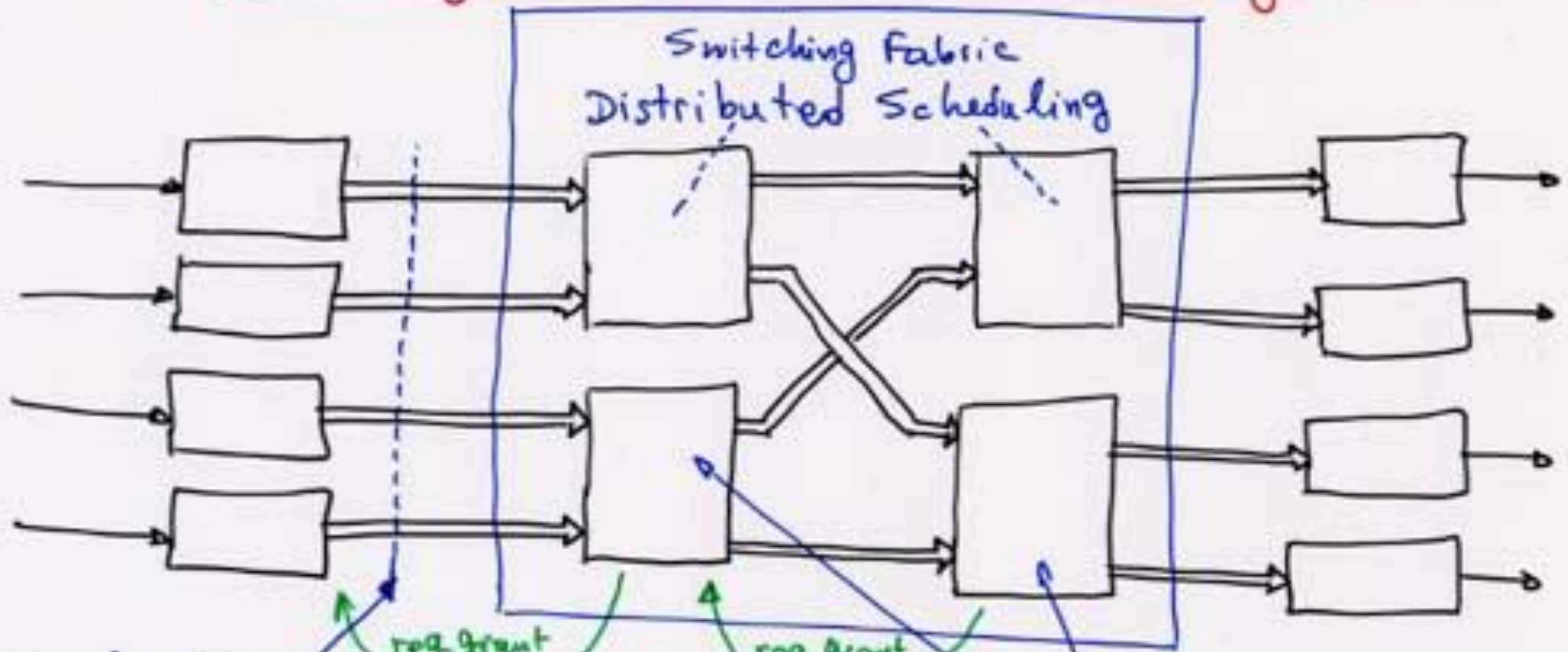
6.1 Credit-Based Flow Control (Backpressure)

- Summary to be added here

Central Scheduler is Impractical for large N

Solution 2: Switching Fabrics with Internal Buffering & Backpressure



the traffic here may have packets that are short-term-conflicting in the switching fabric, but are long-term-non-conflicting in the fabric

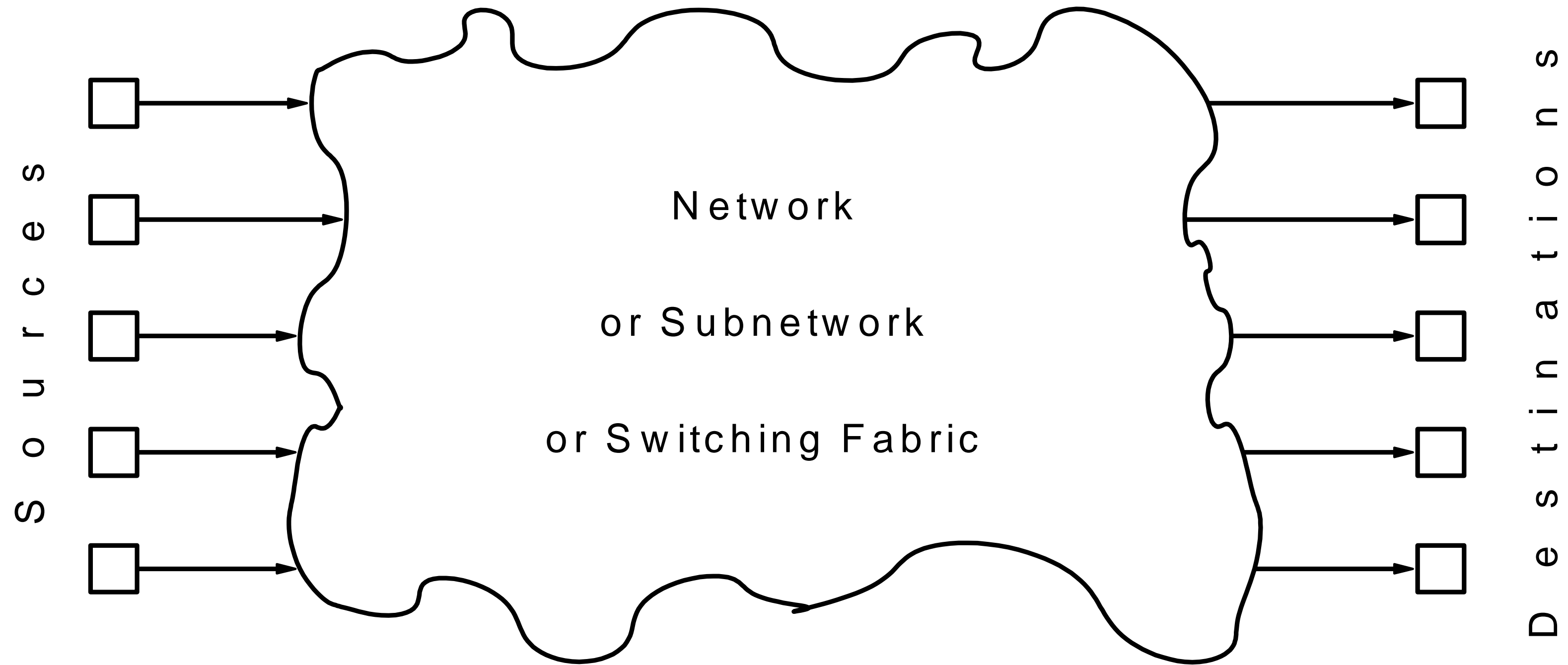
small internal buffers

handled by these

owing to backpressure and distributed scheduling

FLOW CONTROL

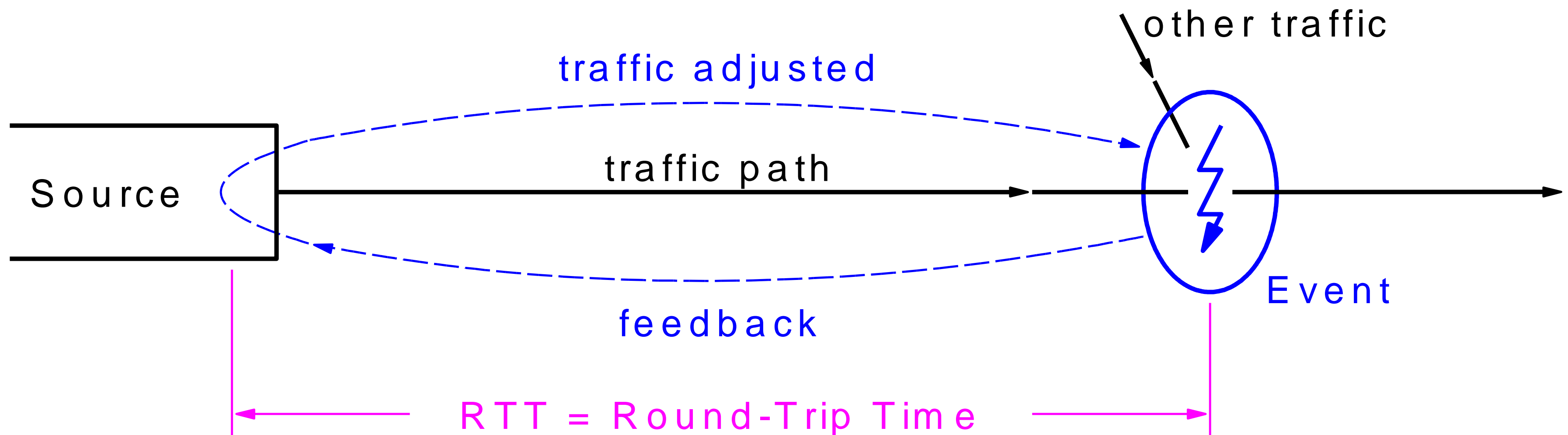
a feedback control problem



- How do the sources know at which rate to transmit?
- How do the sources know when their (collective!) demands exceed the network or the destination capacity?

⇨ Answer: FEEDBACK from the contention point(s), in the network or at the destinations, to the sources

RTT: the Fundamental Time-Constant of Feedback



The traffic is ``blind'' during a time interval of RTT:

- the source will only learn about the effects of a transmission RTT after this transmission has started (or RTT after a request for such transmission has been issued)
- the (corrective) effects of a contention event will only appear at the contention point RTT after the event occurrence

"Blind Mode" bits in faster networks

- For faster networks
  - start transmitting earlier
  - start transmitting at a higher rate

For networks to get faster, an increasing number of bits must be sent in "blind mode"

initial ("blind mode") rate of transmission	**Amount of data transmitted in "blind mode"**			
	Distance = 8 m	80 m	8 km	8,000 km
	RTT = 2 d = 80 ns	*800 ns*	*0.08 ms*	*80 ms*
1 Mb/s	1/100 Bytes	1/10 B	10 B	10KB
1 Gb/s	10 Bytes	100 B	10 KB	10 MB
10 Gb/s	100 Bytes	1 KB	100 KB	100 MB
100 Gb/s	1 KB	10 KB	1 MB	1 GB

``Blind Mode" bits in faster Networks

For faster networks:
- start transmitting earlier
- start transmitting at a higher rate

For networks to get faster,

an increasing number of bits must be sent in ``blind mode"

initial (``blind mode") rate of transmission	amount of data xmitted in ``blind mode"	
	distance = 8 km RTT ~= 0.08 ms	distance = 8,000 km RTT ~= 80 ms
1 Mbit/s	10 Bytes	10 KBytes
1 Gbit/s	10 KBytes	10 Mbytes

Lossy versus Lossless Flow Control

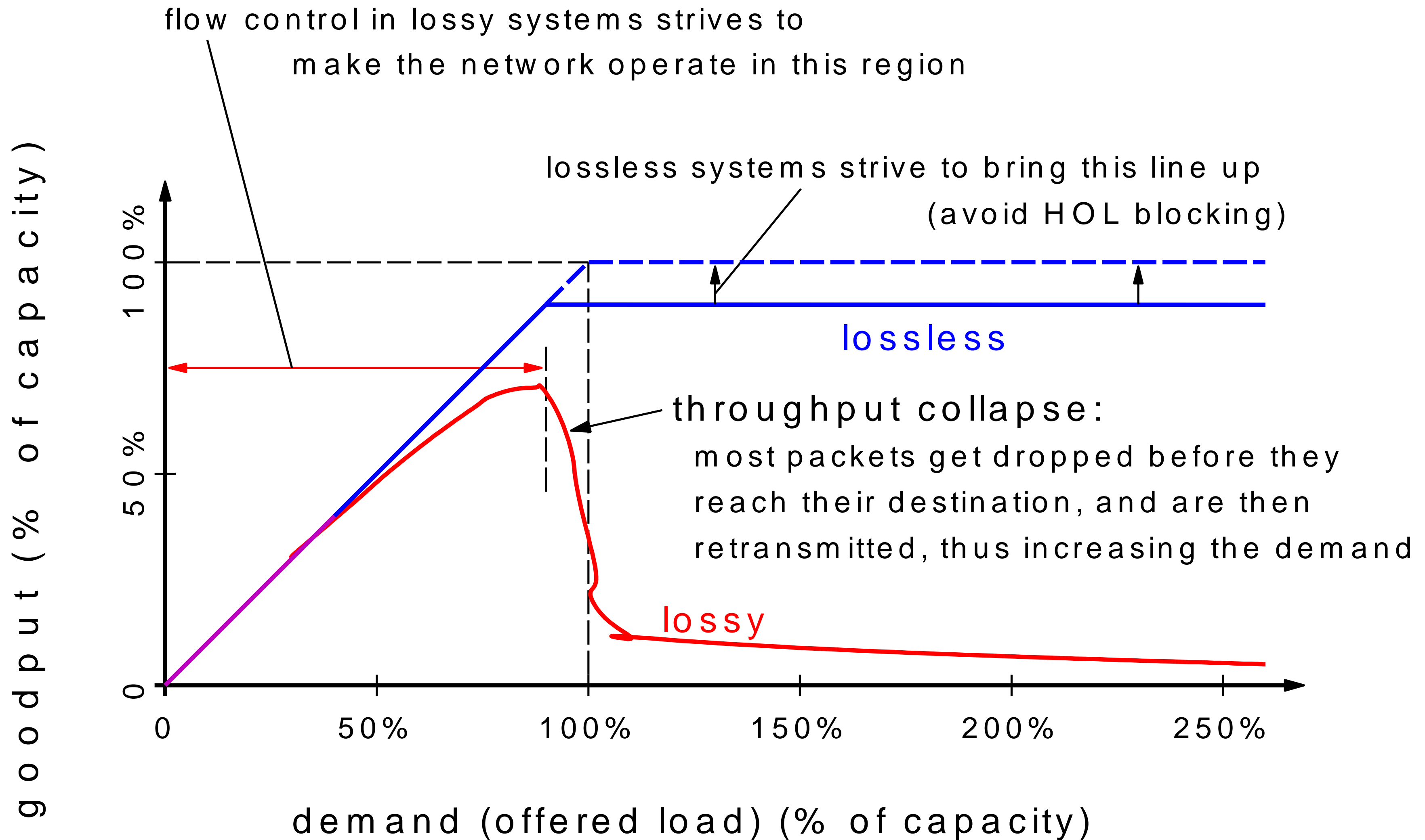
Lossy: *flow control may fail to prevent buffer overflows: packets can be dropped*

- inherited from ``communications engineers'': same as electrical noise
- simple switches
- for data: need retransmissions => long delays, complex if in H/W
- wastes communication capacity: ``goodput'' versus throughput
- need carefully designed protocols to sustain satisfactory goodput

Lossless: *flow control guarantees that buffers will never overflow*

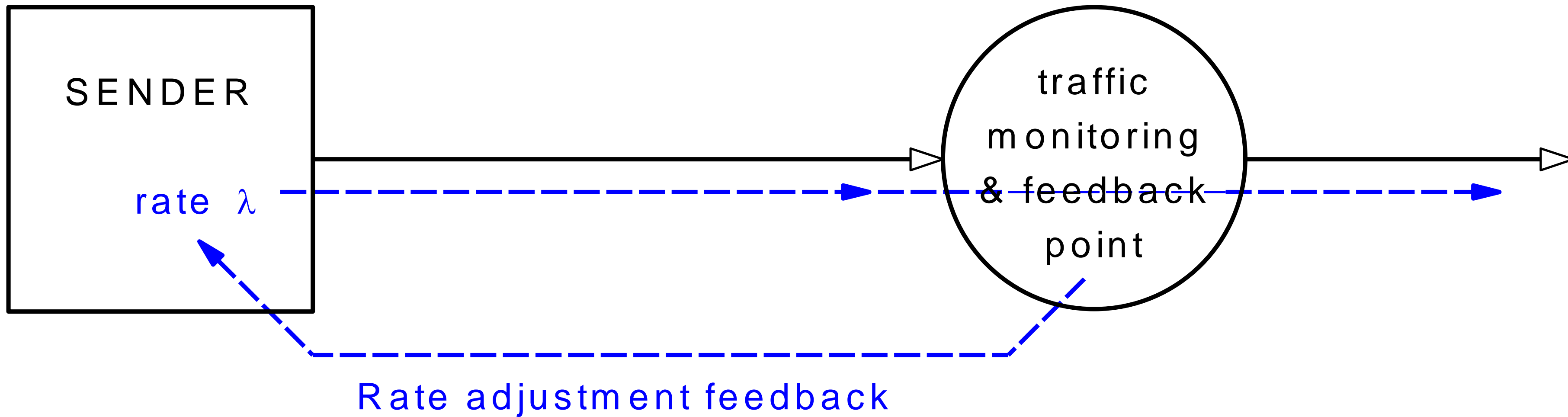
- inherited from ``hardware engineers'': processors never drop data
- no wasted communication capacity, minimizes delay
- need multilane protocols to avoid HOL blocking & deadlocks
- switches are more complex, need H/W support for high speed

Goodput versus Demand in Lossy and Lossless Flow Control



(this slide intentionally left blank)

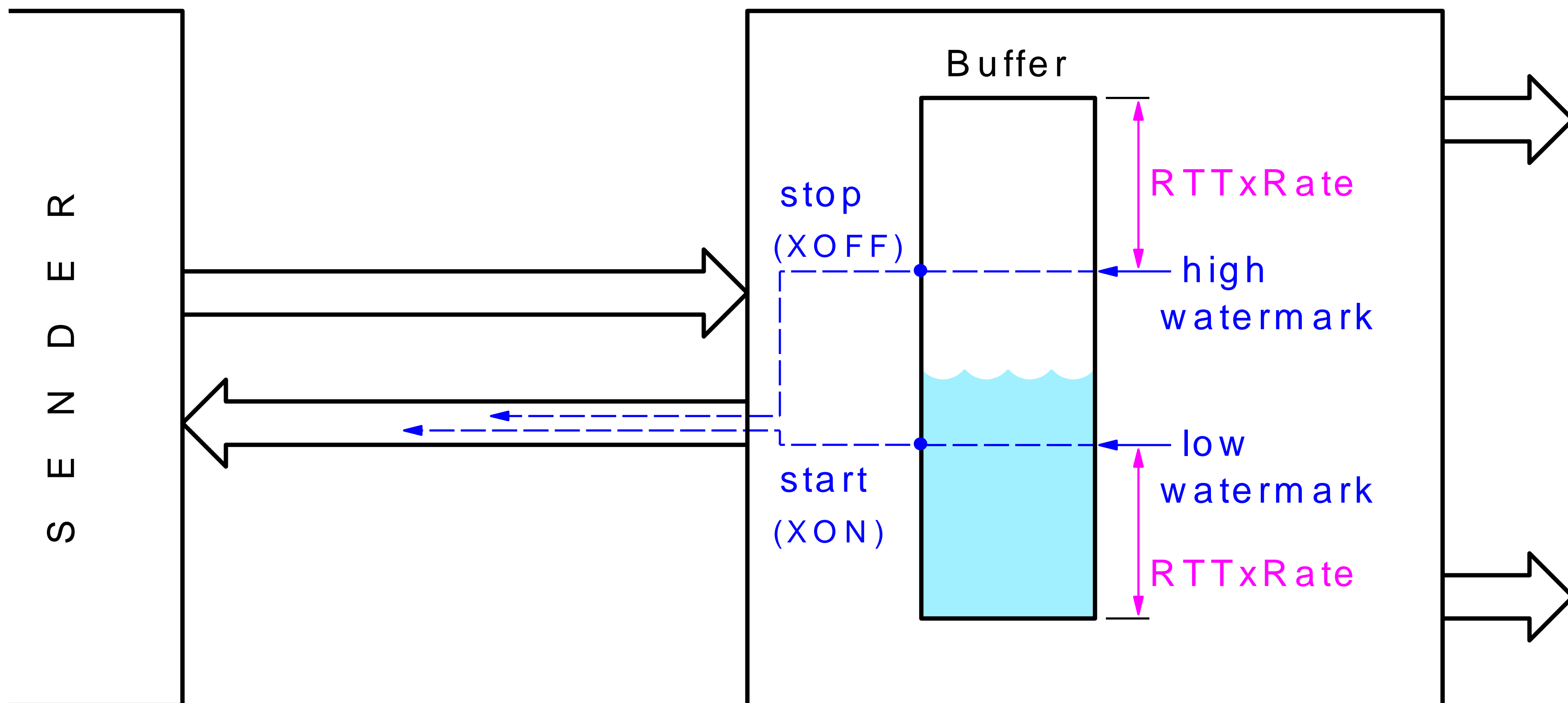
Rate-Based Flow Control



- differential (speed-up / slow-down), or
- absolute (new rate := value)

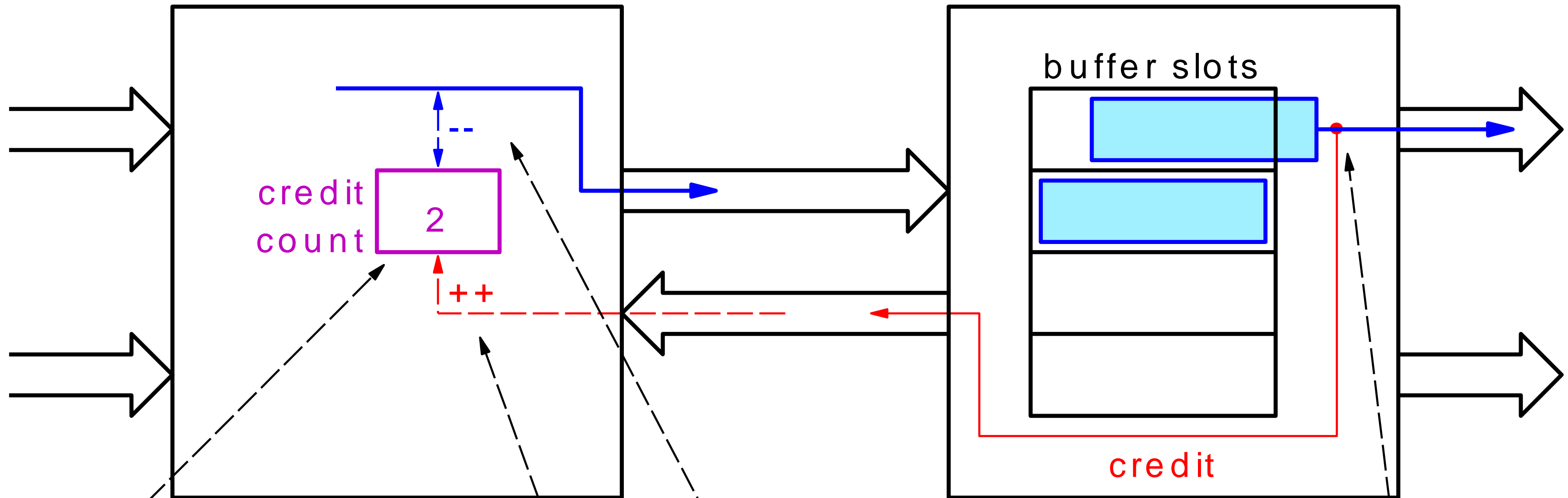
Note: oftentimes, the sender uses a variable-size window mechanism in order to control its rate

ON/OFF (start/stop) (XON/XOFF): simplistic Rate-based FC



- ``start'' ≡ (rate := peak); ``stop'' ≡ (rate := 0)
- rate-based flow control used for lossless transfers
- less than half the buffer efficiency of credit-based flow control

Credit-based (window) (backpressure) Flow Control

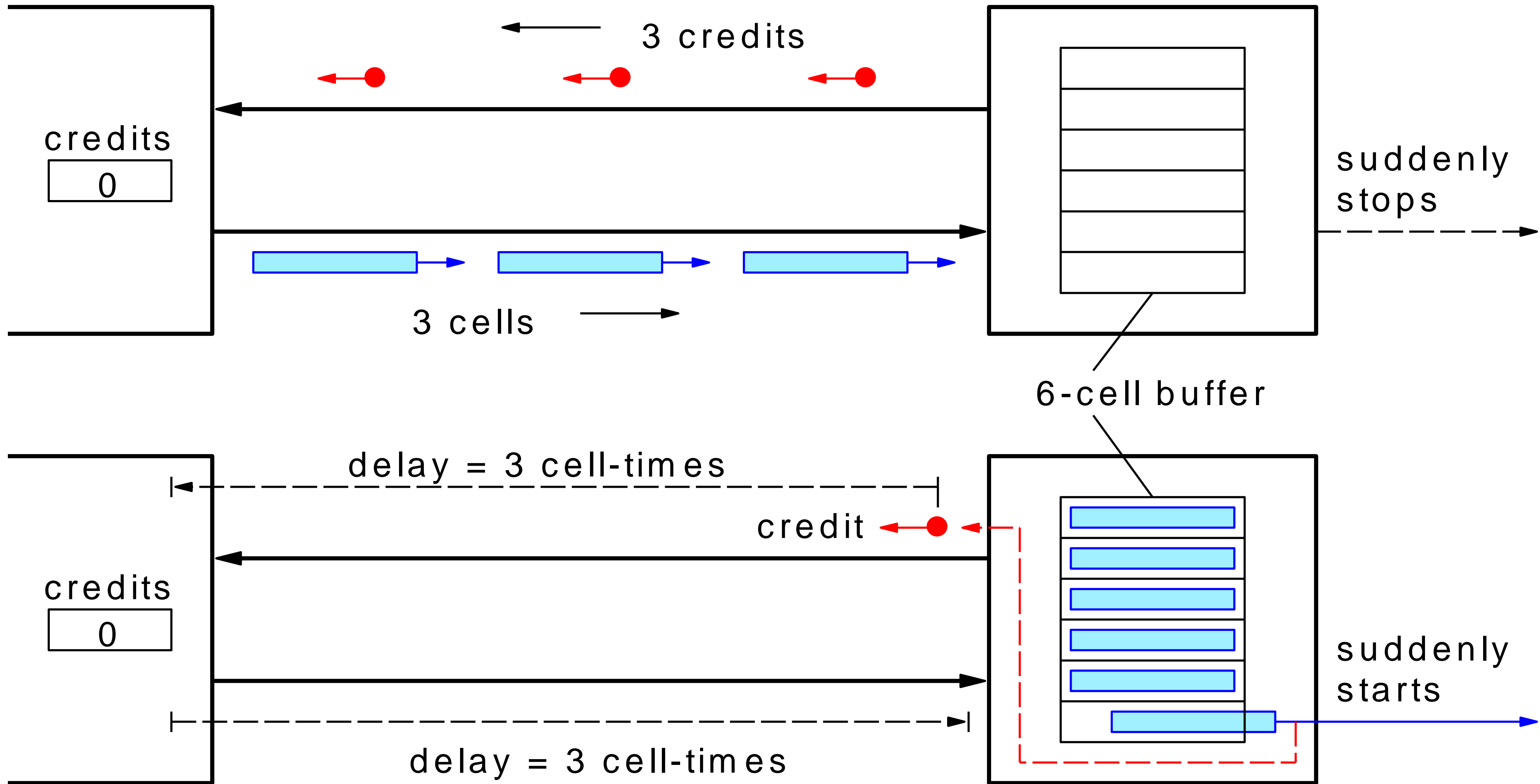


- count of buffer slots known to be available at the downstream site (not allowed to go negative)
- traffic can only depart if and when it acquires (decrements) the credit(s) that correspond to the buffer slot(s) needed
- arriving credits increment the credit count
- when new buffer slots are made available, corresponding credits are sent upstream

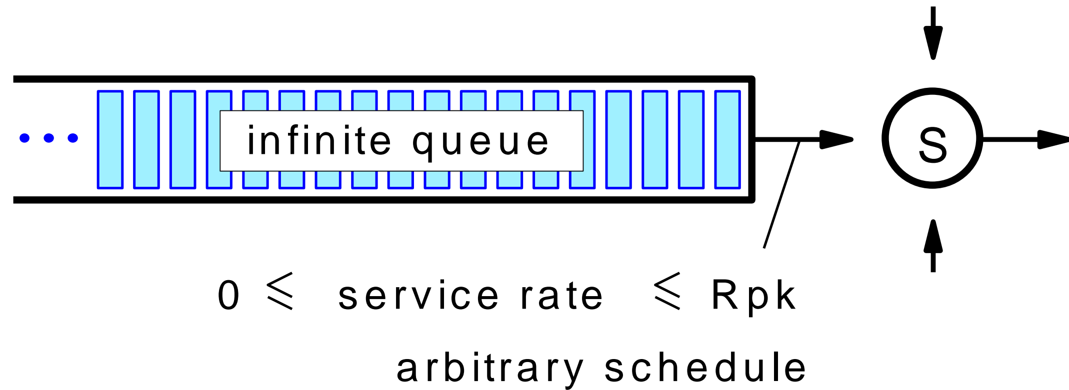
⇒ ***Lossless Flow Control***

Buffer Space = Peak Throughput x Round-Trip Time

necessary & sufficient



Theorem: Infinite Queue Push-Back



is equivalent to:

Upstream Departures

Downstream Arrivals

size = RTT*Rpk

flow control algorithm

infinite queue

FC

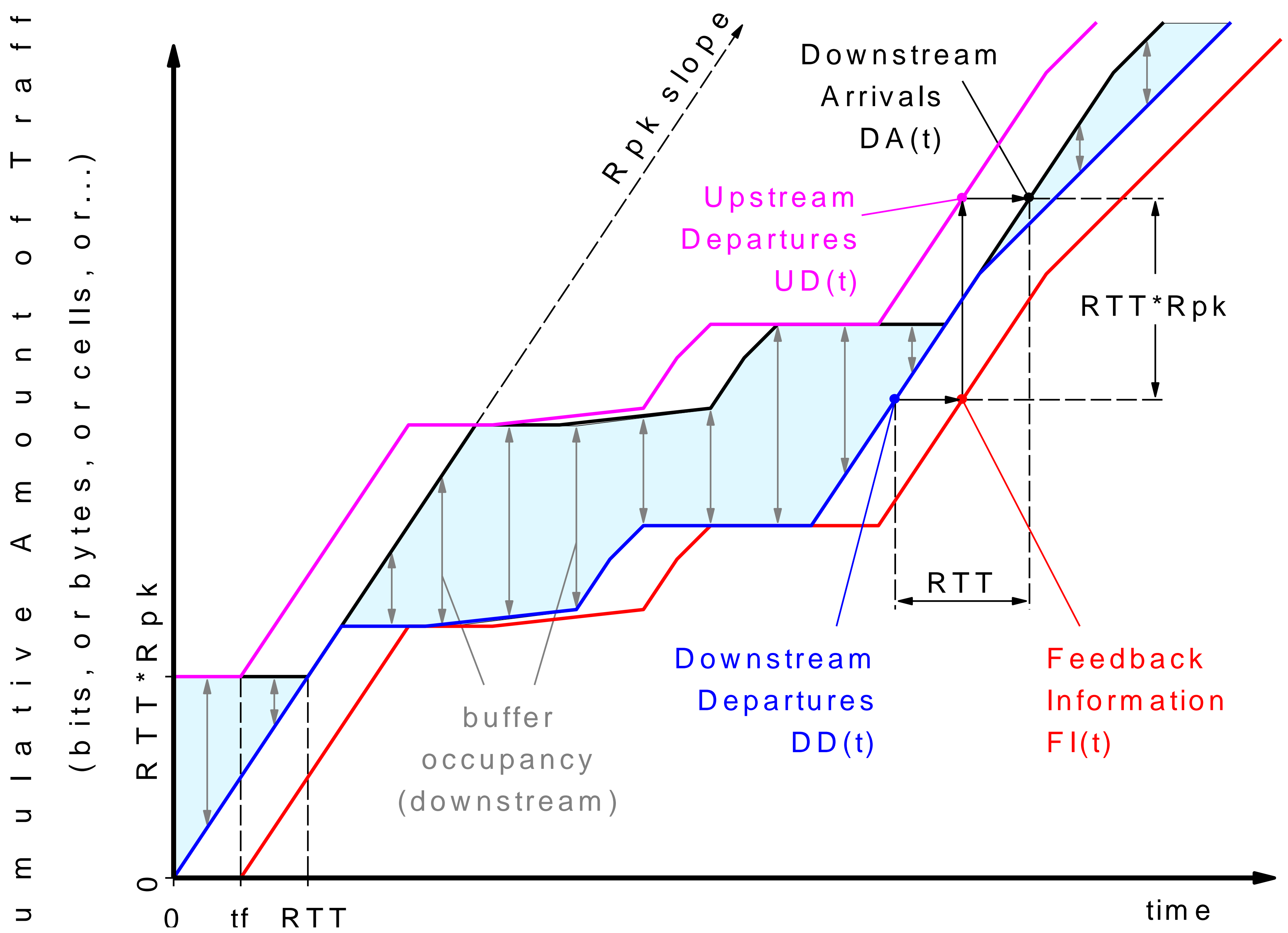
capacity ⩾ Rpk

S

Feedback Information

Downstream Departures

RTT


Cumulative Amount of Traffic
(bits, or bytes, or cells, or...)
RTT*Rpk
Rpk slope
Downstream
Arrivals
DA(t)
Upstream
Departures
UD(t)
RTT*Rpk
RTT
Downstream
Departures
DD(t)
Feedback
Information
FI(t)
buffer
occupancy
(downstream)
0
0
tf
RTT
time

- Downstream Departures DD(t) (cumulative):

arbitrary function of time, provided that its slope satisfies:

$$0 \leqslant \text{service rate of S} \leqslant \text{Rpk} \quad (1)$$

$$\Rightarrow \quad 0 \leqslant DD(t+d) - DD(t) \leqslant d*\text{Rpk} \quad (2)$$

- Upstream Departures UD(t) (cumulative):

UD(t) = DD(t-tf) + RTT*Rpk; this is always feasible, since:

$$\text{link capacity} \geqslant \text{Rpk} \geqslant \text{service rate of S}$$

- Downstream Arrivals: DA(t) = DD(t-RTT) + RTT*Rpk (3)

- Buffer Occupancy (downstream): BO(t) = DA(t) - DD(t)

$$(3) \Rightarrow BO(t) = RTT*Rpk - [\,DD(t) - DD(t-RTT)\,]$$

$$\text{with (2)} \quad \Rightarrow \quad \boxed{0 \leqslant BO(t) \leqslant RTT*Rpk}$$

feasibility of arbitrary departure schedule (provided (1) holds) ↑ (first inequality)

downstream buffer never overflows ↑ (second inequality)

Feedback Format Options:

how to make the function DD(t) known to the upstream neighbor

1. QFC Credit-Based Flow Control
2. Classical (incremental) Credit-Based Flow Control
3. Rate-Based Flow Control

1 Quantum Flow Control (QFC)

http://www.qfc.org

Every time DD(t) changes by more than a given threshold N relative to the last time a feedback message was sent, transmit the current value of:

$$DD(t) \text{ modulo } 2^{28} \quad (\text{or modulo } 2^{8} \text{ for short links})$$

- credit-based flow control: lossless
- robust: even if a feedback message is corrupted (lost), the next one will restore the error

② Classical (incremental) Credit-Based Flow Control

every N downstream departures (DD(t1) = DD(t0) + N),

transmit a credit back (N is an implicit parameter);

the upstream node maintains a credit count CC equal to:

$$CC = RTT*Rpk + DD(t-tf) - UD(t);$$

this is incremented by N on every credit arrival, and

decremented by 1 on every departure of a unit of traffic

- shorter feedback (credit) messages than QFC
- non-robust: loss of a credit leads to buffer and
  transmission capacity underutilization;
  accumulated losses of credits lead to deadlock!

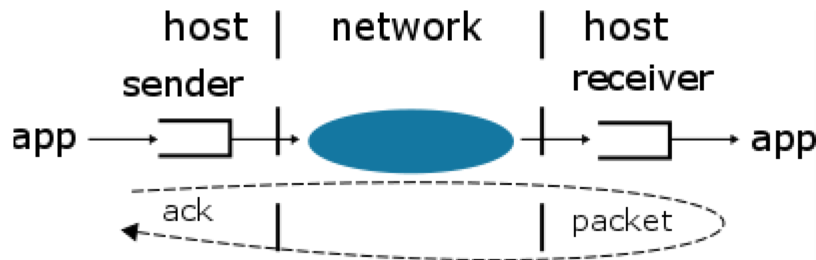
③ Rate-Based Flow Control

On every change of the slope of DD(t) (rate of downstream departures), send back the new value of the rate (slope); upon reception of such feedback, the upstream node adjusts its rate of transmission (slope of UD(t)) to the value received; thus, UD(t) is (almost!) a delayed and shifted-up copy of DD(t).

- Rate-based flow control
- Could be made lossless, but this would not be robust:

  slight mismatches between real & measured rate accumulate to large differences between the values of DD(t-tf), UD(t); similarly, variations in tf (delay of feedback messages) lead to UD(t) value errors.

Automatic repeat request & e2e flow control



Different roles served

- Network is unreliable
  - Automatic repeat request (ARQ): recover damaged/dropped packets
- Network may reorder packets
  - app waits packets in-order : re-sequence pkts at rcv buf
- Receive buffer should not overflow

  → e2e flow control

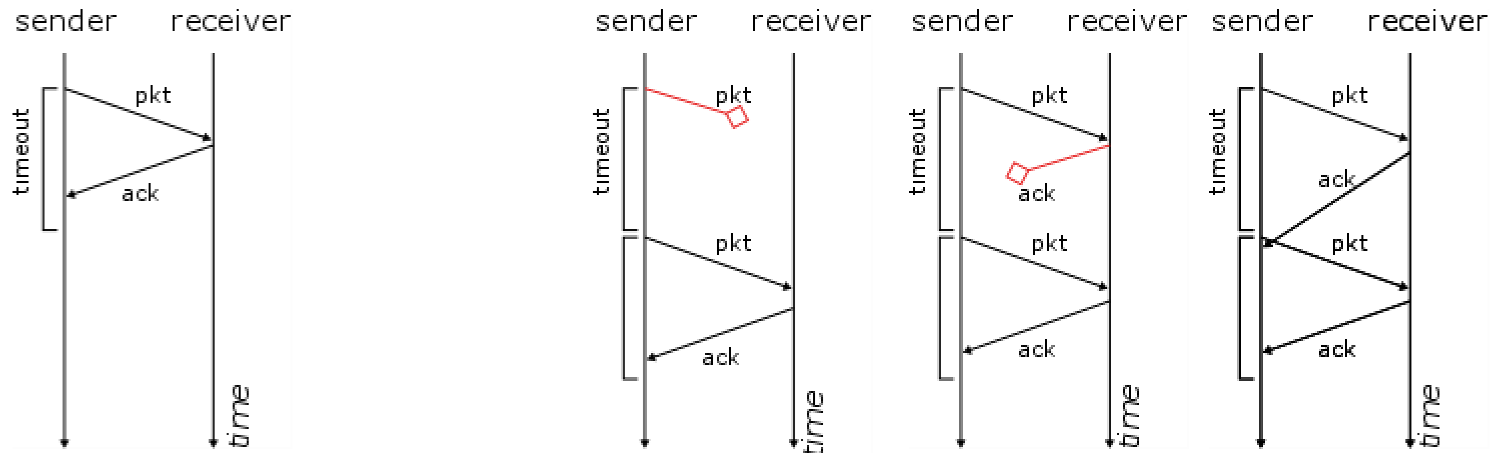
End-to-end flow control

- App may delay draining receiver buffer; also out-of-order (OOO) need wait
- Netw. may or may not have link-level flow
  - but cannot prevent rcv buffer overflow with link-level flow control: what if we stop the next-awaited packet?
- → sender flow controls rcv buf: every injected pkt fits in rcv buf

TCP uses these mechanisms

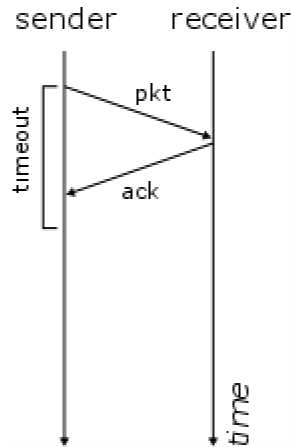
- TCP connections / sockets are bidirectional
  - Acks may be piggybacked on reverse payload packets

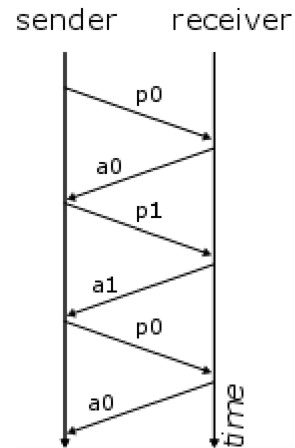
Stop-and-Wait flow control

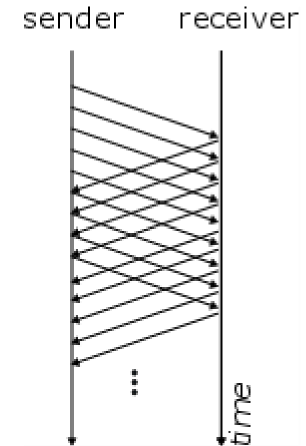


- Sender makes sure at most one packet is pending at a time
  - To all destinations or to each destination?
- Transmit packet, set timer (timeout)
- If ack does not arrive before time → retransmit packet
- Packets are retransmitted when:
  - Packet is lost / damaged
  - Ack is lost / damaged → duplicates at receiver
  - Retransmission timer expires prematurely → duplicates at receiver
    - set timer too soon → false retransmissions;
    - set it too late → retransmissions delay a lot (problem with TCP is scale-out datacenter applications)

Stop-and-Wait flow control

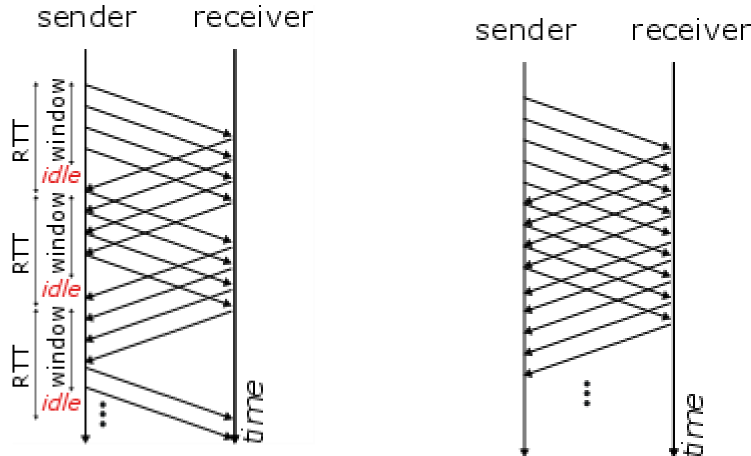






- Sequence numbers: to discover duplicate packets at destination
  - 1-bit sequence numbers suffice : packets & acks numbered either as 0 or 1
- Buffer sizes:
  - Destination buffer for one packet:
  - Sender : retransmission / replay buffer stores one packet waiting ack
- Performance: Stop-and-wait underutilizes the network when
  - Network-delay-product is larger than the packet size
  - *"The bandwidth × delay product (BDP, or C x rtt) represents the amount of data that could be in transit. We would like to be able to send this much data without waiting for the first acknowledgment. The principle at work here is often referred to as keeping the pipe full." [Computer networks: a system's approach, L. Peterson, B. Davie]*

Sliding window algorithms

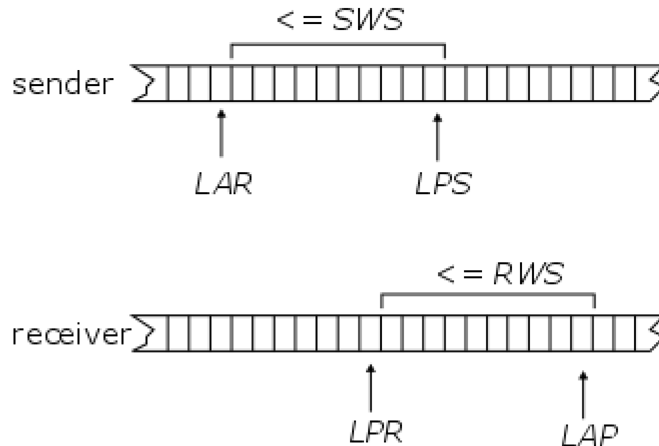


Allow multiple pending/outstanding/unacknowledged packets

Pipe full if window ≥ RTT x capacity (BDP)

- Sender: sequence number on each packet & 3 variables
  - send window size (SWS): upper bound of unacknowledged packets
  - LAR sequence number of the last acknowledgment received
  - LPS denotes the sequence number of the last packet sent
  - Sliding window invariant @ sender
    LPS − LAR ≤ SWS
- Destination variables
  - receive window size (RWS), upper bound of out-of-order pkts the destination is willing to accept
  - LPR sequence number of last packe received
  - LAP sequence number of largest acceptable packet
  - Sliding window invariant @ receiver
    LAP − LPR ≤ RWS

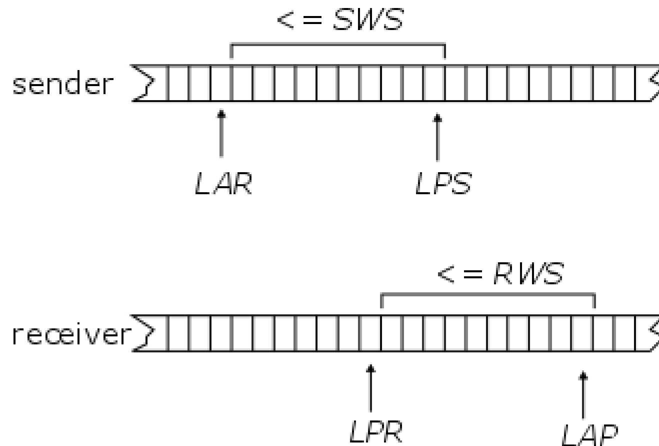
Sliding window algorithms



Allow multiple pending/outstanding/un acknowledged packets

- Sender: sequence number on every packet + three variables:
  - send window size (SWS): upper bound of unacknowledged packets
  - LAR sequence number of the last acknowledgment received
  - LPS denotes the sequence number of the last packet sent
  - Sliding window invariant @ sender LPS − LAR ≤ SWS
- Destination variables
  - receive window size (RWS), upper bound of out-of-order pkts the destination is willing to accept
  - LPR sequence number of last packe received
  - LAP sequence number of largest acceptable packet
  - Sliding window invariant @ receiver LAP − LPR ≤ RWS

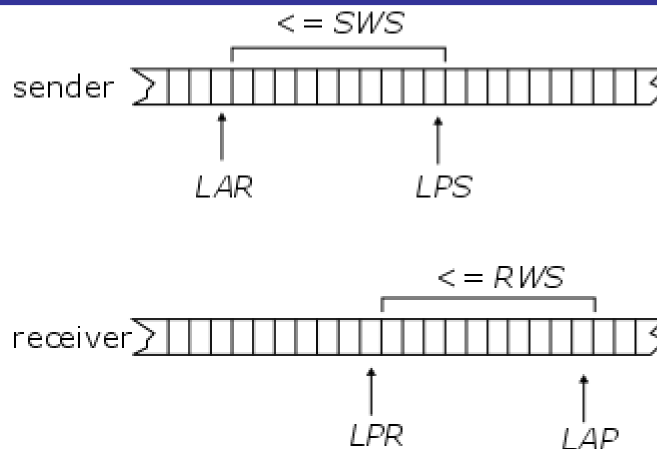
Sliding window algorithms



Dest variable: SeqToAck = largest sequence number such that all packets with sequence numbers less than or equal to SeqToAck have been received.

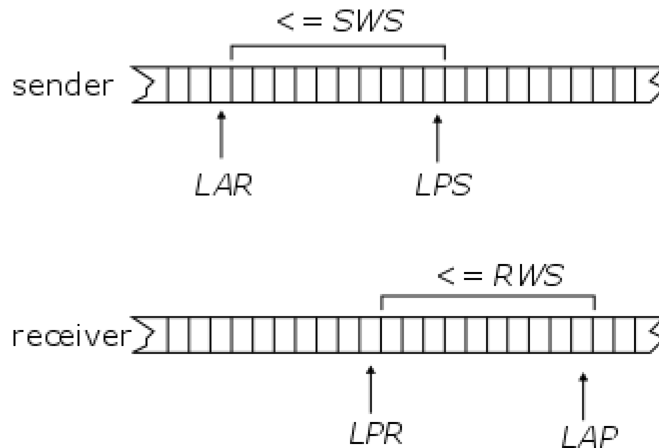
- Packet at destination : Accept if LPR < seq ≤ LAP; drop otherwise
  - The receiver acknowledges the receipt of SeqToAck, even if higher-numbered packets have been received. It then sets LPR = SeqToAck, LAP = LPR+RWS
- Go-back-N: a variation of sliding window w. RWS =1
  - the destination discards all packets except the awaited one → on timeout, need sender has to retransmit all packets in send window [LAR+1, LPS]
- If RWS = SWS → dest can buffer all packets the sender sends
- Selective acknowledgments : receiver acknowledges OOO pkts
  - helps sender to avoid retransmitting packets received OOO on timeout

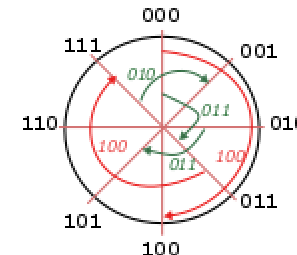
Finite sequence numbers



- Sequence numbers are encoded using finite number of bits
  - How many bits are needed? Answer depends on window size, e.g. stop-and-wait (sws = 1) needed 1 bit
- Suppose num-of-seq-numbers ≥ SWS + 1
  - With RWS = SWS this number of sequence numbers is not sufficient
    - suppose SWS = 7 and 8 seq. numbers [0,7]: *sender transmits [0,6] → receiver gets them, waits [7,5]; all acks lost; sender retransmits [0,6]; receiver cannot tell if packets 0-5 are new or old…*
  - *For Go-back-N (RWS= 1) its OK*
    - *the destination would refuse to accept the second packet with sequence 0 → it waited for seq 7*

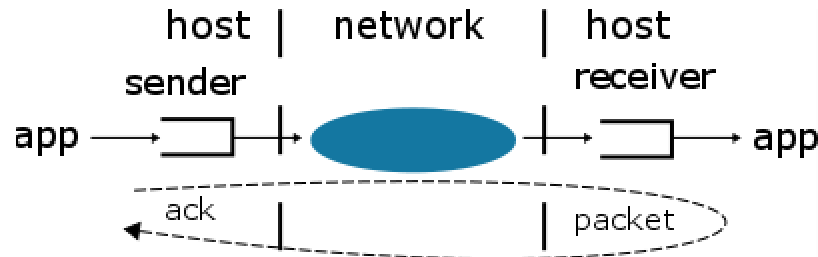
Finite sequence numbers & wrap-around





- In principle, when RWS = SWS we need
  - SWS < (max-sequence-numbers + 1) /2
    - *e.g for SWS = 8, max sequence numbers > 16 -1 = 15*
  - *For simplicity max-sequence-numbers* ≥ 2 * SWS
    - *having (slightly) more sequence number does not do any harm*
- *Suppose SWS = 4 & max-sequence-numbers = 8 encoded in three bits*
  - *We can use 3-bit unsigned integer arithmetic for comparisons*
  - *E.g. when receiving a pkt w. sequence seq at dest, we want to know if* LPR < seq ≤ LAP.
    - *Tmp = seq – LPR: if most-significant-bit of tmp = 0 & tmp != 0, then, seq > LPR*
    - *Tmp = LAP – seq; if most-significant-bit of tmp = 0, then LAP ≥ seq*
- *Same with cumulative credits: sender makes sure not to send > X packets ahead of DownstreamDepartures (DD), where X is the downstream buffer size: if X needs 4 bits, encode UD & DD using 5 bits*

Absolute credits combined with Go-back-N



- Sender puts sequence numbers to packets (last_sent_seq+1)
  - keeps # outstanding packets (OP), incremented with every packet sent
- Destination acks highest in-order packet it received
  - Acks also carry # packets that the dest can accept (last advertised buffer space, LABS)
- Upon ack, sender computes # acked packets (AP) = ack_seq - seq_last_ack
  - OP = OP – AP
- Sender can send when LABS – OP > 0; receiver drops all OOO packets
- Upon timeout, sender retransmits all packets from seq_last_ack to last_sent_seq.
- Note that ACK sent upon receiving packet – not when it is put out of buffer