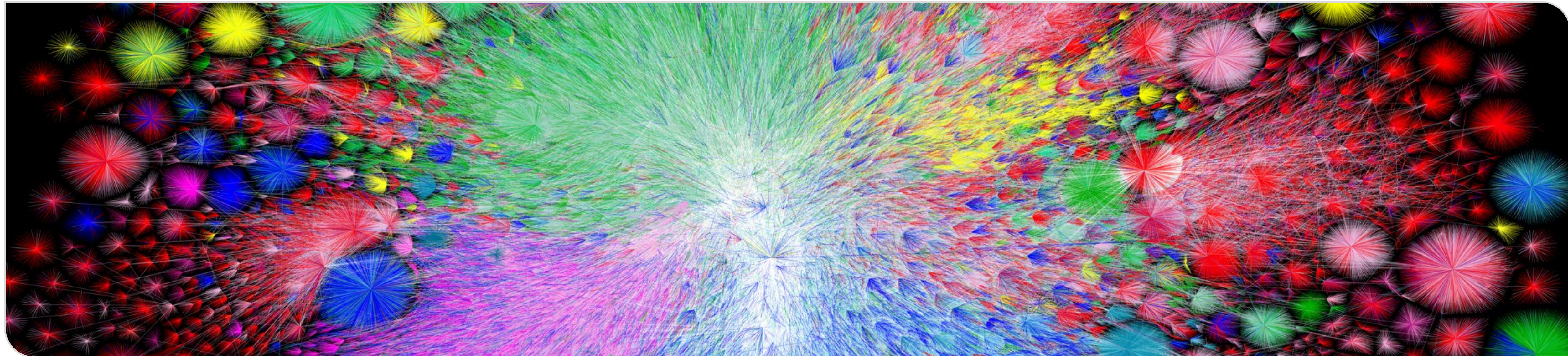
Karlsruher Institut für Technologie

TELEMATICS

9. Data Transport Evolution

Prof. Dr. Martina Zitterbart
Institute of Telematics



Kahoot! 09-01

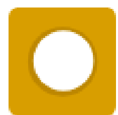
Karlsruher Institut für Technologie

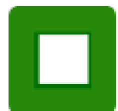
1 - Quiz

Priority-based Flow Control (PFC)

Is used together with RDMA in data centers

Provides different priorities for flows


Is an alternative to PAUSE frames


Provides different priorities at ports

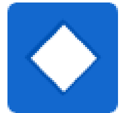
Kahoot! **09-01**

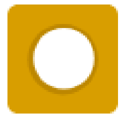
Karlsruher Institut für Technologie

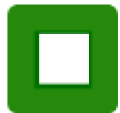
2 - Quiz

DCTCP

Scales congestion window based on estimate of congested traffic


Uses a modified ECN


Uses cumulative ACKs

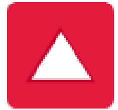

Uses immediate ACKs

Kahoot! 09-01

Karlsruher Institut für Technologie

3 - Quiz

Meta Infrastructure


Uses centralized network control in private WAN

Includes O(100) datacenter regions

Operates with long-lived TCP connections between PoP and datacenter

Uses Converged Ethernet in its private WAN

Kahoot! 09-01

Karlsruher Institut für Technologie

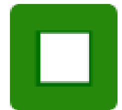
4 - Quiz

Which of the statements are true?

ToR switches provide connectivity to servers

RDMA is used on datacenter switches

PFC may lead to victim packets


DCTCP leads to highly utilized queues

Karlsruher Institut für Technologie

9 Data Transport Evolution

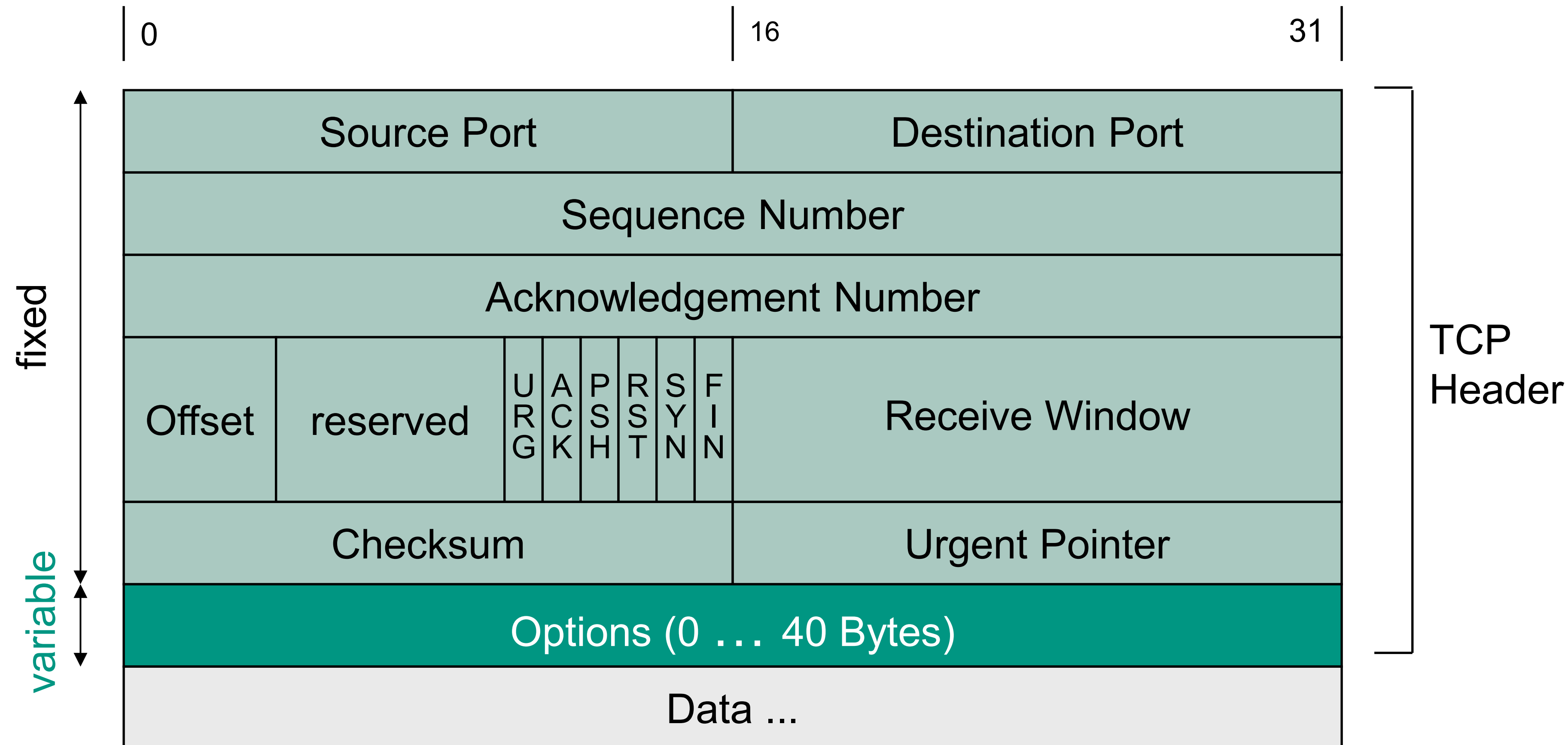
9.1	TCP Extensions
9.2	TCP in Networks with High BDP
9.3	CUBIC TCP
9.4	TCP and Response Time
9.5	QUIC
9.6	BBR TCP

Karlsruher Institut für Technologie

9.1 TCP Extensions

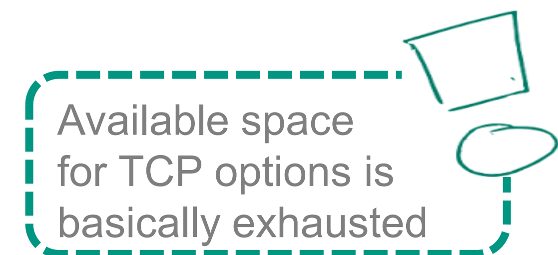
9.1.1 TCP Options: Basics

TCP Header



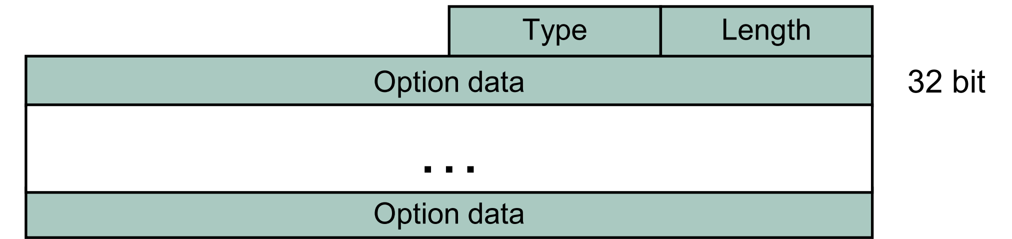
TCP Options

- Goal
  - Flexibility for new developments
- TCP header field
  - Each option is coded in TLV format (Type-Length-Value)
  - Has variable but limited length (max. 40 bytes)
    - Thus, number of options also limited
    - TCP header length at most 60 bytes in total (incl. options)



TCP Options

- Format (TLV)
  - Multiple of 32 bit words
    - If not, padding is needed
  - Type
    - Selective acknowledgements
    - Time stamps
    - Window scaling
    - Maximum segment size
    - Multipath TCP
    - TCP fast open
    - …
  - Length
    - Length of option
  - Value
    - Option data



[https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml]

9.1.2 Option Selective Acknowledgements

Motivation

- TCP (as in RFC 793) uses cumulative acknowledgements
  - Pro: Very robust against loss of ACK segments
  - Con: Inefficient loss recovery
    - Sender can only learn about a single lost segment per RTT

    - Consequently
      - Fast retransmit/fast recovery can only recover one lost segment per RTT
      - Multiple losses often lead to retransmission timeouts and head-of-line blocking

- Improvement: selective acknowledgements
  - Also acknowledge “out-of-order” data
  - Implemented as TCP option (RFC 2018)

RFC 2018 only specifies how to report the information. Does not specify what to do with it.

Selective Acknowledgements (SACK)

Karlsruher Institut für Technologie

- Idea
  - Separately acknowledge continuous blocks of out-of-order data
- Usage of SACK option negotiated during connection establishment

Type=4	Length=2

SACK-Permitted Option
(only set in SYN segments)

- SACK option format
  - Contains different blocks of sequence numbers which are acknowledged

	Type=5	Length
Left Edge of 1st Block		
Right Edge of 1st Block		
...		
Left Edge of *n*-th Block		
Right Edge of *n*-th Block		

- TCP option space is scarce
  - Maximum of 40 bytes for all used options
  - Typically, only 2-4 blocks can be "SACKed" in one segment

[RFC2018]

Selective Acknowledgements (SACK)

Karlsruher Institut für Technologie

- How to deal with such a case?

= received

= missing

time

increasing sequence numbers

- Use first entry of SACK option to report new information
- Use subsequent entries of SACK option for redundancy
  - Used for redundancy, if prior ACKs were lost
  - Should repeat most recently sent blocks first

[RFC2018]

TeleMatics

On Incoming Data

- Different alternatives
  - Segment received *in-order*

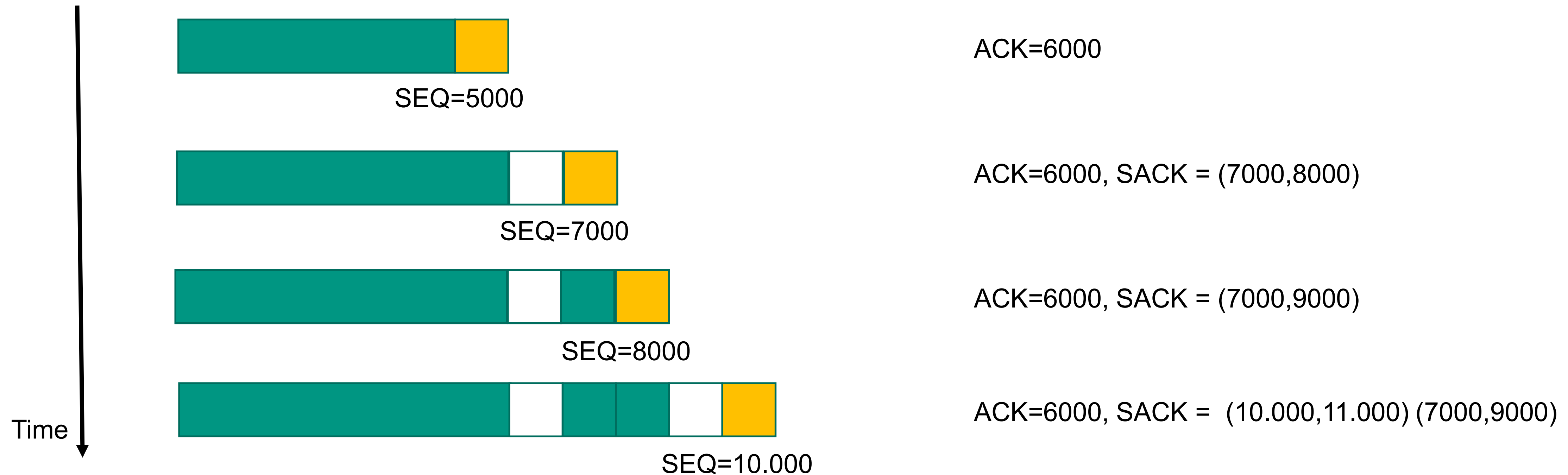
    - ACK regularly
  - Isolated segment received

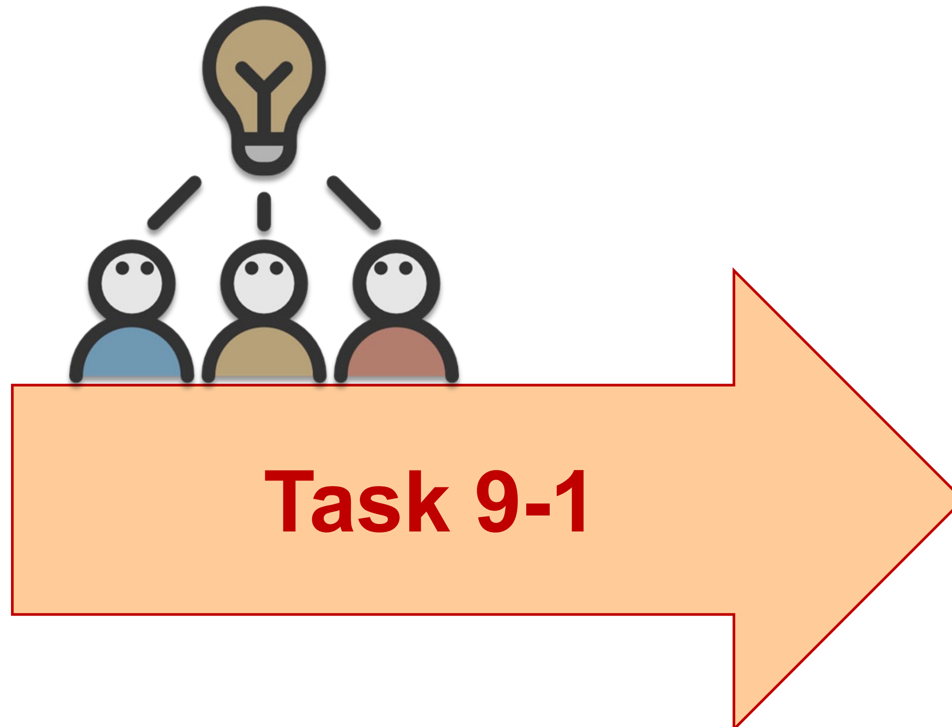
    - SACK newly received segment (in first entry of SACK option)
  - Received out-of-order segment adjacent to prior out-of-order data

    - SACK the whole block the new data belongs to (in first entry of SACK option)
      - This gives additional redundancy

Example

- Last received ACK = 5000, segment size = 1000

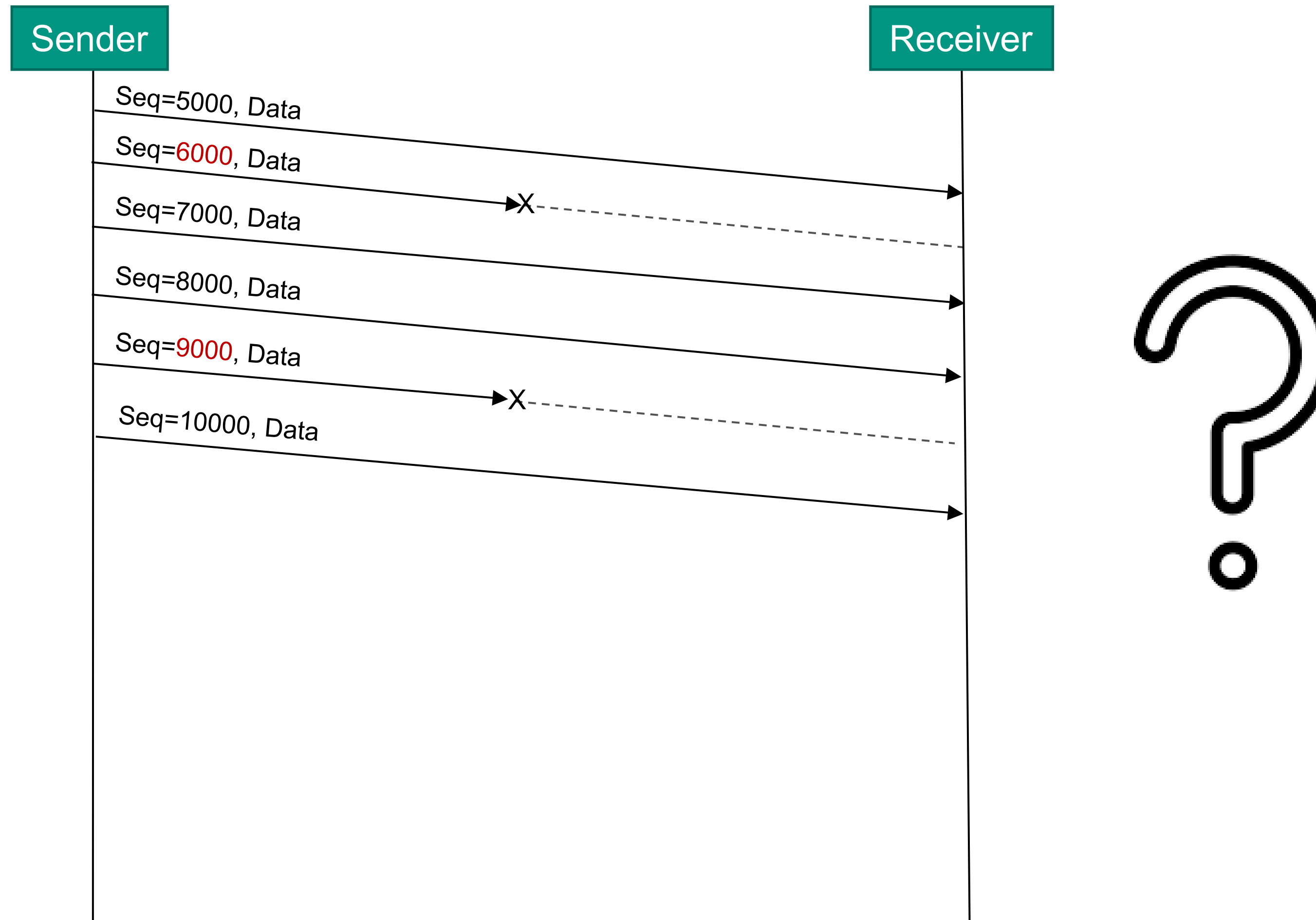



Task 9-1

TCP
ACKs

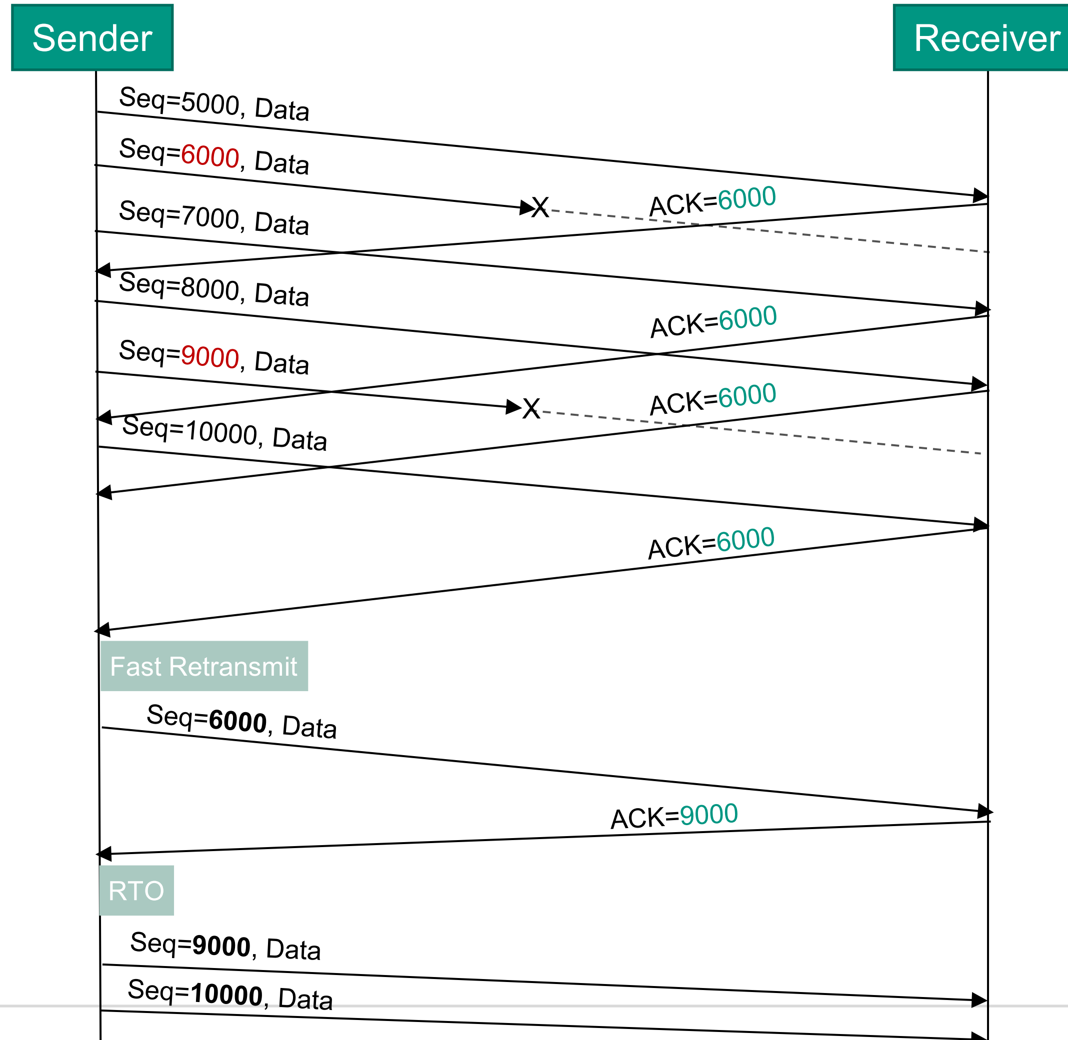
Cumulative Acknowledgements Only (ACK)

- TCP-Reno, last received ACK = 5000, segment size = 1000



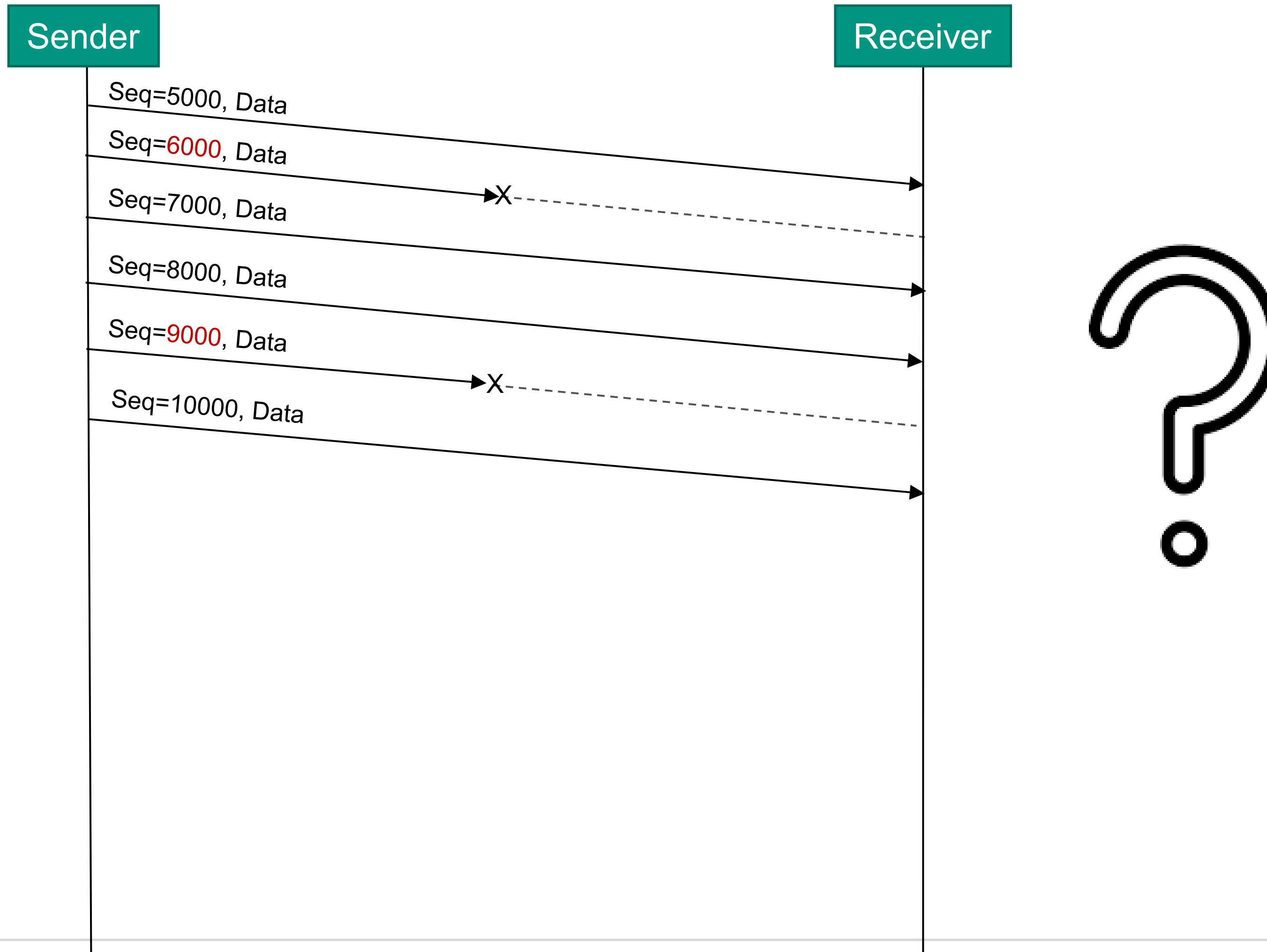
Cumulative Acknowledgements Only (ACK)

- TCP-Reno, last received ACK = 5000, segment size = 1000



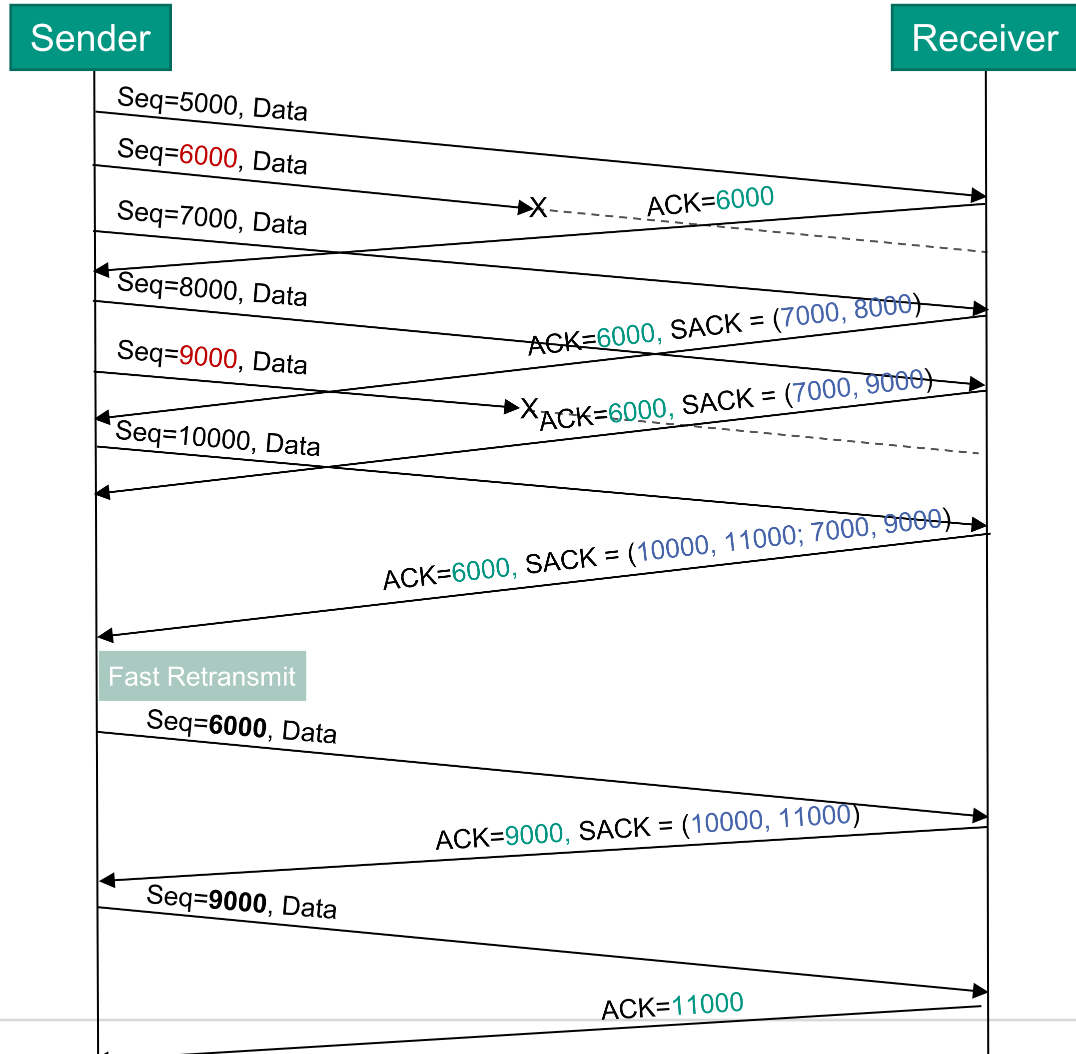
Cumulative and Selective ACKs

- TCP-Reno, last received ACK = 5000, segment size = 1000



Cumulative and Selective ACKs

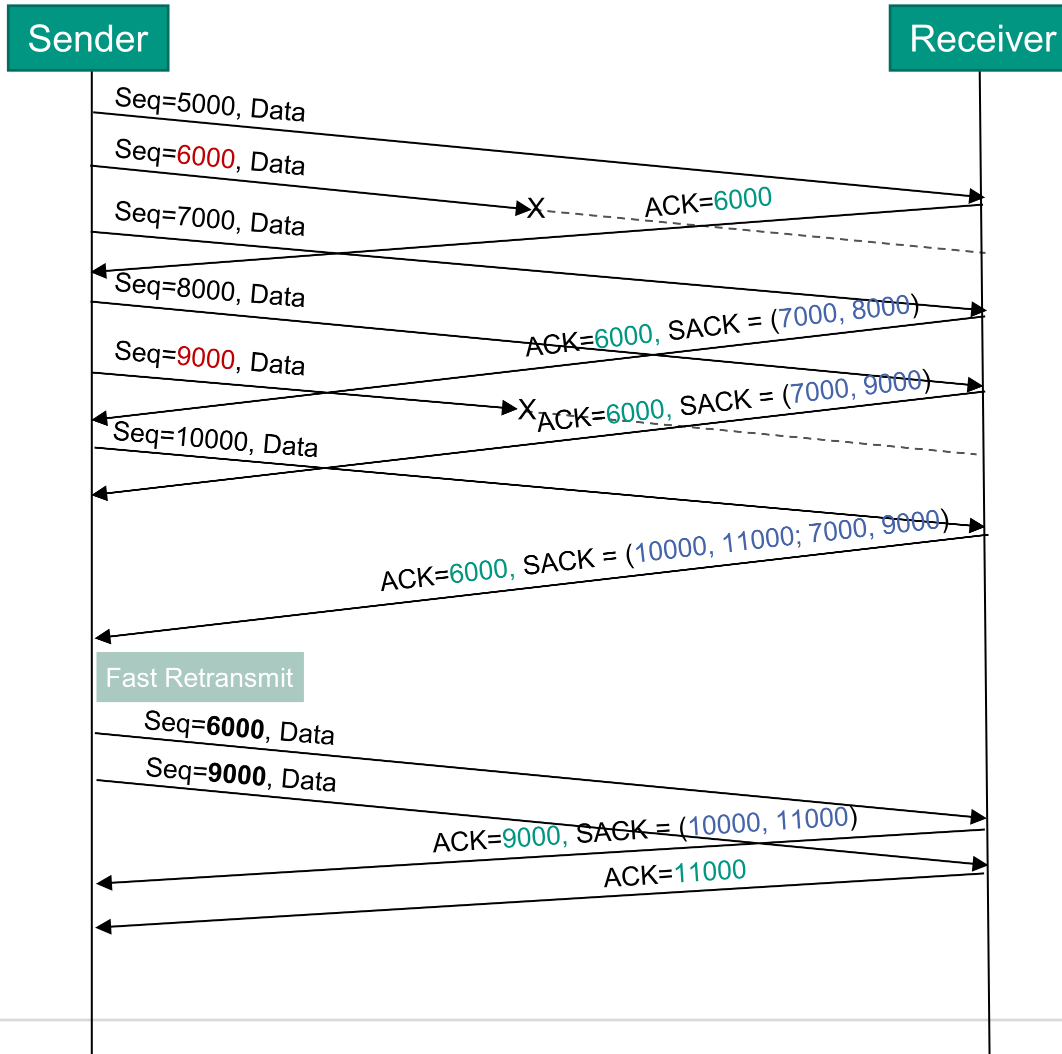
- TCP-Reno, last received ACK = 5000, segment size = 1000



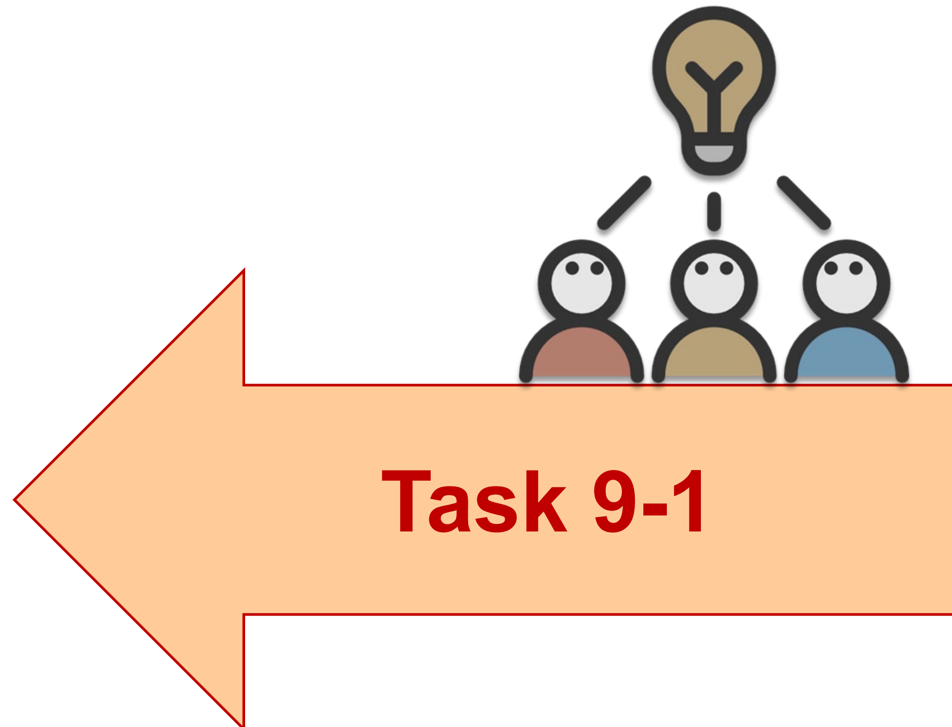
Based on RFC2018 "TCP Selective Acknowledgment Options"

Cumulative and Selective ACKs

- TCP-Reno, last received ACK = 5000, segment size = 1000



Based on RFC6675
“A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP”


Task 9-1

Karlsruher Institut für Technologie

9.1.3 “Option: Packet Mood”

TCP “Option Packet Mood”

- RFC 5841, 01.04.2010

> Packets cannot feel. They are created for the purpose of moving data from one system to another. However, it is clear that in specific situations some measure of emotion can be inferred or added. For instance, a packet that is retransmitted to resend data for a packet for which no ACK was received could be described as an 'angry' packet, or a 'frustrated' packet (if it is not the first retransmission for instance).

```
+--------+--------+--------+--------+
|00011001|00000100|00111010|00101001|
+--------+--------+--------+--------+
 Kind=25  Length=4 ASCII :  ASCII )

+--------+--------+--------+--------+--------+
|00011001|00000101|00111110|00111010|01000000|
+--------+--------+--------+--------+--------+
 Kind=25  Length=5 ASCII >  ASCII :  ASCII @
```

```
ASCII          Mood
=====          ====
:)             Happy
:(             Sad
:D             Amused
%(             Confused
:o             Bored
:O             Surprised
:P             Silly
:@             Frustrated
>:@            Angry
:|             Apathetic
;)             Sneaky
>:)            Evil
```

Karlsruher Institut für Technologie

9.1.4 Extension SYN Cookies

SYN Flooding

Karlsruher Institut für Technologie

- Denial-of-Service (DoS) attack on servers running TCP
  - Client(s) send huge amounts of SYN segments to a server without any desire of establishing a connection
    - Use forged IP source addresses
    - Server will not receive a valid ACK segment
- Problem
  - Server starts establishing a connection, i.e., keeps state information after receiving a SYN segment
    - Client IP address, client port, MSS …
  - Small queue of half-opened connections (missing ACK) maintained

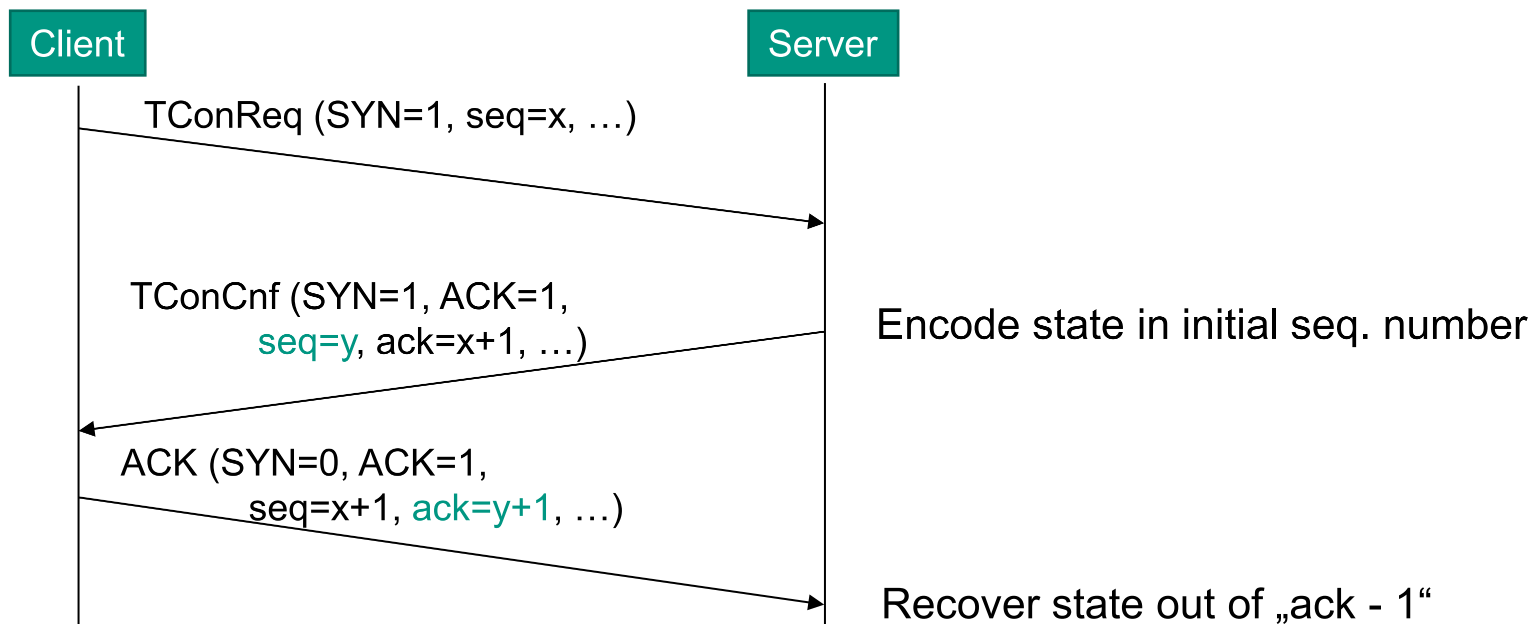
  → Result: exhaustion of resources at server
    - Queue fills up
    - Memory exhausted
    - No further SYN segments can be received
- Solution: SYN cookies

SYN Cookies

- Strategy
  - Avoid keeping state at server as long as possible
    → No state is kept after receiving a SYN segment
  - Server checks validity of SYN/ACK segments solely based on information they carry
    - No stored state information at server needed

SYN Cookies

- Approach
  - Encoding of minimal necessary state within the initial sequence number (32 bit) of the server
    - Client provides state within ACK field of corresponding ACK segment
    - Transparent to clients since initial sequence number is chosen randomly



[RFC4987]

Simple Enconding Example

- On receipt of SYN from client
  - 5 bit: timestamp = t
    - Prevents replay attacks
  - 3 bit: MSS chosen by server = m
    - Limited to 8 predefined values
  - 24 bit: (krypto-)hash over client IP address + port, server IP address + port, timestamp = c
    - Verifies that SYN was actually performed by client

  →32 bit: initial sequence number ($\mathrm{t}|\mathrm{m}|c = y$) of server in SYN/ACK segment

- On receipt of ACK from client
  - ACK $= y + 1 \rightarrow y = ACK - 1$
  - Decode y into t, m, c
    - Is t still valid?
      - Otherwise drop packet
    - Recalculate hash, compare with c
      - If not identical: drop packet
    - Use MSS given in m
  - Continue as usual

Encoding is implementation specific – transparent to client.

Discussion

Karlsruher Institut für Technologie

- Advantage: avoids state keeping at server
  - But: requires computation
- Drawbacks
  - TCP-Options negotiated in the initial SYN are lost
    - … as the receiver does not store the initial SYN to save resources
    - Window scaling
    - Selective acknowledgements
    - …
  - Limits possible values for MSS to predefined values
    - Encoded into the Cookie
- Thus: use SYN cookies only if necessary
  - Filled up queue
  - Resource bottleneck

Pingo 09-01

Karlsruher Institut für Technologie

- Selective acknowledgements (SACK)
  - Replace standard TCP acknowledgements
  - Acknowledge out-of-order data
  - Acknowledge each TCP segment individually
  - Must be negotiated during connection establishment

https://pingo.coactum.de/005694

Pingo 09-02

- What is required for TCP SYN cookies?

  - Additional memory
  - Cryptographic hash functions
  - Explicit client-side support
  - Additional TCP options

https://pingo.coactum.de/005694

Karlsruher Institut für Technologie

9.2 TCP in Networks with High BDP

TeleMatics

Congestion Control Evolution

Karlsruher Institut für Technologie

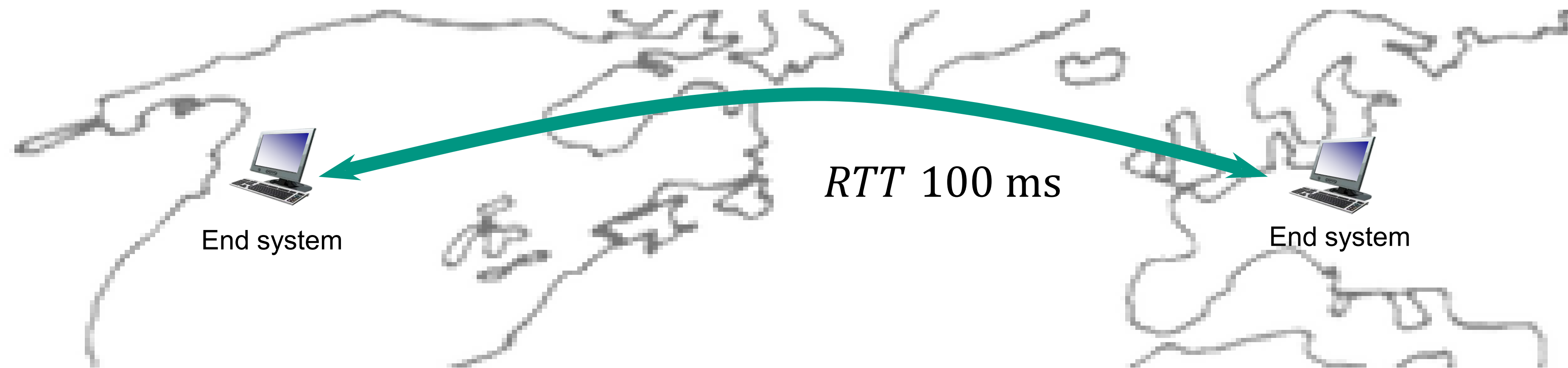
- Phase 1: Network Overload (1990´s) tm-07
  - Problem: High loss rate → Congestion Collapse
  - Solution: Introduction of first Congestion Control Algorithms
  - TCP Tahoe & TCP Reno
- Phase 2: Inefficient Bandwidth Usage (2000´s) tm-09
  - Increasing link capacity
  - Problem: Tahoe & Reno approach link capacity rather slowly → Network underloaded
  - New: TCP CUBIC & BBR
- Phase 3: Reducing Response Time (since 2010´s) tm-09
  - Web-Browsing
    - Typically many small transmissions, interactive application
  - Flow completion time
    - Highly influences perceived quality of experience (QoE) by users
  - Problem
    - Connection finishes before actual data rate reaches link capacity
  - New: TCP Fast Open, QUIC, …

[Welz22]

9.2.1 Scalability Issues

Networks with High Bandwidth-Delay Products

- Characteristic: they "buffer" noticeable amounts of data
- Basic scenario



- Round-trip time $RTT = 100\ ms$
- Data rate $D = 10\ Gbit/s$
- Size of TCP segments $MSS = 1500\ byte$

- Flow control: size of flow control window
- Congestion control: size of congestion window

Maximum Application Data Rate?

Karlsruher Institut für Technologie

- In the scenario on the previous slide
  - Limited by maximum size of flow control window
    - Receive window field in TCP header: 16 bit
      → $\text{max. receive window size} = 65535 \text{ byte}$

  - Remember
    - Maximum amount of data (M) that can be send without receiving acknowledgements
      - $M = min\{congestion\ control\ window, flow\ control\ window\}$

  - Neglect congestion control window for the moment
    - Maximum achievable application data rate ($\max \text{App}D$, goodput)
    - $\max \text{App}D = \frac{M}{RTT} = \frac{65535 \text{ byte}}{100\ ms} = 655350 \frac{\text{byte}}{\text{s}} \approx 5{,}2\ Mbit/s$

      → $max\ AppD$ limited by flow control window to about 5,2 Mbit/s

Window Scaling

- Header field receive window 16 bit
  - Flow control allows maximal 65535 bytes in flight (unacknowledged)
    - Even if receiver could receive more
  - ACKs arrive only after one round-trip time (RTT)
- TCP option: Window Scaling
  - Header field receive window remains unchanged (16 bit)
  - Scaling factor can be changed
    - E.g., measure window size in 32 bit words instead of bytes
  - Option is negotiated during connection establishment
    - Within SYN and SYN/ACK segments
  - Scaling factor remains unchanged during lifetime of a TCP connection

[RFC1323]

What about Congestion Window?

Karlsruher Institut für Technologie

- Necessary size of congestion window ($CWnd$) to fully utilize available resources

$$CWnd = \frac{D \cdot RTT}{MSS} = \frac{10\frac{Gbit}{s} \cdot 100\ ms}{1500\ byte} = 83.333{,}33 \quad \text{TCP segments}$$

- What about the required packet loss rate?
  - According to the periodic model

$$D = \frac{1}{RTT}\sqrt{\frac{3}{2p}} \cdot MSS \qquad p = 1{,}5 \cdot \left(\frac{MSS}{RTT \cdot D}\right)^2 = 1{,}5 \cdot \left(\frac{1500\ byte}{100\ ms \cdot 10\frac{Gbit}{s}}\right)^2 \approx 2 \cdot 10^{-10}$$

→ At most 1 packet loss each 5.000.000.000 TCP segments!

Time to Open the Congestion Window?

Karlsruher Institut für Technologie

- Window sizes above 83.333 TCP segments required to achieve maximum data rate

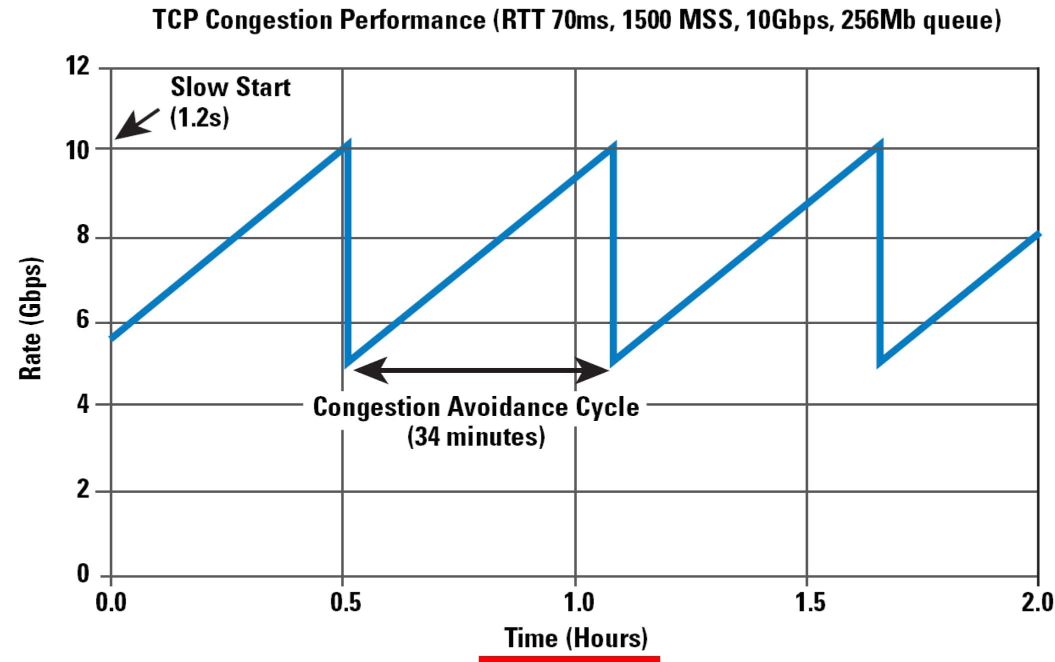
- Congestion window increase through congestion avoidance
  - Increase of 1 MSS (1 TCP segment) per RTT
  - Assumption: start at $CWnd = 41.666$

  - Time t until $CWnd = 83.333$ MSS is reached?
    - Assume: $RTT = 100\ ms$

$$t = 41.666 \cdot RTT = 4166600\ ms \approx 4167\ s \ \approx \mathbf{70\ min}$$

Observed TCP Behavior

- Simulation results



- Observations
  - No packet loss or bit errors in about 34 minutes
    - Bit error rates of less than 10^{-14} required
  - Data transferred in one cycle: 1,95 terabyte
    - Average data rate: 7,5 Gbit/s

→ Congestion avoidance not suited for such networks

[Hous06]

Observation

Karlsruher Institut für Technologie

- It can take very long until the available data rate is fully utilized

- Cause
  - Very conservative behavior of congestion avoidance
    - Congestion window grows by one MSS per RTT
    - Slow window growth in congestion avoidance causes low average data rate

  → Not efficient in networks with high bandwidth-delay products

- Required
  - Faster increase of the congestion window in congestion avoidance

TeleMatics

Karlsruher Institut für Technologie

9.2.2 CUBIC TCP

Overview of presented Congestion Controls

Karlsruher Institut für Technologie

Chapter 7

TCP Tahoe
historical algorithm,
first congestion control

TCP Reno
Improvement of TCP Tahoe

Introduce general „mechanics“ behind Internet congestion control

Chapter 8

DCTCP
Congestion control for data centers

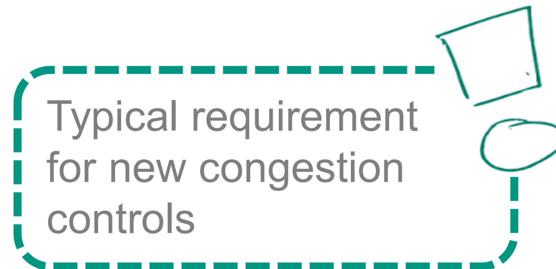
Chapter 9

CUBIC TCP
Improved congestion control for networks with high BDP

Current standard used in many operating systems

CUBIC TCP

- Goals
  - Provide simple algorithm for networks with high bandwidth-delay product
  - TCP-friendly
    - Behaves like standard TCP (i.e., TCP Reno) in networks with short RTTs and low data rate
  - Congestion avoidance
    - Applies cubic function instead of linear window increase
  - Performance should not be worse than TCP Reno



- In comparison to TCP Reno
  - Better RTT fairness
    - Window growth independent of RTT
  - Better scalability to high data rates

- Currently default congestion control in most major operating systems
  - Originally implemented in Linux (Kernel 2.6.16, 2006)
  - Adopted by Apple in MacOS/iOS
  - Windows finally also switched to CUBIC in 2017

Different Types of Fairness

- Intra protocol fairness
  - All senders use same TCP variant
  - Goal: All flows should achieve same data rate
- With new TCP variants: inter protocol fairness
  - Can flows with different congestion control coexist?
  - Striving for identical data rates not always useful
- Furthermore: RTT fairness
  - Fairness among TCP flows with different RTTs
  - Especially challenging in fast networks

CUBIC TCP

Karlsruher Institut für Technologie

- Differs only slightly from TCP Reno
  - Still follows the same design principles
    - Window-based congestion control
    - Congestion window is increased only on ACK receipt
    - Slow start remains the same
    - Fast recovery remains the same
  - Changes address the congestion avoidance phase
    - Faster increase of congestion window
      - Does not apply AIMD from TCP Reno
      - Cubic function
    - Congestion window increase is independent from RTT

Congestion Window Increase

Karlsruher Institut für Technologie

- Independent from RTT
  - Use of actual time t: elapsed time from the beginning of the current congestion avoidance
    - Window growth depends on time between consecutive congestion events

- Apply cubic function
  - Modification of AIMD principle

$$W_{cubic}(t) = C(t - K)^3 + W_{max} \qquad \text{with } K = \sqrt[3]{\frac{W_{max}(1-\beta)}{C}}$$

C: predefined constant that determines aggressiveness of increase

β: multiplicative decrease of congestion window

Note: $\beta = 0{,}5$ for TCP-Reno, $\beta = 0{,}7$ for CUBIC TCP

W_{max}: congestion window size at latest congestion incident

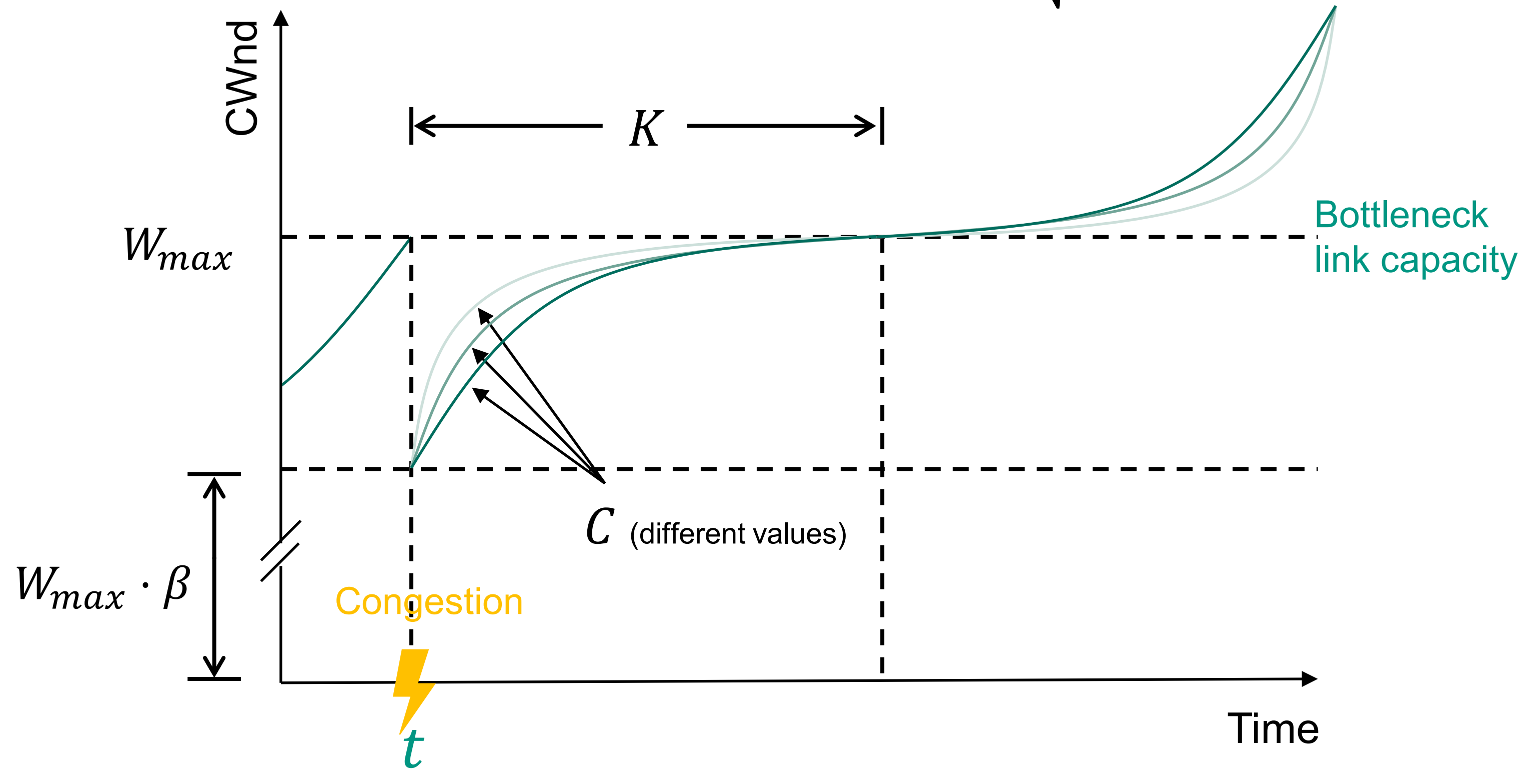
K : time needed to increase current window to W_{max} (no further congestions)

[RFC8312]

TeleMatics

Cubic Function

- $W_{cubic}(t) = C(t - K)^3 + W_{max}$ with $K = \sqrt[3]{\frac{W_{max}(1-\beta)}{C}}$



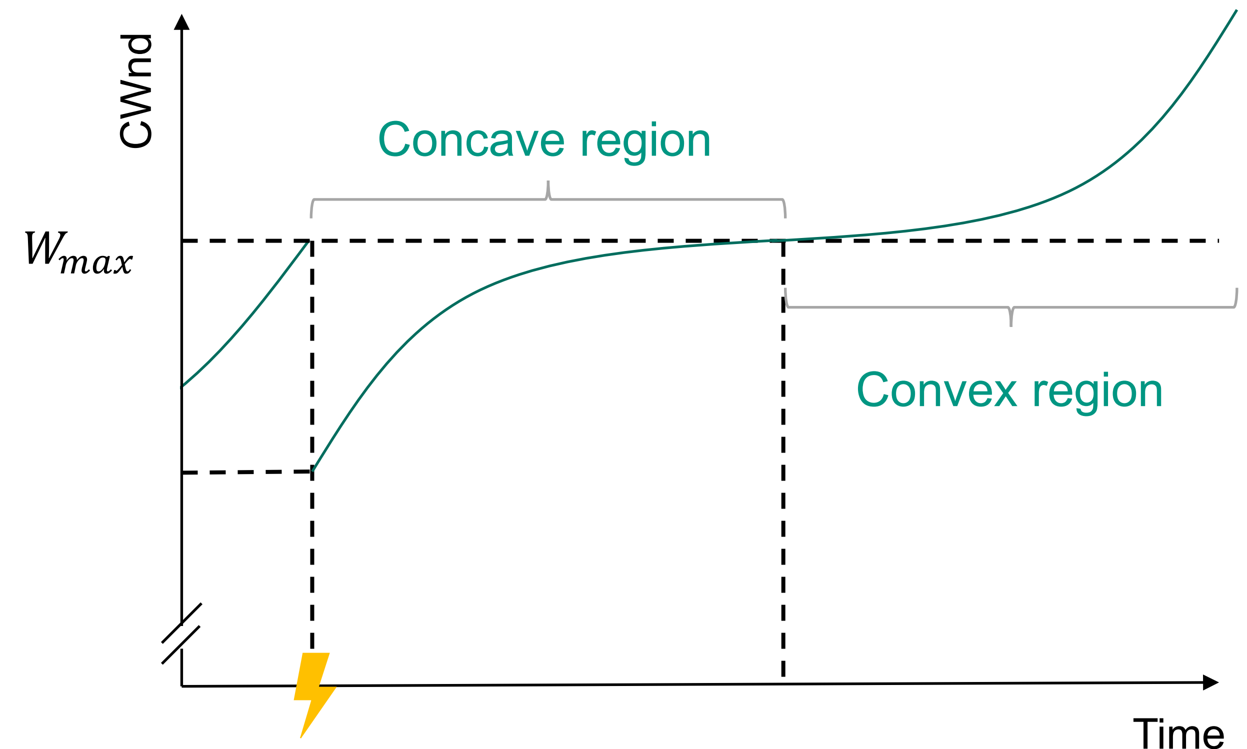
Window increase depends on distance between current time t and K.
Increases are larger when t is further away from K.

- Note: $CWnd \approx W_{max}$: very conservative increase
  - „Plateau“ around $CWnd \approx W_{max}$
- Recommended values (RFC8312): $C = 0.4$ and $\beta = 0.7$

Three CUBIC Modes

Karlsruher Institut für Technologie

- Concave region
  - $CWnd < W_{max}$ and not in TCP-friendly region
- Convex region
  - $CWnd \geq W_{max}$ and not in TCP-friendly region
- TCP-friendly region
  - .. do not perform worse than TCP Reno
  - Ensures that CUBIC achieves at least same data rate as TCP Reno in networks with small RTT and low data rate



TeleMatics

Concave Region

Karlsruher Institut für Technologie

- “Steady-State behavior”
  - Start with large increase
  - Reach previous capacity W_{max} quickly
  - Small increases when approaching W_{max}
  - Remain around W_{max} as long as possible

- Receiving an ACK in congestion avoidance
  - and not in TCP-friendly region
  - and $CWnd < W_{max}$

→ concave region

  - $\text{CW}nd = \text{CW}nd + \frac{W_{cubic}(t+RTT)-CWnd}{CWnd}$
    - For each received ACK

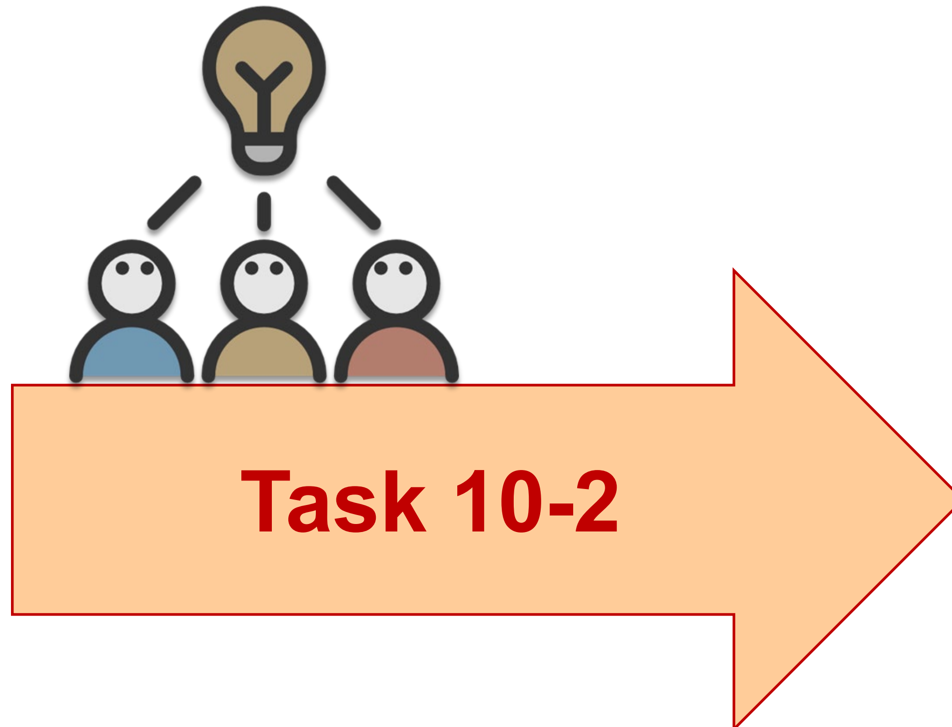
Convex Region

- "Maximum probing phase"
  - Bottleneck bandwidth increased … searching for new W_{max}
  - Increases very slowly at the beginning
  - Gradually increases its increase rate
  - Similar to slow start behavior

- Receiving an ACK in congestion avoidance
  - and not in TCP-friendly region
  - and $CWnd \geq W_{max}$

  → convex region

  - $\text{CW}nd = \text{CW}nd + \frac{W_{cubic}(t+RTT)-CWnd}{CWnd}$
    - For each received ACK


Task 10-2

CUBIC

Task

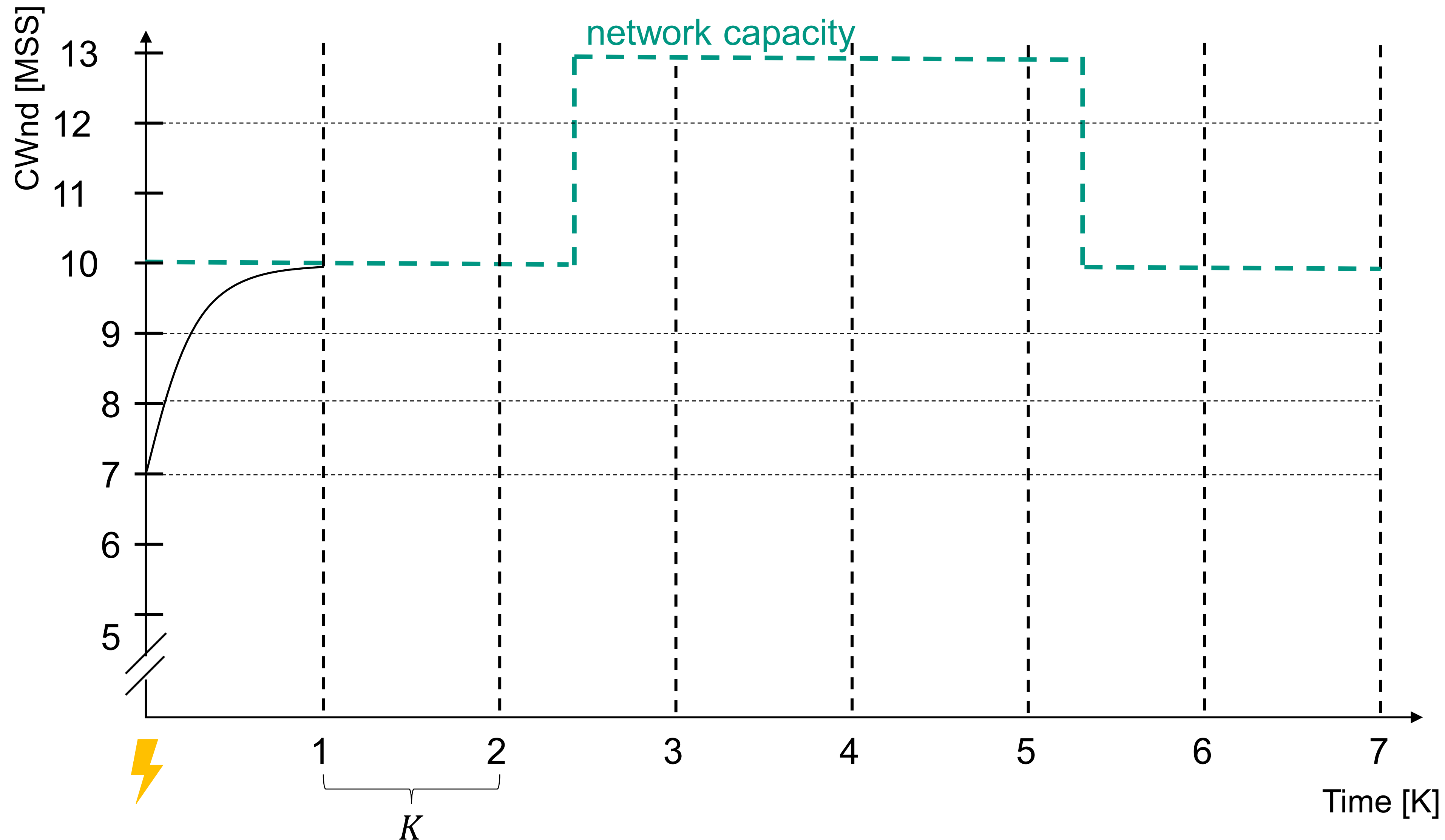
Karlsruher Institut für Technologie

- Scenario
  - A connection uses CUBIC TCP
    - Recall the CUBIC formula: $W_{cubic}(t) = C(t - K)^3 + W_{max}$
  - The available network capacity varies over time
    - First 10 MSS/s, then 13 MSS/s, then again 10 MSS/s
  - Assume a minor congestion signal when $CWnd$ reaches network capacity

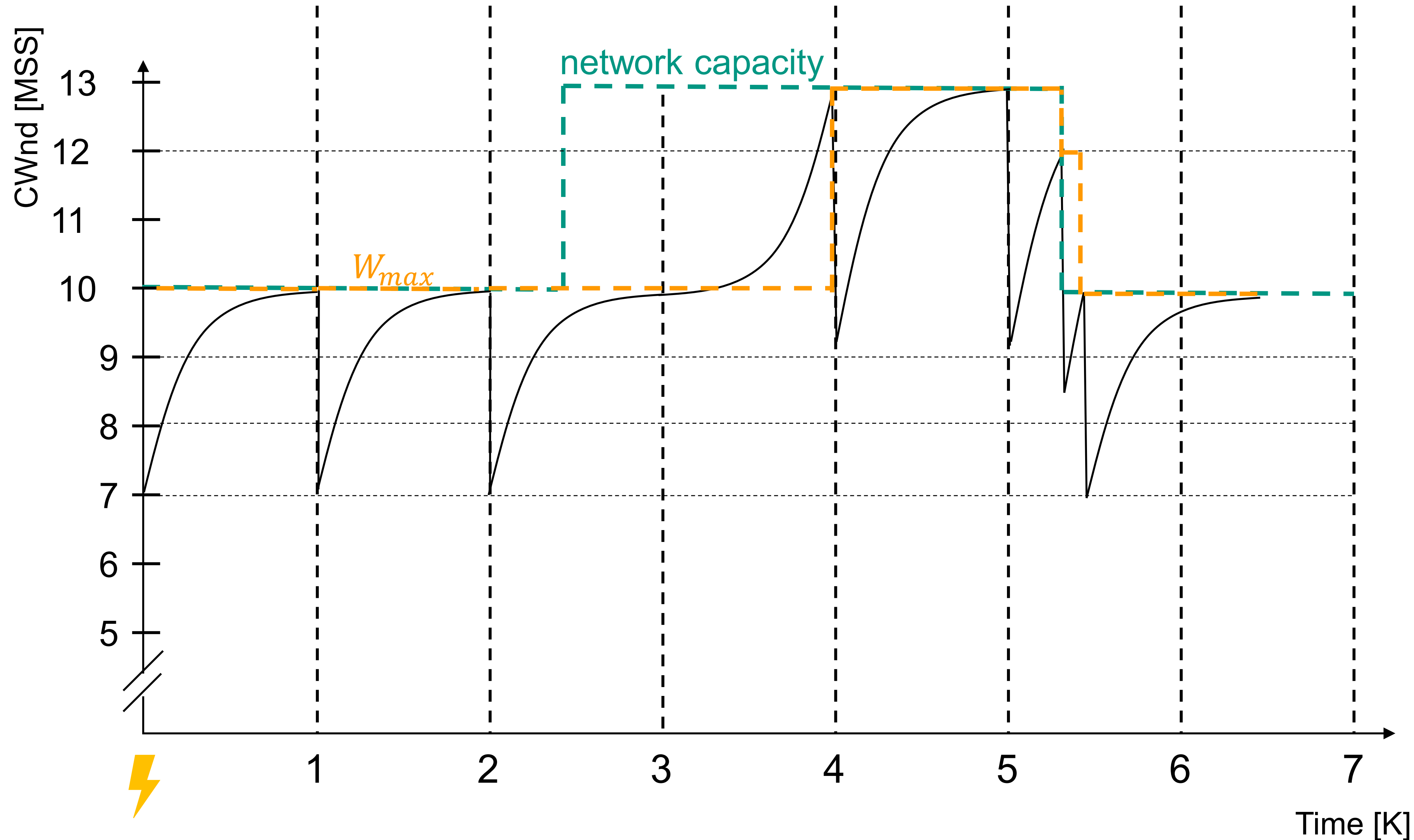
- Task
  - Draw the evolution of the congestion window $CWnd$ over time
    - Use the available K intervals when possible
  - Draw the evolution of W_{max} over time
    - How and when does the parameter W_{max} change?

  - At $t = 0$
    - $CWnd = 10\ MSS$, minor congestion signal observed

Evolution of CUBIC CWnd



Evolution of CUBIC CWnd – Solution



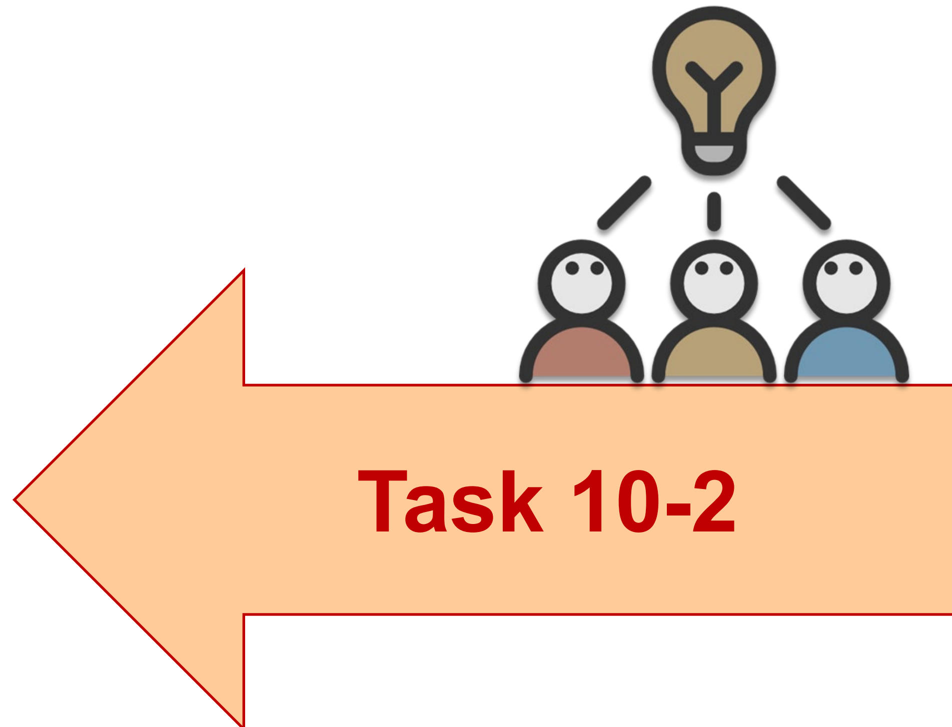
Evolution of CUBIC CWnd – Solution

Karlsruher Institut für Technologie

- How and when does the parameter W_{max} change?

- W_{max} is updated after each congestion event
- W_{max} is set to the value of W_{cubic} directly before the congestion event

TeleMatics


Task 10-2

TCP-friendly Region

- Observation: in networks with small RTTs and low data rates
  - Cubic´s congestion window grows slower than with TCP Reno
  - Approach: “emulation” of TCP Reno (which uses AIMD)
- AIMD (α, β)
  - α: additive increase factor
    - $W = W + \alpha$
  - β : multiplicative decrease factor
    - $W = \beta \cdot W$
- TCP-fair increment: $\alpha = 3 \cdot \frac{1-\beta}{1+\beta}$
  - AIMD $(\frac{3\cdot(1-\beta)}{1+\beta}, \beta)$ achieves same W_{avg} as AIMD $(1, \frac{1}{2})$
  - Average data rate of AIMD calculates to $\frac{1}{RTT}\sqrt{\frac{\alpha\cdot(1+\beta)}{2\cdot(1-\beta)\cdot p}}MSS$ p : loss rate
  - With $\alpha = 1$ and $\beta = 1/2$ average data rate $\frac{1}{RTT}\sqrt{\frac{3}{2\cdot p}}MSS$

[RhXu08,RFC8312]

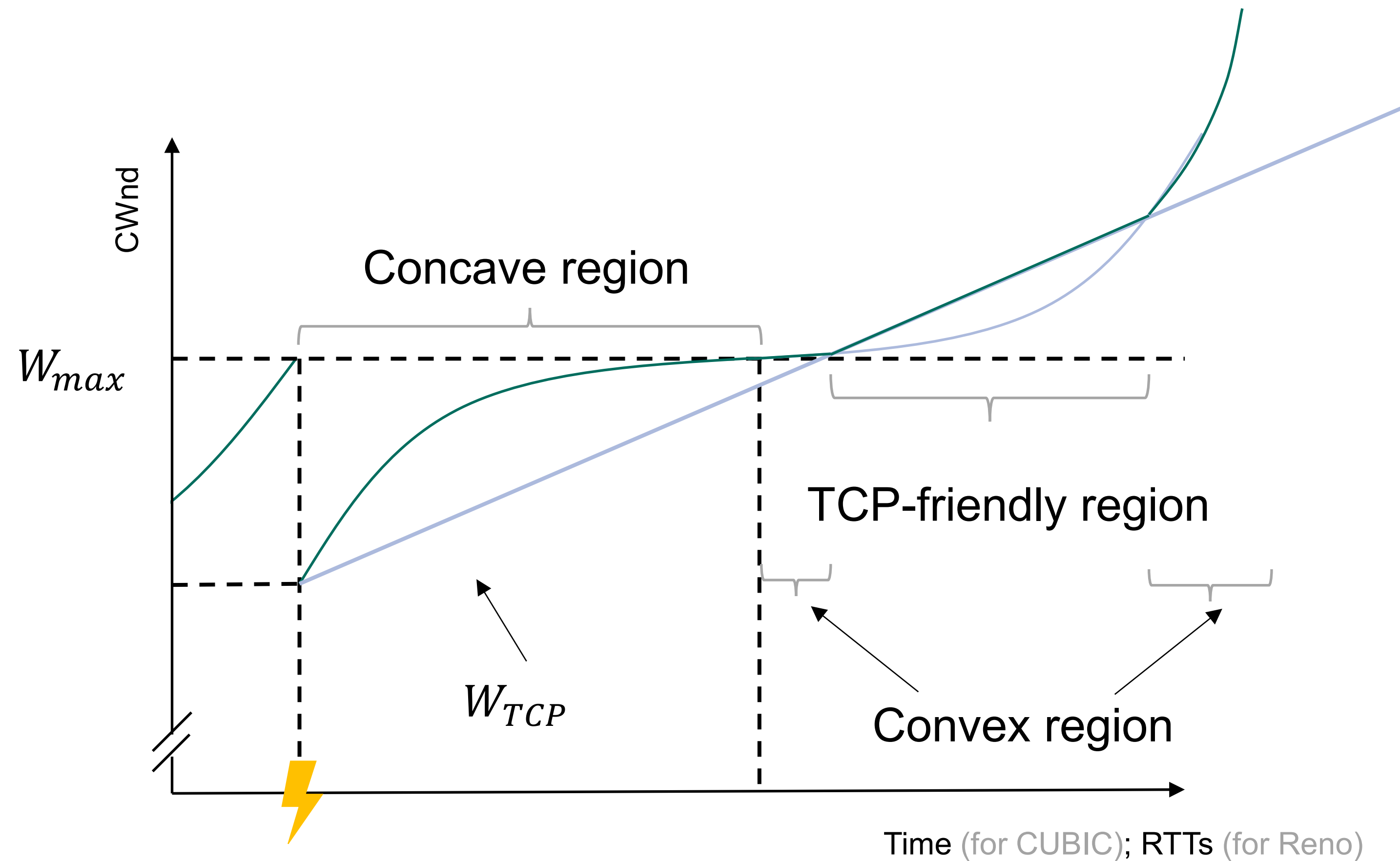
TCP-friendly Region

Karlsruher Institut für Technologie

- Window size of emulated TCP at time t
  - $W_{TCP} = W_{max} \cdot \beta + \frac{3 \cdot (1-\beta)}{1+\beta} \cdot \frac{t}{RTT}$
- Recall window size of TCP cubic
  - $W_{Cubic} = C(t - K)^3 + W_{max}$
- If
  - $W_{CUBIC} < W_{TCP}$ then
    - CWnd is set to W_{TCP} each time an ACK is received
  - otherwise
    - CWnd is set to W_{Cubic} each time an ACK is received

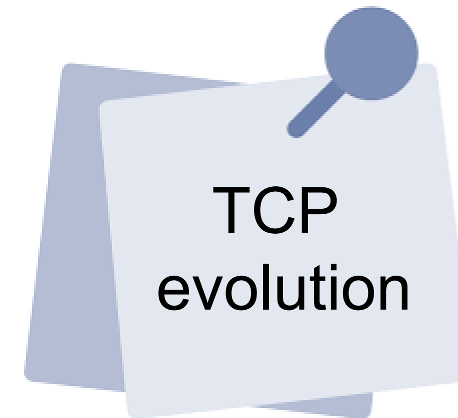
CUBIC Modes with TCP-friendly region

Karlsruher Institut für Technologie



TeleMatics

Homework



9.3 TCP and Response Time

Congestion Control Evolution

Karlsruher Institut für Technologie

- Phase 1: Network Overload (1990´s) — tm-07
  - Problem: High loss rate → Congestion Collapse
  - Solution: Introduction of first Congestion Control Algorithms
  - TCP Tahoe & TCP Reno
- Phase 2: Inefficient Bandwidth Usage (2000´s) — tm-09
  - Increasing link capacity
  - Problem: Tahoe & Reno approach link capacity rather slowly → Network underloaded
  - New: TCP CUBIC & BBR
- Phase 3: Reducing Response Time (since 2010´s) — tm-09
  - Web-Browsing
    - Typically many small transmissions, interactive application
  - Flow completion time
    - Highly influences perceived quality of experience (QoE) by users
  - Problem
    - Connection finishes before actual data rate reaches link capacity
  - New: TCP Fast Open, QUIC, …

[Welz22]

Karlsruher Institut für Technologie

9.3.1 Basic Situation

TELEMATICS

TCP & WWW (World Wide Web)

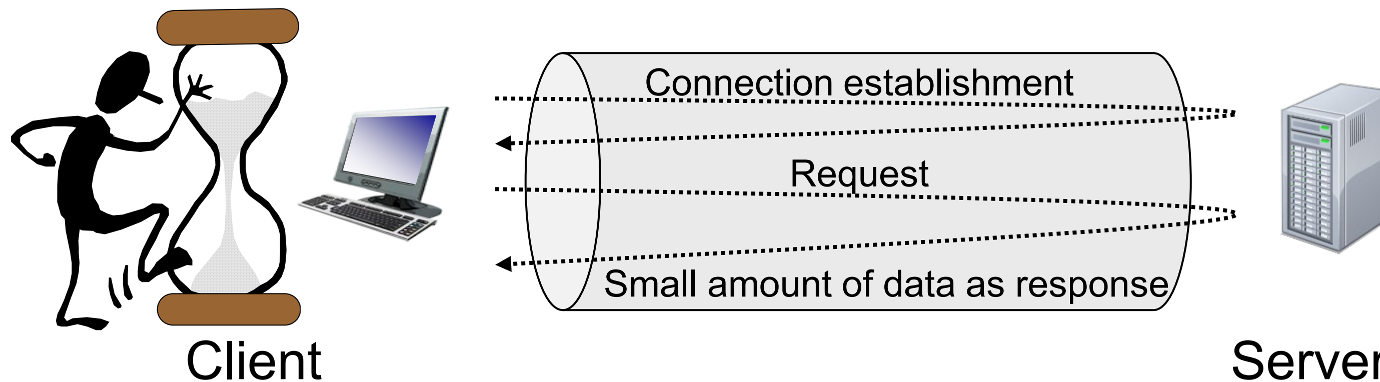
Karlsruher Institut für Technologie

- Web uses HTTP(s), which uses TCP
  - On every click in the Web: setup of (at least) one TCP connection
  - Often … one TCP connection per object
    - Hence, only for a few kilobytes of data per TCP connection
  - … is an *interactive service*
- *Response time* counts!
  - … how long does it take until an object is loaded?
    - Perceived reaction time of web-based services
- Communication latencies (and their fluctuations) can hardly be hidden from users!
  - Transmission time t_s significantly reduced over the years
  - Propagation delay t_a … limited by speed of light
  - … and recall: queueing time in buffers also increases latencies

TCP Response Time

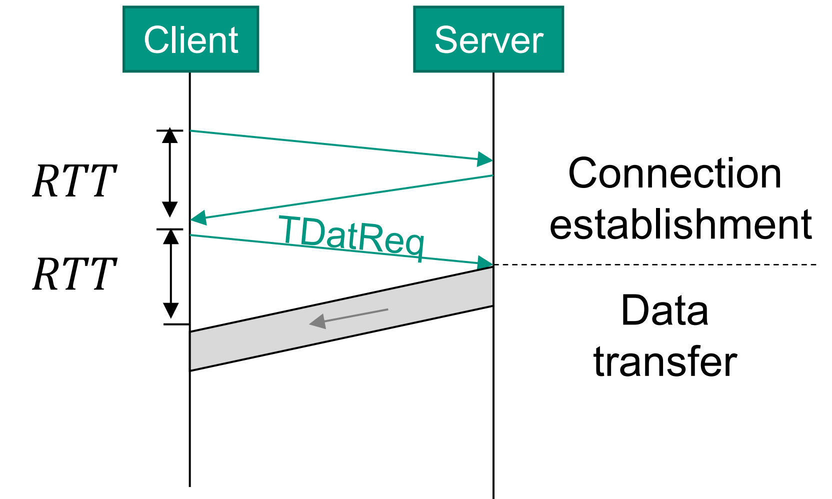
- Time between initiation of a TCP connection and receipt of the requested data

- Important components
  - Handshake of TCP connection establishment
  - Slow start
  - Transmission of the object



TCP Response Time: Coarse Analysis

- Response time without applying congestion control
  - After 1st RTT
    - Client sends object request
  - After 2nd RTT
    - Client begins to receive object data
  - Minimal time for object transmission $t = \frac{object\ size\ O}{data\ rate\ D}$
    - Note: Server may not be able to send all TCP (data) segments back to back (e.g., CWnd too small)
      - Small windows create pauses (waiting for ACKs)
      - Caused, e.g., by initial window of slow start
      - … this „consumes“ RTTs and, thus, increases response time
  - → $Response\ time\ \geq 2\ RTT + \frac{O}{D}$
- With small objects, response time can be dominated by RTTs



Observation

Karlsruher Institut für Technologie

- Majority of TCP connections in the Web has short lifetime
  - Mostly, transmission of a single object … few kilobytes
  - Whole transmission during TCP slow start

  → Slow start has significant impact on response time

- Goals
  - Avoid „empty“ RTTs without data transport
  - Reduce RTTs needed for slow start

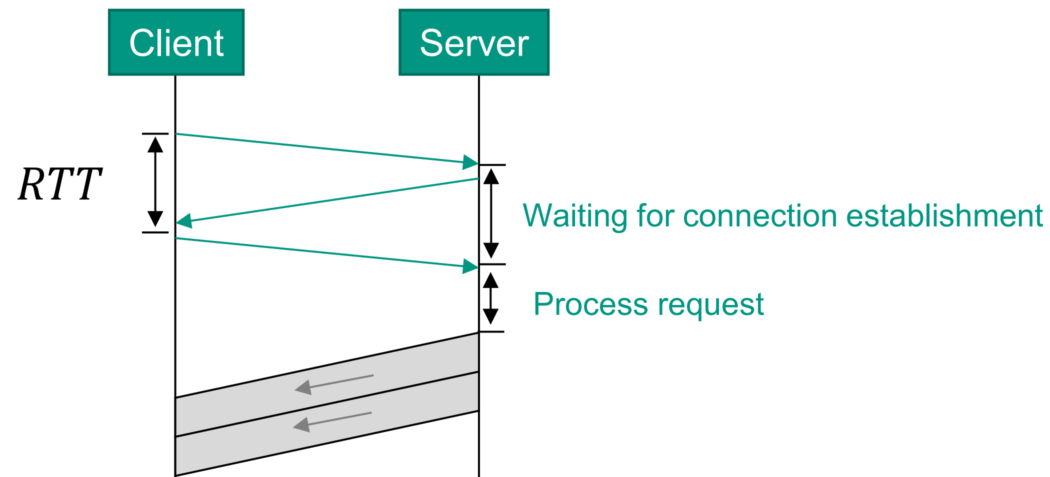
- … but still consider
  - Fairness
  - Conservation of packets

Karlsruher Institut für Technologie

9.3.2 TCP Fast Open

TCP Fast Open (TFO)

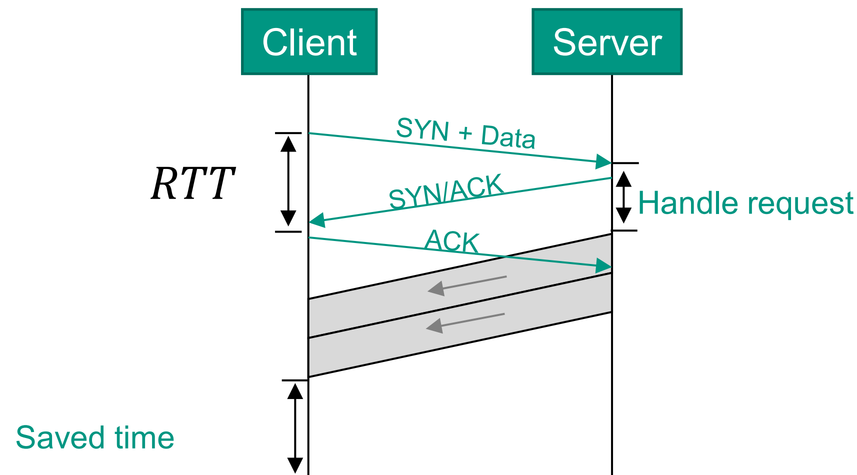
- Goal
  - Reduce delays that precede the transmission of an object
- Initial situation
  - Application can only react after 3-way handshake



- Note
  - SYN segment can carry data
  - … but data, will not be delivered to application before ACK was received

Naïve Idea

- Send data with SYN segment and deliver it to application immediately
  - Reduces latency by one RTT



- But: opens attack surface for several problematic situations
  - DoS attacks with SYN flooding
  - Source address spoofing

  → Security mechanisms necessary

TFO Cookie

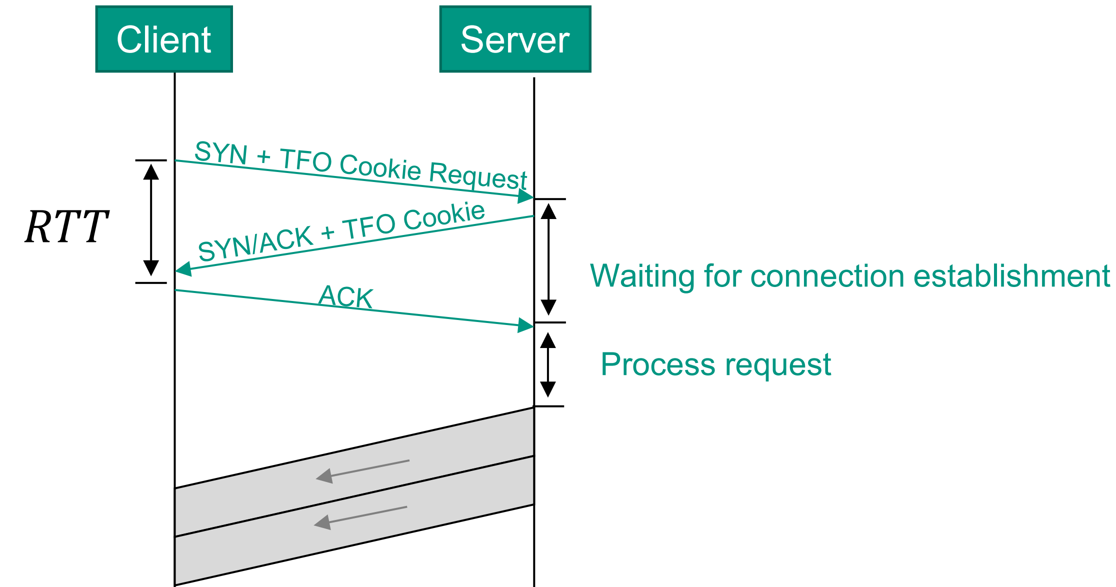
Karlsruher Institut für Technologie

- Goal
  - Avoid DoS attacks
    - Disallow sending data with first SYN segment of first connection establishment to a server
    - Establish cookie for subsequent connections
- Cookie → avoids state keeping at server
  - Used to authenticate client´s source IP address of SYN segment
    - Cookie is a MAC tag (Message Authentication Code)
    - Implementation example: encrypt IP address with AES 128
  - Can only be generated by the server
  - Times out after certain amount of time
    - Periodically change secret server key or
    - Include timestamp when generating cookie
- Basic steps
  1. Client requests TFO cookie from server
     - „Regular“ TCP connection with TCP option TFO
  2. Client uses TFO cookies in subsequent TCP connections

[RFC7413]

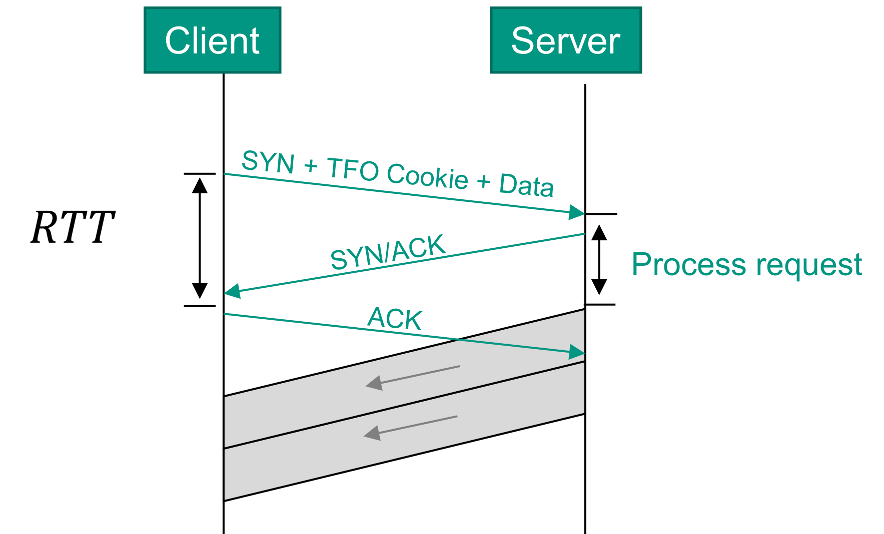
Step 1

- Client does not yet have a TFO cookie
  - „Regular“ connection establishment with TCP option TFO
    - Cookie request in SYN segment of client
    - Server encrypts client IP → cookie
    - Server sends cookie in SYN/ACK segment to client
    - Client saves cookie for later use
    - After receipt of ACK, server can regularly send data



Step 2

- Client already has TFO cookie
  - Client provides cookie with SYN segment
  - Server validates cookie
    - e.g., calculates new cookie, compares it with received cookie
    - If cookie is valid
      - Immediately deliver data to application
    - Else
      - Proceed as usual



Kahoot! 09-02

Karlsruher Institut für Technologie

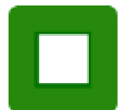
1 - Quiz

With SACKS in TCP

Cumulative ACKs are no longer used

Every single packets is acked by a SACK

Multiple ranges of sequence numbers can be acked per TCP segment


Retransmissions are no longer needed

Kahoot! 09-02

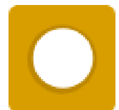
Karlsruher Institut für Technologie

2 - Quiz

SYN cookies

Are used during TCP connection establishment

Are a TCP option


Are compatible with TCP SACK option

Should be used as often as possible

3 - Quiz

CUBIC TCP

Uses a loss-based congestion control

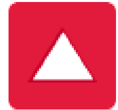
Uses a rate-based congestion control

Uses modified ECN

Scales well to higher data rates

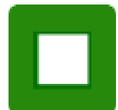
4 - Quiz

TCP Fast Open (TFO) Cookies ...


are generated by clients

avoid denial-of-service attacks

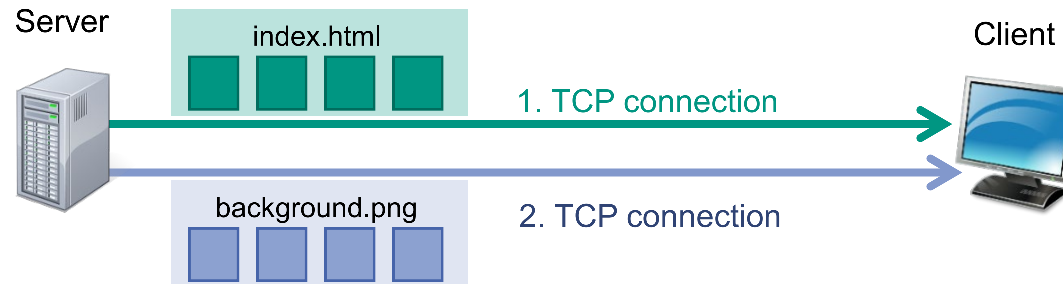
are used for a single TCP connection


allow sending data with SYN segments

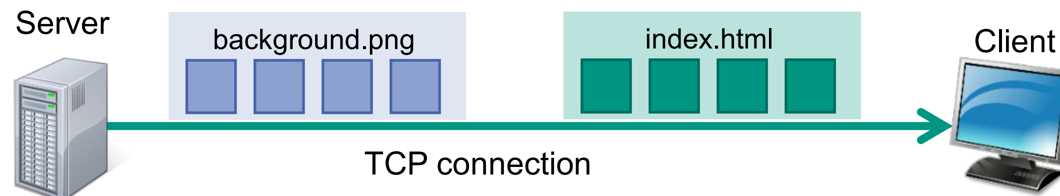
9.3.3 HTTP/2

Evolution of HTTP

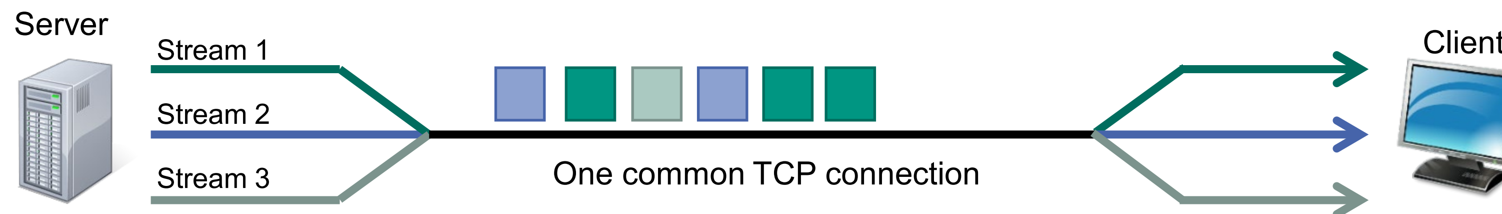
- HTTP/1.0: one connection per HTTP request



- HTTP/1.1: pipelining - multiple requests per connection

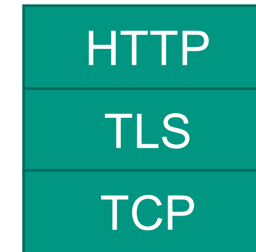


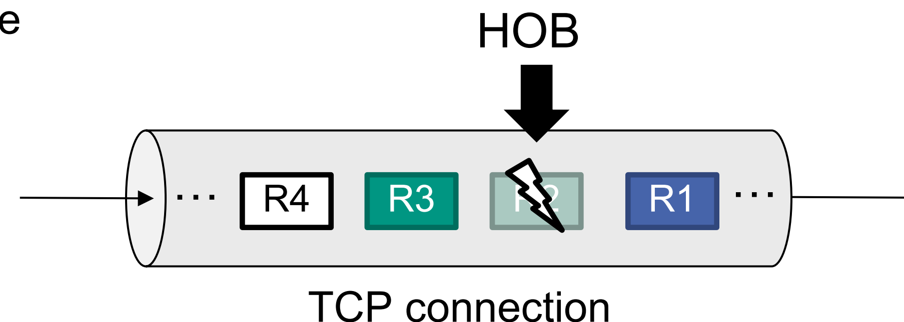
- HTTP/2: multiple streams multiplexed over a common TCP connection



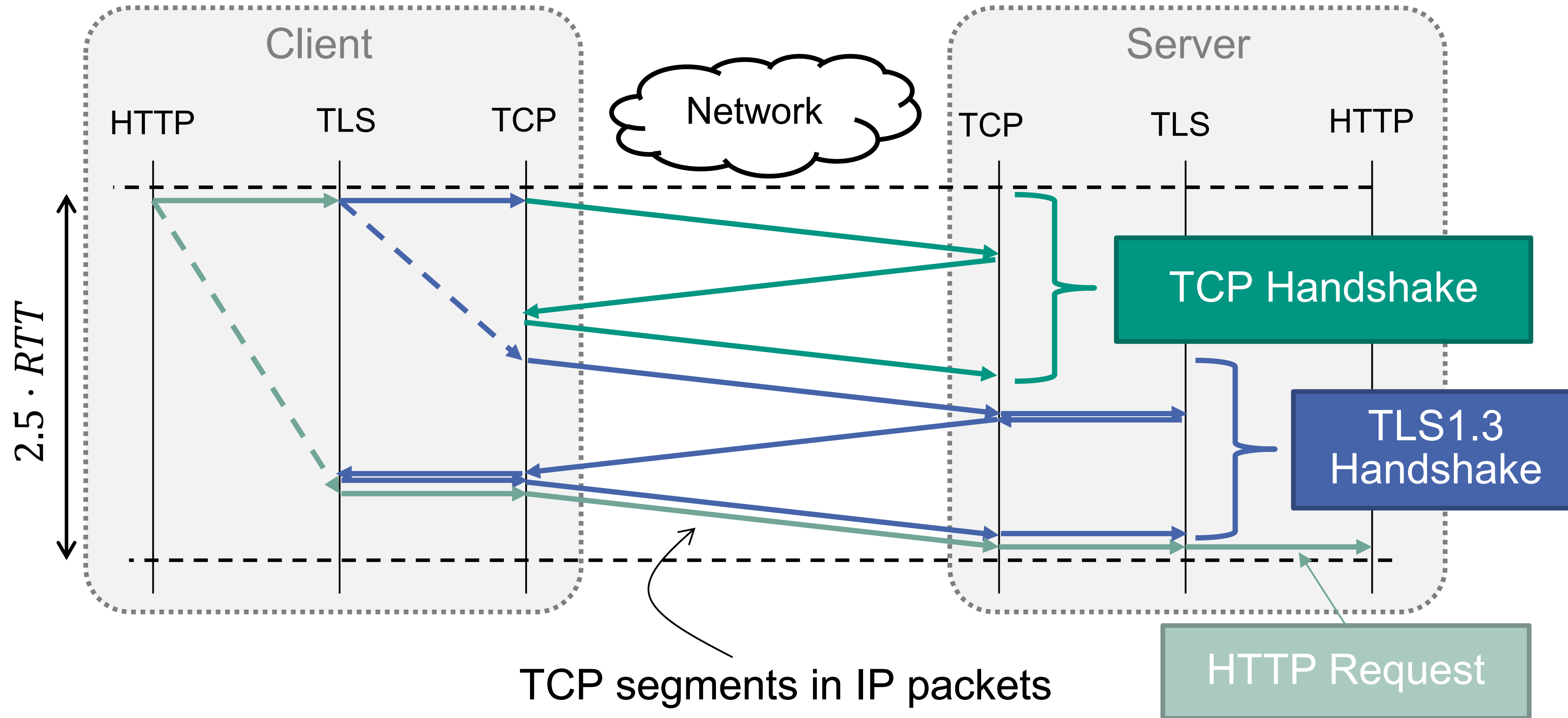
HTTP/1.x

- HTTP/1.0
  - Opens separate TCP connection for each requested object
    - High response time
    - Often many TCP connections are used in parallel
    - Non-optimal network resource usage
- HTTP/1.1
  - Pipelining possible
    - Requests can be send one after the other on the same TCP connection
    - Can lead to head-of-line blocking (HOB)
      - Packet loss in one request/response transaction delays subsequent (loss-free) request/response transactions (they are blocked)
      - Can significantly increase response time





Sequence of a Traditional HTTPS Request



When specifying half RTTs we assume symmetrical latencies

We use the term HTTPS for HTTP over TLS over TCP

Sequence of a Traditional HTTPS Request

- HTTP request
  - Triggers TLS handshake (3-way)
    - Triggers TCP handshake (3-way)
    - Client can start TLS handshake after first RTT
  - Client can start sending HTTP request after second RTT

- → Separation of concerns inefficient
  - Separation of security and transport connection establishment
    - TLS and TCP
  - TLS adds at least 1 RTT latency during handshake

- Simplifying assumptions
  - Certificates/HTTP-Requests fit in single TCP segment
    - Otherwise, multiple segments in one RTT required
    - Does not change the $2.5 \cdot RTT$ delay
  - TLS-Version 1.3
    - Otherwise longer 5-way handshake

HTTP/2

- Multiplexing of request/response messages on same TCP connection
  → Long-lived TCP connections, reduced response time

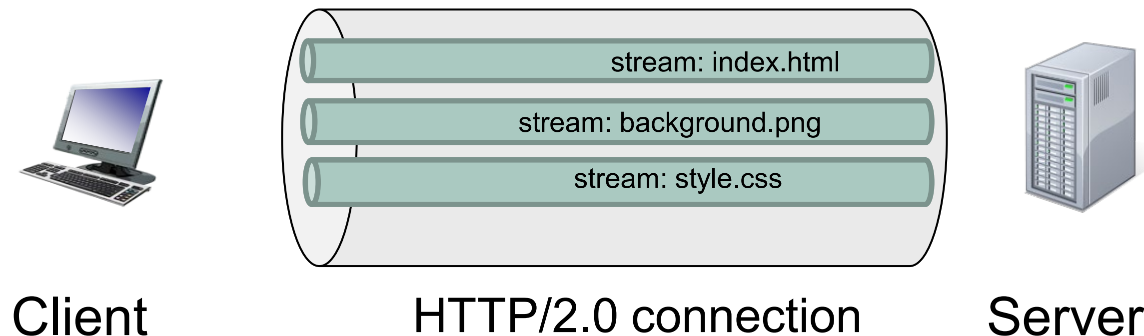
We consider only few aspects of HTTP/2

- New concepts, e.g.,
  - Stream
    - Independent bidirectional sequence of frames
    - Each request/response transaction associated with own stream
    - Streams are largely independent of each other
  - Prioritization
    - Prioritization of requests supported
    - Ensures that limited resources can be directed to most important stream first
  - Flow control
    - Ensures that only data that can be used by a receiver is transmitted

[RFC7540]

Stream

- Independent, bidirectional sequence of frames
  - Exchanged within an HTTP/2.0 connection
    - Single HTTP/2.0 connection can contain multiple streams
  - Each stream is associated with a stream identifier
    - 32 bit integer, increases monotonically, can not be reused
    - Client: odd numbers, server: even numbers
  - Order of frames matters
    - Frames are processed in the order in which they are received



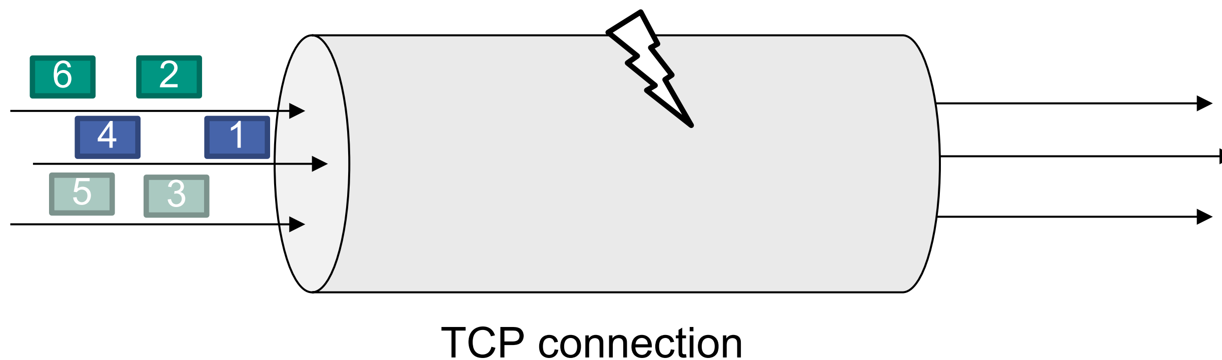
- Streams are multiplexed over a common TCP connection

Some Difficulties

- Slow start after idle ruins idea of long-lived TCP connections
  - Linux resumes with slow start, after connection has been „idle“
    - „Idle“ already after expiry of one retransmission timer

  → Recommended to disable this feature on servers

- Lost segments lead to head-of-line blocking
  - Due to strict in-order delivery: received out-of-order data are not passed to application
  - Loss, that only affects data of a single stream can block all other streams



→ Not solvable with TCP … *new transport protocol required!*

Karlsruher Institut für Technologie

9.3.4 HTTP/3

HTTP/3

Karlsruher Institut für Technologie

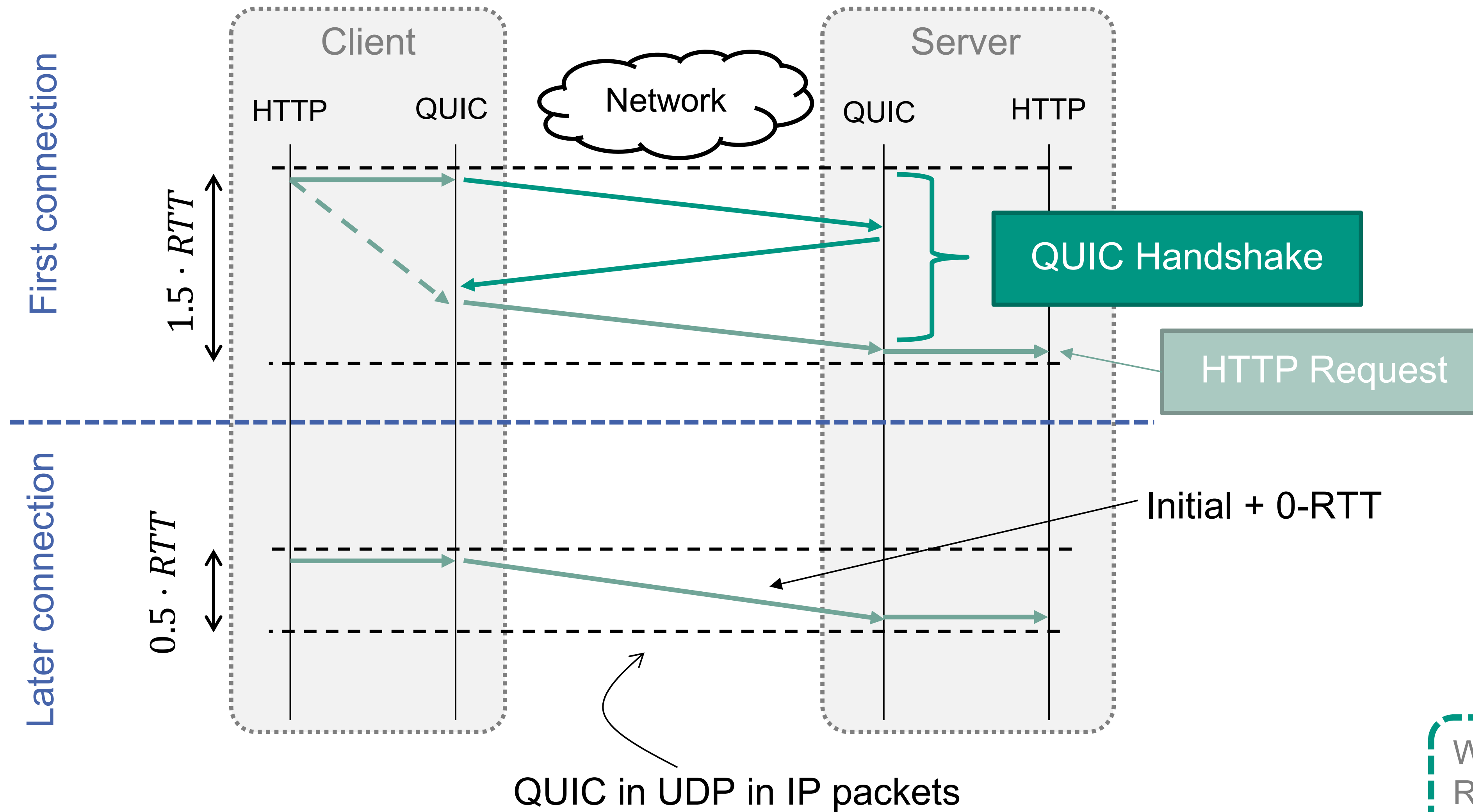
- Goal
  - Improved performance of HTTP

- Motivation: Use advantages of new transport protocol QUIC
  - Low-latency connection establishment
  - Stream multiplexing without Head-of-Line blocking

- HTTP/3
  - Mapping of HTTP/2 semantics over QUIC transport protocol

[RFC9114]

Minimal Sequence of HTTP/3 Request



When specifying half RTTs we assume symmetrical latencies

Sequence of HTTP/3 Request

- Sequence at first connection
  - HTTP request
    - Triggers QUIC handshake
    - Client can send HTTP request after one RTT
- Sequence at later connections
  - Client can send HTTP request immediately
    - Does not need to wait for handshake completion

- Simplifying assumptions
  - Certificates/HTTP-Requests fit in single IP packet
    - Otherwise, multiple UDP packets in one RTT required
    - Does not change the $1.5 \cdot RTT$ delay

Karlsruher Institut für Technologie

9.4 QUIC

QUIC Overview

Karlsruher Institut für Technologie

- New secure general-purpose connection-oriented transport protocol
  - Located on top of UDP (implemented in user space)
  - Originally proposed by Google as a transport protocol optimized for the WWW
  - Specified by IETF in RFCs 8999-9002 (published May 2021)

Google QUIC ≠ IETF QUIC

- Original Goal: Enable even lower HTTP response times

RFCs specify QUIC version 1

- Main new QUIC features

Stream multiplexing without head-of-line blocking	Always authenticated and encrypted (at least TLS1.3)
Optimized low-latency connection establishment	Connection Migration

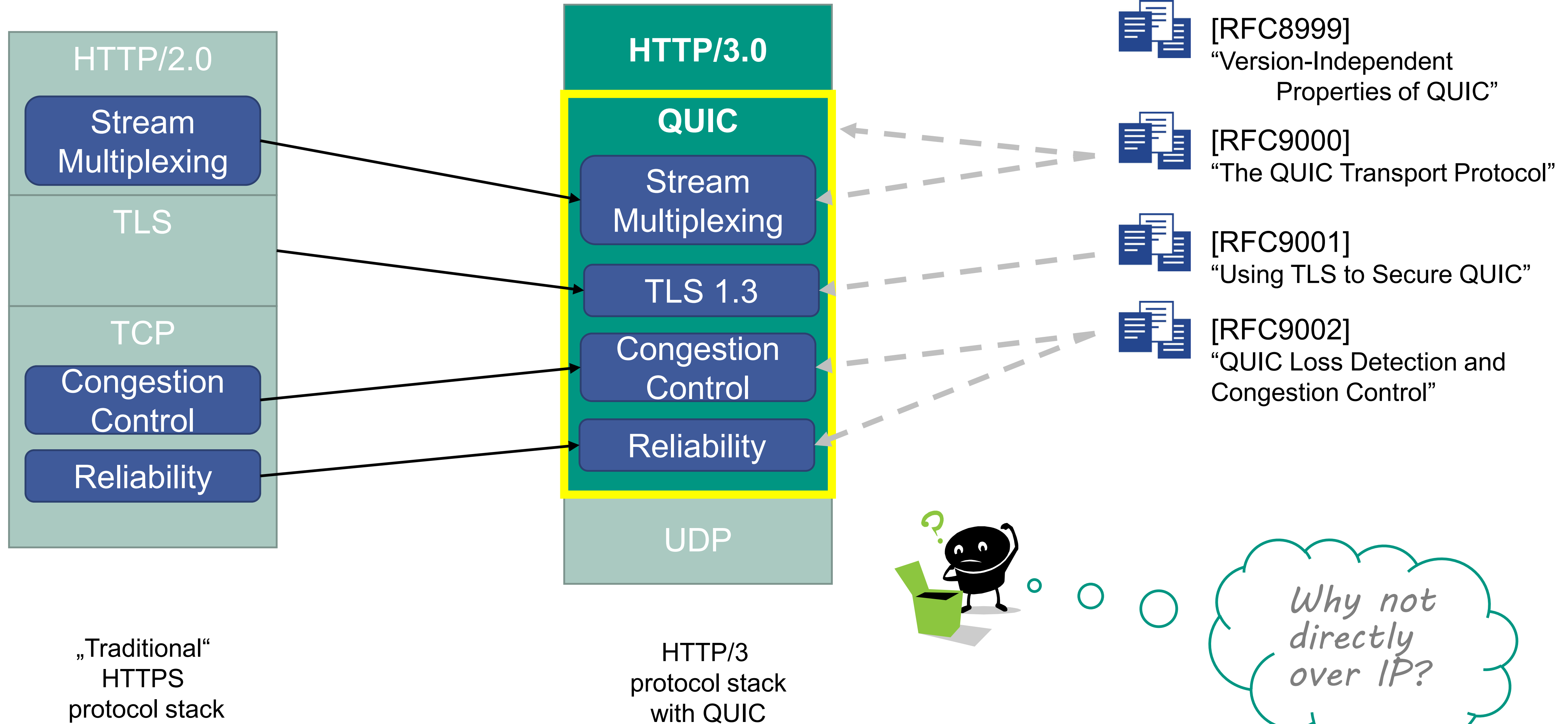
In the following, we assume TLS 1.3 is used

Further features

Karlsruher Institut für Technologie

- Integration of features previously seen in TCP extensions
  - Selective Acknowledgements from TCP SACK
  - 0-RTT data from TCP Fast Open
  - Token-based address validation similar to SYN Cookies

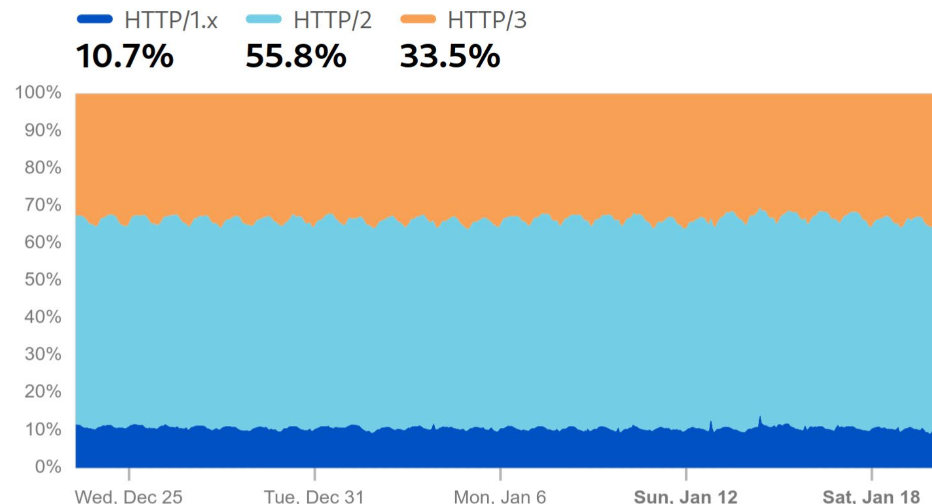
QUIC Protocol Stack



Deployment Status of QUIC

- New but already heavily in use in some deployments
  - Browser vendors + Hyperscalers are on board
    - Google, Facebook, Cloudflare …

- Some “current” statistics
  - Meta (Facebook): 75% of their traffic uses QUIC (2020)
  - Browser support: ~96% of Users
  - Cloudflare (HTTP-Traffic): 33.5 % (+2.1 % over 12 months)



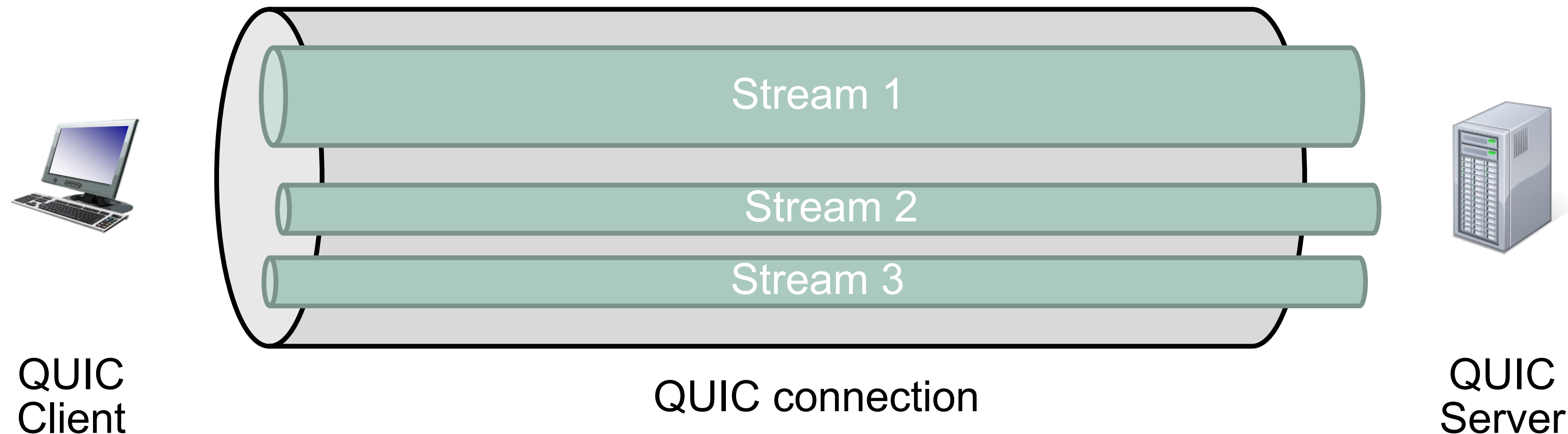
[https://radar.cloudflare.com/adoption-and-usage?range=28d]

Karlsruher Institut für Technologie

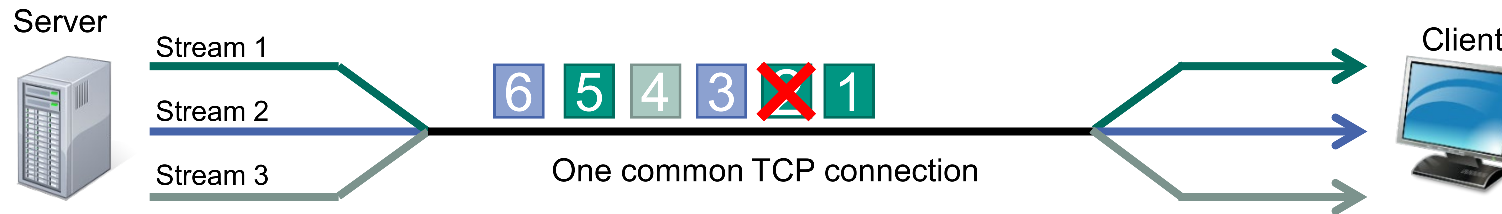
9.4.1 Connections and Streams

Connections and Streams

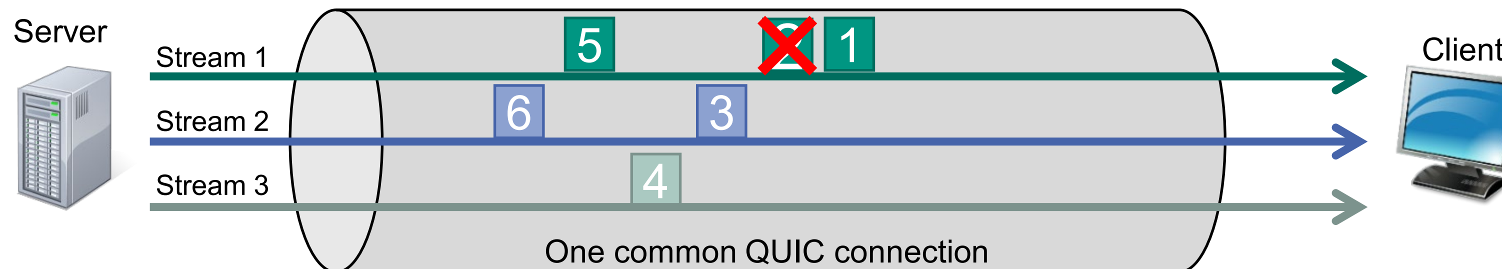
- QUIC is a connection-oriented transport protocol
  - Requires handshake to establish QUIC connection
- Can multiplex multiple application-level streams on single QUIC connection
  - Streams provide a byte-stream abstraction (like TCP)
  - Streams are identified by a StreamID unique within the connection
  - Reliable, ordered delivery of application data within one (!) stream
  - Just start sending data – No separate handshake required



No Head-of-Line-Blocking Across Streams



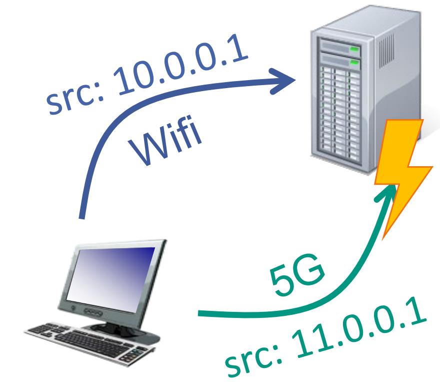
- HTTP/2: Multiplexing within TCP connection with ordered delivery
  - Ordered delivery “before” demultiplexing → Head-of-line blocking possible
    - Example: loss of packet 2. Later packets from other streams are blocked until it is successfully retransmitted



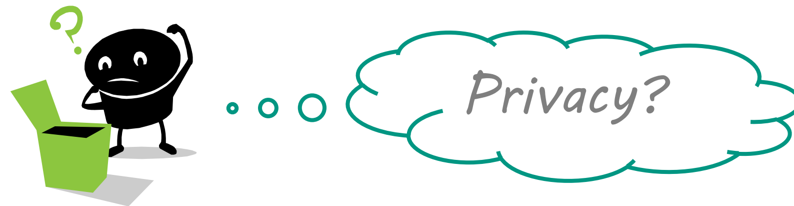
- HTTP/3: Multiplexing within QUIC connection (no ordered delivery)
  - Ordered delivery “after” demultiplexing → No head-of-line blocking for other streams
  - Example: loss of packet 2 blocks only stream 1, i.e., packet 5

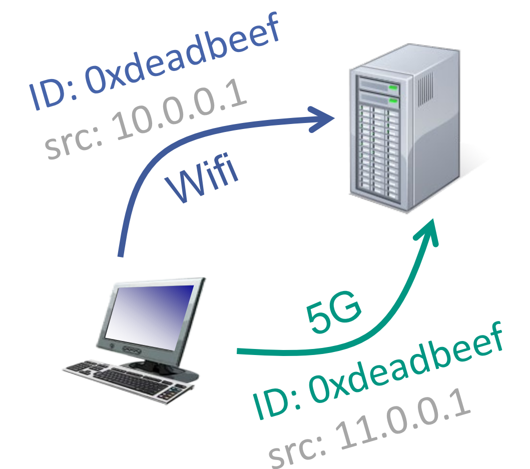
Connection Migration

- Problem: Connection Migration
  - TCP connections are identified by a 5-tuple
    (Source and Destination IP-Addresses and Ports, Protocol)
  - Any change, i.e. with a switch from Wifi to 5G, breaks the connection



- QUIC Connections are identified by connection IDs
  - IDs are independent of lower layer addresses (e.g., UDP and IP)
    - Packets are delivered to correct endpoint in case addresses change
  - Separate connection IDs for source and destination





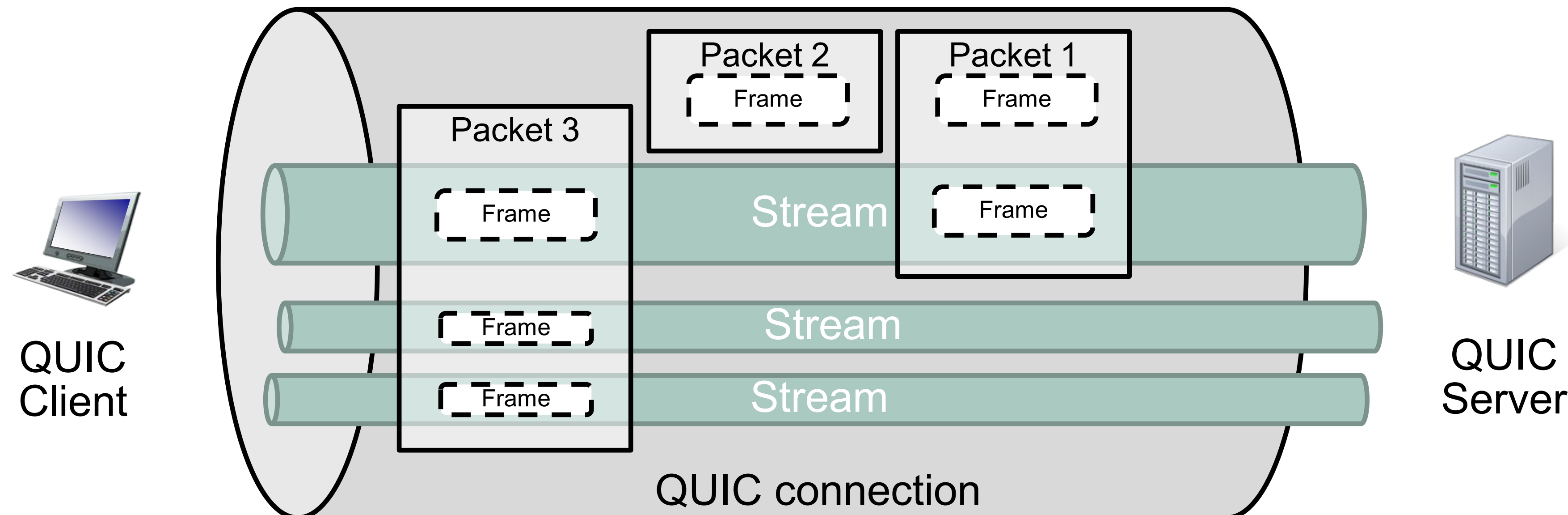
- Connection IDs allow tracking across network boundaries
  - Solution: Each connection has a set of connection IDs
  - Can be switched when changing networks

Karlsruher Institut für Technologie

9.4.2 Packets and Frames

Packets and Frames

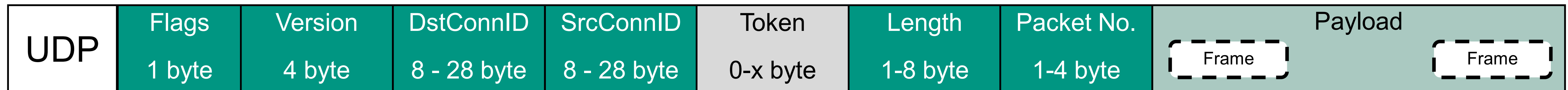
- Endpoints exchange QUIC packets
  - Packets contain frames
    - Frames carry control information and/or application data
    - Frames can belong to different streams
  - Packets are numbered (not Bytes as in TCP)



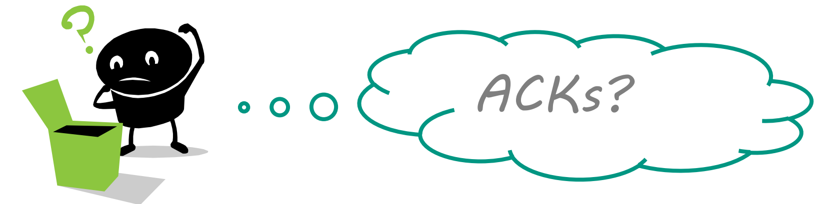
Packets

- QUIC packets are encapsulated into UDP packets
  - A packet must completely fit into one UDP packet

- Long Form Header (during handshake)

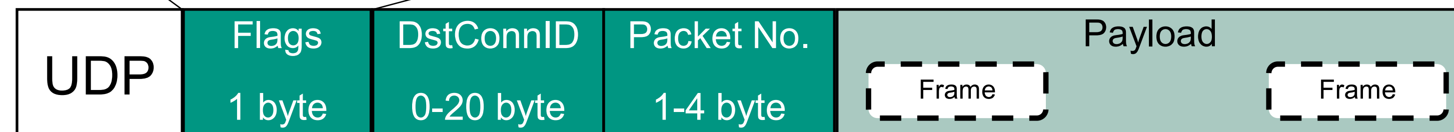


Header Form (1) = 1,
Fixed Bit (1) = 1,
Long Packet Type (2),
Reserved Bits (2),
Packet Number Length (2),



Header Form (1) = 0,
Fixed Bit (1) = 1,
Spin Bit (1),
Reserved Bits (2),
Packet Number Length (2),

- Short Form Header (after handshake)



Packets

Karlsruher Institut für Technologie

- Flags
  - Determines header format and packet type
  - Determines length of packet number field
- Version
  - QUICv1: `0x00000001`
  - QUICv2: `0x6b3343cf`
    - Test version negotiation and prevent middlebox ossification
- DstConnID
  - Determines to which endpoint the packet is delivered
  - Length is negotiated during handshake
  - Typically 20 byte
- SrcConnID
  - only required during handshake
  - so the other endpoint knows which DstConnID to use
- Token
  - Can be used similar to a SYN-Cookie in TCP
- Length
  - Length of payload field
  - For short headers the UDP length field is used
  - Uses variable-length integer encoding
- Packet No.
  - Increased with each packet sent

Types of Frames

KIT
Karlsruher Institut für Technologie

- Frames regarding connection-level state

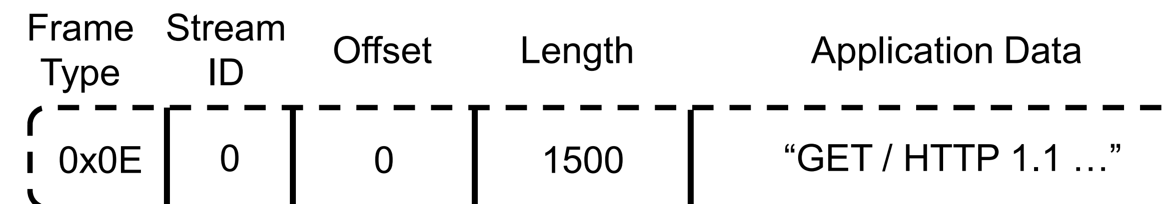
Functionality	Frames
Handshake	CRYPTO, HANDSHAKE_DONE
Flow control	MAX_DATA, DATA_BLOCKED
Acknowledgement	ACK
Connection teardown	CONNECTION_CLOSE
…	…

- Frames regarding stream-level state
  - Always contain StreamID to identify the stream

Functionality	Frames
Flow control	MAX_STREAM_DATA, STREAM_DATA_BLOCKED
Stream cancellation	RESET_STREAM

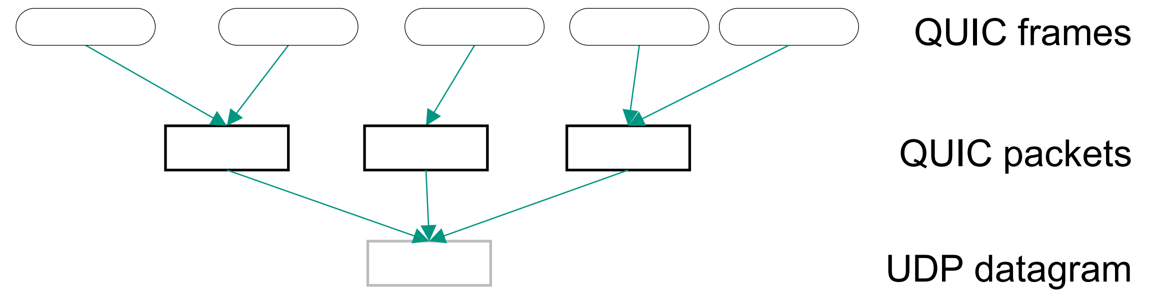
- Frames carrying application data
  - Called STREAM frames
    - Carries a stream segment of application data
    - StreamID and byte offset in frame header

- Example of a STREAM frame

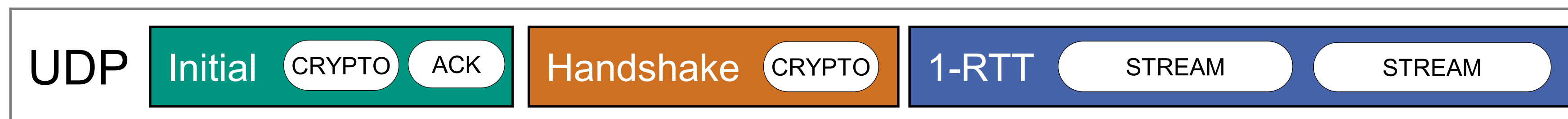


Packetization & Coalescing

- Packetization: a QUIC packet can contain multiple frames
  - E.g., from different streams or connection-level frames
  - Reduces per-packet overhead (e.g., for serialization)

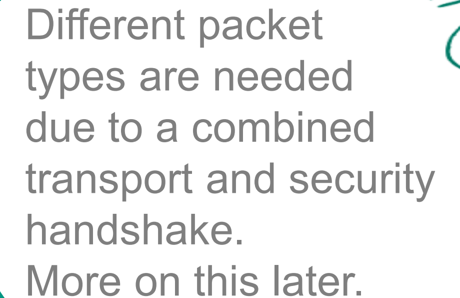


- Coalescing: a UDP datagram can contain multiple QUIC packets
  - Allows different packet types to be sent in a single UDP packet
  - Example



Packets Types

- Only during connection establishment handshake
  - Initial for first handshake packets
  - Handshake for remaining handshake packets
  - 0-RTT for client application data before handshake is done
- After handshake
  - 1-RTT for all post-handshake data (control and application)
    - Uses short header format
- Packets are numbered ascendingly in sending order
  - Three separate number spaces
    - Initial, Handshake, Data (0-RTT and 1-RTT)

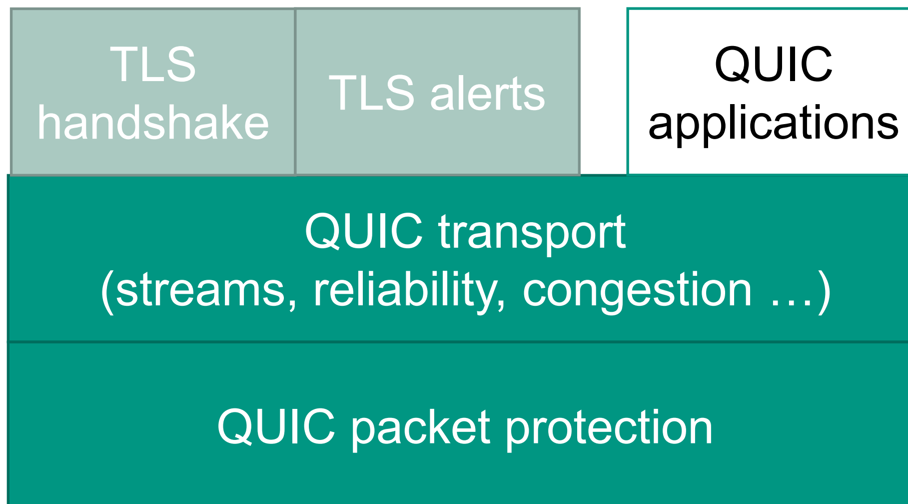


9.4.3 Always Authenticated and Encrypted

QUIC Security

- Protection goals
  - Confidentiality and integrity of packets
  - Authentication
    - Server authentication required, client authentication optional

- TLS1.3 is used as security component

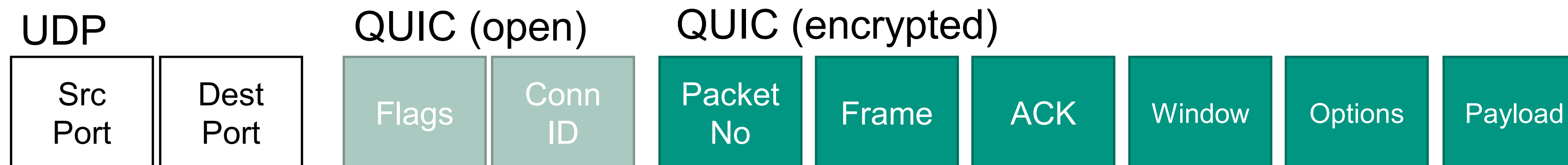


- TLS is used (only) for key negotiation
  - E.g., encryption keys

[RFC9001]

Confidentiality

- QUIC uses end-to-end encryption
  - Not encrypted: public flags and connection ID
    - After connection establishment essentially only the destination connection ID
  - Remainder of packet encrypted
    - → fields can not be inspected and changed by middleboxes



Karlsruher Institut für Technologie

9.4.4 Connection Establishment

TeleMatics

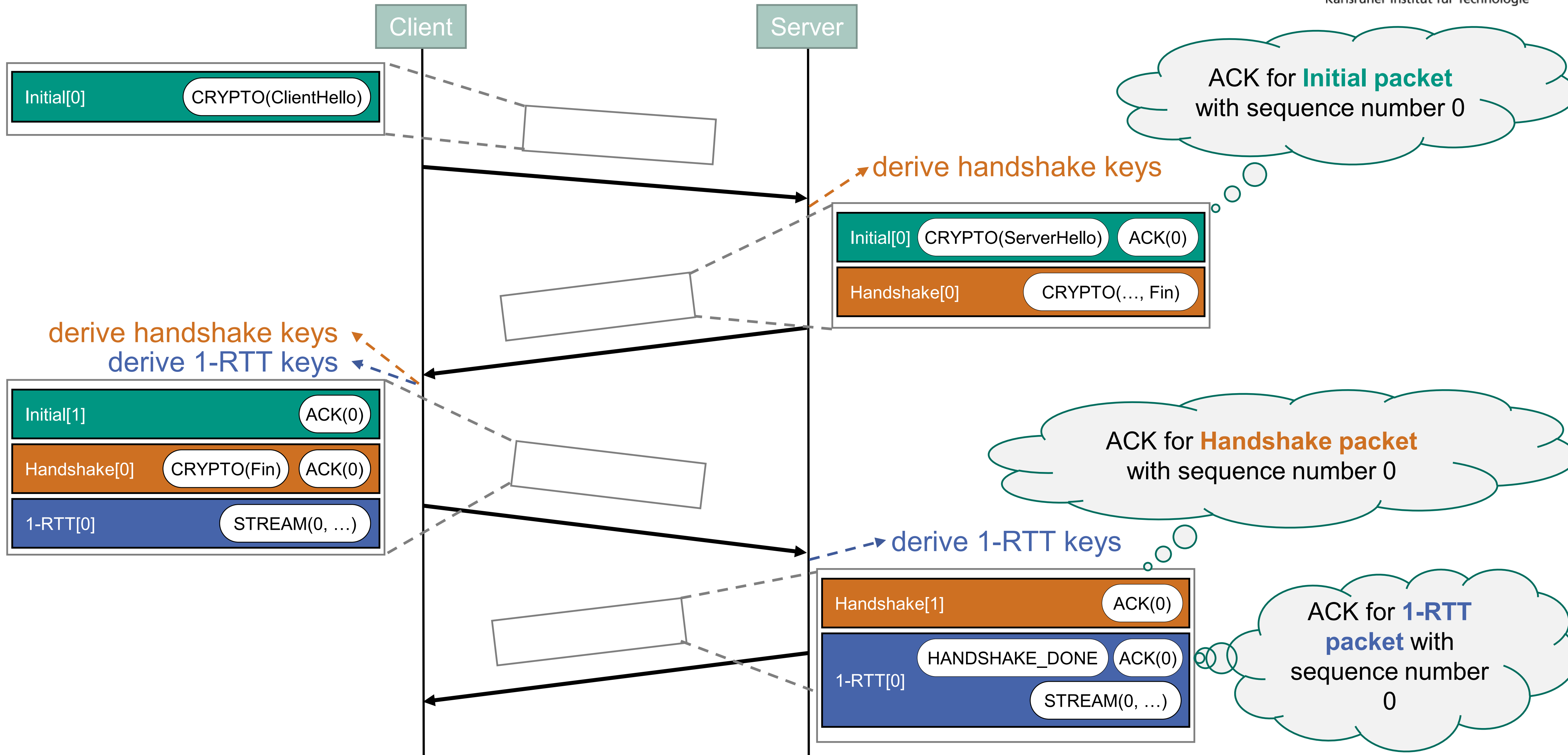
Connection Establishment

Karlsruher Institut für Technologie

- Combined transport and cryptographic handshake
  - No strict layering - TLS and QUIC are co-dependent
    - QUIC uses TLS handshake
    - QUIC uses keys provided by TLS to encrypt its packets
    - TLS uses reliability, ordered delivery and record layer (i.e., CRYPTO frames transmitted in QUIC packets) provided by QUIC
- First connection to server
  - One RTT (1-RTT) handshake
- Subsequent connections to same server
  - … typical usage pattern for web browsing (!)
  - Zero RTT (0-RTT) handshake
    - Reuses parameters and keying material from prior 1-RTT handshake
      - Cryptographic cookie stores exchanged parameters
        - E.g., Diffie-Hellman value for calculating encryption key

[RFC9000]

Connection Establishment Handshake

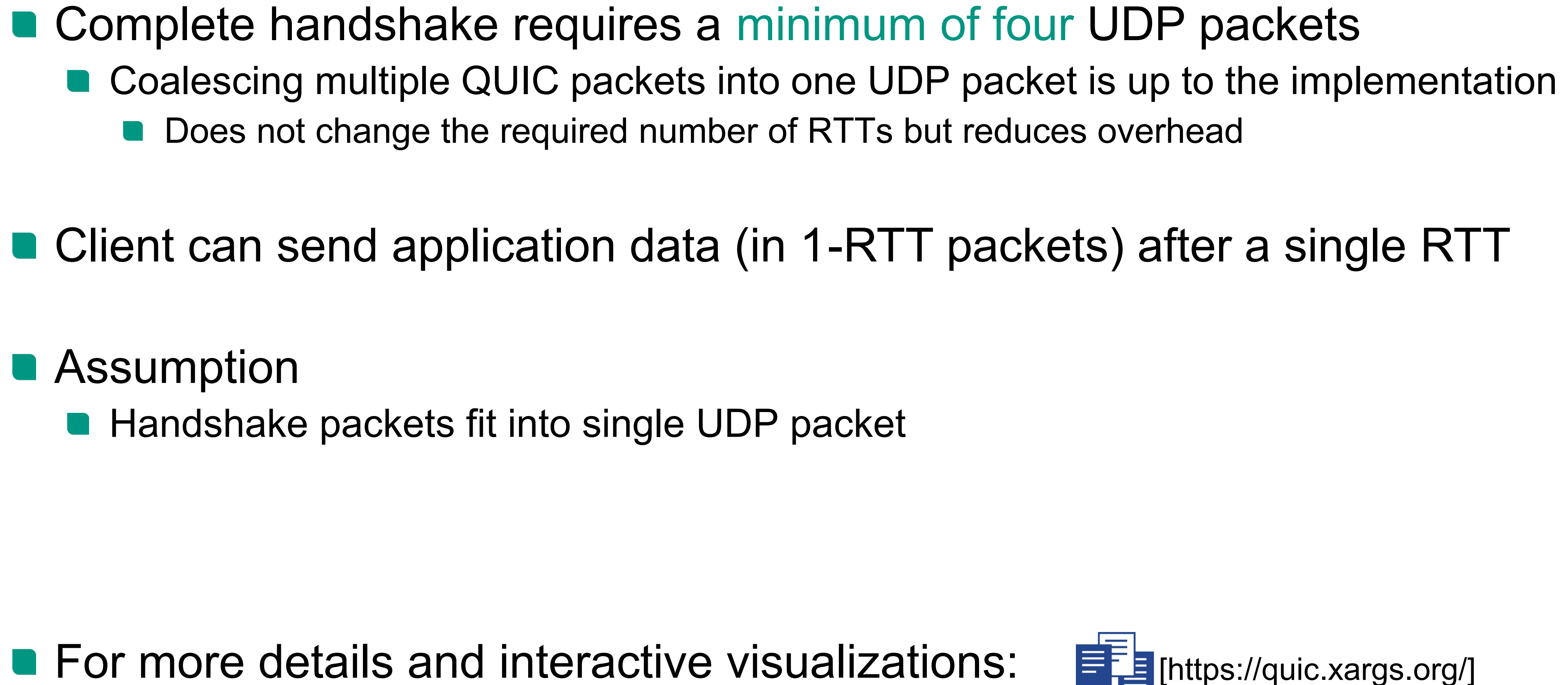


Encryption during Connection Establishment

- Different packet types are encrypted with different encryption keys
  - As they become available during the combined handshake

  - Initial → Keys derived from initial destination connection id
    - Initial destination connection ID is chosen randomly and sent in plaintext
    - Keys can be derived by anyone that sees the packet
    - Protects against off-path attackers trying to influence the handshake
    - Protects against accidental modification
  - Handshake → Keys derived from ClientHello and ServerHello
    - Allows for encrypted negotiation of transport parameters
  - 1-RTT → Keys derived from complete Handshake

Connection Establishment Handshake

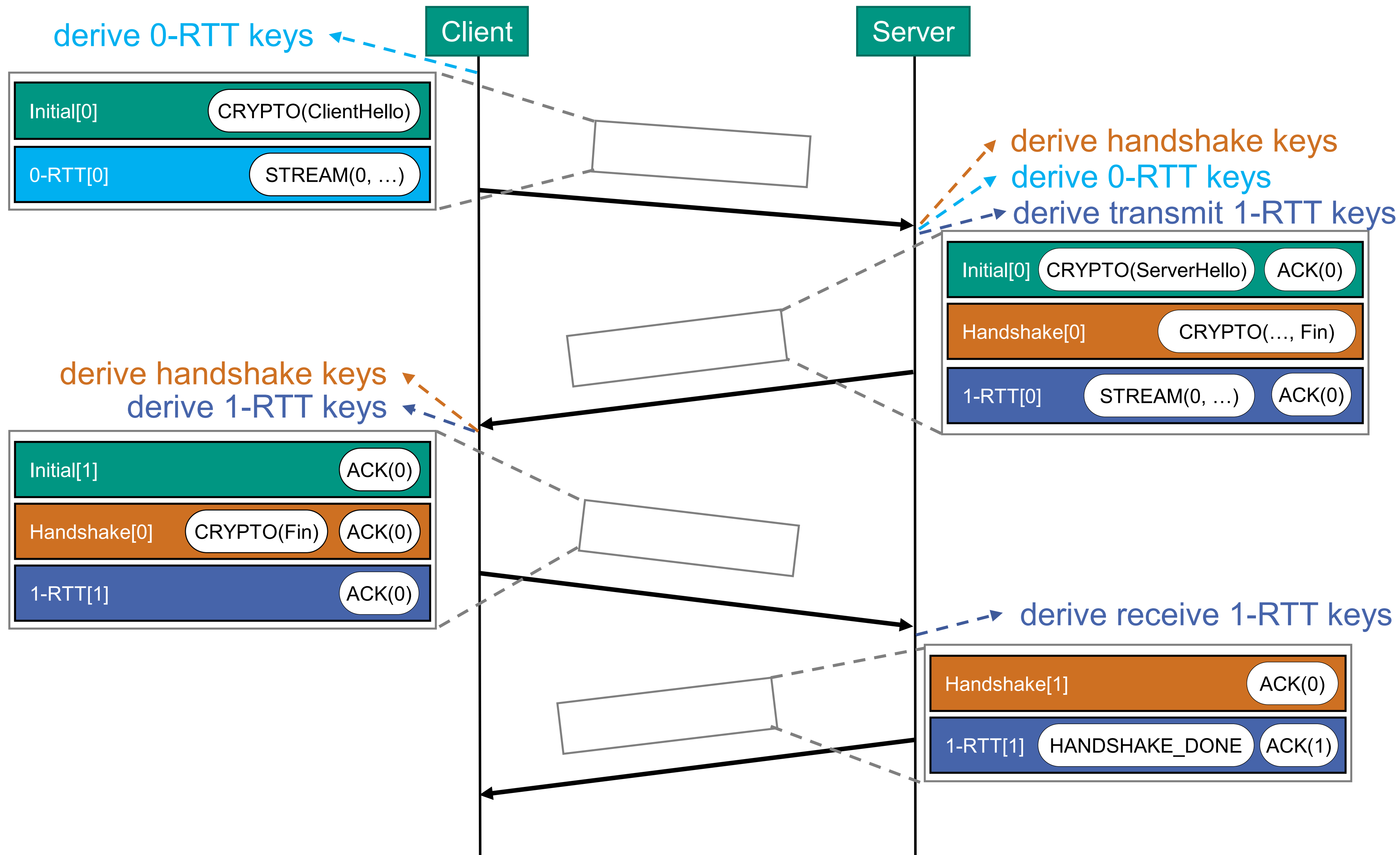


- Complete handshake requires a minimum of four UDP packets
  - Coalescing multiple QUIC packets into one UDP packet is up to the implementation
    - Does not change the required number of RTTs but reduces overhead
- Client can send application data (in 1-RTT packets) after a single RTT
- Assumption
  - Handshake packets fit into single UDP packet
- For more details and interactive visualizations: [https://quic.xargs.org/]

Karlsruher Institut für Technologie

9.4.5 Low Latency Data

Sending 0-RTT Data



Sending 0-RTT Data

- Handshake itself stays the same

- Client requires 0-RTT encryption keys from a previous connection
- Client can send 0-RTT data immediately
  - with/after sending the Initial packet

- Server can start sending data after 0.5 RTT
  - In 1-RTT packets

- After handshake completion the client must not send 0-RTT packets anymore

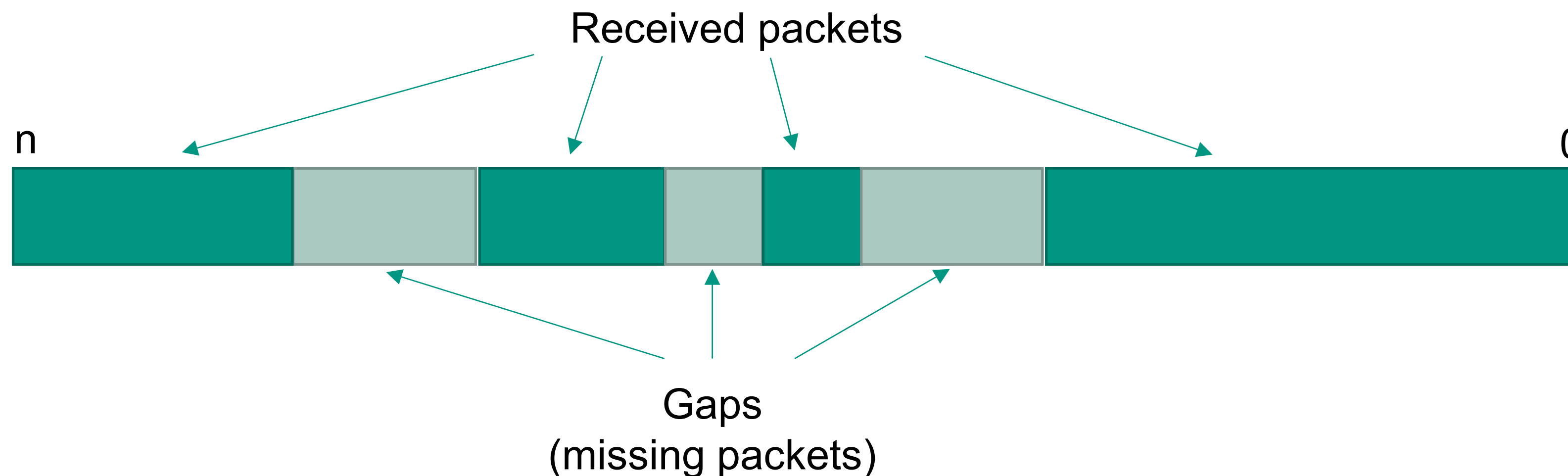
Karlsruher Institut für Technologie

9.4.6 Further Protocol Functionality

Acknowledgements

Karlsruher Institut für Technologie

- QUIC uses only positive selective acknowledgements
  - QUIC acknowledges packets
    - This differs from TCP which acknowledges data (bytes)
  - All frames within a packet are implicitly also acknowledged
- Packet number space is divided into
  - Ack Ranges: Ranges of packets that were received and acknowledged
  - Gaps: Ranges of packets that are not (yet) received



Retransmissions

- Packets (as a whole) are not retransmitted
  - But individual frames in lost packets may need to be
- On packet loss
  - Sender determines which frames were in the lost packet
  - Sender checks which frames need to be retransmitted
    - E.g., application data in STREAM frames
  - Frames to be retransmitted are sent in new QUIC packets
    - New packet number
      - Acknowledgements can be clearly associated with packet. Recall related problem in TCP
    - Data may be reframed
    - New packets may be composed of different set of frames
- How can ordered delivery in a stream be supported?
  - By stream offsets in stream frames
  - Similar to sequence number in TCP

Congestion Control

Karlsruher Institut für Technologie

- No specific congestion control algorithm specified
  - … pluggable interface to allow for experimentation
  - allows for all kinds of congestion control algorithms
    - NewReno, Cubic, BBR, …
- Congestion control is applied to QUIC connection, not to single streams
- Congestion window
  - Based on bytes, not MSS
- Usage of pacing is recommended

That´s it!

- Thanks for attending
- Remember: there is a final exercise next Wednesday

- Got interested in computer networks?
  - We have more advanced lectures, seminars, lab courses
  - BA- and MA-thesis on actual topics

- Good luck in the exam
  - March, 23th

Karlsruher Institut für Technologie

LITERATURE

TeleMatics

Literature

[Card16] N. Cardwell et al; BBR: Congestion-based Congestion Control, ACM Queue, Bd. 14, Nr. 5, S. 50, 2016

[Card24] N. Cardwell et al.; BBRv3: Algorithm Overview and Google´s Public Internet Deployment; presentation IETF 119, Brisbaine, March 2024

[CaSB24] N. Cardwell et al.; BBR Congestion Control; draft-ietf-ccwg-bbr-01, October 2024

[Hock17] M. Hock, R. Bless, M. Zitterbart; Experimental Evaluation of BBR Congestion Control, IEEE 25th International Conference on Network Protocols (ICNP), 2017

[Hous06] G. Houston; Gigabit TCP; The Internet Protocol Journal, Vol. 9, No. 2, Juni 2006

[Maka19] M. Mathis, J. Mahdavi; Deprecating The TCP Macroscopic Model; ACM SIGCOMM Computer Communication Review; Vol. 49, Issue 5, October 2019

[McAk19] N. McKeown, G. Appenzeller, I. Keslassy; Sizing Router Buffers (Redux); ACM SIGCOMM Computer Communication Review, Vol. 49, Issue 5, Oct. 2019

[RFC1323] V. Jacobson, R. Braden, D. Borman; TCP Extensions for High Performance; RFC 1323, May 1992

[RFC2018] S. Floyd, J. Mahdavi, M. Mathis, A. Romanow; TCP Selective Acknowledgment Options; RFC 2018, 1996

[RFC4987] W. Eddy; TCP SYN Flooding Attacks and Common Mitigations; RFC 4987, August 2007

[RFC6675] E. Blanton et al.; A Conservative Loss Recovery Algorithm Based on Selective Acknoledgment (SACK) for TCP; RFC 6675, August 2012

Literature

Karlsruher Institut für Technologie

[RFC7413] Y. Cheng et al.; TCP Fast Open; RFC 7413, December 2014

[RFC7540] M. Belshe et al.; Hypertext Transfer Protocol Version 2 (HTTP/2); RFC 7540, May 2015

[RFC8312] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, R. Scheffenegger; CUBIC for Fast Long-Distance Networks; RFC 8312, 2018

[RFC8999] M. Thomson; Version-Independent Properties of QUIC; RFC 8999, May 2021

[RFC9000] J. Iyengar, M. Thomson; QUIC: A UDP-Based Multiplexed and Secure Transport; RFC 9000, May 2021

[RFC9001] M. Thomson, S. Turner; Using TLS to Secure QUIC; RFC 9001, May 2021

[RFC9002] J. Iyengar, I. Swett; QUIC Loss Detection and Congestion Control; RFC 9002, May 2021

[RFC9114] M. Bishop; HTTP/3; RFC 9114, Jun2 2022

[RhXu08] I. Rhee, L. Xu; CUBIC: A New TCP-Friendly High-Speed TCP Variant; ACM SIGOPS 2008

[Welz22] M. Welzl, P. Teymoori, S. Islam, D. Hutchison and S. Gjessing, "Future Internet Congestion Control: The Diminishing Feedback Problem" in IEEE Communications Magazine, vol. 60, no. 9, pp. 87-92, Sept. 2022