The Network as a Shared Resource

Prof. C. Tschudin, M. Sifalakis, T. Meyer,
G. Bouabene, M. Monti

University of Basel

Cs321 - HS 2011

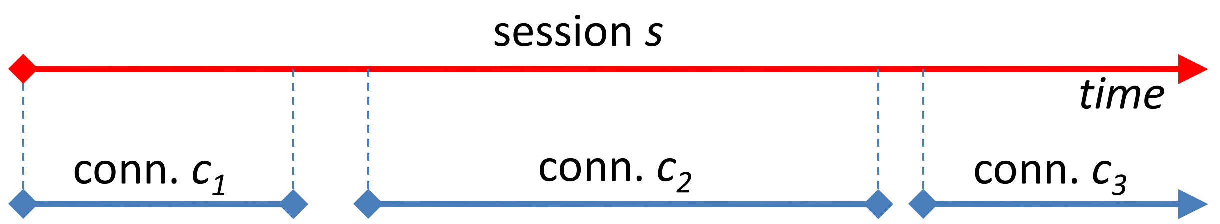
Overview

- Session Setup and Management
  - The problem of reliable session set-up (how many messages)
- Sharing the network
  - Load and Fairness
  - Scheduling and Queuing
  - Congestion Control
  - Congestion Avoidance

Session Management

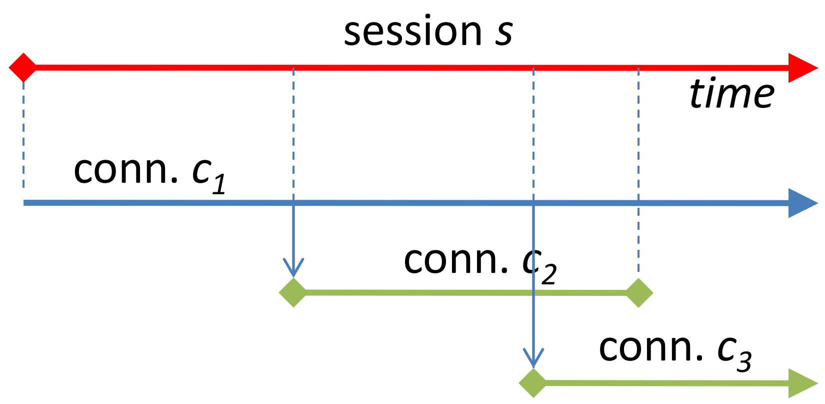
- Current network infrastructure schedules access to resources: Communication Sessions
- OSI model has a specific session (management) layer: "dialog control"
  - Session establishment
  - Session maintenance and synchronisation
- Session > Connection
  - One session can span across several connections (in space or time)
  - Checkpointing (for acknowledgement)
- Examples from the non-OSI world
  - RPC, X-windows, LDAP, NFS, VoIP, ...

Session > Connection



<u>Multiple connections in time</u>

E.g. Cookie-based services



<u>Multiple connections in space</u>

E.g. FTP, Torrent, VoIP, ...

Session Set-up: How many messages ?

- How many messages need to be exchanged for reliably establishing a session/connection ?
  - Sounds trivial ?
- Belnes 1976: 5 messages
- Schoone 1987: formally proven
- Attiya & Rapport 1997: prove different properties of different levels/models of handshake

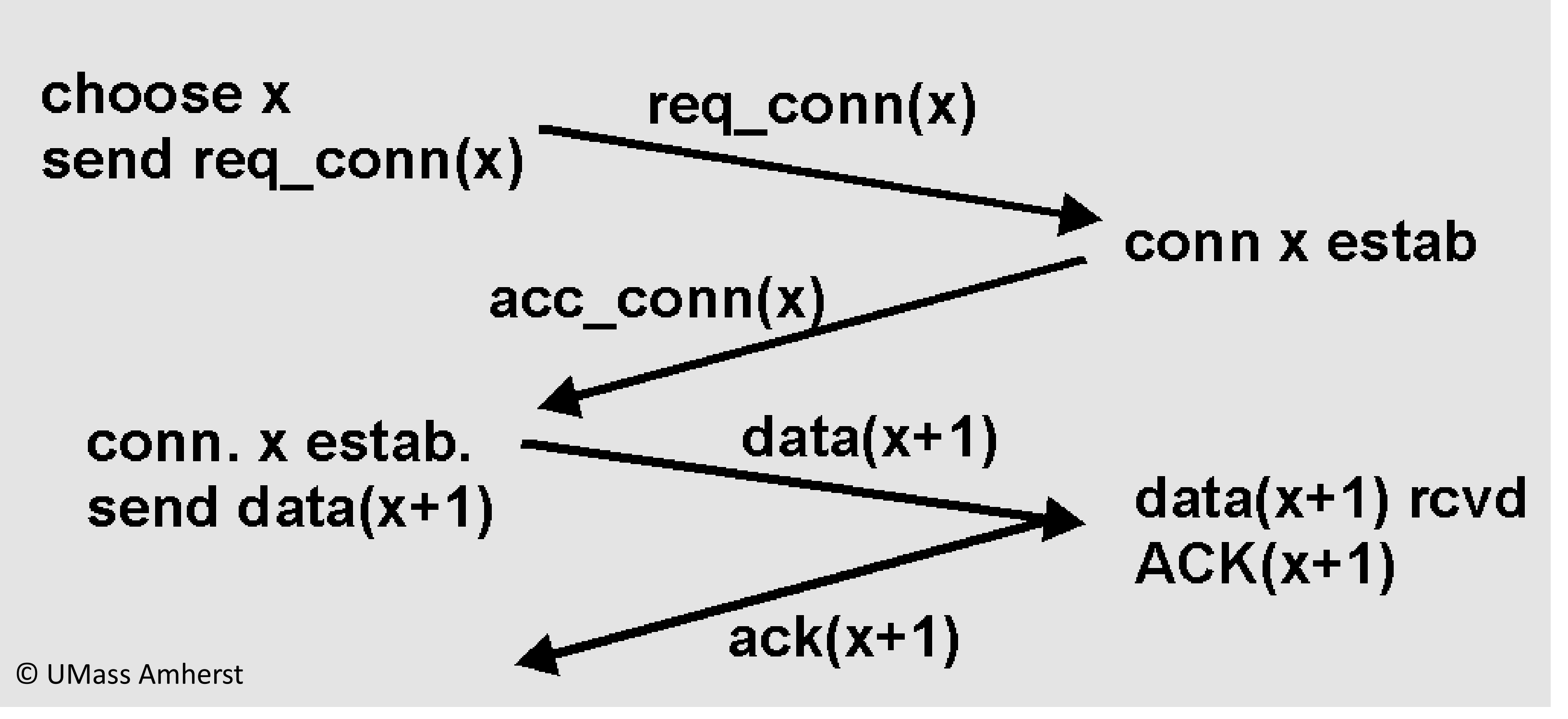
  "(4) if a bound on maximum packet lifetime is known, then a 2-way handshake incarnation management protocol exists, in which the server does not retain connection-specific information between incarnations."
- How reliable ?

The problem of (reliable) Session/Connection Set-up

- Setting: 2 hosts setting up many connections over time (incarnations)
- Task: Synchronise the hosts with a handshake mechanism
  - Lost, delayed, duplicated messages
  - Nodes crash (losing connection state)
- Error to protect against
  - Data from one session delivered in another session

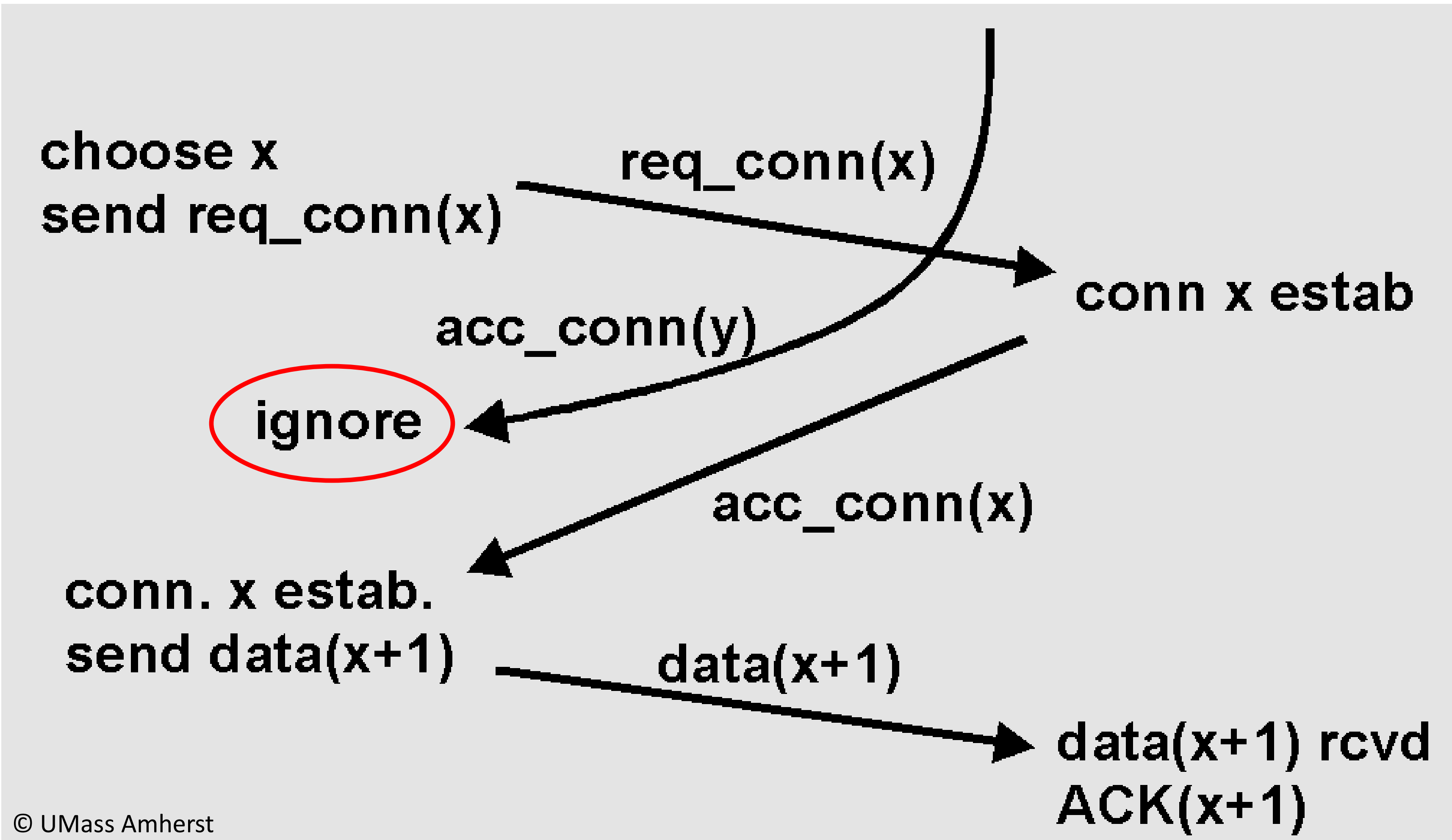
Simple 2-way Handshake

- Unidirectional Connection Setup
- Data and ACKs have sequence numbers

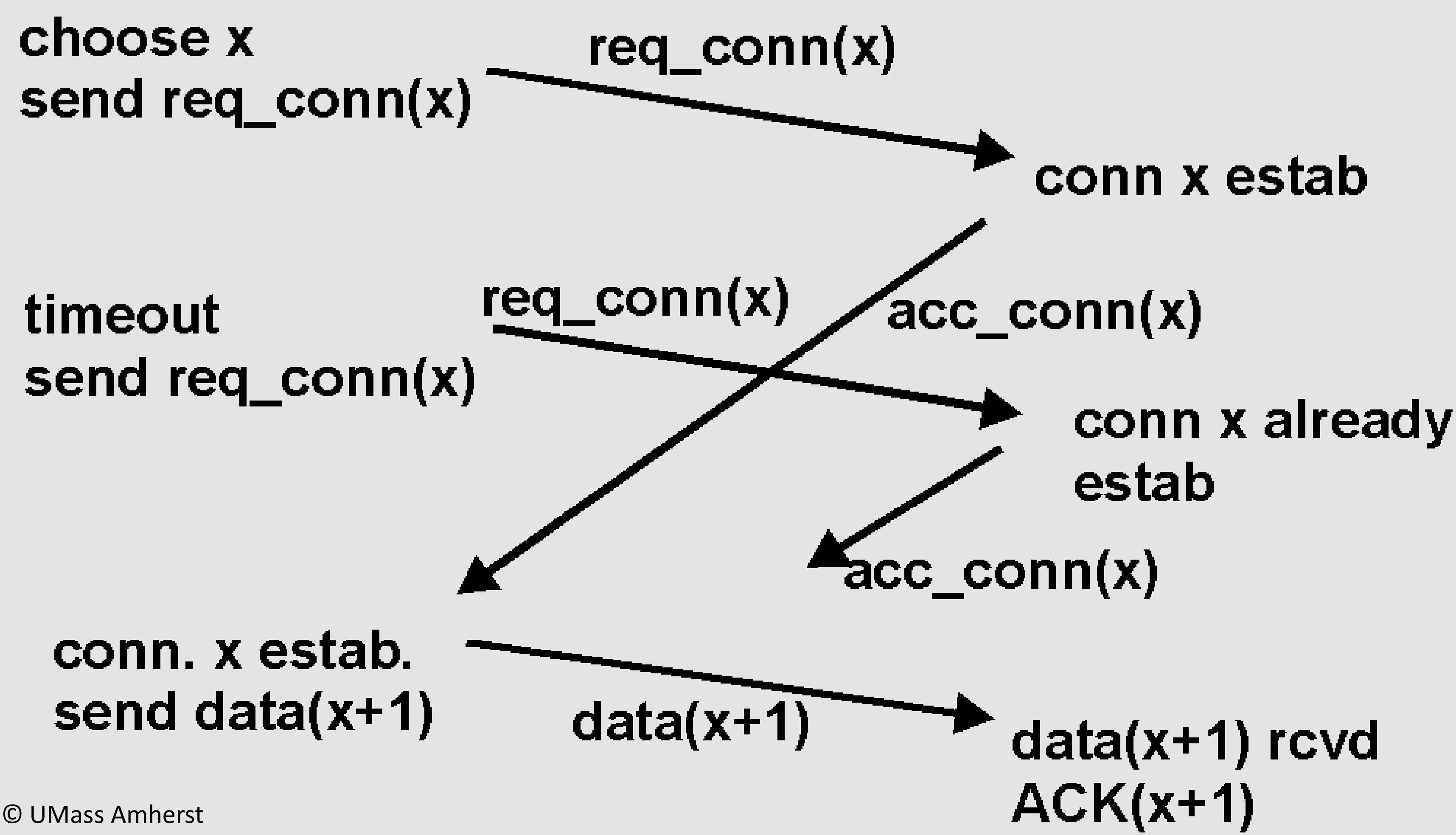


© UMass Amherst

2-way Handshake:
Delayed confirmations ignored

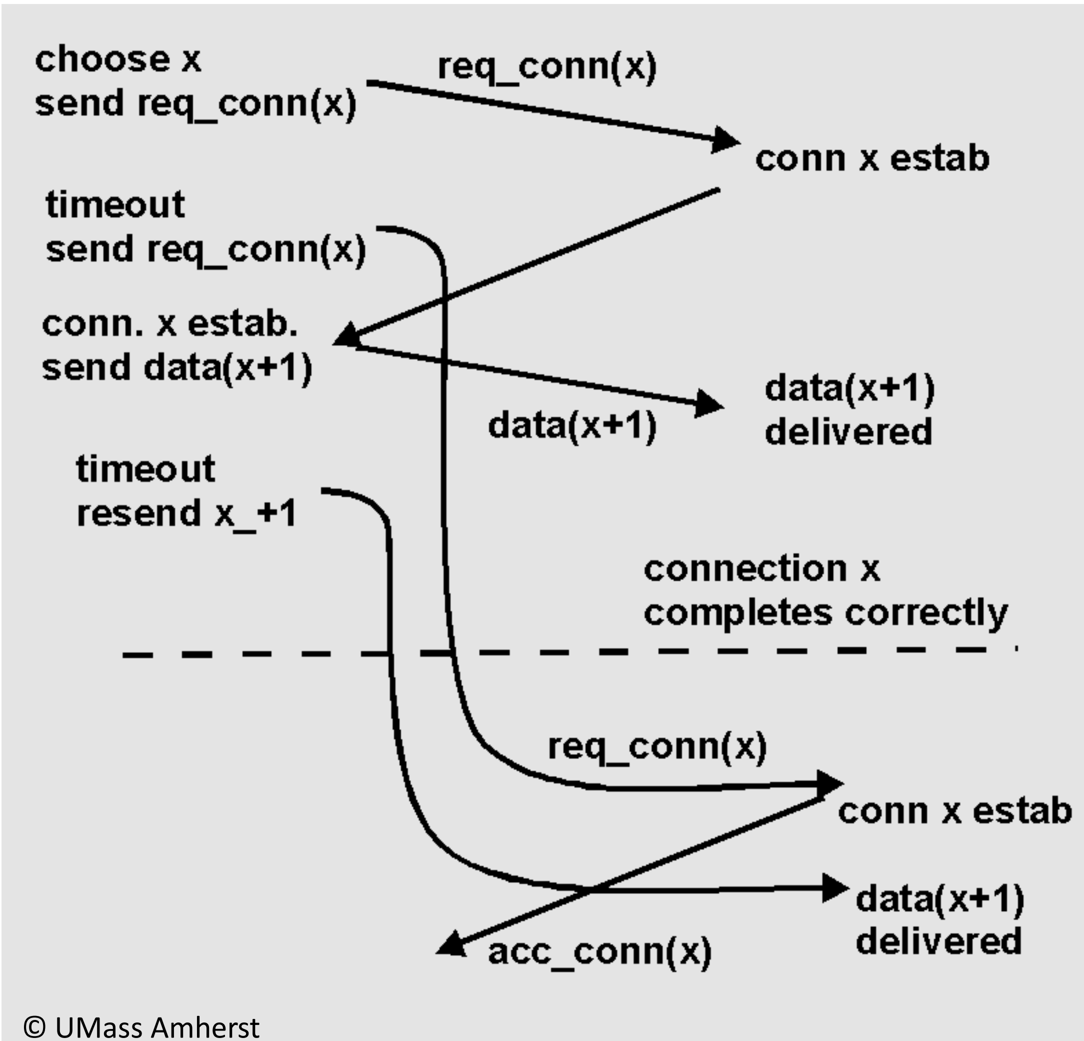


2-way Handshake: Duplicate requests blocked



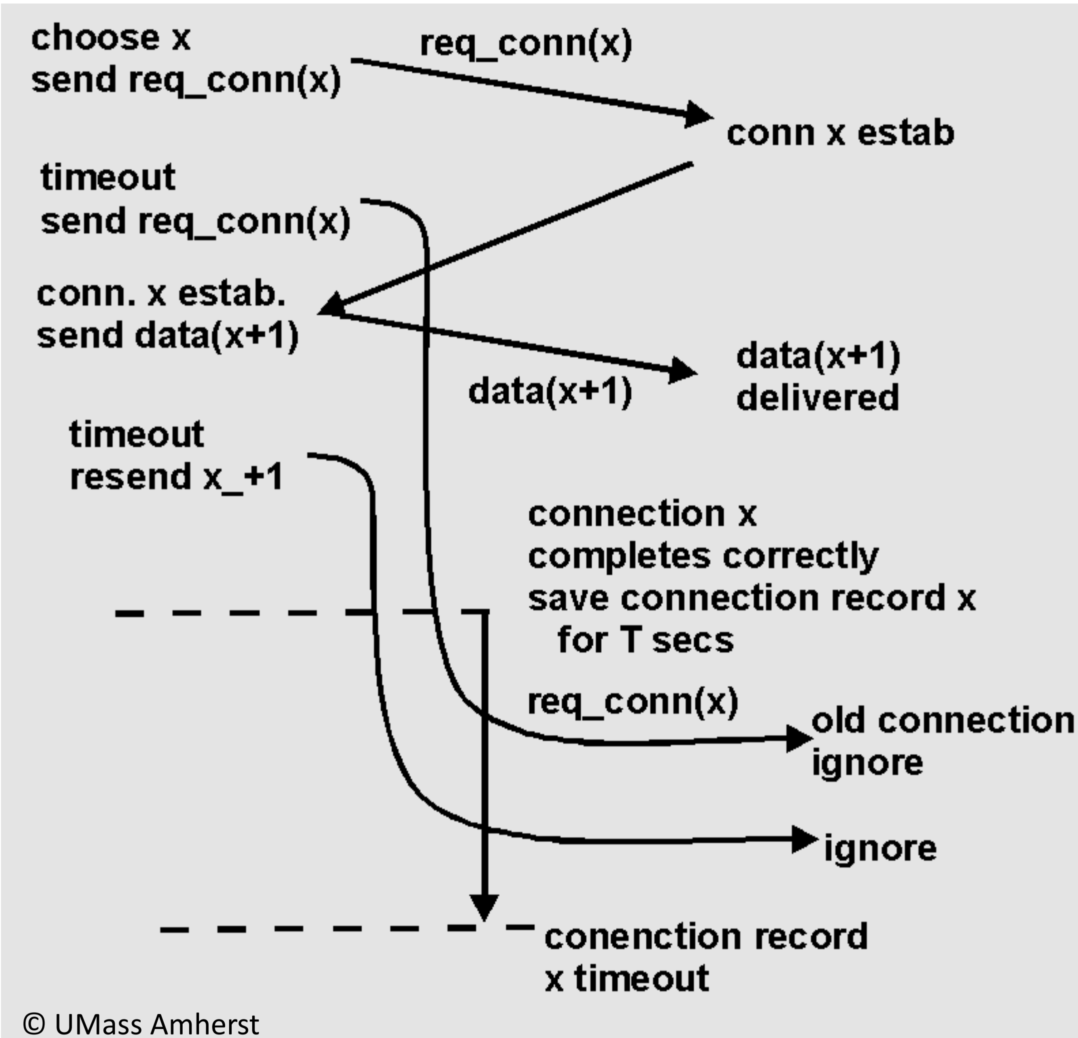
© UMass Amherst

2-way Handshake: … but well not always



- If connection establishment is delayed too much …
- … and in the mean time the old connection is lost …
- Then, the delayed request may be handled as a new connection
- … and old retransmissions are delivered again

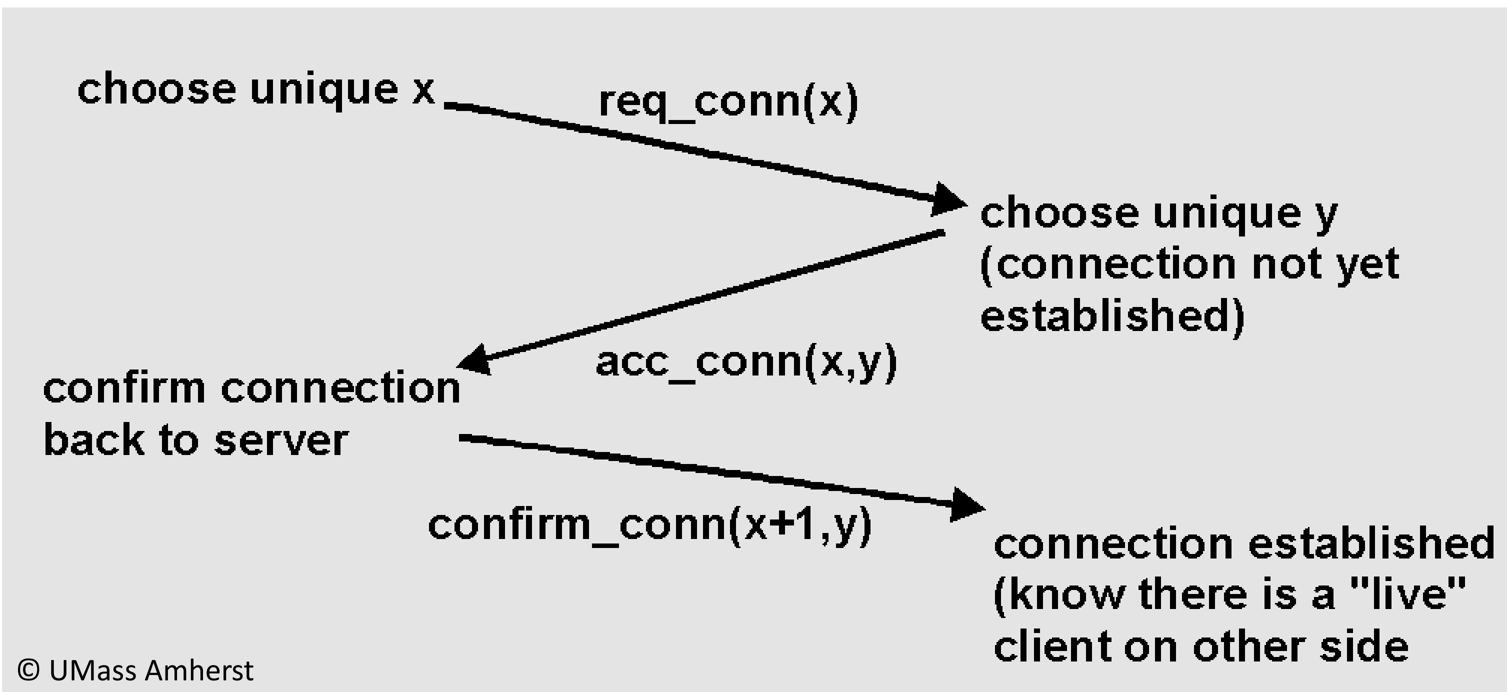
2-way Handshake:
... Timers to the rescue



- Setting a maximum packet life time, addresses the issue
- But requires a Timer and Memory on the server side
- Exploitable for DoS

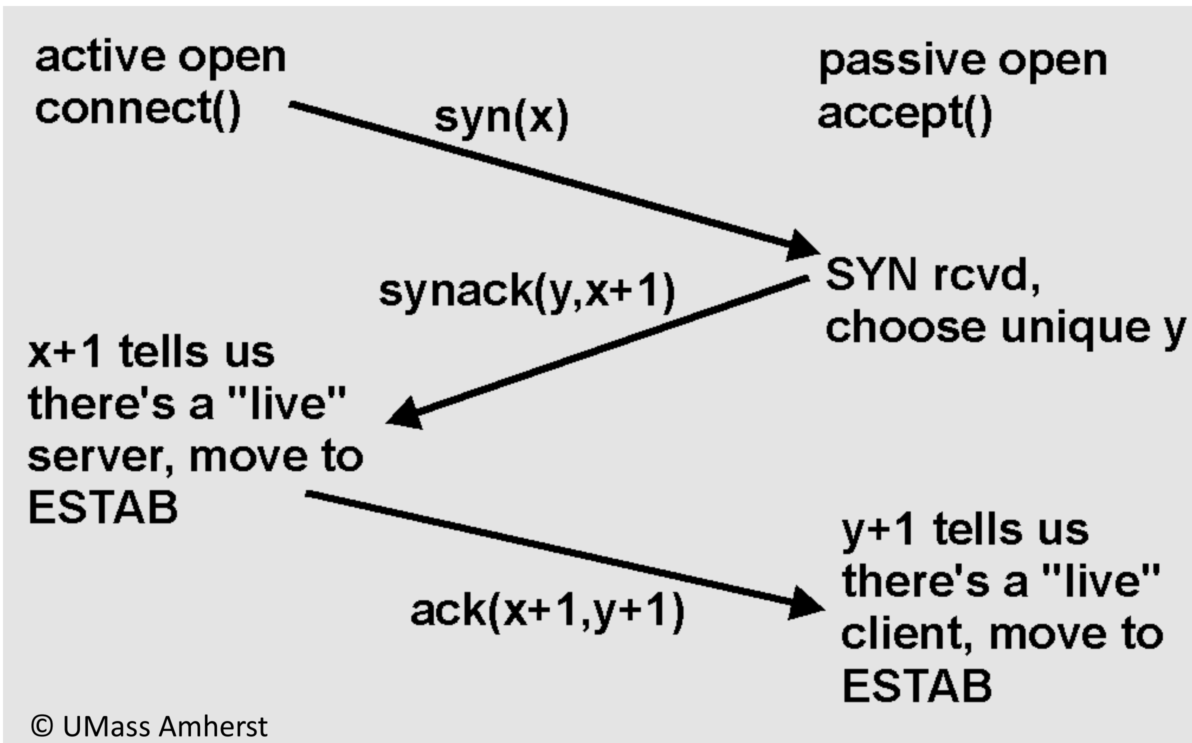
A better solution: 3-way Handshake + Unique IDs

- No Timers needed on the server
- (Eternally) unique session IDs on both sides
- Still requires some memory!



An even better solution: TCP's 3-way Handshake

- Almost unique session IDs
- Bilateral SYN/ACK
- FIN_WAIT state: guard timer
- No Memory



- Accidents still possible, but low probability (increases with very large numbers of parallel TCP sessions)

Sharing the Network

- Communication sessions compete for resources
- Intent: multiplexing multiple communications over the same resources
- Resource allocation: who will use/get how much
  - Storage, processing, access to a link
- Scheduling: who uses the resource, and when (implements resource allocation)
  - Resource allocation as resource occupancy in time (time multiplexing)

Some key concepts

- Method of access to a resource (e.g. Link)
  - Opportunistic (e.g. Aloha),
  - Token-based (e.g. Token ring),
  - Synchronised (e.g. Time divided – TDMA , Duty cycles)
- Level of access (scheduling decisions granularity)
  - Per information quantum (e.g. Packet, frame)
  - Per information flow (e.g. Src/Dst)
  - Per service type (e.g. Traffic class, ToS)
- Distribution Metric (optimisation of schedule)
  - Load
  - Fairness

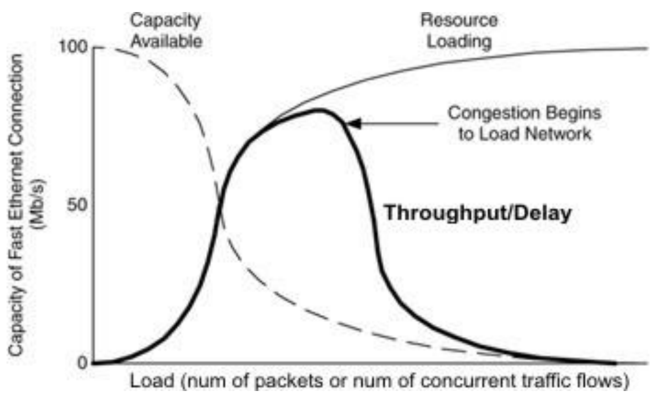
Understanding “Load” as a metric

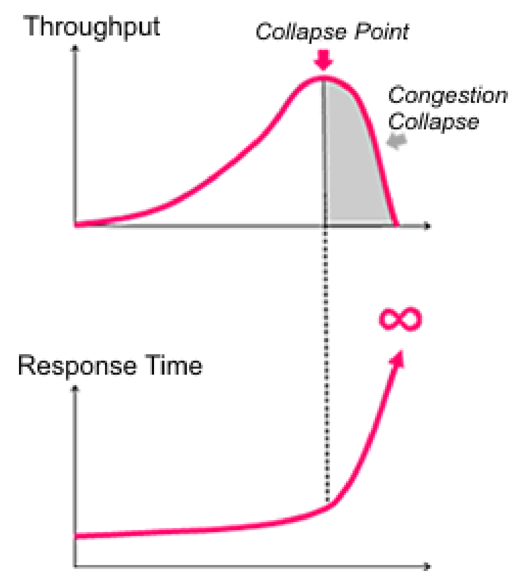
- Two principle (quantifiable) flow metrics in the network:
  - Throughput = how much information gets transmitted (pkts/bits/bytes per sec)
  - Delay = how long to be delivered
- Power of the network:

  P = Throughput / Delay
- Boundaries
  - Upper: Link capacity
  - Lower: Queue length

Understanding “Load” as a metric

- Increasing queue size increases utilisation of the network up to a point, beyond which the system thrashes





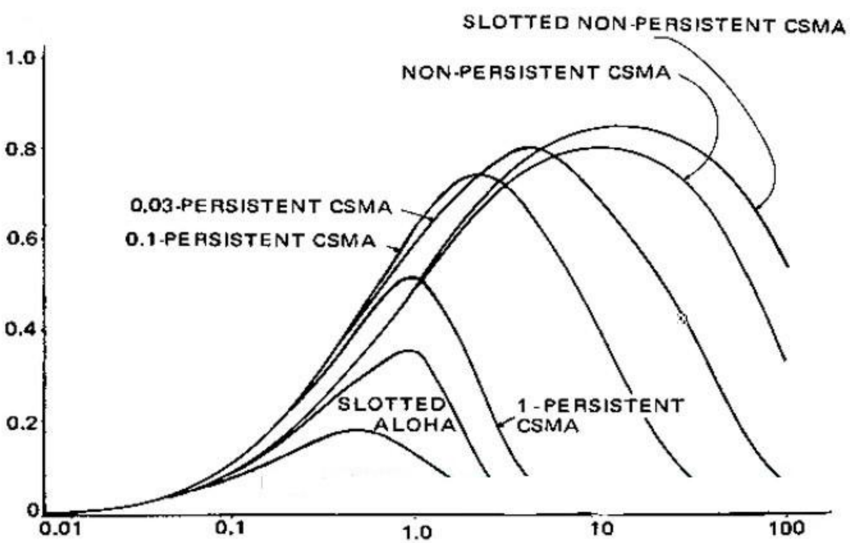
© Figure: McCabe J. D. Piscitello D. M., “Network Analysis, Architecture and Design”

© Figure: APICA http://www.apicasystem.com/

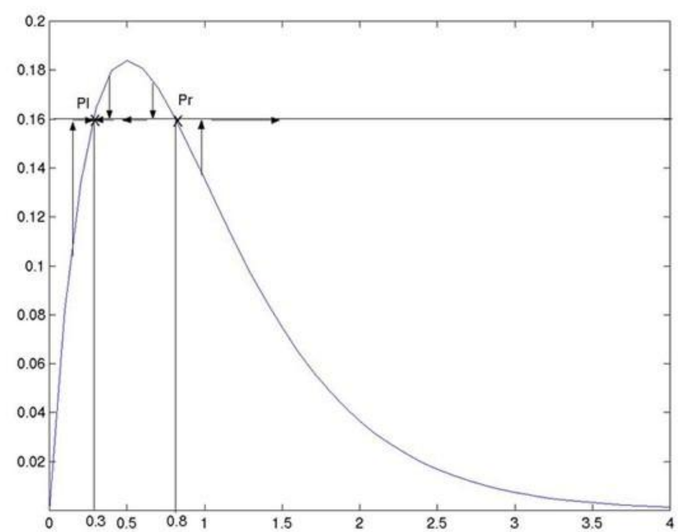
Understanding "Load" as a metric

- Controlling the situation is unfortunately not as simple as setting a knob (more soon on congestion control/ avoidance)
- Engineering efforts in two parallel directions

a) Improve/prevent the boundary conditions at the link/queue level



b) Regulate flow behaviour near the boundary conditions at the protocol level



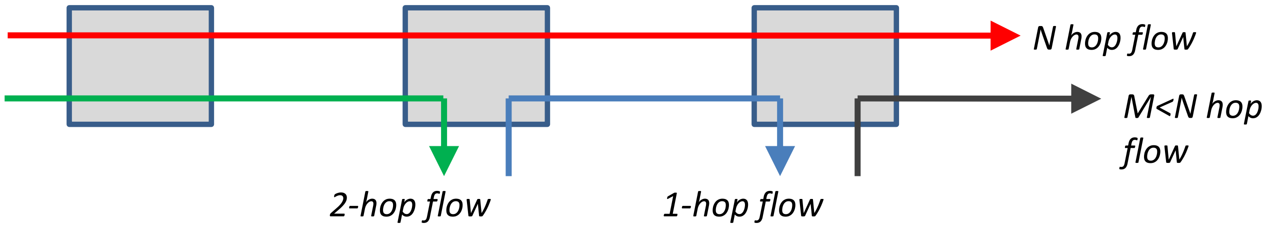
© Figures: Henrik Schiøler, Aalborg University Department of Control Engineering

Defining "Fairness" as a metric

- R. Jain '84: Given a set of flow throughputs *(x_1, x_2,..., x_1)* the fairness of a scheme is determined by the index (1 mean a scheme is 100% fair):

$$\mathcal{J}(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2}$$

- But what does "fair" mean for a queuing system ?
  - When comparing flows of different path lengths



  - Or when competing flows have different timeliness/throughput needs ?

Back to Scheduling

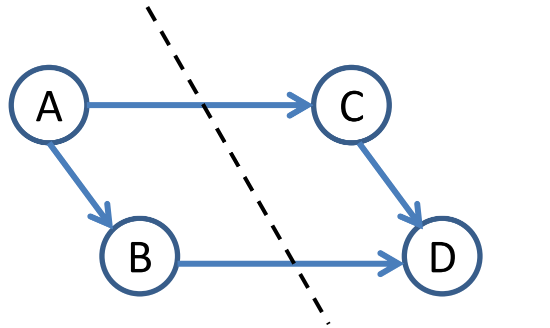
- Scheduling process
  - Scheduler: decision making "machinery"
  - Schedule: algorithm or strategy
- Optimisation criterion:
  - Fairness: each process receives a fair share of the resource
  - Efficiency: keep utilisation above a certain level
  - Response time: minimise delay between request and response
  - Turnaround: Minimise delay between submission of a request and its completion – batch mode
  - Throughput: Get as much information across as possible

Some of these goals cannot be satisfied simultaneously!

Preemptive Scheduling

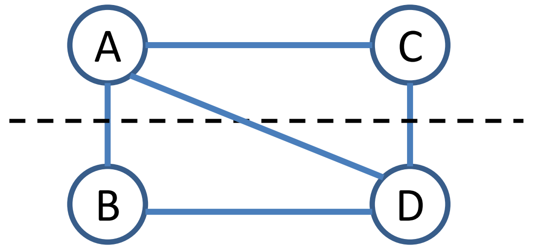
- Early scheduling systems where abiding to a *run-to-completion policy* (!) – aka *non-preemptive*
  - Access to the resource was maintained until the task is finished
- In *preemptive* scheduling, a task can be suspended and the resource can be granted to another task
  - Care needed to prevent race conditions: semaphores, monitors, messages, in (distr.) OS design
  - Difficult to support at large scale in the network
    - For race conditions counter-measures need to be taken within the competing processes (e.g. flow and congestion control)
    - Preempting only at packet granularity

Different assumptions for the schedule



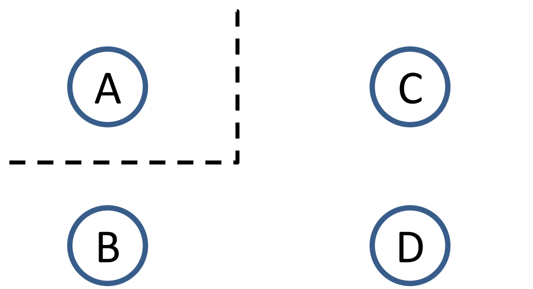
Precedence process model (distributed systems):

- Ordering and priority (arrows) between service flows need to be respected
- Processes interact synchronously
- Data insight needed (e.g. return codes, ACKs)



Communication process model (internet services):

- Relevance and communication (links) across service flows need to be respected
- Processes communicate asynchronously
- Service (process) dependence insight is needed

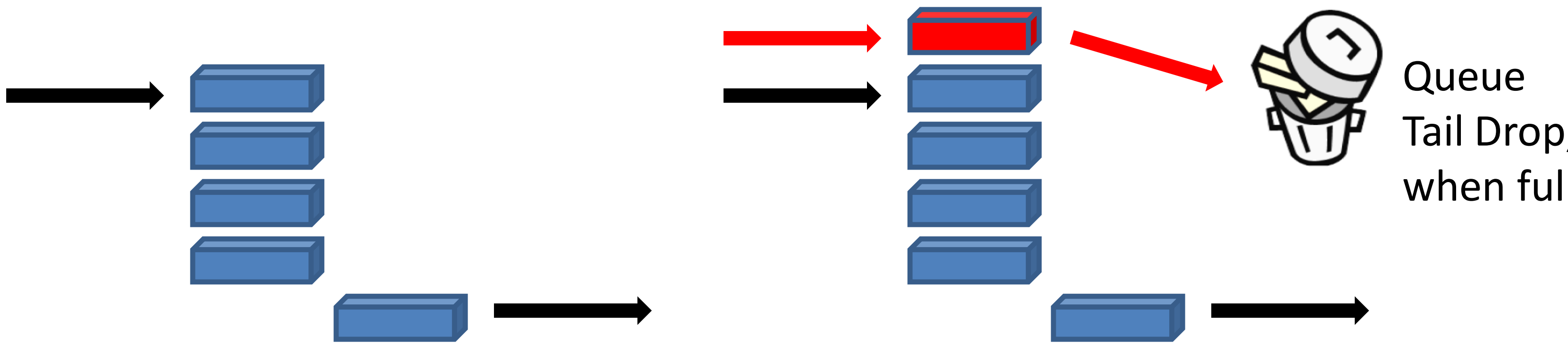


Disjoint process model (initial design in IP):

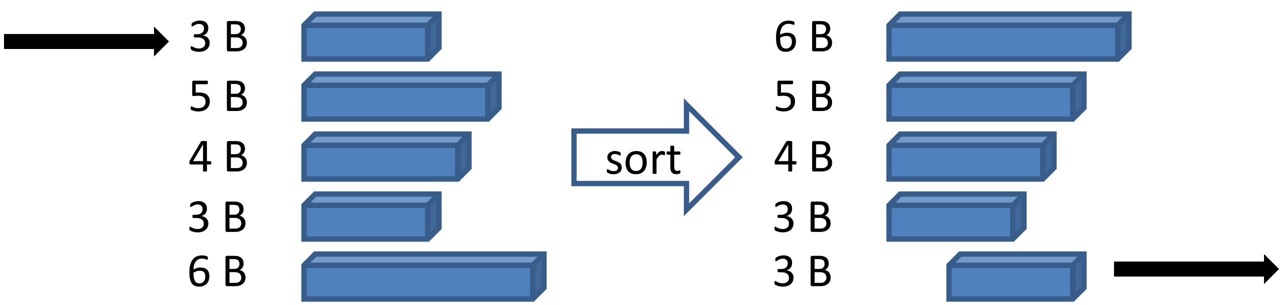
- Nothing about the processes is known to the scheduler
- Processes interact implicitly
- No insight needed

Common Scheduling strategies

- Single queue, FIFO schedule (+ tail-drop policy)



- Single queue, Shortest Job First

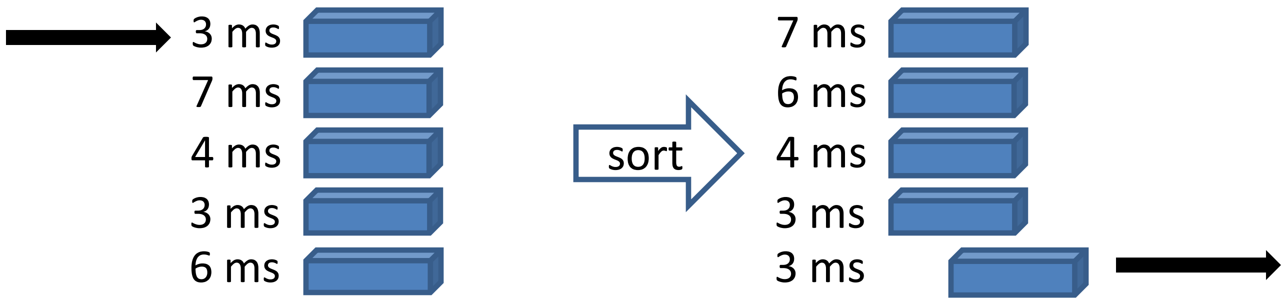


For 1 Bps throughout
Avg. wait time: 13.4ms

For 1 Bps throughput
Avg. wait time: 11ms

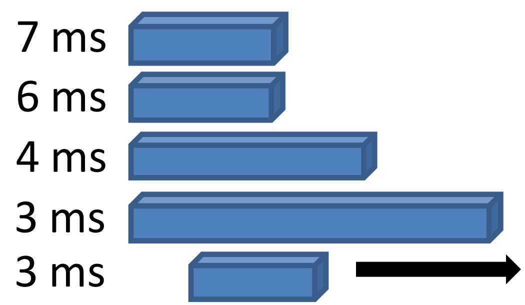
Common Scheduling strategies

- Single queue, Earliest Deadline First
  - Introduced for real-time scheduling in operating systems (works well with preemptive scheduling of processes)



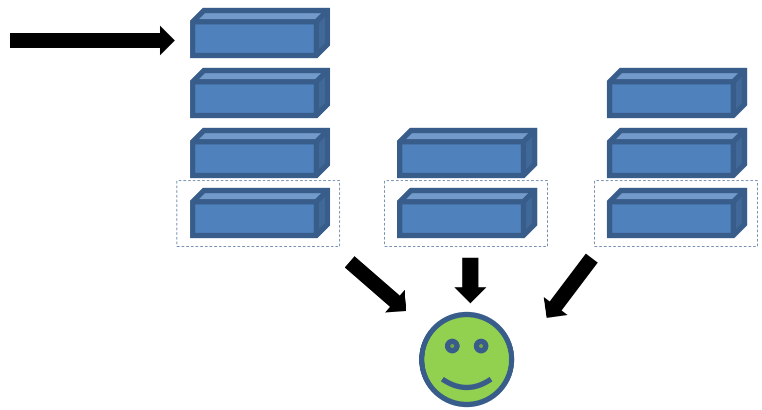
  - Problematic for network queueing unless all packets have the same size

Hard to respect real-time deadlines in such a case



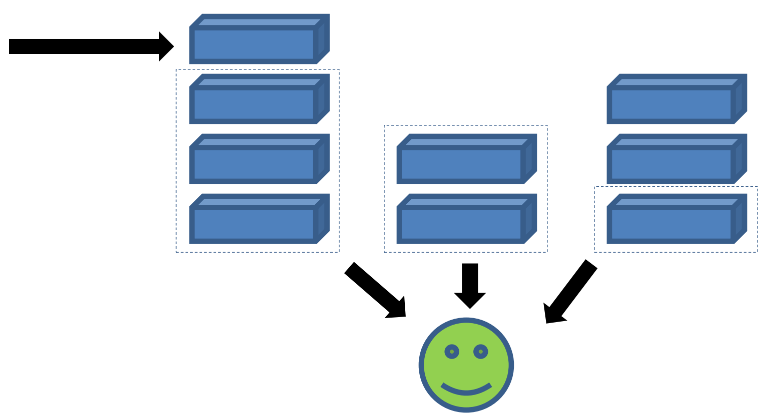
Common Scheduling strategies

- Multiple queues, Round-robin (fair) schedule



Each (aggregate) flow gets its own queue

- Multiple queues, Priority (weighted fair) schedule



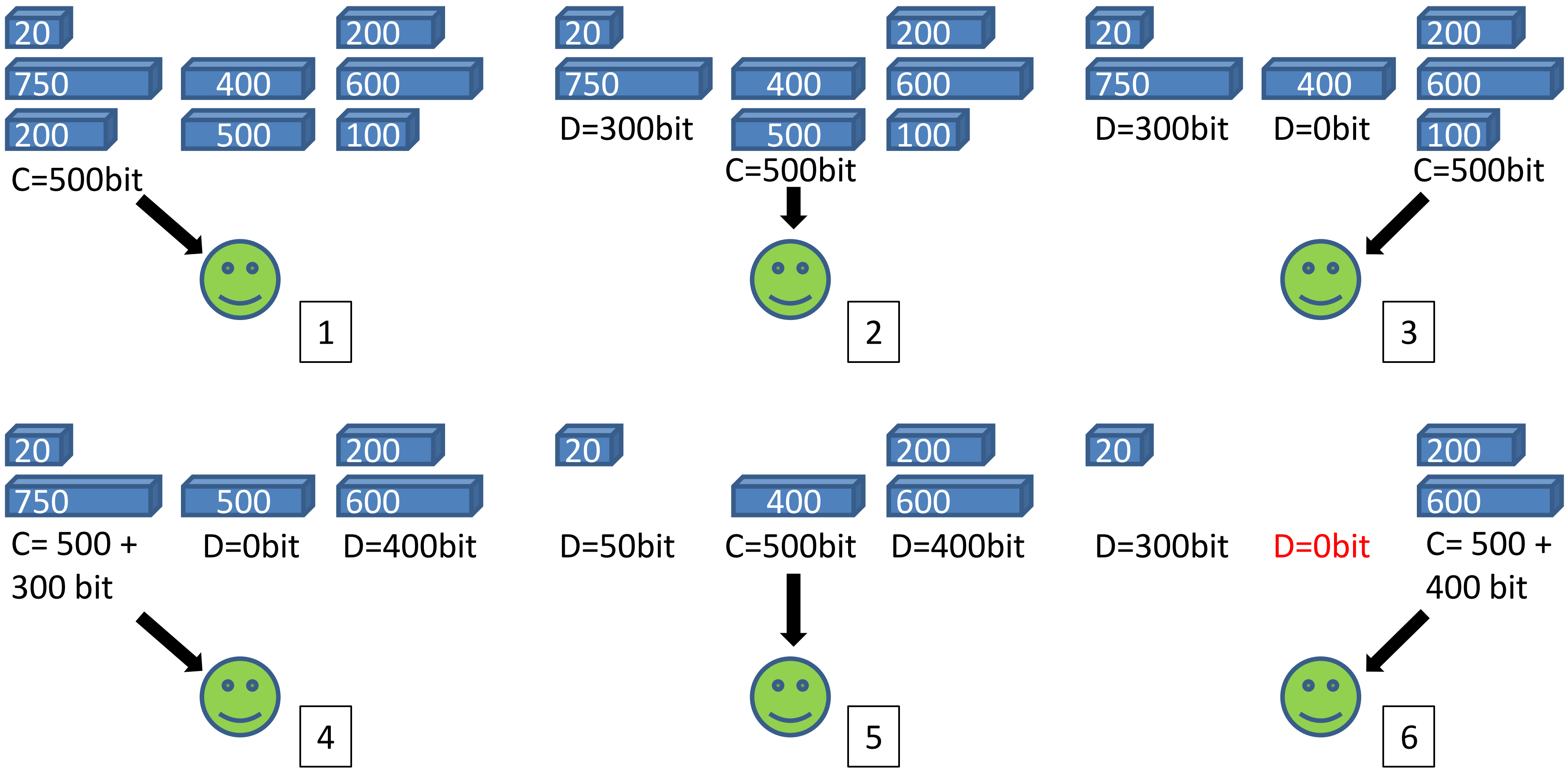
Each class of traffic (various premiums for customers) or each type of service gets its own queue

Common Scheduling strategies

- “Real” fair queuing: bit-by-bit Round Robin
  - In packet-by-packet RR a 2B packet in one queue consumes 2x bandwidth of a 1B packet in another
  - Bit-level alternative: scheduler “trades” with each queue, waiting time for the num of bits transmitted (EDF in every service Round of the queues)
  - Deficit Round-Robin, an approx to bit-by-bit RR

Common Scheduling strategies

- Deficit Round-Robin (not providing delay bounds)



Network Logistics

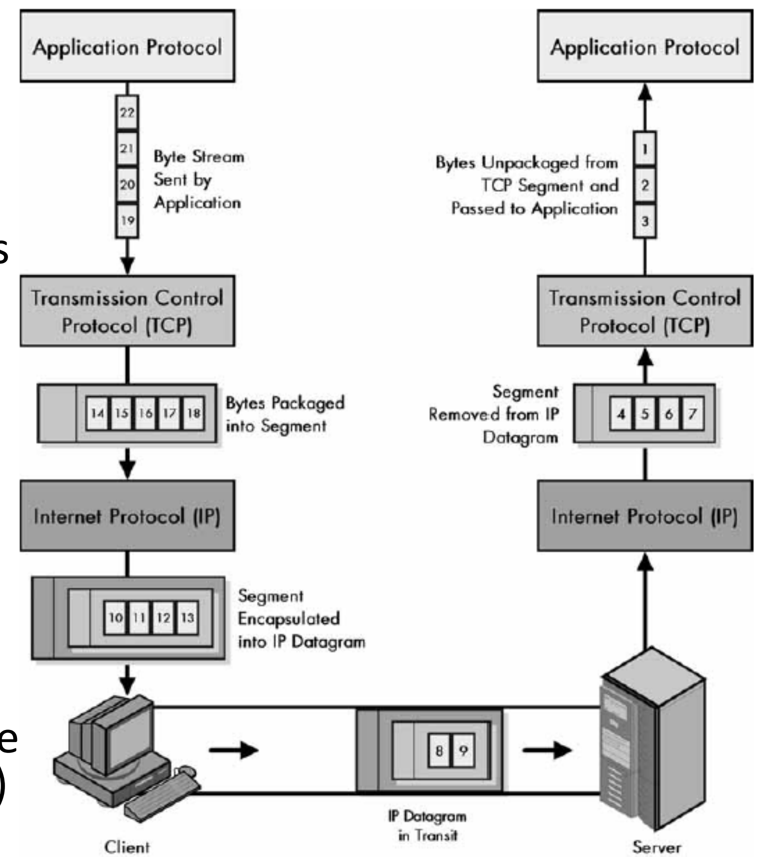
- In practice large networks can be very dynamic (number of users, service flows)
- A very fast or efficient scheduling approach can only help make better use of the medium capacity in face of competition, by applying rate control
- Can prolong, but not preempt or re-act to congestion collapse. Needs “cooperation” by the service flows
  - Congestion control
  - Congestion avoidance

Congestion Control

- Predominant examples of text book mechanisms for congestion control are those of TCP
- Probably the only protocol that has such level of sophistication ...and complexity
- A long time of incremental refinements and empirical testing to mature
- ...it all started when the whole Internet was at the verge of congestion collapse (mid 80s)

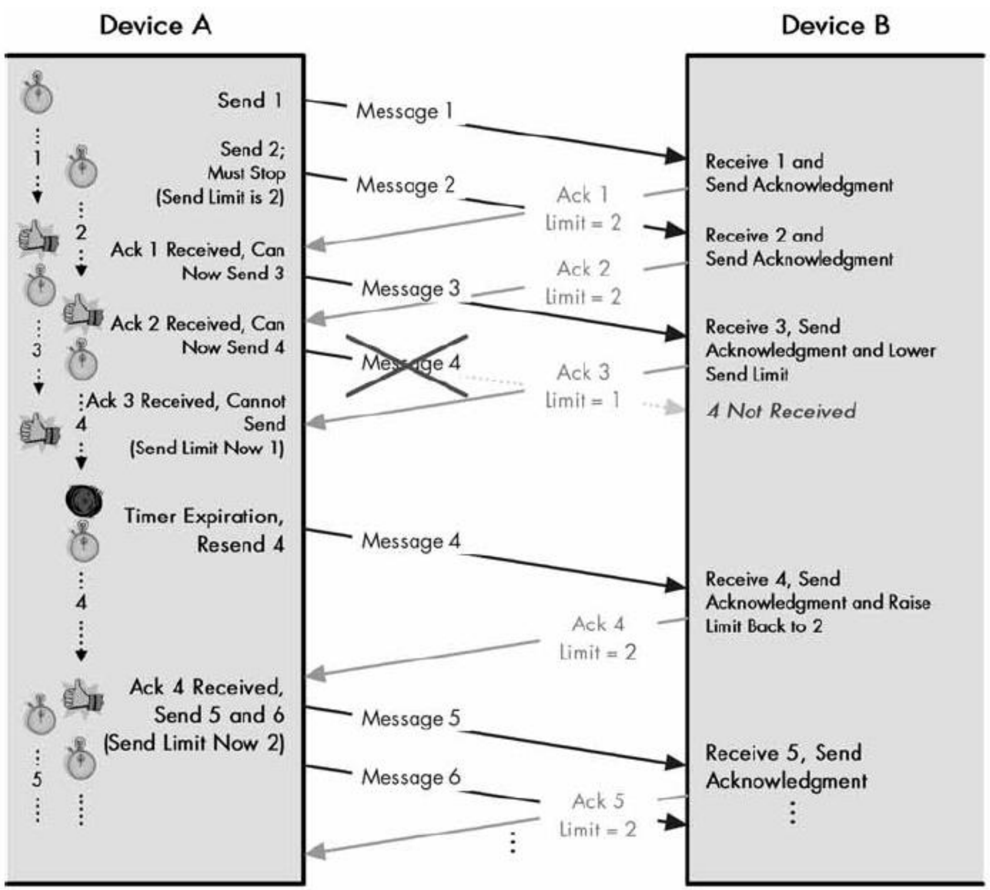
TCP brief overview: general

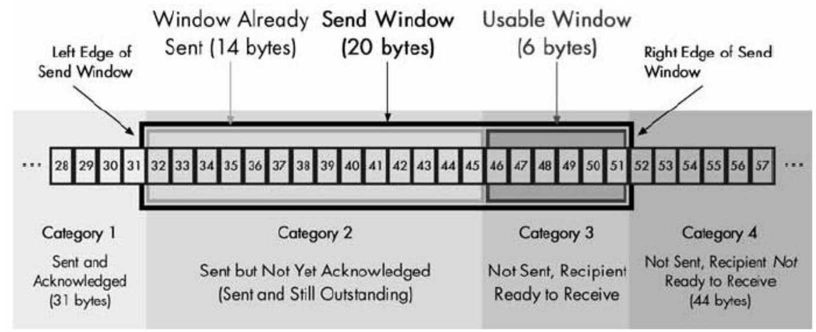
- TCP emulates connection-oriented communication over a packet switched network
  - Apps send data in continuous form (byte stream)
  - Synchronicity between communicating peers
- Continuous byte streams are divided in segments (of MSS size) which are sent <u>reliably</u> and <u>efficiently</u> over the network
  - Sequencing (like registered mail in Post)
  - ACK-feedback system (like recorded delivery in Post)
  - Transmission window system (like the postman delivery in Post)
  - Retransmission capability (copies kept in case messages get lost – more efficient than Post)

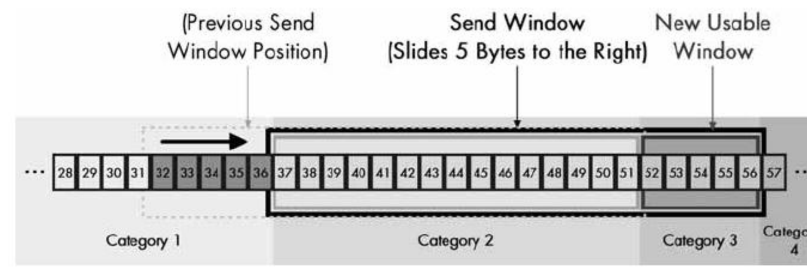


© Figure C. M. Kozierok, “The TCP/IP Guide”

TCP brief overview: flow control







© Figures C. M. Kozierok, "The TCP/IP Guide"

$\text{Delta} = RTT_{measure} - RTT_{estim}$

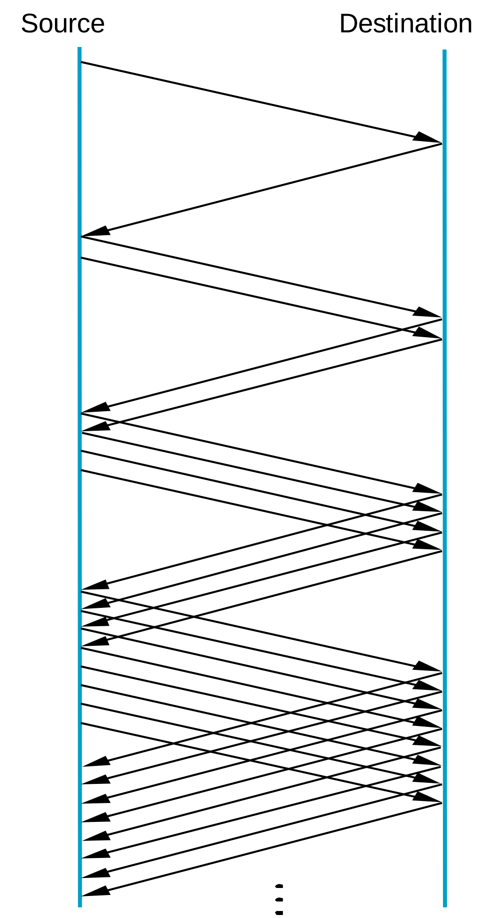
$RTT_{estim} = RTT_{measure} + \alpha * \text{Delta}$

$RTT\_Dev \mathrel{+}= \delta * (|\text{Delta}| - RTT\_Dev)$

$\text{Timeout} = RTT_{estim} + \mu * RTT\_Dev$

TCP congestion control 1: Slow Start

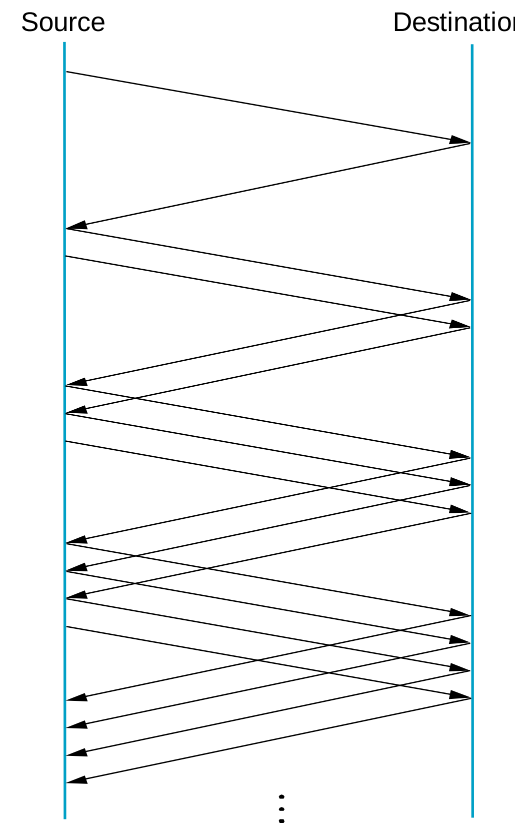
- *Congestion window* is used by the source to limit the number of packets in transit at a given time, up to the *Advertised window* size by the peer
- At cold start: exponential (2x) increase of congestion window after each ACK (every RTT), until a *Congestion threshold* is found (packet loss incurred)
- After packet loss Slow Start is re-initiated up to Congestion threshold (unless fast-recovery is used – see later)
- “Slow” because the congestion window builds up progressively as opposed to starting with the full *Advertised window* size
- Yet ... Quite aggressive! Can have up to 50% of the window packet drop (worst case)



© Figure L. Peterson, B. Davies, “Computer Networks”

TCP congestion control 2: Additive Increase / Multiplicative Decrease

- Additive: After Slow Start phase, increase Congestion window by 1, every time an entire batch of Congestion window worth messages has been ACKed
  - Slowly detecting bandwidth availability
- Multiplicative: Every time a timeout occurs, divide Congestion window by 2
  - Timeouts are indicators of congestion!
- Note that decrease happens much faster than increase
  - Key for congestion control mechanism's stability

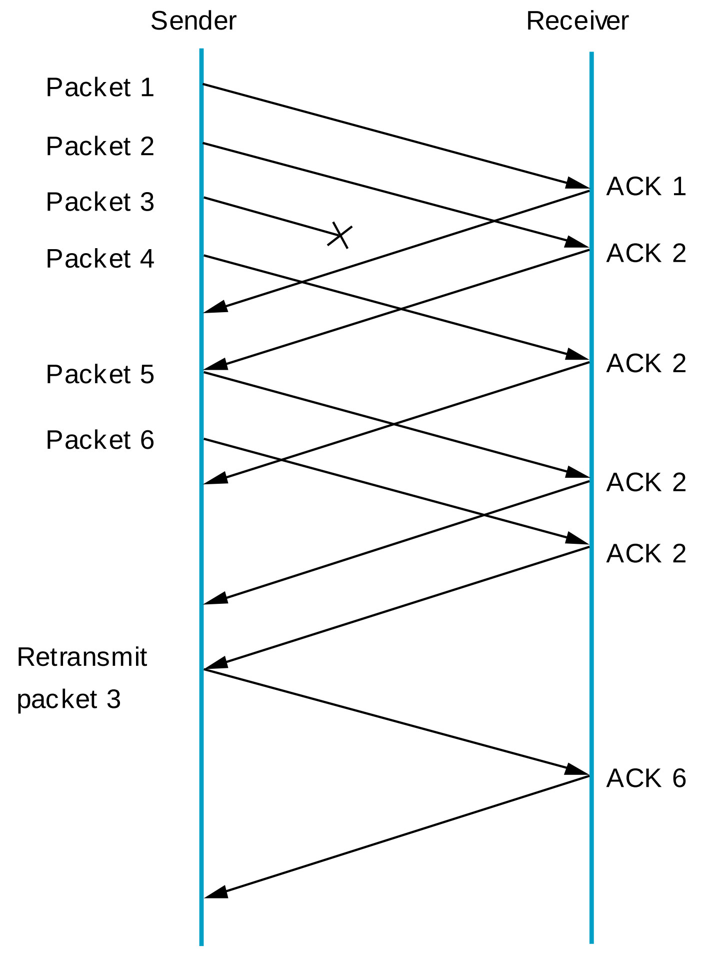


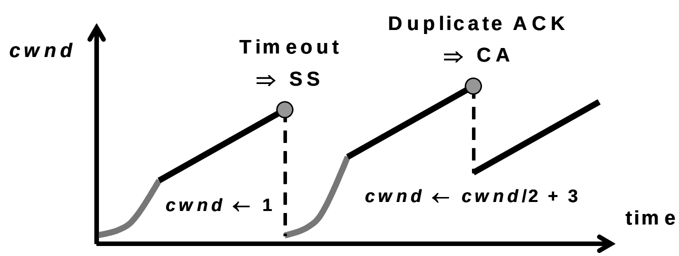
© Figure L. Peterson, B. Davies, "Computer Networks"

TCP congestion control 3: Fast retransmit with Fast recovery

- Avoids long idle periods after timeouts
- When a packet arrives at destination, it responds with an ACK.
- If the packet is received out of order the ACK is a duplicate of the last ACK
- When the sender sees 3 duplicate ACKs, it retransmits the assumed lost packet

- After fast retransmit the sender halves the Congestion window, and starts recovery using additive increase

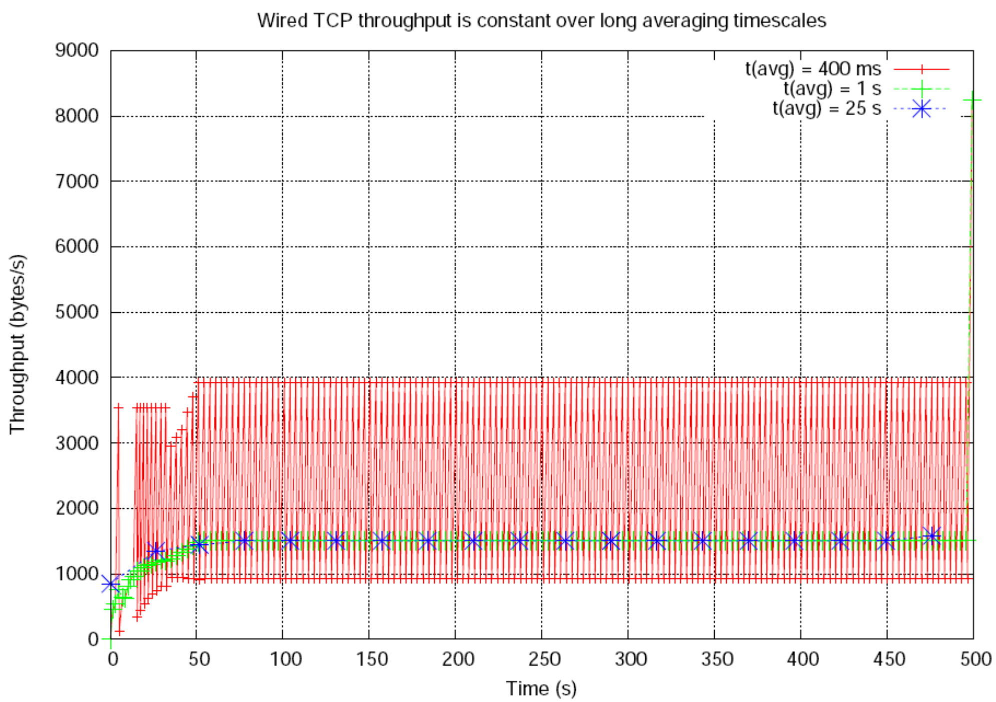




© Figure L. Peterson, B. Davies, “Computer Networks”

TCP performance with congestion control mechanisms

- TCP performance over long time scales looks a bit like this ... (quite stable)



© V. Raghunathan, P. R. Kumar, *"A Counterexample in Congestion Control of Wireless Networks"*

Assumptions that TCP makes

- *A network of wires, not wireless:*
  - Packet loss is the result of network congestion, rather than bit-level corruption
  - Stability in the RTT, because TCP uses a method of damping down the changes in the RTT estimate
- *A best-path route-selection:*
  - Single best metric path to any destination (all packets in a session follow the same path) such that packet reordering is an exception,
  - or else the order of packets within each flow is preserved by some network-level mechanism
- *A network with fixed bandwidth circuits, not varying bandwidth:*
  - At least no variation over short time intervals. End-to-end control loop based on RTTs to control the sending rate
  - Rapidly changing bandwidth force TCP to make very conservative assumptions about available network capacity

Assumptions that TCP makes

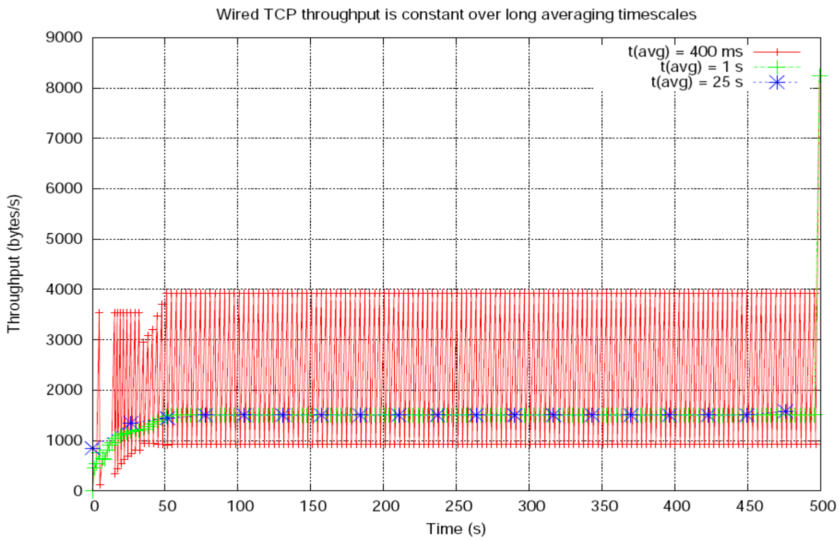
- *A switched network with first-in, first-out (FIFO) buffers:*
  - TCP assumes that the switching elements use simple FIFO queues to resolve contention within the switches
  - Work efficiently when the buffer of a network interface is of the order of the delay-bandwidth product of the associated link
- *Long duration of sessions:*
  - Expected to last for some number of round-trip times, so that the overhead of the initial protocol handshake is not detrimental to the efficiency of the application
  - Short sessions ("*TCP mice*") in transactional applications and short Web transfers impact the efficiency of TCP
- *Interaction with other TCP sessions:*
  - TCP is a *gentleman* assuming that other active TCP sessions should operate cooperatively to share available bandwidth in order to maximize network efficiency.
  - TCP may not interact well with other forms of flow-control protocols

Things are different in a wireless environment

- No strict notion of <u>link</u> since transmissions from near by nodes interfere with each other
  - Congestion is of spatial nature and cross interfering links all affect each other's congestion level in a <u>competitive or mutually exclusive</u> manner
  - Congestion feedback cannot be acquired by TCP's mechanism from neighborhood interfering "links"
- Bit errors occur very frequently rather than sporadically (high BER)
  - *Fast Retransmit* mechanism becomes victim or these errors with high probability, which in turn causes very often the collapse of the cwnd to the slow-start phase
  - L2 ARQ mechanisms retransmit link-level fragments to correct the data corruption, which may halt the packet flow for an entire link RTT interval
  - TCP cannot distinguish between random link-errors and congestion (assumes all packet loss is due to congestion)
  - The congestion control mechanism kicks-in even when there is no congestion causing the excessive congestion window to build up

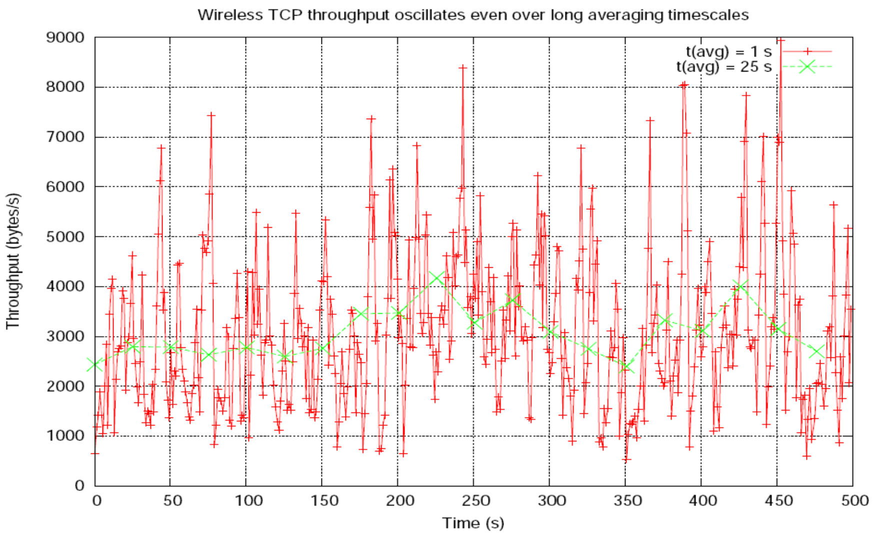
TCP performance in wireless

- TCP has no stable equilibrium point at the desired fair solution
- TCP need to take into account what is happening in other layers



Wired

→



Wireless

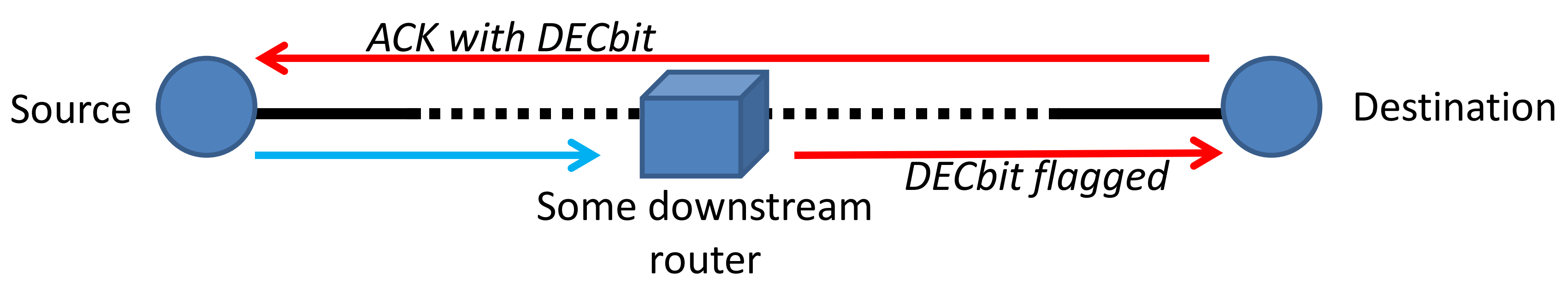
© V. Raghunathan, P. R. Kumar, *"A Counterexample in Congestion Control of Wireless Networks"*

Congestion Avoidance

- Congestion control is a response measure to the increase in network load. TCP aggressively creates congestion (timeouts) to discover its limit capacity
- Congestion avoidance is a set of solutions that try to pre-empt congestion and avoid it
  - DECbit
  - Random Early Detection (RED)
  - Traffic shaping
- Congestion avoidance mechanisms involve cooperation from queuing system (routers)

Congestion Avoidance: DECbit

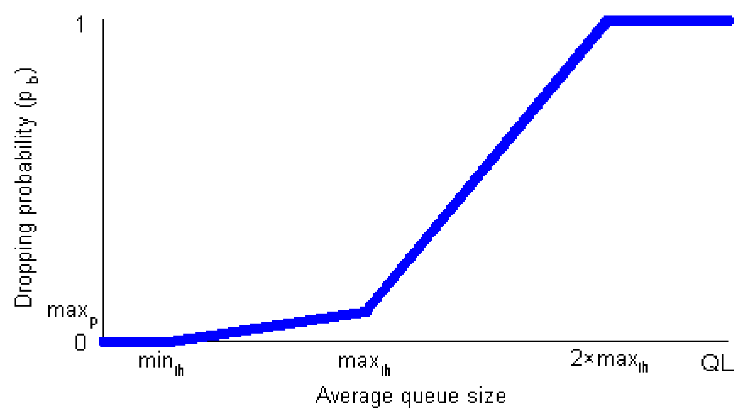
- Every router along a path monitors its queues' occupancy
- If the queue occupancy exceeds an average threshold over a time-interval when a packet arrives, it flags the DECbit in the packet downstream
- When the destination sees the DECbit flagged, it copies it to the ACKs sent back to the source
- The source calculates how many packets in the last congestion window resulted in flaged DECbit and if more then 50% reduces its Congestion window with a multiplicative decrease policy



Congestion avoidance: RED

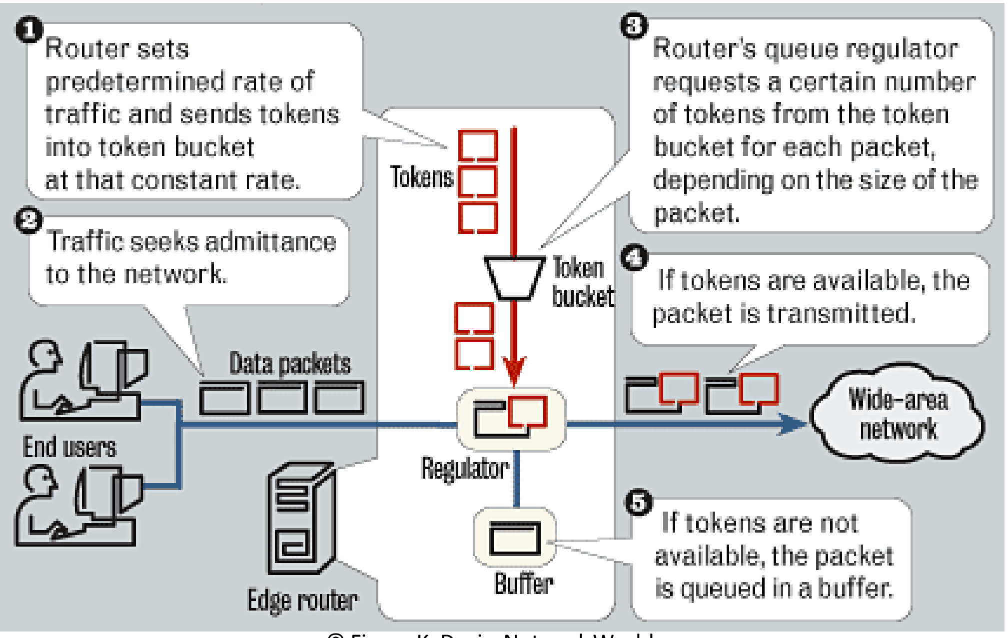
- Similar to DECbit, yet more sophisticated (or subtle) to operate with TCP!
- Instead of flagging a bit, the router that detects likely congestion, drops a packet with a certain probability
- This puts the respective TCP flow in "fast-retransmit with fast-recovery" mode (i.e. halve its Congestion window) and therefore reduce the flow rate
- Drop probability is calculated based on Avg queue length
  - Becomes more aggressive based on load

$$QLen_{avg} = (1-\alpha)\, QLen_{avg} + \alpha * QLen_{measure}$$



Congestion avoidance through traffic shaping: Token Bucket Policing

- Usually performed at the ingress routers in the network
- Smoothen or control the rate of flows in face of burstiness or lack of congestion control mechanisms at the protocol level (UDP)



© Figure K. Davis, Network World

Questions ?