Internet routing

- Problem: Route from any node to any other node
  - IP addresses are split into network numbers, route towards the network
  - Network addresses are split into subnetworks at the local site - route towards the subnetwork which contains the destination
  - All nodes within a given subnetwork will be bridged, go through bridges (transparent to IP) to reach the dest.
- E.g. routing from yahoo.com to this podium PC
  - Yahoo server routes to nd network (backbone)
  - Nd forwards to debartolo (site)
  - Bridged from router to 1st floor bridge to wing bridge

Route Propagation

- Know a smarter router
  - hosts know local router (simplicity)
  - local routers know site routers
  - site routers know core router
  - core routers know everything (complex)
- Autonomous System (AS)
  - corresponds to an administrative domain
  - examples: University, company, backbone network
  - assign each AS a 16-bit number
- Two-level route propagation hierarchy
  - interior gateway protocol (each AS selects its own)
  - exterior gateway protocol (Internet-wide standard)

Popular Interior Gateway Protocols

- RIP: Route Information Protocol
  - developed for XNS
  - distributed with Unix
  - distance-vector algorithm
  - based on hop-count
- OSPF: Open Shortest Path First
  - recent Internet standard
  - uses link-state algorithm
  - supports load balancing
  - supports authentication

EGP: Exterior Gateway Protocol

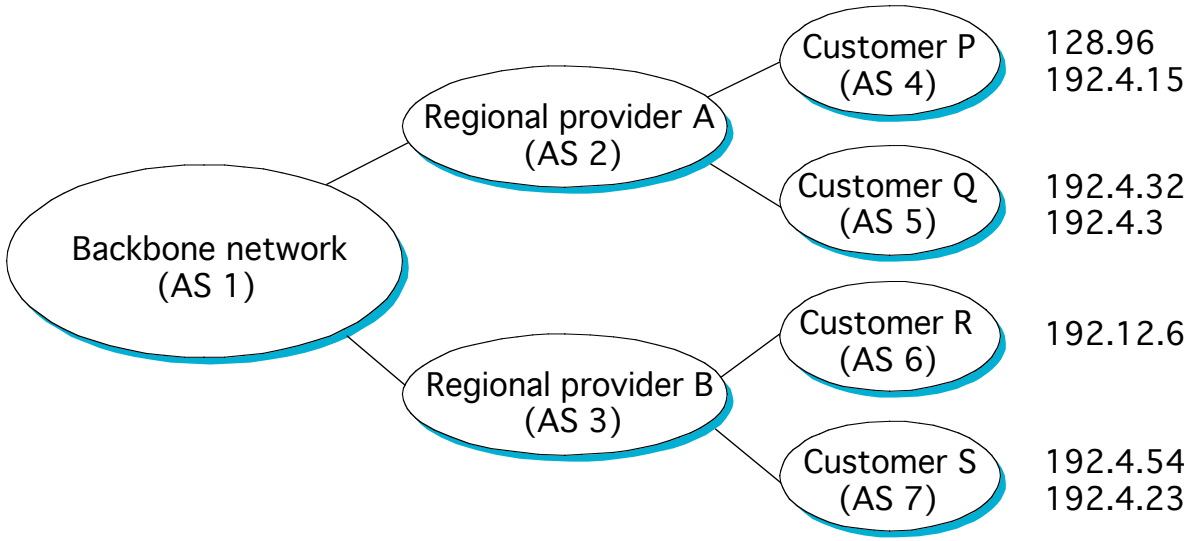
- Overview
  - designed for tree-structured Internet
  - concerned with reachability, not optimal routes
- Protocol messages
  - neighbor acquisition: one router requests that another be its peer; peers exchange reachability information
  - neighbor reachability: one router periodically tests if the another is still reachable; exchange HELLO/ACK messages; uses a k-out-of-n rule
  - routing updates: peers periodically exchange their routing tables (distance-vector)

BGP-4: Border Gateway Protocol

- AS Types
  - stub AS: has a single connection to one other AS
    - carries local traffic only
  - multihomed AS: has connections to more than one AS
    - refuses to carry transit traffic
  - transit AS: has connections to more than one AS
    - carries both transit and local traffic
- Each AS has:
  - one or more border routers
  - one BGP speaker that advertises:
    - local networks
    - other reachable networks (transit AS only)
    - gives path information

BGP Example

- Speaker for AS2 advertises reachability to P and Q
  - network 128.96, 192.4.153, 192.4.32, and 192.4.3, can be reached directly from AS2



- Speaker for backbone advertises
  - networks 128.96, 192.4.153, 192.4.32, and 192.4.3 can be reached along the path (AS1, AS2).
- Speaker can cancel previously advertised paths

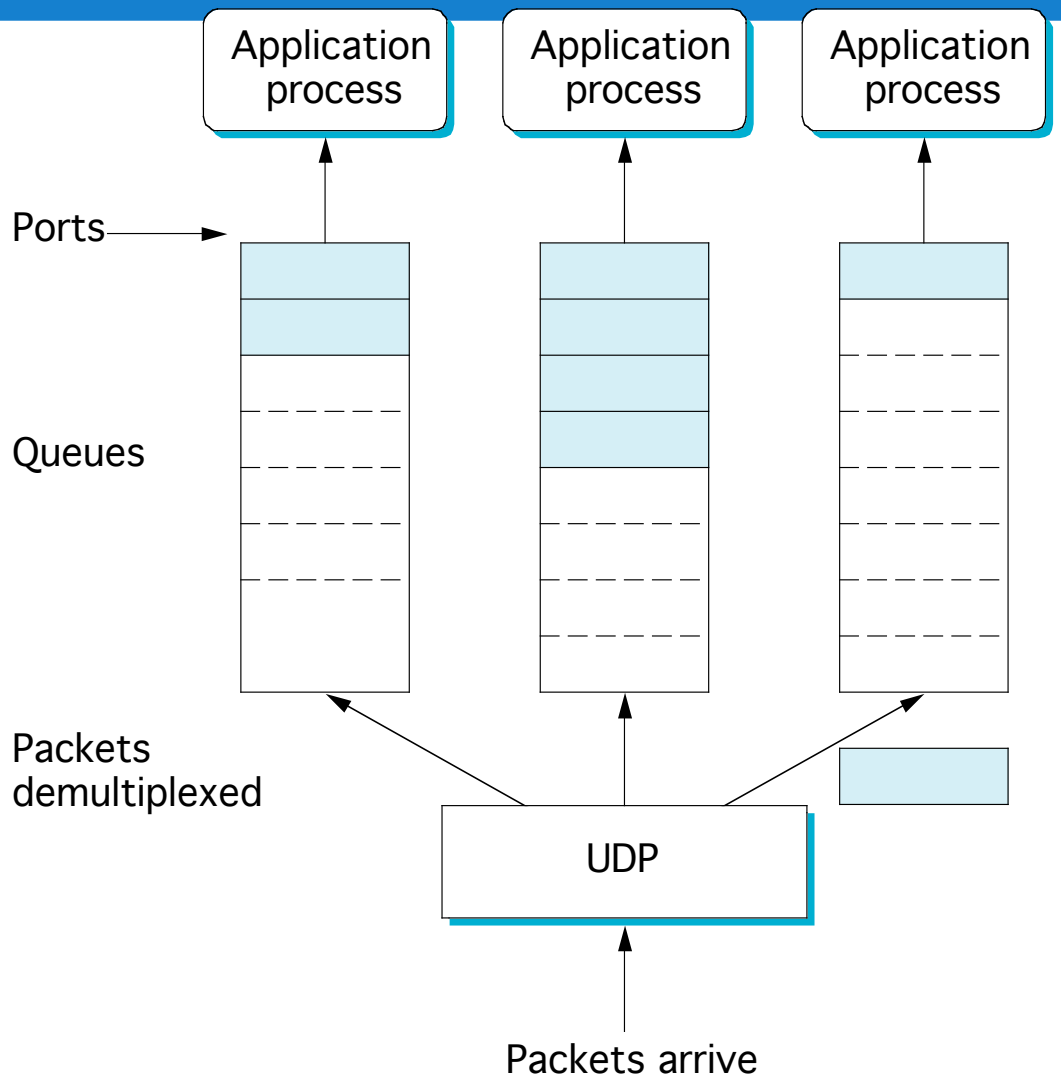
Summary

- We have seen techniques to globally name nodes (ipv4, ipv6), mechanisms to map from global names to IP address (DNS), IP addresses to physical address (ARP), route from local to site router (bridge), within site (RIP), and the internet (BGP).
- We can connect from any host to any other host
- Next problem is, how do we get these hosts to communicate and do interesting things

End-to-End Protocols

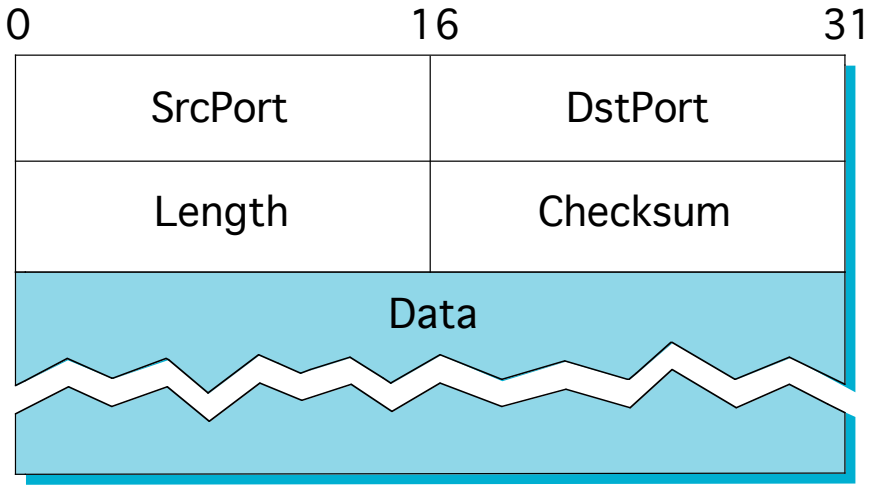
- Underlying best-effort network
  - drop messages
  - re-orders messages
  - delivers duplicate copies of a given message
  - limits messages to some finite size
  - delivers messages after an arbitrarily long delay
- Common end-to-end services
  - guarantee message delivery
  - deliver messages in the same order they are sent
  - deliver at most one copy of each message
  - support arbitrarily large messages
  - support synchronization
  - allow the receiver to flow control the sender
  - support multiple application processes on each host

UDP message queue



Simple De-multiplexor (UDP)

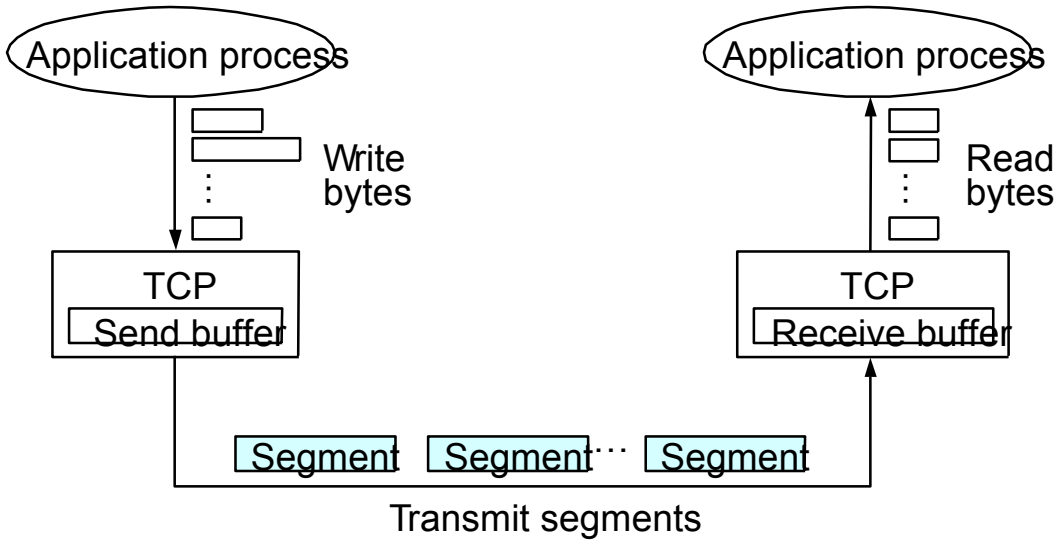
- Unreliable and unordered datagram service
- No flow control, sender and receiver are not capable of caring for what the network can handle
- Adds multiplexing: endpoints identified by ports
  - servers have well-known ports
  - see /etc/services on Unix
- Header format



- Optional checksum
  - psuedo header + UDP header + data

TCP Overview

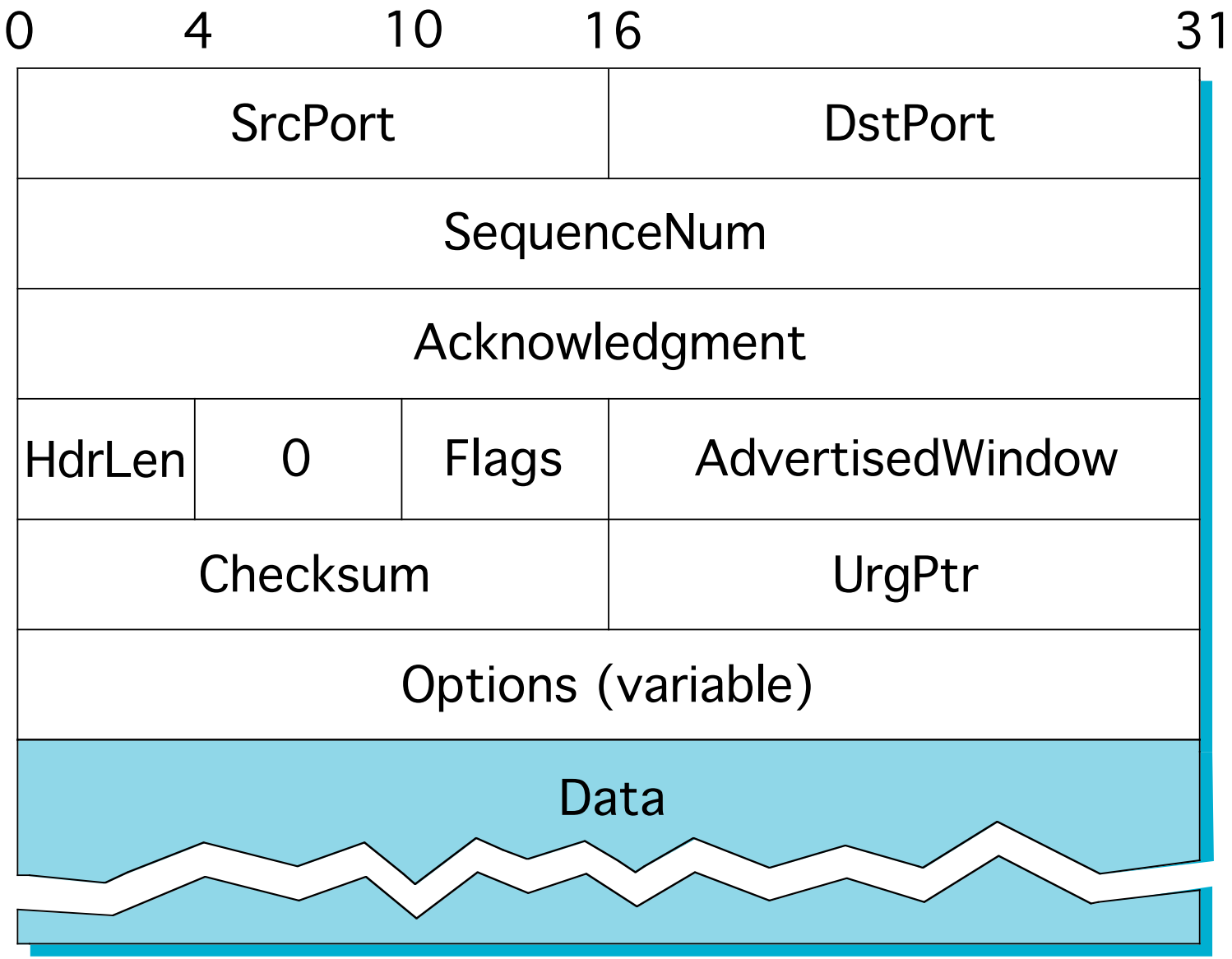
- Connection-oriented
- Byte-stream
  - app writes bytes
  - TCP sends segments
  - app reads bytes
- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network



Data Link Versus Transport

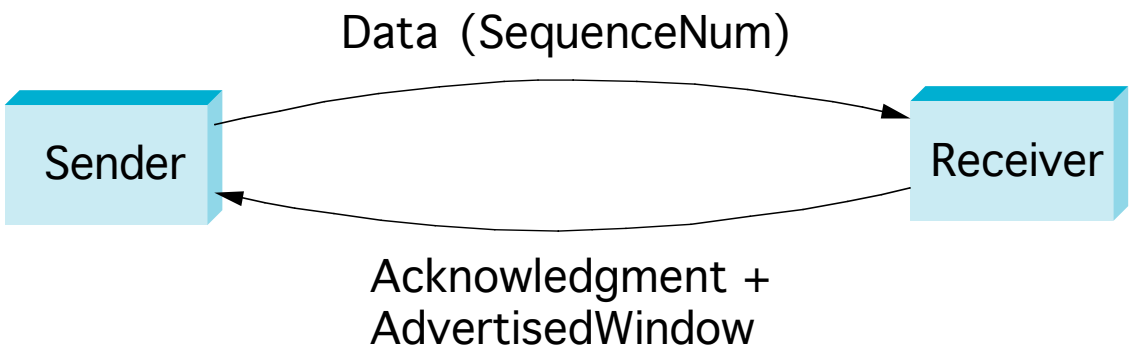
- Potentially connects many different hosts
  - need explicit connection establishment and termination
- Potentially different RTT
  - need adaptive timeout mechanism
- Potentially long delay in network
  - need to be prepared for arrival of very old packets
- Potentially different capacity at destination
  - need to accommodate different node capacity
- Potentially different network capacity
  - need to be prepared for network congestion

Segment Format



Segment Format (cont)

- Each connection identified with 4-tuple:
  - (SrcPort, SrcIPAddr, DstPort, DstIPAddr)
- Sliding window + flow control
  - acknowledgment, SequenceNum, AdvertisedWindow



- Flags
  - SYN, FIN, RESET, PUSH, URG, ACK
- Checksum
  - pseudo header + TCP header + data

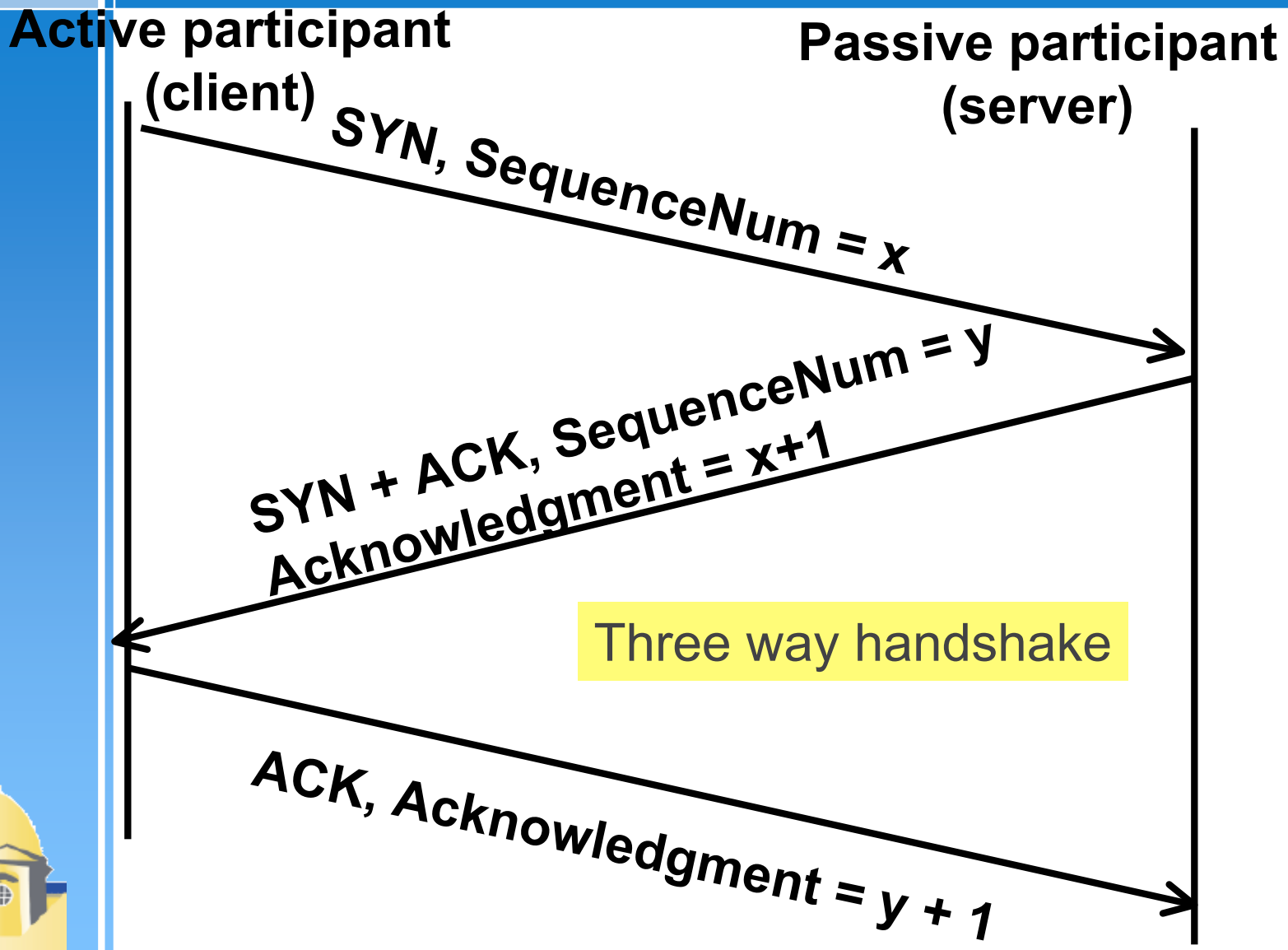
Sequence Number Selection

- Initial sequence number (ISN) selection
  - Why not simply chose 0?
    - Must avoid overlap with earlier incarnation
- Requirements for ISN selection
  - Must operate correctly
    - Without synchronized clocks
    - Despite node failures

ISN and Quiet Time

- Use local clock to select ISN
  - Clock wraparound must be greater than max segment lifetime (MSL)
- Upon startup, cannot assign sequence numbers for MSL seconds
- Can still have sequence number overlap
  - If sequence number space not large enough for high-bandwidth connections

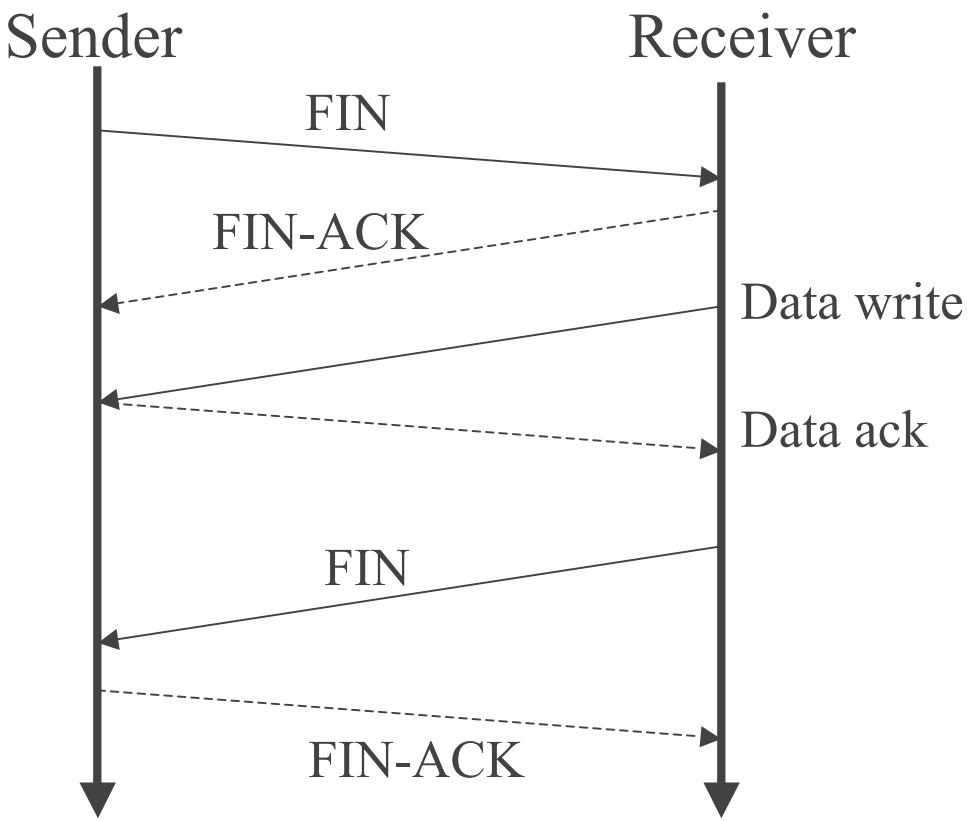
Connection Establishment and Termination



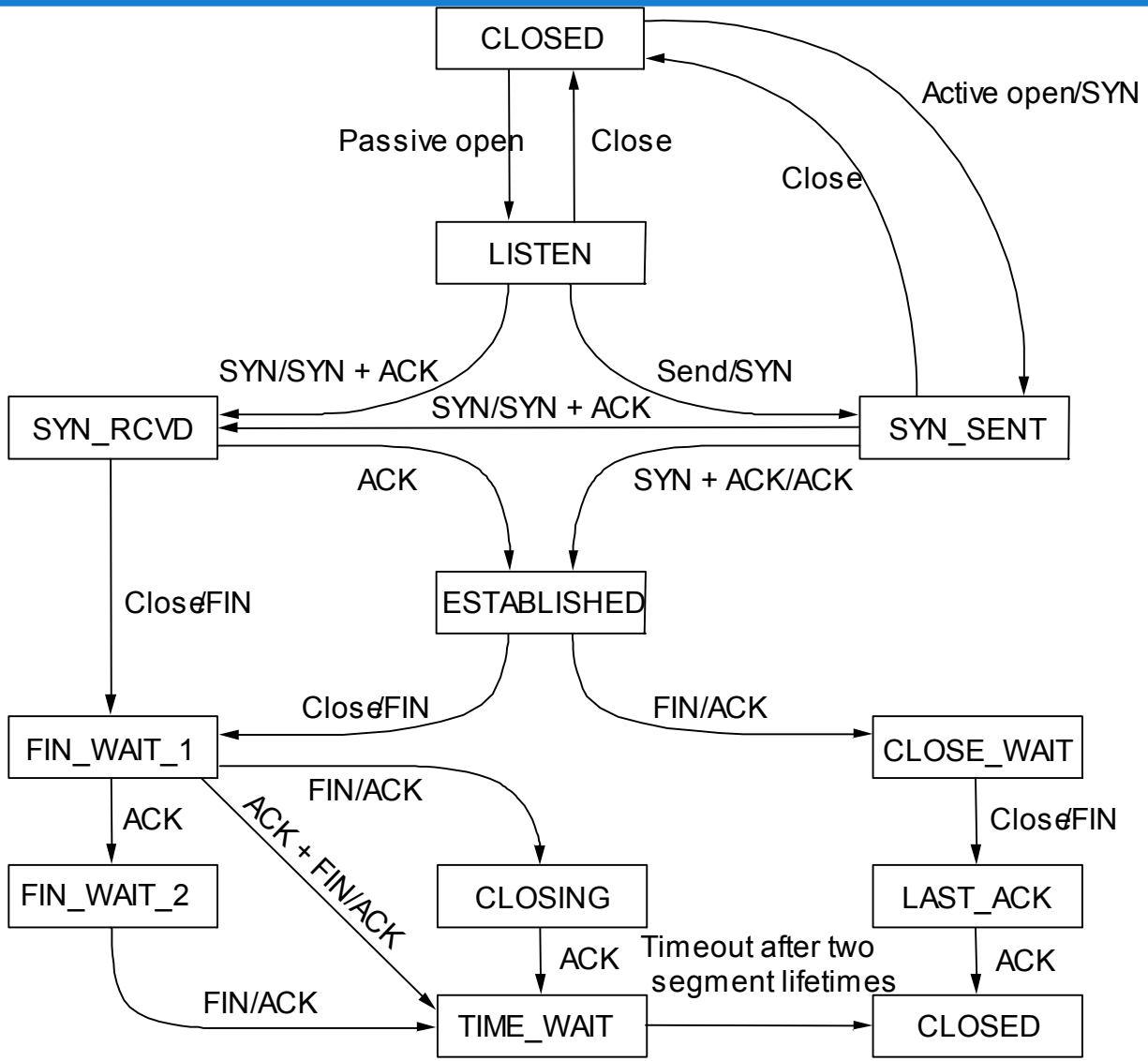
Connection Tear-down

- Normal termination
  - Allow unilateral close
  - Avoid sequence number overlap
- TCP must continue to receive data even after closing
  - Cannot close connection immediately: what if a new connection restarts and uses same sequence number and receives retransmitted FIN from the current session?

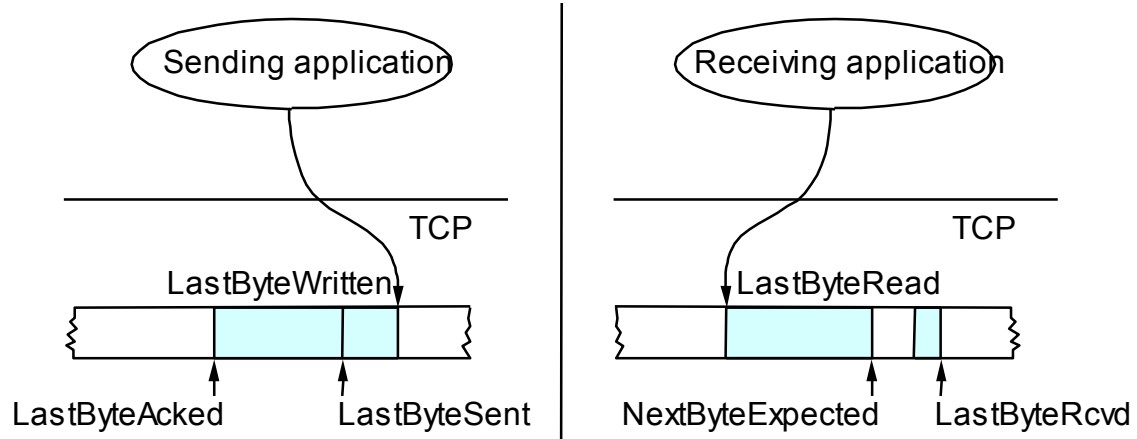
Tear-down Packet Exchange



State Transition Diagram



Sliding Window Revisited



- Sending side
  - `LastByteAcked <= LastByteSent`
  - `LastByteSent <= LastByteWritten`
  - buffer bytes between `LastByteAcked` and `LastByteWritten`

- Receiving side
  - `LastByteRead < NextByteExpected`
  - `NextByteExpected <= LastByteRcvd +1`
  - buffer bytes between `NextByteRead` and `LastByteRcvd`

Flow Control

- Fast sender can overrun receiver:
  - Packet loss, unnecessary retransmissions
- Possible solutions:
  - Sender transmits at pre-negotiated rate
  - Sender limited to a window's worth of unacknowledged data
- Flow control different from congestion control

Flow Control

- Send buffer size: MaxSendBuffer
- Receive buffer size: MaxRcvBuffer
- Receiving side
  - LastByteRcvd - LastByteRead < = MaxRcvBuffer
  - AdvertisedWindow = MaxRcvBuffer - (NextByteExpected - NextByteRead)
- Sending side
  - LastByteSent - LastByteAcked < = AdvertisedWindow
  - EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)
  - LastByteWritten - LastByteAcked < = MaxSendBuffer
  - block sender if (LastByteWritten - LastByteAcked) + y > MaxSenderBuffer
- Always send ACK in response to arriving data segment
- Persist when AdvertisedWindow = 0

Round-trip Time Estimation

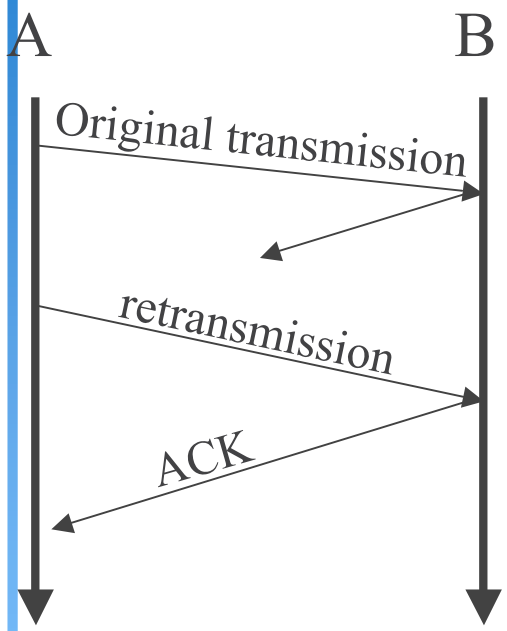
- Wait at least one RTT before retransmitting
- Importance of accurate RTT estimators:
  - Low RTT -> unneeded retransmissions
  - High RTT -> poor throughput
- RTT estimator must adapt to change in RTT
  - But not too fast, or too slow!
- Problem: If the instantaneously calculated RTT is 10, 20, 5, 12, 3 , 5, 6; what RTT should we use for calculations?

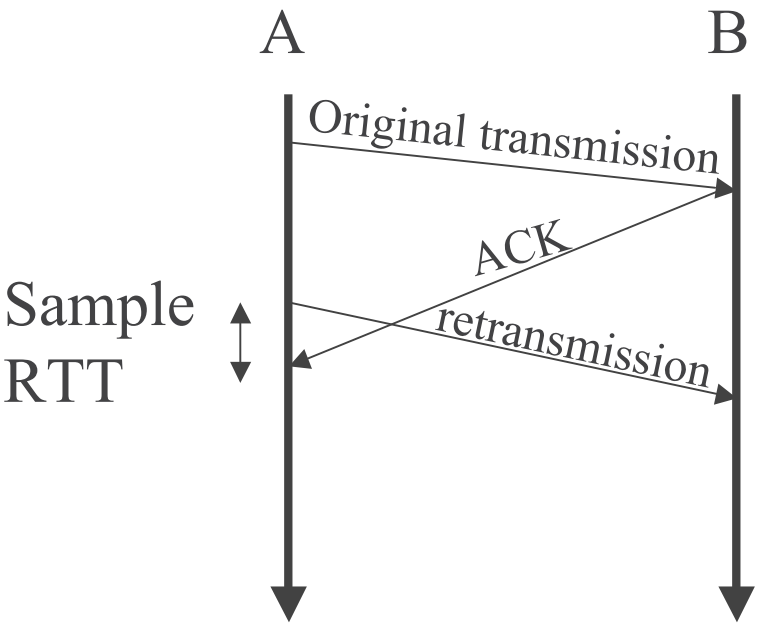
Initial Round-trip Estimator

Round trip times exponentially averaged:

- New RTT = α (old RTT) + (1 - α) (new sample)
- Recommended value for α: 0.8 - 0.9
- Retransmit timer set to β RTT, where β = 2
- Every time timer expires, RTO exponentially backed-off

Retransmission Ambiguity



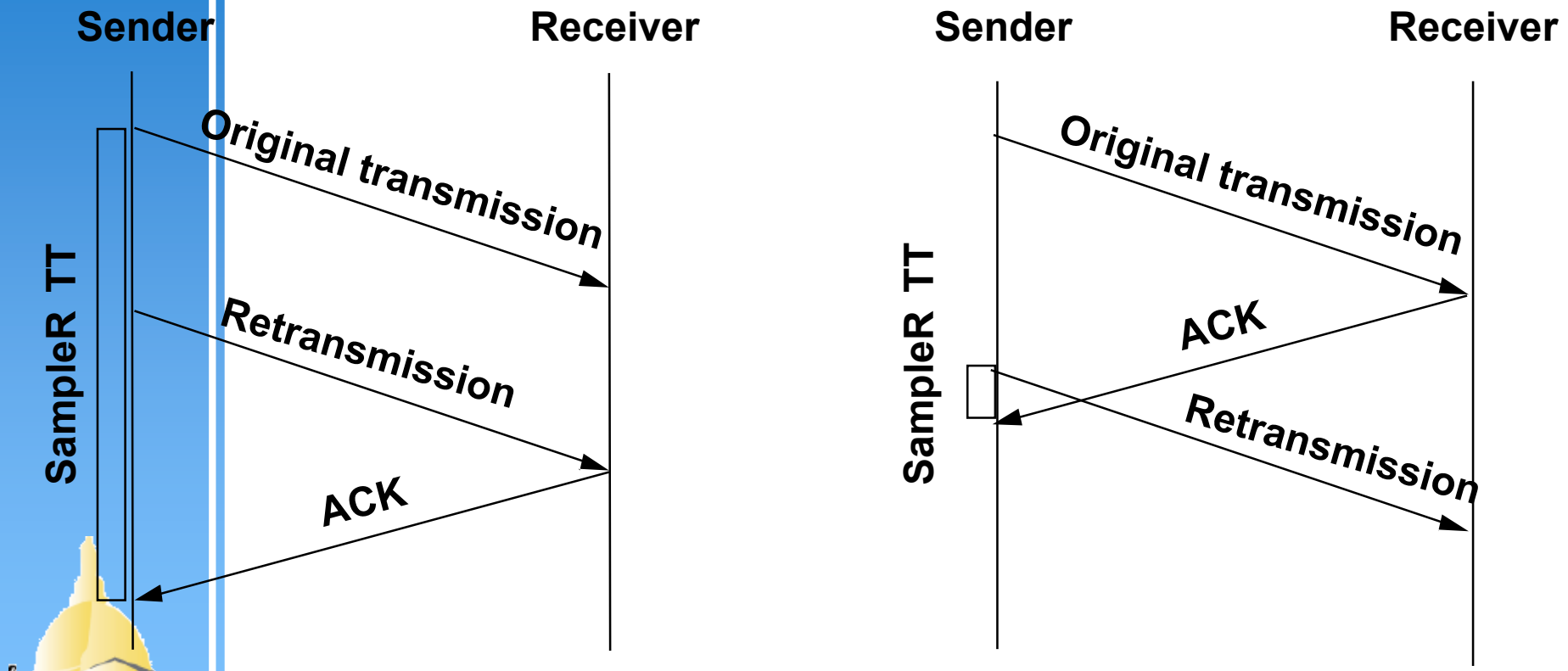


Karn's Retransmission Timeout Estimator

- Accounts for retransmission ambiguity
- If a segment has been retransmitted:
  - Don't count RTT sample on ACKs for this segment
  - Keep backed off time-out for next packet
  - Reuse RTT estimate only after one successful transmission

Karn/Partridge Algorithm

- Do not sample RTT when retransmitting
- Double timeout after each retransmission



Jacobson's Retransmission Timeout Estimator

- Key observation:
  - Using β RTT for timeout doesn't work
  - At high loads round trip variance is high
- Solution:
  - If D denotes mean variation
  - Timeout = RTT + 4D

Jacobson/ Karels Algorithm

- New Calculations for average RTT
- Diff = SampleRTT - EstRTT
- EstRTT = EstRTT + (d x Diff)
- Dev = Dev + d(|Diff| - Dev)
  - where d is a factor between 0 and 1
- Consider variance when setting timeout value
- TimeOut = m x EstRTT + f x Dev
  - where m = 1 and f = 4
- Notes
  - algorithm only as good as granularity of clock (500ms on Unix)
  - accurate timeout mechanism important to congestion control (later)