Intuitively, we want to give users with "small" demands what they want, and evenly distribute the rest

Max-min fair allocation is such that

the lowest demand is maximized

after the lowest demand has been satisfied,
the second lowest demand is maximized

after the second lowest demand has been satisfied,
the third lowest demand is maximized

and so on...

Max-min fair allocation can easily be computed

step 1	Start with all flows at rate 0
step 2	Increase the flows until there is a new bottleneck in the network
step 3	Hold the fixed rate of the flows that are bottlenecked
step 4	Go to step 2 for the remaining flows

Done!

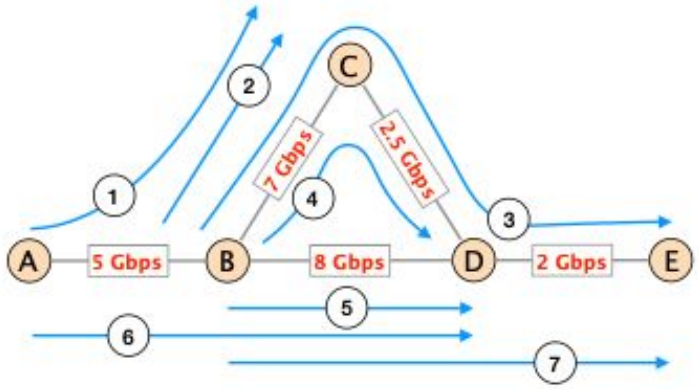
Example

Demands: {2, 2.6, 4, 5}

Capacity: 10

Example

Consider the network on the right consisting of 5 nodes (A to E). Each link has a maximal bandwidth indicated in red. 7 flows (1 to 7) are using the network at the same time. You can assume that they have to send a lot of traffic and will use whatever bandwidth they will get. Apply the max-min fair allocation algorithm to find a fair bandwidth allocation for each flow.



A network with shared links and 7 flows.

For each flow, what is the bottleneck link?

Example

Consider the network on the right consisting of 5 nodes (A to E). Each link has a maximal bandwidth indicated in red. 7 flows (1 to 7) are using the network at the same time. You can assume that they have to send a lot of traffic and will use whatever bandwidth they will get. Apply the max-min fair allocation algorithm to find a fair bandwidth allocation for each flow.

For each flow, what is the bottleneck link?

Bottleneck link	D-E	C-D	B-C	A-B	B-D
Flow 1 A - B - C	1	1.5	**2.25**		
Flow 2 B - C	1	1.5	**2.25**		
Flow 3 B - C - D - E	**1**				
Flow 4 B - C - D	1	**1.5**			
Flow 5 B - D	1	1.5	2.25	2.75	**4.25**
Flow 6 A - B - D	1	1.5	2.25	**2.75**	
Flow 7 B - D - E	**1**				

Max-min fair allocation can be approximated by slowly increasing W until a loss is detected

Intuition

Progressively increase
the sending window size

max=receiving window

Whenever a loss is detected,
decrease the window size

signal of congestion

Repeat

Design a *correct*, *timely*, *efficient* and *fair* transport mechanism

knowing that

packets can get lost

corrupted

reordered

delayed

duplicated

Dealing with corruption is easy:
Rely on a checksum, treat corrupted packets as lost

The effect of reordering depends on the type of ACKing mechanism used

individual ACKs	no problem
full feedback	no problem
cumm. ACKs	create duplicate ACKs

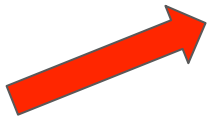
Long delays can create useless timeouts, for all designs

Packet duplicates can lead to duplicate ACKs whose effects will depend on the ACKing mechanism used

individual ACKs	no problem
full feedback	no problem
cumm. ACKs	problematic

Here is one correct, timely, efficient and fair transport mechanism

ACKing	full information ACK
retransmission	after timeout
	after *k* subsequent ACKs
window management	additive increase upon successful delivery
	multiple decrease when timeouts



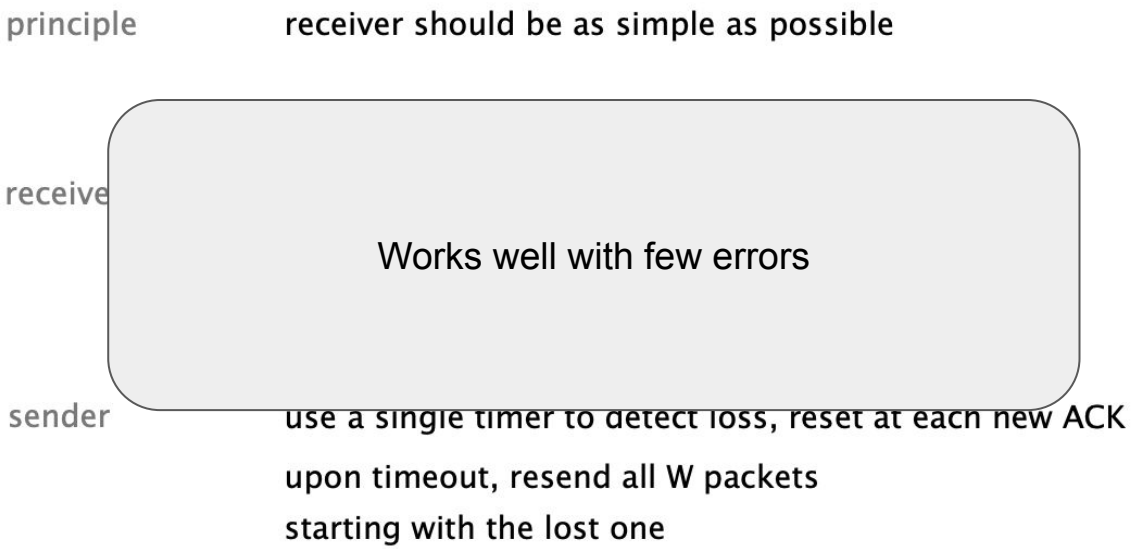
We'll come back to this when we see TCP

Reliable Transport Examples

Go-Back-N (GBN) is a simple sliding window protocol using cumulative ACKs

principle	receiver should be as simple as possible
receiver	delivers packets in-order to the upper layer for each received segment, ACK the last in-order packet delivered (cumulative)
sender	use a single timer to detect loss, reset at each new ACK upon timeout, resend all W packets starting with the lost one

Go-Back-N (GBN) is a simple sliding window protocol using cumulative ACKs



Selective Repeat (SR) avoids unnecessary retransmissions by using per-packet ACKs

principle	avoids unnecessary retransmissions
receiver	acknowledge each packet, in-order or not buffer out-of-order packets
sender	use per-packet timer to detect loss upon loss, only resend the lost packet

Selective Repeat (SR) avoids unnecessary retransmissions by using per-packet ACKs

principle	avoids unnecessary retransmissions
receiver	Only retransmit the packets that were lost, receiver is more complex
sender	use per-packet timer to detect loss upon loss, only resend the lost packet

Illustration

https://www2.tkn.tu-berlin.de/teaching/rn/animations/gbn_sr/

GBN Question

Assume you have a Go-Back-N (GBN) sender and receiver. The receiver acknowledges each data segment with a cumulative ACK which indicates the next expected data segment. Furthermore, it saves out-of-order segments in a buffer. The sender and receiver buffer can contain four segments each. The time-out period is much larger than the time required for the sender to transmit four segments in a row.

- The sender wants to transmit 10 data segments (0,. . . ,9) to the receiver. Assume that exactly one segment is lost. How many segments has the sender to transmit in the best and worst case? For each case, indicate which segment was lost