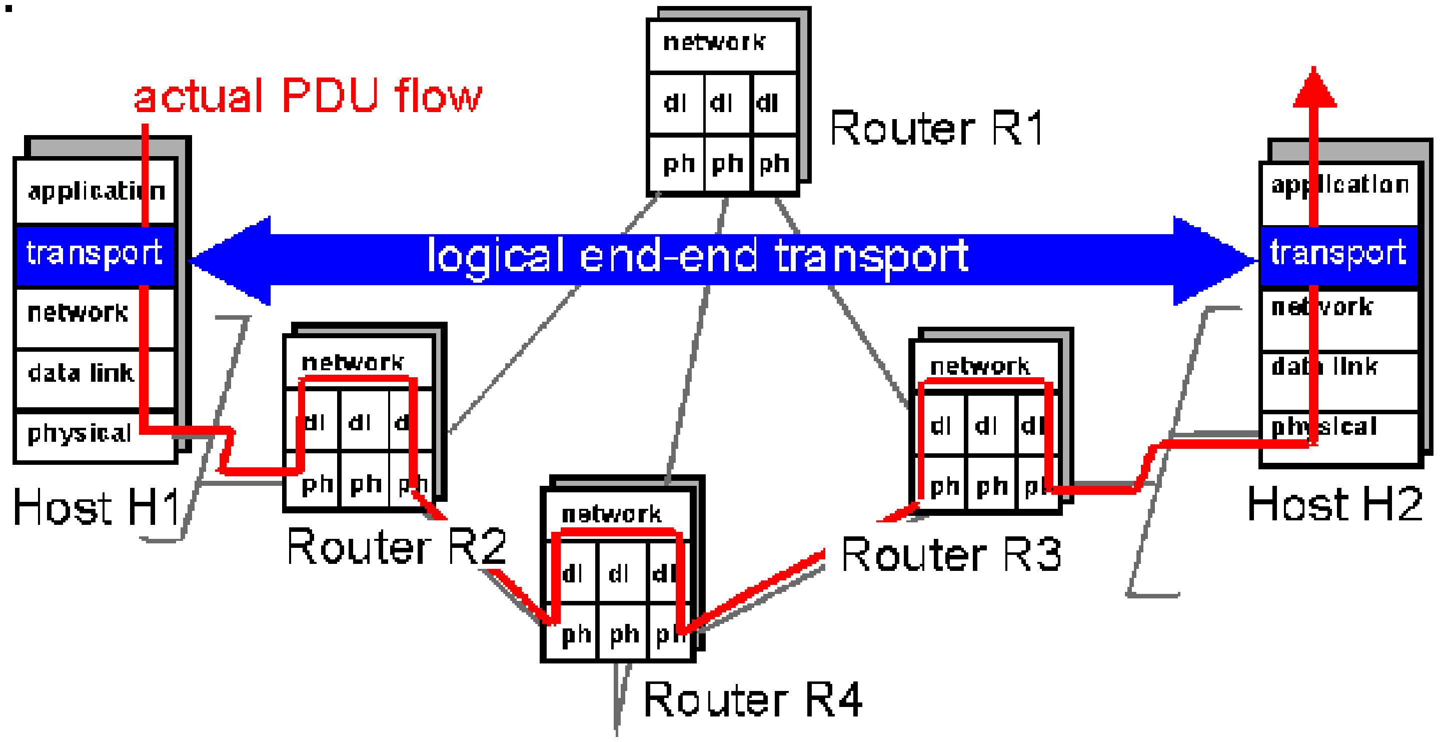
Transport Layer – II :
Flow Control andCongestion Control

Error Control and Flow Control

- **Error Control:** to ensure that the data is delivered with the desired level of reliability, usually that all of the data is delivered without any errors.
- **Flow Control:** to keep a fast transmitter from overrunning a slow receiver.



Error Control and Flow Control

- Recap from Link Layer:

1. A frame carries an error-detecting code (e.g., a CRC or checksum) that is used to check if the information was correctly received.
2. A frame carries a sequence number to identify itself and is retransmitted by the sender until it receives an acknowledgement of successful receipt from the receiver. This is called ARQ (Automatic Repeat reQuest).
3. There is a maximum number of frames that the sender will allow to be outstanding at any time, pausing if the receiver is not acknowledging frames quickly enough. If this maximum is one packet the protocol is called stop-and-wait. Larger windows enable pipelining and improve performance on long, fast links.
4. The sliding window protocol combines these features and is also used to support bidirectional data transfer.

Error Control and Flow Control : Key difference(s) with Link Layer

1. The transport layer checksum protects a segment while it crosses an entire network path. It is an end-to-end check, which is not the same as having a check on every link.

The transport layer check that runs end-to-end is essential for correctness, and the link layer checks are not essential but nonetheless valuable for improving performance (since without them a corrupted packet can be sent along the entire path unnecessarily).

Difference in Degree:

- For wireless links, retransmissions are necessary and sliding window size is small. For wired links, retransmissions may not be necessary. For TCP, larger window-size and retransmissions both are necessary.

- Larger window sizes means buffering at sender and receiver becomes important issue.

Buffering:

1. Since a host may have many connections, each of which is treated separately, it may need a substantial amount of buffering for the sliding windows.
2. The buffers are needed at sender to hold all transmitted but as yet unacknowledged segments.
3. Since the sender is buffering, the receiver may or may not dedicate specific buffers to specific connections, as it sees fit.

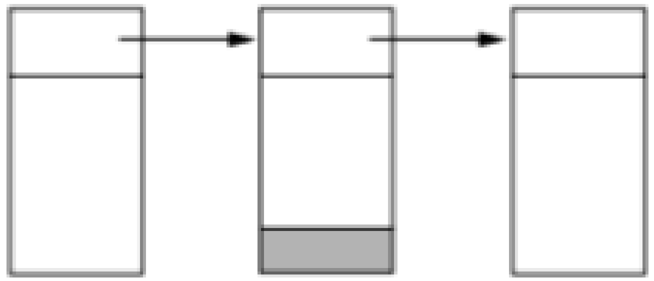
The best trade-off between source buffering and destination buffering depends on the type of traffic carried by the connection.

1. For low-bandwidth bursty traffic – No dedicated buffers
2. For high-bandwidth traffic – dedicated buffers to achieve high performance

How to organize the buffer pool?

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)

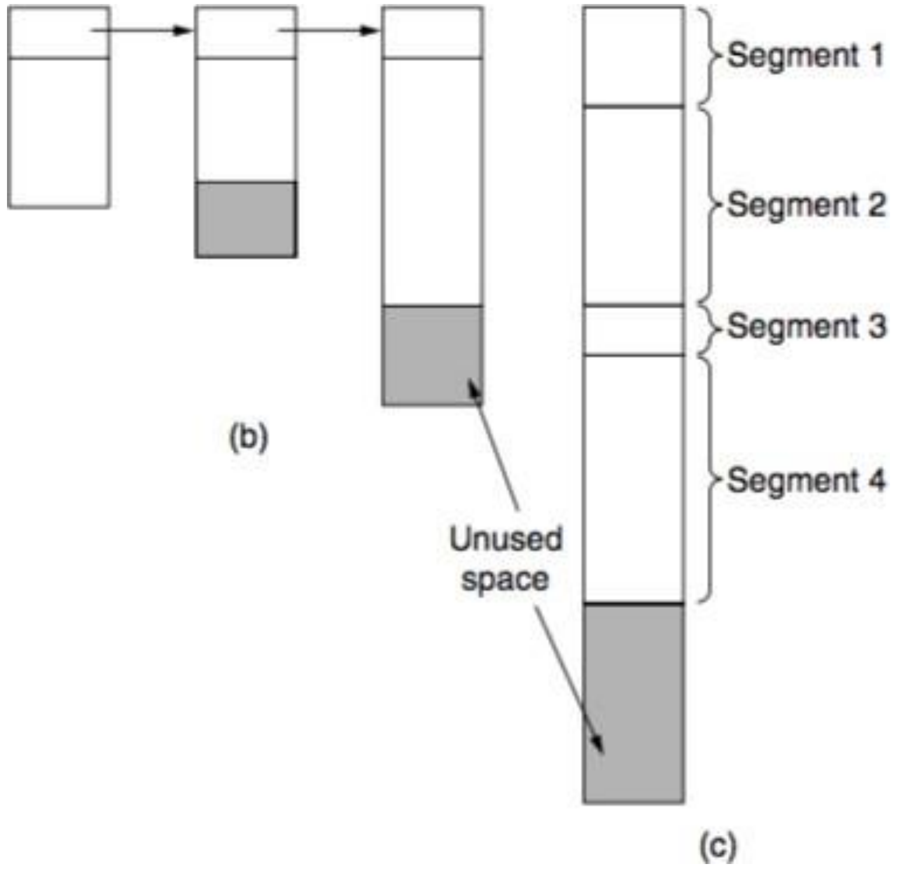


- Space would be wasted if segment sizes are widely varied
- Small buffer size – multiple buffers to store a single segment – added complexity in implementation

Organizing Transport Buffer Pool

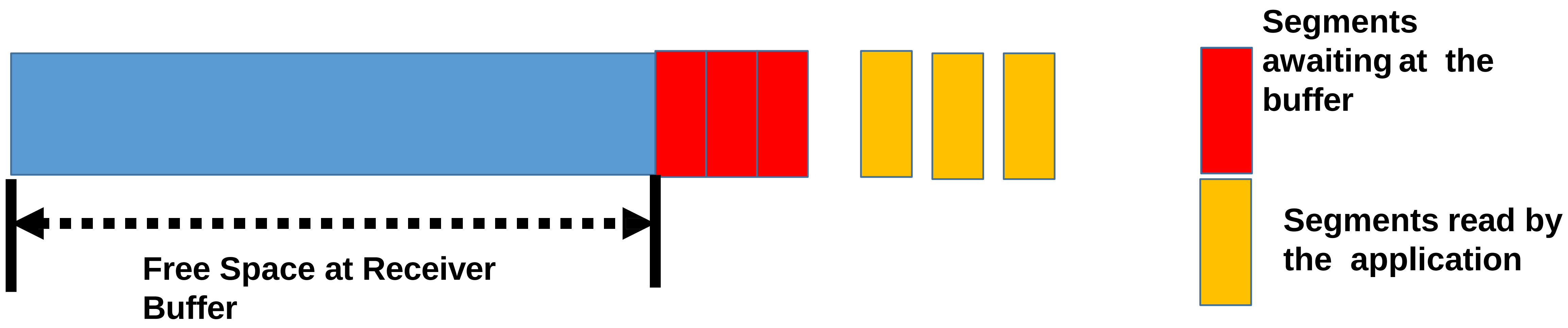
- **Variable size buffers (b)**
  - Advantage: better memory utilization
  - Disadvantage: Complicated implementation

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



Dynamic Buffer Management for Window Based Flow Control

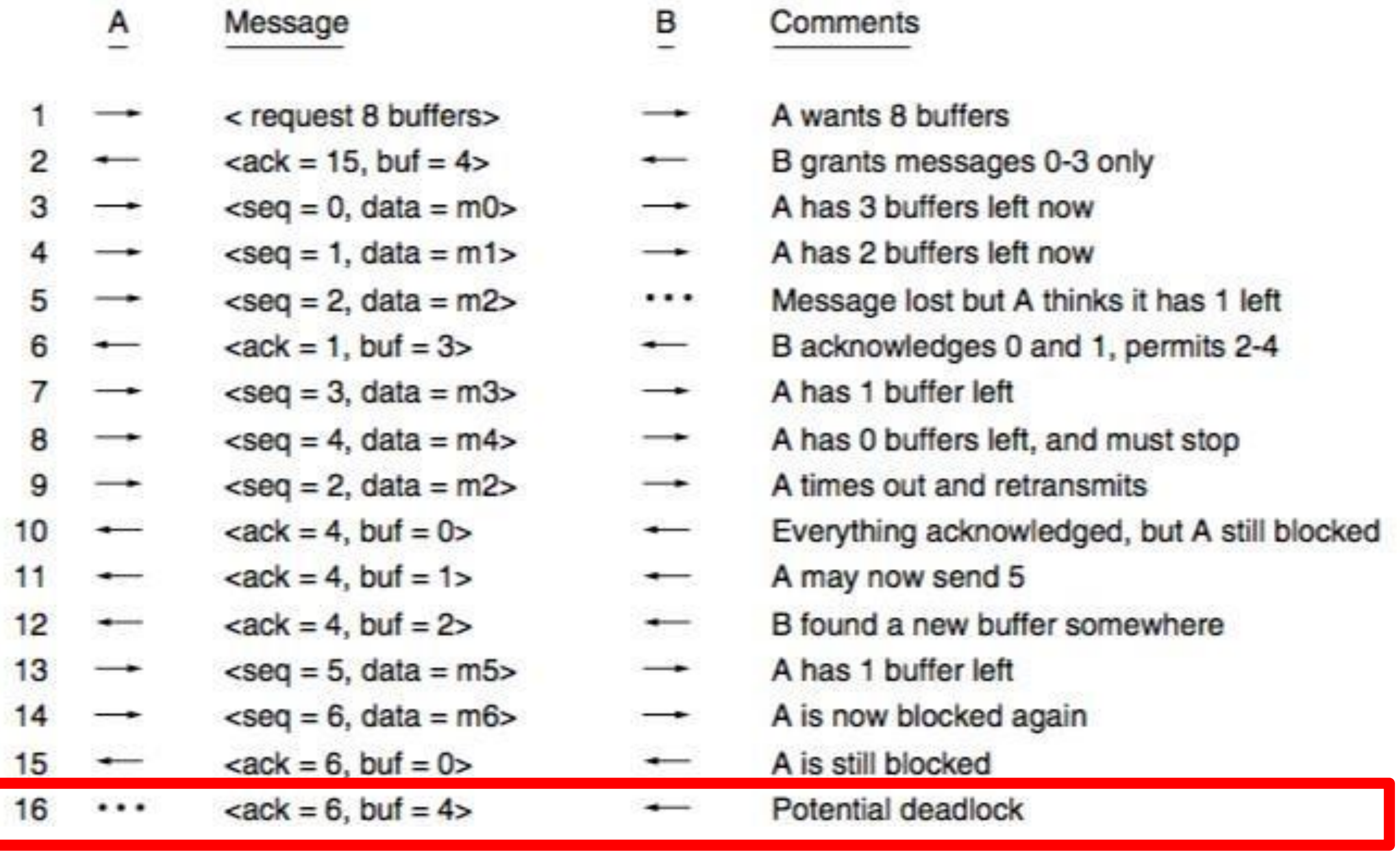
- As connections are opened and closed and as the traffic pattern changes, sender and receiver needs to dynamically adjust buffer allocations
- Based on the rate difference between the **receive rate by the transport entity** and the **receive rate by the application**, the available size of the receiver buffer changes


Segments awaiting at the buffer
Segments read by the application
Free Space at Receiver Buffer

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

- Decouple the buffering from the acknowledgements, in contrast to the sliding window protocol of LL.

Dynamic Buffer Management for Window Based Flow Control



	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

Another bottleneck: Carrying capacity of the network

1. What is needed is a mechanism that limits transmissions from the sender based on the network's carrying capacity rather than on the receiver's buffering capacity.

2. A sliding window flow-control scheme in which the sender dynamically adjusts the window size to match the network's carrying capacity.

3. This means that a dynamic sliding window can implement both flow control and congestion control.

4. If the network can handle c segments/sec and the round-trip time is r, the sender's window should be cr.

5. With a window of this size, the sender normally operates with the pipeline full.

Crash Recovery: What to do when hosts carash?

Strategy used by sending host	Strategy used by receiving host: First ACK, then write			First write, then ACK		
	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

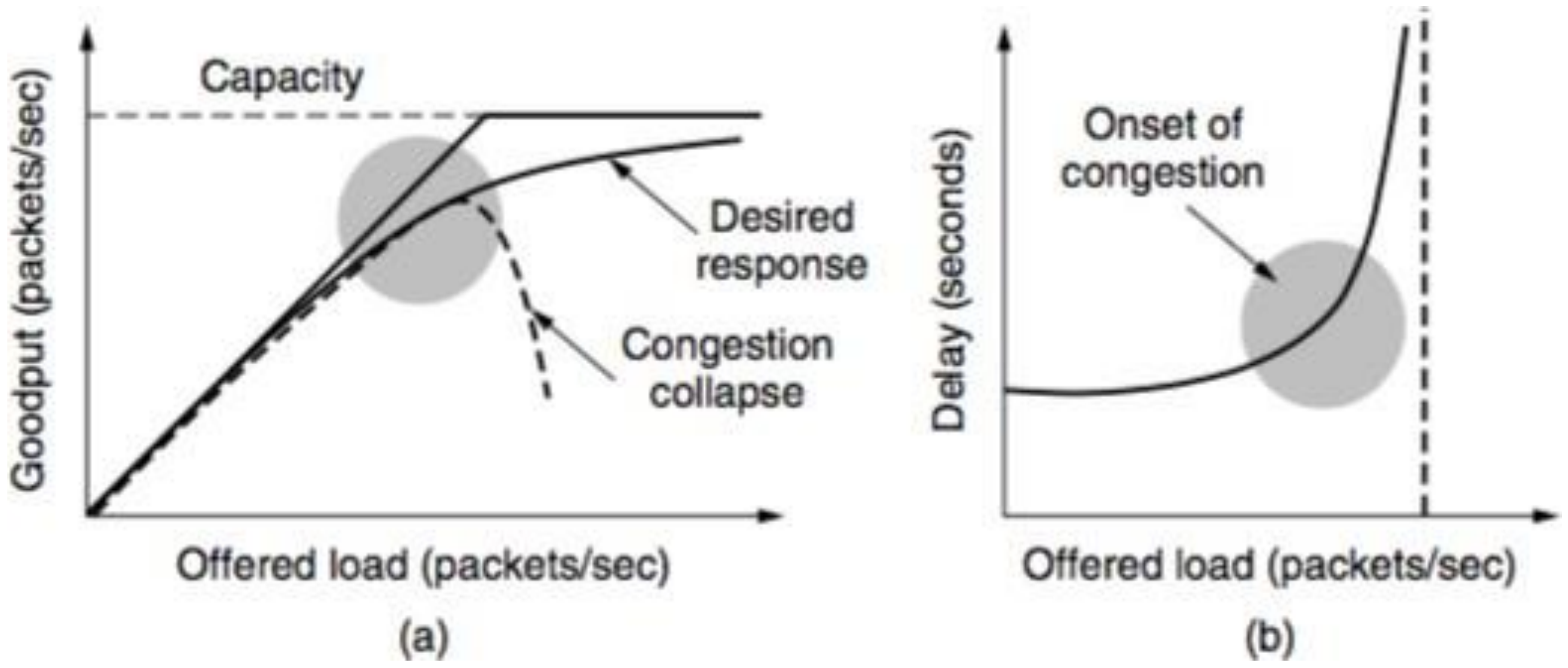
OK	= Protocol functions correctly
DUP	= Protocol generates a duplicate message
LOST	= Protocol loses a message

Congestion Control

- Controlling congestion is the combined responsibility of the network and transport layers.
- We need to regulate the rate at which packets are sent in to the network.
- The goal is more than to simply avoid congestion. It is to find a good allocation of bandwidth to the transport entities that are using the network.
- A good allocation will deliver good performance because it uses all the available bandwidth but avoids congestion, it will be fair across competing transport entities, and it will quickly track changes in traffic demands.

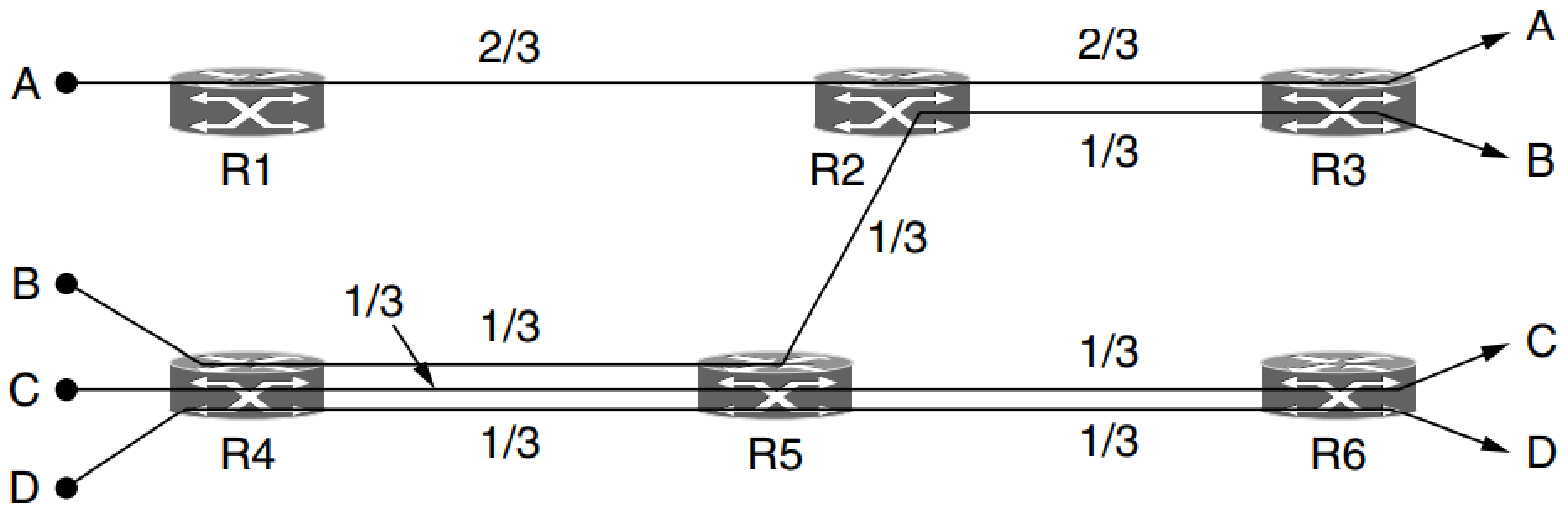
Efficiency and Power

- An efficient allocation of bandwidth across transport entities will use all of the network capacity that is available.
- Does not mean 100 Mbps divided among 5 entities means 20 Mbps each !! The reason is that the traffic is often bursty.



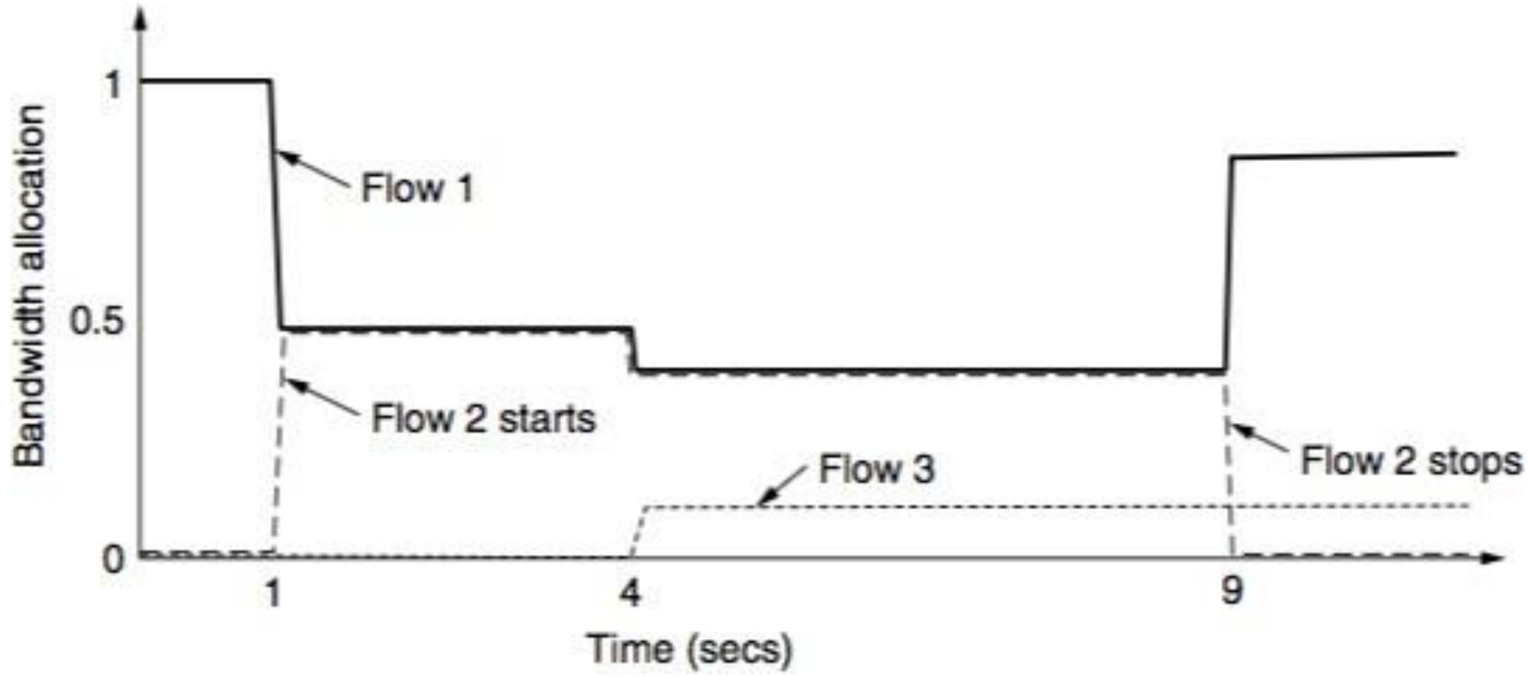
(a) Goodput and (b) delay as a function of offered load.

Max-Min Fairness: An Example



Max-min allocations can be computed given a global knowledge of the network.

Convergence

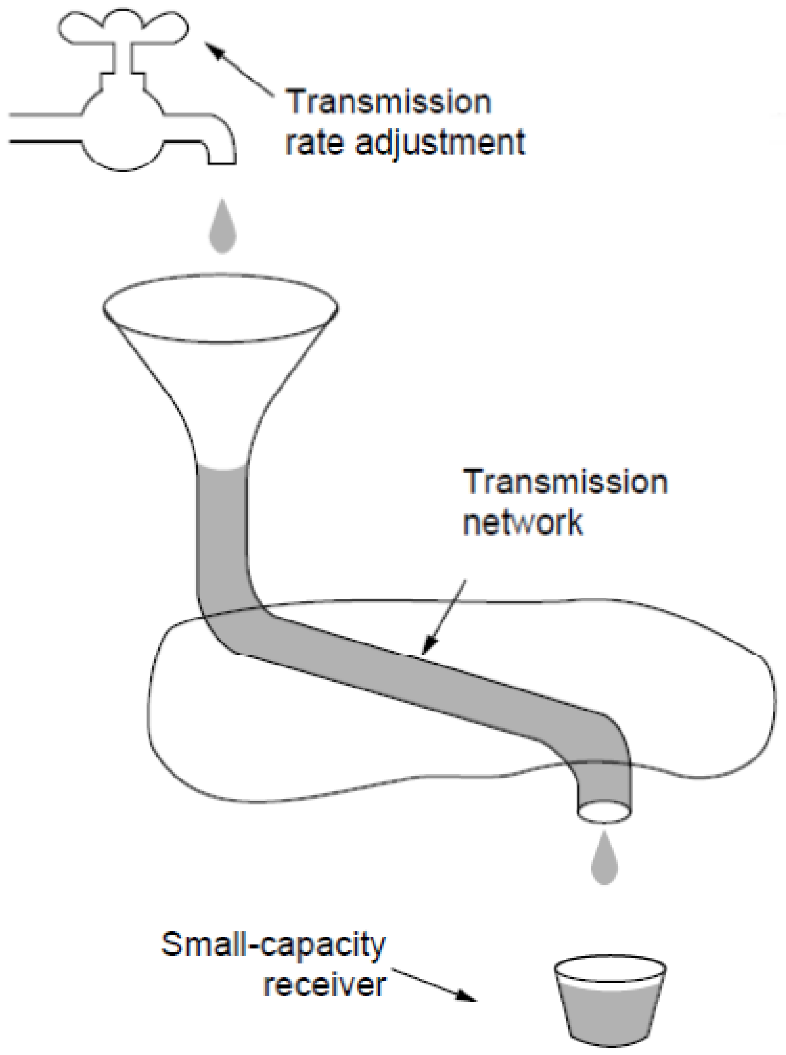


Changing Bandwidth Allocation over Time

Regulating the sending rate

Sender may need to slow down for different reasons:

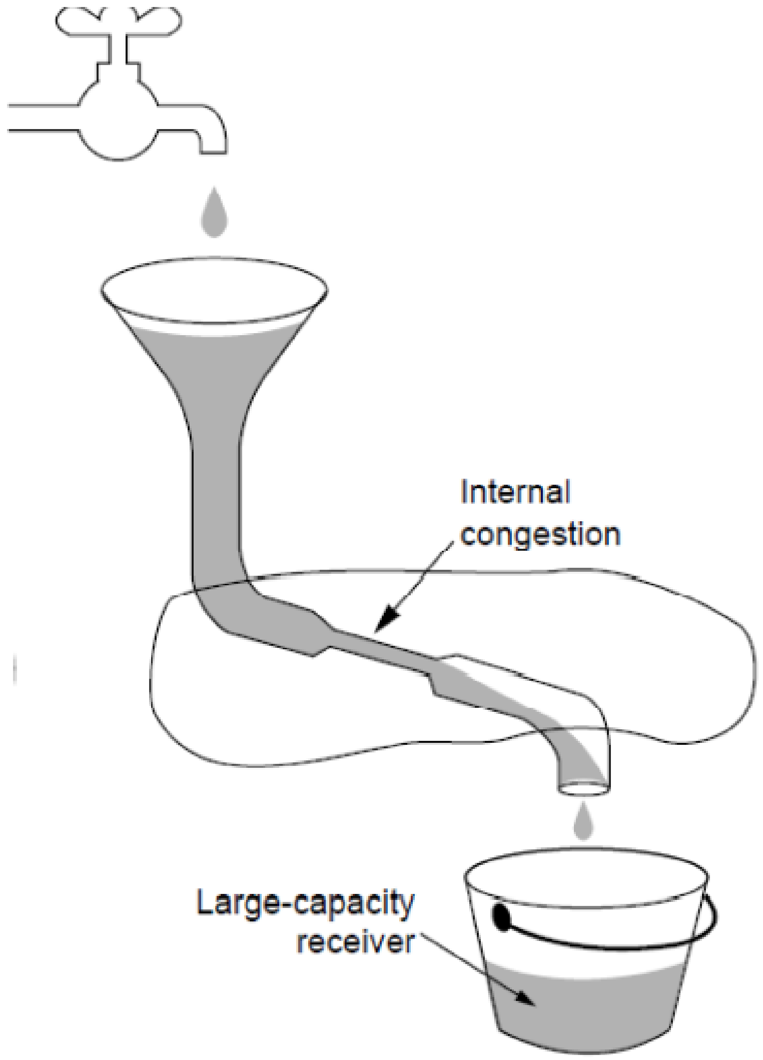
- Flow control, when the receiver is not fast enough [right]
- Congestion, when the network is not fast enough [over]



A fast network feeding a low-capacity receiver
→ flow control is needed

Regulating the sending rate

Our focus is dealing with this problem – congestion



A slow network feeding a high-capacity receiver → congestion control is needed

Regulating the sending rate

- The way that a transport protocol should regulate the sending rate depends on the form of the feedback returned by the network.
- **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

Regulating the sending rate

Different congestion signals the network may use to tell the transport endpoint to slow down (or speed up)

Protocol	Signal	Explicit?	Precise?
XCP	Rate to use	Yes	Yes
TCP with ECN	Congestion warning	Yes	No
FAST TCP	End-to-end delay	No	Yes
CUBIC TCP	Packet loss	No	No
TCP	Packet loss	No	No

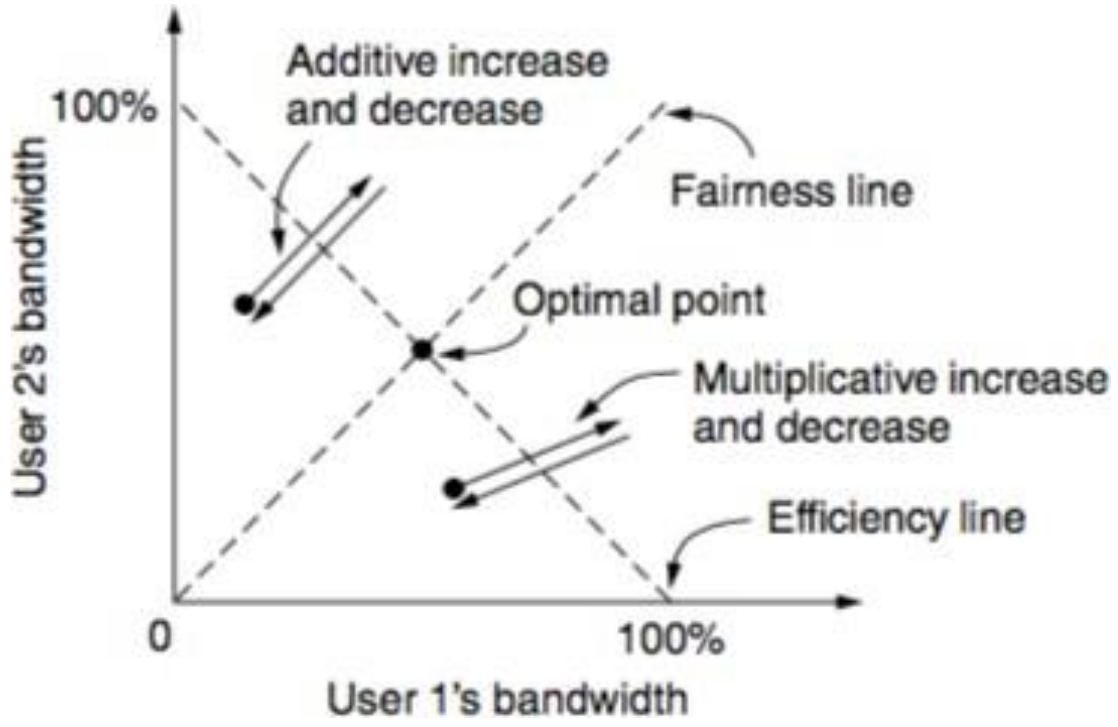
AIMD: Efficient and Fair Operating Point for Congestion Control

- **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

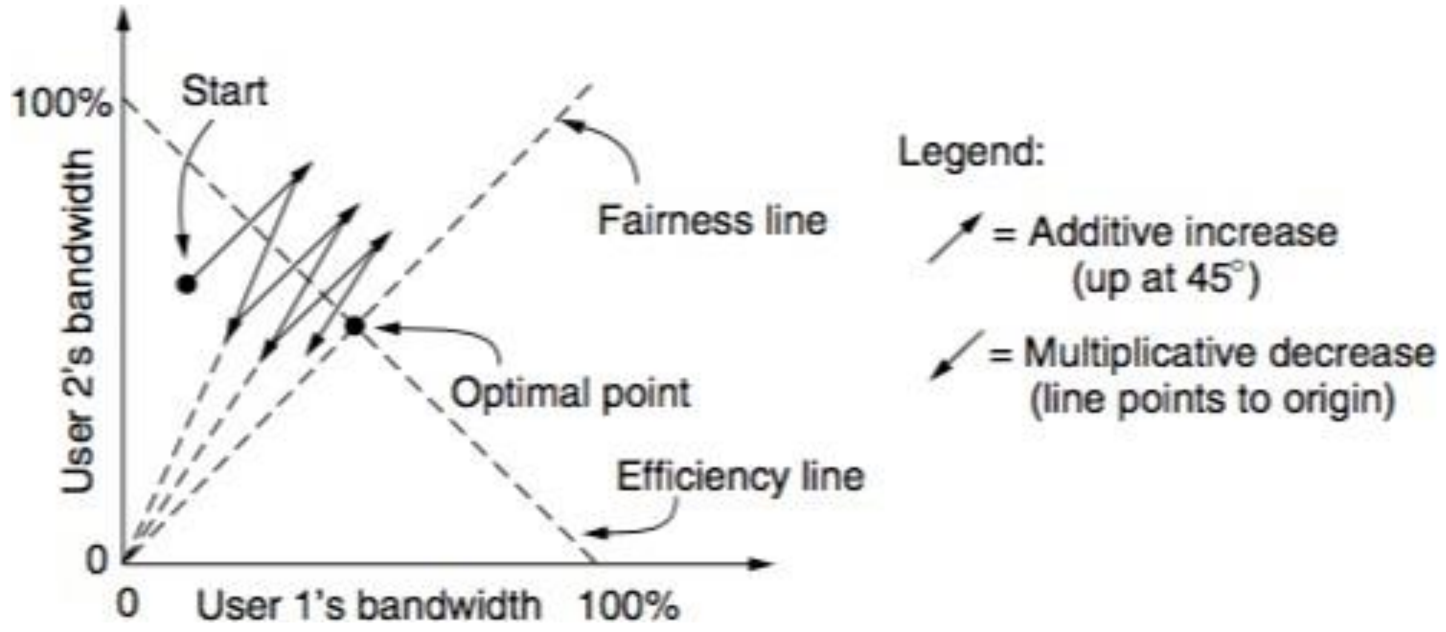
$$w(t+1) = \begin{cases} w(t) + a & \text{if congestion is not detected} \\ w(t) \times b & \text{if congestion is detected} \end{cases}$$

AIMD: Design Rationale (Two Flows Example)



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

AIMD – Design Rationale (Two Flows Example)



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

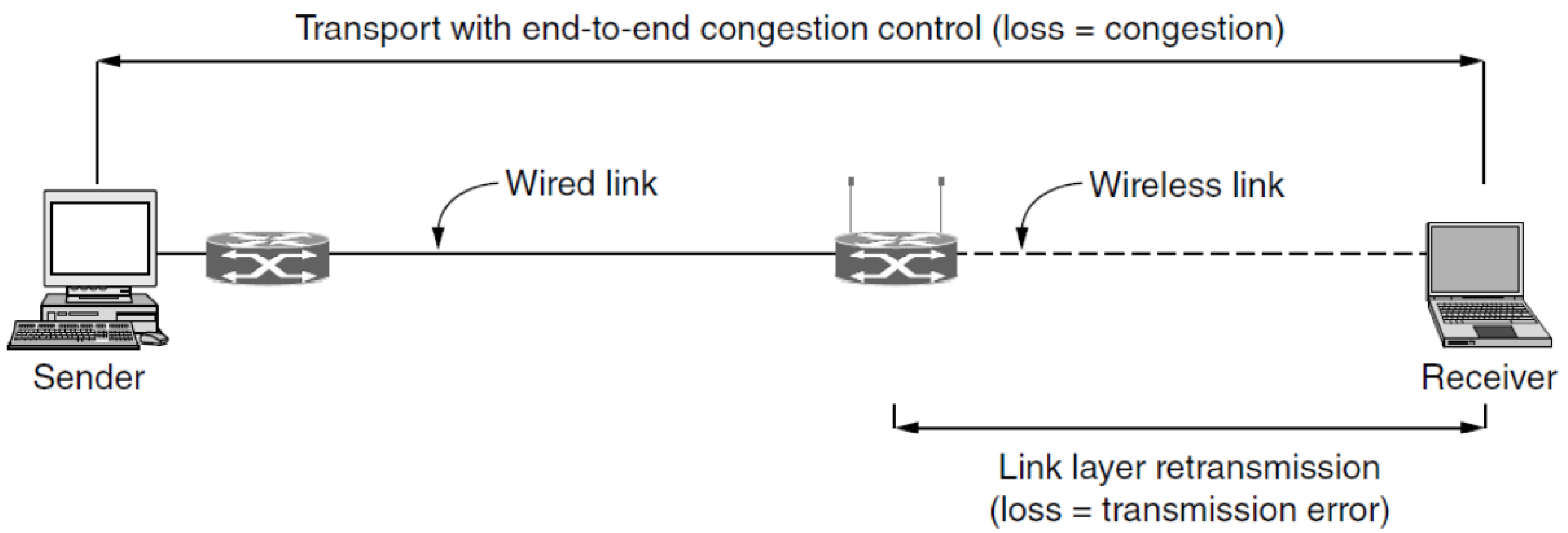
Wireless Issues

Wireless links lose packets due to transmission errors

- Do not want to confuse this loss with congestion
- Or connection will run slowly over wireless links!

Strategy:

- Wireless links use ARQ, which masks errors


Transport with end-to-end congestion control (loss = congestion)
Wired link
Wireless link
Sender
Receiver
Link layer retransmission
(loss = transmission error)

Next we will look into the Transmission Control Protocol