Transmission Control Protocol

Advanced Issues

Petr Grygárek

Motivation

- TCP is prevalent protocol for today's client-server internet non-realtime application
- TCP implements mechanism to utilize network infrastructure efficiently
  - Retransmission policy for various media may be conflicting
    - Lost ACKs may indicate congestion on reliable links, so transmitter should back-off
    - Lost ACKs may indicate errored transmission on unreliable (radio) links, so transmitter should repeat transmission immediately
- QoS enforcement mechanisms implemented in network infrastructure utilize knowledge of TCP behavior to control dataflow generated by TCP senders
  - manipulating TCP feedback (ACK delaying/discard)

TCP Implementation

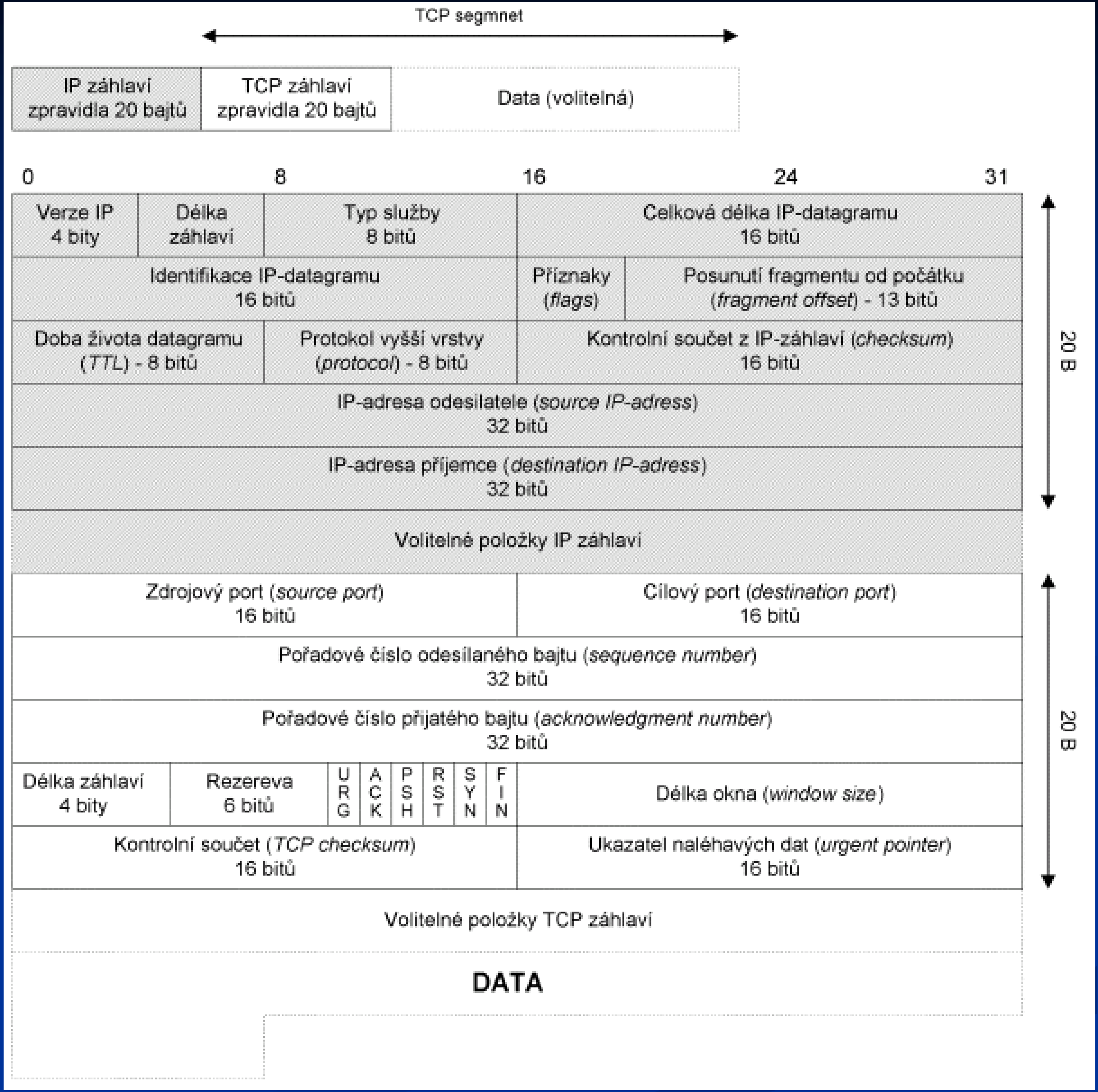
- Basic behavior and header structures are given
- During it’s history, TCP has been thoroughly tested
  - many special cases were discovered and solution proposed
  - TCP is now considered a reliable, well-tested protocols with good implementations
- TCP supports many optional mechanism described in multiple separate RFCs
- Every implementation is interoperable with others, but some combinations man be less effective

Quick Reminder of TCP

Basic properties of TCP

- Provides bidirectional reliable data stream between two applications
- Particular application identified by port number (16b)
  - (in scope of individual IP address)
- TCP connection identified by 4-tuple
  - < local IP addr, local port, remote IP addr, remote port >
- Data stream chopped into segments
  - Max segment length given by Path MTU (- IP header length)
- TCP performs retransmission of lost segments, resequencing of segment into correct order and descarding of duplicated segments

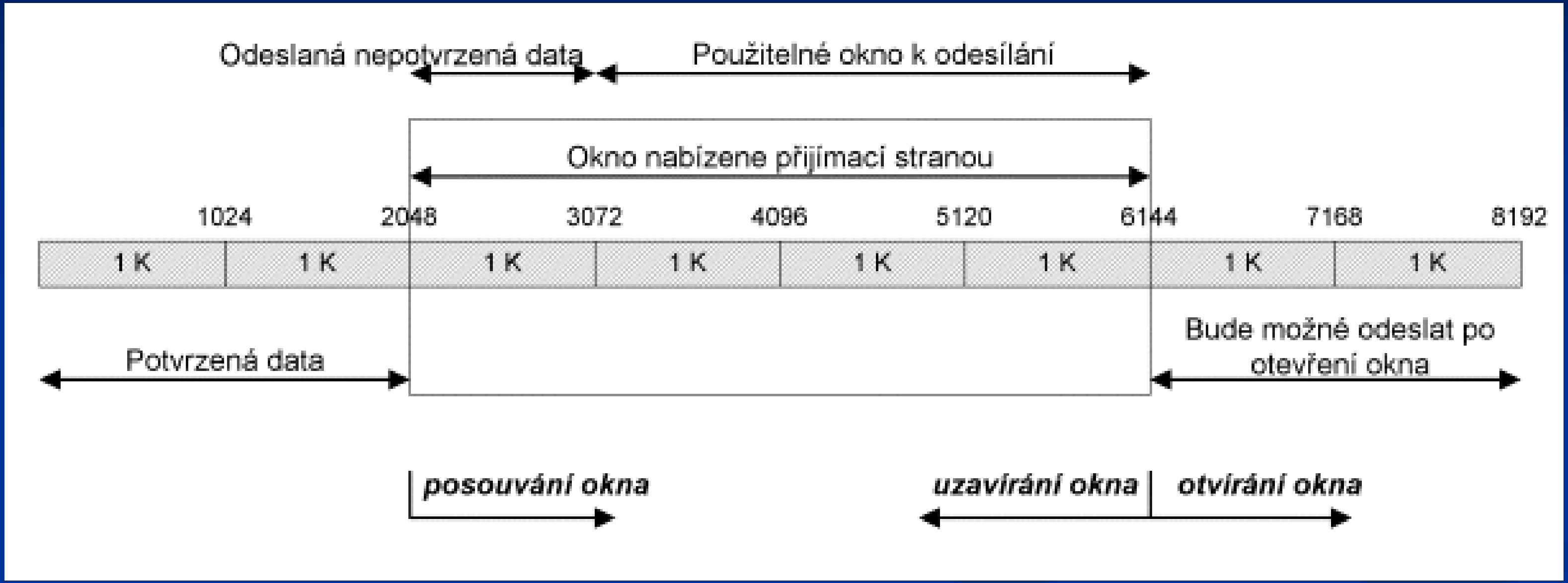
TCP Header



TCP segmnet
IP záhlaví zpravidla 20 bajtů
TCP záhlaví zpravidla 20 bajtů
Data (volitelná)
0
8
16
24
31
Verze IP 4 bity
Délka záhlaví
Typ služby 8 bitů
Celková délka IP-datagramu 16 bitů
Identifikace IP-datagramu 16 bitů
Příznaky (*flags*)
Posunutí fragmentu od počátku (*fragment offset*) - 13 bitů
Doba života datagramu (*TTL*) - 8 bitů
Protokol vyšší vrstvy (*protocol*) - 8 bitů
Kontrolní součet z IP-záhlaví (*checksum*) 16 bitů
IP-adresa odesilatele (*source IP-adress*) 32 bitů
IP-adresa příjemce (*destination IP-adress*) 32 bitů
Volitelné položky IP záhlaví
20 B
Zdrojový port (*source port*) 16 bitů
Cílový port (*destination port*) 16 bitů
Pořadové číslo odesílaného bajtu (*sequence number*) 32 bitů
Pořadové číslo přijatého bajtu (*acknowledgment number*) 32 bitů
Délka záhlaví 4 bity
Rezereva 6 bitů
URG
ACK
PSH
RST
SYN
FIN
Délka okna (*window size*)
Kontrolní součet (*TCP checksum*) 16 bitů
Ukazatel naléhavých dat (*urgent pointer*) 16 bitů
20 B
Volitelné položky TCP záhlaví
DATA

Picture from: Dostálek, Kabelová: Velký průvodce TCP/IP a systémem DNS

Sliding Window


Odeslaná nepotvrzená data
Použitelné okno k odesílání
Okno nabízene přijímací stranou
1024
2048
3072
4096
5120
6144
7168
8192
1 K
1 K
1 K
1 K
1 K
1 K
1 K
1 K
Potvrzená data
Bude možné odeslat po otevření okna
posouvání okna
uzavírání okna
otvírání okna

Picture from: Dostálek, Kabelová: Velký průvodce TCP/IP a systémem DNS

- Bytes of data stream numbred independently in both directions. Sequence number is related to first data byte carried in particular TCP segment

TCP Segment Length

- Communicating party may advertise how long TCP segments it is willing to accept
  - Using TCP option MSS (Maximum Segment Size) in SYN segment
- If MSS not present, sender uses
  - Local LAN's MTU when communicating with destinations at local LAN
  - Implicit value 536 for other destinations
    - 536=576B (minimal IP packet len) - 20B (IP Header) - 20B (TCP Header)

Data Acknowledgements

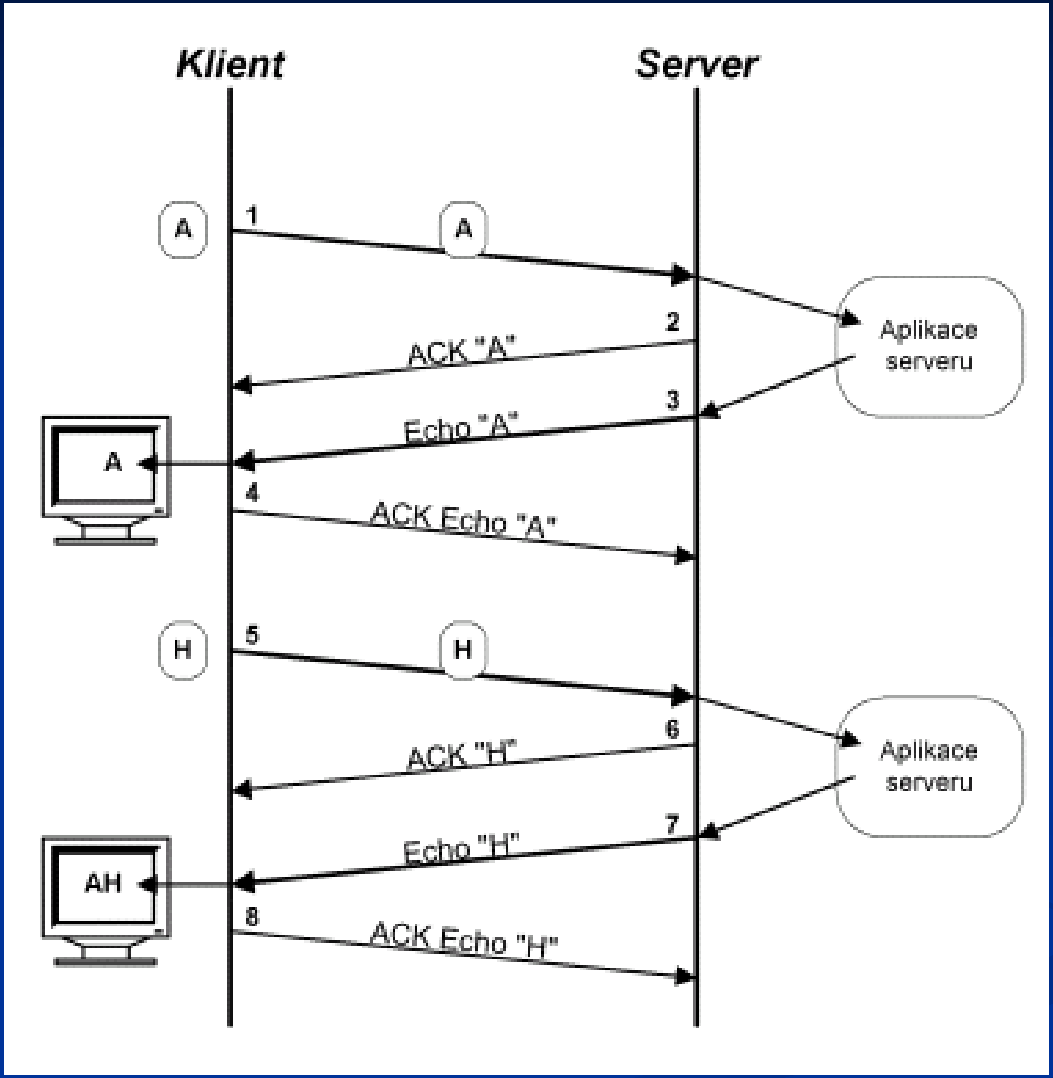
TCP Acknowledgements Paradigm

- Inclusive ACKs – ACK N also acknowledges all previous segments (ISN < n < =N)
- TCP does not support NAKs
- Uses piggybacking – ACK may send together with date in opposite direction
- ACK generation process started in reaction of receipt of every TCP segment
  - but ACK transmission is sometimes delayed and/or single ACK sent for multiple received segments

Delayed ACK Technique

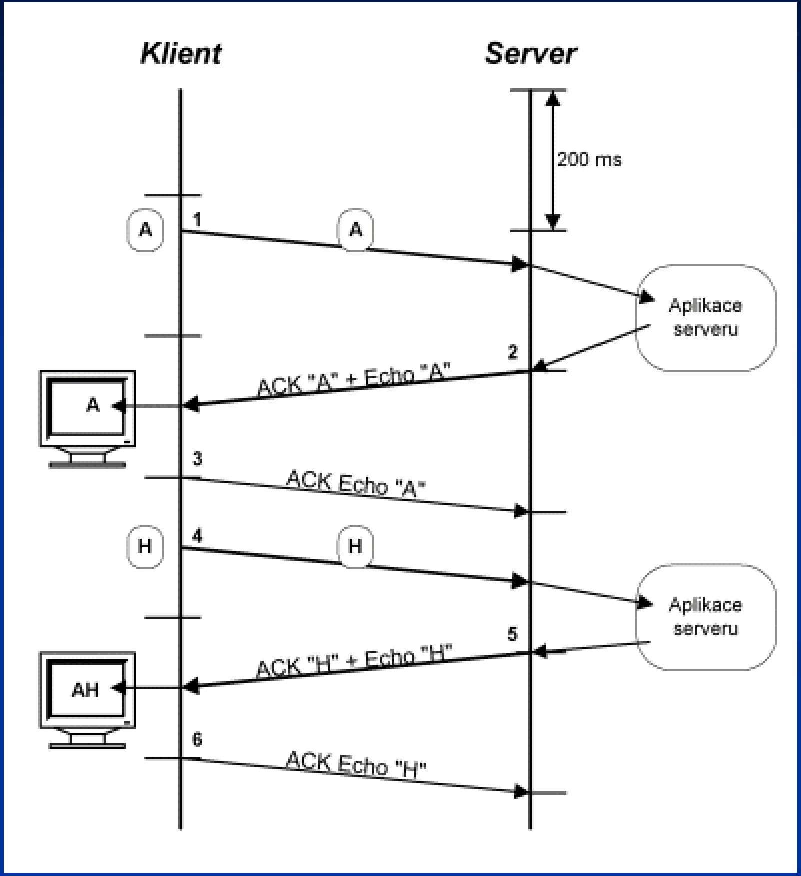
- Receiver does not generate ACK immediately after data reception, but waits a while for data in opposite direction generated in response by receiving application
  - ACK piggybacked into other direction's data in that case
  - Typical delay is 200 ms, standard allows for 500 ms maximum
- In case of reception of out-of-order segment, receiver generates "duplicate" ACK immediately
  - allows sender to detect lost segment

Interactive applications and delayed ACKs



Picture from: Dostálek, Kabelová: Velký průvodce TCP/IP a systémem DNS

Delayed ACKs – Common Implementation



Picture from: Dostálek, Kabelová: Velký průvodce TCP/IP a systémem DNS

OS timer ticks every 200ms to see whether ACK could have be sent

Nagle's Algorithm

Used for interactive applications which generate small packets ("tinygrams")

- Short packets have big overhead and overwhelm slow WAN links

Principle:

- Sender may send just one short packet to the network, the next one only after previous ACK comes
    - In the meantime, sender buffers outgoing data (until it reaches MSS)
- Automatically adapts enforced load to current network response

Must be turned off for some interactive (realtime) applications

- e.g. XWindow – oneway short messages about mouse movement must be sent always immediately

How does sender detect lost segment ?

- ACK timer expiration
- Duplicite ACK (multiple ACKs of the same sequence number)
  - Indicate that segments are getting to receiver, but some segment is missing
  - Two ACKs of the same sequence number may be caused by out-of-order reception, no action taken by sender
    - it is assumed that not-yet-received segment will come shortly delayed
  - Three or more duplicate ACKs are commonly considered an indication of lost segment
    - Sender may retransmit particular segment(s) without waiting for ACK timer expiration

TCP Retransmission Strategy

Retransmission Timeout (RTO) Calculation

- RTO derived from smoothed Round-trip Time (RTT)
- RTT Definition: time interval between transmission of segment with particular Seq# and reception of first ACK acknowledging that Seq# (directly or as inclusive ACK)

RTO Calculation – original version

- Let M be measured RTT
- Let R denote smoothed RTT calculated as follows:

  $R=a.R+(1-a).M$
- $RTO=R.b$

a – smoothing coefficient (commonly 0.9)

b – delay variation factor (2 is recommended)

RTO Calculation – improved version

- Takes into account higher RTT fluctuation
- Utilizes RTT average and variance
- Let A be smoothed value of RTT
- Let D be smoothed variance of Err=M-A
  - M is current RTT measurement

A=A+g.Err (g=1/8)
D=D+h.(|Err|-D) (h=1/4)
RTO=A+4.D

Karn's Algorithm

- RTO calculation improvement
- If timeout expires and segment is retransmitted, RTT of the following ACK is not taken into account into smoothed RTT
  - we are not sure whether that ACK was generated as a reaction of reception of original segment or retransmitted one

Fast Retransmit

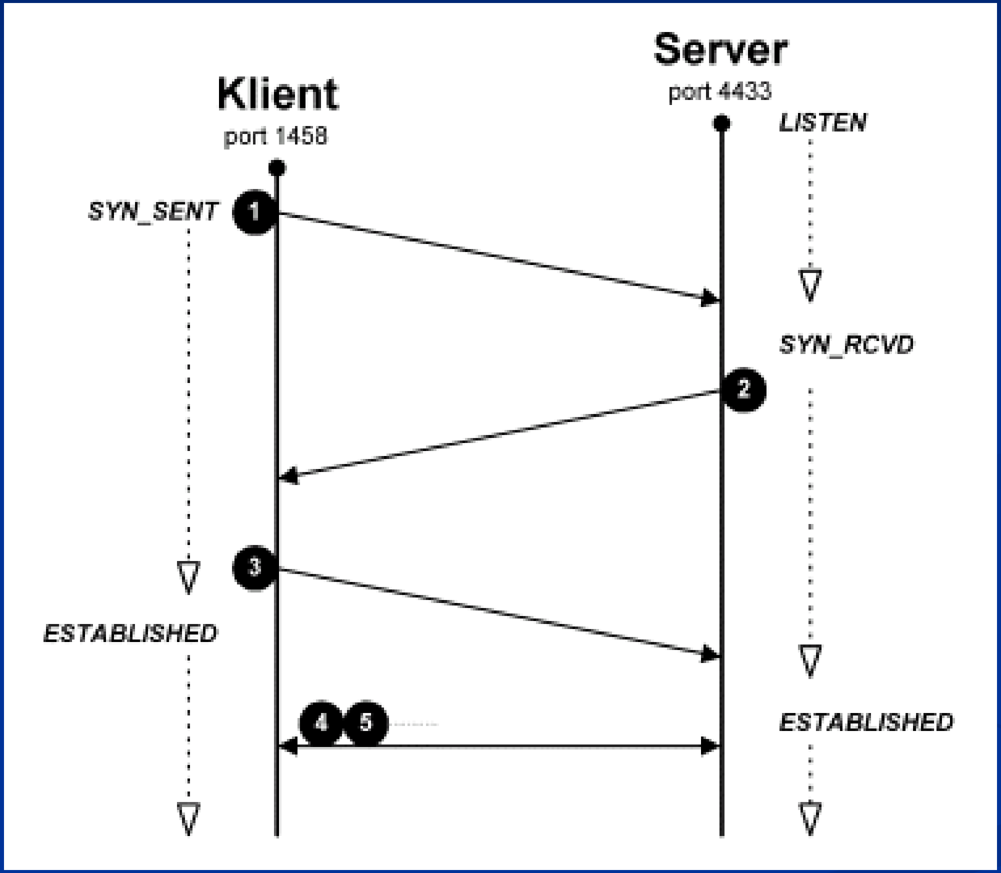
- If three duplicite ACKs are received, the segment considered lost is retransmitted without waiting to it's ACK timeout expiration
  - Fast retransmit is applied only if three duplicate ACKs come before normal retransmission caused by timeout expiration
- It is assumed that receiver generates duplicite ACK immediately after reception of out-of-order segment
- Fast Retransimit is useful for large sending window
  - Sender does not need to send as much data as it would have when waiting to expiration of individual timers of segments sent after lost one and not acknowledged by receiver because of out-of-order receipt
    - also saves time

TCP connection establishment and termination

Connection establishment

- Three-way handshake
- Random generation of Initial Sequence Numbers (ISN)
  - protects agaist potential lost and again appeared segments from previous broken connections
- SYN “consumes” one sequence number
  - SYN segment appears as 1B TCP segment
- If server does not responds to SYN, client gradually extends the period between consecutive attempts
  - BSD implementation reports error to application after 75 secs of unsuccessful attempts

TCP Stage Diagram – connection establishment



Picture from: Dostálek, Kabelová: Velký průvodce TCP/IP a systémem DNS

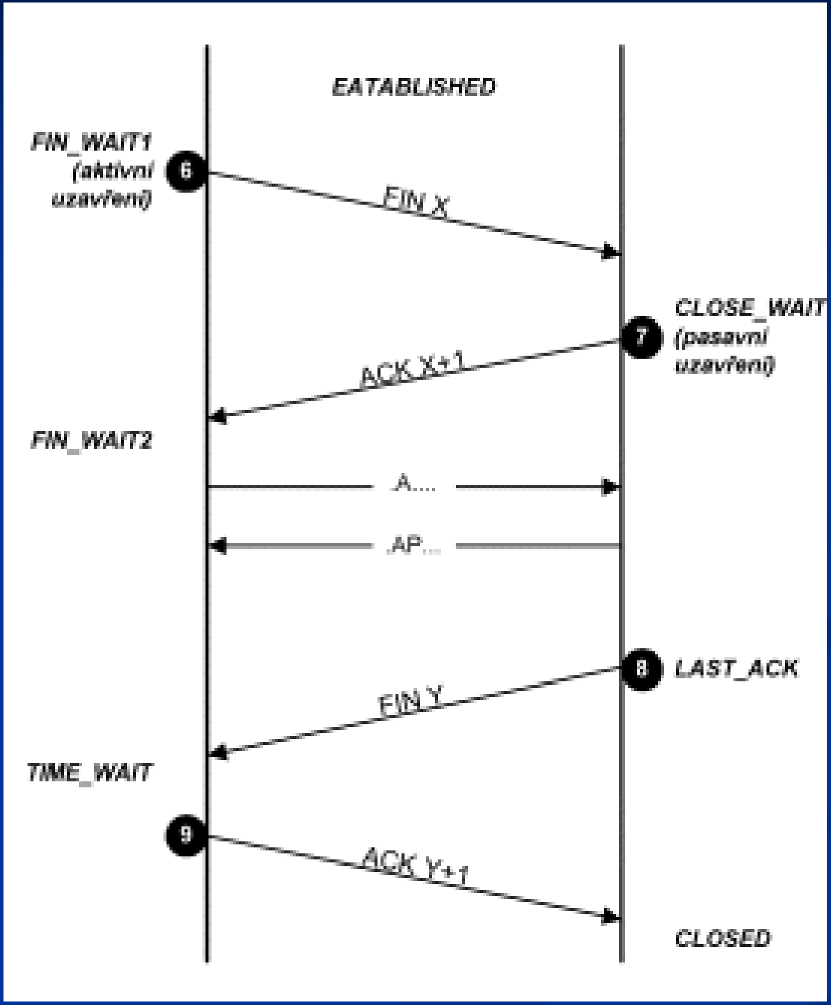
Server Listening Queue

- Maintains connection for which three-way handshake took place
- Established by listen(socket,BACKLOG) call
  - queue created for port bound with socket
  - BACKLOG=number of slots in the queue
- Application accepts connections using accept(socket) call
- If no slot is free in listening queue, another received SYNs are ignored
  - client will re-send SYN again, some slot may be freed in the queue meanwhile

Connection Termination

- FIN+ACK independently in both directions
  - active and passive closing
- Half-close state: station indicates end of data transmission, but is still able to receive data
  - Usage example: *rsh myhostname.cz sort < datafile*
- FIN "consumes" one sequence number (similar to SYN)

TCP Stage Diagram – connection termination



Picture from: Dostálek, Kabelová: Velký průvodce TCP/IP a systémem DNS

2MSL State
(Maximum Segment Lifetime)

- On the side of station closing connection actively
  - Sends last ACK
- Must be able to resend last ACK if it is lost
  - If last ACK is lost, repeated FIN arrives
- Station must wait whether repeated FIN comes
  - wait time stated as sum of expected maximums of last ACK and potential repeated FIN propagation time (2 * Maximum Segment Lifetime, 2MSL).
  - 2MSL time set to 30 secs or 2 minutes in most TCP implementations

2MSL State Problems

- OS must maintain state information of just closed socket for 2MSL time
  - during that time, it is not possible to reopen TCP connection with the same parameters (local/remote IP address, local/remote port).
    - Some implementations even do not allow to reopen the same local port for 2MSL state
  - causes problem with restart of servers listening at well-known ports
    - (2MSL state restrictions may be overriden with Socket call option)
  - clients are not influenced, because clients use ephemeral ports anyway
    - this is why the connection is closed by client in most real client-server applications

Connection refusement/reset

- Indicated by RST flag
  - if client tries to establish connection to port not bound with any server process
  - if some communicating party detects the peer is untrustworthy

TCP Flow Control

Why to control data flow rate ?

The aim is to avoid congestion

- If anybody transmits without regard to others, congestion occurs and nobody is able to transfer data

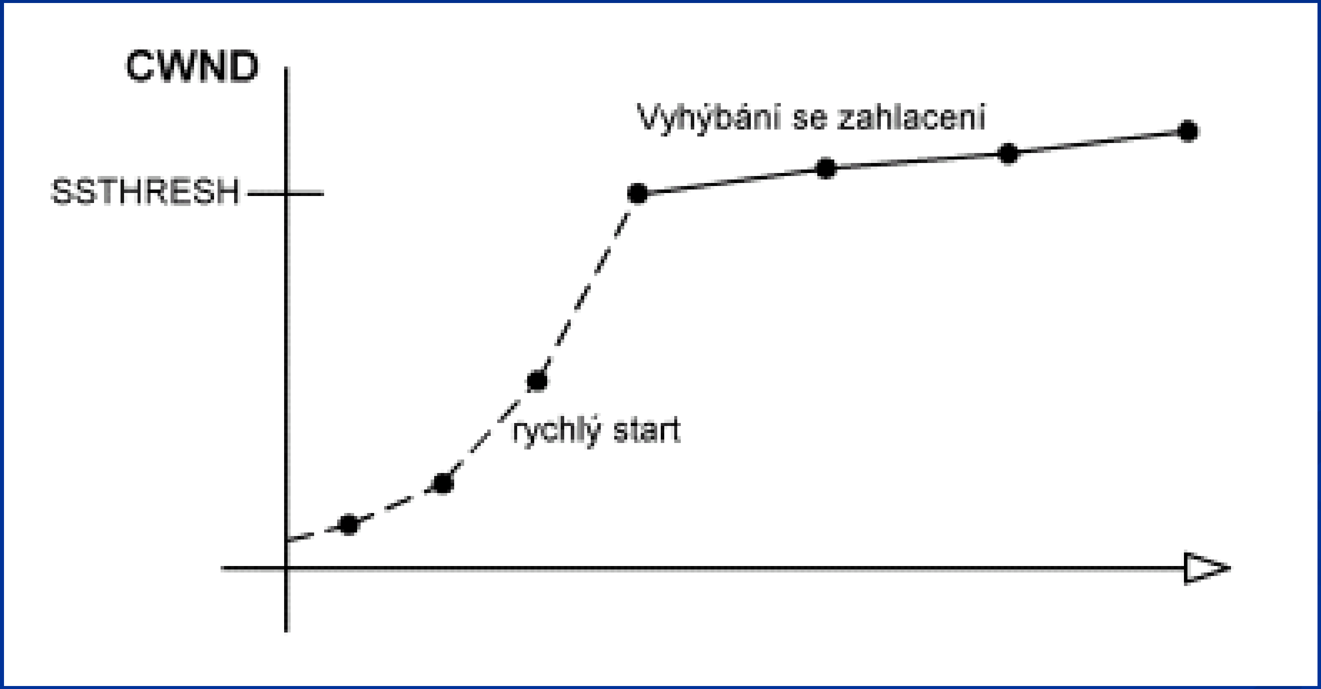
Most common reasons of congestion:

- Data are transmitted from faster network to slower one
- No free space in router queues
  - (multiple flows sum up into single output link queue)

Where flow control takes place ?

- Data flow controlled by receiver
  - Receiver advertises current receiving window size according to remaining place in the receiver buffer
  - Window field in TCP header is 16 bit long
    - TCP Scale option in SYN segment allows to shift WINDOW value by n bits left
    - useful for big *bandwidth*delay* products
- Sender adapts sending window size according to network current throughput
  - feedback using ACK reception, slow-start

Slow Start and Congestion Avoidance



Picture from: Dostálek, Kabelová: Velký průvodce TCP/IP a systémem DNS

Slow Start

- If new TCP connection would start to produce data in maximum rate immediately and network operates just below it capacity limit, all existing TCP connections would become unusable
- For that reason, sender will accelerate gradually and adapt it transmission rate to the rate of received ACK
- Used only for connections out of local LAN
- Initiated when timeout expires, not when segment loss detected by duplicate ACKs
  - In that case, some segments are still able to get to receiver

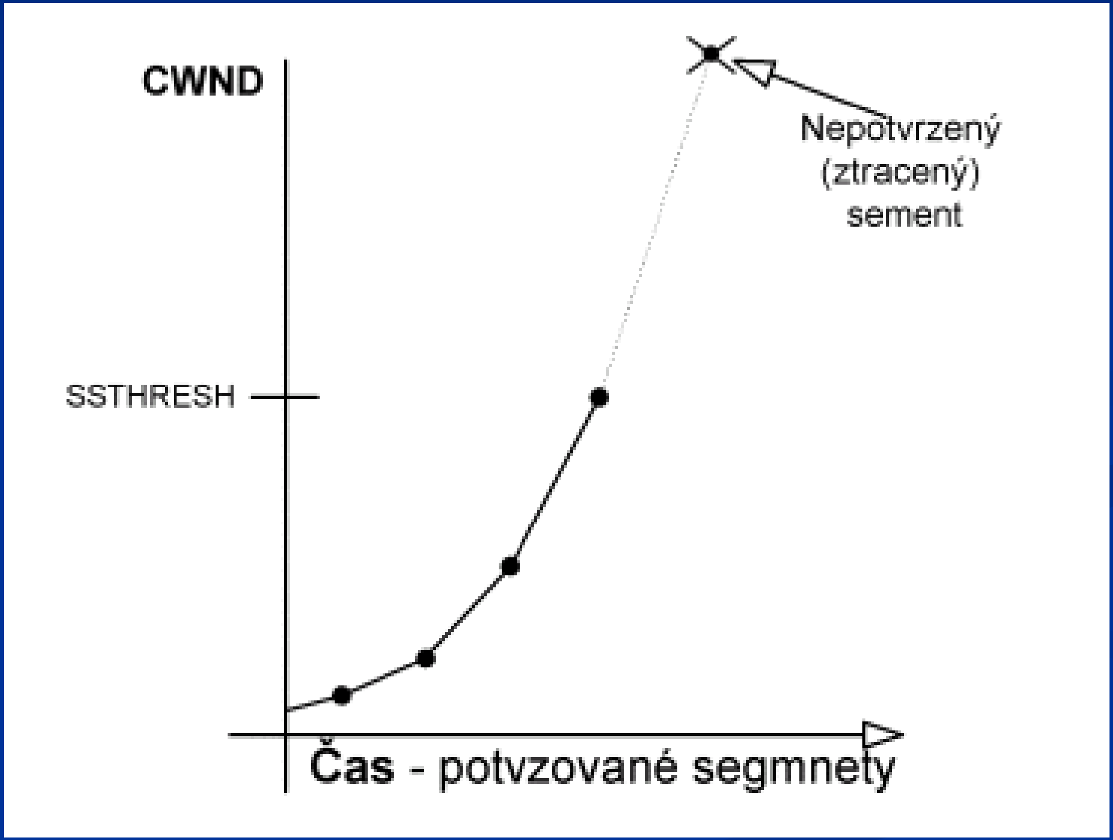
Slow Start Implementation

Slow Start Algorithm:

- Initialize CWND=1 segment (MSS bytes)
- After receipt of every ACK of transmitted segment (possibly inclusive ACK), increase CWND by one segment (MSS bytes)
- Continue until first loss of ACK

As a result, number of segments sent increases exponentially (1,2,4,8,16,...).

Slow Start Behavior



Picture from: Dostálek, Kabelová: Velký průvodce TCP/IP a systémem DNS

Slow Start and Congestion Avoidance Implementation

Commonly implemented together.

Sender maintains and adapts

- size of sending window (CWND,"Congestion Window“)
  - calculated in bytes
- Threshold value (SSTHRESH) = size of CWND when network congestion may start to occur
- The aim is to maintain CWND little above SSTHRESH, where network is utilized as much as possible, but congestion still do not occur

Sender flow control implementation

- Initialize CWND=1xMSS, SSTHRESH=65535 B.
- Maximum number of bytes transmitted: min(CWND,WINDOW)

- When segment has to be transmitted:
  - If CWND < SSTHRESH, CWND grows exponentially according to Slow Start algorithm
  - Otherwise linear increase of CWND is applied (Congestion Avoidance algorithm)
    - CWND+=(MSSxMSS/CWND + MSS/8) - integer aritmetics used
    - Second term causes faster window open for larger segments
- When segment loss is detected:
  - SSTHRESH=min(CWND/2,WINDOW), but not less than 2*MSS
  - If loss detected by timeout expiration: CWND=1xMSS, Slow start follows
  - If loss detected by duplicate ACK: only adapts SSTHRESH=max(CWND/2, 2*MSS)

Silly Window Syndrome

- Occurs when receiver advertises short receiver window multiple times
- Causes ineffective transmission of multiple short segments instead of one longer
- It is better for receiver to wait until it will be able to advertise larger window

Another TCP Issues

Persist Timer

- If receiver closes window (i.e. advertises zero size) and then opens again, but second ACK with nonzero size gets lost, deadlock occurs
- Solution: communicating parties may periodically check for window size of other party using window probes
  - window probes are normal segments with zero size and last sequence number repeated
  - Receiver must react with ACK with current window size

Keepalive Timer

- May be used to detect broken TCP connection (using keepalives)
  - server may release allocated resources
- May be turned on in Sockets API
- Inconsistent with original TCP philosophy, connection will be disconnected even in case of temporary connectivity loss
  - but some programmers are used to get immediate information when communication channel fails to operate
- It is commonly better to check for connectivity at application layer

Today’s TCP problems

- Fast links – big data volumes may be transferred in short time, but propagation delay is still restricted by physical constraints
  - Need for larger window in Sliding Window algorithm
    - large receiver’s window consumes receiver memory (MB)
    - TCP doesn’t support this – Scale option introduced
- Development of wireless Internet infrastructure changes characteristics of communication environment
  - in the near past, we could count with reliable links, but we can not do that anymore