Computer Networks (part 3)

Rémi Emonet – 2021
Université Jean Monnet – Laboratoire Hubert Curien

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Computer Networks: global overview

1. Introduction to computer networks
2. Networking application layer (HTTP, FTP, DNS, ...)
3. Data transfer layer (UDP, TCP, ...)
4. Network layer (routing, IP, ICMP, NAT, ...)
5. Lower layers, wireless and mobile (Ethernet, ARP, ...)
6. Security (SSL, ...)

Computer Networks 3: Plan

- Goal: transport layer
  - understand its main roles and mechanisms
  - understand the how TCP (and UDP) are implemented
- Overview
  - Transport layer: context and services (role)
  - Multiplexing and demultiplexing
  - UDP
  - Reliable communications, please!
  - Pipelining: principle and algos
  - Implementation of TCP
  - Online timeout estimation
  - Congestion: principle and algos
  - TCP: optimality? equity?

application
transport
network
link
physical

Computer Networks 3: Plan

- Goal: transport layer
  - understand its main roles and mechanisms
  - understand the how TCP (and UDP) are implemented
- Overview
  - Transport layer: context and services (role)
  - Multiplexing and demultiplexing
  - UDP
  - Reliable communications, please!
  - Pipelining: principle and algos
  - Implementation of TCP
  - Online timeout estimation
  - Congestion: principle and algos
  - TCP: optimality? equity?

application
transport
network
link
physical

Transport Layer: an abstraction on top of the network

- Process to process comm.
  - for the applications
- Role: source
  - gets messages from the app.
  - split it in *segments*
  - sends segments using the "network" layer
- Role: destination
  - receives segments
  - re-assemble them in a message
  - transmits the message to the application
- Different protocols
  - UDP: minimal, packets
  - TCP: connection, streams

Distinction between transport and network layers

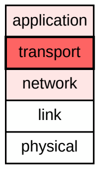
- Transport layer
  - communication (logical) between processes
  - built on the network layer
- Network layer
  - communication between hosts
  - still a logical communication
- Analogies: sending (snail) mail
  - situation: exchanges between 2 schools
    - 10 young children in school A, 10 in school B
    - teachers handle the mails
  - host == school
  - process == child
  - application message == letter
  - transport layer == teachers(ses)
  - network layer == post service

Services of the transport layer (in Internet)

- TCP protocol
  - in-order and reliable transmission
  - flow and congestion control
  - connections establishment
- UDP protocol
  - unreliable, un-ordered
  - minimal transport layer over sur IP
- Missing services in both TCP and UDP
  - bandwidth guarantees
  - latency guarantees (delay)
  - security

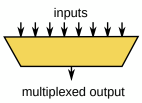
Computer Networks 3: Plan

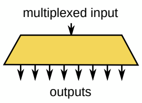
- Goal: transport layer
  - understand its main roles and mechanisms
  - understand the how TCP (and UDP) are implemented
- Overview
  - Transport layer: context and services (role)
  - Multiplexing and demultiplexing
  - UDP
  - Reliable communications, please!
  - Pipelining: principle and algos
  - Implementation of TCP
  - Online timeout estimation
  - Congestion: principle and algos
  - TCP: optimality? equity?

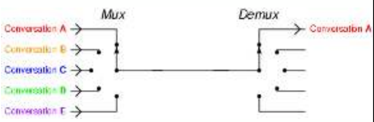


Multiplexing and Demultiplexing

- General principle (electronics)
  - have multiple signals in a sigle wire
  - generally, in sequence (time)
- Current case
  - have multiple …
  - … communications between processes (transport layer)
  - … inside a single host communication (network layer)
- multiplexing/demultiplexing

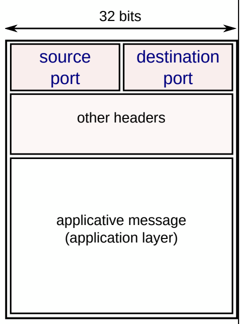






Multiplexing with TCP and UDP: ports

- Network layer
  - messages: IP datagram
  - IP addresses
  - source (host)
  - destination (host)
  - each datagram contains a segment
- Transport layer
  - message: TCP/UDP segment
  - ports (+ IP)
  - redirects a segment to the proper socket



TCP or UDP Segment

What is the biggest port number?

- Ports are numbered from 1 to
- How much is ?

What could identify a socket in UDP? (for demultiplexing)

- Demultiplexing
  - the transport layer
  - … redirects segments
  - … to the proper sockets

Demultiplexing for Connection-less Situations (UDP)

- Used information
  - destination IP address (network)
  - destination port (UDP)
- When a host receives a UDP segment
  - it reads the destination port
  - it redirect the segment to the UDP socket having this number
- Consequence
  - segments having the same destination port
  - … are sent to the same socket
  - … whatever the source
  - (cf Java program)
- UDP == demultiplexing based on local port number

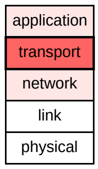
What could identify a socket in TCP? (for demultiplexing)

Demultiplexing with Connections (TCP)

- Used information
  - destination IP address (network)
  - source IP address, source port and destination port (TCP)
- When a host receives a TCP segment
  - it reads the destination port
  - il reads the source IP address
  - il reads the source port
  - it redirect the segment to the corresponding TCP socket
- Consequence
  - each client (source) is distinguishable
  - one socket per client
  - demultiplexing provided by TCP
- TCP == demultiplexing based on the client (local-port / remote-port / remote-IP)

Computer Networks 3: Plan

- Goal: transport layer
  - understand its main roles and mechanisms
  - understand the how TCP (and UDP) are implemented
- Overview
  - Transport layer: context and services (role)
  - Multiplexing and demultiplexing
  - UDP
  - Reliable communications, please!
  - Pipelining: principle and algos
  - Implementation of TCP
  - Online timeout estimation
  - Congestion: principle and algos
  - TCP: optimality? equity?

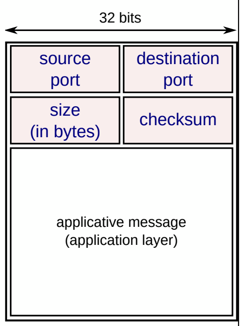


UDP: User Datagram Protocol [RFC 768]

- Minimal transport protocol
  - best-effort protocol
    - packets can be lost
    - packets can be received in a different order
  - datagram == message from the network layer
  - adds error detection
- Connection-less
  - no hand-shaking (no initialization overhead)
  - independents packets
- Reliable transfer with UDP?
  - to implement yourself
  - … by the applications
- Advantages of UDP
  - simple, lower latency (no connection), smaller packets
  - used for: DNS, SNMP (routers), streaming

UDP: header format for UDP segments

- Total: 8 bytes
- Source port (16 bits) and destination port (16 bits)
- Size
  - total size of the segment
  - including the header
- Checksum
  - for error detection (altered bits, …)
  - the emitter
    - computes the checksum
    - inserts the header
  - the receiver
    - extracts the checksum value from the segment
    - validates that it agrees with the rest of the segment



UDP Segment

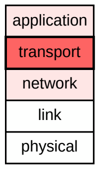
Computing a Checksum

```
value 1:     1  1  1  0  0  1  1  0  0  1  1  0  0  1  1  0
value 2:     1  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1

    sum:  1  1  0  1  1  1  0  1  1  1  0  1  1  1  0  1  1

             1  0  1  1  1  0  1  1  1  0  1  1  1  1  0  0

checksum:    0  1  0  0  0  1  0  0  0  1  0  0  0  0  1  1
```

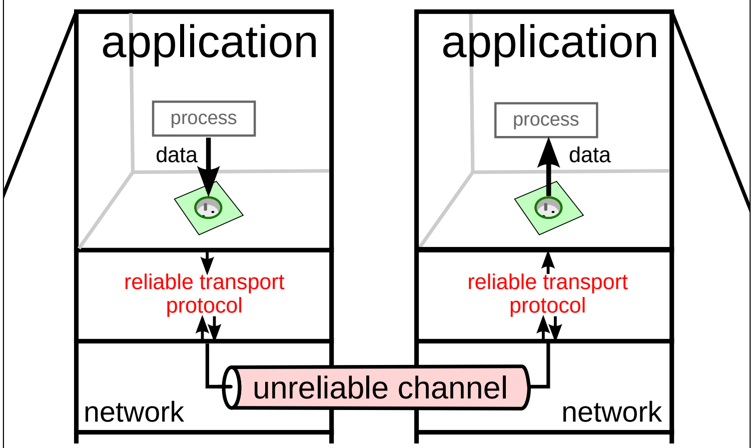
- View the data as a set of 16-bit numbers
- Successive sums
- Circular carry (fr: *retenue*) propagation
- One's complement sum (bits inversion)
- UDP: computed on datagram + pseudo-header (IPs+length+…)

Computer Networks 3: Plan

- Goal: transport layer
  - understand its main roles and mechanisms
  - understand the how TCP (and UDP) are implemented
- Overview
  - Transport layer: context and services (role)
  - Multiplexing and demultiplexing
  - UDP
  - Reliable communications, please!
  - Pipelining: principle and algos
  - Implementation of TCP
  - Online timeout estimation
  - Congestion: principle and algos
  - TCP: optimality? equity?



Reliable Communication: principle



For a reliable communication what do we need?

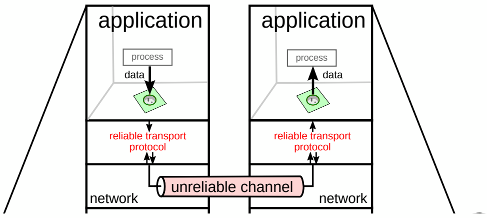
Key Points for a Reliable Communication

- Error detection: checksum
- Packet loss detection: acknowledgment, ACK
- Loss compensation: timeout, resending after a delay
- Double reception detection: sequence number
- Error signaling? negative ACK?

In case of a detected error (wrong checksum) in a message (or ACK), what should the protocol do?

Protocol Description: interfaces

- Interface
  - emitter
    - appSend(data): called by the application layer
    - netSend(pckt): call to the network layer
    - netReceive(pckt): callback from the network layer
  - receiver
    - appDeliver(data): call to the application layer
    - netReceive(pckt): callback from the network layer
    - netSend(pckt): call to the network layer



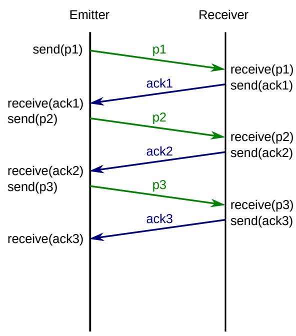
Protocol Description: emitter side

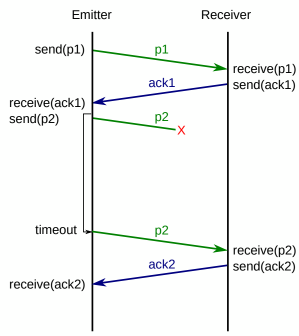
- State 1 (initial)
  - when `appSend(data)`
    - prepare `pckt` (header, checksum, number etc)
    - store the number as `expected`
    - `netSend(pckt)`
    - starts a *timer*
    - switch to State 2
- State 2 (waiting ACK)
  - when `appSend(data)`, buffer it or reject it
  - when `netReceive(pckt)`
    - if `correct(pckt) ∧ isAck(pckt, expected)`
      - stop the timer
      - switch to State 1
    - else: nothing
  - on timeout (timer event)
    - resend `pckt`
    - restart a timer

Protocol Description: receiver side

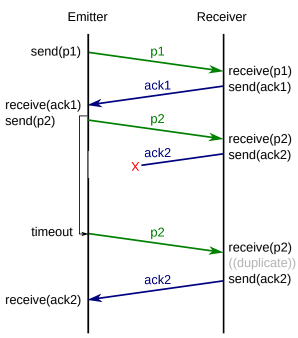
- State 1 (initial, waiting for packet `next`)
  - when `netReceive(pckt)`
    - if `correct(pckt) ∧ isNum(pckt, next)`
      - `appDeliver(getData(pckt))`
      - create `ackPckt` with the number and checksum
      - `netSend(ackPckt)`
      - `next++`
    - else
      - `netSend(ackPckt)` (the previous ACK)

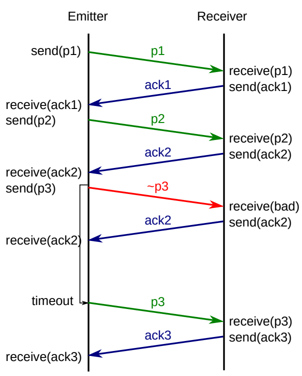
Reliable Protocols: examples



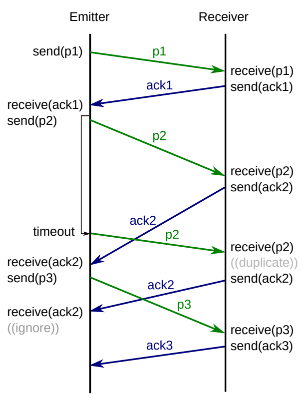


Reliable Protocols: examples

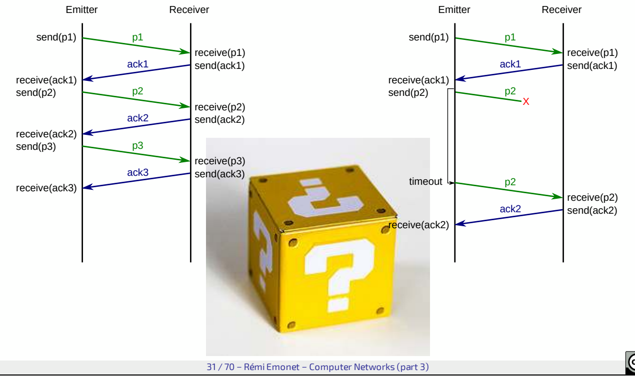




Reliable Protocols: examples



Can we do better in terms of network usage?



How to better use the network when sending multiple packets and acks?

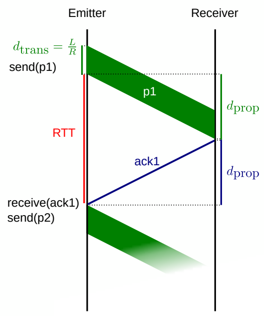
Computer Networks 3: Plan

- Goal: transport layer
  - understand its main roles and mechanisms
  - understand the how TCP (and UDP) are implemented
- Overview
  - Transport layer: context and services (role)
  - Multiplexing and demultiplexing
  - UDP
  - Reliable communications, please!
  - Pipelining: principle and algos
  - Implementation of TCP
  - Online timeout estimation
  - Congestion: principle and algos
  - TCP: optimality? equity?

application
transport
network
link
physical

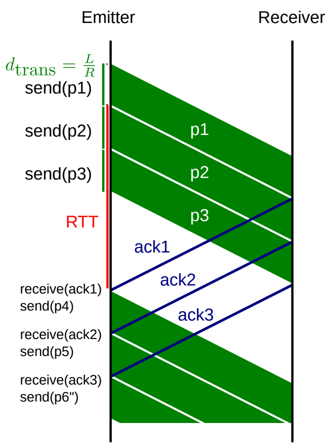
Pipelining Motivation

- Case study
  - 1 Gbps link
  - 15 ms propagation
  - 10 kbits packet
- Computations
  - $d_{\text{trans}} = \frac{L}{R} = \frac{10 \text{ kbits}}{1 \text{ Gb s}^{-1}} = 10\mu s$
  - $d_{\text{prop}} = 15ms$
  - Link usage: $U = \frac{d_{\text{trans}}}{d_{\text{trans}} + RTT}$



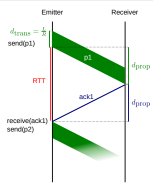
$$U = \frac{d_{\text{trans}}}{d_{\text{trans}} + 2 \cdot d_{\text{prop}}} = \frac{10}{30010} = 0.000333222$$

Pipelining Usefulness



Pipelining Usefulness

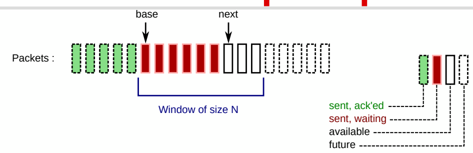
- Ex:
  - 1 Gbps, $t_{\text{prop}} = 15ms$, 10 kbits packets
- Without pipelining
  - $U = \frac{d_{\text{trans}}}{d_{\text{trans}} + RTT} = 0.00033$
- With 3-packet pipelining
  - $U = \frac{3\ d_{\text{trans}}}{d_{\text{trans}} + RTT}$
  - $U = \frac{30}{30010} = 0.000997$
  - link use is 3 times bigger
  - NB: diagram is not exact



Pipelining: two principal approaches

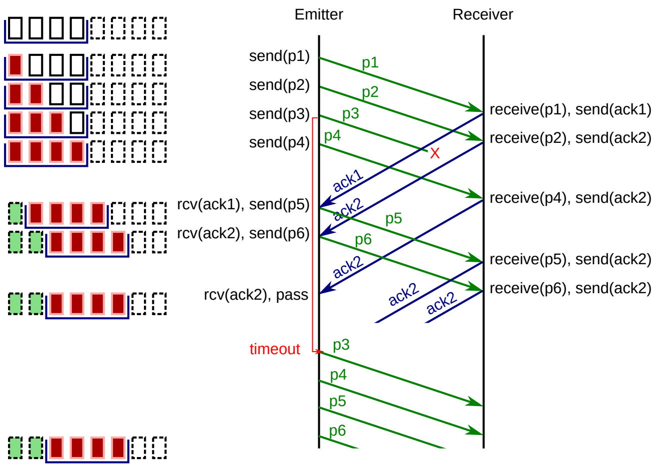
- Go-back-N
  - the emitter
    - can have N packets in the pipeline
  - the receiver
    - sends cumulative acks only
    - do not send the ack if a packet is missing
  - the emitter
    - has a time based on the oldest packet
    - re-sends all packets in case of timeout
- Selective Repeat
  - the emitter can have N packets in the pipeline
  - the receiver acknowledges each packet separately
  - the emitter keeps a timer for each packet

Go-back-N: principle

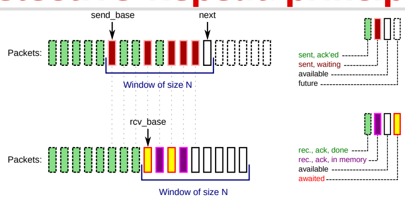


- Emitter
  - allows up to N consecutive non-ack'd packets
  - `ack(i)` means: all packets up to `i` arrived
  - timer
    - based on the oldest packet (base)
    - on timeout: sends all packets starting from base
- Receiver
  - `expect`: number of the next expected packet
  - drops all packets except the number `expect` (no buffering)
  - always a ack for the last packet properly received (`expect-1`)

Go-back-N: example

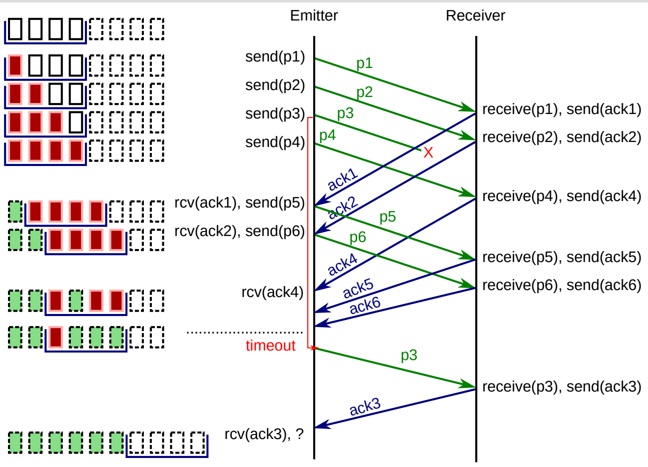


Selective-Repeat: principle



- Emitter
  - window of size N
  - `ack(i)` means: packet `i` arrived
  - a timer, for each packet (independently)
    - on timeout: re-sends the corresponding packet
- Receiver
  - it's own window of size N
  - ack for each packet (even if some are missing before)
  - buffering of packets that arrived out-of-order

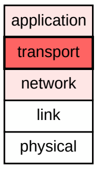
Selective Repeat: example



Can we do the same without ack?

Computer Networks 3: Plan

- Goal: transport layer
  - understand its main roles and mechanisms
  - understand the how TCP (and UDP) are implemented
- Overview
  - Transport layer: context and services (role)
  - Multiplexing and demultiplexing
  - UDP
  - Reliable communications, please!
  - Pipelining: principle and algos
  - Implementation of TCP
  - Online timeout estimation
  - Congestion: principle and algos
  - TCP: optimality? equity?



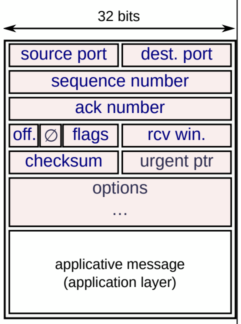
TCP: Transmission Control Protocol

[RFC 793,1122,1323, 2018, 2581]

- TCP communication
  - between two processes
  - reliable, ordered
  - stream-based: no separation between messages
  - bi-directional
  - connection oriented, stateful, handshake at the beginning
  - pipelined (with a variable window size)
  - with flow control

TCP: formats of segment headers

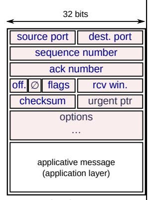
- Total size: variable (20 bytes minimum)
- Source and dest. ports (16 bits each)
- Sequence and ack numbers: **in bytes**
- *offset*: size of the header (cf. options)
- *flags*, booleans, including
  - ACK: this segment contains a ACK
  - SYN: connection initialization (exchange of the initial sequence number)
  - FIN: end of connection
- *receive window*: size in bytes that the receiver wants
- *checksum* as in UDP



TCP Segment

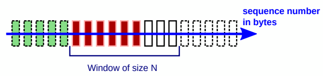
TCP: sequence and ack numbers

- Reminder: TCP handles a stream of bytes
- Sequence numbers
  - index in the stream of the first byte of the segment
- Acknowledgment (ack)
  - cumulative acknowledgment (as in go-back-N)
  - stream index of the next byte to be received
  - ACK flag is set to 1

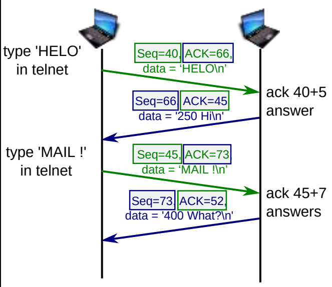


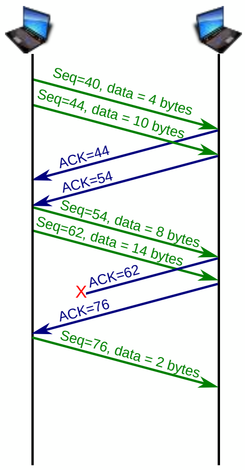
TCP Segment

- go-back-N or selective repeat?
  - cumulative acknowledgment
  - TCP, by default, does not specify what to do with out-of-order packets
  - RFC2018: option for using *selective repeat* (SACK)



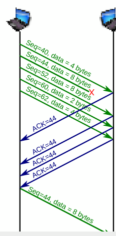
TCP: examples



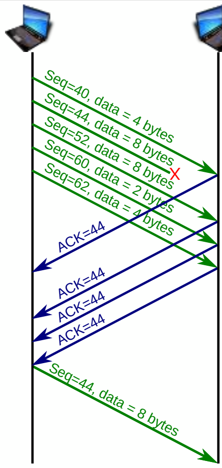


TCP: multiples acknowledgment

- TCP acknowledgment (reminder)
  - ack with the index of the next expected byte
  - out-of-order packets ⇒ double ack
- Using double acks for fast re-transmission
  - the timeout is generally long
  - if we receive multiple double acks, there was probably a loss
  - TCP strategy
    - on the 3rd double ack (4th identical)
    - send the packet again

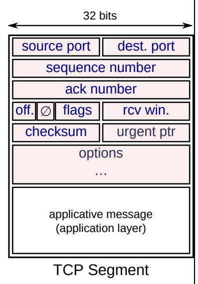


TCP Fast Retransmit



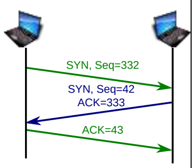
TCP Flow Control

- Goal: do not flood the receiver
- TCP has a receive buffer
  - typical size: 4096 kB
  - the system can adapt it dynamically
- rcv win
  - send in the tcp header of every segment
  - amount of available space in the buffer



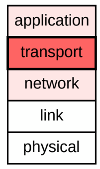
TCP Connection Opening

- Client
  - generation of a sequence number
  - emission of a SYN packet
- Server
  - generation of a sequence number
  - emission of a SYNACK packet (ACK + SYN)
- Client
  - emission of a ACK packet
  - can also start sending data in this packet



Computer Networks 3: Plan

- Goal: transport layer
  - understand its main roles and mechanisms
  - understand the how TCP (and UDP) are implemented
- Overview
  - Transport layer: context and services (role)
  - Multiplexing and demultiplexing
  - UDP
  - Reliable communications, please!
  - Pipelining: principle and algos
  - Implementation of TCP
  - Online timeout estimation
  - Congestion: principle and algos
  - TCP: optimality? equity?

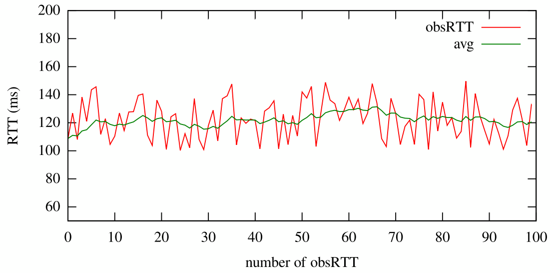


What should be the timeout in TCP? (before retransmitting a non-ACK'd packet)

Timeout and Round Trip Time

- Choice of the timeout
  - longer than RTT
  - but RTT varies
  - if too short: useless retransmissions
  - if too long: long delay on loss
- Estimation of RTT
  - obsRTT_i: time between sending packet i and receiving its ACK
  - obsRTT_i varies and is unstable ⇒ moving average

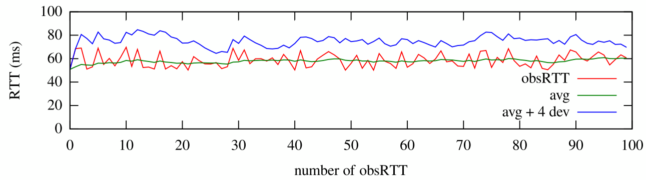
Timeout in TCP: RTT estimation



- At each new ACK (obsRTT_i)
  - $\text{avg} = (1-\alpha)\ \text{avg} + \alpha\ \text{obsRTT}_i$
  - moving/rolling/running average
  - exponential weighting
  - value for α : 0.125

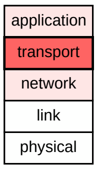
Timeout Computation in TCP

- Estimation of RTT
  - $\text{avg} = (1-\alpha)\ \text{avg} + \alpha\ \text{obsRTT}_i$
- Estimation of the mean deviation
  - $\text{dev} = (1-\beta)\ \text{dev} + \beta\ \ |\text{obsRTT}_i - \text{avg}|$
- Transmission delay $= \text{avg} + 4 \cdot \text{dev}$
- Values: $\alpha = 0.125 \quad \beta = 0.25$

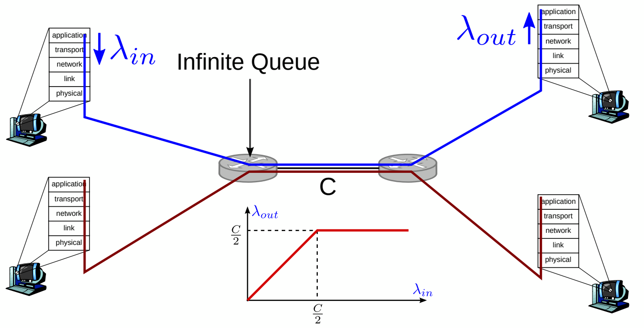


Computer Networks 3: Plan

- Goal: transport layer
  - understand its main roles and mechanisms
  - understand the how TCP (and UDP) are implemented
- Overview
  - Transport layer: context and services (role)
  - Multiplexing and demultiplexing
  - UDP
  - Reliable communications, please!
  - Pipelining: principle and algos
  - Implementation of TCP
  - Online timeout estimation
  - Congestion: principle and algos
  - TCP: optimality? equity?

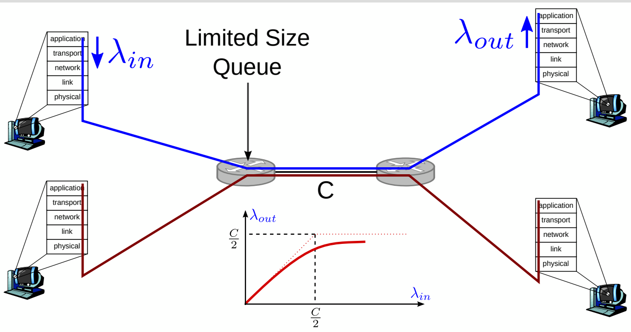


Congestion: principles (infinite queue)



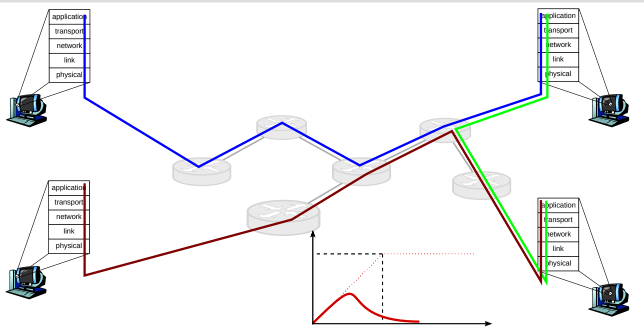
- Ideal case of an infinite queue
- Maximal bandwidth
- Unreasonable delays (time spend in the queue)

Congestion: principles



- More realistic case with limited buffers and packet loss
- Retransmitting packets due to timeout (loss, delay)
- Reduced bandwidth due to retransmissions

Congestion: in a network



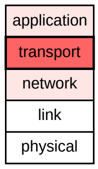
- Augmenting the rate of a connection can penalize the rest
- When a router drops packets,
  all the bandwidth used to bring the packet there is wasted

Congestion Control/avoidance: two kinds of approaches

- Congestion-control assisted by the network
  - smart routers
  - router → host messages on congestion level
  - the network tells the host what bandwidth to use
  - disadvantages
    - expensive routers
    - difficult to get robustness
- Congestion-control at a host level
  - the network is a black box
  - based on observed delays and losses
  - used by TCP

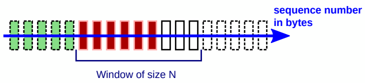
Computer Networks 3: Plan

- Goal: transport layer
  - understand its main roles and mechanisms
  - understand the how TCP (and UDP) are implemented
- Overview
  - Transport layer: context and services (role)
  - Multiplexing and demultiplexing
  - UDP
  - Reliable communications, please!
  - Pipelining: principle and algos
  - Implementation of TCP
  - Online timeout estimation
  - Congestion: principle and algos
  - TCP: optimality? equity?



TCP Congestion Control (+reminders)

- Sequence number in bytes



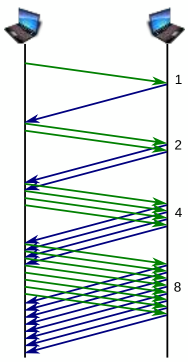
- Sliding window
  - limitation on the sender side
  - (last byte sent - last byte ACK'd) ≤ N
  - N is also denoted $cwnd$ (Congestion Window)
- TCP transmission rate
  - approximately: $\frac{cwnd}{RTT}$
- Congestion control in TCP
  - dynamic adaptation of $cwnd$
  - as a function of the observed delays and losses
  - different possible algorithms (still evolving)

TCP Congestion Control: principles

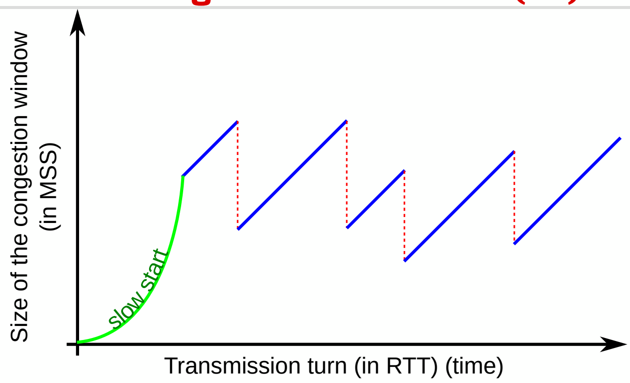
- Given MSS: maximum TCP segment size
- Increases the window size (emission rate)
  - goal: use at best the available bandwidth
  - additive increase ($cwnd = cwnd + MSS$)
  - reacts in case of packet loss
- In case of loss
  - diminishes the window size
  - multiplicative decrease ($cwnd = 0.5 \times cwnd$)
- Concept of “slow start”
  - initial phase
  - goal: reach/find as fast as possible the bandwidth limit
  - multiplicative increase at the beginning ($cwnd = 2 \times cwnd$)

TCP Slow Start

- Exponential increase of the window size
  - initialization: $cwnd = MSS$
  - MSS: maximum segment size
  - $cwnd = 2 \times cwnd$ at each RTT
  - $cwnd = cwnd + MSS$ at each ACK
- Slow start
  - starts with a low rate
  - increases the rate exponentially
  - end of the slow start phase
    - in case of loss
    - or, when a threshold is reached: $cwnd \geq ssthres$



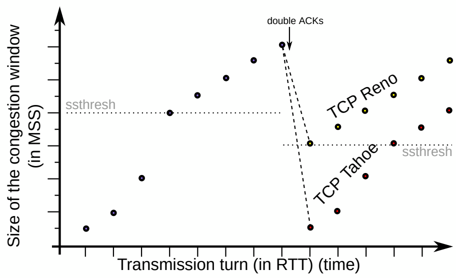
TCP Congestion Avoidance (CA)



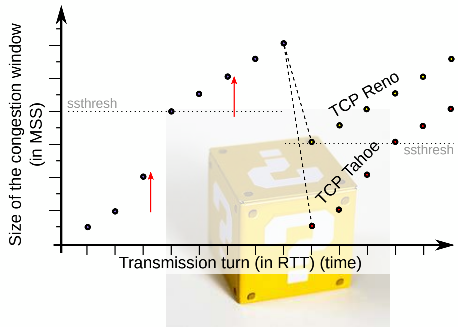
- Additive increase
- Multiplicative decrease

Loss Detection+Compensation (TCP Reno)

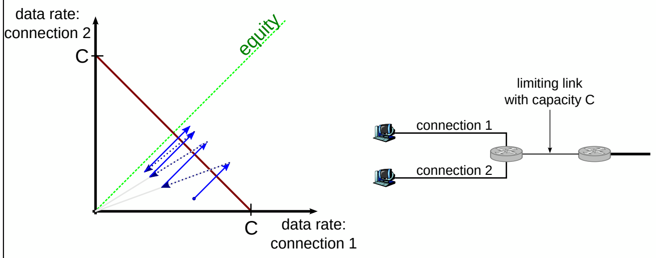
- Timeout expiration
  - re-initialization: $cwnd = MSS$; then slow start again
- Triple double-ACK (4 times the same ACK)
  - $cwnd = 0.5 \times cwnd$; then linear increase
  - TCP Tahoe (older): re-initializing cwnd



Number of packets emitted… from the start to the 1st red arrow? up to the 2nd red arrow?



Optimality and Equity of TCP



(In)Equity of TCP

- UDP has no congestion control
  - sends packets, interpolation/correction in case of packet loss
  - no emission reduction in case of congestion
  - some routers are blocking UDP?
- Multiple TCP connections
  - TCP provides connection-equity
  - opening of multiple connections
    - common for web browsers, etc
  - example, if there are already 4 connections
    - the new application opens 1 connection ⇒ effective rate of $\frac{R}{5}$
    - the new application opens 4 connections ⇒ effective rate of $\frac{R}{2}$
  - NB: routers can still do IP based drop