Services



Socket

- Sockets **allow communication between two different processes** on the same or different machines.
- **To the kernel**, a socket is an **endpoint of communication**.
- **To an application**, a socket is **a file descriptor** that lets the application read/write from/to the network.
  - All Unix I/O devices, including networks, are modeled as files.
- Sockets are UNIQUELY identified by **Internet address, end-to-end protocol, and port number.**
- Clients and servers communicate with each by reading from and writing to socket descriptors.
- **Functions: Define an "end- point" for communication**, *Initiate and accept a connection*, **Send and receive data**, *Terminate a connection gracefully.*

Type of Socket

- **Socket are Two types.**
- **Stream socket :(connection- oriented socket)**
  - It provides reliable, connected networking service
  - If you send through the stream socket three items "A, B, C", they will arrive in the same order − "A, B, C".
  - These sockets use TCP for data transmission.
  - If delivery is impossible, the sender receives an error indicator.
  - applications: telnet/ ssh, http, …
- **Datagram socket :(connectionless socket)**
  - It provides unreliable, best- effort networking service.
  - Packets may be lost; may arrive out of order (uses UDP).
  - applications: streaming audio/ video (real-player).

Primitive	Meaning
SOCKET	Create a new communication endpoint
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
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

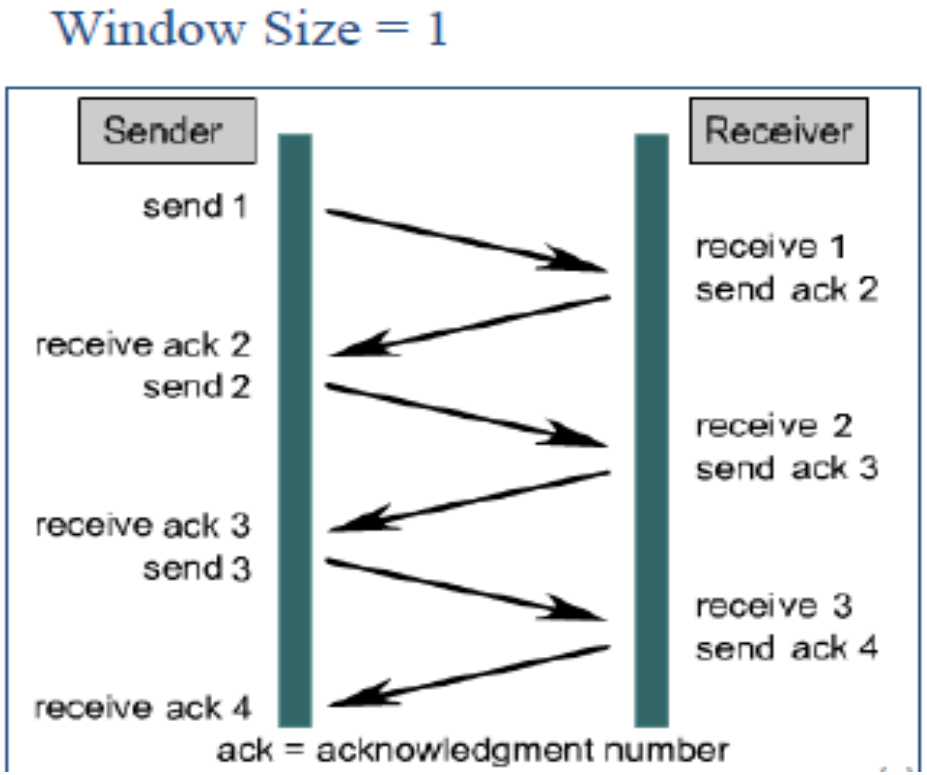
Figure 6-5. The socket primitives for TCP.

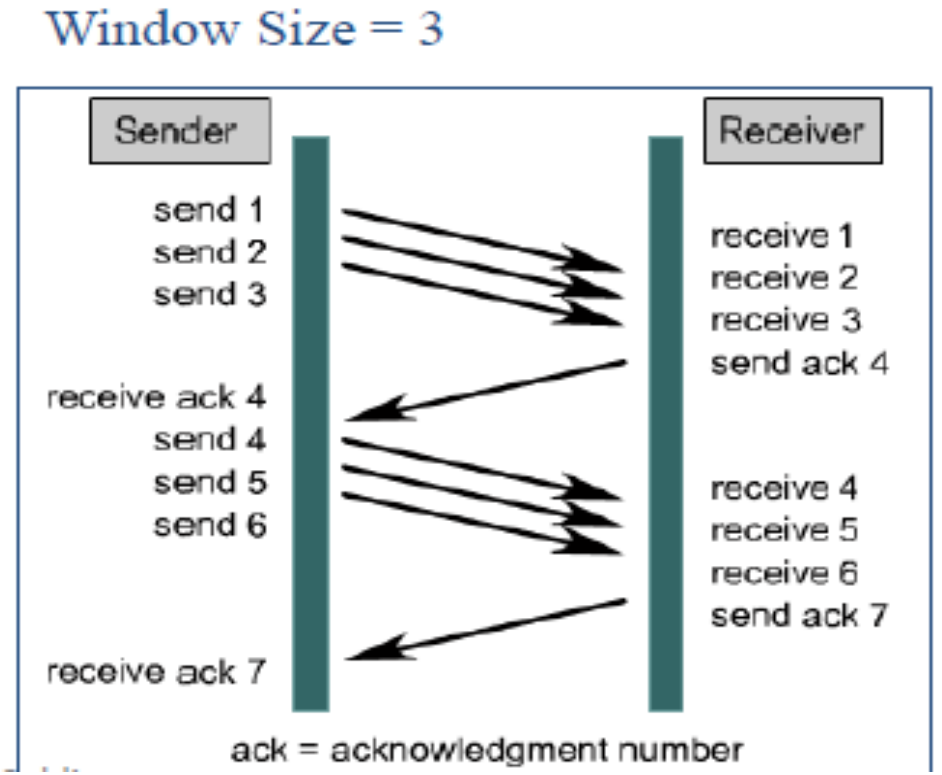
Flow Control and Buffering

- Flow control is **keeping a fast transmitte**r from **overrunning a slow receiver**.
- Flow control at this layer is **performed end-to-end** rather than across a single link.
- A **sliding window is used** to make data transmission more efficient as well as to **control the flow of data** so that the receiver does not become overwhelmed.
- The **buffers** are needed at both the **sender and the receiver**.
- **For low-bandwidth bursty traffic**, it is reasonable not to dedicate any buffers, but rather to acquire them dynamically at both ends, relying on buffering at the sender if segments must occasionally be discarded.
- On the other hand, **for file transfer and other high-bandwidth traffic**, it is better if the receiver does dedicate a full window of buffers, to allow the data to flow at maximum speed.

TCP Windows and Flow Control

- Data often is too large to be sent in a single segment.
- TCP splits the data into multiple segments.
- TCP provides flow control through “windowing” to set the pace of how much data is sent at a time
  - how many bytes per window, and how many windows between ACKs.





Flow Control and Buffering

- Even if the receiver **has agreed to do the buffering**, there still remains the **question** of **the buffer size**.
- If most **TPDUs are nearly the same size**, it is natural to organize the buffers as a pool of **identically-sized buffers**, with one TPDU per buffer, **as in Fig.(a).**
- However, if there is wide **variation in TPDU size**, from a few characters typed at a terminal to thousands of characters from file transfers, a pool of **fixed-sized buffers presents problems.**
- If the buffer size is **chosen** equal to the **largest possible TPDU**, **space** will be **wasted** whenever a **short TPDU arrives**.
- If the buffer size is chosen **less than the maximum TPDU size**, **multiple buffers will be needed for long TPDUs**, with the **attendant complexity**.

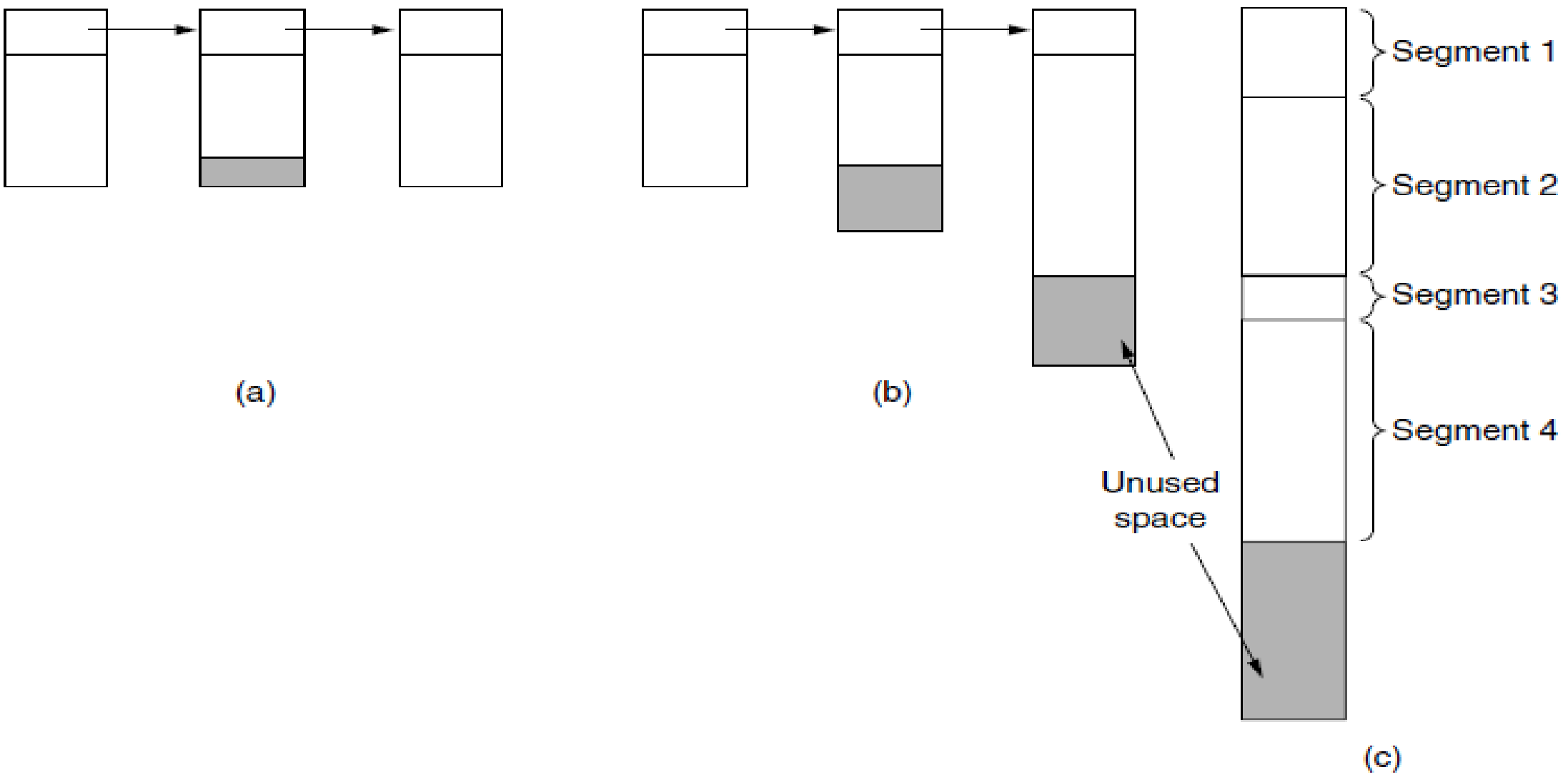


Figure 6-15. (a) Chained fixed-size buffers. (b) Chained variable-sized buffers. (c) One large circular buffer per connection.

Flow Control and Buffering

- **Another approach** to the buffer size problem is to use **variable-sized buffers, as in Fig. (b).**
- The advantage here is **better memory utilization**, at the price of more complicated buffer management.
- A third possibility is to **dedicate a single large circular buffer per connection**, as in Fig.(c).
- This system also makes **good use of memory**, provided that all connections are heavily loaded, but is poor if some connections are lightly loaded.

Congestion control algorithm

- When **too many packets present** in (a part of) the network causes packet delay and loss that degrades performance. This situation is called **congestion**.
- The **network and transport layers** share the **responsibility** for **handling congestion.**

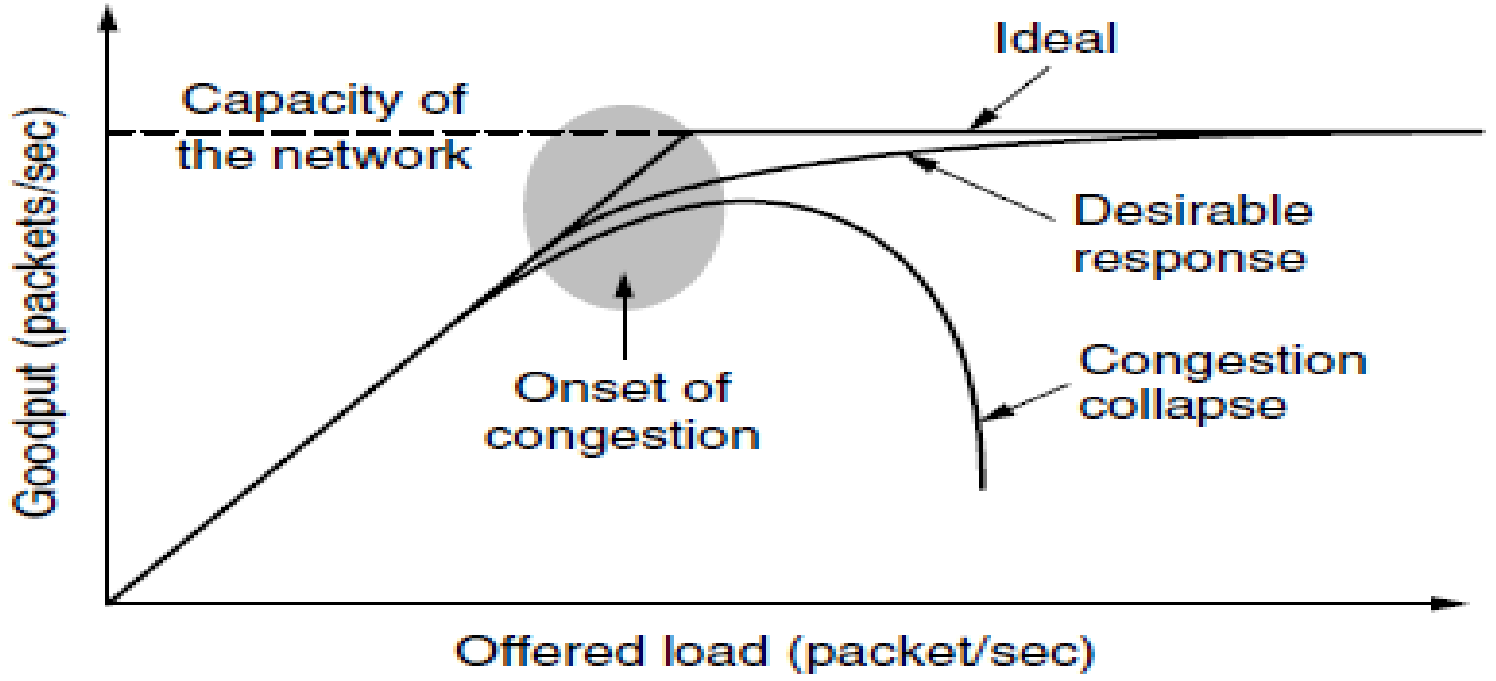


Figure 5-21. With too much traffic, performance drops sharply.

Congestion control algorithm

- If all of a sudden, **streams of packets begin arriving on three or four input lines** and all need the same output line, a **queue will build up**. If there is **insufficient memory** to hold all of them, **packets will be lost**.
- **Low-bandwidth links** or **routers that process packets more slowly** than the line rate can also become congested.
- If **router has no free buffers**, it must discard newly arriving packet. Sender may timeout and retransmits again and again increasing traffic.
- Routers have an **infinite amount of memory**, congestion **gets worse**, not better, because by **the time packets get to the front of the queue**, they have already **timed out** (repeatedly) and **duplicates have been sent**.

Difference between Flow Control and Congestion

- **Congestion control** has to do with making sure the subnet is able to carry the offered traffic. It is a **global issue**, involving the behavior of all the hosts, all the routers, the store-and-forwarding processing within the routers, and all the other factors that tend to diminish the carrying capacity of the subnet.
- **Flow control**, in contrast, relates to **the point-to-point traffic** between a given sender and a given receiver. Its job is to make sure that a fast sender cannot continually transmit data faster than the receiver is able to absorb it.
- **Flow control** frequently involves **some direct feedback from the receiver** to the sender to tell the sender how things are doing at the other end.

Approaches of Congestion Control

- The presence of congestion means that the **load is** (temporarily) **greater than the resources** (in a part of the network) can handle.
- Two **solutions come to mind**: ***increase the resources or decrease the load***.
- Solutions are usually **applied on different time scales** to either **prevent congestion or react to it** once it has occurred.

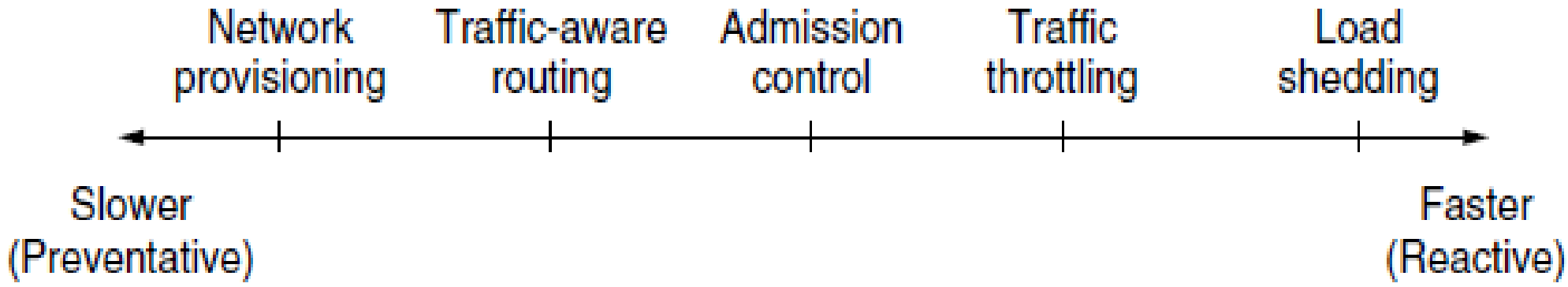


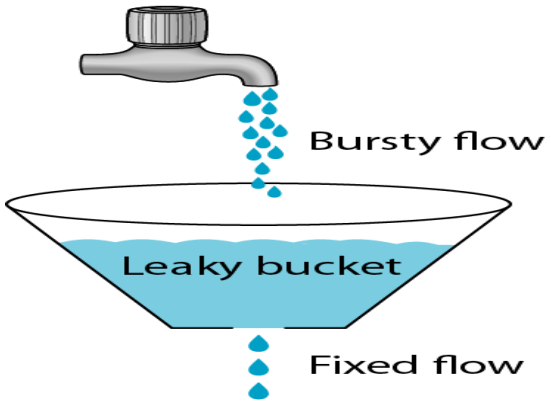
Figure 5-22. Timescales of approaches to congestion control.

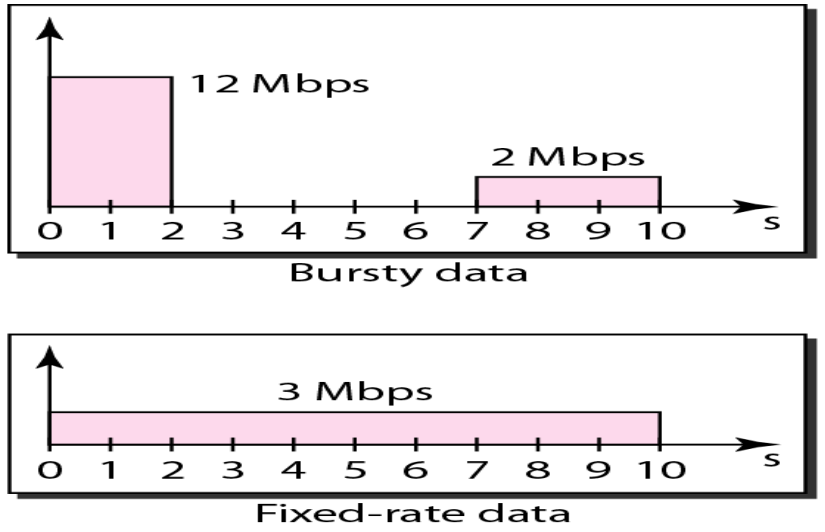
Approaches of Congestion Control

- Links and routers that are **regularly heavily utilized are upgraded** at the earliest opportunity. This is called **provisioning** and happens on a time scale of months, driven by long-term traffic trends.
- **Traffic-aware routing**: Splitting traffic across multiple paths.
- In a virtual-circuit network, **new connections can be refused** if they would **cause the network to become congested**. This is called **admission control**.
- When **congestion is imminent** the network can deliver **feedback to the sources** whose traffic flows are responsible for the problem. The network can request these sources **to throttle their traffic**, or it can **slow down the traffic itself.**
- **Load shedding:** the network is forced to discard packets that it cannot deliver.

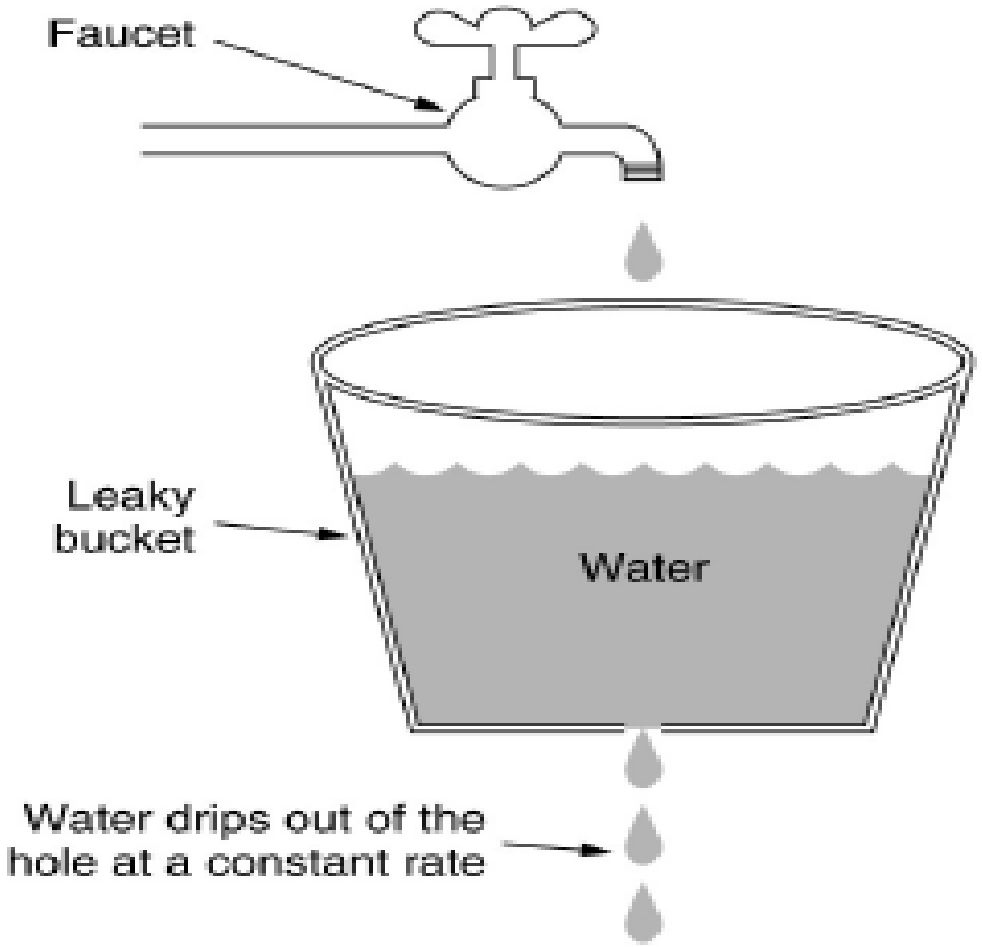
The Leaky Bucket Algorithm

- This arrangement can be built into the hardware interface or simulated by the host operating system. It was first proposed by Turner (1986) and is called the **leaky bucket algorithm**.
- The leaky bucket consists of a **finite queue**. When a packet arrives, if there is **room on the queue** it is **appended to the queue**; **otherwise, it is discarded**. At **every clock tick**, one **packet is transmitted** (unless the queue is empty).
- A leaky bucket algorithm **shapes bursty traffic into fixed-rate traffic** by averaging the data rate. It may **drop the packets if the bucket is full**.

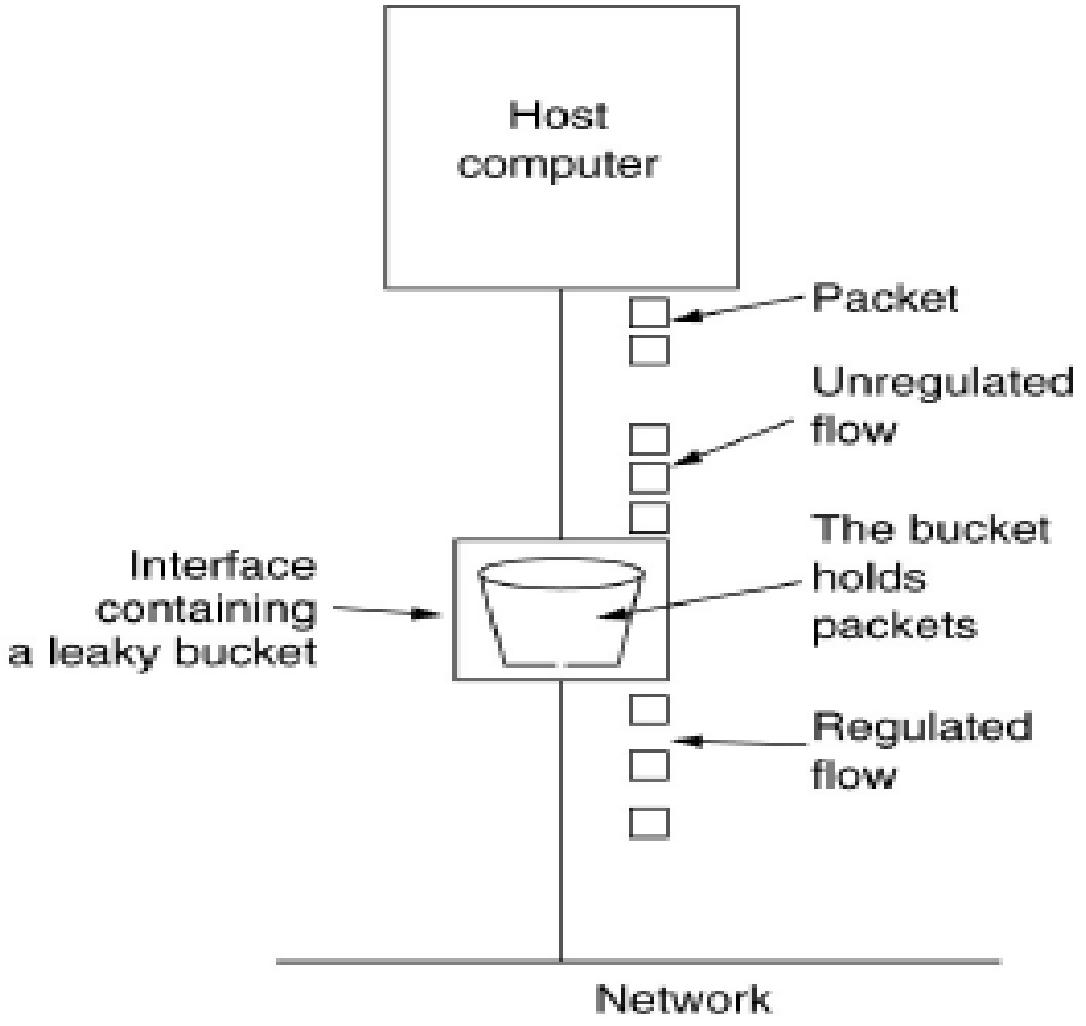




The Leaky Bucket Algorithm



(a) A leaky bucket with water.

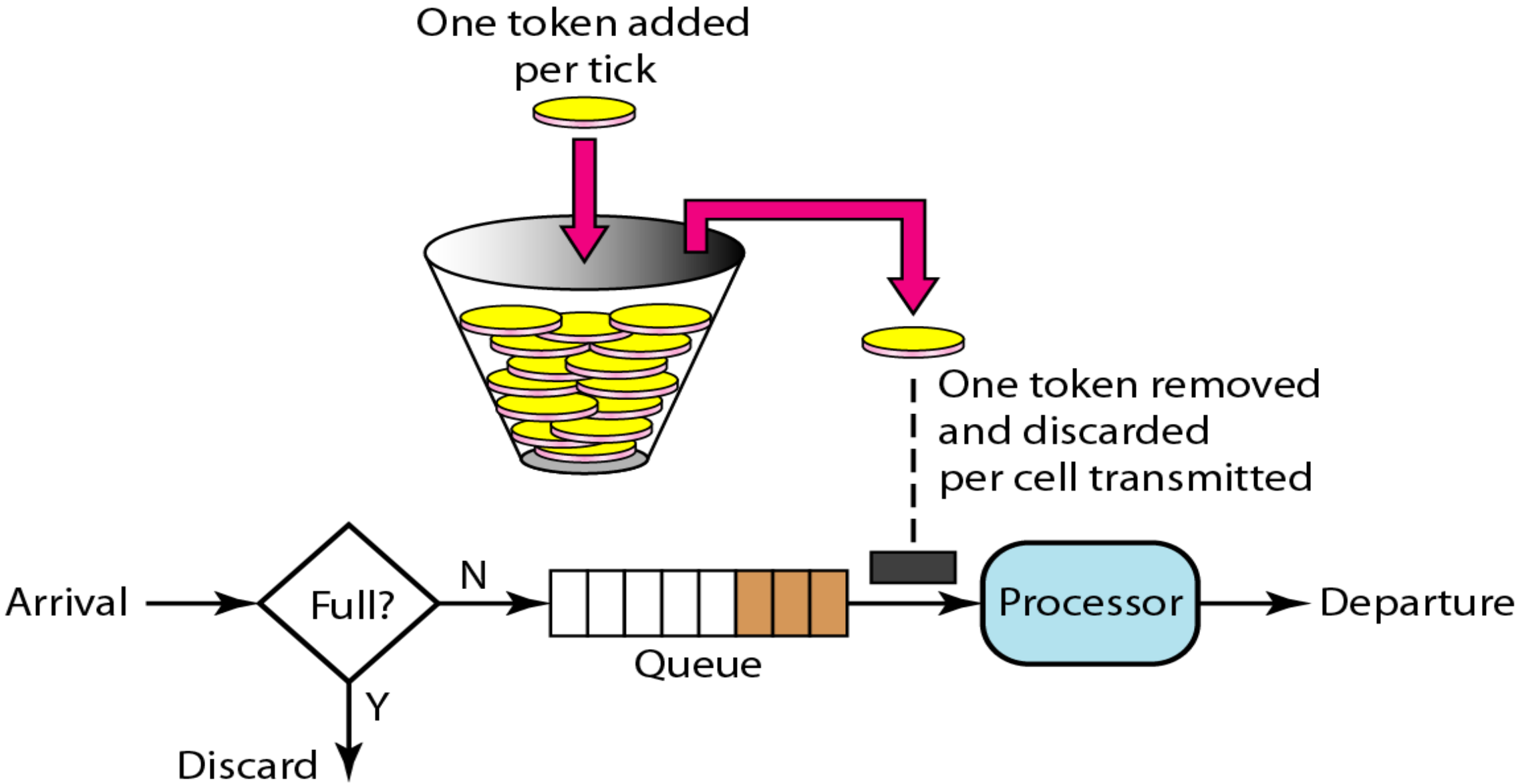


(b) A leaky bucket with packets

The Leaky Bucket Algorithm

- Try to imagine a bucket with a **small hole in the bottom**, as shown in figure.
- **No matter the rate** at which water enters the bucket, the **outflow is at a constant rate**, R, when there is any water in the bucket and zero when the bucket is empty.
- Also, once the bucket is **full to capacity**, any **additional water** entering it spills over the sides and is **lost**.
- The **same idea** can be **applied to packets**, as shown in Fig. (b). Conceptually, each host is **connected to the network** by an interface containing a **leaky bucket**, that is**, a finite internal queue**.
- If a **packet arrives** when the **bucket is full**, the packet must **either be queued** until enough water **leaks out to hold it or be discarded.**

The Token Bucket Algorithm



The Token Bucket Algorithm

- For many applications, it is better to allow the output to speed up somewhat when large bursts arrive, so a **more flexible algorithm** is needed, preferably one that never loses data. One such algorithm is **the token bucket algorithm**
- ***Tokens arrive at constant rate in the token bucket***. If bucket is full, tokens are discarded. A packet from the buffer **can be taken out** only if **a token in the token bucket** can be drawn.
- The token bucket algorithm does allow saving, up to the maximum size of the bucket, n. This property means that bursts of up to n packets can be sent at once, allowing some burstiness in the output stream and giving faster response to sudden bursts of input.

The Token Bucket Algorithm

- **For example**, if a host is **not sending for a while**, its bucket **becomes empty**. Now if the host has **bursty data**, the leaky bucket allows only **an average rate. The time when the host was idle is not taken into account.**
- On the other hand, the token bucket algorithm **allows idle hosts to accumulate credit** for the future in the form of tokens.
- For **each tick of the clock**, the system **sends *n* tokens to the bucket**. The system removes one token for every cell (or byte) of data sent.
- For example, if *n* is 100 and the host is idle for 100 ticks, the bucket collects 10,000 tokens.
- Now the host can consume all these tokens in one tick with 10,000 cells, or the host takes 1000 ticks with 10 cells per tick.
- In other words, the host **can send bursty data as long as the bucket is not empty.**

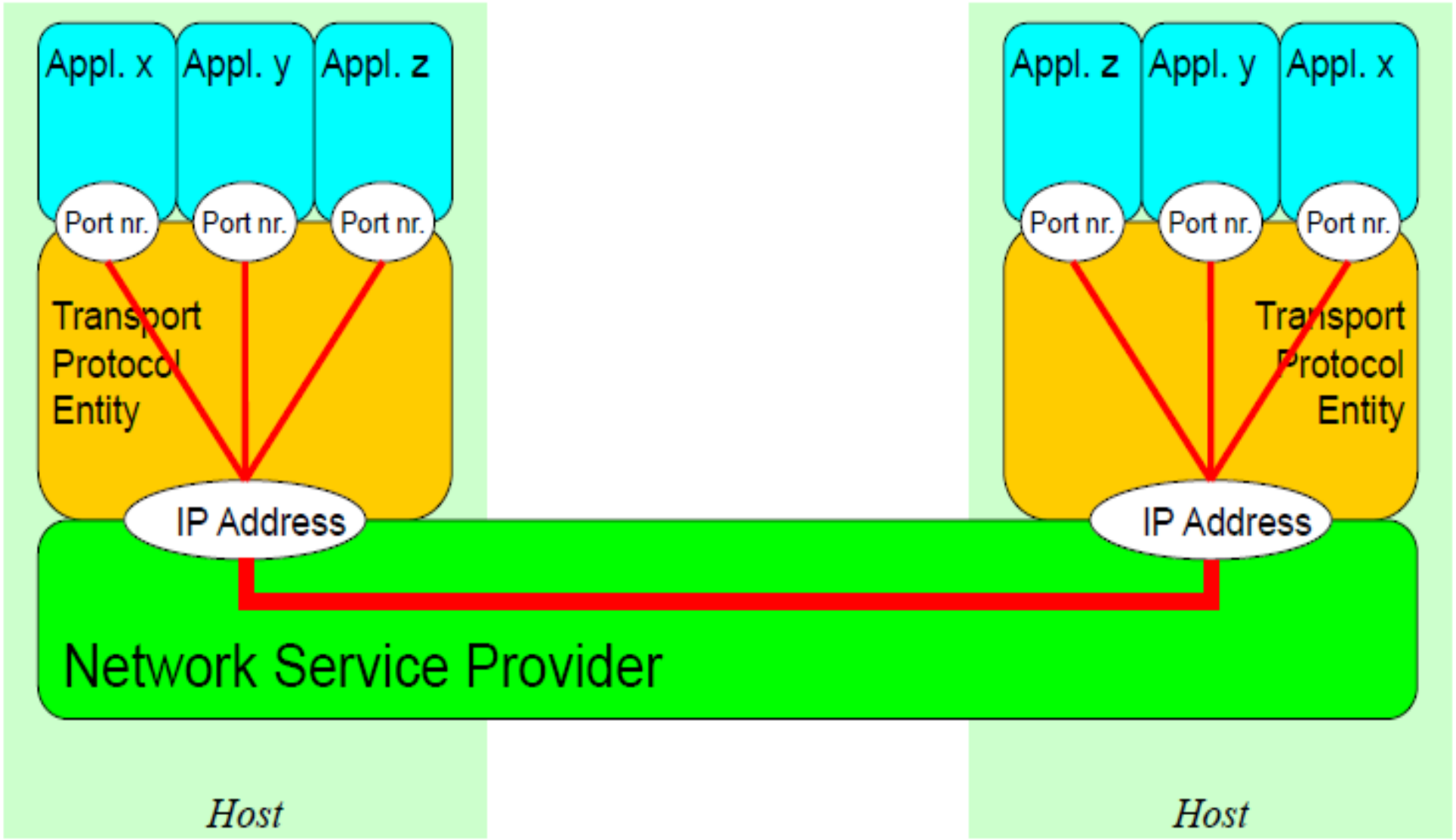
Token Bucket Vs. Leaky Bucket

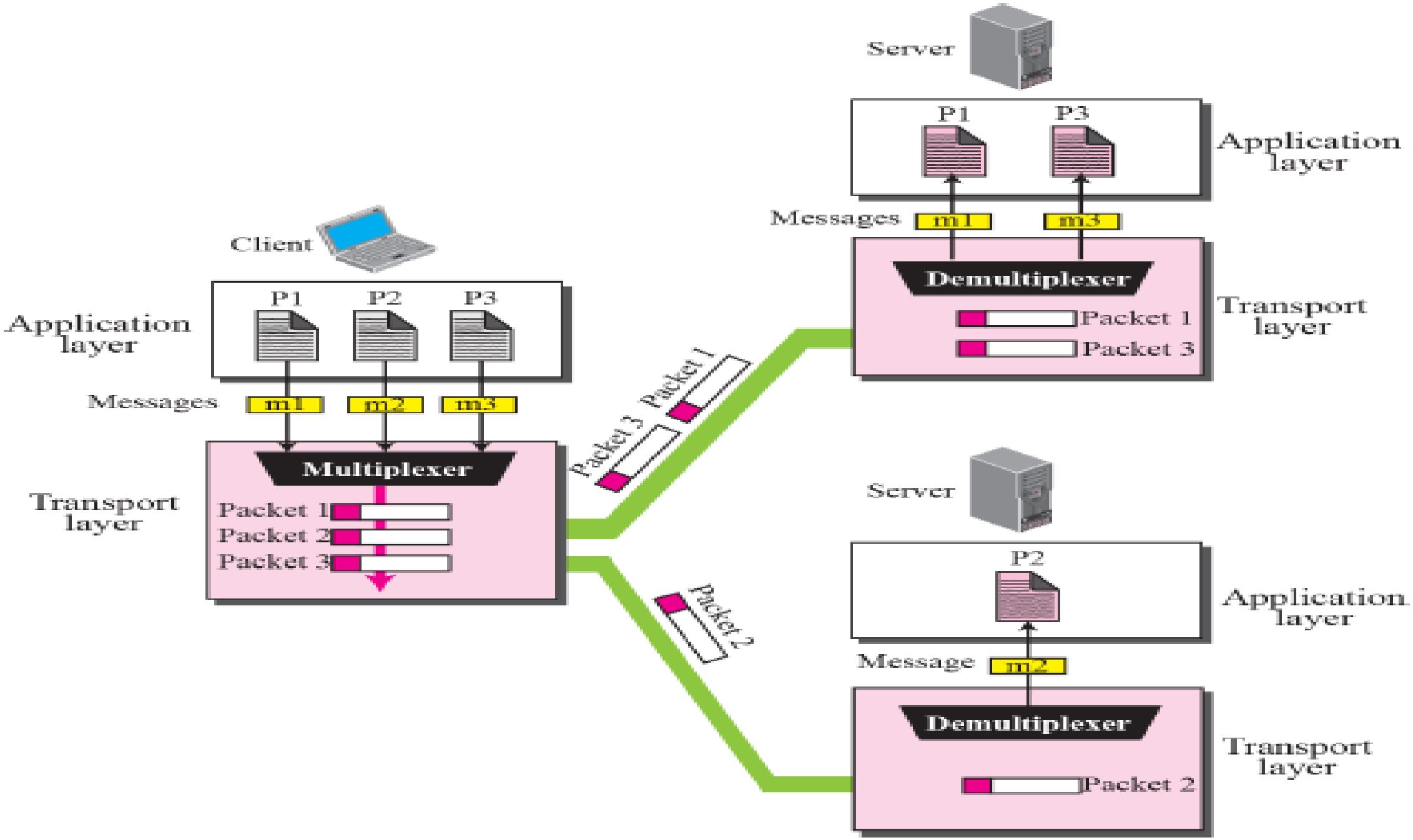
- The token bucket algorithm provides a different kind of traffic shaping than that of the leaky bucket algorithm. The **leaky bucket algorithm does not allow idle hosts to save up permission** to send large bursts later.
- Another difference between the two algorithms is that the token bucket algorithm throws away tokens (i.e., transmission capacity) when the bucket fills up but never discards packets. In contrast, the **leaky bucket algorithm discards packets when the bucket fills up**.

Multiplexing & de-multiplexing

- **Multiplexing at sending host:** gathering data from multiple sockets, enveloping data with header (later used for de-multiplexing).
- **De-multiplexing at receiving host:** delivering received segments to correct socket.
- How **de-multiplexing works** ?
  - Host receives IP datagram
    - ✓each datagram has source IP address, destination IP address
    - ✓each datagram carries 1 transport layer segment
    - ✓each segment has source, destination port number
  - Host uses IP addresses & port numbers to direct segment to appropriate socket.

Multiplexing & de-multiplexing




Server
P1
P3
Application layer
Messages
m1
m3
Demultiplexer
Packet 1
Packet 3
Transport layer
Client
P1
P2
P3
Application layer
Messages
m1
m2
m3
Multiplexer
Transport layer
Packet 1
Packet 2
Packet 3
Packet 3
Packet 1
Packet 2
Server
P2
Application layer
Message
m2
Demultiplexer
Packet 2
Transport layer

Thank You
???

References:

- Data Communications and Networking " Behrouz A. Forouzan"
- Computer Networks "A. S. Tanenbaum" Fifth Edition
- Data and Computer Communications "William Stallings" Tenth Edition.