Multiplexing

- sharing of resources among multiple users
- involves resource allocation & congestion control (prevention or response to overload conditions) mechanisms
- performance objectives:
  - efficient resource utilization
  - stability & fast transient response
  - minimum queuing delays, quality of service
  - fairness
- mechanisms can be described as:
  - router-centric vs host-centric
  - reservation based vs feedback based
  - window based vs rate based
- A best-effort service model uses feedback rather than reservations; tends to be host-centric with possibly some help from routers eg. Internet

  A QoS-based service model usually involves reservations, which need significant router involvement; often rate based since windows are only indirectly related to service requirements

- A <u>circuit-switched network</u> uses reservations, implemented with FDM/TDM, guarantees a constant rate, eg. telephone network

  A <u>packet-switched network</u> allows statistical multiplexing → more efficient resource sharing
  - overhead of packet identifiers
  - most packet switches use store-&-forward transmission (switch receives entire packet before beginning transmission of the packet on the outband link) → introduces store-&-forward delay ($\frac{\text{pkt length}}{\text{link rate}}$) at each hop

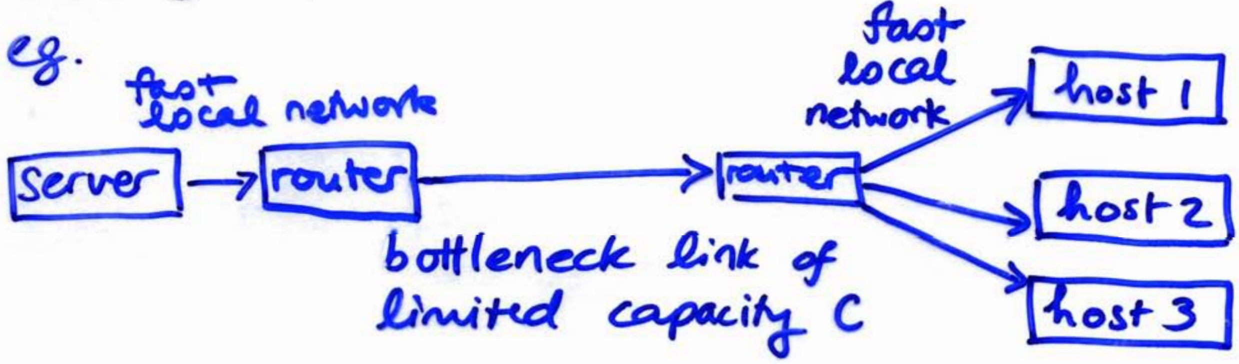
- <u>Elastic traffic</u> does not have intrinsic temporal behavior; delay & delay variability can be tolerated — can tolerate delay in recovering lost packets for reliable transfer

  <u>Stream traffic</u> has intrinsic temporal behavior which must be reproduced at the receiver; requires controlled delay (average & variation) but can tolerate some data loss (because of redundancy in speech/images)

  eg. real-time interactive speech or video telephony (1-way streaming media can be elastic)

Elastic traffic in a packet network

- predominant type of traffic in packet networks, includes file transfer, web browsing, email
- basic problem: transfer a file in its entirety from a source to a destination machine
- feedback control of congestion & bandwidth sharing
- eg.



- suppose host 1 is initially the only user — should get throughput C ideally
- If host 2 subsequently starts a download, if the first transfer is proceeding at rate C, there is congestion — need feedback to bring throughput of both to $\frac{C}{2}$ (prevent congestion & ensure fairness)

TCP congestion control

- introduced into the Internet in the late 1980s by Van Jacobson, about 8 years after the TCP/IP stack had become operational, to deal with congestion collapse (congestion → packet loss → retransmissions → more congestion → drastic ↓ in network performance)

- TCP source maintains a state variable, CongestionWindow, to limit unacknowledged data

$$\boxed{W_{max} = \min(\text{CongestionWindow}, \text{AdvertisedWindow})}$$

  - previously we had $W_{max} = \text{AdvertisedWindow}$ for flow control only
  - as before,

$$\boxed{\text{EffectiveWindow} = W_{max} - (\text{LastByteSent} - \text{LastByteAcked})}$$

- TCP source sets CongestionWindow based on its perception of network congestion level
  - CongestionWindow is defined in terms of bytes, but easier to understand if we think in terms of packets; it is always ≥ MSS (1 packet)
  - We describe below TCP Reno (most modern OSs)

- TCP interprets packet loss as a sign of congestion (assumption that congestion-related losses are more common than error-related losses — not always true in wireless networks)
  → sender ↓s CongestionWindow upon a "loss event"
    - a timeout, or
    - receipt of 3 duplicate ACKs → <u>fast retransmit</u>
- TCP interprets non-duplicate ACKs (ACKs of previously unacknowledged data) as a sign that there is sufficient available capacity & tries to get more
  → sender ↑s CongestionWindow upon receipt of non-duplicate ACKs
    - since ACKs are used to "clock" window size ↑, TCP is called <u>self-clocking</u>
- the amounts by which CongestionWindow is changed depend on whether TCP is in the <u>slow-start</u> or <u>congestion avoidance</u> phase

- Additive increase / Multiplicative decrease
  - when a triple duplicate ACK event occurs, the TCP sender halves CongestionWindow (multiplicative decrease)
  - when a non-duplicate ACK is received, the TCP sender in congestion avoidance phase increases the window by $\frac{1}{\text{CongWindow}}$ packets
    - for CongestionWindow = W, the window becomes $W + \frac{1}{W}$ packets after 1 ACK

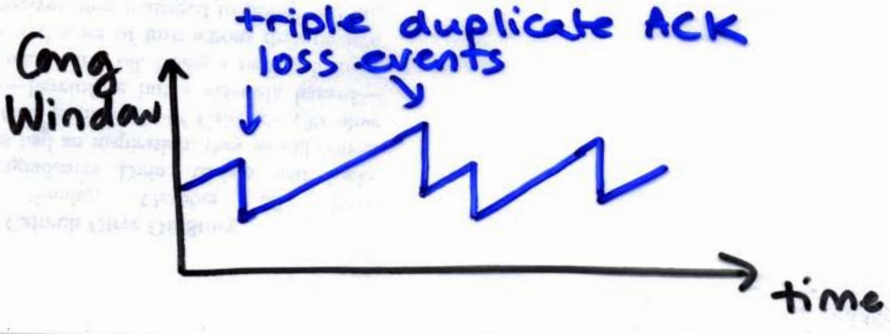
      $W + \frac{1}{W} + \frac{1}{W + \frac{1}{W}}$ packets after 2 ACKs

      etc
    - for large W, the window after W ACKs is $\approx W + W \times \frac{1}{W} = W + 1$

      i.e. CongestionWindow ↑s by ≈1 packet (MSS bytes) every RTT

  (additive increase)

→ saw-toothed pattern in long-lived TCP connections



- Slow Start

- at the beginning of a connection, the source does not know the available network capacity (even if it did, it needs to increase window gradually, as sending a burst of packets can cause buffer overflow at an intermediate router)
  → CongestionWindow is initially set to 1 packet (MSS bytes)

- to quickly ramp up the connection & discover the available capacity, slow-start increases CongestionWindow exponentially
  - CongestionWindow increases by 1 packet for each Ack received
    → effectively doubles every RTT

- If a triple duplicate ACK loss event occurs,
  - CongestionWindow is halved & the new value is stored as the variable CongestionThreshold
  - TCP sender enters congestion avoidance phase

- Slow start also occurs after a timeout loss event (indicates worse congestion than triple duplicate ACKs where at least some packets are getting through)

- CongestionWindow is set to 1 packet, ↑s exponentially with each non-duplicate ACK until a loss event occurs OR CongestionWindow reaches CongestionThreshold
  - When CongestionWindow reaches Conges^n Threshold TCP enters congestion avoidance phase
- Reaction to loss events is the same under both slow start & congestion avoidance
- An older version TCP Tahoe cuts CongestionWindow to 1 packet & enters slow start for both types of loss events (more conservative)
  - the version described above (TCP Reno) removes the slow start phase following a triple duplicate ACK (fast recovery)