IIT KHARAGPUR

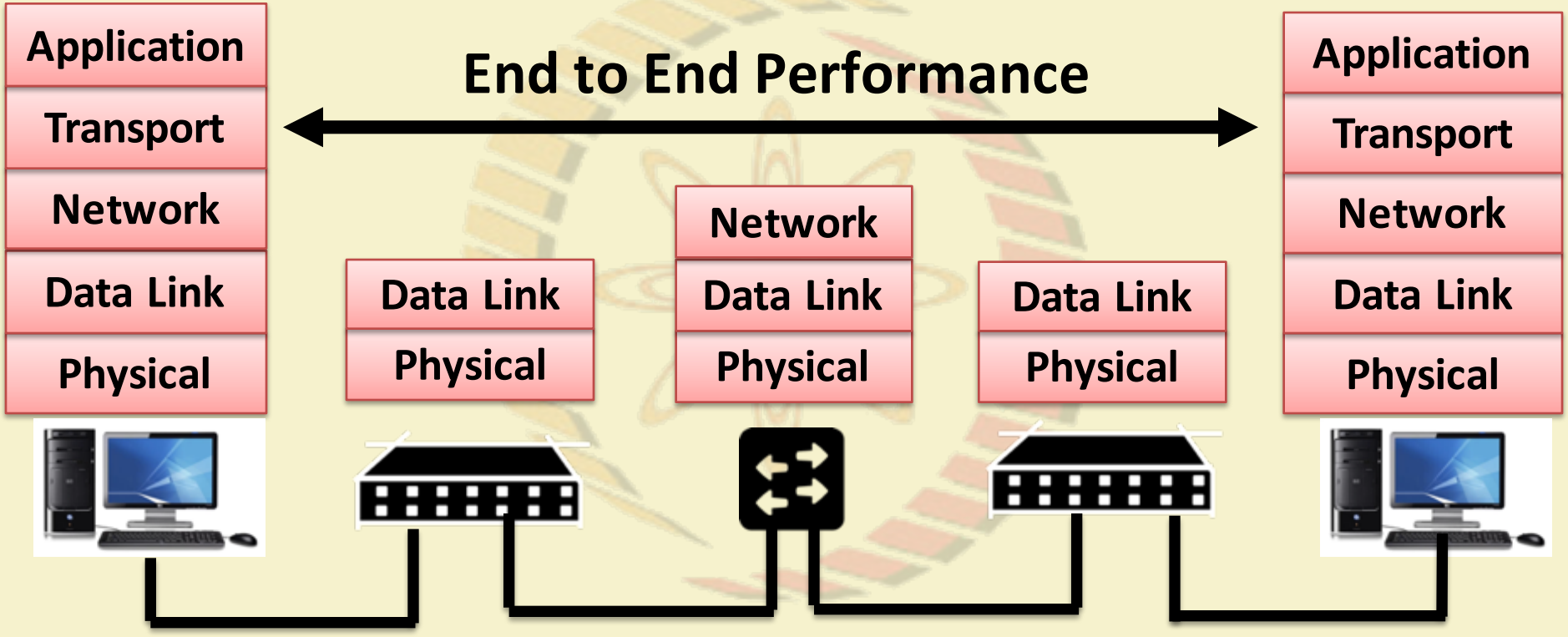
NPTEL ONLINE
CERTIFICATION COURSES

COMPUTER NETWORKS AND INTERNET PROTOCOLS

SOUMYA K GHOSH
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

SANDIP CHAKRABORTY
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

Transport Layer - VI (Performance)



Bandwidth Delay Product

- **Bandwidth Delay Product (BDP) = Link Bandwidth x Link Delay** – an important metric for flow control

- Consider Bandwidth = 50 Kbps, one way transit time (delay) = 250 msec
  - BDP 12.5 Kbit
  - Assume 1000 bit segment size; BDP = 12.5 segments

- Consider the event of a segment transmission and the corresponding ACK reception – this takes a round trip time (RTT) – twice the one way latency.

- Maximum number of segments that can be outstanding during this duration = 12.5 x 2 = 25 segments

Bandwidth Delay Product – Implication on Window Size

- Maximum number of segments that can be outstanding within this duration = 25 + 1 (as the ACK is sent only when the first segment is received) = 26
  - This gives the maximum link utilization – **the link will always be busy in transmitting data segments**

- Let **BD** denotes the number of frames equivalent to the BDP, **w** is the maximum window size

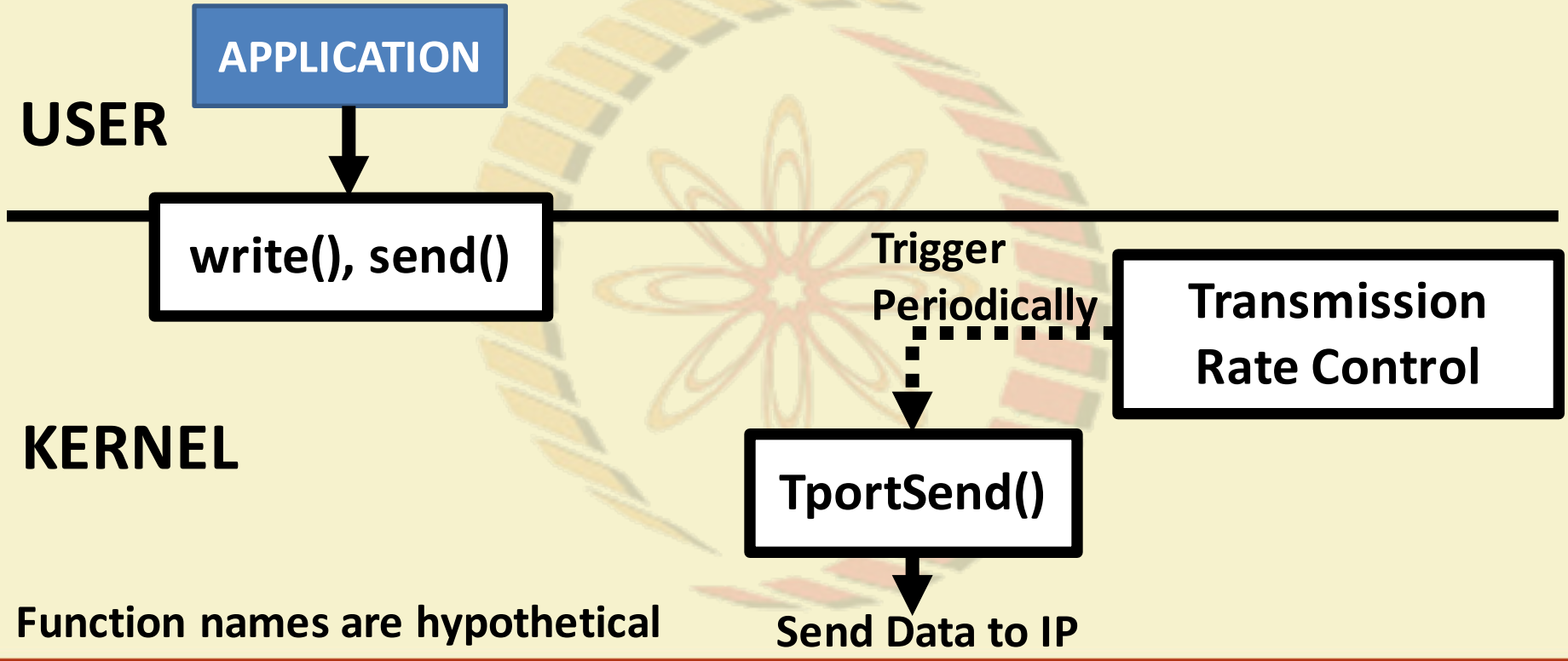
- So, **w = 2BD + 1** gives the maximum link utilization – **this is an important concept to decide the window size for a window based flow control mechanism**

Implication of BDP on Protocol Design Choice

- Consider the link bandwidth = 1Mbps, Delay = 1ms
- Consider a network, where segment size is 1 KB (1024 bytes)
- Which protocol is better for flow control?
  - (a) stop and wait,
  - (b) Go back N,
  - (c) Selective Repeat

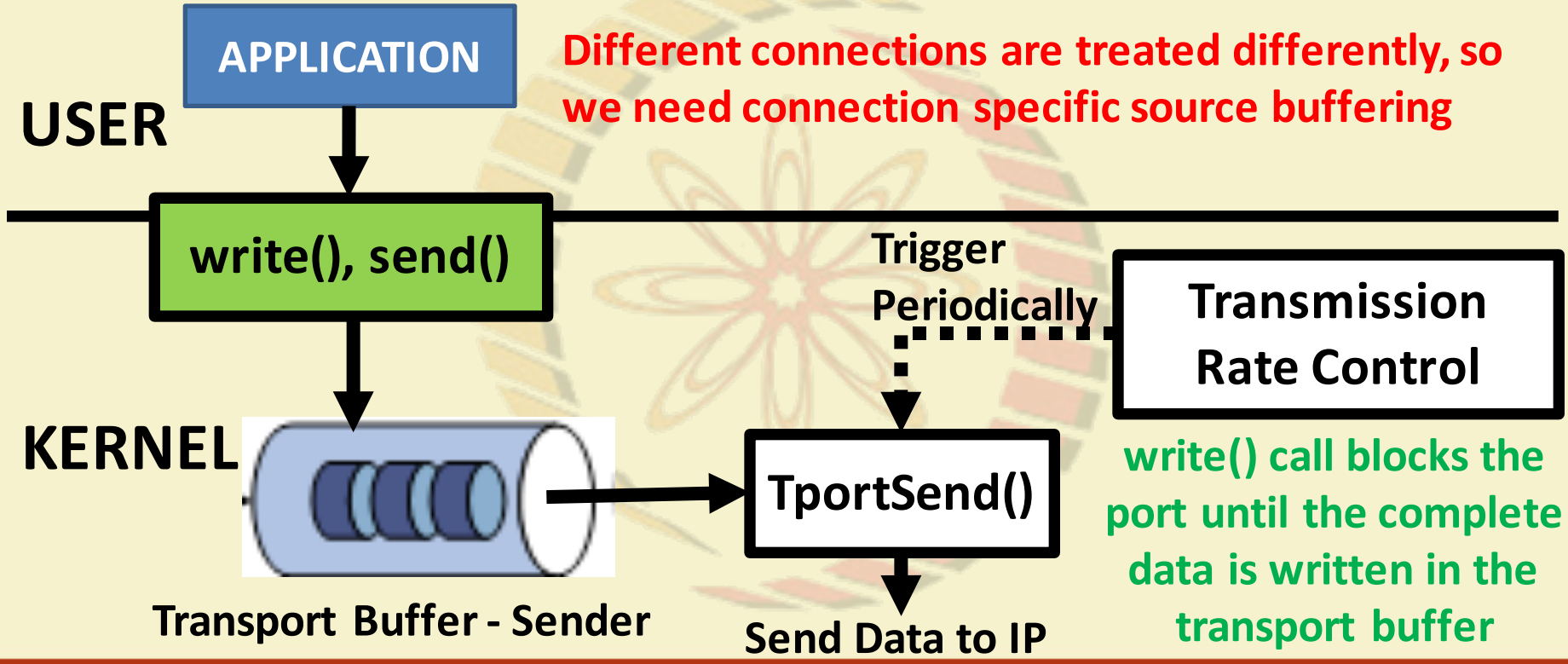
- **BDP = 1 Mbps x 1ms = 1 Kb (1024 bits)**
- **The segment size is eight times larger than the BDP -> the link can not hold an entire segment completely**
- **Sliding window protocols do not improve performance**
- **Stop and Wait is better – less complexity**

Application Transport Interfacing – Sender Side

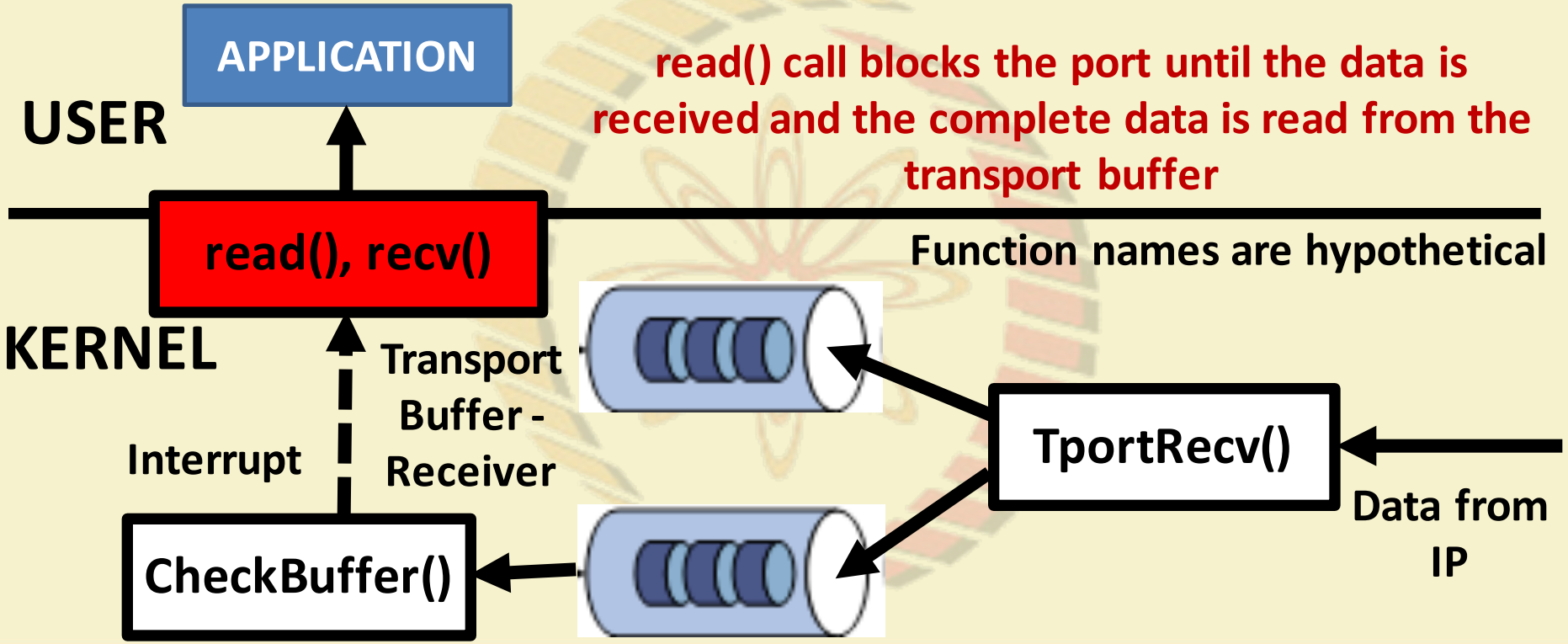


Function names are hypothetical

Application Transport Interfacing – Sender Side

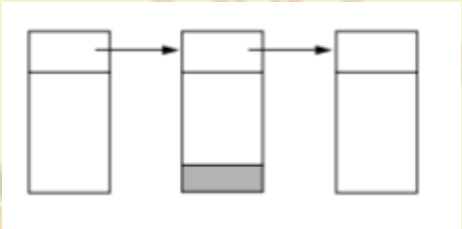


Application Transport Interfacing – Receiver Side



Organizing Transport Buffer Pool

- If most segments are nearly the same size, organize the buffer as a pool of identically sized buffers (one segment per buffer)
- For variable segment size – **chained fixed sized buffer** (buffer size = maximum segment size)

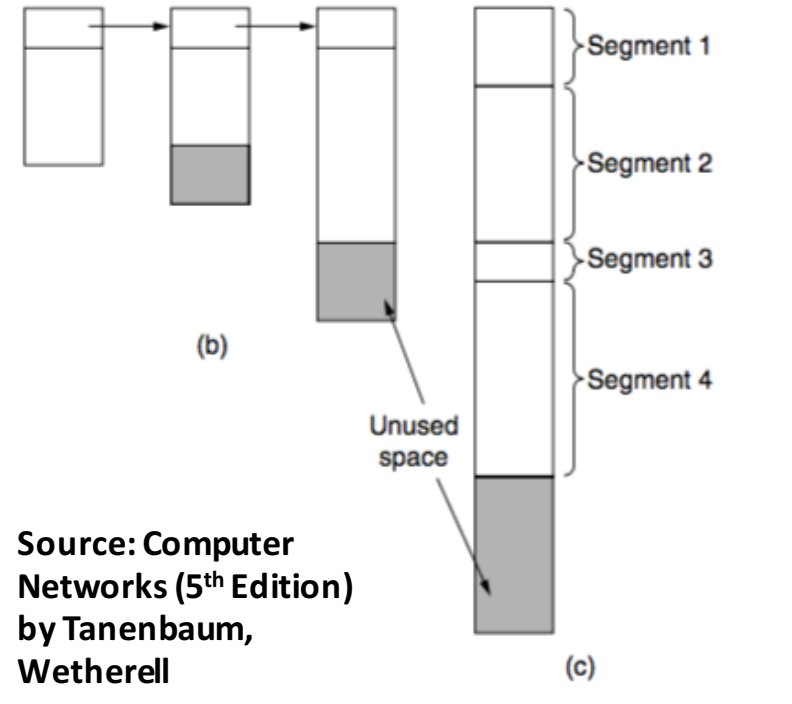


- Space would be wasted if segment sizes are widely varied
- Small buffer size – multiple buffers to store a single segment – added complexity in implementation

Organizing Transport Buffer Pool

- **Variable size buffers (b)**
  - Advantage: better memory utilization
  - Disadvantage: Complicated implementation

- Single large **circular buffer** for every connection (c)
  - Good use of memory only when connections are heavily loaded



Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell


thank you!

IIT KHARAGPUR

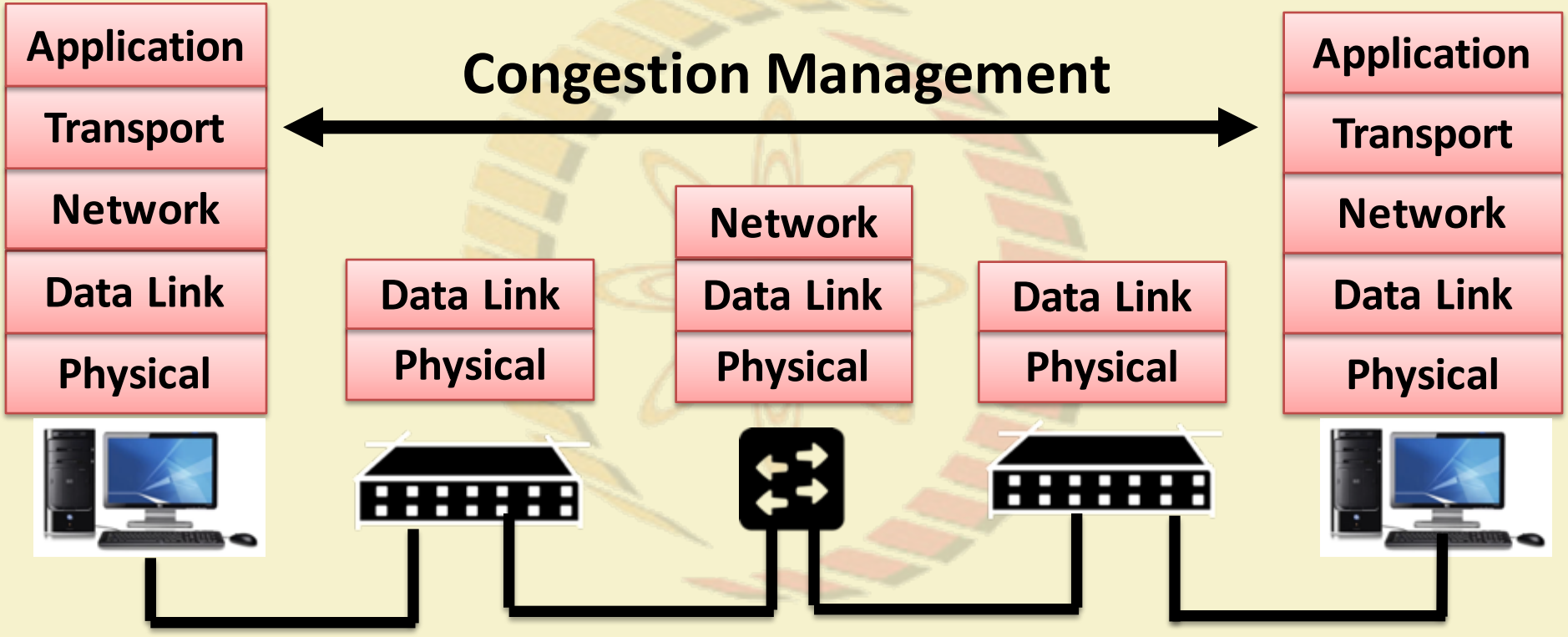
NPTEL ONLINE
CERTIFICATION COURSES

COMPUTER NETWORKS AND INTERNET PROTOCOLS

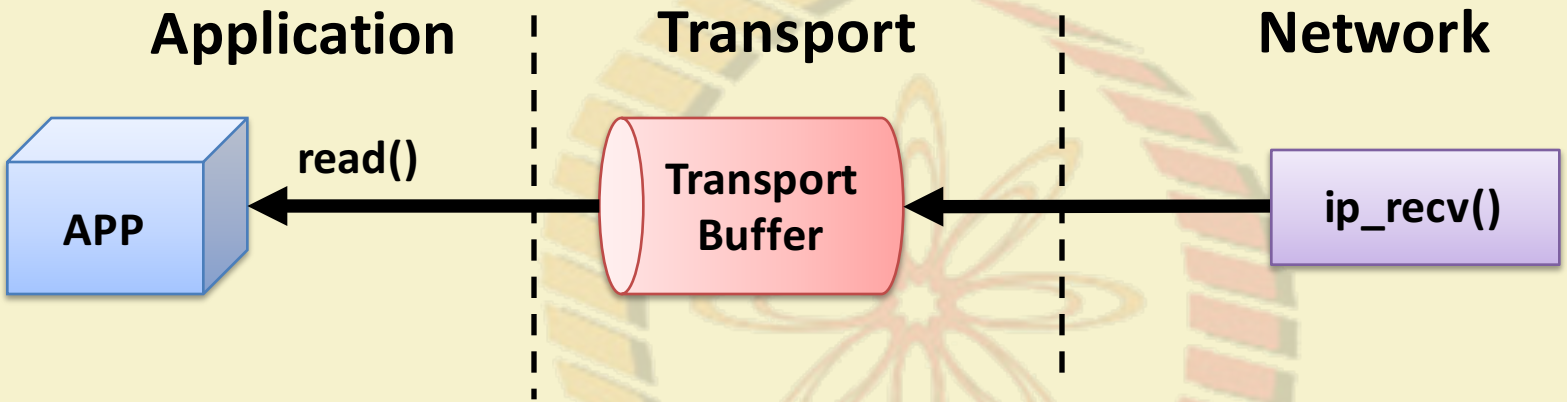
SOUMYA K GHOSH
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

SANDIP CHAKRABORTY
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

Transport Layer - VII (Buffer Management and Congestion Control)



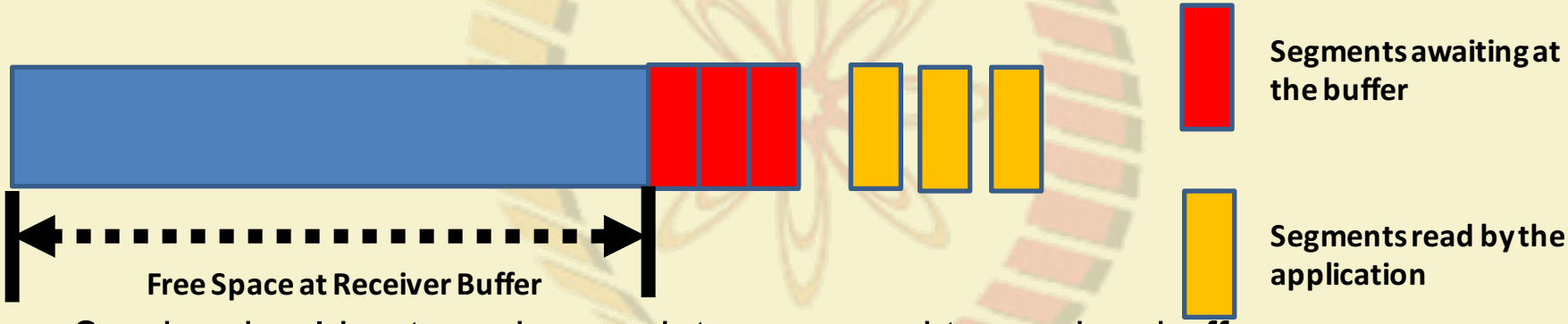
Transport Buffer at the Receiver Side



- There can be rate difference between
  - The rate of application read
  - The rate of transport receive

Dynamic Buffer Management for Window Based Flow Control

- Sender and receiver needs to dynamically adjust buffer allocations
- Based on the rate difference between the **transport entity** and the **application**, the available size of the receiver buffer changes



- Sender should not send more data compared to receiver buffer space – dynamically adjust the window size based on availability of receiver buffer space

Dynamic Buffer Management for Window Based Flow Control

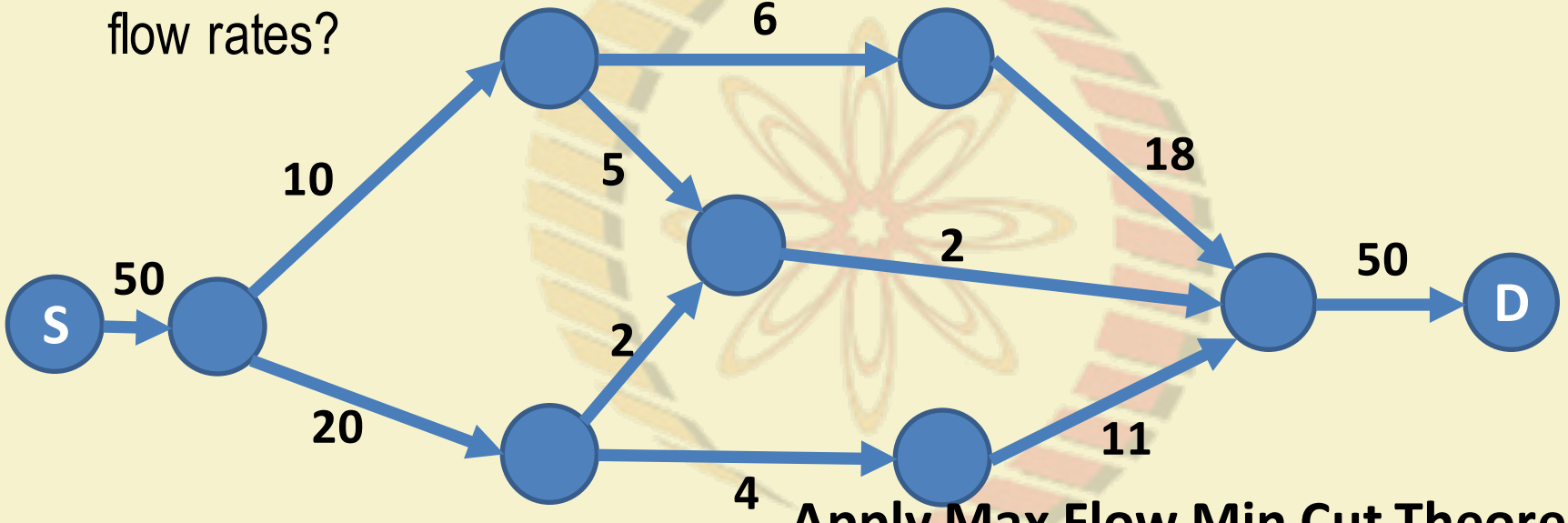
- Receiver forwards available buffer space through ACK

	A	Message	B	Comments
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	•••	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	•••	<ack = 6, buf = 4>	←	Potential deadlock

Ensure that the ACKs are flowing in the network continuously

Congestion Control in the Network

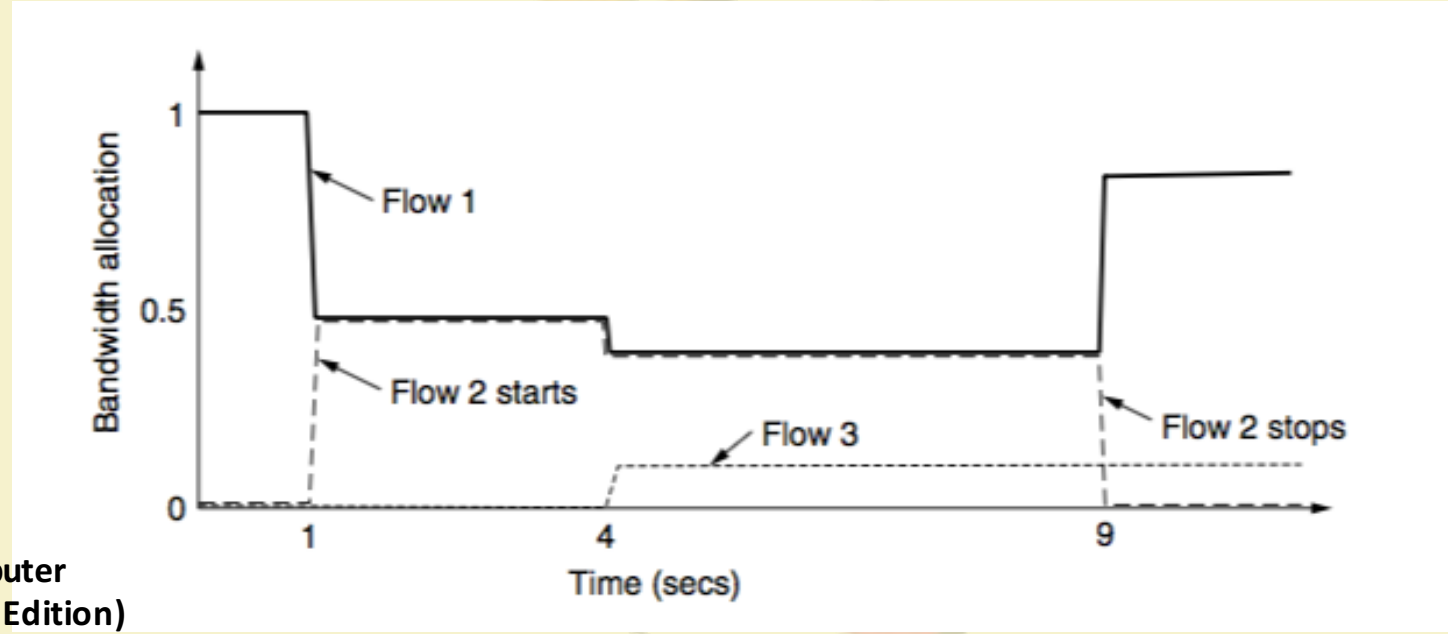
- Consider a centralized network scenario – how can you maintain optimal flow rates?



Apply Max Flow Min Cut Theorem !

But this is hard in a real network …

Congestion Control in the Network



Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Changing Bandwidth Allocation over Time

Congestion Control in the Network

- Flows enter and exit network dynamically – so applying an algorithm for congestion control is difficult

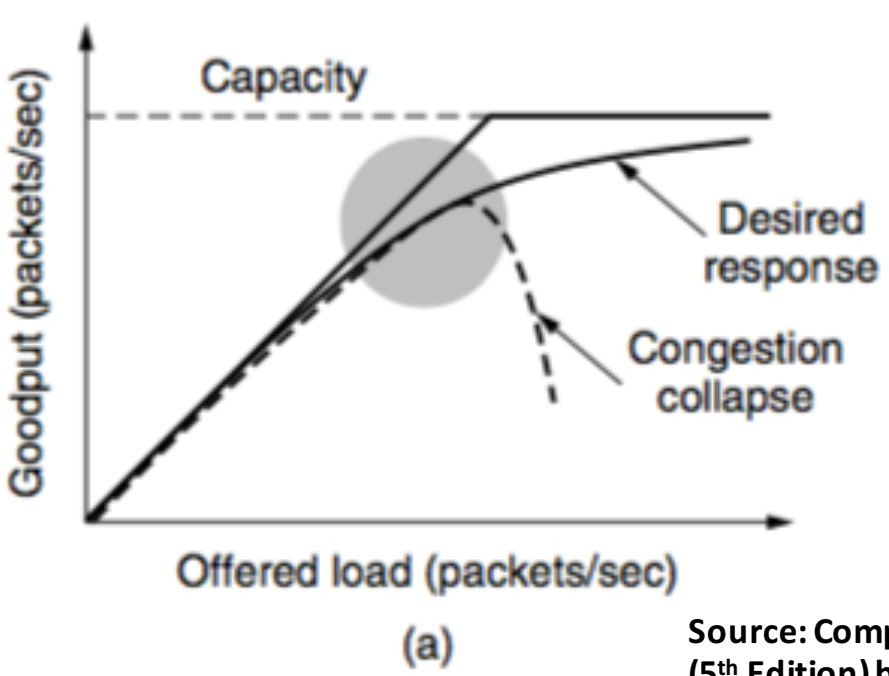
- **Congestion avoidance:** Regulate the sending rate based on what the network can support

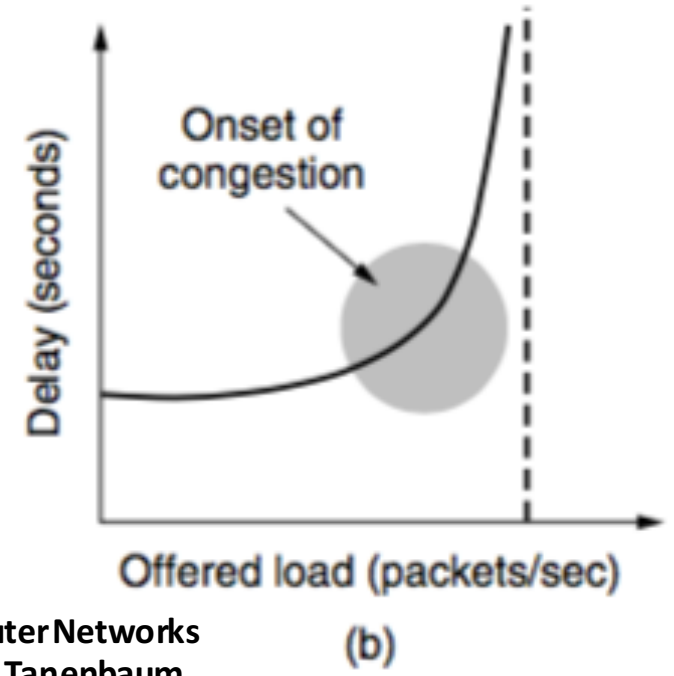
Sending Rate = minimum (network rate, Receiver rate)

Gradually increase the network rate and observe the effect on flow rates (packet loss)

Comes from flow control – receiver advertised window size for a sliding window flow control

Network Congestion – Impact over Goodput and Delay



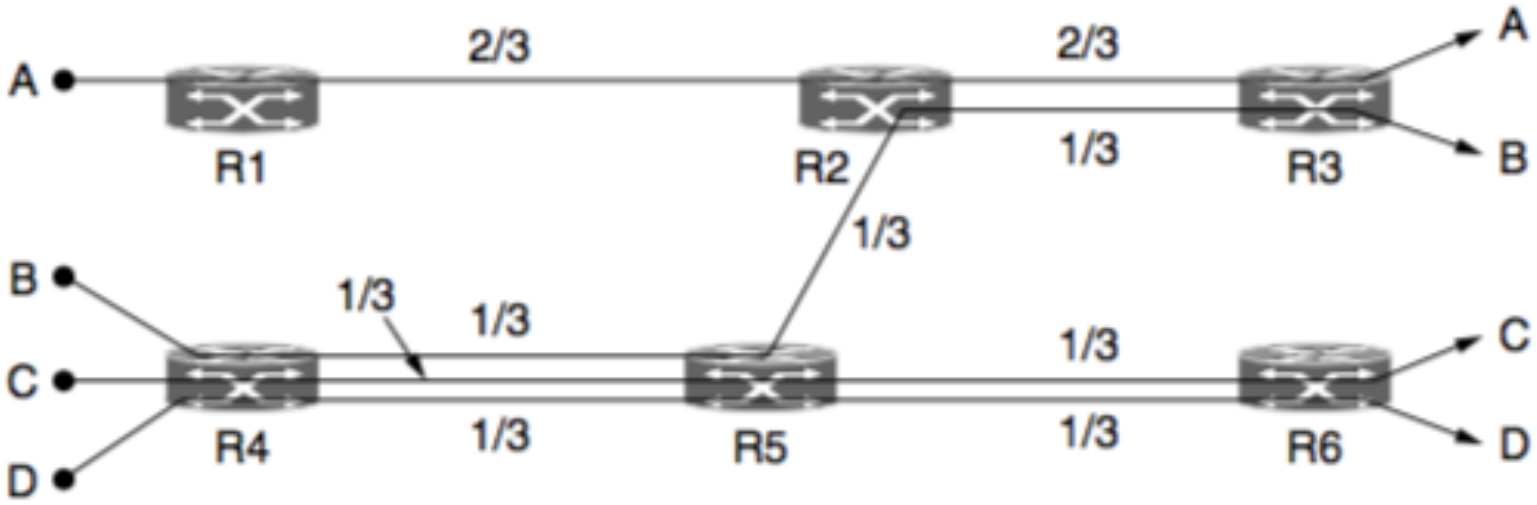


Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Congestion Control and Fairness

- Ensure that the rate of all the flows in the network is controlled in a **fair way**
- A bad congestion control algorithm may affect fairness - Some flows can get starved
- Hard fairness in a decentralized network is difficult to implement
- **Max-Min Fairness**: An allocation is max-min fair if the bandwidth given to one flow cannot be increased without decreasing the bandwidth given to another flow with an allocation.

Max-Min Fairness – An Example



Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

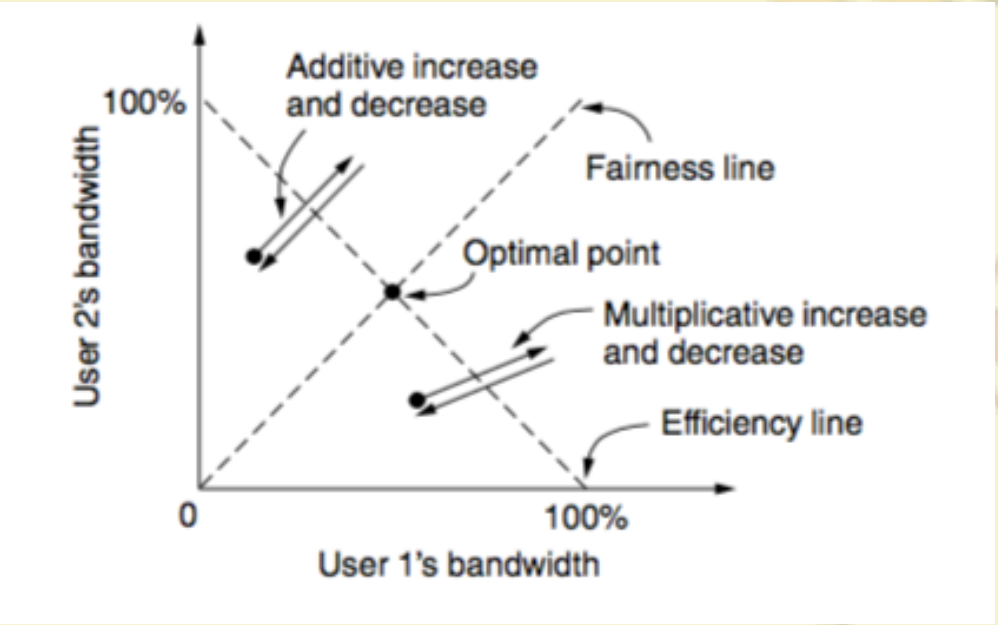
AIMD – Efficient and Fair Operating Point for Congestion Control

- **Additive Increase Multiplicative Decrease (AIMD)** – Chiu and Jain (1989)

- Let *w(t)* be the sending rate. *a (a > 0)* is the additive increase factor, and *b (0<b<1)* is the multiplicative decrease factor

$$w(t+1) = \begin{cases} w(t) + a & \text{if congestion is not detected} \\ w(t) \times b & \text{if congestion is detected} \end{cases}$$

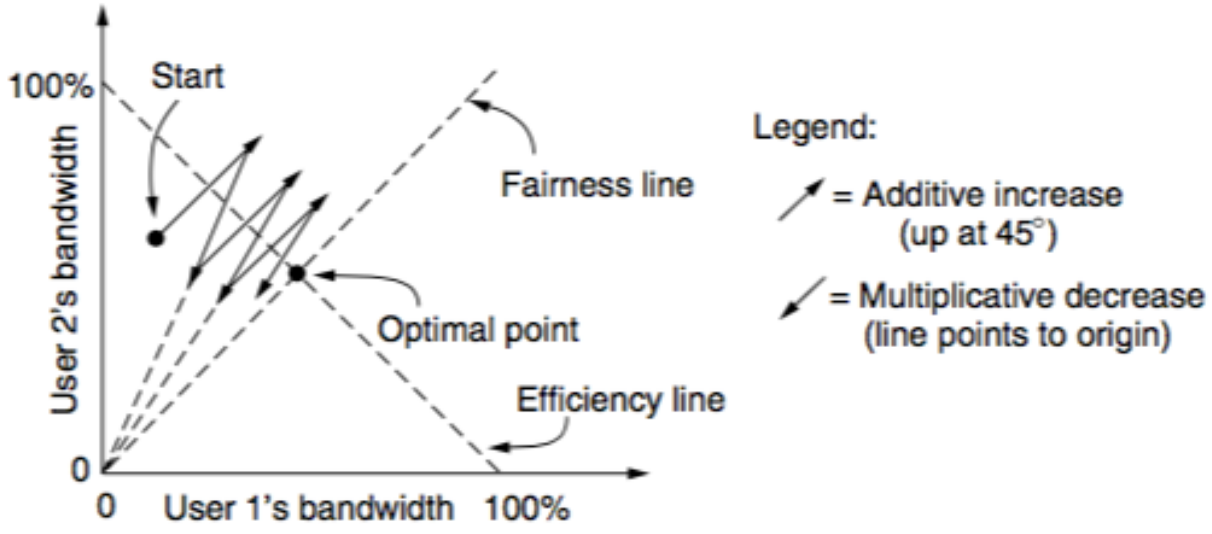
AIMD – Design Rationale (Two Flows Example)



- **AIAD** – Oscillate across the efficiency line
- **MIMD** – Oscillate across the efficiency line (different slope from AIAD)

Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

AIMD – Design Rationale (Two Flows Example)



Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

- The path converges towards the optimal point
- Used by TCP - Adjust the size of the sliding window to control the rates


thank you!

IIT KHARAGPUR

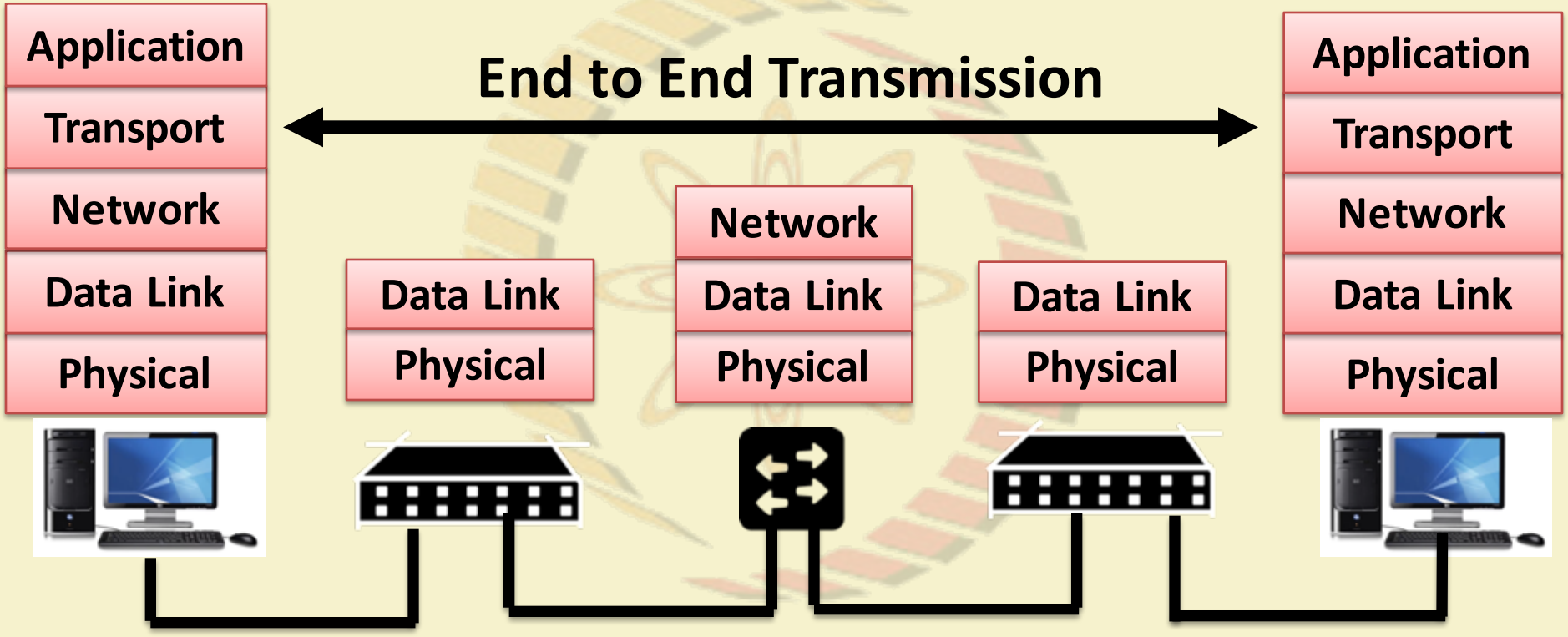
NPTEL ONLINE
CERTIFICATION COURSES

COMPUTER NETWORKS AND INTERNET PROTOCOLS

SOUMYA K GHOSH
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

SANDIP CHAKRABORTY
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

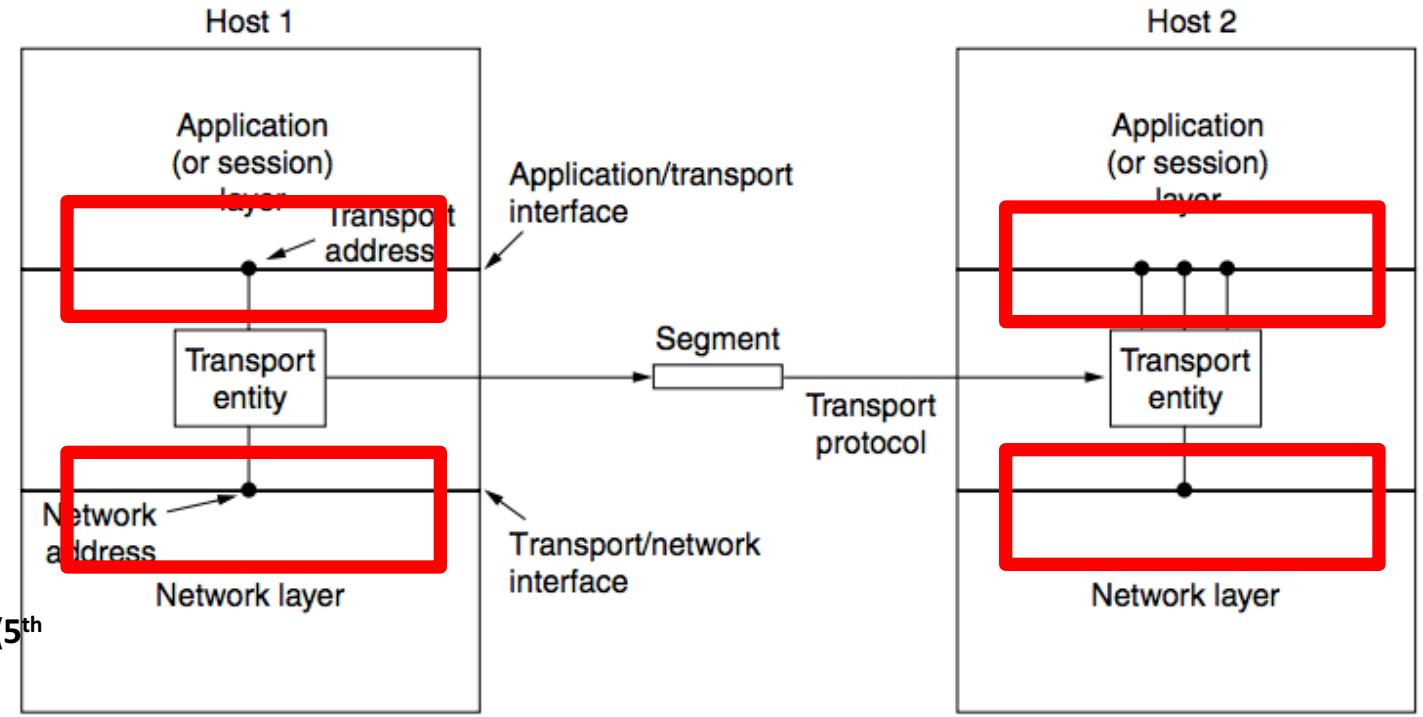
Transport Layer - VI (Primitives)



Transport Layer – Interfacing with Application and Network

Port Number

IP Address

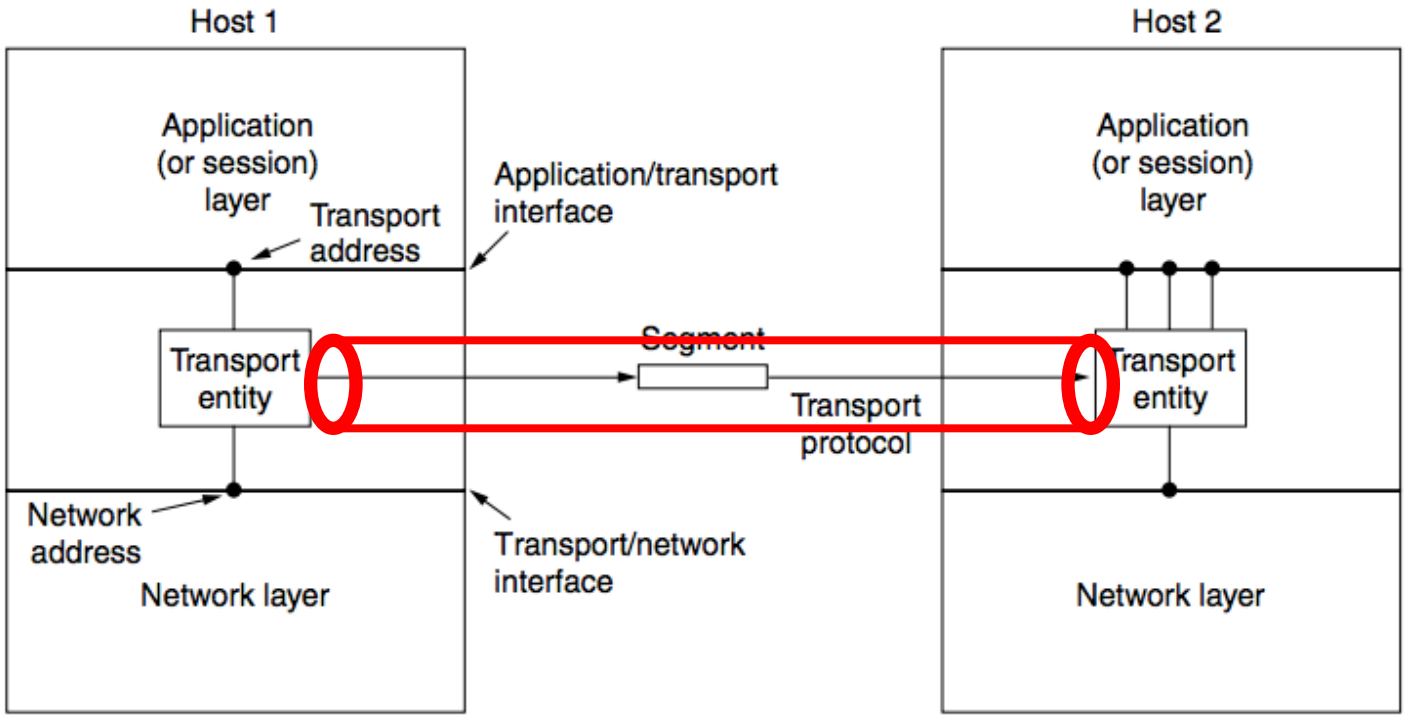


Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Transport Layer – Interfacing with Application and Network

Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Create a logical pipe between the sender and the receiver and monitor the data transmission through this pipe



Transport Service Primitives

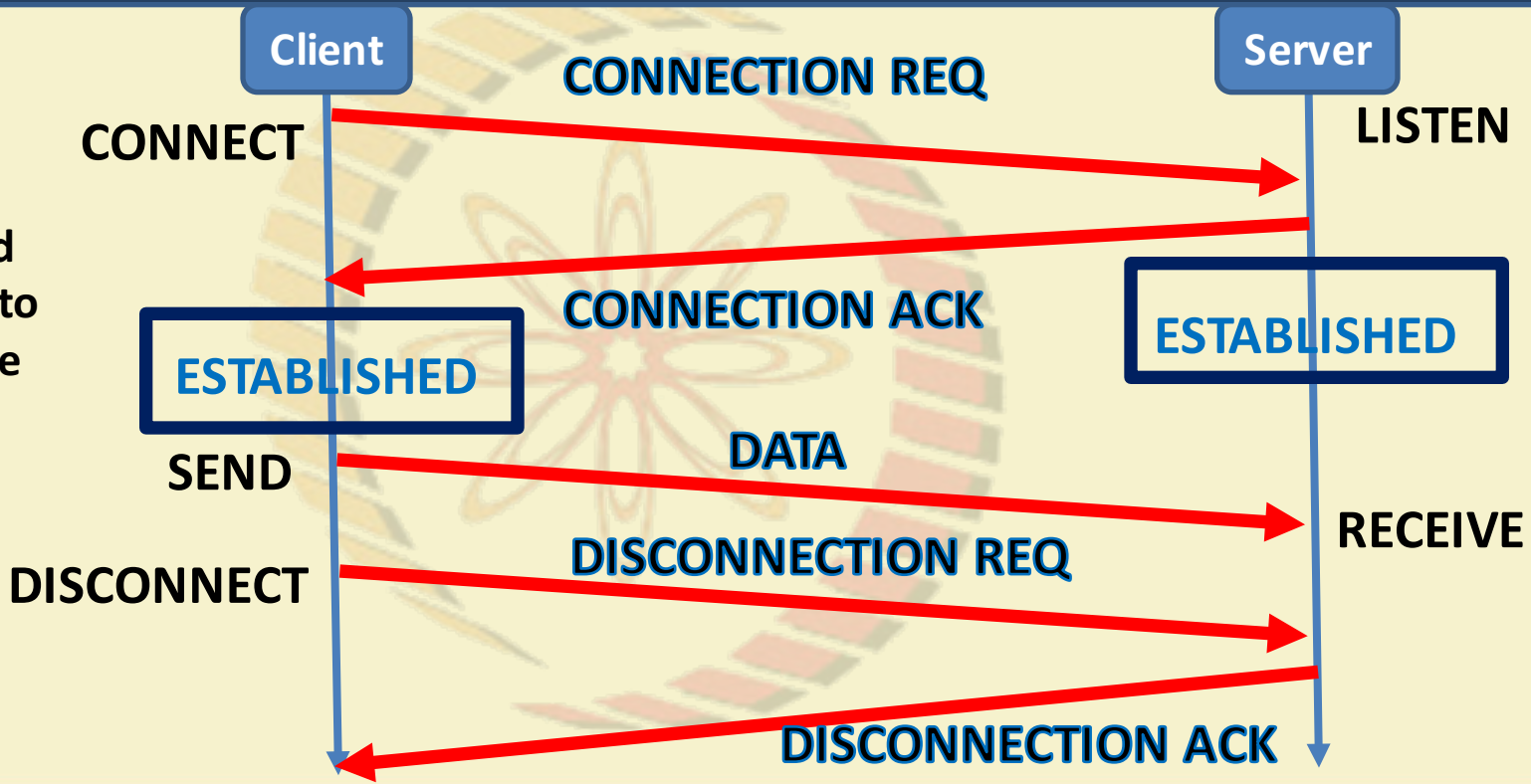
- To allow users to access transport service, the transport layer must provide some operations to the application programs.

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	Request a release of the connection

The transport layer needs to remember the state of the pipe, so that appropriate actions can be taken. We need a stateful protocol for transport layer.

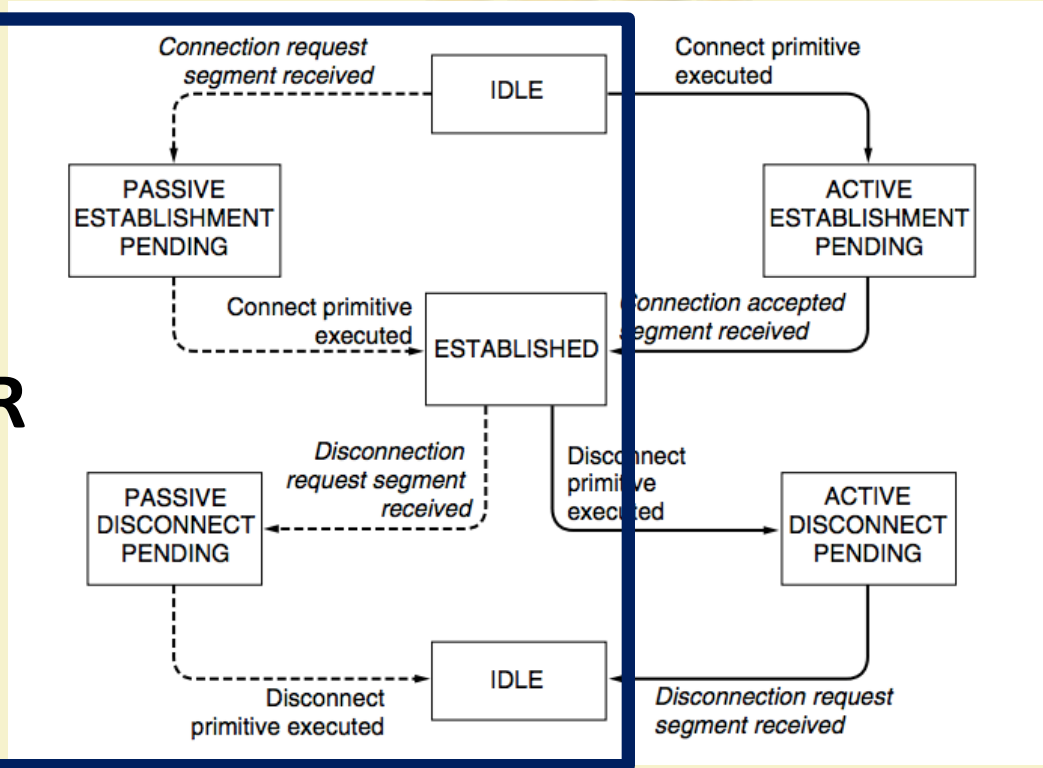
Transport Service Primitive – Connection Establishment

The client and server needs to remember the state



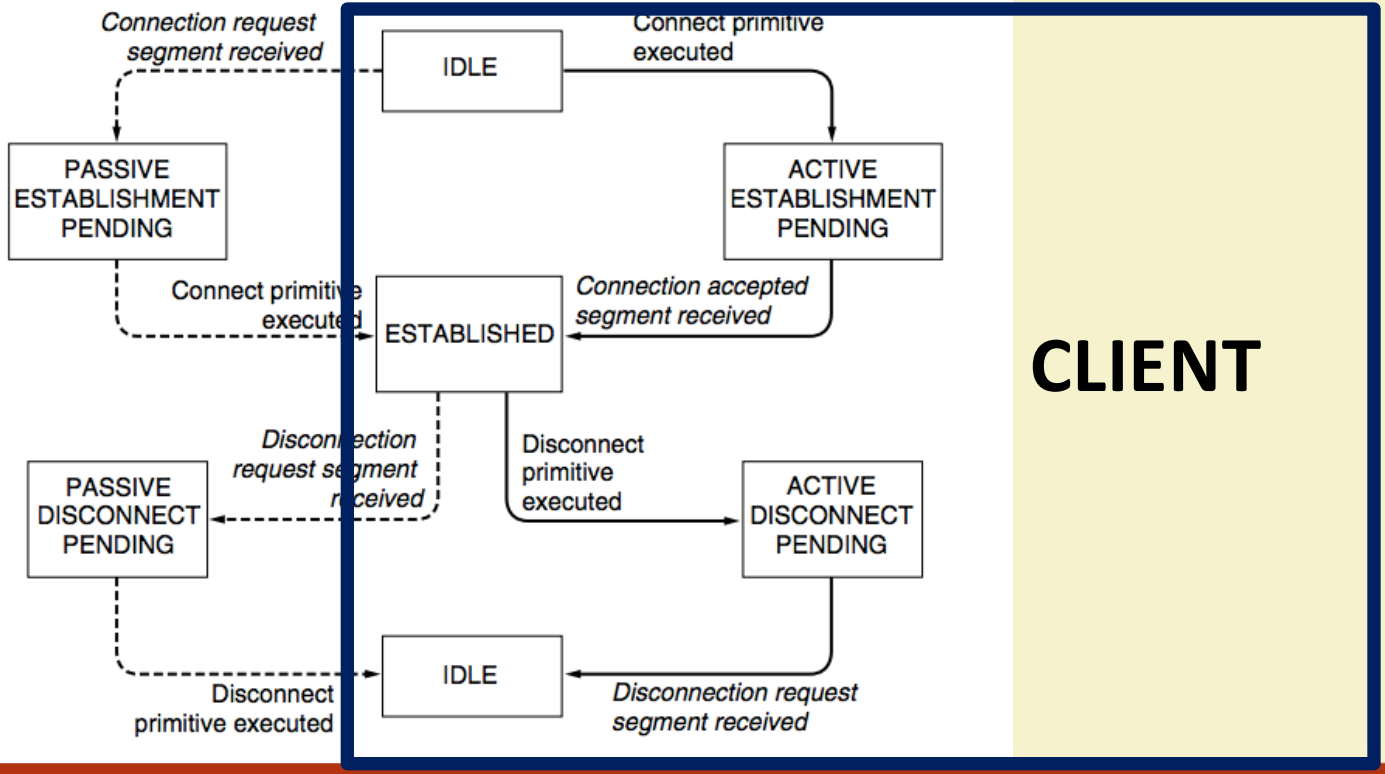
Transport Layer Protocol – State Diagram

SERVER



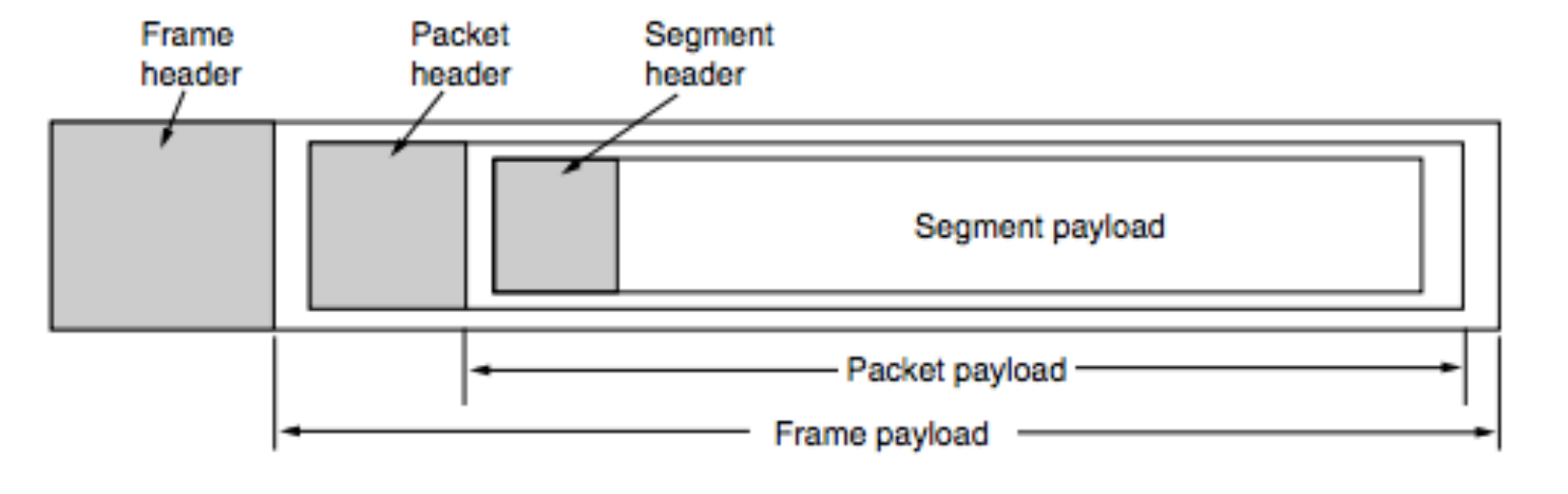
Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Transport Layer Protocol – State Diagram



Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Segment, Packet (or Datagram) and Frame



Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

Transport Layer Process Flow

- **Connection Establishment** - Initiate a connection by selecting the initial sequence numbers, ensuring that the initial sequence numbers do not fall within the forbidden region of the previous connection between the same <source IP, source port, destination IP, destination port>
  - Sequence number becomes a part of a transport layer connection
  - <source IP, source port, source initial sequence number, destination IP, destination port, destination initial sequence number> - uniquely identifies a connection

Transport Layer Process Flow

- **Flow Control and Reliability** - Use ARQ protocols for ensuring flow control and reliability
  - Sender will not send data at a rate higher than the receiver rate
  - Sequence numbers are used to uniquely identify each byte/each packet
  - Loss in the communication path is handled through retransmission
- **Congestion Control** - reduce transmission rate once congestion is detected

Transport Layer Process Flow

- **Congestion Control -** reduce transmission rate once congestion is detected
  - Improves performance for end-to-end data delivery

- **Connection Closure** - close the connection when data transmission is complete
  - Synchronous closure with timeout


thank you!

IIT KHARAGPUR

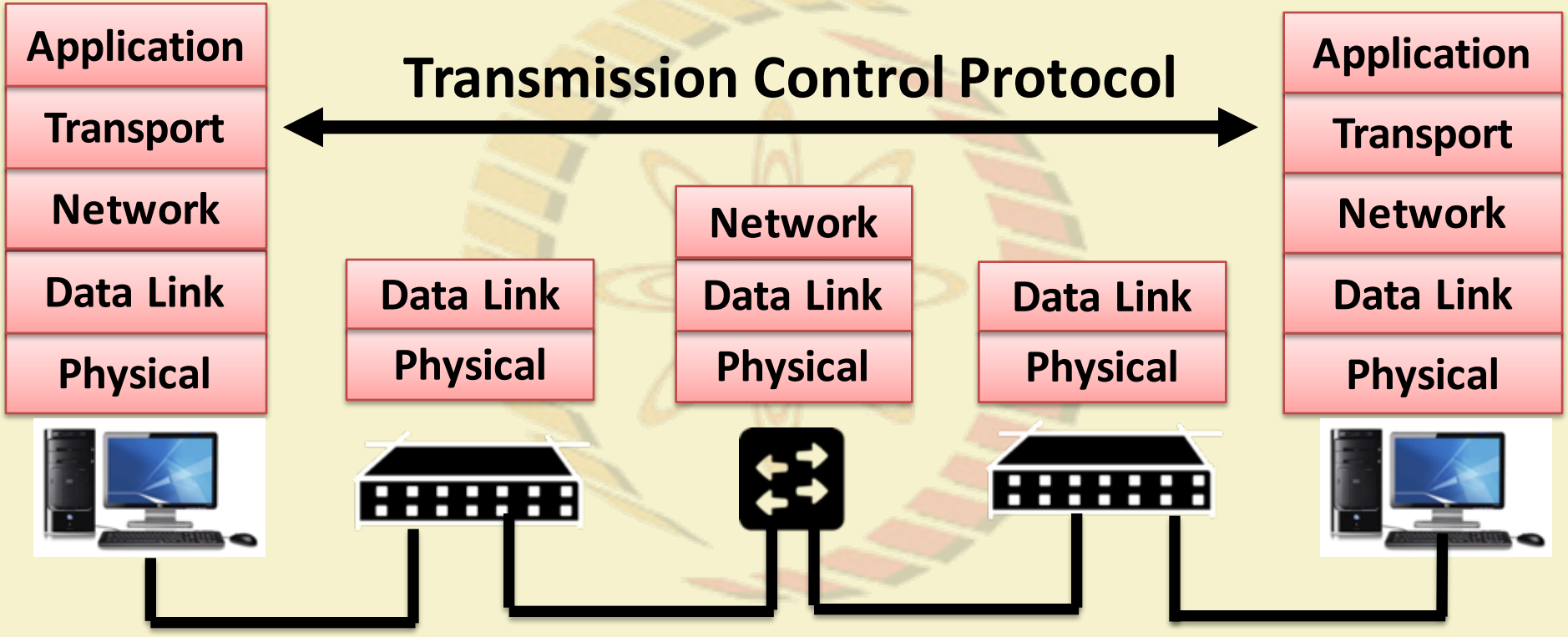
NPTEL ONLINE
CERTIFICATION COURSES

COMPUTER NETWORKS AND INTERNET PROTOCOLS

SOUMYA K GHOSH
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

SANDIP CHAKRABORTY
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

Transmission Control Protocol I (Primitives)



Transmission Control Protocol (TCP)

- TCP was specifically designed to provide a reliable, end-to-end byte stream over an unreliable **internetwork**.

- **Internetwork** – different parts may have widely different topologies, bandwidths, delays, packet sizes and other parameters

Transmission Control Protocol (TCP)

- **TCP** dynamically adapts to properties of the internetwork and is robust in the face of many kinds of failures.

- RFC 793 (September 1981) – Base protocol
  - RFC 1122 (clarifications and bug fixes), RFC 1323 (High performance), RFC 2018 (SACK), RFC 2581 (Congestion Control), RFC 3168 (Explicit Congestion Notification)

TCP Service Model

- All TCP connections are full-diplex and point-to-point. TCP does not support multicasting or broadcasting.

- Uses **Sockets** to define an end-to-end connection (Source IP, Source Port, Source Initial Sequence Number, Destination IP, Destination Port, Destination Initial Sequence Number)

TCP Service Model

- **Unix Model of Socket Implementation:**
  - A single daemon process, called **Internet Daemon (inetd)** runs all the times at different **well known ports**, and wait for the first incoming connection
  - When a first incoming connection comes, *inetd* forks a new process and starts the corresponding daemon (for example *httpd* at port 80, *ftpd* at port 21 etc.)

TCP Service Model

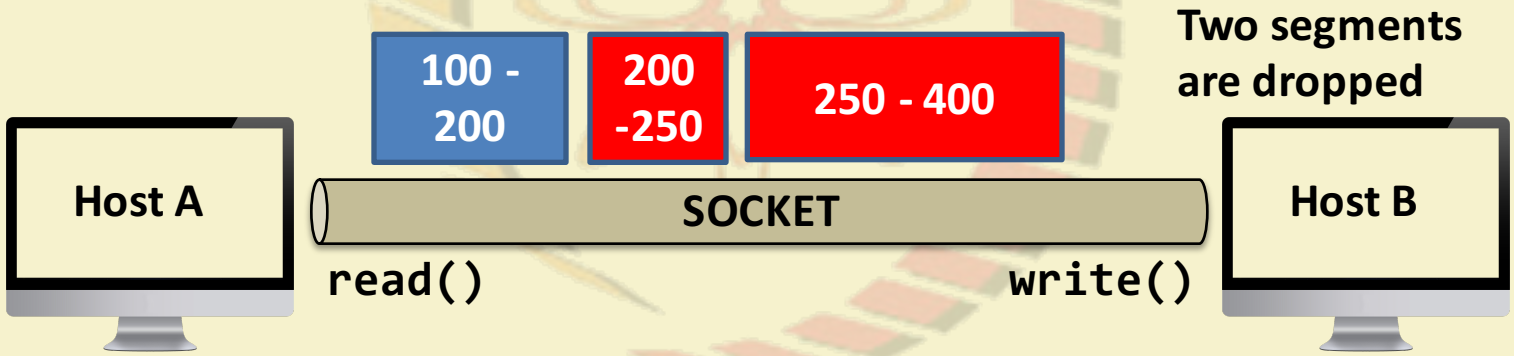
- A TCP connection is a **byte stream**, not a message stream

- Message boundaries are not preserved end-to-end

TCP Service Model

- A TCP connection is a **byte stream**, not a message stream

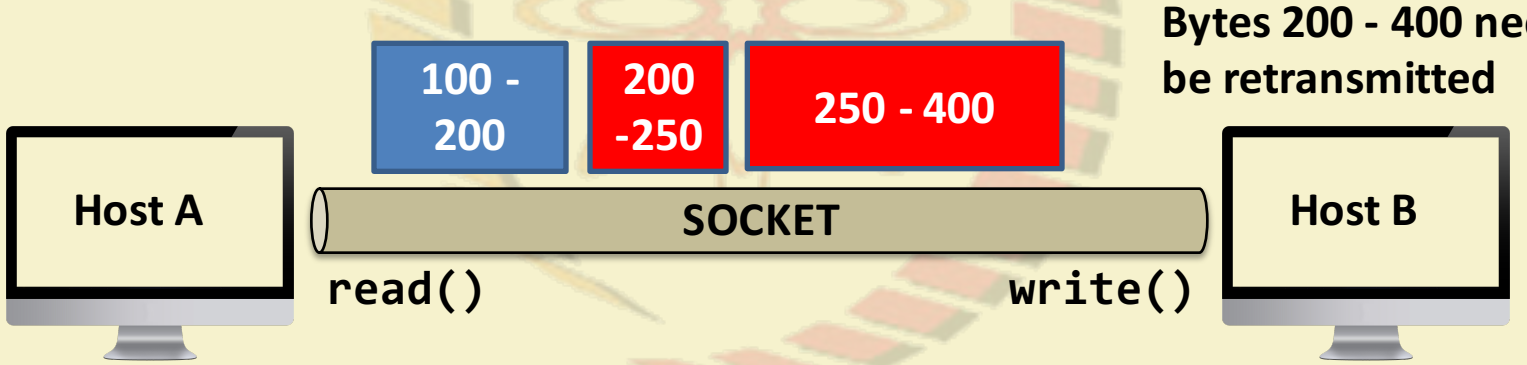
- Message boundaries are not preserved end-to-end



TCP Service Model

- A TCP connection is a **byte stream**, not a message stream

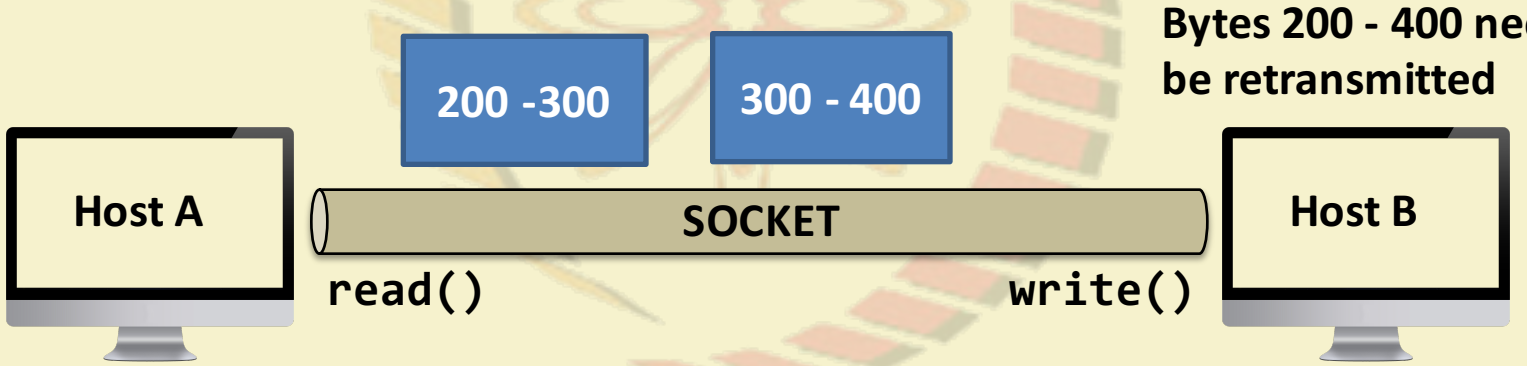
- Message boundaries are not preserved end-to-end



TCP Service Model

- A TCP connection is a **byte stream**, not a message stream

- Message boundaries are not preserved end-to-end

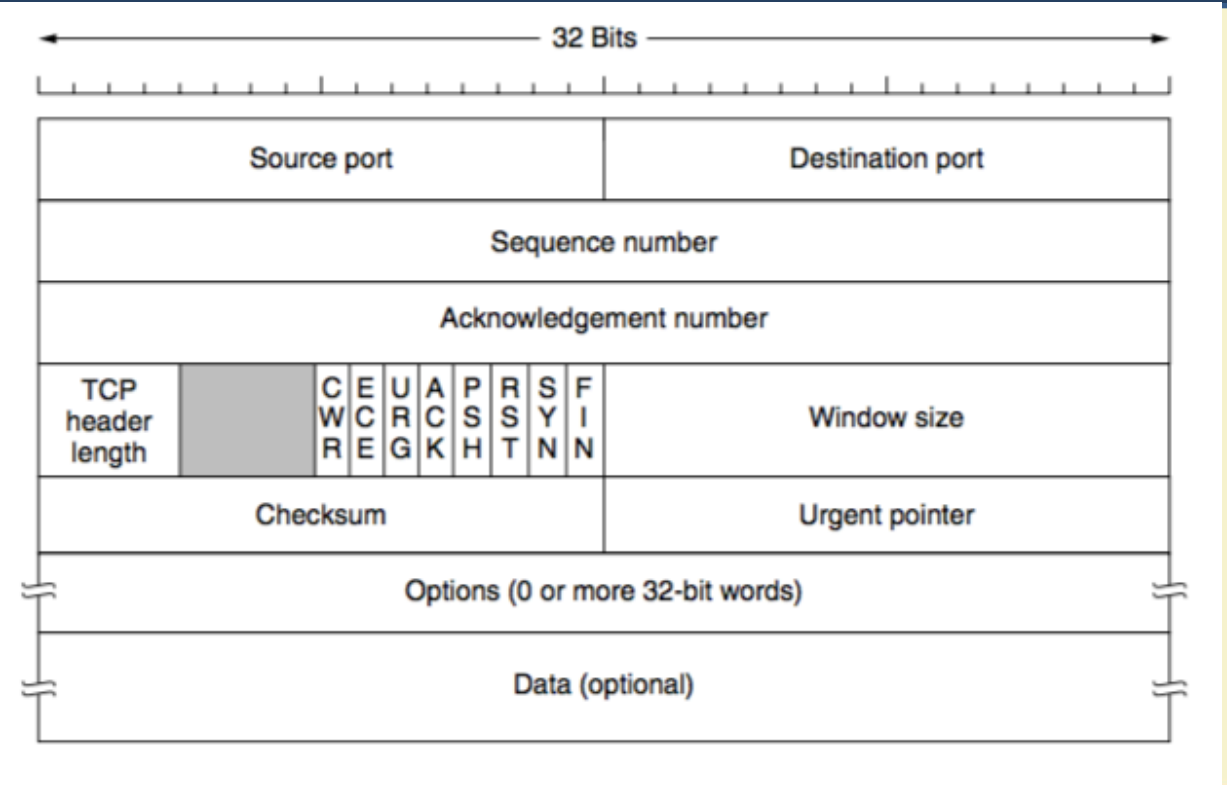


TCP Service Model

Example:

- The sending process does four 512 byte writes to a TCP stream – for `write()` call to the TCP socket
- These data may be delivered as – four 512 byte chunks, two 1024 byte chunks, one 2048 byte chunk or some other way
- There is no way for the receiver to detect the unit(s) in which the data were written by the sending process.

The TCP Protocol – The Header



Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

TCP Sequence Number and Acknowledgement Number

- 32 bits sequence number and acknowledgement number
- Every byte on a TCP connection has its own 32 bit sequence number – a **byte stream** oriented connection
- TCP uses sliding window based flow control – the acknowledgement number contains next expected byte in order, which acknowledges the **cumulative bytes** that has been received by the receiver.
  - ACK number 31245 means that the receiver has correctly received up to 31244 bytes and expecting for byte 31245

TCP Segments

- The sending and receiving TCP entities exchange data in the form of **segments**.

- A TCP segment consists of a fixed 20 byte header (plus an optional part) followed by zero or more data bytes.

TCP Segments

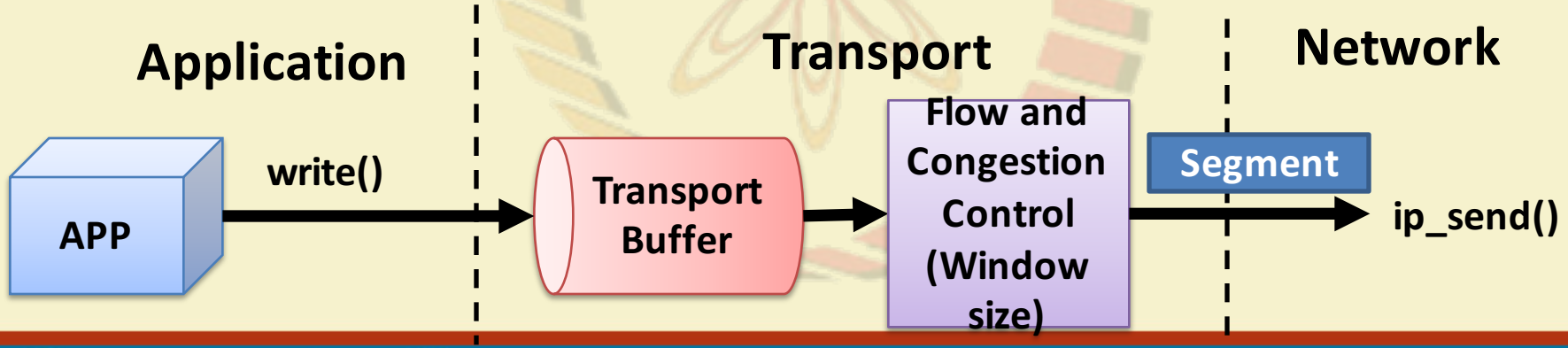
- TCP can accumulate data from several `write()` calls into one segment, or split data from one `write()` into multiple segments

- A segment size is restricted by two parameters
  - IP Payload (65515 bytes)
  - Maximum Transmission Unit (MTU) of the link

How a TCP Segment is Created

- `Write()` calls from the applications write data to the TCP sender buffer.

- Sender maintains a dynamic window size based on the flow and congestion control algorithm



How a TCP Segment is Created

- Modern implementations of TCP uses **path MTU discovery** to determine the MTU of the end-to-end path (uses ICMP protocol), and sets up the **Maximum Segment Size** (**MSS**) during connection establishment
  - May depend on other parameters (buffer implementation).

- Check the sender window after receiving an ACK. If the window size is less than MSS, construct a single segment; otherwise construct multiple segments, each equals to the MSS

Challenges in TCP Design

- Segments are constructed dynamically, so retransmissions do not guarantee the retransmission of the same data segment – a retransmission may contain additional data or less data

- Segments may arrive out-of-order. TCP receiver should handle out-of-order segments in a proper way, so that data wastage is minimized.

Window Size field in the TCP Segment Header

- Flow control in TCP is handled using a variable sized sliding window.
- The *window size* field tells how many bytes the receiver can receive based on the current free size at its buffer space.
- **What is meant by window size 0?**
- TCP Acknowledgement – combination of acknowledgement number and window size


thank you!

IIT KHARAGPUR

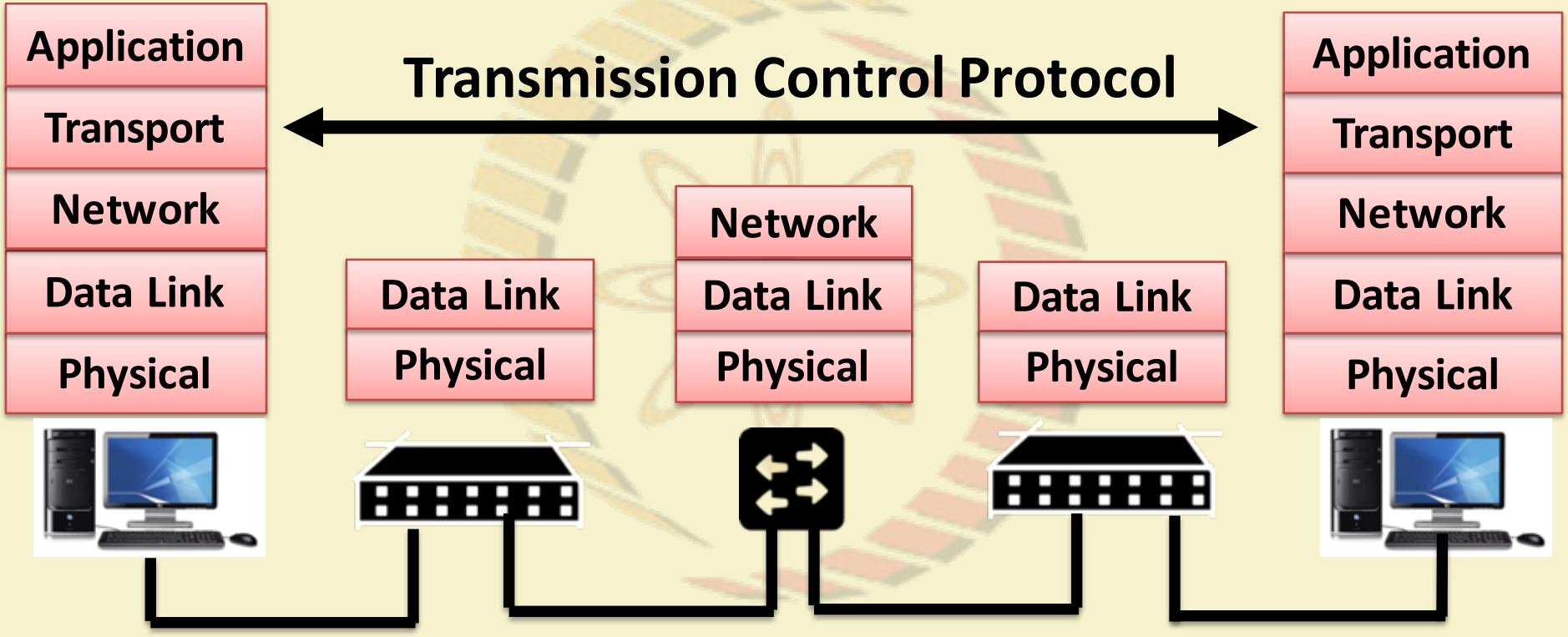
NPTEL ONLINE
CERTIFICATION COURSES

COMPUTER NETWORKS AND INTERNET PROTOCOLS

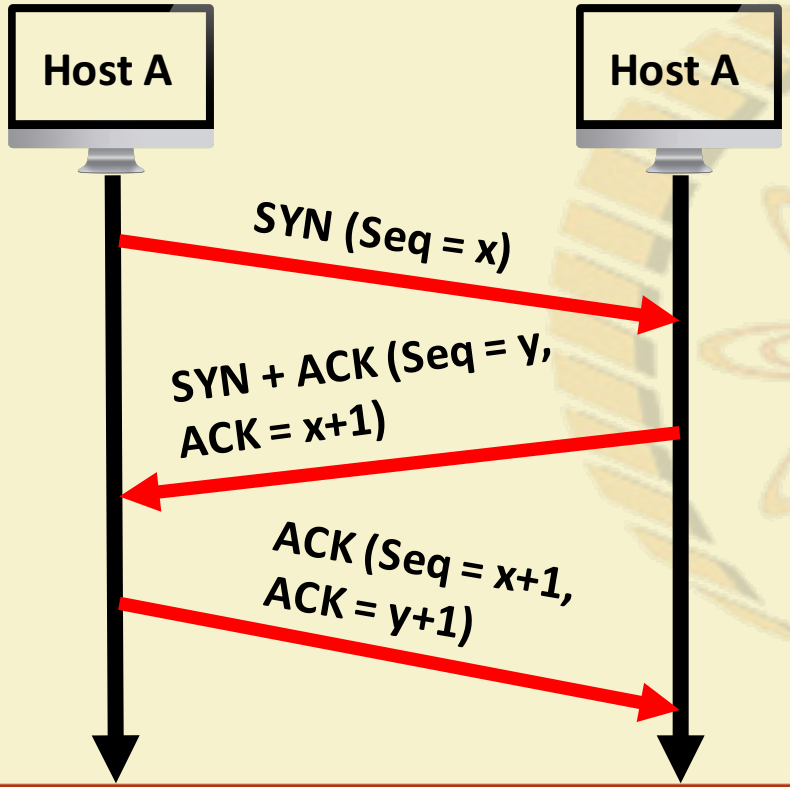
SOUMYA K GHOSH
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

SANDIP CHAKRABORTY
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

Transmission Control Protocol II (Connections)

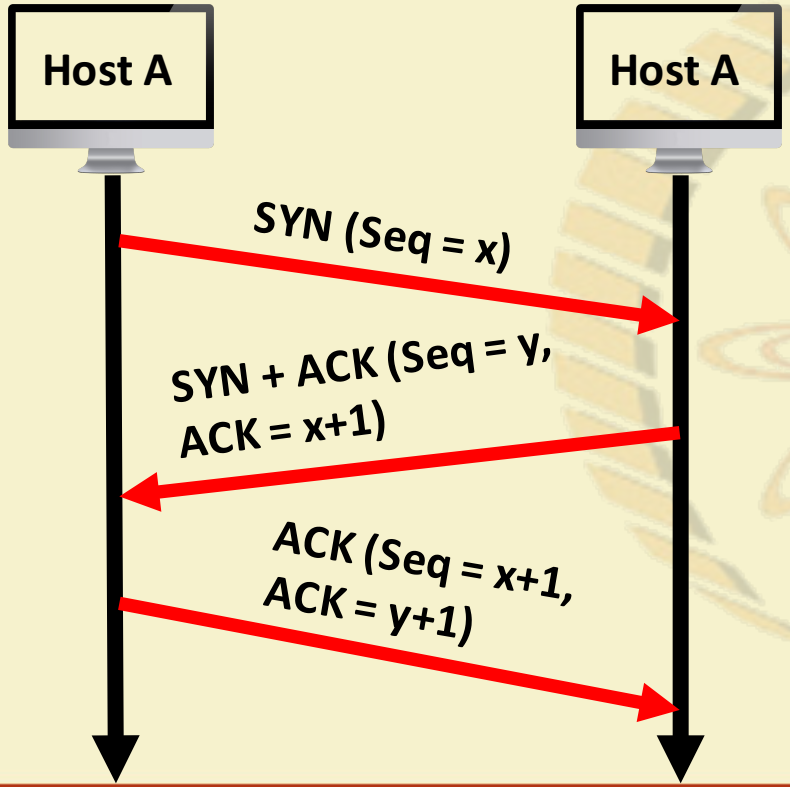


TCP Connection Establishment



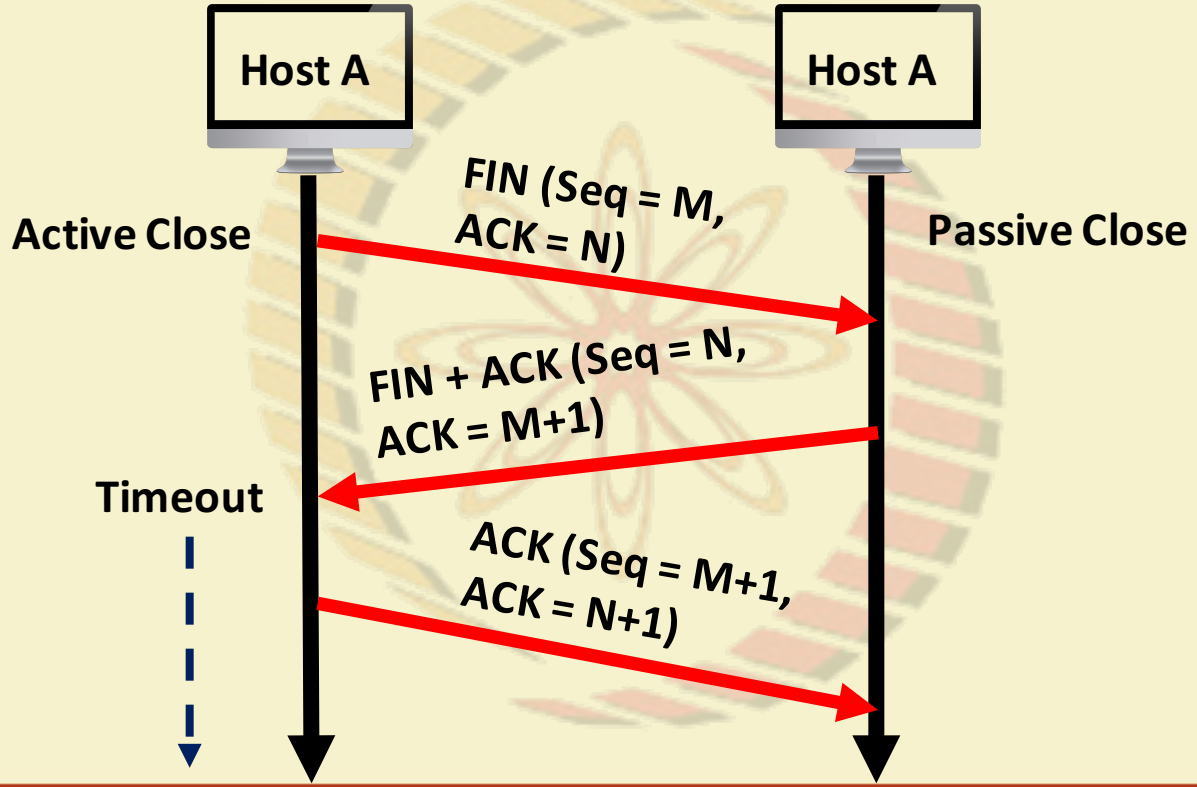
- **How to choose the initial sequence number?**
  - Avoid delayed duplicates, do now generate the initial sequence number for every connection from 0
  - Original implementation of TCP used a clock based approach, the clock ticked every 4 microseconds, the value of the clock cycles from 0 to 2^{32}-1. The value of the clock gives the initial sequence number

TCP Connection Establishment

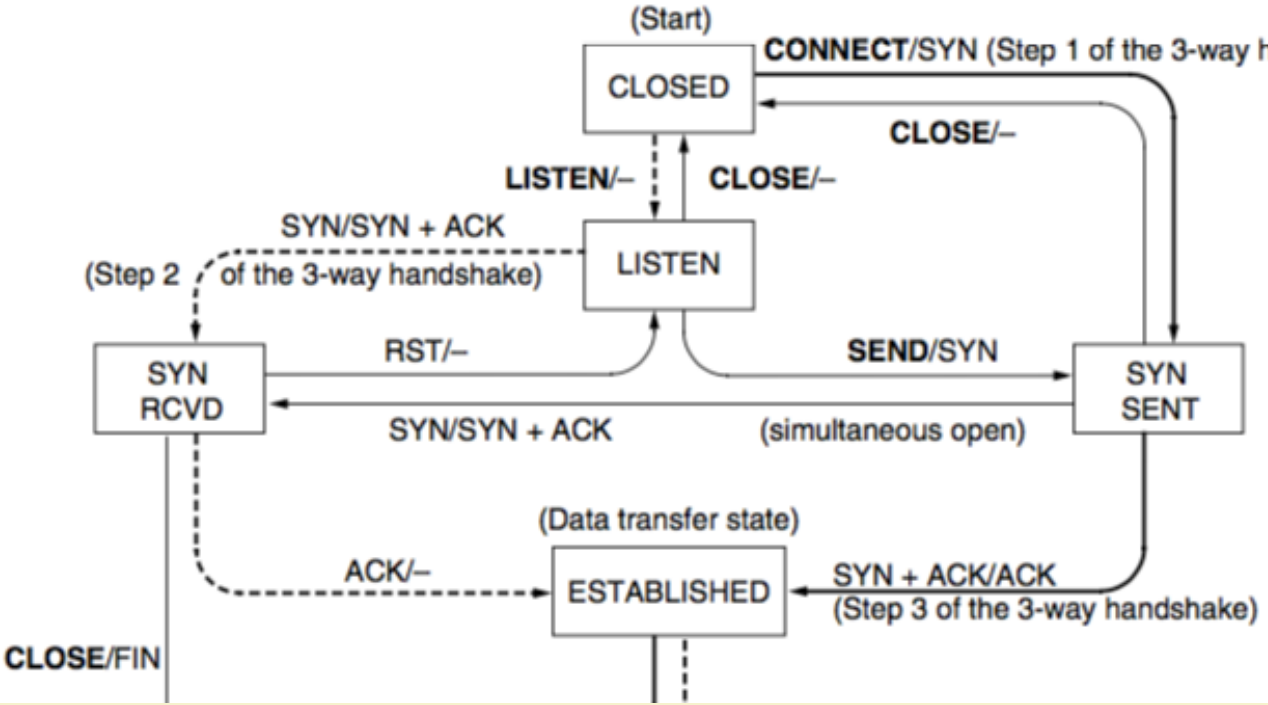


- **TCP SYN flood attack**
  - Solution: Use cryptographic function to generate sequence numbers

TCP Connection Release



TCP State Transition Diagram – Connection Modeling

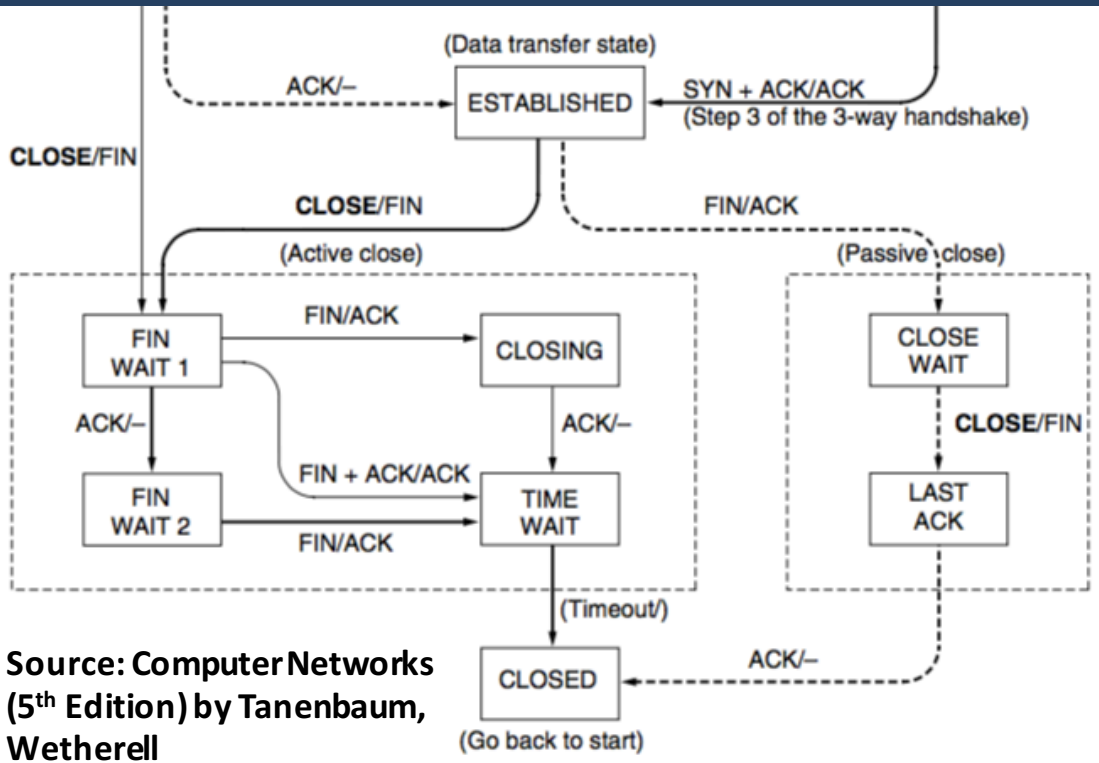


State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIME 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

Event/Action
Dashed: Server
Solid: Client

Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

TCP State Transition Diagram – Connection Modeling



Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

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
TIME 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

Event/Action

Dashed: Server

Solid: Client


thank you!