Computer Networking and IT Security (CNS)

INHN0012 – WiSe 2024/25

Prof. Dr.-Ing. Stephan Günther

Chair of Distributed Systems and Security
School of Computation, Information and Technology
Technical University of Munich

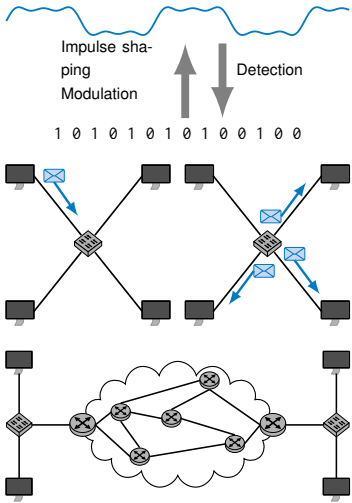
Chapter 4: Transport layer

Chapter 4: Transport layer

Motivation

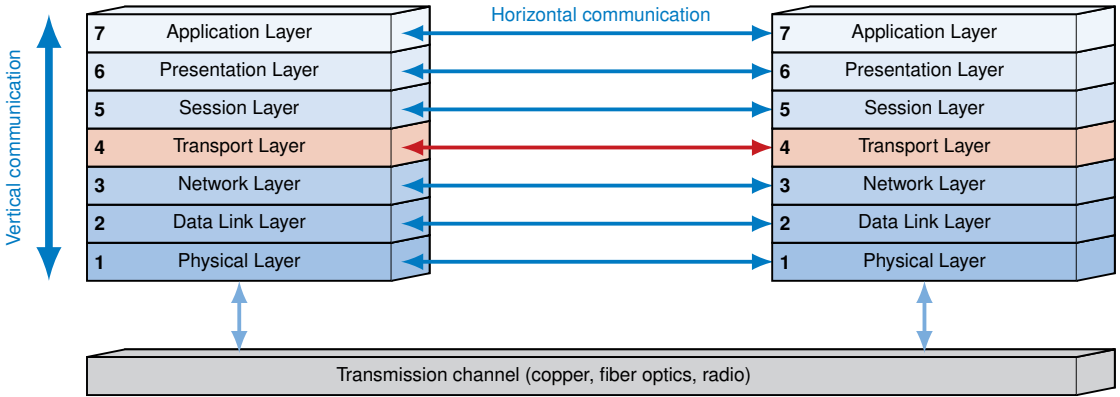
We have seen so far:

- How digital data is represented, transmitted, and reconstructed by measurable quantities (layer 1)
- How access to the transmission medium is controlled (MAC)
- How the respective next-hop is addressed (layer 2)
- How hosts are addressed end-to-end on the basis of logical addresses and how data is transmitted in a packet-oriented manner (layer 3)
- How layer 2 addresses are resolved given a layer 3 address



Motivation

Placement in the ISO/OSI model



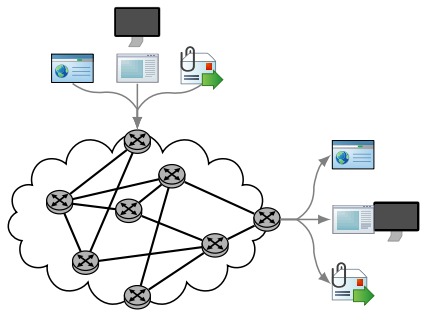
Tasks of the transport layer

The main tasks of the transport layer are

- multiplexing of data streams of different applications,

Multiplexing:

- Segmentation of data streams from different applications (browser, chat, email, . . .)
- Segmente werden in jeweils unabhängigen IP-Paketen zum Empfänger geroutet
- Segments are routed in independent IP packets to the destination
- Receiver must assign the segments to the individual data streams and pass them on to the respective application



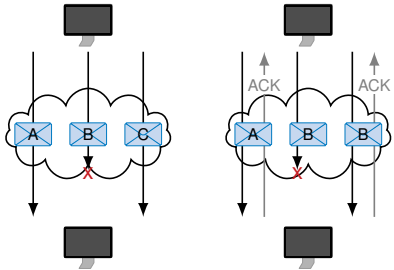
Tasks of the transport layer

The main tasks of the transport layer are

- multiplexing of data streams of different applications,
- providing connectionless and connection-oriented transport mechanisms, and

Transport services:

- Connectionless (best effort)
  - From the perspective of the transport layer, segments are independent from each other
  - No sequence numbers, retransmits, or guarantees of correct sequence
- Connection-oriented
  - Retransmits on errors
  - Guarantee of the correct sequence of individual segments



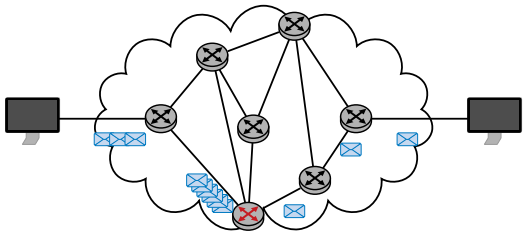
Tasks of the transport layer

The main tasks of the transport layer are

- multiplexing of data streams of different applications,
- providing connectionless and connection-oriented transport mechanisms, and
- mechanisms for congestion and flow control.

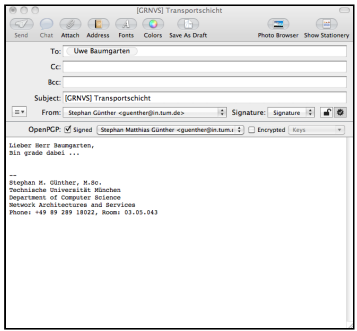
Congestion and flow control:

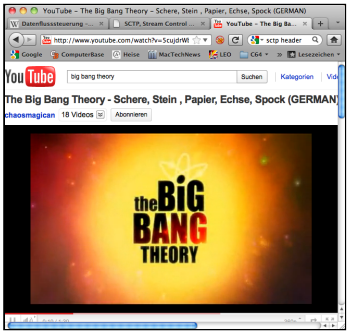
- Congestion control
  - Reaction to impending overload in the network
  - Dynamically adapt transmission speed
- Flow control
  - Load control by the receiver (avoid to overwhelm a slow receiver)



Chapter 4: Transport layer

Multiplexing





OSI Schicht 7 – 5

01011101 (UTF 8)

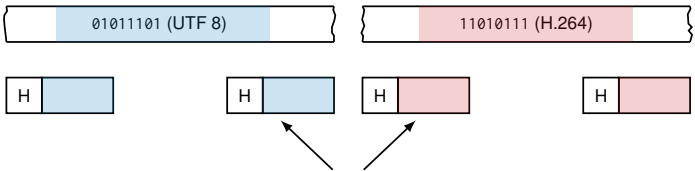
11010111 (H.264)

Transport layer

Multiplexing

At the transport layer,

1. the encoded data streams are divided into segments or datagrams, and
2. each segment is provided with a header.



Such a header contains at least a

- source port and
- destination port,

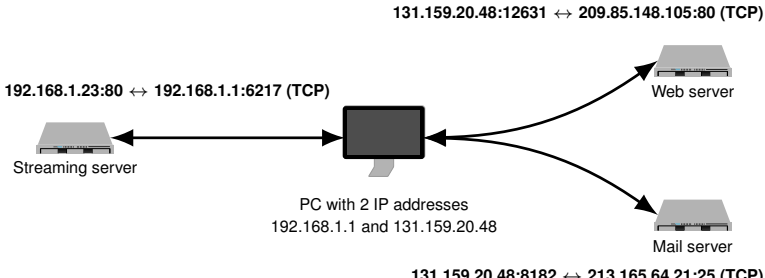
which, together with the source and destination IP address and the transport protocol used, uniquely identify the application on the source and destination.

⇒ 5-tuple consisting of:

```
(SrcIPAddr, SrcPort, DstIPAddr, DstPort, Protocol)
```

Multiplexing

Example:



- Port numbers are 16 bit long for the commonly used transport protocols.
- Operating systems use the 5-typle (IP addresses, port numbers, and protocol) to provide sockets as interface to applications.
- An application in turn addresses a socket using a file descriptor (integer value).
- Connection-oriented sockets can be used very easily after the connection is established, since the receiver is already fixed. (Reading and writing from / to such a socket is possible by using the system calls `read()` and `write()`.)
- Connectionless sockets require address information to whom to send, or provide information from whom data has been received (`sendto()` and `recvfrom()`).

Chapter 4: Transport layer

Connectionless transmission

The header of a transport layer protocols consists at least of

- the source and destination port as well as
- a length specification of the payload.

This allows an application to specify

- the recipient (IP address) and
- the receiving application (protocol and destination port).

Problems: Since the segments are sent independently of each other and from the perspective of the transport layer stateless, it cannot be ensured that

- segments arrive at the destination (packets may be lost) or
- arrive at the recipient in the correct order (packets are routed independently of each other).

Consequently, one speaks of unsecured, connectionless, and message-oriented communication.

Notes:

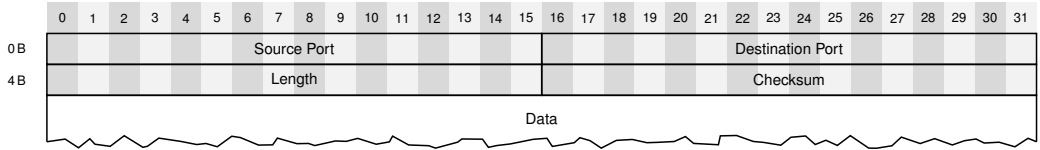
- Connectionless POSIX sockets are identified by the macro `SOCK_DGRAM`.
- `DGRAM` stands for datagram, which is simply a message of a certain length that is to be transmitted as a unit from the point of view of the transport layer.

User Datagram Protocol (UDP)

The User Datagram Protcol (UDP) is one of the most commonly used transport layer protocols in the internet. It provides

- unsecured and message-oriented transmission with
- low overhead (small header, no additional functionality).

UDP header:



- Length specifies the length of header plus payload in multiples of Bytes.
- The checksum is calculated over the UDP payload and parts of the IP and UDP headers:
  - Using the UDP checksum is optional with IPv4 but mandatory with IPv6.
  - If unused, the field is set to zero.
  - If used, a pseudo header is used for the calculation (some kind of “default IP header” that is only used to calculate the checksum). It contains the source and destination IP address, a 8 bit field set to zero, protocol ID, and length of the UDP datagram.

User Datagram Protocol (UDP)

Advantages of UDP:

- Low overhead
- No delay due to connection establishment, retransmits, and reordering of segments
- The data rate is not adapted by flow and congestion control
- May be used for realtime applications such as voice over IP where some packet loss is acceptable
- May be used to implement custom “transport protocols” such as QUIC which, strictly speaking, is not a transport layer protocol

Disadvantages of UDP:

- No guarantees, best effort delivery only
- Datagrams may arrive out-of-order
- No flow control (a fast sender may overwhelm a slower receiver)
- No congestion control (overload in the network results in high loss rates)

Where is UDP used?

UDP may be used wherever

- occasional loss of datagrams can be tolerated or compensated for by higher layers, or where
- a time-consuming connection setup, as required by other transport protocols, cannot be tolerated.

Examples:

- Name resolution using the Domain Name System (DNS)
  - DNS is used to resolve "web addresses" such as `www.tum.de` to IP addresses.
  - Before the name resolution is not completed, no connection to the target can be established either ($\rightarrow$ time delay).
- Data traffic with real-time requirements
  - Flow and congestion control mechanisms can introduce non-deterministic latencies.
  - Data arriving too late is often no longer relevant here.
- Google's QUIC
  - Protocol for accelerating TLS 1.3 encrypted connections.
  - Missing mechanisms of UDP are implemented on the application layer.
  - As a result, application must implement QUIC directly (or use user-space libraries). There is no direct implementation of QUIC in the kernel / operating system.
  - In this respect, it is not correct to speak of a transport protocol in the sense of the ISO / OSI model in the case of QUIC.

Chapter 4: Transport layer

Connection-oriented transmission

Basic idea: use sequence numbers in the protocol header. This allows for

- acknowledgements of successfully received segments,
- identification missing segments,
- re-transmit of missing segments, and
- reassembly of segments in the correct order.

Problems: Sender and recipient must

- be synchronized (exchange of initial sequence numbers) and
- keep state (current sequence number, segments acknowledged, . . .).

Connection-oriented transmission

Basic idea: use sequence numbers in the protocol header. This allows for

- acknowledgements of successfully received segments,
- identification missing segments,
- re-transmit of missing segments, and
- reassembly of segments in the correct order.

Problems: Sender and recipient must

- be synchronized (exchange of initial sequence numbers) and
- keep state (current sequence number, segments acknowledged, . . .).

Connection phases:

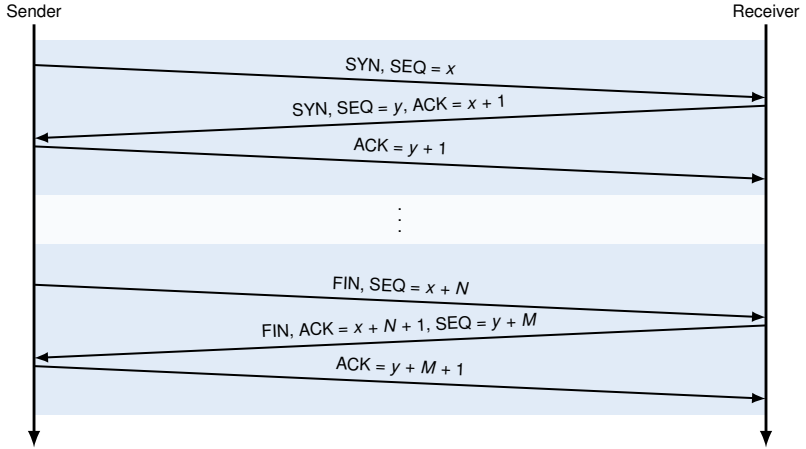
1. Connection establishment (handshake)
2. Data transmission
3. Connection teardown

Assumptions: For now, we assume that

- whole segments are acknowledged and
- the sequence number contained in an acknowledgement specifies the next segment that is expected by the recipient.

Connection-oriented transmission

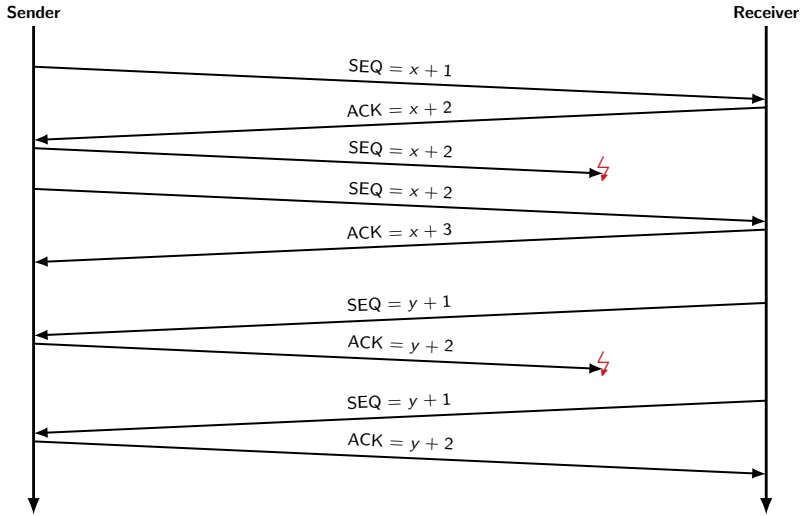
Example: Establishment and teardown of a connection



This type of connection establishment is called 3-way handshake.

Connection-oriented transmission

Example: Transmission phase



Sliding window approaches

So far:

- In the previous example, the transmitter always sent only one segment and then waited for an acknowledgement.
- This method is inefficient, because depending on the round trip time (RTT) between sender and receiver a lot of bandwidth remains unused (“stop and wait method).

Idea: Tell the sender how many segments **after** the last confirmed segment may be transmitted at once without the sender having to wait for an acknowledgement.

Advantages:

- Time between sending a segment and receiving an acknowledgement can be used more efficiently
- By negotiating this window size, the receiver can control the data rate → flow control
- By algorithmically adjusting the window size, the data rate can be matched to the available data rate on the transmission path between the sender and receiver → congestion control

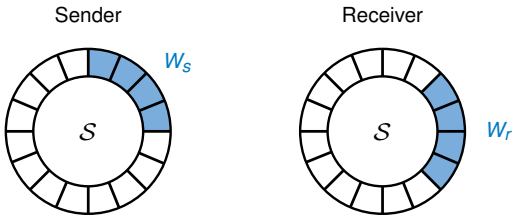
Problems:

- Sender and receiver must keep track of more state (what has already been received? What is expected next?)
- The sequence number space is finite → what happens at a wrap around?
- The RTT has to be estimated

Notation:

- Sender and receiver use the same sequence number space $\mathcal{S} = \{0,1,2,\ldots,N-1\}$.

  Example: $N = 16$:



- Send window $W_s \subset \mathcal{S}$, $|W_s| = w_s$:
  w_s segments may be sent at once after the last confirmed segment.
- Receive window $W_r \subset \mathcal{S}$, $|W_r| = w_r$:
  Sequence numbers of the segments that will be accepted next.
- Send and receive windows “shift” and overlap during data exchange.

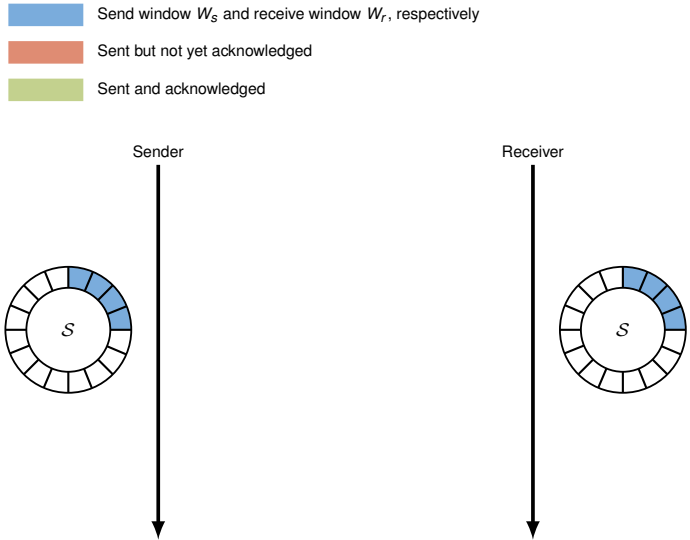
Conventions:

- An acknowledgement ACK = $m + 1$ confirms the correction reception of SEQ $\leq m$. This is called cumulative acknowledgements.
- Usually each successfully received segment triggers the sending of an acknowledgement, always acknowledging the next expected segment. This is called forward acknowledgement.

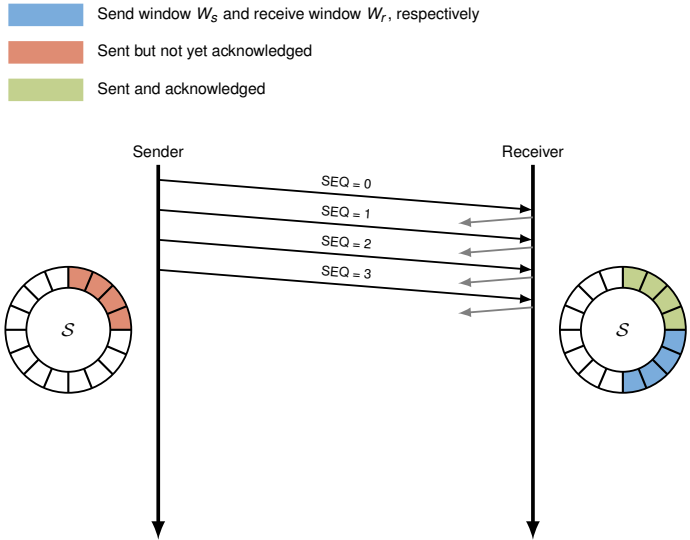
Important:

- In the following graphs, most confirmations are only indicated (gray arrows) for clarity.
- The effects on send and receive windows refer only to the receipt of the acknowledgements drawn in black.
- This is equivalent to assuming that the implied confirmations are lost.

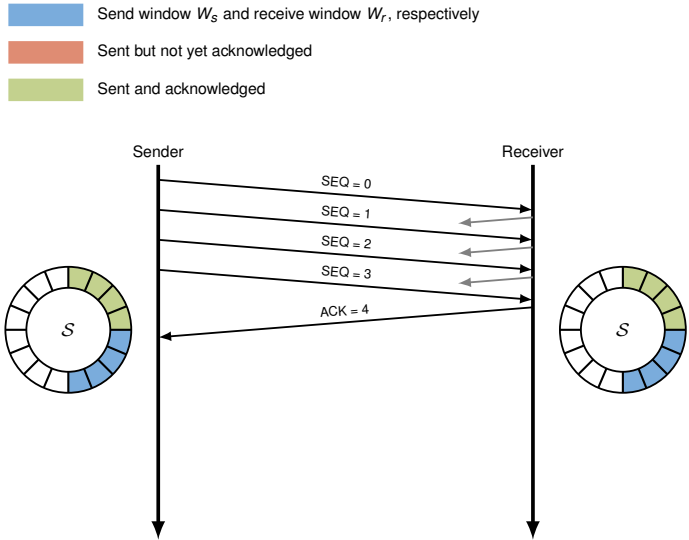
Sliding window approaches



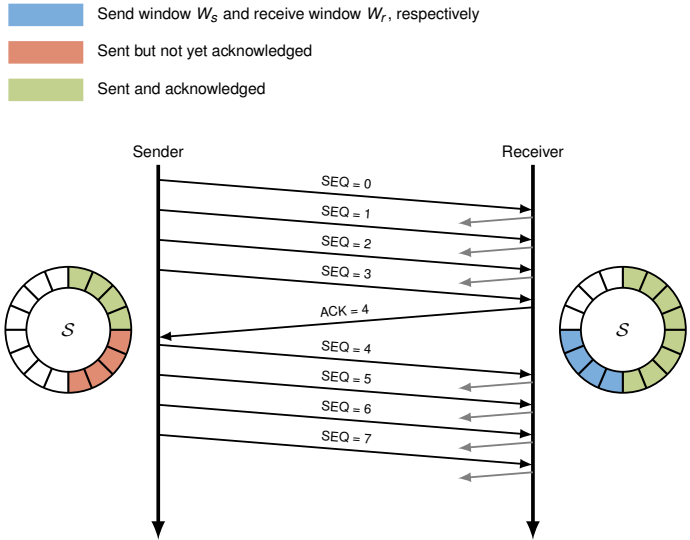
Sliding window approaches



Sliding window approaches



Sliding window approaches

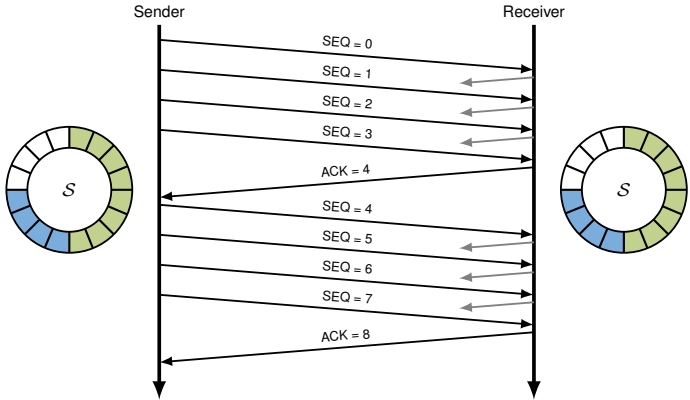


Sliding window approaches

Send window W_s and receive window W_r, respectively

Sent but not yet acknowledged

Sent and acknowledged



New problem: How to deal with lost segments?

There are two possibilities:

1. Go-back-N
   - Always accept only the next expected sequence number
   - All other segments are discarded
2. Selective repeat
   - Accept all sequence numbers that fall within the current receive window
   - These must be buffered until missing segments have been retransmitted

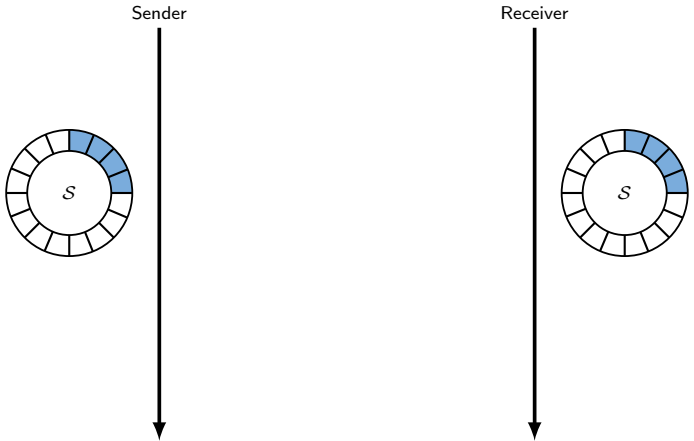
Important:

- In both cases, the sequence number space must be chosen so that repeated segments can be clearly distinguished from new segments.
- Otherwise there would be confusion $\rightarrow$ delivery of duplicates to higher layers, no correct order.

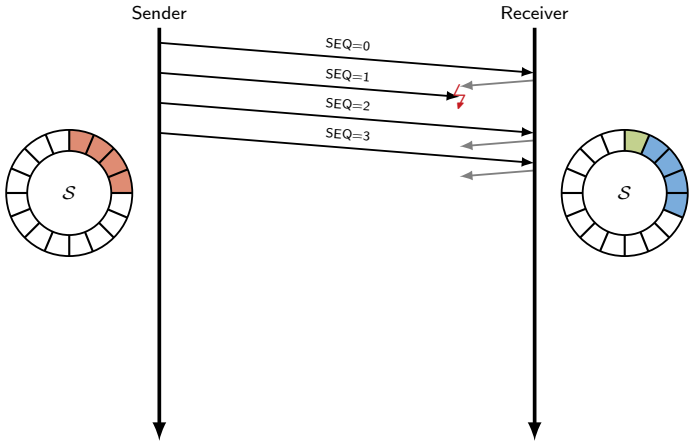
Question: What is the maximum size of the send window W_s that can be chosen as a function of the sequence number space S so that the methods work? (see tutorial)

Sliding window approaches

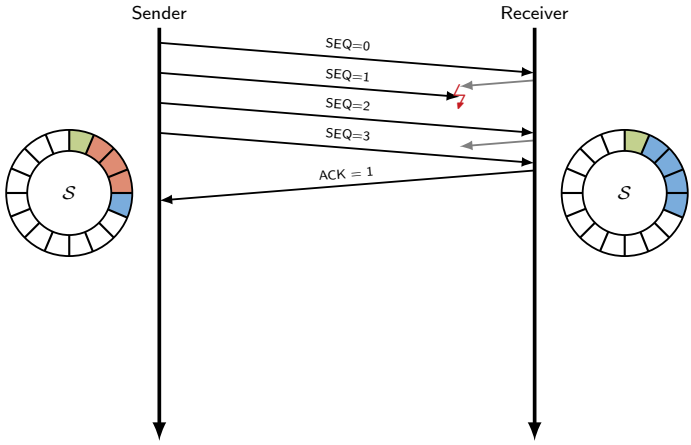
Go-back-N: $N = 16$, $w_s = 4$, $w_r = 4$



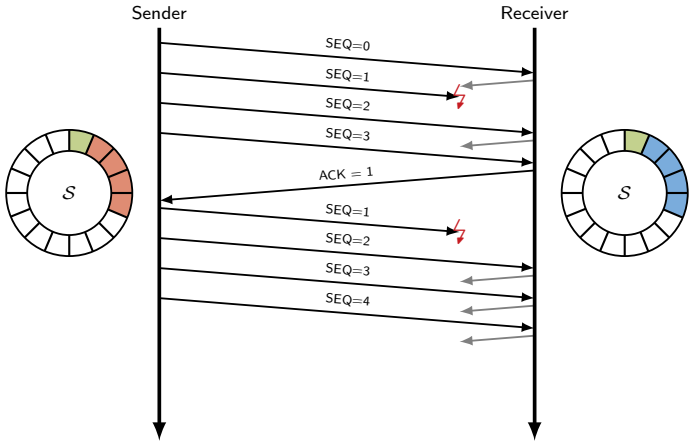
Go-back-N: $N = 16$, $w_s = 4$, $w_r = 4$



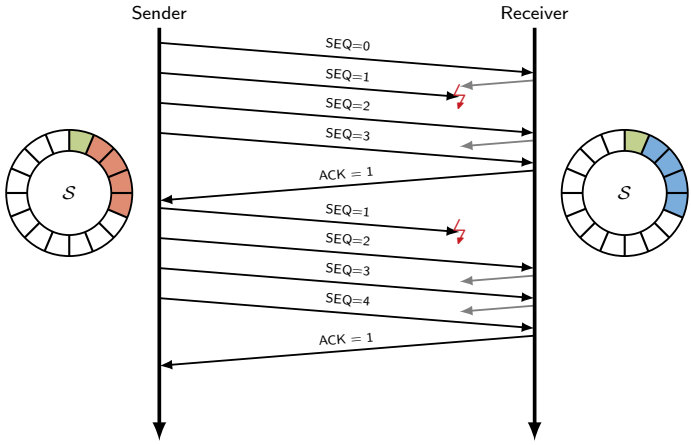
Go-back-N: $N = 16$, $w_s = 4$, $w_r = 4$



Go-back-N: $N = 16$, $w_s = 4$, $w_r = 4$



Go-back-N: $N = 16$, $w_s = 4$, $w_r = 4$



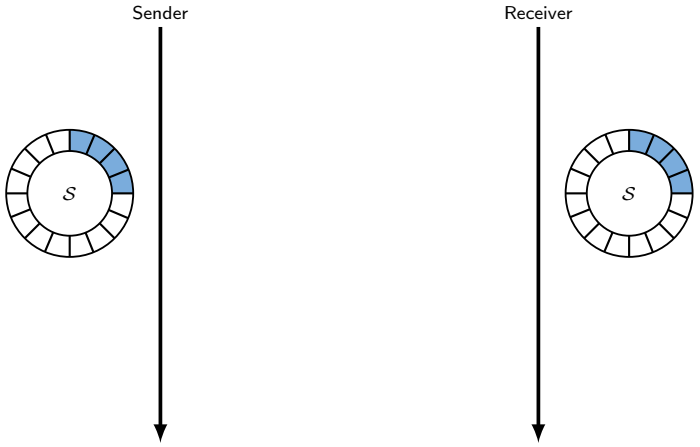
Notes on Go-Back-N

- Since the receiver always accepts only the next expected segment, a receive window of size $w_r = 1$ is sufficient in principle. Regardless, a sufficiently large receive buffer must be available for practical implementations.
- For a sequence number space of cardinality N, the following must always hold for the transmit window:

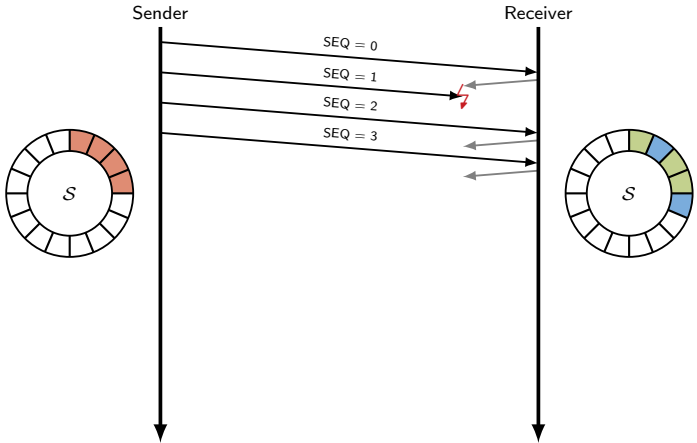
  $$w_s \leq N - 1.$$

  Otherwise, confusion may occur (see tutorial).
- Discarding segments that were successfully transmitted but did not arrive in the expected order makes the process easy to implement but less efficient.

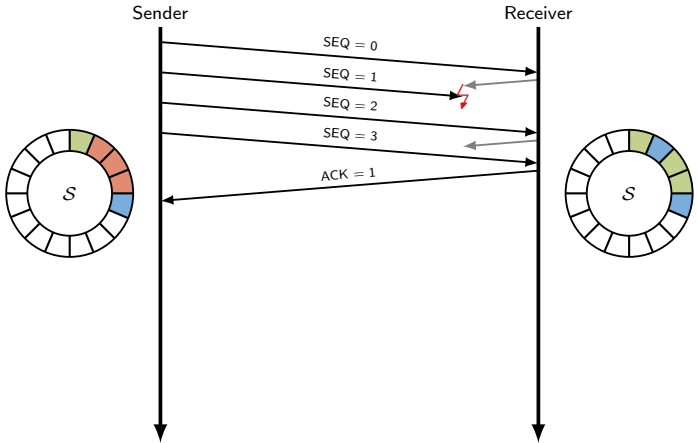
Selective repeat: $N = 16$, $w_s = 4$, $w_r = 4$



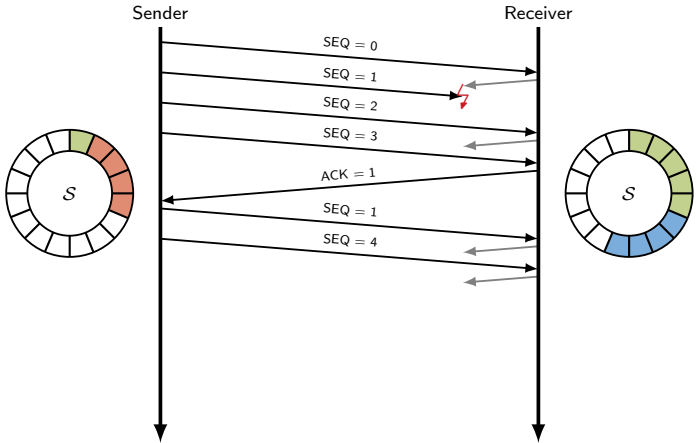
Selective repeat: $N = 16$, $w_s = 4$, $w_r = 4$



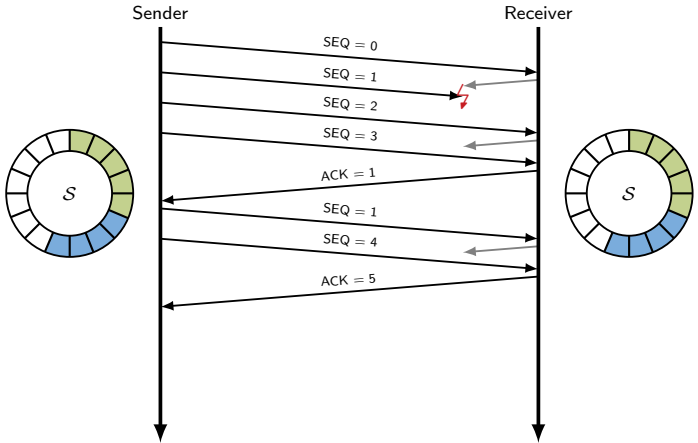
Selective repeat: $N = 16$, $w_s = 4$, $w_r = 4$



Selective repeat: $N = 16$, $w_s = 4$, $w_r = 4$



Selective repeat: $N = 16$, $w_s = 4$, $w_r = 4$



Notes on selective repeat

- Choosing $w_r = 1$ and w_s independent of w_r, selective repeat degenerates to Go-back-N.
- For a sequence number space of cardinality N, the following must always hold for the transmit window:

$$w_s \leq \left\lfloor \frac{N}{2} \right\rfloor .$$

  Otherwise, confusion may occur (see tutorial).

General notes

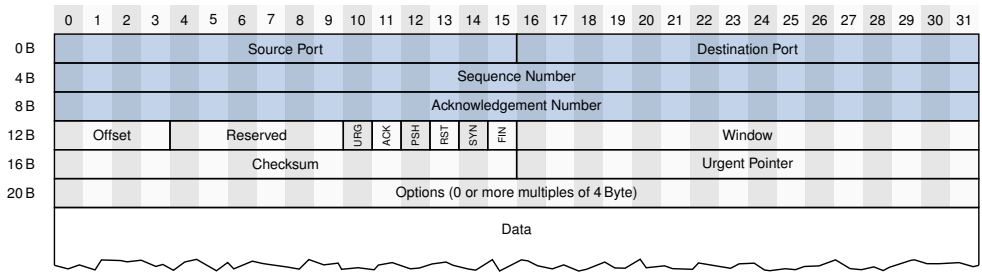
- In an implementation of these concepts, the receiver in particular requires a receive buffer whose size is adapted to the transmit and receive windows.
- For practical applications, the sizes of W_s and W_r are dynamically adjusted (see case study on TCP), enabling algorithms for congestion control and flow control at layer 4.

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



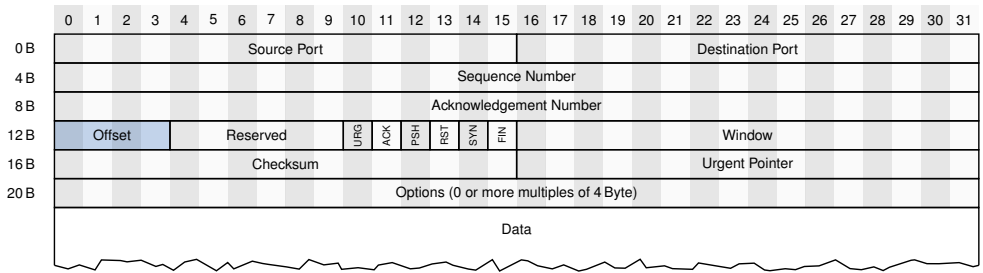
- Source and destination port are used analogously to UDP.
- Sequence and acknowledgement numbers are used for secured data transfer. TCP not acknowledges whole segments but individual Bytes (stream oriented transmission).

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



(Data) Offset

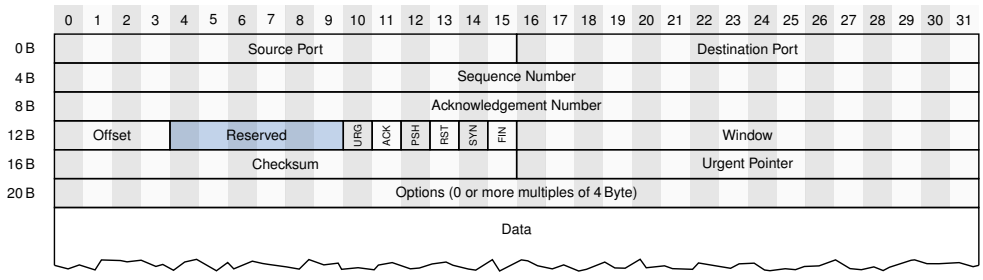
- Specifies the length of the TCP header in multiples of 4 B.
- The TCP header has variable length (options, cmp. IPv4 header)

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



Reserved

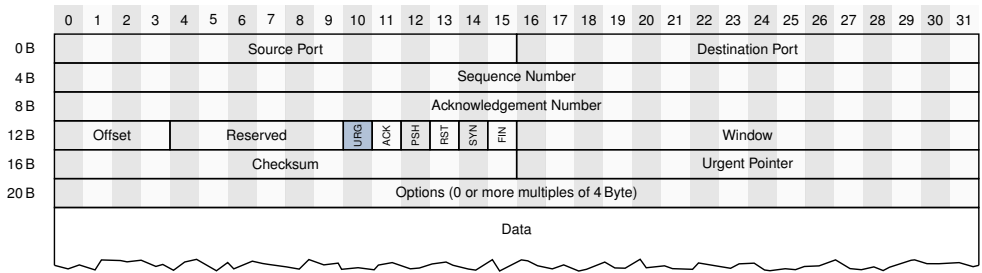
- Has no use in previous TCP versions. Must be set to 0 so that future TCP versions can use the field if needed.

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



Flag URG (“urgent”) (rarely used)

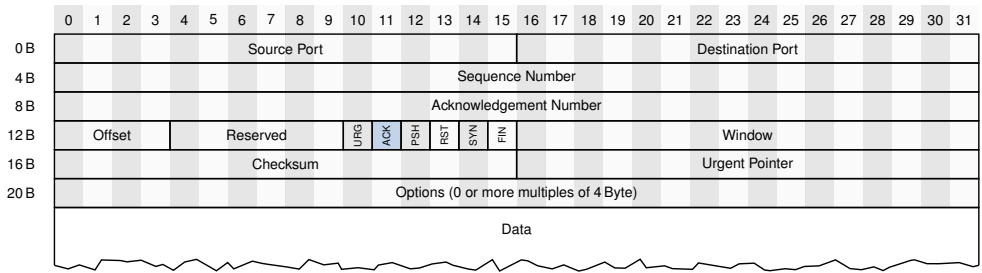
- If the flag is set, the data in the current TCP segment is immediately forwarded to higher layers starting with the first byte up to where the urgent pointer field points.

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



Flag ACK (“acknowledgement”)

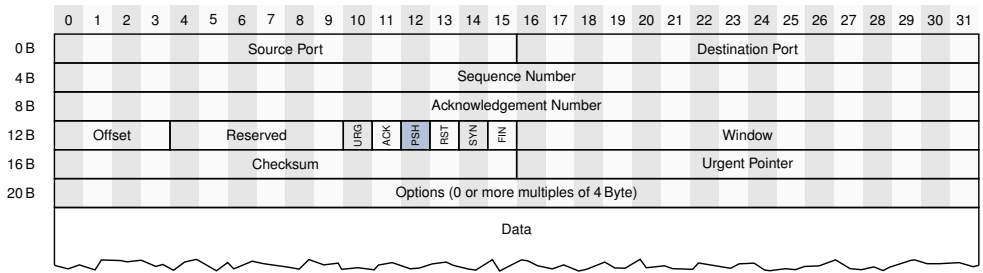
- If the flag is set, it is an acknowledgement.
- Acknowledgements can also be piggy backed on data segments, i. e., a segment containing data is transmitted from *A* to *B* and a segment previously sent from *B* to *A* is acknowledged at the same time.
- The acknowledgement number always specifies the next expected byte for TCP.

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



Flag PSH (“push”)

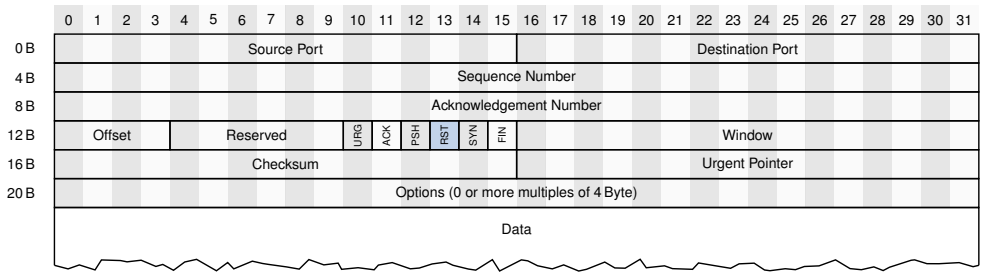
- If the flag is set, send- and receive-side buffers of the TCP stack are bypassed.
- Useful for interactive applications (e. g. telnet connections).

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



Flag RST (“reset”)

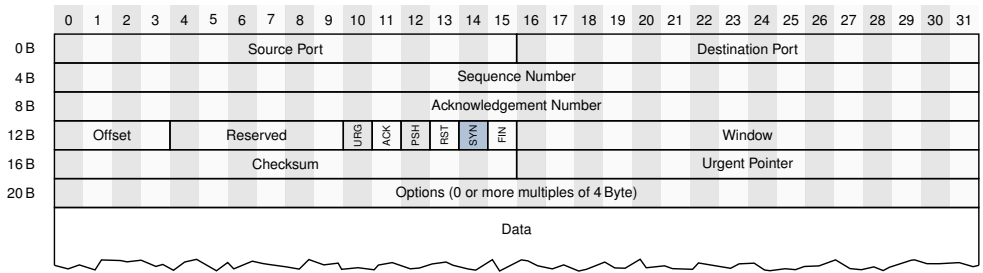
- Used to terminate a TCP connection without proper teardown.

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



Flag SYN (“synchronization”)

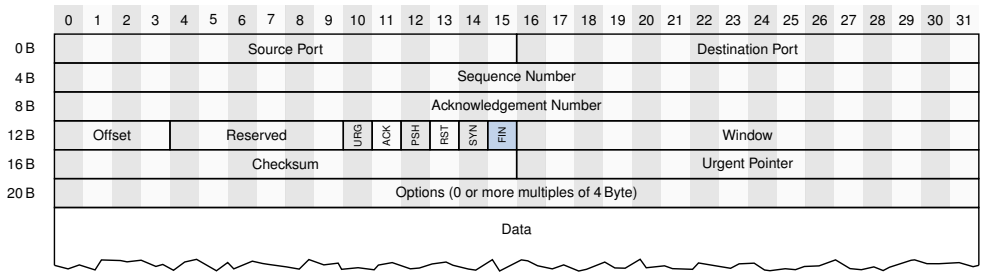
- If the flag is set, it is a segment that belongs to the connection setup (initial exchange of sequence numbers).
- A set SYN flag increments sequence and confirmation numbers by 1 although no payload is transported.

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



Flag FIN (“finish”)

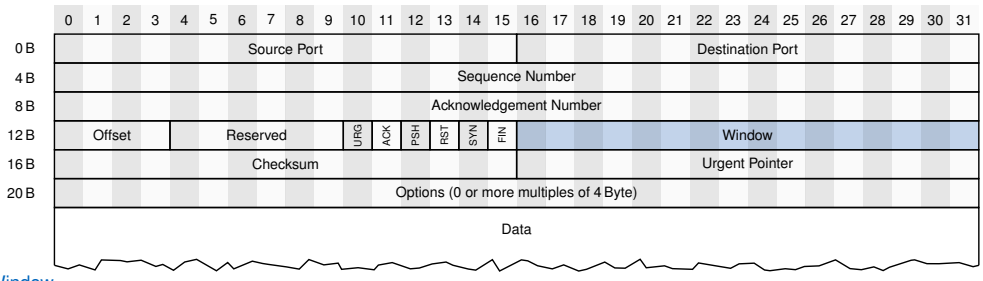
- If the flag is set, it is a segment that belongs to the teardown of a connection.
- A set FIN flag increments sequence and confirmation numbers by 1 although no user data is transported.

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



Receive Window

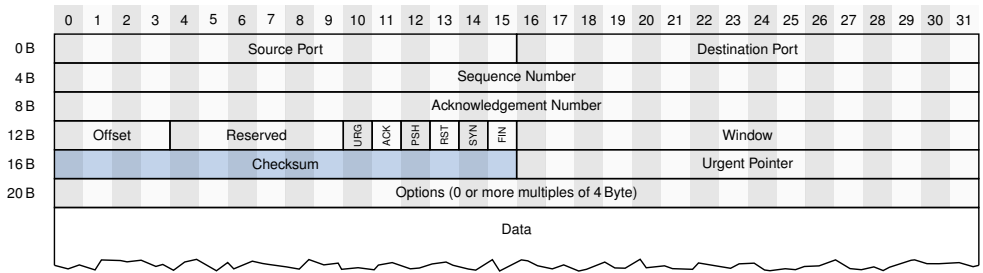
- Specifies the size of the current receive window W_r in Byte.
- Allows the receiver to throttle the transmit rate of the sender.

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



Checksum

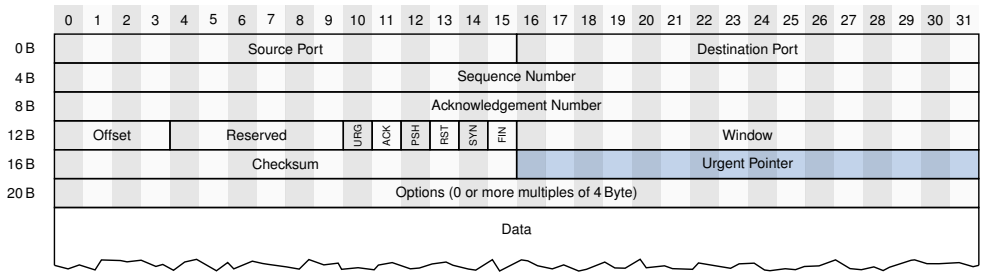
- Checksum over header and data.
- As with UDP, a pseudo header is used for the calculation.

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



Urgent pointer (rarely used)

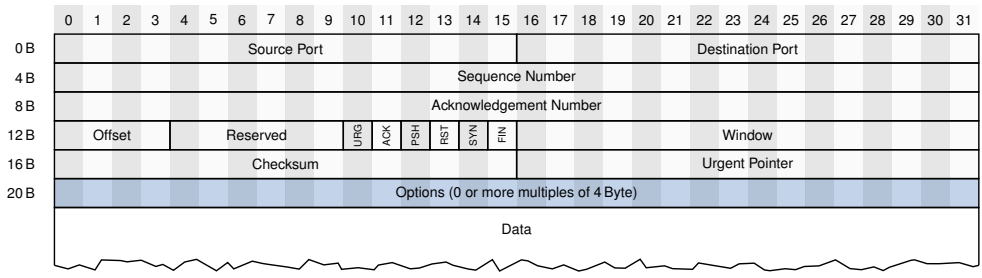
- Specifies the end of“urgent data” that start immediately after the header and are forwarded without delay to higher layers if the urgent flag is set.

Transmission Control Protocol (TCP)

TCP is the dominant transport protocol on the Internet (around 90 % of traffic on the internet [1]). It provides

- secured / stream-oriented transmission using sliding windows and selective repeat as well as
- mechanisms for flow and congestion control.

TCP header:



Options

- Additional options, e. g. window scaling (see tutorial), selective acknowledgements, or specification of the maximum segment size (MSS).

Notes regarding the MSS

- The MSS specifies the maximum size of a TCP segment (payload without TCP header).
- The MTU instead specifies the maximum size of the payload from the perspective of layer 2 (everything including the IP header).
- In practice, the MSS should be chosen such that no IP fragmentation is neccessary. [1]

Examples:

- MSS and FastEthernet
  - MTU is 1500 B.
  - 20 B are IPv4 header followed by 20 B TCP header (in case of no options)
  - The optimal MSS is therefore 1460 B.
- DSL dial-up connections
  - Between Ethernet and IP header a 8 B PPPoE header is inserted.
  - The optimal MSS is therefore 1452 B.
- VPN connections
  - Depending on the authentication and encryption protocols in use, there may be additionals headers.
  - The optimal MSS is not obvious, and probably not known at layer 4.

[1] That is not always possible since at Layer 4 it is not known which protocol is used at Layer 3 /2 and whether options are used or not. There may even be more protocols in between, e. g. PPPoE when using DSL dial-up connections.

TCP Flow Control
Flow control prevents an overload situation at the receiver. The receiver determines a maximum size of the send window.

- The receiver tells the sender the current size of the receive window W_r.
- W_r is the maximum number of Bytes the sender may transmit before receiving an acknowledgement.
- By reducing that value, the receiver can throttle the data rate of the transmitter.

TCP Flow Control
Flow control prevents an overload situation at the receiver. The receiver determines a maximum size of the send window.

- The receiver tells the sender the current size of the receive window W_r.
- W_r is the maximum number of Bytes the sender may transmit before receiving an acknowledgement.
- By reducing that value, the receiver can throttle the data rate of the transmitter.

TCP Congestion Control
Congestion control prevents overloads within the network. To that end, the transmitter must be able to detect or guess bottlenecks in the network and to adapt the size of the send window accordingly.

The sender therefore maintains an additional congestion window W_c:

- W_c is increased as long as data can be transferred without loss.
- W_c is decreased if segments are lost.
- For the actual size of the send window always holds $w_s = \min\{w_c, w_r\}$.

TCP Congestion Avoidance

TCP differentiates between two different phases of congestion avoidance:

1. Slow start:
   - For each segment that is acknowledged W_c is increased by one MSS.
   - This results in an exponential growth of the congestion windows until a threshold is hit.
   - Afterwards, TCP continues with the congestion avoidance phase.
2. Congestion avoidance:
   - For each acknowleged segment, W_c is increaes by only $(1/w_c)$ MSS, i. .e., after acknowledging a whole congestion window it is increased by exactly one MSS.
   - Such a window can be acknowledged earliest after 1 RTT.
   - This results in a linear increase of the congestion window dependent on the RTT.

TCP variants:

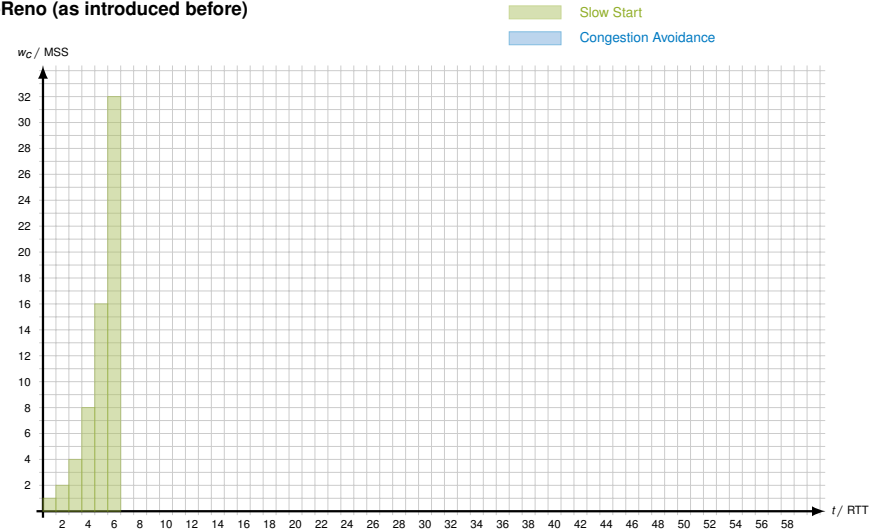
- We consider a simplified version of TCP based on TCP Reno.
- Different versions of TCP, e g. Tahoe, Reno, New Reno, Cubic, . . . , differ in details, but remain compatible to each other.
- TCP *Cubic*, for instance, increases the congestion window faster than other variants.

The following description refers to a simplified version of TCP *Reno*:

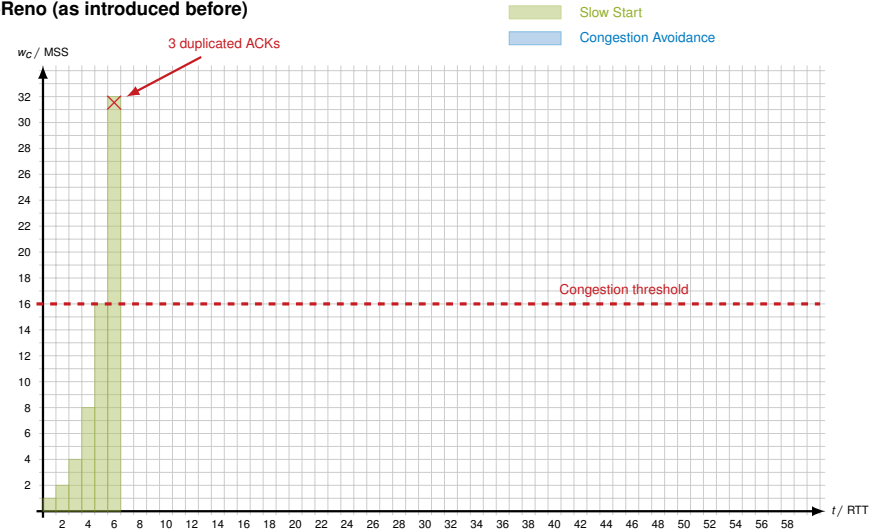
1. 3 duplicate ACKs ($\rightarrow$ something was probably lost, but segments still get through)
   - Set the threshold of the congestion phase to $w_c/2$.
   - Reduce W_c to the value of that threshold.
   - Restart the congestion avoidance phase.
2. Timeout ($\rightarrow$ no response, i. e., the network seems to be overloaded)
   - Set the threshold for the congestion phase to $w_c/2$.
   - Reset w_c to $w_c = 1$ MSS.
   - Restart with a slow start phase.

- The predecessor TCP Tahoe does, for instance, not differentiate between these two cases but always reacts with restarting the slow start phase.
- In general, all TCP variants are compatible to each other, However, due to differences in the adaption of the congestion control window they might negatively impact one another (fairness).

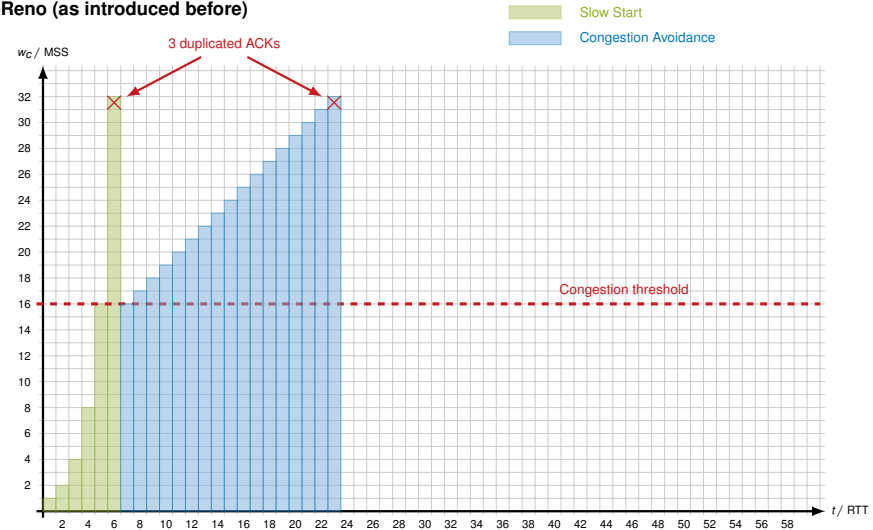
Example: TCP-Reno (as introduced before)



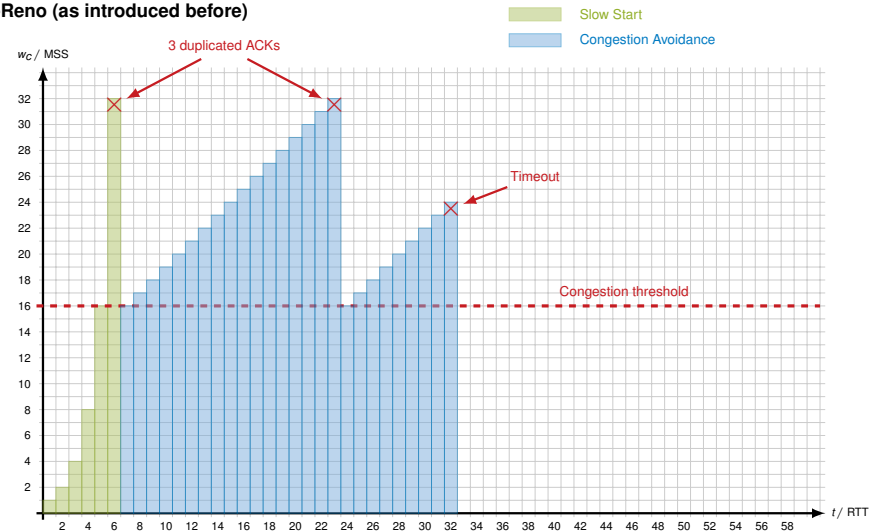
Example: TCP-Reno (as introduced before)



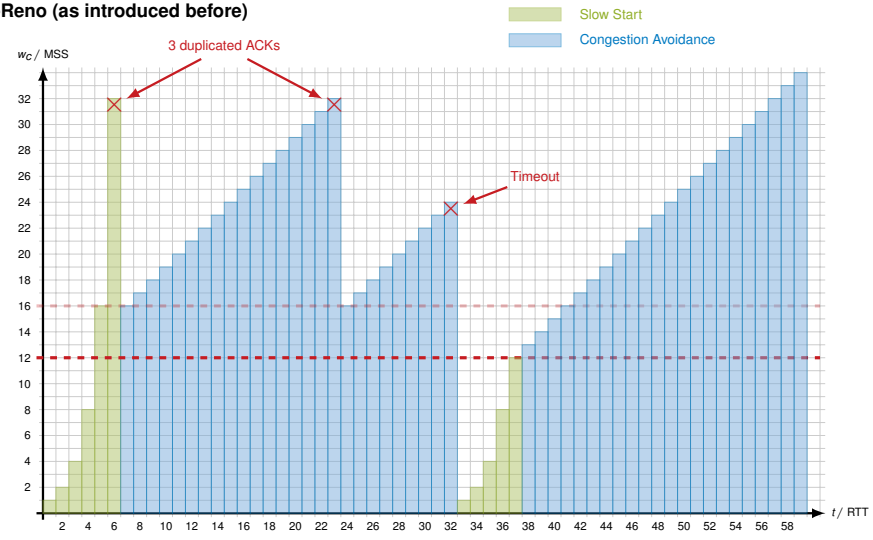
Example: TCP-Reno (as introduced before)



Example: TCP-Reno (as introduced before)



Example: TCP-Reno (as introduced before)



Notes TCP allows for secured connections. However, the mechanisms for error correction were developed with congestion within the network in mind:

- TCP always interprets the loss of segments as a result of congestion in the network and **not** as a result of bit errors due to an unreliable communication channel.
- The reaction is always a reduction of the average transmit rate by reducing the congestion window.
- If packet loss is consequence of random bit errors on the channel, this reaction is wrong.
- Due to the permanent reduction of the window and restarts with slow starts, the send window would never grow to the desired size.
- TCP would not perform well on links with 1 % packet loss rate (that is not due to congestion).

$\Rightarrow$ Layers 1 – 3 must provide a link of "sufficient quality" for TCP.

- In practice, packet loss rates of 10^{-3} and lower is desireable.
- If necessary, layer 2 needs to implement retransmission schemes (automated repeat and request) as, for instance, IEEE 802.11 does.

Chapter 4: Transport layer

Network Address Translation (NAT)

In Chapter 3 we learned that

- IP addresses are used for end to end connections,
- they have to be globally unique for that reason, and
- in particular IPv4 free addresses are very scarce today.

Question: Do IP addresses always have to be unique?

Network Address Translation (NAT)

In Chapter 3 we learned that

- IP addresses are used for end to end connections,
- they have to be globally unique for that reason, and
- in particular IPv4 free addresses are very scarce today.

Question: Do IP addresses always have to be unique?

Answer: No, IP addresses do not have to be unique if

- no communication with hosts in the internet is required or
- private addresses, which are not unique by nature, are somehow translated to public addresses.

Definition: NAT
Network Address Translation (NAT) refers in general to approaches to map $N \geq 1$ IP addresses to $M \geq 1$ different IP addresses. With IPv4, the most common use case is the mapping of N private to M public (globally unique) addresses:

- $N \leq M$: The mapping is either static or dynamic. Each private address is mapped to at least one public address. (That is the unusual case.)
- $N > M$: In this case, a public address is used by different computers at the same time. The differentiation of different connections can be realized by port multiplexing. The most common case is $M = 1$, e. g. a private internet connection.

Network Address Translation (NAT)

What are private IP addresses?

Private IP addresses are special address ranges that

- are released for private use without prior registration,
- thus may be used in multiple (local) networks,
- cannot be used for end to end addressing between public networks for that reason, and
- therefore are not routed in the internet.

The private address ranges for IPv4 are:

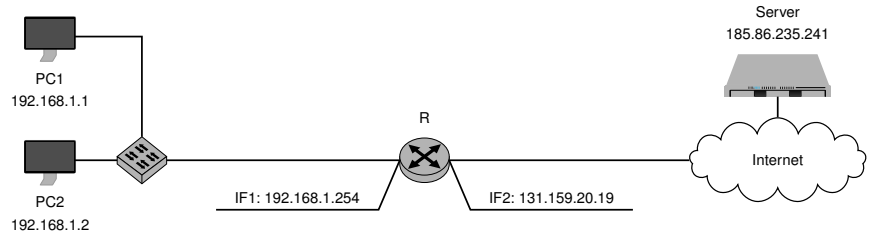
- 10.0.0.0 / 8
- 172.16.0.0 / 12
- 169.254.0.0 / 16
- 192.168.0.0 / 16

The range 169.254.0.0 / 16 is used for automatic address assignment (Automatic Private IP Adressing):

- When a computer without statically configured address starts, it normally tries to contact a DHCP server.
- If no DHCP server is found, the operating systems assigns a random IP address from this block.
- Before using this address, an ARP request for that address is sent.
- If that address is already in use, an ARP reply would be received.
- In that case, a different address is chosen and the process repeated.

How does NAT work?

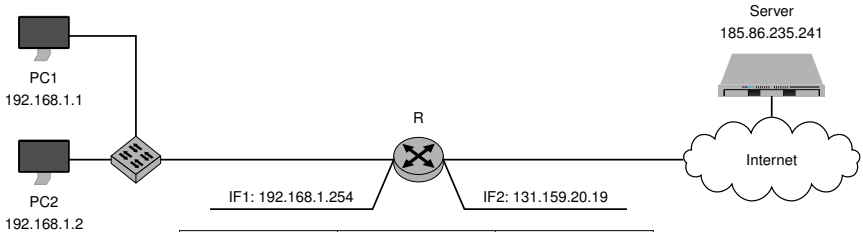
Typically, routers perform network address translation:



- PC1, PC2, and R can communicate with each other using private IP addresses in the 192.168.1.0/24 subnet.
- R can be reached globally via its public address 131.159.20.19.
- PC1 and PC2 cannot communicate directly with other hosts on the Internet because of their private addresses.
- Hosts on the internet cannot reach PC1 or PC2 either – even if they know that PC1 and PC2 are behind R and the global address of R is known.

Network Address Translation (NAT)

PC1 accesses a web page located on the server with IP address 185.86.235.241:

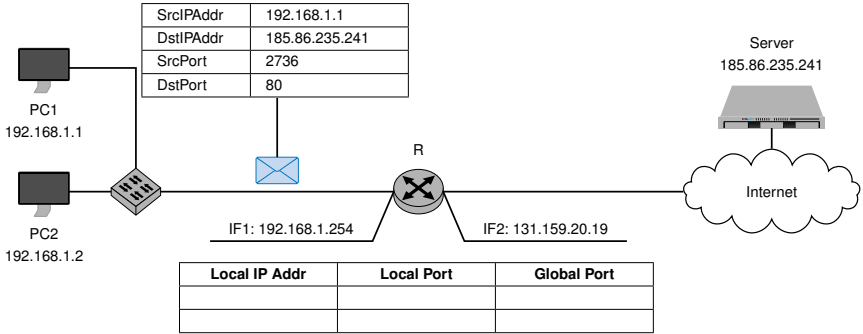


Local IP Addr	Local Port	Global Port

- The NAT table of R is initially empty.

Network Address Translation (NAT)

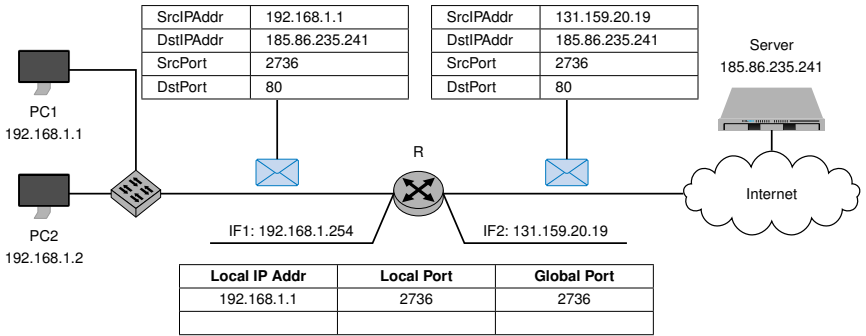
PC1 accesses a web page located on the server with IP address 185.86.235.241:



- The NAT table of R is initially empty.
- PC1 sends a packet (TCP SYN) to the server:
  - PC1 uses its private IP address as sender address.
  - The source port is chosen randomly from the range [1024,65535] (so called Ephemeral Ports).
  - The destination port is determined by the layer 7 protocol (80 = HTTP).

Network Address Translation (NAT)

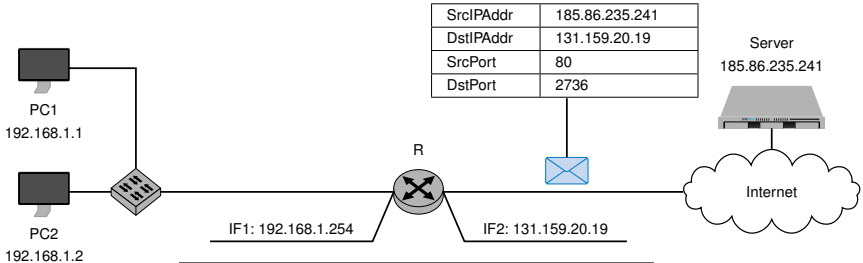
PC1 accesses a web page located on the server with IP address 185.86.235.241:



- The NAT table of R is initially empty.
- PC1 sends a packet (TCP SYN) to the server:
  - PC1 uses its private IP address as sender address.
  - The source port is chosen randomly from the range [1024,65535] (so called Ephemeral Ports).
  - The destination port is determined by the layer 7 protocol (80 = HTTP).
- Address translation at R:
  - R replaces the sender address with its own global address.
  - Provided that the source port does not lead to a collision in the NAT table, it is retained.
  - R creates a new entry in its NAT table documenting the changes to the packet.

Network Address Translation (NAT)

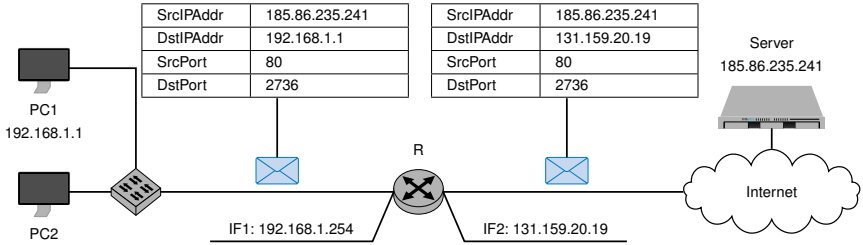
Answer from the server to PC1:



- The server generates a response:
  - The server does not know about the address translation and considers R to be PC1.
  - The recipient address is therefore the public IP address of R, the destination port is the source port translated by R from the previous message.

Network Address Translation (NAT)

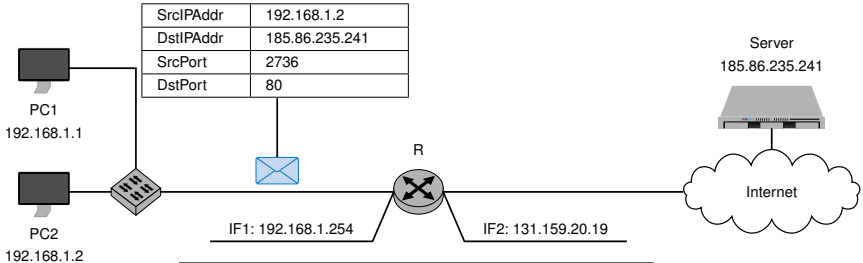
Answer from the server to PC1:



- The server generates a response:
  - The server does not know about the address translation and considers R to be PC1.
  - The recipient address is therefore the public IP address of R, the destination port is the source port translated by R from the previous message.
- R reverts the address translation:
  - The NAT table is searched for the destination port number in the Global Port column, this is translated back to Local Port and the destination IP address of the packet is exchanged for the private IP address of PC1.
  - The modified packet is forwarded to PC1.
  - Like the server, PC1 does not know about the address translation.

Network Address Translation (NAT)

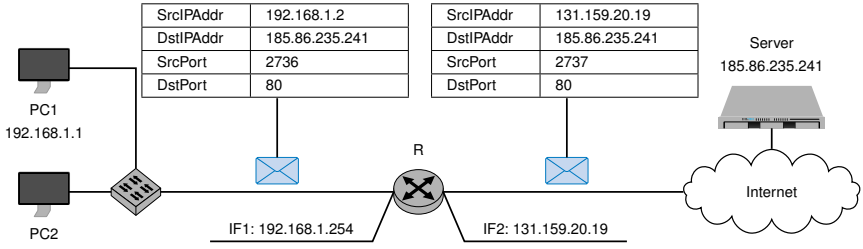
PC2 now also accesses the server:



- PC2 also accesses the server:
  - By chance, PC2 chooses the same source port as PC1 (2736).

Network Address Translation (NAT)

PC2 now also accesses the server:



Local IP Addr	Local Port	Global Port
192.168.1.1	2736	2736
192.168.1.2	2736	2737

- PC2 also accesses the server:
  - By chance, PC2 chooses the same source port as PC1 (2736).
- Address translation at R:
  - R notices that there is already an entry belonging to PC1 for local port 2736.
  - R creates a new entry in the NAT table, where a random value is chosen for the global port (e. g. the original port of PC2 + 1).
  - The packet from PC2 is modified accordingly and forwarded to the server.
- From the server's point of view, the "computer" R has simply established two TCP connections.

Network Address Translation (NAT)

A router could include additional information in the NAT table:

- Destination IP and destination port
- The layer 4 protocol in use (e. g. TCP, UDP)
- The own global IP address (useful if a router has more than one global IP address).

Depending on the information stored, a distinction is made between different types of NAT. The variant just discussed (plus a notation of the protocol in the NAT table) is called Full Cone NAT.

Properties of Full Cone NAT:

- For incoming connections, there is no check of the sender IP address or the sender port, since the NAT table contains only the destination port and the associated IP address or port number in the local network.
- Thus, once an entry exists in the NAT table, an internal host from the Internet can also be reached via this entry by anyone who sends a TCP or UDP packet to the correct port number.

Other NAT variants:

- Port Restricted NAT
- Address Restricted NAT
- Port and Address Restricted NAT
- Symmetric NAT

General notes

- Is NAT a firewall [2] [1]?
  - **No!**
  - Restrictive NAT variants do provide basic protection in that they do **not** (and cannot) allow incoming connections without first establishing a connection from the local network, but this should not be combined with the functions of a firewall.
  - Filtering beyond this (as would be the case with a firewall) does not take place.

[1] Firewalls allow incoming and outgoing traffic to be filtered based on IP addresses, port numbers as well as previous events („Stateful Firewalls“). These functions are often performed by routers. In some cases, the content of individual messages is also inspected („Deep Packet Inspection“).

General notes

- Is NAT a firewall [2][1]?
  - **No!**
  - Restrictive NAT variants do provide basic protection in that they do **not** (and cannot) allow incoming connections without first establishing a connection from the local network, but this should not be combined with the functions of a firewall.
  - Filtering beyond this (as would be the case with a firewall) does not take place.
- How many entries can a NAT table hold?
  - In the simplest case (Full Cone NAT), the theoretical maximum limit is 2^{16} per transport protocol (TCP and UDP) and per global IP address.
  - For more complex NAT types, more combinations are possible by including the destination ports.
  - In practice, the size is limited by the capabilities of the router (a few 1000 mappings).

[1] Firewalls allow incoming and outgoing traffic to be filtered based on IP addresses, port numbers as well as previous events („Stateful Firewalls“). These functions are often performed by routers. In some cases, the content of individual messages is also inspected („Deep Packet Inspection“).

General notes

- Is NAT a firewall [2][1]?
  - **No!**
  - Restrictive NAT variants do provide basic protection in that they do **not** (and cannot) allow incoming connections without first establishing a connection from the local network, but this should not be combined with the functions of a firewall.
  - Filtering beyond this (as would be the case with a firewall) does not take place.
- How many entries can a NAT table hold?
  - In the simplest case (Full Cone NAT), the theoretical maximum limit is 2^{16} per transport protocol (TCP and UDP) and per global IP address.
  - For more complex NAT types, more combinations are possible by including the destination ports.
  - In practice, the size is limited by the capabilities of the router (a few 1000 mappings).
- Are mappings deleted from the NAT table again?
  - Dynamically created mappings are deleted after a certain period of inactivity
  - Under certain circumstances, a NAT-capable router will also remove mappings immediately when it detects a TCP connection drop (implementation-dependent).

[1] Firewalls allow incoming and outgoing traffic to be filtered based on IP addresses, port numbers as well as previous events („Stateful Firewalls“). These functions are often performed by routers. In some cases, the content of individual messages is also inspected („Deep Packet Inspection“).

General notes

- Is NAT a firewall [2][1]?
  - **No!**
  - Restrictive NAT variants do provide basic protection in that they do **not** (and cannot) allow incoming connections without first establishing a connection from the local network, but this should not be combined with the functions of a firewall.
  - Filtering beyond this (as would be the case with a firewall) does not take place.
- How many entries can a NAT table hold?
  - In the simplest case (Full Cone NAT), the theoretical maximum limit is 2^{16} per transport protocol (TCP and UDP) and per global IP address.
  - For more complex NAT types, more combinations are possible by including the destination ports.
  - In practice, the size is limited by the capabilities of the router (a few 1000 mappings).
- Are mappings deleted from the NAT table again?
  - Dynamically created mappings are deleted after a certain period of inactivity
  - Under certain circumstances, a NAT-capable router will also remove mappings immediately when it detects a TCP connection drop (implementation-dependent).
- Can entries in the NAT table also be generated manually?
  - Yes, this is known as Port Forwarding.
  - In this way, it becomes possible to operate a server that is publicly accessible on a specific port behind a NAT.

[1] Firewalls allow incoming and outgoing traffic to be filtered based on IP addresses, port numbers as well as previous events („Stateful Firewalls“). These functions are often performed by routers. In some cases, the content of individual messages is also inspected („Deep Packet Inspection“).

NAT and ICMP

- NAT uses port numbers of the transport protocol.
- What if the transport protocol has no port numbers or IP packets are sent without TCP/UDP headers, e. g. ICMP?

NAT and ICMP

- NAT uses port numbers of the transport protocol.
- What if the transport protocol has no port numbers or IP packets are sent without TCP/UDP headers, e. g. ICMP?

Answer: The ICMP ID can be used instead of the port numbers.

NAT and ICMP

- NAT uses port numbers of the transport protocol.
- What if the transport protocol has no port numbers or IP packets are sent without TCP/UDP headers, e. g. ICMP?

Answer: The ICMP ID can be used instead of the port numbers.

Problem: Traceroute does not work with some virtualization solutions, e.g. when using older versions of the Virtualbox NAT implementation.

- Traceroute is based on ICMP TTL Exceeded messages.
- These messages (unlike an ICMP Echo Reply) do not have an ICMP ID.
- This is because any IP packet (not necessarily an ICMP echo request) can trigger an ICMP TTL Exceeded, and this (as in the case of a TCP packet) naturally has no ICMP ID.
- Instead, the time-exceeded carries the full IP header and the first 8 bytes of the payload of the packet that triggered the time-exceeded.
- In the case of a TTL Exceeded, a NAT implementation would have to search in these first 8 bytes for the ICMP ID of an echo request or for the port numbers of a transport protocol in order to be able to reverse the translation.
- This is exactly the back translation that older versions of Virtualbox's NAT implementation do not perform.

NAT and IPv6

- NAT can also be used for IPv6.

Network Address Translation (NAT)

NAT and IPv6

- NAT can also be used for IPv6.

Prefix translation

- NAT for IPv6 has specific problems and challenges. RFC 6296 specifies IPv6-to-IPv6 Network Prefix Translation.
- This creates a 1:1 mapping of addresses.
  - This would also be possible with IPv4 to a limited extent (provided that sufficient public addresses are available to an organization),
  - but in most cases not meaningful.
- This can be used to translate Unique-Local Unicast addresses (`fc00::/7`, i.e. private IPv6 addresses) into globally valid addresses.
- The translation is done on prefixes:
  - For example, an internal prefix `fd01:0203:0405::/48` is mapped to the global prefix `2001:db8:0001::/48`.
- No layer 4 features (ports, identifiers) are used.
- The translation is stateless, except for the configuration of the address prefixes. There is no need for a NAT table.
- To avoid having to modify Layer 4 checksums due to address translation, the address translation can be chosen so that the original checksums are still correct.

Network Address Translation (NAT)

NAT and IPv6

- NAT can also be used for IPv6.

Prefix translation

- NAT for IPv6 has specific problems and challenges. RFC 6296 specifies IPv6-to-IPv6 Network Prefix Translation.
- This creates a 1:1 mapping of addresses.
  - This would also be possible with IPv4 to a limited extent (provided that sufficient public addresses are available to an organization),
  - but in most cases not meaningful.
- This can be used to translate Unique-Local Unicast addresses (`fc00::/7`, i.e. private IPv6 addresses) into globally valid addresses.
- The translation is done on prefixes:
  - For example, an internal prefix `fd01:0203:0405::/48` is mapped to the global prefix `2001:db8:0001::/48`.
- No layer 4 features (ports, identifiers) are used.
- The translation is stateless, except for the configuration of the address prefixes. There is no need for a NAT table.
- To avoid having to modify Layer 4 checksums due to address translation, the address translation can be chosen so that the original checksums are still correct.

Usage of NAT with IPv6

- However, a common reason for NAT (the address shortage with IPv4) is not present with IPv6.

Chapter 4: Transport layer

References

[1] Analyzing UDP Usage in Internet Traffic, 2009.
http://www.caida.org/research/traffic-analysis/tcpudpratio/.

[2] L. Zhang.
A Retrospective View of NAT.
IETF Journal, 3(2), 2007.
http://www.internetsociety.org/articles/retrospective-view-nat.