UNIVERSITY OF Southampton

School of Electronics and Computer Science

Transport Layer 2

ELEC3227/ELEC6255

Alex Weddell
asw@ecs.soton.ac.uk

Photo by Alessandro Venturi on Unsplash

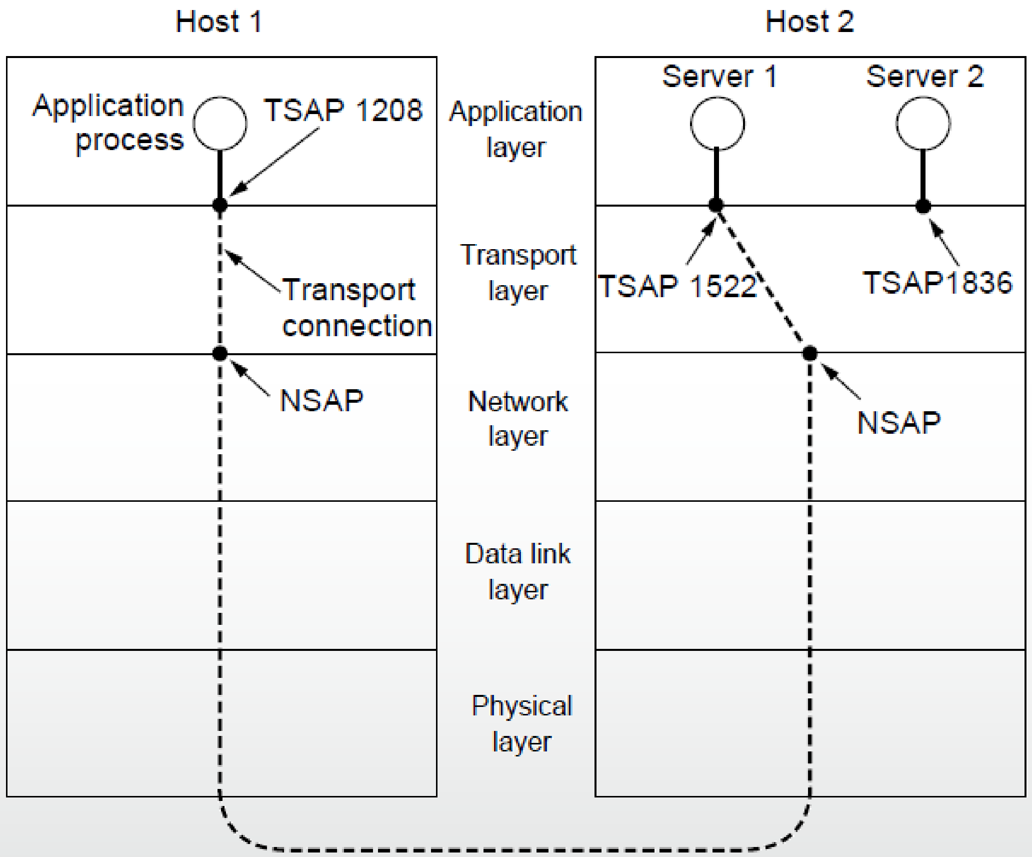
Elements of Transport Protocols

- Addressing
- Connection establishment
- Connection release
- Error control and flow control
- Crash Recovery
- Regulating the Sending Rate

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Addressing

- Transport layer adds TSAPs (Transport Service Access Points)
- Multiple clients and servers can run on a host with a single network (IP) address
- TSAPs are **ports** for TCP/UDP



Connection Establishment

Main aim is to ensure reliability even though packets may be **lost**, **corrupted**, **delayed**, and **duplicated**

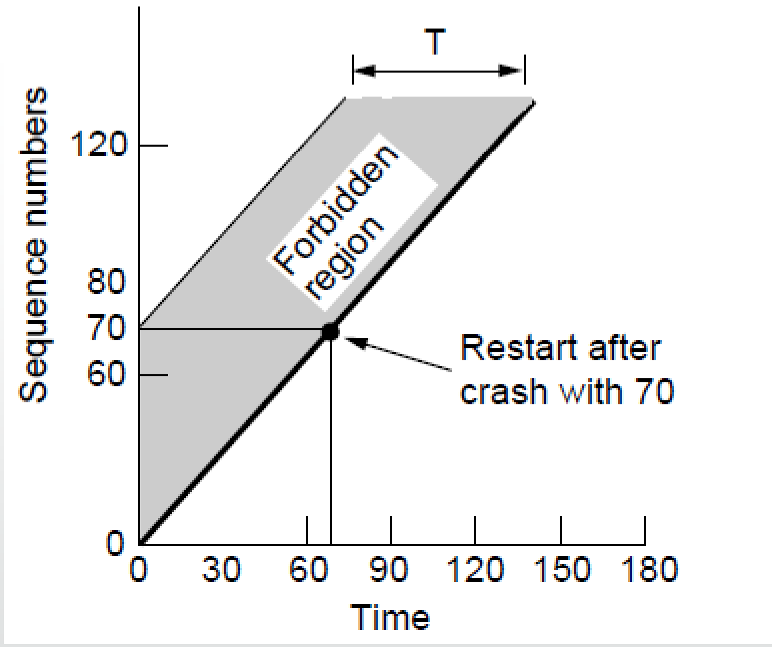
- Don’t treat an old or duplicate packet as new
- Use ARQ and checksums for loss/corruption

Approach:

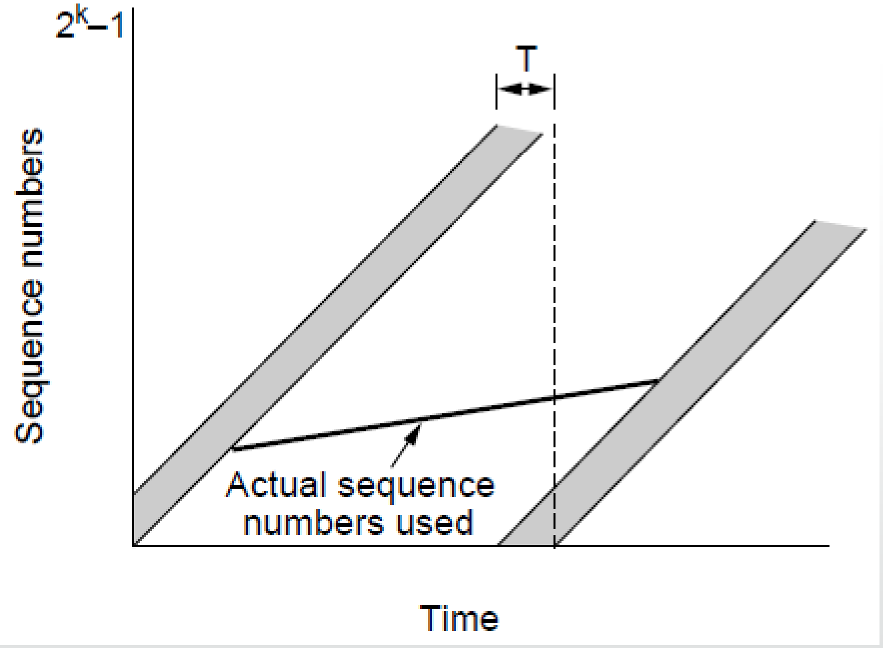
- Don’t reuse sequence numbers within twice the MSL (Maximum Segment Lifetime) of 2T = 240 secs
- Three-way handshake for establishing connection

Sequence Numbers

- Use a sequence number space large enough that it will not wrap, even when sending at full rate
  - Clock (high bits) advances & keeps state over crash



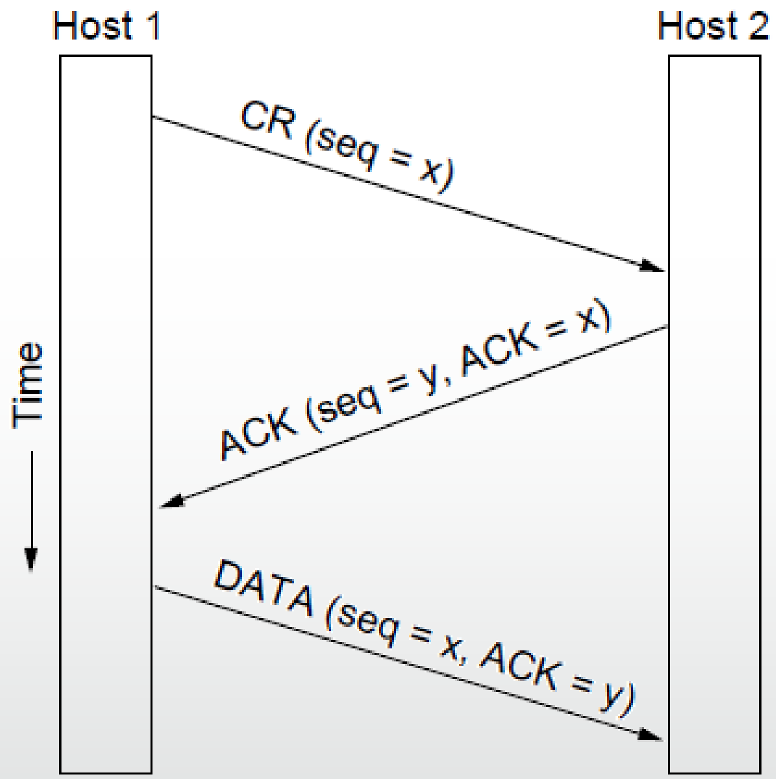
Need seq. number not to wrap within T seconds



Need seq. number not to climb too slowly for too long

Connection Establishment

- Three-way handshake used for initial packet
  - Since no state from previous connection
  - Both hosts contribute fresh seq. numbers
  - CR = Connect Request



UNIVERSITY OF Southampton
School of Electronics and Computer Science

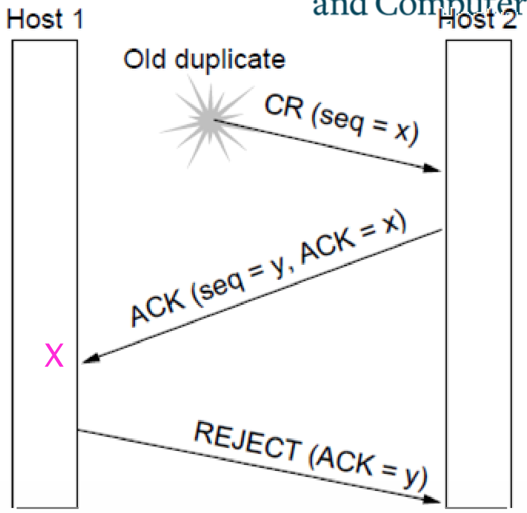
Connection Establishment

- Three-way handshake protects against odd cases:

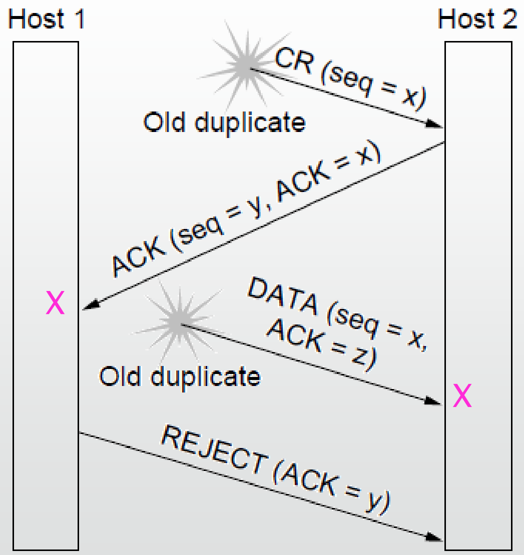
a) Duplicate CR. Spurious ACK does not connect

b) Duplicate CR and DATA. Same plus DATA will be rejected (wrong ACK).

a)



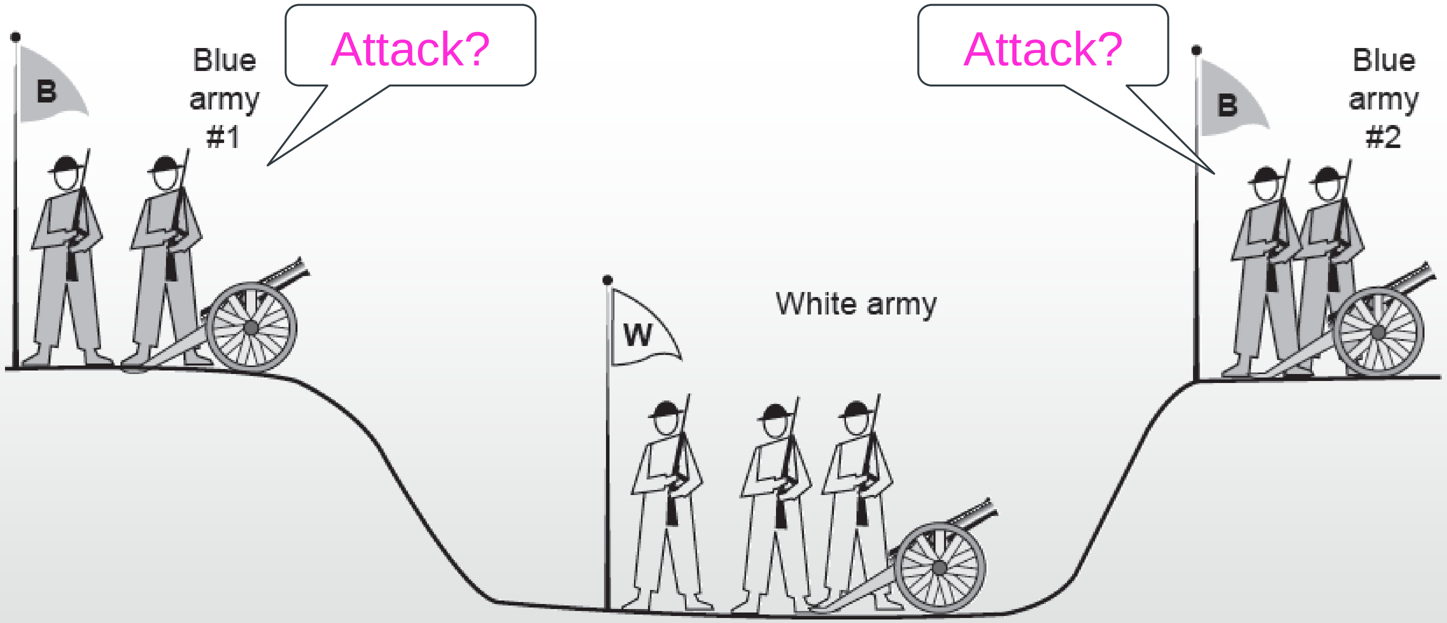
b)



Connection Release

Symmetric release (both sides agree to release) can’t be handled solely by the transport layer

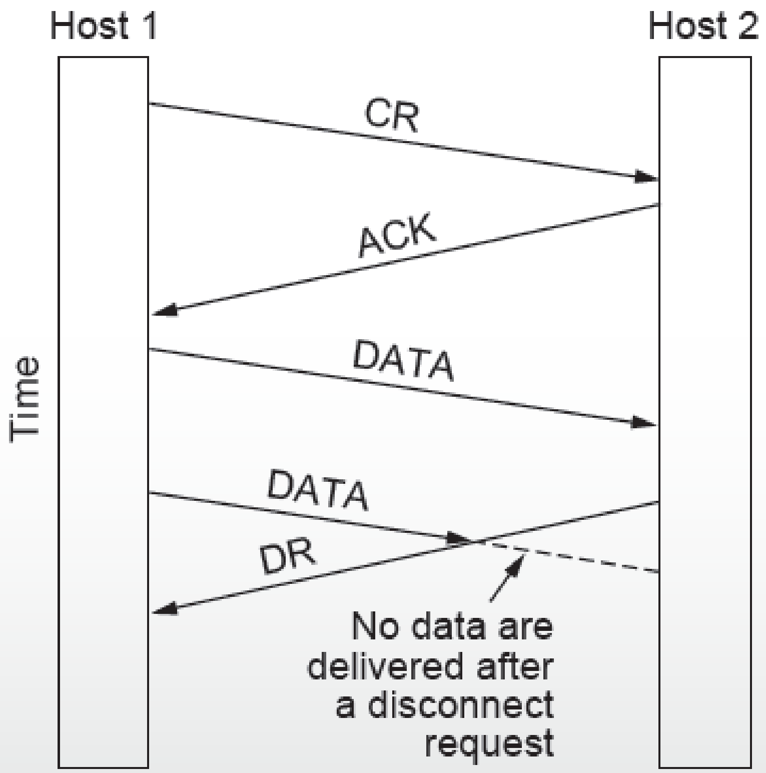
- Two-army problem shows pitfall of agreement



UNIVERSITY OF Southampton
School of Electronics and Computer Science

Connection Release

- Main aim is to ensure reliability while releasing
- Asymmetric release (when one side breaks connection) is abrupt and may lose data

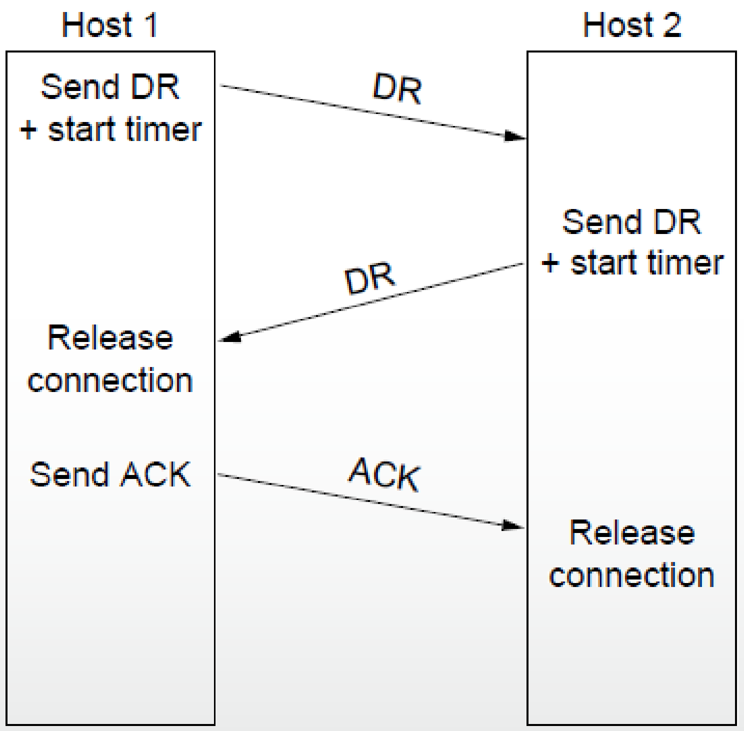


UNIVERSITY OF Southampton
School of Electronics and Computer Science

Connection Release

Normal release sequence, initiated by transport user on Host 1

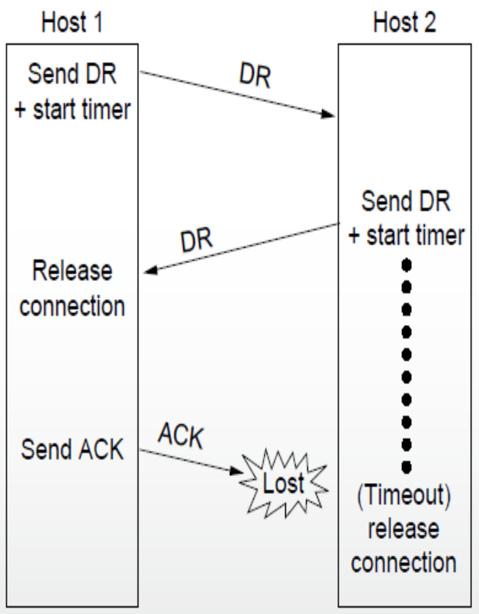
- DR=Disconnect Request
- Both DRs are ACKed by the other side



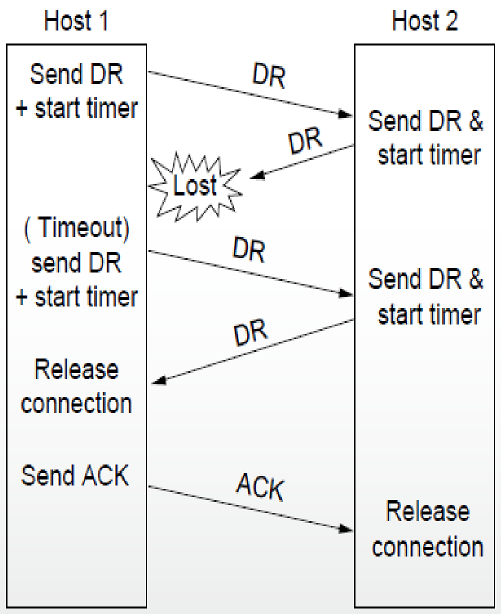
UNIVERSITY OF Southampton
School of Electronics and Computer Science

Connection Release

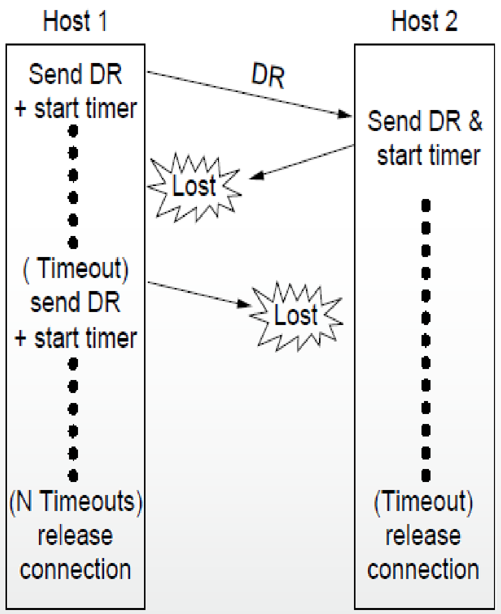
- Error cases are handled with timer and retransmission



Final ACK lost, Host 2 times out



Lost DR causes retransmissions



Host 1
Host 2
Send DR + start timer
DR
Send DR & start timer
Lost
(Timeout) send DR + start timer
Lost
(N Timeouts) release connection
(Timeout) release connection

Extreme: Many lost DRs cause both hosts to timeout

Error Control and Flow Control

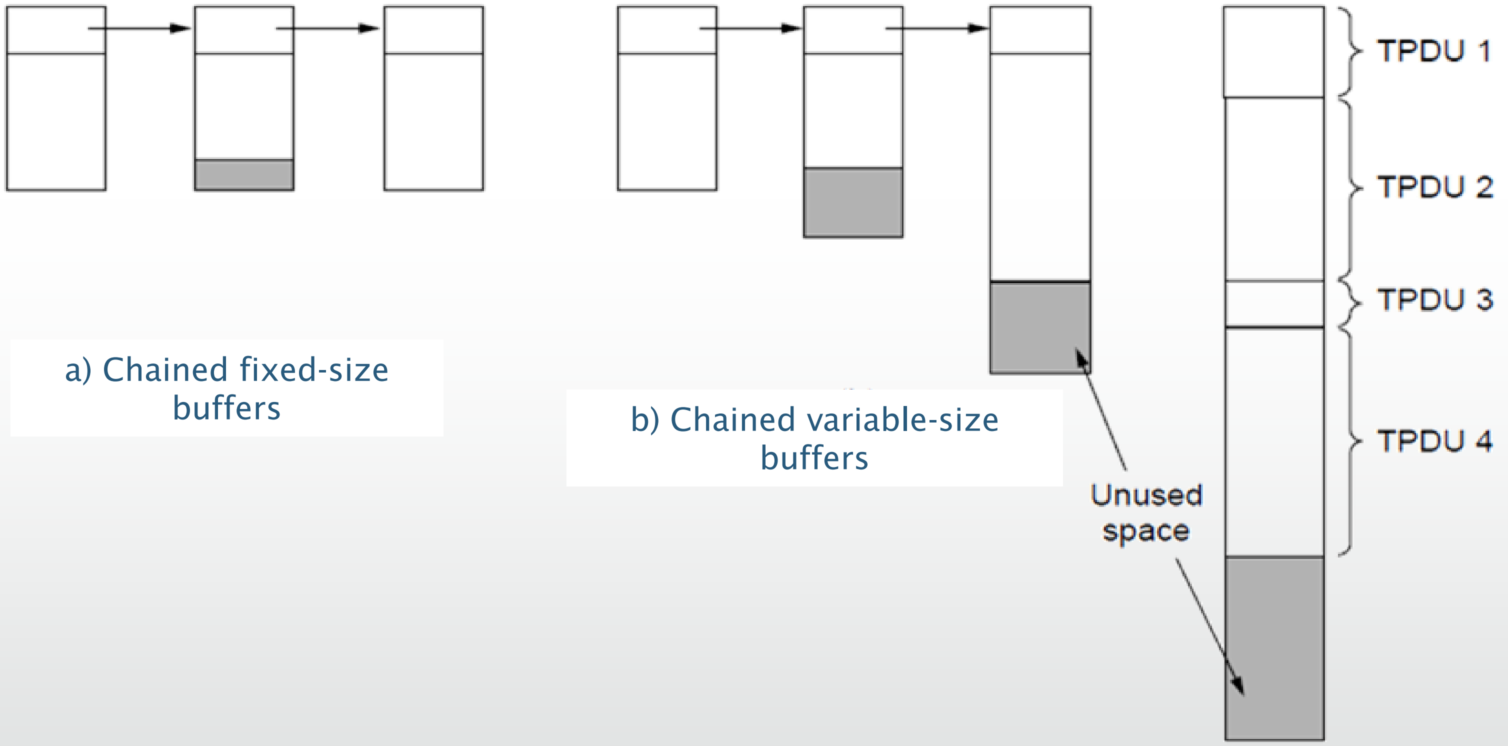
Foundation for error control is a sliding window (from Link layer) with checksums and retransmissions

Flow control manages buffering at sender/receiver

- Issue is that data goes to/from the network and applications at different times
- Window tells sender available buffering at receiver
- Makes a variable-size sliding window

Error Control and Flow Control

- Different buffer strategies trade efficiency / complexity



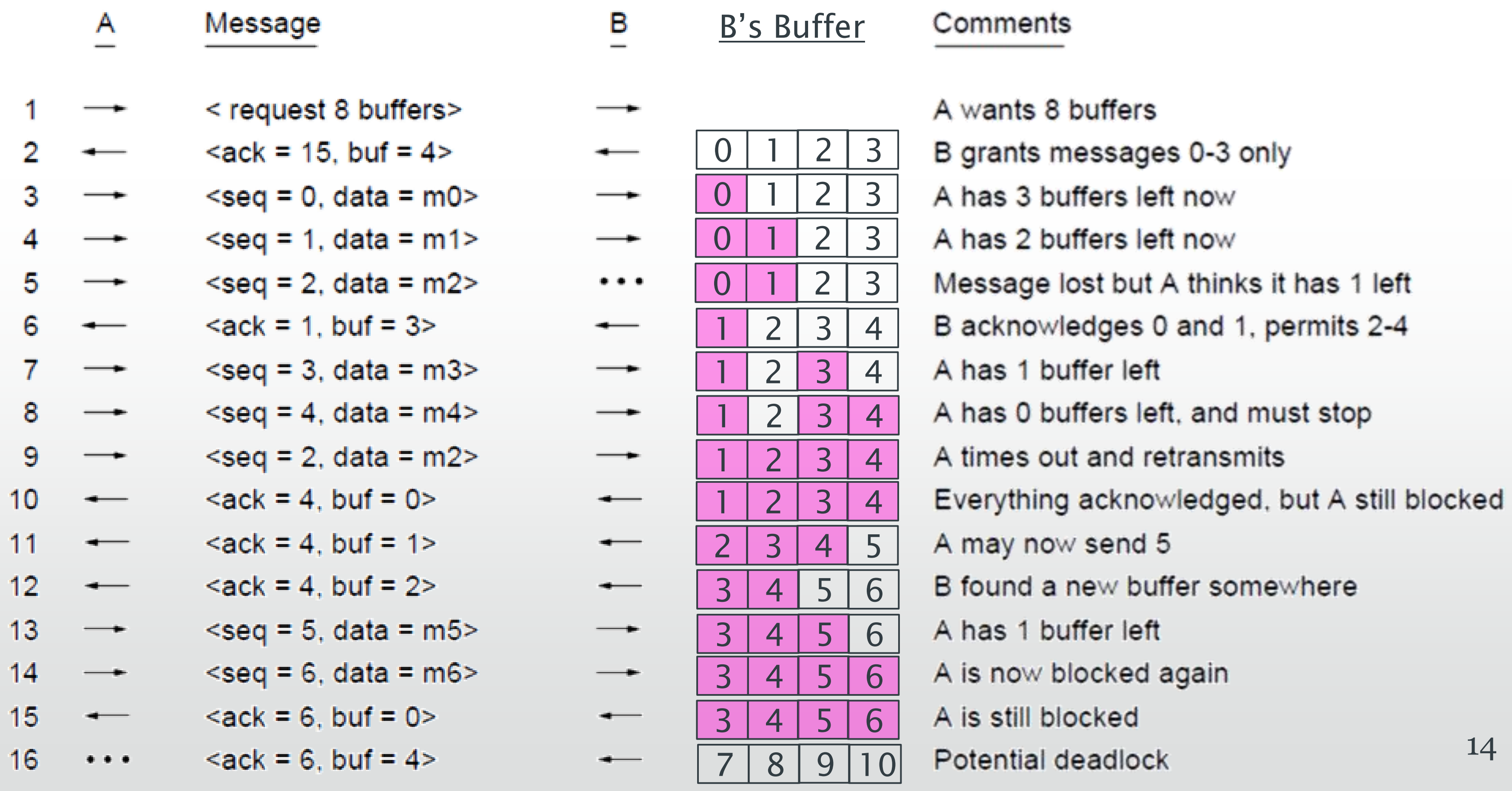
a) Chained fixed-size buffers

b) Chained variable-size buffers

c) One large circular buffer

Error Control and Flow Control

- Flow control example: A’s data is limited by B’s buffer



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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Crash Recovery

- Application needs to help recovering from a C(rash)
  - Transport can fail since A(ck) / W(rite) not atomic

	Strategy used by receiving host					
	First ACK, then write			First write, then ACK		
Strategy used by sending host	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

Two layers are responsible for congestion control:

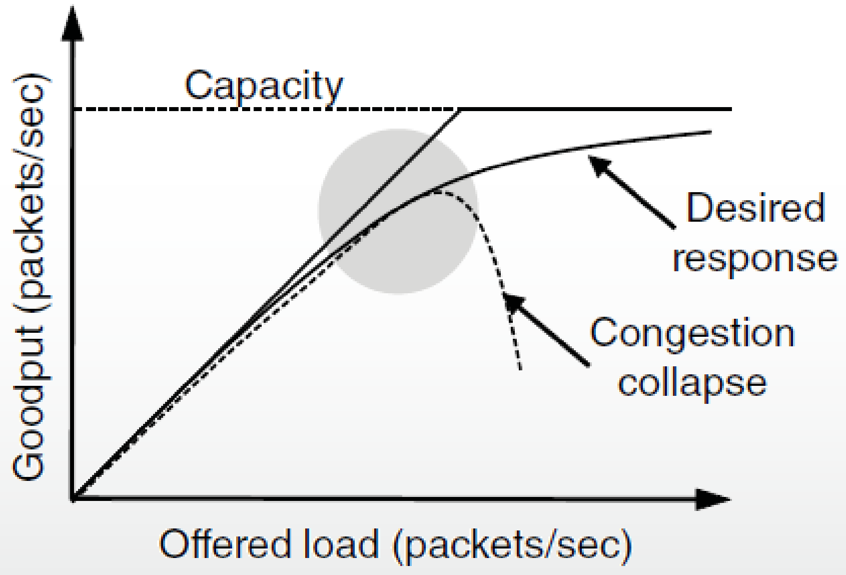
- **Transport** layer, controls the offered load
- **Network** layer, experiences congestion

- "**Goodput**" is useful throughput (minus retransmissions)

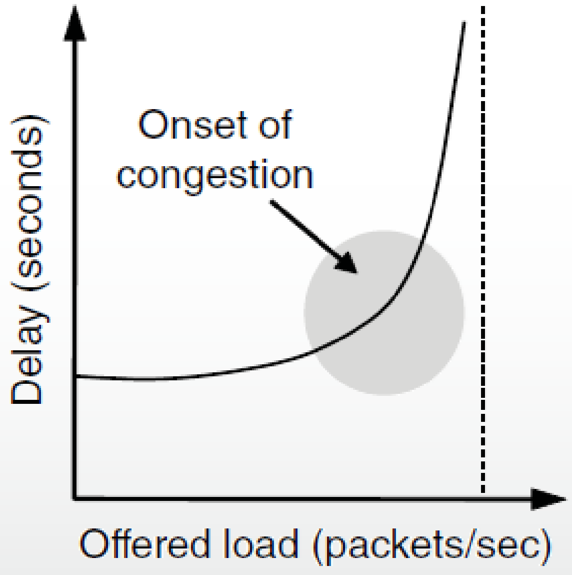
UNIVERSITY OF Southampton
School of Electronics and Computer Science

Desirable Bandwidth Allocation

- Efficient use of bandwidth gives high goodput, low delay



Goodput rises more slowly than load when congestion sets in

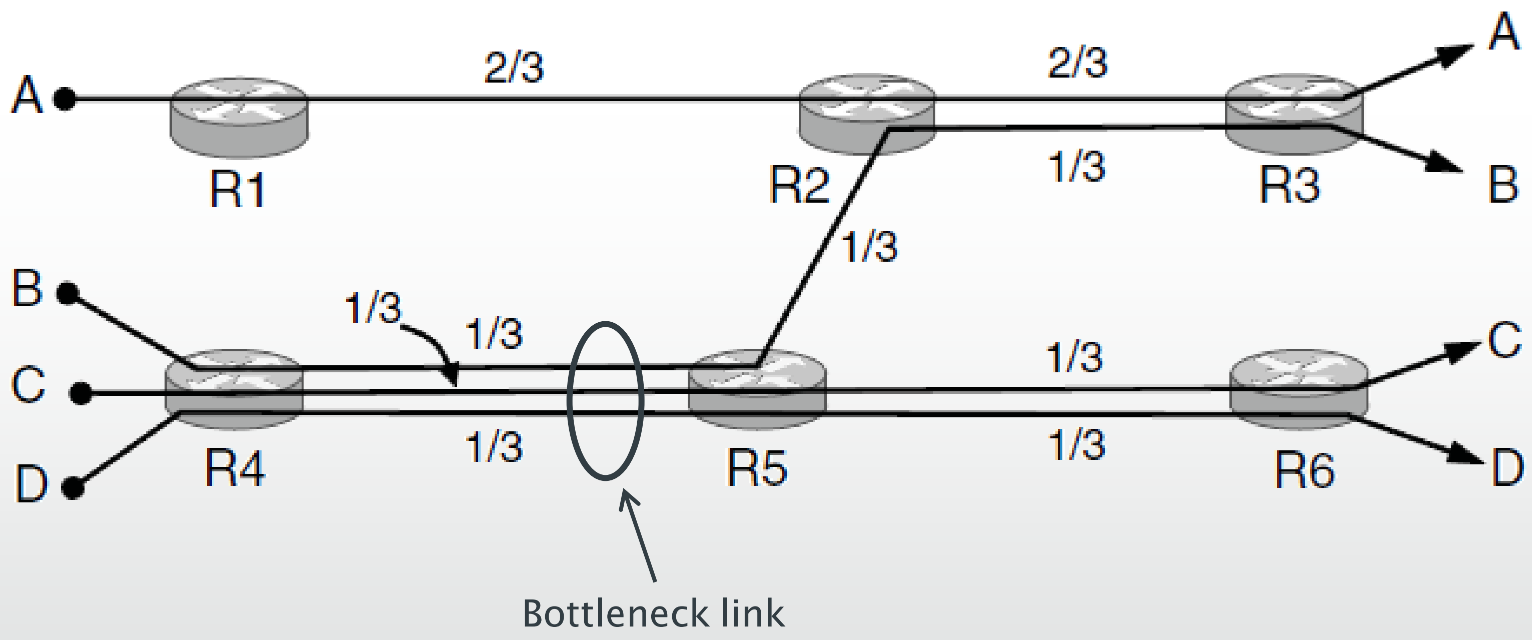


Delay begins to rise sharply when congestion sets in

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Desirable Bandwidth Allocation

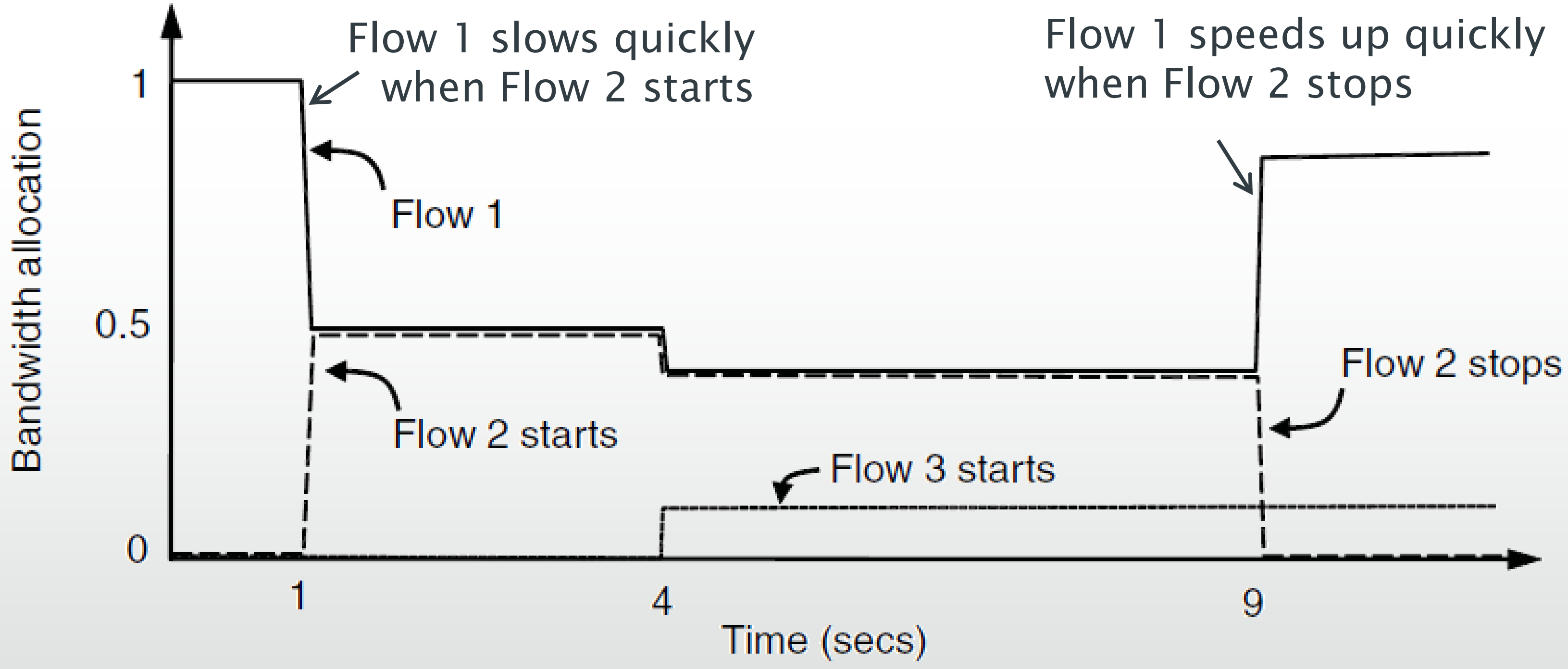
- Fair use gives bandwidth to all flows (no starvation)
  - Max-min fairness gives equal shares of bottleneck



UNIVERSITY OF Southampton
School of Electronics and Computer Science

Desirable Bandwidth Allocation

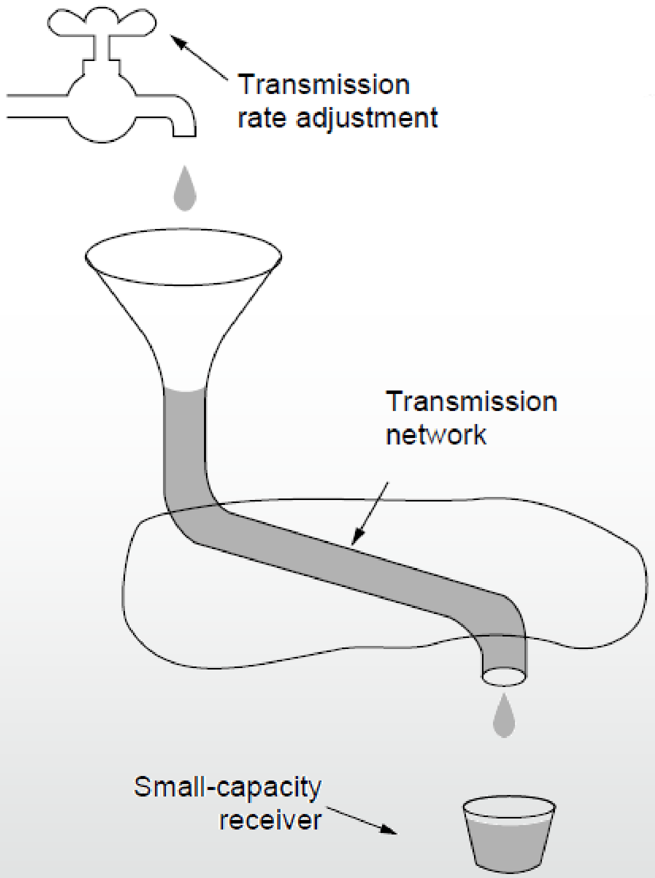
- We want bandwidth levels to converge quickly when traffic patterns change



Regulating the Sending Rate

Sender may need to slow down for different reasons:

- **Flow control**, when the receiver is not fast enough



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

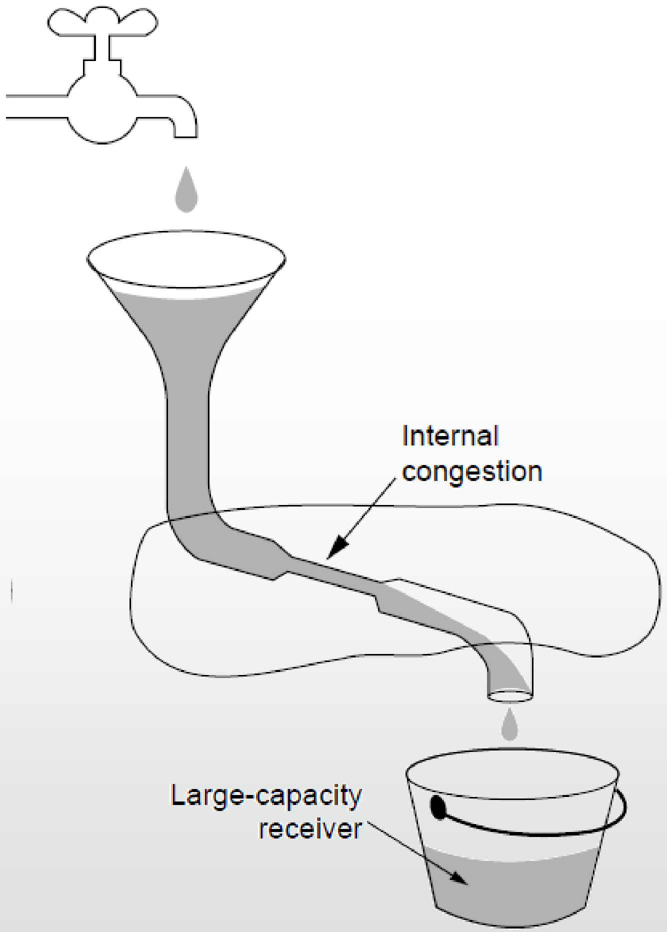
UNIVERSITY OF Southampton
School of Electronics and Computer Science

Regulating the Sending Rate

Sender may need to slow down for different reasons:

- **Flow control**, when the receiver is not fast enough
- **Congestion**, when the network is not fast enough

Focus here is on dealing with congestion.



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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Regulating the Sending Rate

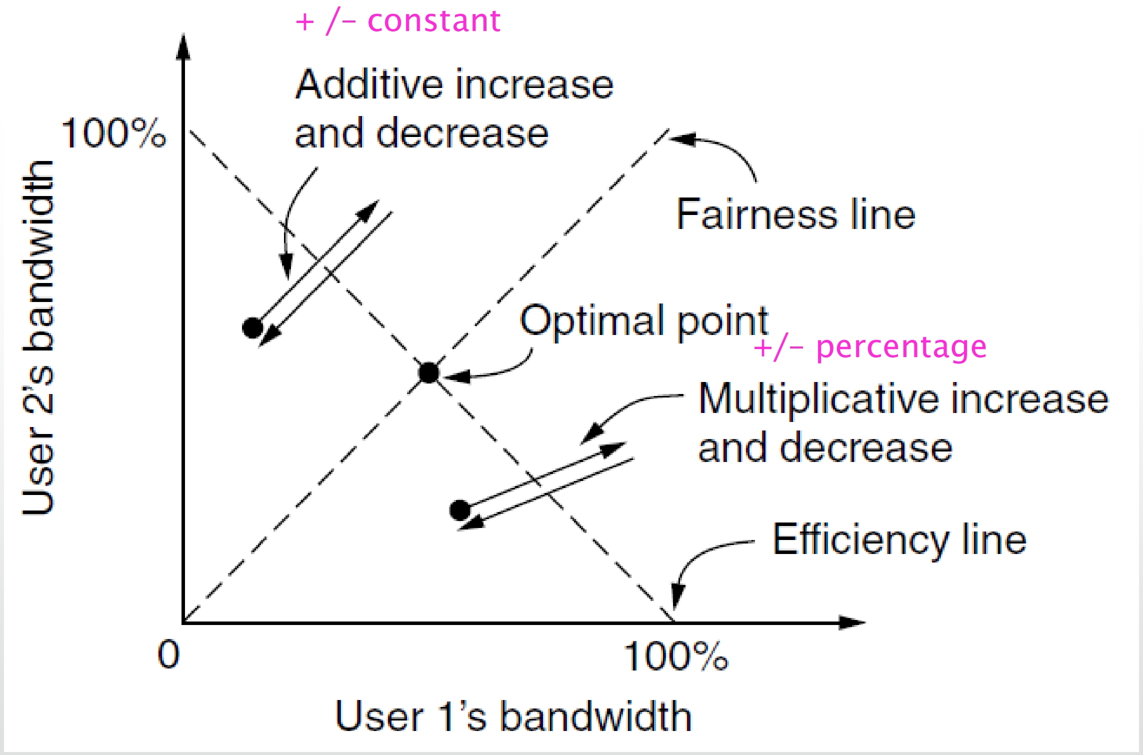
- Different congestion signals the network may use to tell the transport endpoint to slow down (or speed up)

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

UNIVERSITY OF Southampton
School of Electronics and Computer Science

Regulating the Sending Rate

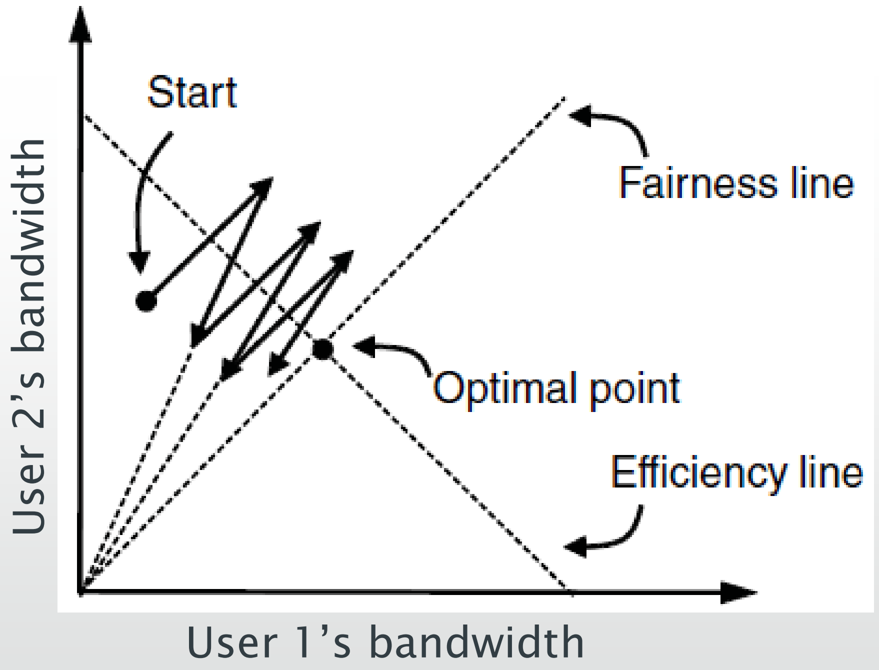
- If two flows increase/decrease their bandwidth in the same way when the network signals free/busy they will not converge to a fair allocation



UNIVERSITY OF Southampton
School of Electronics and Computer Science

Regulating the Sending Rate

- The AIMD (Additive Increase Multiplicative Decrease) control law does converge to a fair and efficient point!
  - TCP uses AIMD for this reason



Summary

- Addressing
- Connection establishment
- Connection release
- Error control and flow control
- Crash Recovery
- Regulating the Sending Rate