Advanced Computer Networking (ACN)

IN2097 – WiSe 2019-2020

Prof. Dr.-Ing. Georg Carle

Sebastian Gallenmüller, Max Helm, Benedikt Jaeger, Patrick Sattler, Johannes Zirngibl

Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich
Chapter 7: Transport Layer Protocols

TCP
  Basics
  Flow Control
  Congestion Control

UDP

SCTP

QUIC
  QUIC Features
  In Practice
  Standardization

Bibliography
Chapter 7: Transport Layer Protocols

TCP
- Basics
- Flow Control
- Congestion Control

UDP

SCTP

QUIC

Bibliography
What is TCP?

- Short for Transmission Control Protocol
- Defined in RFC 793 [1] and many more RFCs
- Connection-oriented service
- In-sequence delivery of byte stream → Stream-oriented
- Reliability properties
  - Bit error detection
  - TSDU (Transport Service Data Unit) loss detection and retransmission
- Provides Flow Control (sender will not overwhelm receiver)
- Provides Congestion Control (sender will not overwhelm network)

When is it used?

- HTTP
- FTP
- SMTP / POP3 / IMAP
- SSL / TLS
- SSH, BGP, Backups, …
Basics

Properties

Point-to-Point
- One sender, one receiver

Reliable
- Everything that was sent will be received at some point in time

In-Order
- Sending order is receiving order

Stream-oriented
- Data is one continuous stream, no message boundaries
- Example:
  - `send("Hello"); send("World");`
  - `recv()
    "HelloWorld", "Hello", "", "Hell"
  - not possible:
    "World", "HeWorld"

Connection-oriented
- Handshakes, holding state on both sides, tear-downs
- 3-way handshake
  `SYN → SYN/ACK → ACK`
  `FIN → FIN/ACK → ACK`

Flow controlled
- Sender can only send as much as the receiver can utilize

Congestion controlled
- Sender throttles bandwidth to not overwhelm network
**Basics**

**TCP Header**

|     | 0    | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   | 16   | 17   | 18   | 19   | 20   | 21   | 22   | 23   | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 0B  |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 4B  | Source Port |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 8B  |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 12B | Offset | Reserved |      |      | URG  | ACK  | PSH  | RST  | SYN  | FIN  | Window |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 16B |      |      |      |      |      |      |      |      |      |      | Checksum |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 20B |      |      |      |      |      |      |      |      |      |      | Options (0 or more multiples of 4 Byte) |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|     |      |      |      |      |      |      |      |      |      |      | Data |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |

- **Source Port**: Identifier for sending application
- **Destination Port**: Identifier for receiving application
- **Sequence Number**: Identifier for segment, byte sequence number of first byte of the segment
- **Acknowledgment Number**: Next expected sequence number
- **Offset**: Offset to start of payload (header length including options)
- **Reserved**: Reserved for future use
- **Window**: Size of receiver window (buffer size of receiver), used for Flow Control
- **Flags**:
  - **URG**: Urgent pointer is set
  - **ACK**: Acknowledgment is set
  - **PSH**: Push, OS should not buffer
  - **RST**: Reset connection (instant termination)
  - **SYN**: Synchronize using handshake packets
  - **FIN**: Finish, start to tear-down connection
- **Checksum**: 16 bit one’s complement of the one’s complement sum of all 16-bit words in TCP pseudo-header and payload
- **Urgent Pointer**: Points to urgent data
- **Options**: Optional extensions
Basic Operation

**Sender:**

- Sends segments
- Expects acknowledgments

**How does the sender know if a segment is lost?**

- **ACK is not received within a certain time interval**
  - Timeout occurs
  - Sender retransmits segment
  - Takes at least timeout value + 1 RTT until lost segment is acknowledged

- **Multiple duplicate ACKs arrive**
  - Can be caused by reordering or packet loss
  - Segments arrived out-of-order, or at least one segment was lost
  - 3 duplicate ACKs $\Rightarrow$ Sender does Fast Retransmit
  - **Remark:** 3 duplicate ACKs $\neq 3$ lost segments
  - One or more lost segments always cause duplicate ACKs as long as there are subsequent segments
Basics
TCP Round Trip and Timeout

How to set the TCP timeout value?

- longer than RTT, but RTT varies
- too short: premature timeout
  - unnecessary retransmissions (Spurious Retransmission)
- too long: slow reaction to segment loss

How to estimate RTT

\[
\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}
\]

- Called: Exponential weighted moving average (EWMA)
- Influence of past sample decreases
- More weight on recent samples than on older samples
- Typical value: \( \alpha = 0.125 \)
Basics
TCP Round Trip and Timeout

How to set the TCP timeout value?

- longer than RTT, but RTT varies
- too short: premature timeout
  - unnecessary retransmissions (Spurious Retransmission)
- too long: slow reaction to segment loss

How to estimate RTT

- SampleRTT: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- SampleRTT will vary, want estimated RTT “smoother”
  - average several recent measurements, not just current SampleRTT

\[
\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT} \quad (1)
\]

- Called: Exponential weighted moving average (EWMA)
- influence of past sample decreases
- more weight on recent samples than on older samples
- typical value: \( \alpha = 0.125 \)
Basics

Exponential weighted moving average (EWMA)
TCP Options

Maximum Segment Size Option (RFC 6691)

- Announce MSS during handshake: accept no segments larger than MSS
- Can be completely independently in each direction
- MSS counts only data octets in the segment, it does not count the TCP/IP header.

TCP Timestamp Option (RFC 7323)

- Prevents ambiguity of ACKs (is the ACK from the original packet or the retransmission?)
- Sender and receiver have a (virtual) "timestamp clock"
- Append two "timestamps" to each sent TCP segment:
  - Timestamp Value (TSVal): current "timestamp" when the packet is sent
  - Timestamp Echo Reply (TSecr): latest TSVal received before sending the packet
- On receive compute: TSecr − current timestamp

TCP Window Scale Option (RFC 7323)

- 2B windows size field → at most about 65kB receive buffer
- Scale the announced window by a factor (shift the window)

Selective Acknowledgment Options (RFC 2018)

- TCP only provides feedback about the next expected segment
- What if each second segment gets lost? → many RTTs to retransmit everything
- Goal: provide more information about received/missing segments
Flow Control

What is Flow Control?

- Receiver may be a resource limited device
- OS kernel buffers segments for applications to process
- Buffer is of limited (maybe small) size
- If the buffer is full: incoming segments are dropped
- **Flow Control** sets a maximum of data the sender is allowed to send
- Implemented by using the window field in the header
- Used in order to avoid overwhelming of receiver buffer
Flow Control

What is Flow Control?

- Receiver may be a resource limited device
- OS kernel buffers segments for applications to process
- Buffer is of limited (maybe small) size
- If the buffer is full: incoming segments are dropped
- Flow Control sets a maximum of data the sender is allowed to send
- Implemented by using the window field in the header
- Used in order to avoid overwhelming of receiver buffer
Definition:

- Informally: “Too many sources sending too much data too fast for the network to handle”
- Different from flow control (which handles overload at the recipient)

Manifestations:

- Lost packets (buffer overflow at routers)
- Long delays (router buffer queues fill up)
Congestion Control
Goals and problems

What do we hope for?

- Reasonable behavior in case of high load of network
- Without controlling amount of outgoing data, capacity may drop dramatically because of congestion collapse
- Fair resource sharing
- Criteria: effective, simple, robust, end-host driven
Congestion Control
Operation Point

Every path can be described with two parameters:

- **Round-trip propagation delay:**
  \[ RT_{prop} = \sum_i RT_{prop_i} \] with \( RT_{prop_i} \) being the transmission delay of link \( i \)

- **Bottleneck Bandwidth:**
  \[ BtlBw = \min(BtlBw_i) \] with \( BtlBw_i \) being the bandwidth of link \( i \)

- **BtlneckBufSize** = buffer size at the bottleneck link

- **Amount inflight:** data which is sent but not acknowledged

- **Bandwidth-delay product:** \( BDP = RT_{prop} \cdot BtlBw \)
  "How much data can fit on a link with bandwidth BtlBw and propagation delay RTprop"

- **Application Limited** if Inflight < BDP
  - link underutilization
  - low latency

- **Bandwidth Limited** if BDP < Inflight < BDP + BtlneckBufSize
  - full link utilization
  - buffer starts filling

- **Buffer Limited** if BDP + BtlneckBufSize < Inflight
  - packet loss leads to lower goodput (retransmission consume bandwidth)
  - unpredictable latency due to retransmissions
Congestion Control
Operation Point

RTT

Delivery Rate

BDP

Amount Inflight

BDP + BtlneckBufSize

Chapter 7: Transport Layer Protocols – TCP 7-14
Congestion Control
Operation Point

Chapter 7: Transport Layer Protocols – TCP

RTT
BDP
RTprop
BDP + BtlneckBufSize

Amount Inflight
Application Limited
Delivery Rate

BDP
BDP + BtlneckBufSize
Congestion Control

Operation Point

Application Limited
Bandwidth Limited

RTprop
RTT
BDP
BDP + BtlneckBufSize
BtlBw
Amount Inflight
Delivery Rate

Chapter 7: Transport Layer Protocols – TCP 7-14
Introduction

The problem

Remy

Evaluation

Discussion

The march of congestion control mechanisms

DECbit

Tahoe

CARDDUAL

RenoNewReno

REDBLUE

Vegas

ECN

SACKEBCC

CHOKe

WestwoodBinomial

DUAL

Veno

Compound

LEDBAT

Tahoe

Reno

NewReno

BIC

H-TCP

Cubic

PRR

Sprout

1980s 1990s 2000s 2010s

DeTail

LEDBAT

GPSWFQ

In-net

WFQ

BLUE

1980s

1990s

2000s

2010s

Figure 1: TCP congestion control algorithms

Metrics to take into consideration (according to RFC 5166)

- Throughput, Delay, and Loss Rates
- Response Times and Minimizing Oscillations
- Fairness and Convergence
  - Fairness between Flows
  - Fairness between Flows with Different Resource Requirements
  - Fairness to other TCP congestion control algorithms
- Response to Active Queue Management (AQM)
- Robustness for Challenging Environments
- Robustness to Failures and to Misbehaving Users
- Deployability
Robustness for challenging environments

- Robustness needs to be explored for paths with reordering, corruption, variable bandwidth, asymmetric routing, router configuration changes, mobility, and the like.
- In general, the Internet architecture has valued robustness over efficiency.

Energy consumption

- In mobile environments, the energy consumption for the mobile end-node can be a key metric.

The goodput ratio

- For wireless networks, the *goodput ratio* can be a useful metric
- The goodput ratio can be defined as the useful data delivered to users as a fraction of the total amount of data transmitted on the network
### Deployment requirements

- Only at the sender (e.g., NewReno in TCP)
- Only at the receiver (e.g., delayed acknowledgments in TCP)
- Both the sender and receiver (e.g., Selective Acknowledgment (SACK) TCP)
- At a single router (e.g., Random Early Detection (RED))
- All of the routers along the end-to-end path
- Both end-nodes and all routers along the path (e.g., Explicit Control Protocol (XCP))

### Other points

- Complexity of the code
- Support in operation systems
Congestion Control
Is TCP Fair?

Problem:

- Multiple TCP flows use the same path
- Do all of them get an equal share of the bandwidth?
- Multiple different congestion control algorithms may be used!

Results:

- Both Cubic and BBR claim fairness
Metrics often used for assessing numerically the fairness between \( n \) flows with \( x_i \) the bandwidth of flow \( i \):

- **The product measure:**
  \[
  \prod_{i} x_i
  \]

- **Epsilon-fairness:** A rate allocation is defined as epsilon-fair if
  \[
  \frac{\min_i x_i}{\max_i x_i} \geq 1 - \epsilon
  \]

- **Jain’s fairness index:**
  \[
  \left( \frac{\sum_i x_i}{\sum_i x_i^2} \right)^2 \in \left[ \frac{1}{n}, 1 \right]
  \]
  - Returns a value between 0 and 1
  - Scale free
  - Arbitrary number of flows
  - Is \( \frac{k}{n} \) if there are \( k \) flows are perfectly fair while the other \( n - k \) shares are 0
Congestion Control Algorithms

How does TCP regulate the sending rate?

- Only have a well-defined number of bytes (or segments) in the network, which have not yet been acknowledged
- This number of bytes is called the Congestion Window

How to process available information to modify the congestion window size?

- There are a lot of algorithms
- Different classes:
  - loss-based
  - delay-based
  - model-based
  - hybrid approach
  - ...
- Most popular / interesting:
  - TCP Tahoe (slow start, congestion avoidance)
  - TCP Reno (fast retransmit, fast recovery, today: TCP New Reno)
  - TCP Vegas
  - TCP Cubic (current default in the Linux kernel)
  - TCP BBR (new, proposed by Google)
Loss-based Congestion Control

How do loss-based algorithms detect congestion?

- Assumption: Packet loss only happens due to congestion
- No packet loss → increase congestion window
- Packet loss → decrease congestion window
- Advantages: robust, reliable, efficient
- Disadvantages: buffers are kept full → high latency, performance drop on lossy links
- Examples: Reno, Bic, Cubic
TCP Reno
Basics of the Algorithm

Theory behind Reno:

• Every packet loss is induced by a network overload
  • Therefore, TCP senders should reduce data rate
  • However, think about lossy links!

• AIMD strategy: Additive Increase Multiplicative Decrease

• Two modes of operation:
  • Slow Start: Exponential growth of congestion window
  • Congestion Avoidance: Linear growth of congestion window
TCP Reno
Algorithm

Variables:

- **CWND**: Congestion Window, limits the amount of inflight data
- **ssthresh**: Slow Start threshold

Slow Start:

- For every acknowledged **MSS**, increase **CWND** by 1 MSS
- Use this mode, if **CWND** < **ssthresh**

Congestion Avoidance:

- For every acknowledged **MSS**, increase **CWND** by 1/CWND effectively increase **CWND** by 1 **MSS** each RTT → additive increase
- Use this mode, if **CWND** ≥ **ssthresh**

Reception of 3 duplicated acknowledgments:

- Set **ssthresh** to **CWND**/2
- Set **CWND** to **ssthresh** (Fast Recovery) → multiplicative decrease

Acknowledgement timeout:

- Set **ssthresh** to **CWND**/2
- Set **CWND** to 1 MSS
- Restart with Slow Start
TCP Reno

Example

Chapter 7: Transport Layer Protocols – TCP 7-25
TCP Reno
Value of ssthresh

According to RFC5681

- "The initial value of ssthresh SHOULD be set arbitrarily high"
- "Setting ssthresh as high as possible allows the network conditions, rather than some arbitrary host limit, to dictate the sending rate."
- "In cases where the end systems have a solid understanding of the network path, more carefully setting the initial ssthresh value may have merit."

In the Linux kernel

- In net/ipv4/tcp.c: `tp->snd_ssthresh = TCP_INFINITE_SSTHRESH;`
- In include/net/tcp.h: `#define TCP_INFINITE_SSTHRESH 0xffffffff`
TCP Reno

Problems of TCP Reno

- Low performance on lossy links
- Buffers are filled
- Increase depends on the RTT \(\rightarrow\) slow growth on long distance links
- Has problems fully utilizing large BDP links