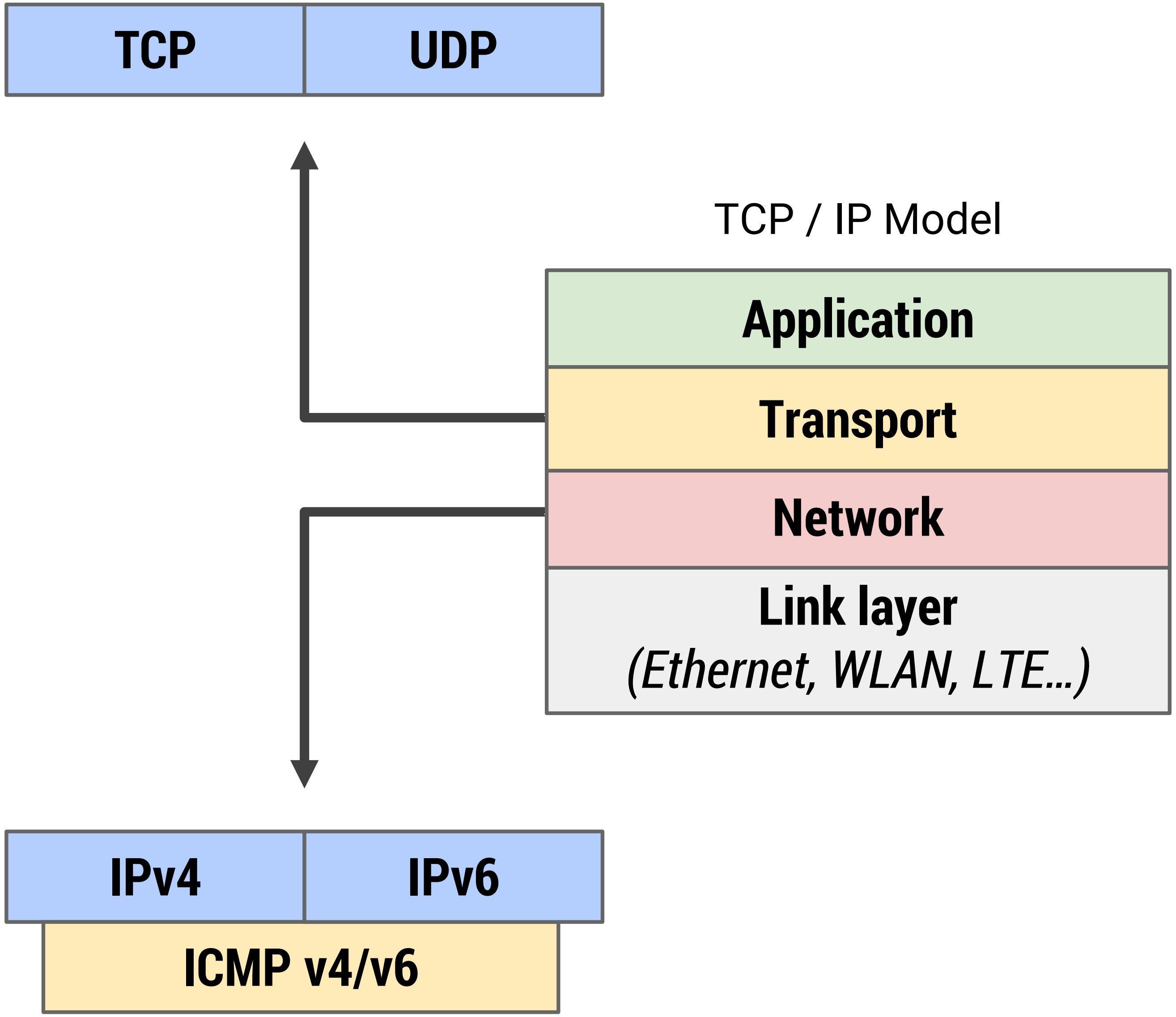
Network & Transport Layer

Computer Organization and Networks 2020

Johannes Feichtner
johannes.feichtner@iaik.tugraz.at

Outline

- IPv6
  - Header Structure
  - Addressing
- ICMPv6 Functionality
  - NDP
- TCP & UDP
  - Error Handling?
  - Flow & Congestion Control



IPv4 Depletion


With the Americas running out of IPv4, it's official: The Internet is full

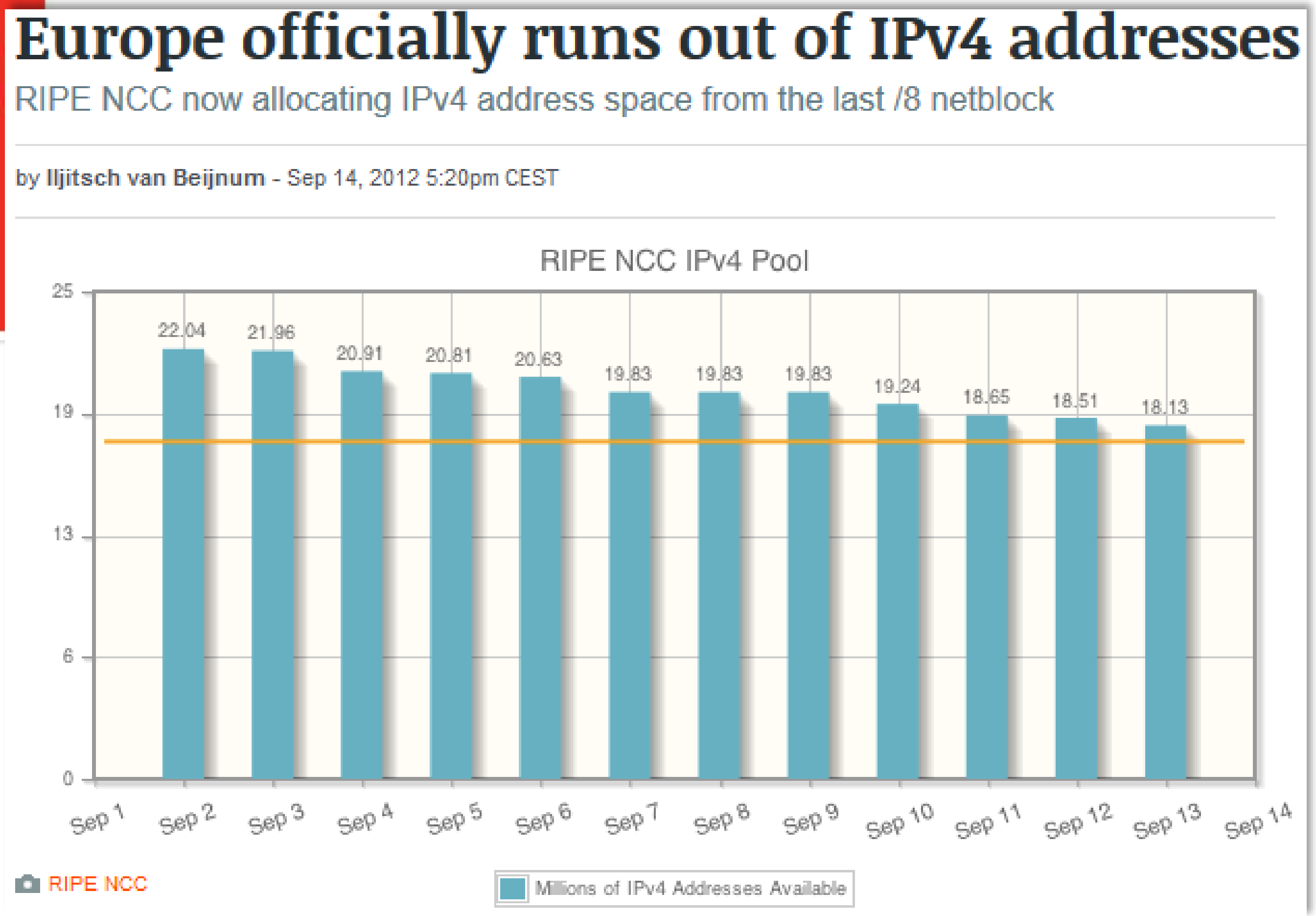
Where did all those IP addresses go?

by Iljitsch van Beijnum - Jun 12, 2014 10:50pm CEST

269

Source: http://goo.gl/13Rswl

Source: http://goo.gl/AglMMw


Europe officially runs out of IPv4 addresses

RIPE NCC now allocating IPv4 address space from the last /8 netblock

by Iljitsch van Beijnum - Sep 14, 2012 5:20pm CEST

What happened?

- Global registries have no more IPv4 addresses to assign
- Until then: 200 mio. new addresses were assigned / year

The RIPE NCC has run out of IPv4 Addresses

Today, at 15:35 (UTC+1) on 25 November 2019, we made our final /22 IPv4 allocation from the last remaining addresses in our available pool. We have now run out of IPv4 addresses.

Our announcement will not come as a surprise for network operators - IPv4 run-out has long been anticipated and planned for by the RIPE community. In fact, it is due to the community's responsible stewardship of these resources that we have been able to provide many thousands of new networks in our service region with /22 allocations after we reached our last /8 in 2012.

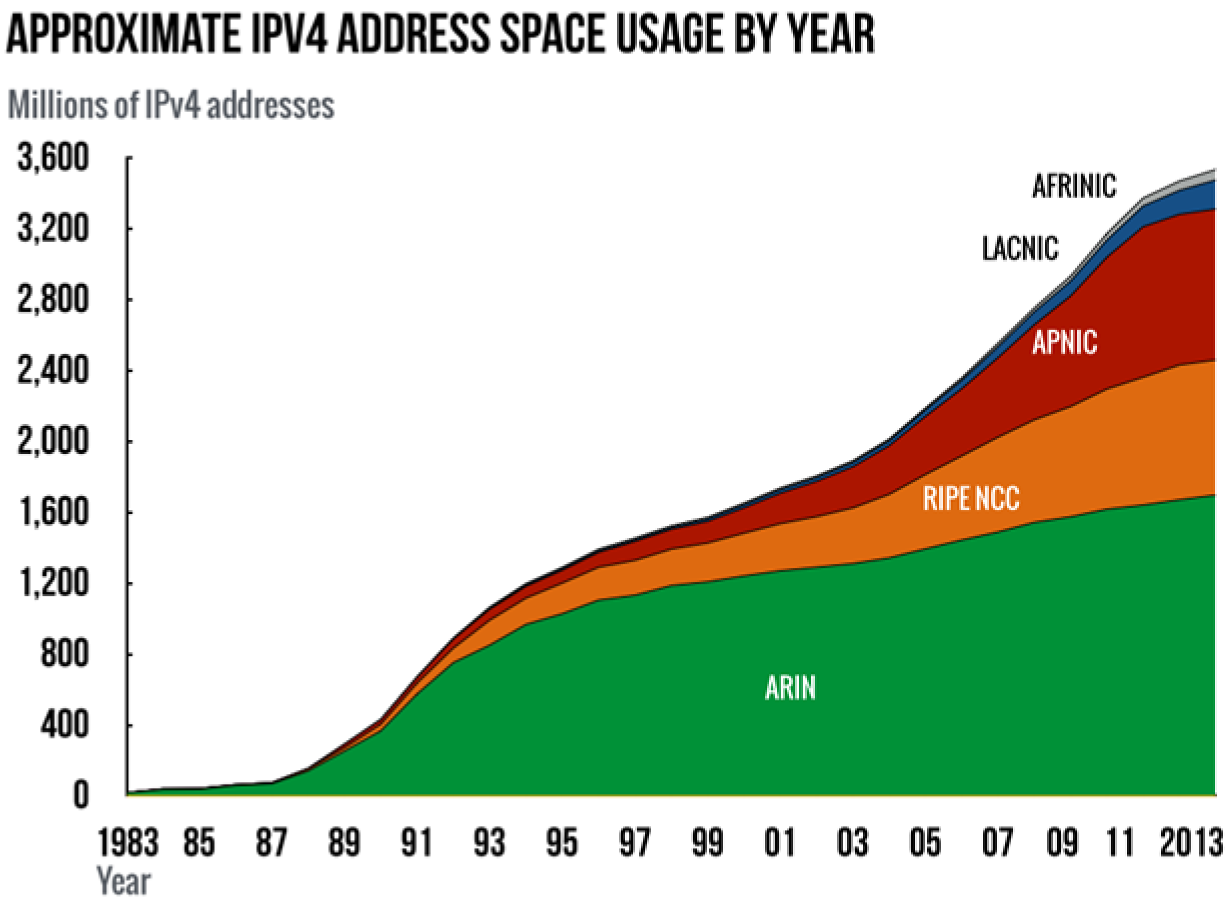
Recovered IPv4 Addresses and the Waiting List

Even though we have run out, we will continue to recover IPv4 addresses in the future. These will come from organisations that have gone out of business or are closed, or from networks that return addresses they no longer need. These addresses will be allocated to our members (LIRs) according to their position on a new waiting list that is now active.

IPv4 Depletion

Problem

- Rise of always-on connections
  - Broadband instead of dial-up
  - Mobile devices
- Inefficient address use
  - Often far more addresses allocated than needed, e.g. /8 block
  - Not all addresses usable in subnets
- NAT makes PCs unaddressable from outside



Source: http://goo.gl/13Rswl

→ Solution: IPv6

IPv6 Design Goals

In the 90s…

- Only three classes of IP addresses
  - Class A (/8): 128 networks for 16.277.216 connected hosts each
  - Class B (/16): 16.384 networks for 65.536 computers
  - Class C (/24): 2.097.152 networks for 256 hosts each
- Classes replaced by CIDR in order to reduce amount of wasted addresses

→ *Need for larger address space!*

- IPv4: 32-bit → max. 2^{32} addresses
- IPv6: 128-bit → max. 2^{128} addresses

IPv6 Design Goals

Aside from more addresses…

- Emphasis on ***end-to-end** principle* → global reachability *without* NAT

- **Simplify** the processing of IPv6 packet **headers** for routers
  - Less computational effort needed for forwarding

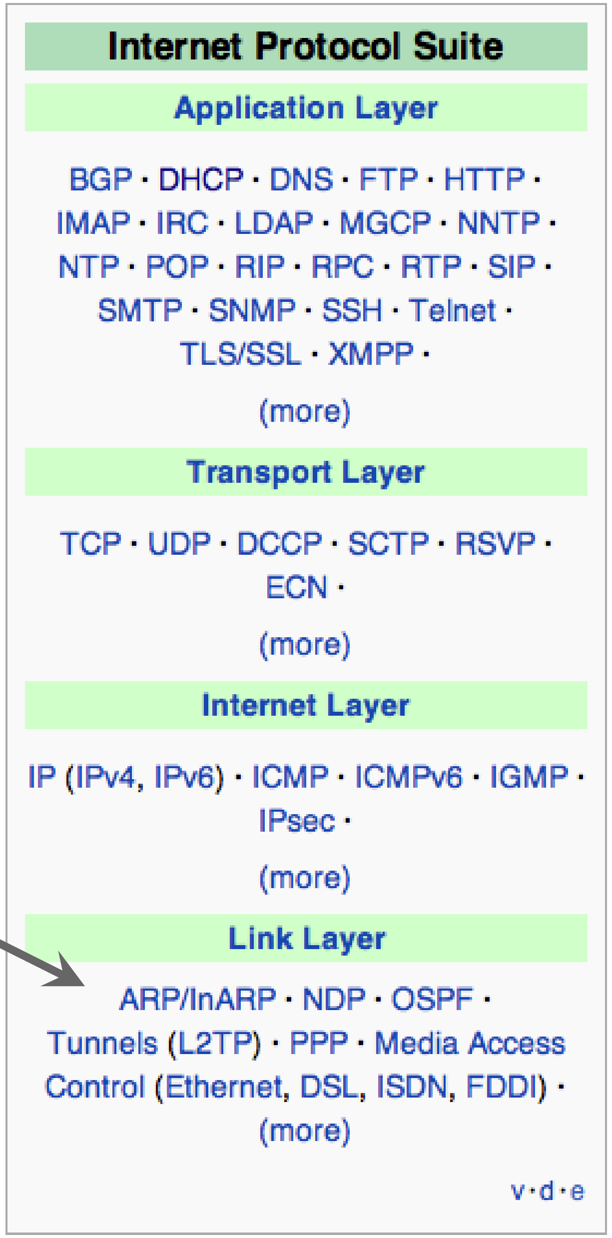
- Self-configuration of nodes in networks
  - „Stateless address autoconfiguration“ (**SLAAC**)
  - Instead of stateful DHCP, use ICMPv6 and „Neighbor Discovery Protocol“ (NDP)

- Native support for network techniques:
  Quality of Service, Multicasting, IPSec

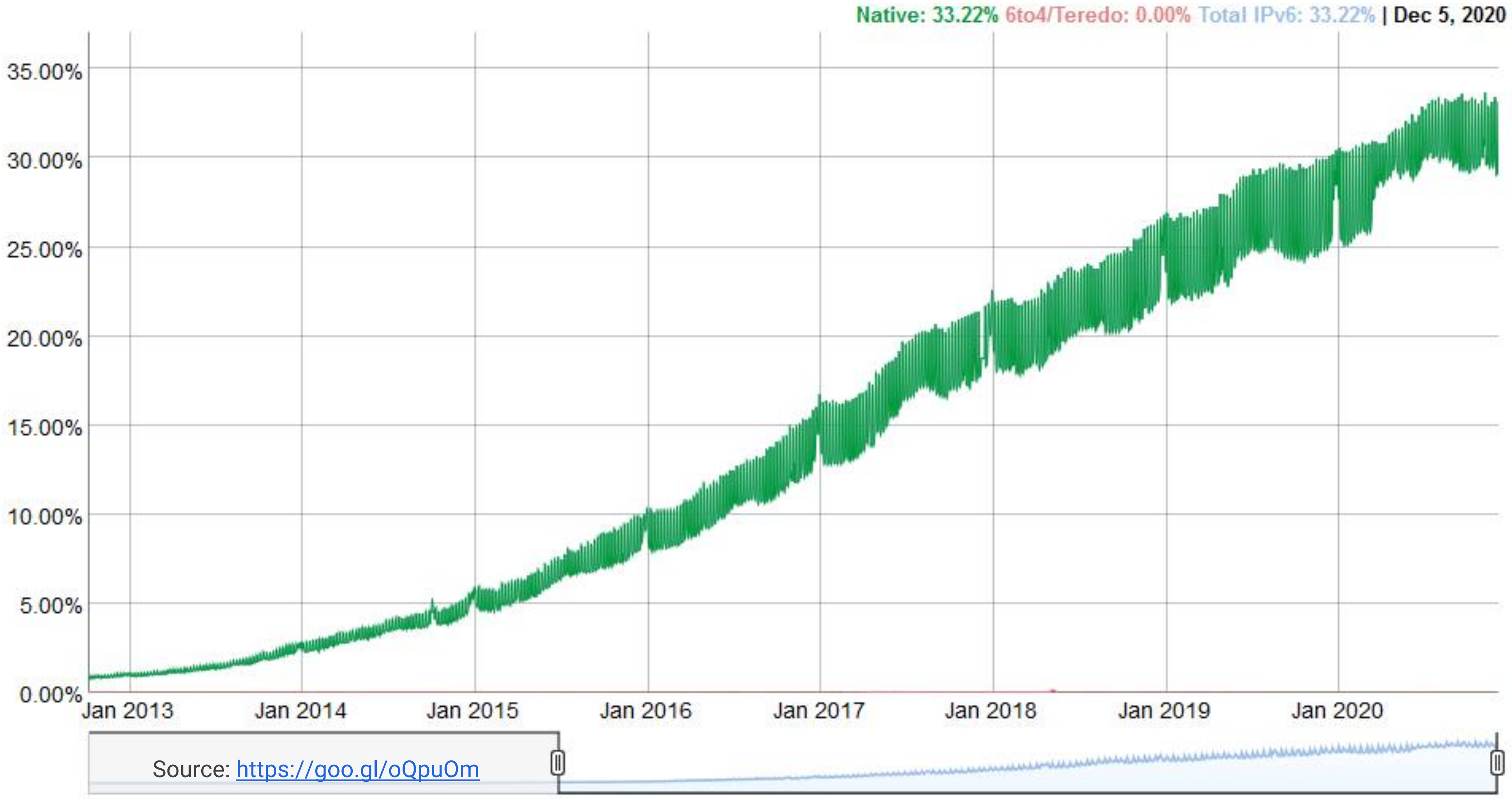
IPv6 Properties

- IPv6 replaces IPv4
  - Still connection-less packet switching (datagrams)
  - 128-bit addresses vs. 32-bit addresses
- IPv6 handled as separate protocol family
  - Adaptions mostly on network layer but also others
  - ICMPv6 replaces ICMPv4, IGMP and ARP (!!)
  - DHCP → DHCPv6, e.g. for DNS server discovery
  - DNS adapted by AAAA record
- No more NAT needed
  - Good for Peer-to-peer application development

Privacy problem!



IPv6 Adoption Rate



Source: https://goo.gl/oQpuOm

IPv6 Addressing

IPv6 Addresses

Same principle as for IPv4 but…

- Addresses no longer identify hosts but network interfaces
  - IPv4: „Network address“ → IPv6: „Prefix“
  - IPv4: „Host address“ → IPv6: „Interface address“

- No broadcast addresses anymore → now performed via Multicast

- One interface can be assigned multiple addresses (of different scope types)

```
ip -6 addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536
    inet6 ::1/128 scope host
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
    inet6 2001:11d0:8:650b::3/64 scope global
    inet6 fe80::250:56ff:fe05:866c/64 scope link
```

IPv6 Notation

- 128-bit addresses → max. 2^{128} addresses
- Hexa-decimal notation: x:x:x:x:x:x:x:x → Eight 16-bit pieces, separated by :
  Example: `201a:0000:0000:0945:daa2:5eff:fe8e:e553`

Rules

- A set of consecutive null blocks can be replaced by two colons
  - Only once per address in order to prevent ambiguous representations

```
201a::0945:daa2:5eff:fe8e:e553
```

- Leading zeros within each 16-bit part can be removed, e.g.

```
201a::945:daa2:5eff:fe8e:e553
```

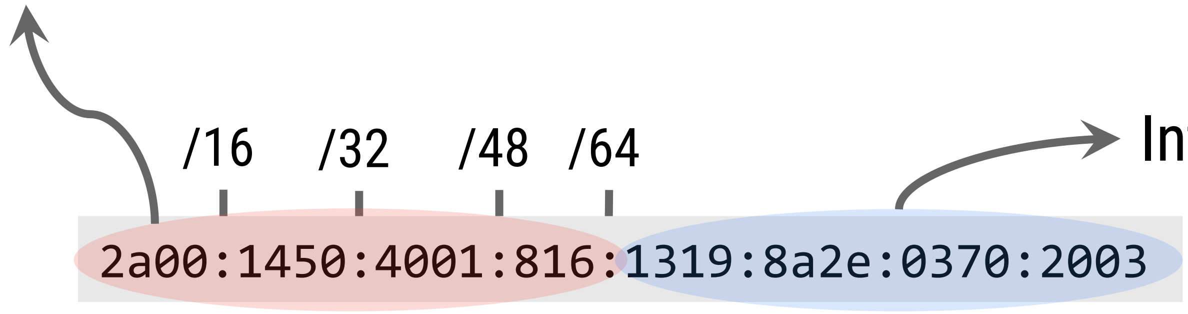
- Addressing via IP & port: `https://[201a::945:daa2:5eff:fe8e:e553]:443`

IPv6 Subnets

- Denoted in CIDR representation, e.g. `2001:0db8:1234::/48` includes `2001:0db8:1234:0:0:0:0:0` to `2001:0db8:1234:ffff:ffff:ffff:ffff:ffff`

Address Allocation

Prefix: `2a00:1450:4001:816::/64` with $2^{(128-64)} = 2^{64}$ addresses



Interface Identifier: `1319:8a2e:0370:2003`

- 64 bits reserved for interface ID
  → possibility for 2^{64} hosts in one LAN
  - Typically ISPs give /64 blocks to customers!

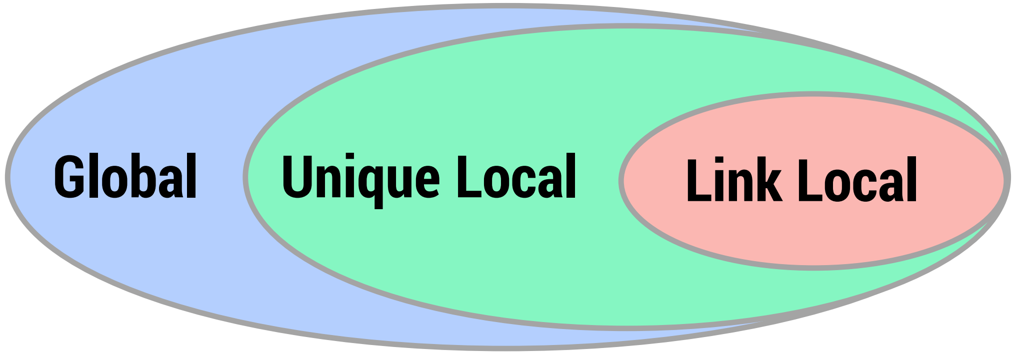
IPv6 Address Types

No more broadcast address!

- Unicast (*one-to-one*)
  - Address of single interface → Packet delivery for one receiver
  - Comparable with classic IPv4 address
- Multicast (*one-to-many*)
  - Address for a set of interfaces (typically owned by different hosts)
    → Packet for multicast address delivered to **all** interfaces with that address
- Anycast (*one-to-nearest*)
  - Same as multicast but
    → Packet for anycast address delivered to **one** interface with that address
  - The „closest" one according to the routing protocol's distance metric

IPv6 Address Scopes

In which part of a network (scope) is an address valid?



- Unicast / Anycast
  - **Link local:** Loopback, only valid on current link (network) → not routable
  - **Unique local:** Only for communication in small subnets → private addresses
  - **Global:** Globally valid, routed via Internet

- Multicast
  - Prefix `ff00::` identifies scope

```
ip -6 addr
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP>
   mtu 1500 qlen 1000
     inet6 2001:11d0:8:650b::3/64 scope global
     inet6 fe80::250:56ff:fe05:866c/64 scope link
```

Unicast Addresses

Link local addresses

- `::1/128`: Loopback address, same as 127.0.0.1 on IPv4
- `fe80::/10`: Only valid and unique on single link
  - Can be used for communication between two IPv6 devices (like MAC address but on layer 3)
  - Existence mandatory on every IPv6-enabled device!

```
ip -6 addr | grep fe80
inet6 fe80::250:56ff:fe05:866c/64 scope link
```

Unique local addresses

fc00::7	Global ID	Subnet ID	Interface ID
7 bits	40 bits	16 bits	

= 128 bits address

- Comparable to private address 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 in IPv4
- For local communications or inter-site VPNs. Not routable in Internet!

Unicast Addresses

Global addresses

- Addresses for generic use of IPv6
  - Like IPv4 addresses globally unique, public, and routable
- `2000::/3` (`2000`… to `3fff`) → Only 1/8 of total address space for now
  - RIRs typically get blocks from /12 to /23
  - ISP mostly get /32

See: https://goo.gl/LwXQDy

- Enables efficient route aggregation, e.g.
  - Customer 1 gets `2001:db8:1:/48`
  - Customer 2 gets `2001:db8:2:/48`

  → ISP only routes the /32 prefix `2001:b8::/32`

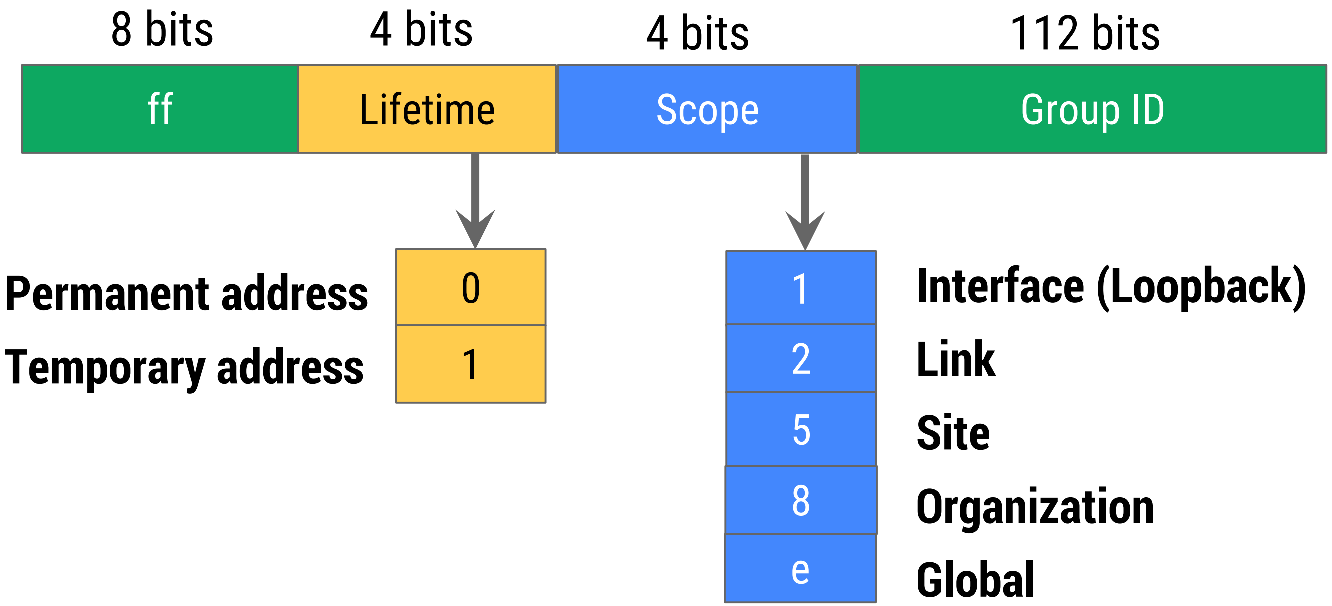
Somewhat overkill for LANs with 10 machines!
→ RFC 6177 suggests variably-sized subnets between /48 and /64, depending on needs…

Multicast Addresses

Sending out broadcasts…

- IPv6 has no broadcast addresses as in IPv4, e.g., *192.168.1.255*
- Use Multicast instead → prefix: `ff00`

Schema



Examples

- `ff02::1` = Send broadcast to all nodes in LAN segment
- `ff02::2` = All routers in LAN segment

See: https://goo.gl/RXkKDP

IPv6 Header

IPv6 Header

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="4">Version</td><td colspan="8">Traffic Class</td><td colspan="20">Flow Label</td></tr>
<tr><td>4</td><td>32</td><td colspan="16">Payload Length</td><td colspan="8">Next Header</td><td colspan="8">Hop Limit</td></tr>
<tr><td>8</td><td>64</td><td colspan="32" rowspan="4">Source Address</td></tr>
<tr><td>12</td><td>96</td></tr>
<tr><td>16</td><td>128</td></tr>
<tr><td>20</td><td>160</td></tr>
<tr><td>24</td><td>192</td><td colspan="32" rowspan="4">Destination Address</td></tr>
<tr><td>28</td><td>224</td></tr>
<tr><td>32</td><td>256</td></tr>
<tr><td>36</td><td>288</td></tr>
</table>

- Version (4 bits): IP protocol number → 6
- Traffic Class (8 bits): Like DSCP field in IPv4 used for traffic prioritization, e.g. low latency for streaming media

IPv6 Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	*Version*				*Traffic Class*								*Flow Label*																			
4	32	*Payload Length*																*Next Header*								*Hop Limit*							
8	64	*Source Address*																															
12	96																																
16	128																																
20	160																																
24	192	*Destination Address*																															
28	224																																
32	256																																
36	288																																

- Flow Label (20 bits): Identifies packet using labels, e.g., VoIP conversation
  Hint for routers to use same outgoing path for these packets to avoid re-ordering at receiver side → can be useful for real-time applications

- Payload Length (16 bits): Only size of sent data because
  IPv6 header size is fixed (40 bytes). In IPv4 → total header length

IPv6 Header

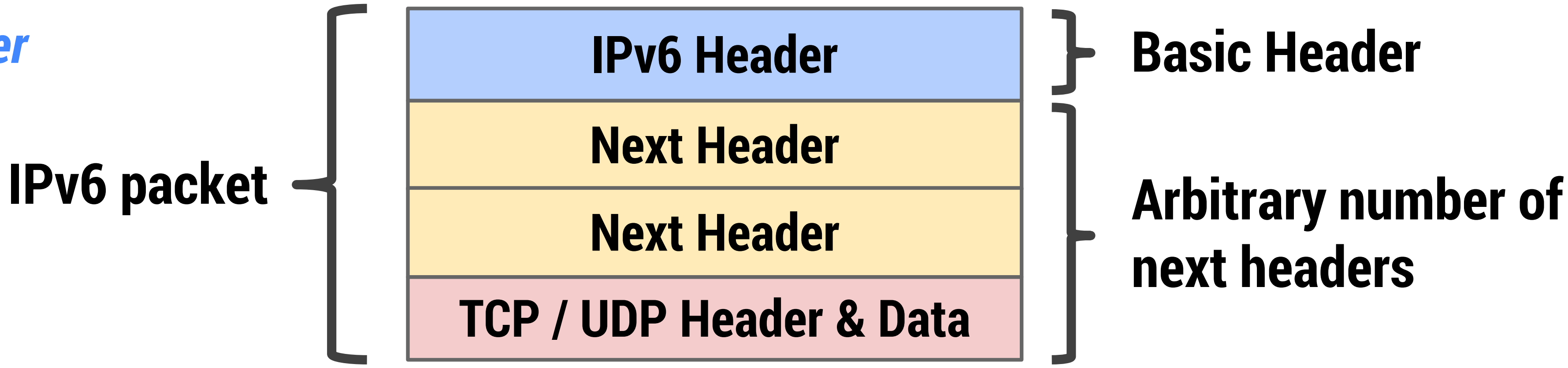
<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="4">Version</td><td colspan="8">Traffic Class</td><td colspan="20">Flow Label</td></tr>
<tr><td>4</td><td>32</td><td colspan="16">Payload Length</td><td colspan="8">Next Header</td><td colspan="8">Hop Limit</td></tr>
<tr><td>8</td><td>64</td><td colspan="32" rowspan="4">Source Address</td></tr>
<tr><td>12</td><td>96</td></tr>
<tr><td>16</td><td>128</td></tr>
<tr><td>20</td><td>160</td></tr>
<tr><td>24</td><td>192</td><td colspan="32" rowspan="4">Destination Address</td></tr>
<tr><td>28</td><td>224</td></tr>
<tr><td>32</td><td>256</td></tr>
<tr><td>36</td><td>288</td></tr>
</table>

- Next Header (8 bits): Next header following the IPv6 header
  Like protocol identifier in IPv4 header, e.g. TCP (6), UDP (17), ICMPv6 (58)
  or code of new IPv6 extension header

- Hop Limit (8 bits): Like TTL field in IPv4
  Decremented by one at each visited node. If 0 → packet discarded

IPv6 Extension Header

Carry __optional__ Internet Layer information



Extension Header	Description
Routing	Source specifies route
Fragment	Parameters for fragmentation (if still needed)
Authentication (AH)	Integrity of IPv6 packets (IPSec)
Encapsulation (ESP)	Encryption / decryption of IPv6 packets (IPSec)
Destination options	Examined only by destination device
Hop-by-hop options	Examined by all devices on path, e.g. Jumbograms = IPv6 packets > 65.535 bytes

- Placed between IPv6 Header and higher layer protocol
- Last **Next Header** in chain must be upper layer protocol
- Most IPv6 packets without extension headers

Difference to IPv4 Header?

Missing fields

- Fragmentation
  - Fields: Identification, Flags, Fragment Offset
  - Moving intelligence to clients → Routers **never** fragment IPv6 packets

- Header Length, Options
  - 40 bytes fixed header size instead of variable length
  - Some IPv4 Options moved to extension headers

- Header checksum
  - Processing overhead reduced → TCP, UDP should do this
  - IPv4 routers needed to re-calculate checksum **after** decreasing the TTL value

Difference to IPv4 Header?

Renamed fields

- IPv4 Protocol → IPv6 Next header
- IPv4 Total Length → IPv6 Payload Length
  - Remember: IPv6 header size always 40 bytes
- IPv4 TTL → IPv6 Hop Limit
  - Count number of hops instead of seconds

New fields

- Traffic Class
  - Packet Prioritization / Quality of Service (QoS), e.g., for VoIP or A/V streaming
- Flow Label
  - Distinguish flow of packets that need same treatment / routes

IPv6 Header

Wireshark Example

```
Internet Protocol Version 6, Src: 2a00:11c0:          , Dst: 2a02:8388:e301:c00:79ef:cd87:3ac2:79ad
    0110 .... = Version: 6
    .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 44
    Next header: TCP (6)
    Hop limit: 54
    Source: 2a00:11c0:
    Destination: 2a02:8388:e301:c00:79ef:cd87:3ac2:79ad
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
Transmission Control Protocol, Src Port: 5938 (5938), Dst Port: 55780 (55780), Seq: 25, Ack: 97, Len: 24
```

IPv6 Fragmentation

Important: In IPv6, routers never fragment packets!

- MTU Path discovery mandatory for IPv6 clients
- Alternative for IPv6: Use Minimum MTU size 1280 bytes
  - Any link must be able to transfer this size without end-to-end fragmentation
  - Also practical, e.g. for small systems → can omit MTU Path Discovery

MTU Path Discovery in IPv6

- As in IPv4, packets with size > MTU are dropped by router
  → ICMP error message: „Packet too big“

- Actual fragmentation task of upper-layer protocol (TCP, UDP)

IPv6 Functionality

ICMPv6

= Replaces ICMPv4, IGMP, ARP (!)

RFC 4443

- As in IPv4, used to send error and information (e.g., „ping“) messages
- ICMPv4 was often blocked by firewalls → ICMPv6 **must** be allowed!
  - No fragmentation in IPv6 → MTU Path discovery necessary

<table>
<tr><th></th><th>Bits 0–3</th><th>Bits 4–11</th><th>Bits 12–15</th><th>Bits 16–23</th><th>Bits 24–31</th></tr>
<tr><td rowspan="4">IPv6 Header
(40 bytes)</td><td>Version</td><td>Traffic Class</td><td colspan="3">Flow Label</td></tr>
<tr><td colspan="3">Payload Length</td><td>Next Header</td><td>Hop Limit</td></tr>
<tr><td colspan="5">Source IP address</td></tr>
<tr><td colspan="5">Destination IP address</td></tr>
<tr><td>ICMPv6 Header
(8 bytes)</td><td colspan="2">Type of message</td><td colspan="1">Code</td><td colspan="2">Checksum</td></tr>
<tr><td>ICMPv6 Payload</td><td colspan="5">Payload Data</td></tr>
</table>

For ICMPv6:
Next Header = 58

ICMPv6 Codes

Mostly used…

Type	Code	Description
1 – Destination Unreachable	3	Address unreachable
	4	Port unreachable
2 – Packet too big	0	→ Points to actual MTU to be used
3 – Time Exceeded	0	Hop limit exceeded in transit
128 – Echo Request	0	Client requests IPv6 addresses (ping)
129 – Echo Reply	0	Answer (ping)
133 – Router Solicitation	0	Request for router advertisements
134 – Router Advertisement	0	Router sends Internet parameters
135 – Neighbor Solicitation	0	Get MAC address of neighbor node
136 – Neighbor Advertisement	0	Node sends his (new) MAC address
137 - Redirect	0	Inform hosts of better first hop

Error messages (types 1–3)

Information messages (types 128–137)

For more codes, see http://goo.gl/ONIm9d

ICMPv6 Ping

```
ping google.at
Reply by 2a00:1450:4001:812::2003: time=21ms
```

Echo Request

```
> Internet Protocol Version 6, Src: 2a02:8388:e301:                        , Dst: 2a00:1450:4001:812::2003
v Internet Control Message Protocol v6
    Type: Echo (ping) request (128)
    Code: 0
    Checksum: 0xcb3d [correct]
    Identifier: 0x0001
    Sequence: 10
    [Response In: 23]
  v Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f7071727374757677 61...
      [Length: 32]

0000  80 c6 ab 73 f5 64 c8 60  00 c9 e2 77 86 dd 60 00   ...s.d.`
0010  00 00 00 28 3a 80 2a 02  83 88 e3 01 0c 00 b8 7f   ...(:.*.
0020  7f 99 11 27 7d 7d 2a 00  14 50 40 01 08 12 00 00   ...'}}*.
0030  00 00 00 00 20 03 80 00  cb 3d 00 01 00 0a 61 62   .... ...
0040  63 64 65 66 67 68 69 6a  6b 6c 6d 6e 6f 70 71 72   cdefghij
0050  73 74 75 76 77 61 62 63  64 65 66 67 68 69         stuvwabc
```

Echo Reply

```
> Internet Protocol Version 6, Src: 2a00:1450:4001:812::2003, Dst: 2a02:8388:e301:c
v Internet Control Message Protocol v6
    Type: Echo (ping) reply (129)
    Code: 0
    Checksum: 0xca3d [correct]
    Identifier: 0x0001
    Sequence: 10
    [Response To: 22]
    [Response Time: 21.867 ms]
  v Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f7071727374757677 61...
      [Length: 32]

0000  c8 60 00 c9 e2 77 80 c6  ab 73 f5 64 86 dd 60 00   .`...w.. .s.d..`.
0010  00 00 00 28 3a 37 2a 00  14 50 40 01 08 12 00 00   ...(:7*. .P@.....
0020  00 00 00 00 20 03 2a 02  83 88 e3 01 0c 00 b8 7f   .... .*. ........
0030  7f 99 11 27 7d 7d 81 00  ca 3d 00 01 00 0a 61 62   ...'}}.. .=....ab
0040  63 64 65 66 67 68 69 6a  6b 6c 6d 6e 6f 70 71 72   cdefghij klmnopqr
0050  73 74 75 76 77 61 62 63  64 65 66 67 68 69         stuvwabc defghi
```

Neighbor Discovery Protocol

RFC 4861

= NDP

- Operates on link layer but part of ICMPv6
  - 5 ICMPv6 packet types for information exchange
    - 133 – Router Solicitation, 134 – Router Advertisement
    - 135 – Neighbor Solicitation
    - 136 – Neighbor Advertisement
    - 137 – Redirect

Purpose

- Replaces functionality of ARP
- Hosts use it to discover routers, check neighbors
- Auto-configuration of addresses

Internet protocol suite

Application layer

BGP · DHCP · DNS · FTP · HTTP · IMAP · LDAP · MGCP · NNTP · NTP · POP · ONC/RPC · RTP · RTSP · RIP · SIP · SMTP · SNMP · SSH · Telnet · TLS/SSL · XMPP · *more...*

Transport layer

TCP · UDP · DCCP · SCTP · RSVP · *more...*

Internet layer

IP (IPv4 · IPv6) · ICMP · ICMPv6 · ECN · IGMP · IPsec · *more...*

Link layer

ARP · **NDP** · OSPF · Tunnels (L2TP) · PPP · MAC (Ethernet · DSL · ISDN · FDDI) · *more...*

V · T · E

NDP Tasks

RFC 4861

Discovery

- Router: Available routers in network?
- Prefix: Set of prefixes for current link?
- Parameter: Which MTU or hop limit to put on outgoing packets?

Addresses

- Resolution: Replacement for ARP
- Auto-configuration (SLAAC): Assign IPv6 address to interface **without** DHCP
- Detection of duplicates (DAD) and neighbor unreachability (NUD)

Routing

- Next-hop determination: Link-layer address for next hop (default gateway)
- Redirection: Routers tell hosts there is a better first-hop available

NDP – Router Discovery

In IPv4

Client has static routes or learned via DHCP

In IPv6

- Host joins network
- Sends out: „Router Solicitation“ via Multicast to group „all routers“
  - ICMPv6 message type 133
- Router advertises / sends periodically to Multicast group „all nodes“
  - ICMPv6 message type 134
  - One or more prefixes, lifetime of prefixes, router information

Security issue: Everybody can send out RA and pretend to be a router

MITM attack!

NDP – Router Discovery

Wireshark Example

```
15…  712.341…  fe80::82c6:abff:f…  ff02::1            ICMPv6     174 Router Advertisement from 80:c6:ab:

> Frame 11667: 174 bytes on wire (1392 bits), 174 bytes captured (1392 bits) on interface 0
> Ethernet II, Src: Technico_            (80:c6:ab:            ), Dst: IPv6mcast_01 (33:33:00:00:00:01)
> Internet Protocol Version 6, Src: fe80::82c6:abff:fe73:f564, Dst: ff02::1
v Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0x02b0 [correct]
    Cur hop limit: 64
  > Flags: 0xc0
    Router lifetime (s): 1800
    Reachable time (ms): 0
    Retrans timer (ms): 0
  > ICMPv6 Option (Source link-layer address : 80:c6:ab:
  > ICMPv6 Option (Prefix information : 2a02:8388:e301:c00::/64)
  > ICMPv6 Option (Route Information : Medium 2a02:8388:e301:c00::/57)
  > ICMPv6 Option (Recursive DNS Server 2001:730:3e62::53 2001:730:3e62:1000::53)
```

NDP – Router Discovery

Wireshark Example

```
15…  712.341…  fe80::82c6:abff:f…  ff02::1            ICMPv6      174 Router Advertisement from 80:c6:ab:
```

```
⌄ ICMPv6 Option (Prefix information : 2a02:8388:e301:c00::/64)
    Type: Prefix information (3)
    Length: 4 (32 bytes)
    Prefix Length: 64
  ⌄ Flag: 0xc0
      1... .... = On-link flag(L): Set
      .1.. .... = Autonomous address-configuration flag(A): Set
      ..0. .... = Router address flag(R): Not set
      ...0 0000 = Reserved: 0
    Valid Lifetime: 1209600
    Preferred Lifetime: 604800
    Reserved
    Prefix: 2a02:8388:e301:c00::
⌄ ICMPv6 Option (Route Information : Medium 2a02:8388:e301:c00::/57)
    Type: Route Information (24)
    Length: 3 (24 bytes)
    Prefix Length: 57
  › Flag: 0x00
    Route Lifetime: 1209600
    Prefix: 2a02:8388:e301:c00::
› ICMPv6 Option (Recursive DNS Server 2001:730:3e62::53 2001:730:3e62:1000::53)
```

NDP – Address Resolution

Replaces functionality of ARP in IPv4

- Instead of ARP table → „Neighborhood cache“

How to get the MAC address from IPv6 address?

1. Send out: „Neighbor solicitation“ to *solicited-node* multicast address
   - Request not broadcasted to *all* nodes (as with ARP in IPv4)!
   - **Only** nodes that joined this multicast group will see the request!
   - ICMPv6 message type 135
2. If node is present: „Neighbor advertisement“
   - ICMPv6 message type 136

→ Neighborhood Cache is updated
with mapping IP → MAC address

See: https://goo.gl/WnkCgk

NDP – Address Resolution

Neighborhood Cache

```
ip -6 neigh
2001:41d1:8:650b::14 dev eth0 lladdr 00:50:56:08:b9:ba REACHABLE
fe80::250:56ff:fe00:aefc dev eth0 lladdr 00:50:56:00:ae:fc STALE
2001:41d1:8:65ff:ff:ff:ff:ff dev eth0 lladdr 00:05:73:a0:00:00 router REACHABLE
```

Windows:

```
netsh interface ipv6 show neighbors
2001:41d1:8:65ff:ff:ff:ff:ff                00-00-00-00-00-00   Not reachable
fe80::82c6:abff:fe73:f564                   80-c6-ab-..-..-..   Reachable (Router)
ff02::1                                     33-33-00-00-00-01   Permanent
ff02::2                                     33-33-00-00-00-02   Permanent
ff02::c                                     33-33-00-00-00-0c   Permanent
…
```

NDP – Auto-configuration

Stateless address auto-configuration (SLAAC)

- Interface can obtain IPv6 address without router / server (*stateless*)
- Enables plug-and-play operation of host

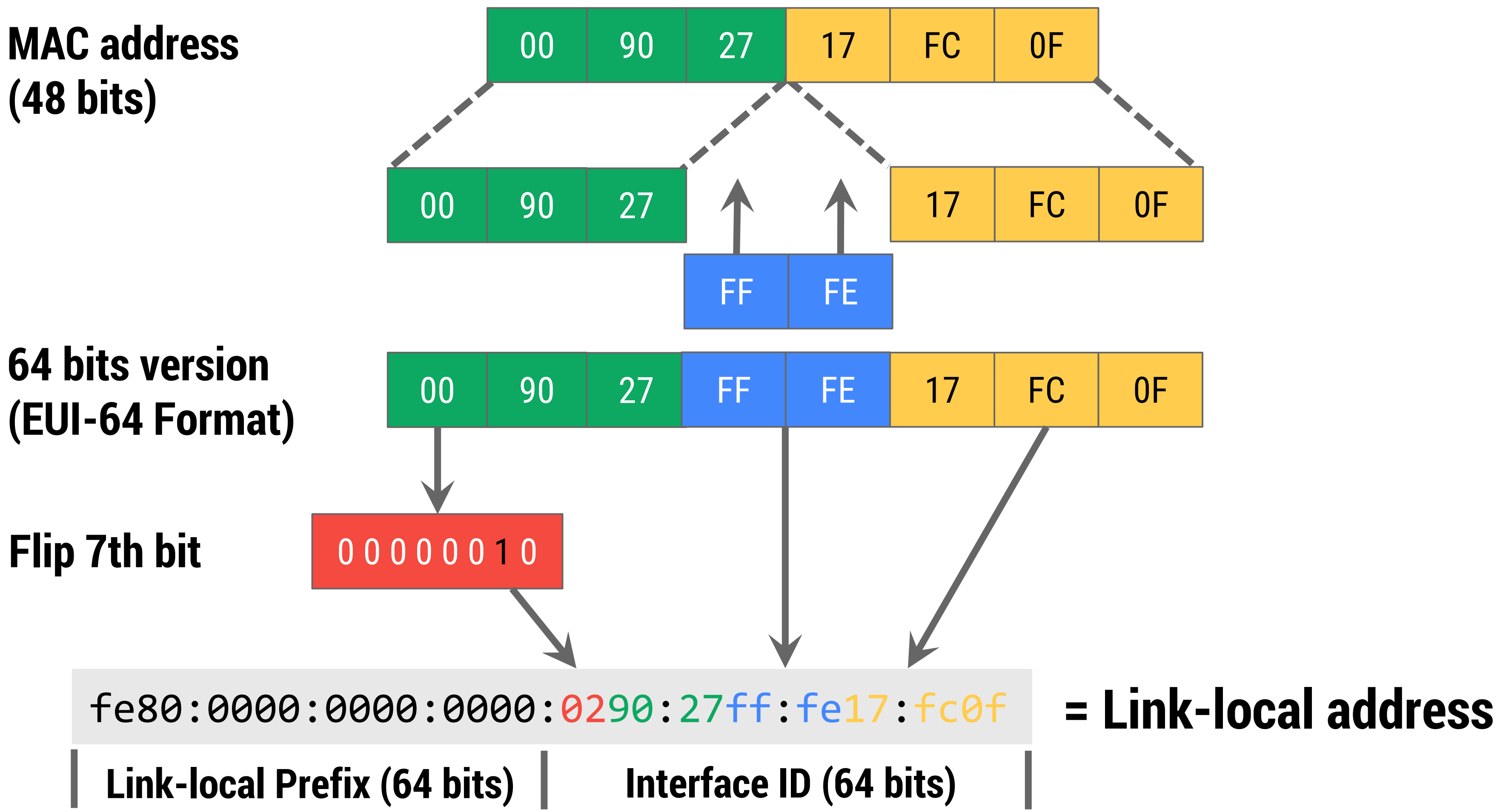
Workflow

1. Multicast-capable interface comes up
2. Derive IPv6 *link-local* address from link layer address (MAC)
3. Check for potentially duplicate addresses (prevent collision)
4. Perform router / prefix discovery in order to get address with *global* or *unique-local* scope

NDP – Auto-configuration

How to derive a link-local address from the MAC address?

See: https://goo.gl/BwAih8



EUI = Extended Unique Identifier

NDP – Auto-configuration

Is this address already in use?
= Duplicate Address Detection (DAD)

1. Send „Neighbor Solicitation“ to local network
2. „Neighbor Advertisement“ is sent (only) if other host uses this address

→ In practice, collision very unlikely due to large address space

Get global IPv6 address

- Wait for router advertisement with prefix or
- Send „Router Solicitation“ to multicast address `ff02::2` (all routers)
- Reply: „Router Advertisement“
  with global prefixes

`2a00:1450:dead:beef:0290:27ff:fe17:fc0f`

Global Prefix (64 bits)	**Interface ID (64 bits)**

NDP – Auto-configuration

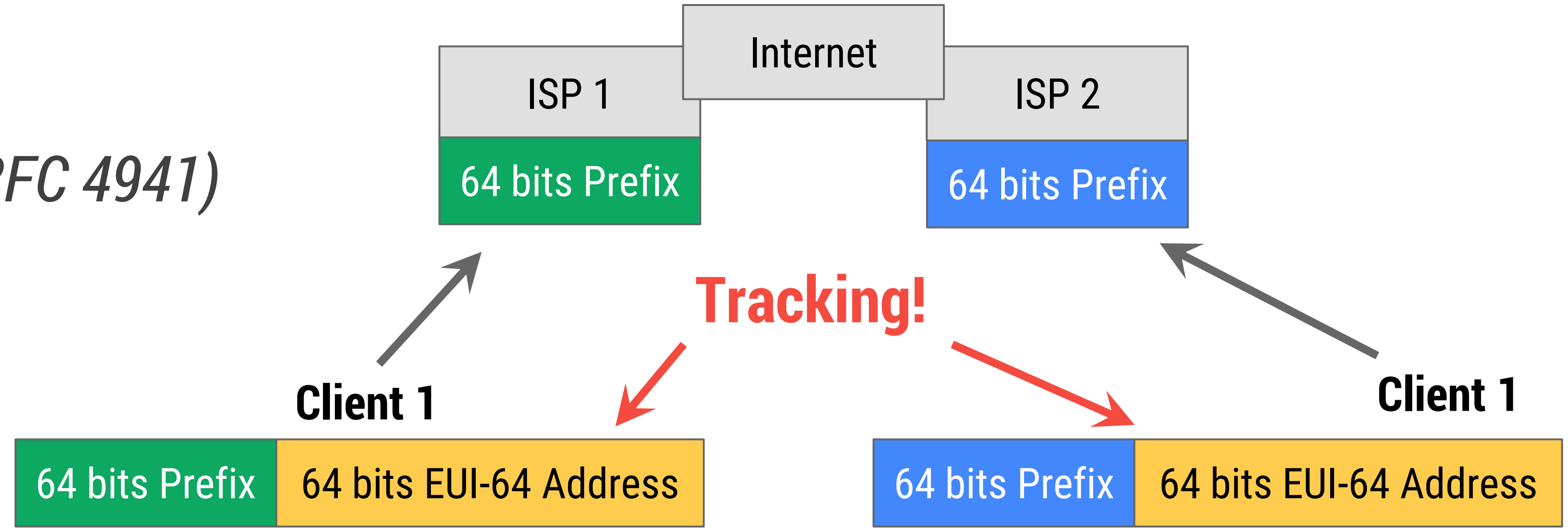
Privacy?

- Unique addresses
  - With IPv6, every interface gets a unique IPv6 address
  - By design, your interface MAC address is globally unique
- No dynamic IP addresses / NAT needed → huge address space

If notebook location changes → new prefix but still same interface ID!

Remedy:

„Privacy Extensions for SLAAC“ (RFC 4941)
→ Temporary IPv6 addresses



Transport Layer

Transport Layer

Purpose

- Data channels for individual applications
  - Transport end-to-end messages between particular services
  - Use multiplexing to differentiate multiple, separate applications

- Reliability
  - „Best-effort“ delivery is not good enough
  - Re-order incoming packets according to their sending order
  - Detect errors → request faulty packets again

- Flow Control & Congestion Avoidance
  - Sender must not overwhelm receiver with packets
  - Handle too much traffic in the network

Transport Layer

Protocols

- TCP: Transmission Control Protocol
  - Connection-oriented
  - Reliable but „heavyweight“ end-to-end transport of data
  - Error detection, Flow & congestion control, Ordered Delivery

  → Applications: HTTP(S), FTP, SSH, SMTP, IMAP, POP3, …

- UDP: User Datagram Protocol
  - Connection-less
  - Unreliable → sender does not know if destination reached
  - No congestion control

  → Applications: DNS, DHCP, SNMP, often also in VPNs

Internet Protocol Suite

Application Layer

BGP · DHCP · DNS · FTP · HTTP · IMAP · IRC · LDAP · MGCP · NNTP · NTP · POP · RIP · RPC · RTP · SIP · SMTP · SNMP · SSH · Telnet · TLS/SSL · XMPP ·

(more)

Transport Layer

TCP · UDP · DCCP · SCTP · RSVP · ECN ·

(more)

Internet Layer

IP (IPv4, IPv6) · ICMP · ICMPv6 · IGMP · IPsec ·

(more)

Link Layer

ARP/InARP · NDP · OSPF · Tunnels (L2TP) · PPP · Media Access Control (Ethernet, DSL, ISDN, FDDI) · (more)

v·d·e

Transport Layer – Ports

In UDP and TCP…

- Service is provided to higher layers through ports
  - Same port number for different TCP and UDP service is possible

- Used for multiplexing
  - Ports allow to speak to different applications running on same host
  - Addressing via IP address and port number, e.g.

```
192.168.1.1:80 or [201a::945:daa2:5eff:fe8e:e553]:443
```

- Session: Communication between client and server on a socket pair
  - TCP: Established after fulfilling a handshake
  - UDP: Identified on higher level, e.g. using session cookies

Transport Layer – Ports

16-bit numbers between 0 - 65535

Three categories

- Well-known ports: 0 – 1023
  - Reserved by convention for specific, widely-used services
  - On Linux: Can be opened only by superuser (`root`)

- Registered ports: 1024 – 49151
  - Proprietary applications

- Dynamic ports: 49152 – 65535
  - Ephemeral or short-lived ports
  - Dynamically opened / closed by applications during sessions

Transport Layer – Ports

Mostly used…

Port Number	Service
20	File Transfer Protocol (FTP) – Data
21	FTP - Control channel
22	Secure Shell (SSH)
23	telnet
25	Simple Mail Transfer Protocol (SMTP)
53	Domain Name System (DNS)
80	Hypertext Transfer Protocol (HTTP)
443	HTTP Secure (HTTPS)
1194	OpenVPN
3306	MySQL Database System
5060	VoIP Signalling (SIP)

Well-known Ports (20–443)

Registered Ports (1194–5060)

For more ports, see http://goo.gl/Ds3BTj

Transport Layer – Ports

Example Scenario

Client (129.27.142.14) wants to connect to HTTP Server at 129.27.142.13

Workflow

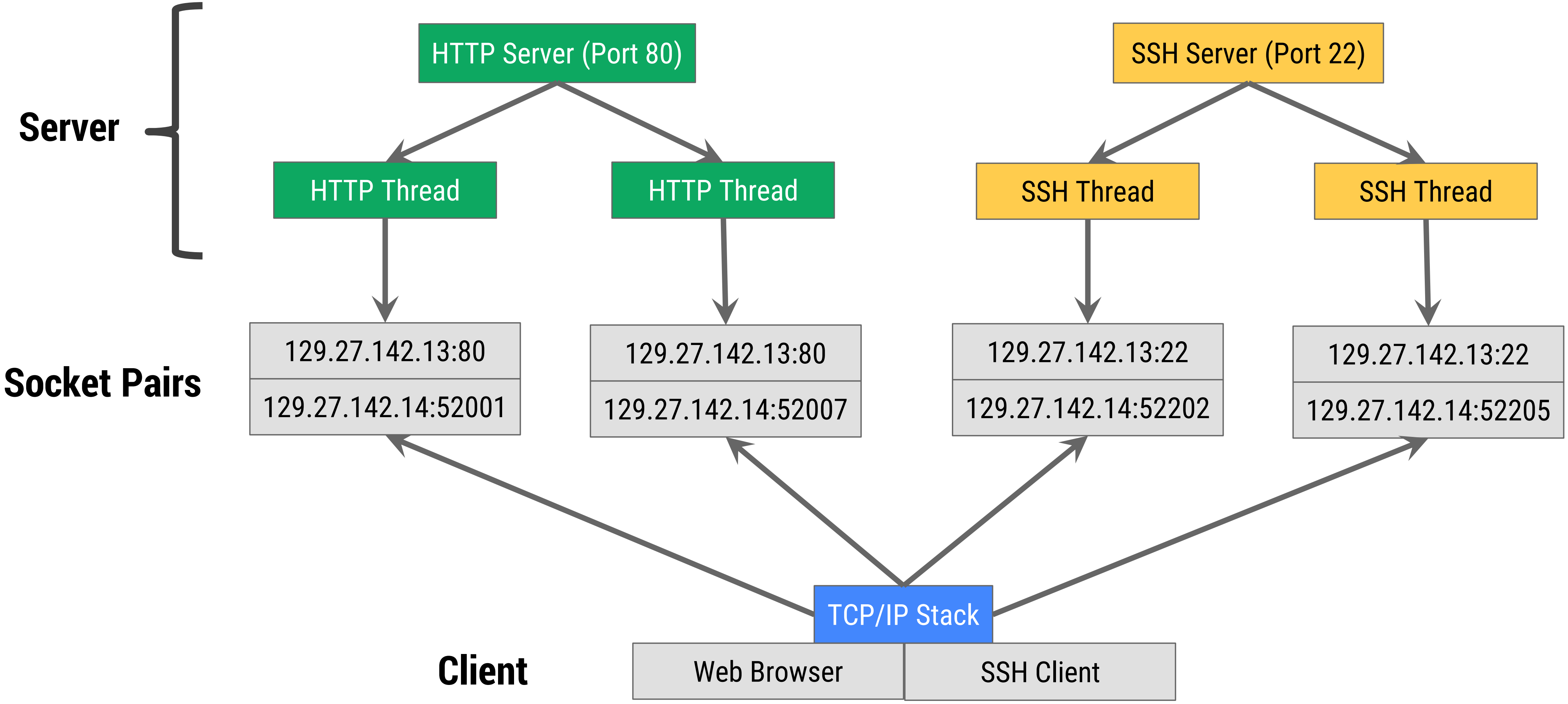
1. Client chooses source port > 49151
2. Destination port known / fixed at 80 (HTTP)

```
129.27.142.14:52312 -> 129.27.142.13:80
```

Note:

Another connection from 129.27.142.14
would use another source port!

Transport Layer – Ports



Transport Layer – Ports

```
netstat -an
Proto  Recv-Q Send-Q Local Address         Foreign Address       State
tcp    0      0      129.27.142.13:22      0.0.0.0:*             LISTEN
tcp    0      0      129.27.142.13:80      0.0.0.0:*             LISTEN
tcp    0      0      129.27.142.13:80      129.27.142.14:52001   ESTABLISHED
tcp    0      0      129.27.142.13:80      129.27.142.14:52007   ESTABLISHED
tcp    0      0      129.27.142.13:22      129.27.142.14:52202   ESTABLISHED
tcp    0      0      129.27.142.13:22      129.27.142.14:52205   ESTABLISHED
```

Server

```
netstat -an
Proto  Recv-Q Send-Q Local Address         Foreign Address       State
tcp    0      0      129.27.142.14:52001   129.27.142.13:80      ESTABLISHED
tcp    0      0      129.27.142.14:52007   129.27.142.13:80      ESTABLISHED
tcp    0      0      129.27.142.14:52202   129.27.142.13:22      ESTABLISHED
tcp    0      0      129.27.142.14:52205   129.27.142.13:22      ESTABLISHED
```

Client

Transport Layer – Firewalls

Firewalls typically use rules to decide on connections:
→ `Allow / Drop | Source IP + Port | Destination IP + Port`

Example: How to block all SSH connections on 129.27.142.13?

```
iptables -I INPUT -d 129.27.142.13 --dport 22 -j DROP
```

= Drop incoming connections to 129.27.142.13:22

- Firewalls nowadays do stateful inspection
  - Understand protocols and connection state (build-up, usage, teardown)
- Often also inspect higher-level protocols
  - E.g. helps to detect brute-force attempts

UDP

UDP

RFC 768

= User Datagram Protocol

Attributes

- Stateless: Great for large number of clients (streaming)
- Transaction-oriented (= „connection-less“)
- Unreliable
  - Packets could be dropped, corrupted, out-of-order, … but UDP does **not** detect that!

Usage where…

- Re-transmission of lost packets makes no sense, e.g. VoIP, Streaming, …
- Small implementations are needed (SNMP, TFTP, DHCP)
- Simple request / response is enough (DNS, NTP)

Internet protocol suite

Application layer
BGP · DHCP · DNS · FTP · HTTP · IMAP · LDAP · MGCP · NNTP · NTP · POP · ONC/RPC · RTP · RTSP · RIP · SIP · SMTP · SNMP · SSH · Telnet · TLS/SSL · XMPP · *more...*

Transport layer
TCP · **UDP** · DCCP · SCTP · RSVP · *more...*

Internet layer
IP (IPv4 · IPv6) · ICMP · ICMPv6 · ECN · IGMP · IPsec · *more...*

Link layer
ARP · NDP · OSPF · Tunnels (L2TP) · PPP · MAC (Ethernet · DSL · ISDN · FDDI) · *more...*

V · T · E

UDP Header

Bit	+0..7	+8..15	+16..23	+24..31
0	Source Port		Destination Port	
32	Length		Checksum	
...	Payload			

- Length (16 bits): Total UDP Packet length
  - Header + Payload
  - Payload in IPv4 limited to 65.507 bytes (65.535 - 8 (UDP header) - 20 (IPv4 header) bytes)
- Checksum (16 bits): Checksum over header and data
  - Optional in IPv4, mandatory in IPv6
- Payload: Data to be sent, e.g. DNS request

UDP Header

Bit	+0..7	+8..15	+16..23	+24..31
0	Source Port		Destination Port	
32	Length		Checksum	
...	Payload			

- Source Port (16 bits): 0 – 65535
  - Identifies application of sender (client)

- Destination Port (16 bits): 0 - 65535
  - Identifies application of receiver (server)

TCP

TCP

RFC 793

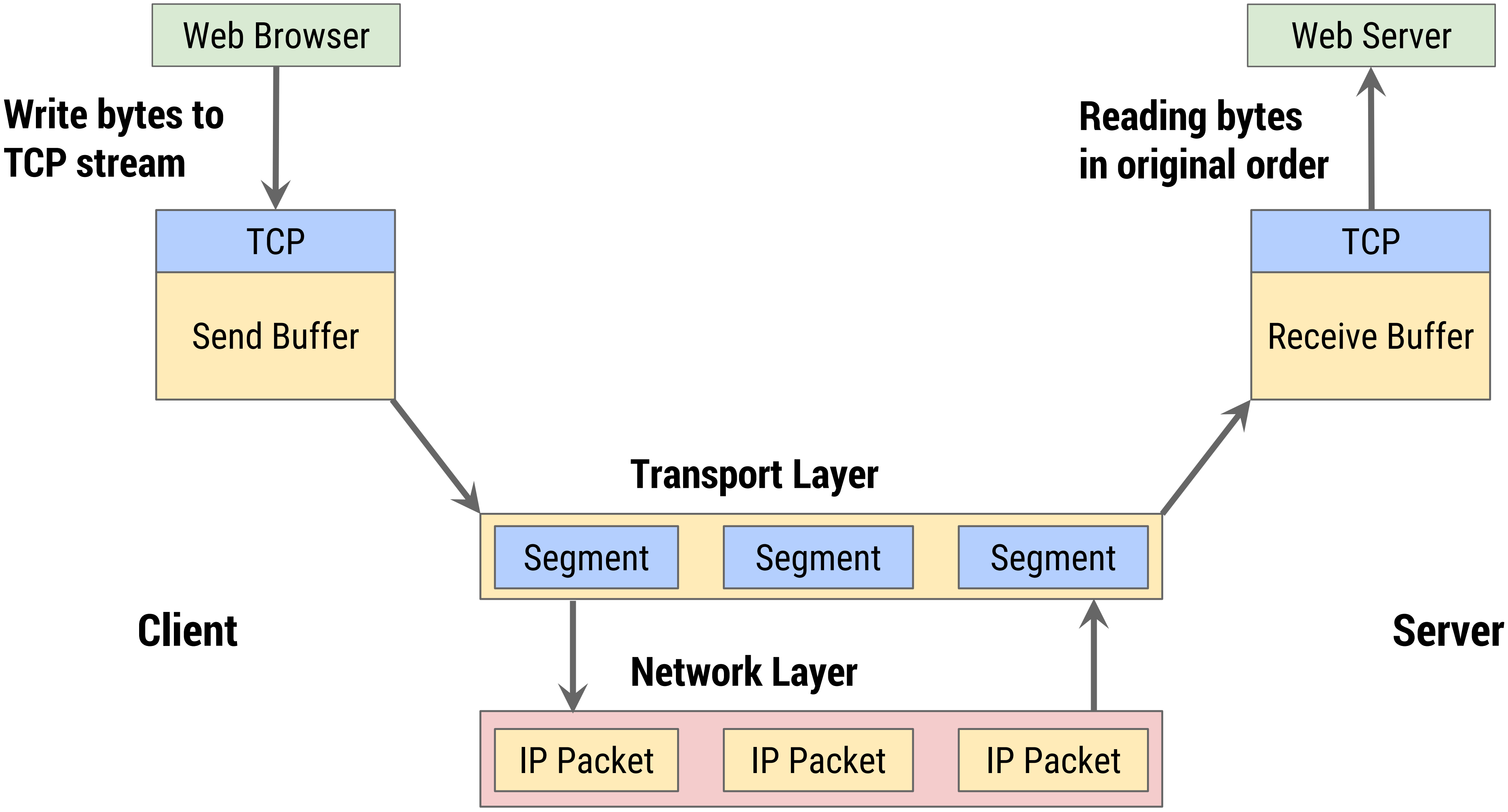
= Transmission Control Protocol

Attributes

- Connection-oriented
  - Data delivery only possible after „Three way handshake“
- Stateful
  - Check if destination is alive → Connection establishment
  - Have packets been lost? → Acknowledgement
  - Did packets arrive in correct-order? → Sequence
  - Can receiver follow speed of sender? → Flow Control (Buffers!)

- Application data is regarded as byte stream
  - TCP ensure reliable transmission of byte segments

TCP Transmission



TCP Header

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="16">Source port</td><td colspan="16">Destination port</td></tr>
<tr><td>4</td><td>32</td><td colspan="32">Sequence number</td></tr>
<tr><td>8</td><td>64</td><td colspan="32">Acknowledgment number (if ACK set)</td></tr>
<tr><td>12</td><td>96</td><td colspan="4">Data offset</td><td colspan="3">Reserved
0 0 0</td><td>N
S</td><td>C
W
R</td><td>E
C
E</td><td>U
R
G</td><td>A
C
K</td><td>P
S
H</td><td>R
S
T</td><td>S
Y
N</td><td>F
I
N</td><td colspan="16">Window Size</td></tr>
<tr><td>16</td><td>128</td><td colspan="16">Checksum</td><td colspan="16">Urgent pointer (if URG set)</td></tr>
<tr><td>20
...</td><td>160
...</td><td colspan="32">Options (if data offset > 5. Padded at the end with "0" bytes if necessary.)
...</td></tr>
</table>

- <u>Source & Destination Port (16 bits each):</u> As in UDP
  Note: An interface can listen on a TCP and UDP port simultaneously!

- <u>Sequence & Acknowledgement Number (32 bits each):</u>
  Take over various roles in concept of TCP – sessions, error handling, order, …

TCP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

- <u>Data Offset (4 bits):</u> = Header Length
  - Necessary because of variably-sized options
- <u>Reserved (3 bits):</u> Always 0

TCP Header

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="16">Source port</td><td colspan="16">Destination port</td></tr>
<tr><td>4</td><td>32</td><td colspan="32">Sequence number</td></tr>
<tr><td>8</td><td>64</td><td colspan="32">Acknowledgment number (if ACK set)</td></tr>
<tr><td>12</td><td>96</td><td colspan="4">Data offset</td><td colspan="3">Reserved
0 0 0</td><td>NS</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="16">Window Size</td></tr>
<tr><td>16</td><td>128</td><td colspan="16">Checksum</td><td colspan="16">Urgent pointer (if URG set)</td></tr>
<tr><td>20
...</td><td>160
...</td><td colspan="32">Options (if data offset > 5. Padded at the end with "0" bytes if necessary.)
...</td></tr>
</table>

- Flags (9 bits): = Control bits
  - Needed for connection establishment, tear-down, error-handling, etc.

- Window Size (16 bits): Flow Control Mechanism
  - Indicates how many bytes the sender is allowed to send without overloading the receiver

TCP Header Flags

Flags	Description
NS, ECE	Explicit Congestion Notification
SYN	Indicates connection request
	Sequence number synchronization
ACK	Used to acknowledge the receipt of data
	Always set, except in very first segment
FIN	Indicates no more data from sender
	Current segment is the last
RST	Immediately kill the connection
PSH	TCP should push the segment immediately to the application (no buffering)
URG	Used to mark urgend data with urgent pointer

TCP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

- Checksum (16 bits): As in UDP
  - Includes header and data
  - Usage mandatory with TCP – and very important for IPv6 (no checksum in header)

- Urgent pointer (16 bits): Indicates „urgent“ data

TCP Header

Wireshark Example

```
⌄ Transmission Control Protocol, Src Port: 22 (22), Dst Port: 51716 (51716), Seq: 1, Ack: 49, Len: 0
    Source Port: 22
    Destination Port: 51716
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 1    (relative sequence number)
    Acknowledgment number: 49    (relative ack number)
    Header Length: 20 bytes
  ⌄ Flags: 0x010 (ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: *******A****]
    Window size value: 322
    [Calculated window size: 322]
    [Window size scaling factor: -1 (unknown)]
  › Checksum: 0xa026 [validation disabled]
    Urgent pointer: 0
```

TCP States

TCP is connection-oriented which enables a reliable transmission

3 Phases

1. Connection Establishment – „Build-up“
   - Performed before data can be sent
2. Data Transmission
3. Connection Termination – „Tear-down“
   - Indicates to both sides that no more data is going to be sent

In between these phases, TCP undergoes a series of **state** changes, e.g.

CLOSED, SYN_SENT, SYN_RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE_WAIT, CLOSING, LAST_ACK, TIME_WAIT, CLOSED

TCP Terms

Sequence Numbers (SEQ)

- If **SYN** flag set: Initial sequence number, should be random-generated
- Without **SYN** flag: Position number for first data byte of this segment

Acknowledgement Numbers (ACK)

- If **ACK** flag set: Next sequence number (SEQ) the receiver is expecting
- Acknowledges receipt of all prior bytes up to ACK number - 1

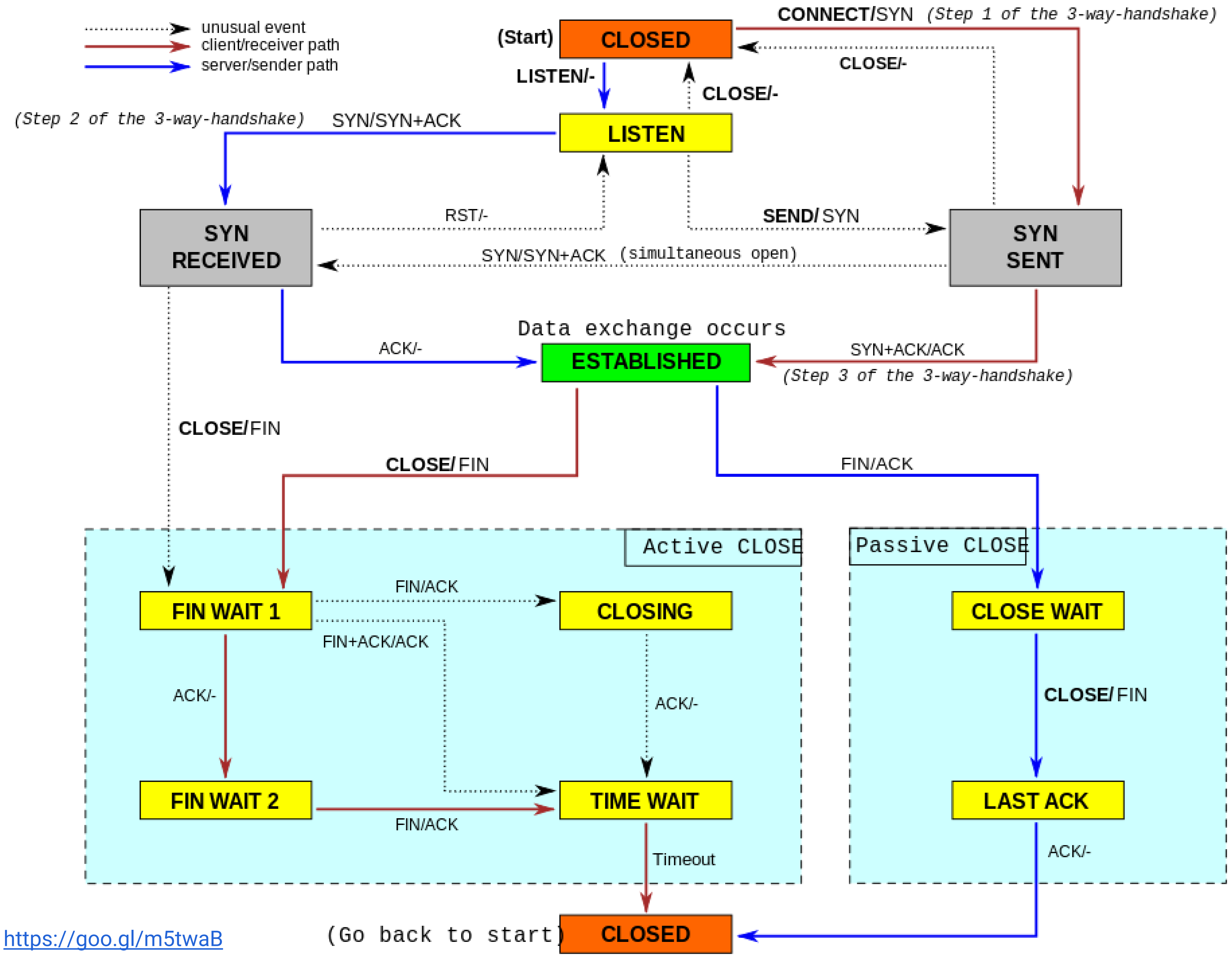
Maximum Segment Size (MSS)

Max. size of TCP payload, chosen in a way to avoid IP packet fragmentation

Maximum Segment Life (MSL)

120sec max. time a TCP segment can be in transit

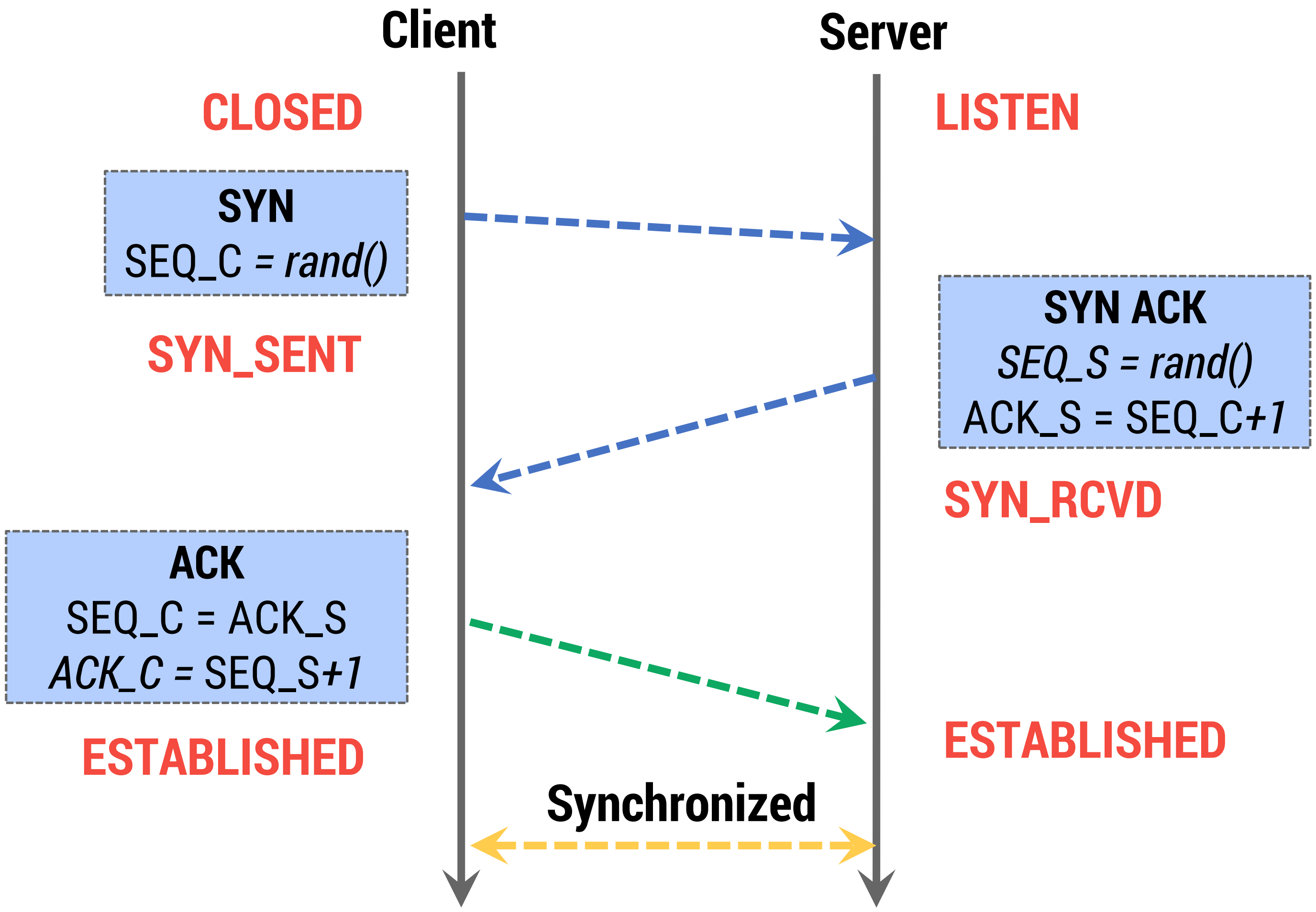
TCP State Diagram



Source: https://goo.gl/m5twaB

Three-Way Handshake

Before a sender transmits data, a connection needs to be built-up…



Workflow

1. Client: Send **SYN** packet
   Pick random sequence number
2. Server: Send **SYN, ACK** packet
   SEQ = Pick own random number
   ACK = Client SEQ + 1
3. Client: Send **ACK** packet
   SEQ = Received ACK number
   ACK = Server SEQ + 1

Wireshark Example

No.	Time	Source	Destination	Protocol	Length	Info
7	0.063488	192.168.0.13	129.27.2.210	TCP	66	60931 → 443 [SYN] Seq=1638621052 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
10	0.086221	129.27.2.210	192.168.0.13	TCP	62	443 → 60931 [SYN, ACK] Seq=2622560818 Ack=1638621053 Win=4260 Len=0 MSS=1420 SACK_PERM=1
11	0.086331	192.168.0.13	129.27.2.210	TCP	54	60931 → 443 [ACK] Seq=1638621053 Ack=2622560819 Win=65320 Len=0

1. Client → Server: SYN

```
> Internet Protocol Version 4, Src: 192.168.0.13, Dst: 129.27.2.210
v Transmission Control Protocol, Src Port: 60931 (60931), Dst Port: 443 (443), Seq: 1638621052, Len: 0
    Source Port: 60931
    Destination Port: 443
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 1638621052
    Acknowledgment number: 0
    Header Length: 32 bytes
  > Flags: 0x002 (SYN)
    Window size value: 8192
    [Calculated window size: 8192]
  > Checksum: 0x5987 [validation disabled]
    Urgent pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window s
```

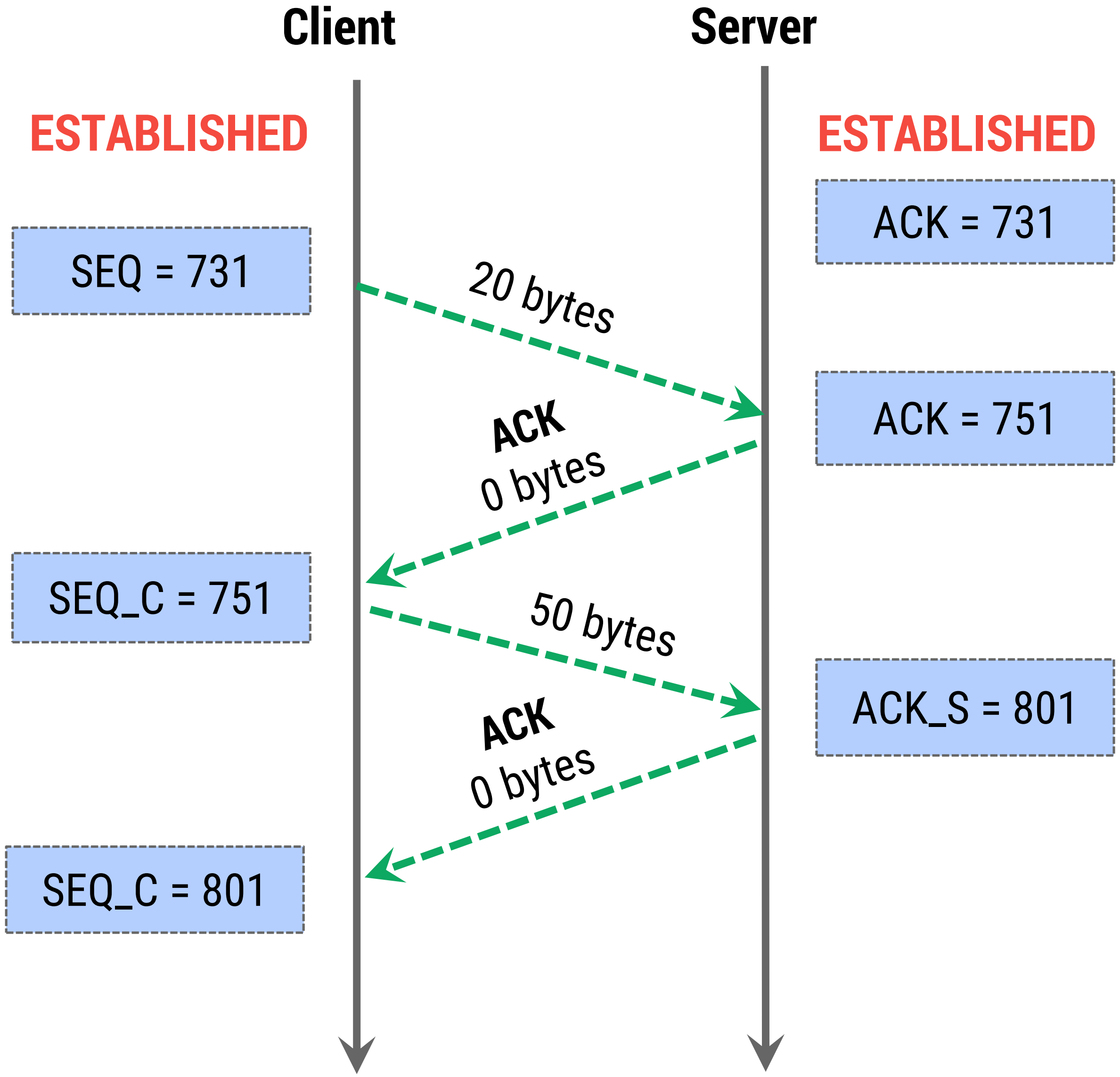
2. Server → Client: SYN ACK

```
> Internet Protocol Version 4, Src: 129.27.2.210, Dst: 192.168.0.13
v Transmission Control Protocol, Src Port: 443 (443), Dst Port: 60931 (60931),
    Source Port: 443
    Destination Port: 60931
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 2622560818
    Acknowledgment number: 1638621053
    Header Length: 28 bytes
  > Flags: 0x012 (SYN, ACK)
```

3. Client → Server: ACK

```
> Internet Protocol Version 4, Src: 192.168.0.13, Dst: 129.27.2.210
v Transmission Control Protocol, Src Port: 60931 (60931), Dst Port: 443 (443)
    Source Port: 60931
    Destination Port: 443
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 1638621053
    Acknowledgment number: 2622560819
    Header Length: 20 bytes
  > Flags: 0x010 (ACK)
```

Data Transfer



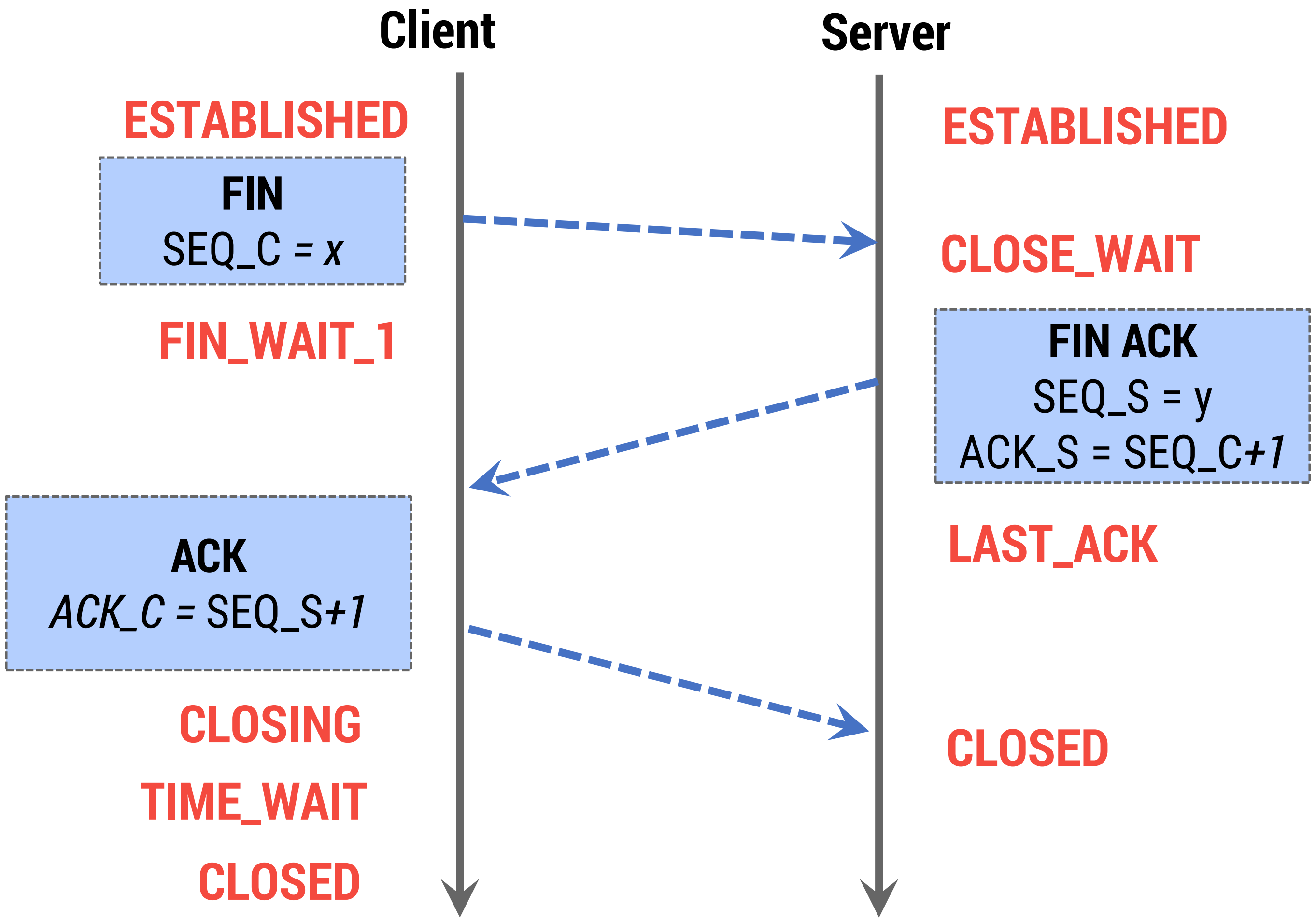
One-way example:

Client sends 70 bytes to server in two packets

1. Client: Send TCP packet
   SEQ = 731
   20 bytes payload
2. Server: Send **ACK** packet
   ACK = SEQ + 20 bytes = 751
   → Expecting byte 20 now
3. Client: Send TCP packet
   SEQ = 751
   50 bytes payload
4. Server: Send **ACK** packet
   ACK = SEQ + 50 bytes = 801

Connection Tear-Down

Note: Disconnect works equally in both directions...



Workflow

1. Client: Send **FIN** packet
   SEQ_C = x
2. Server: Send **FIN, ACK** packet
   SEQ_S = y
   ACK_S = SEQ_C + 1
3. Client: Send **ACK** packet
   ACK_C = y + 1

TCP Properties

Error Handling

Losing packets?

- Transmission errors (wrong checksums)
- Lost packets, e.g., due to bad connection

How to recognize that?

- Sender‘s task to detect and re-transmit lost data
- Basically, if no ACK received for certain byte range → data has to be re-sent
- Two mechanisms
  - Retransmission timeout (RTO)
  - Duplicate cumulative acknowledgements (DupAcks)

Error Handling

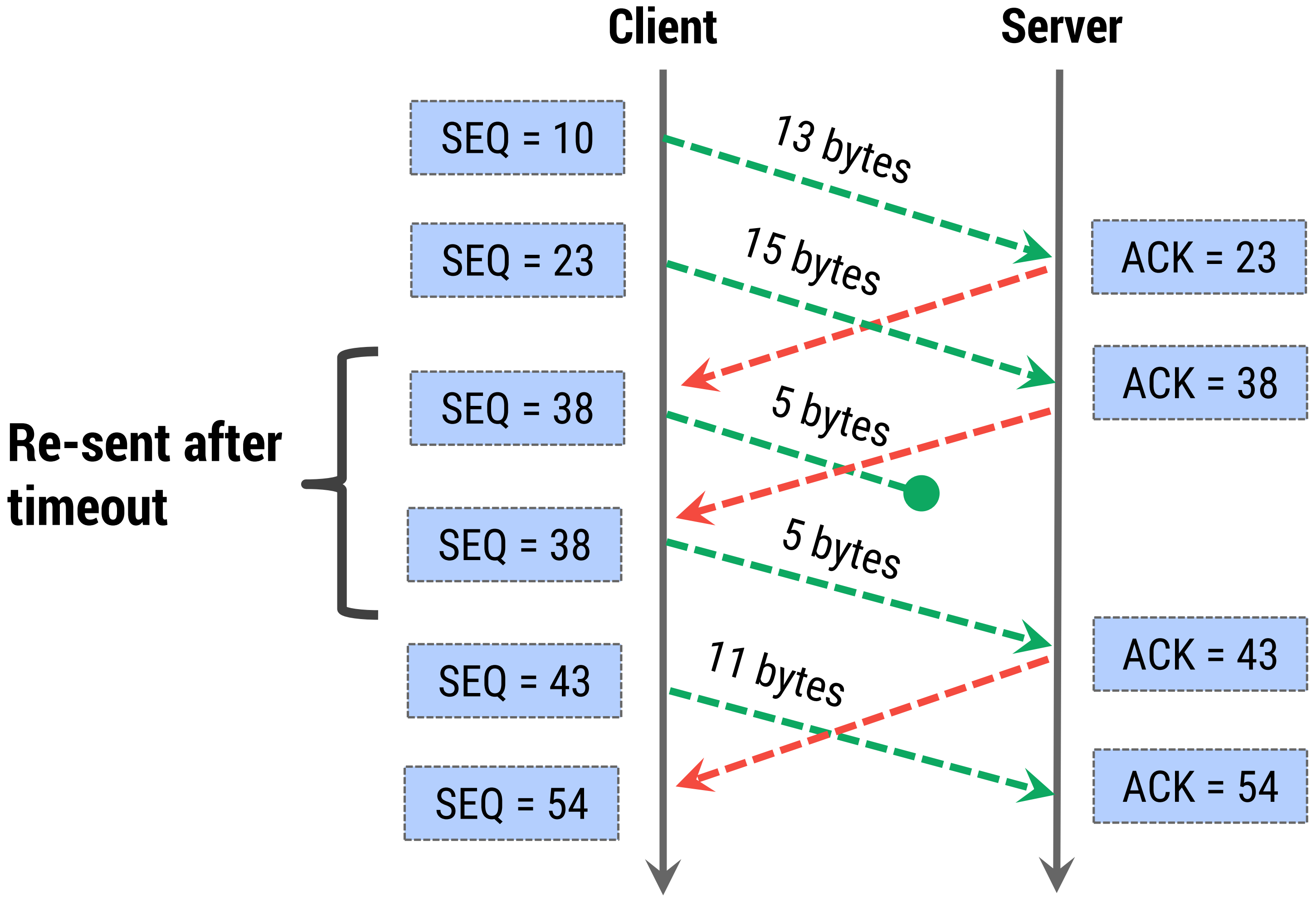
Assuming...

Packet to server is lost

→ Server does not ACK packet

Solution

Client waits for ACK until timeout and resends packet



Flow & Congestion Control

Flow Control

- Prevent sender from overwhelming receiver with too much data
  - Receiver could be busy, under heavy load or have limited buffer space
  - Consequences: Packet drops, causing re-transmissions
- Sender‘s „speed“ must be adapted to receiver
- Issue between **sender - receiver**

Congestion Control

- Prevent sender from injecting too much data into network
  - Consequences: Overload of switches / routers
- Issue between **hosts - networks**

Flow Control

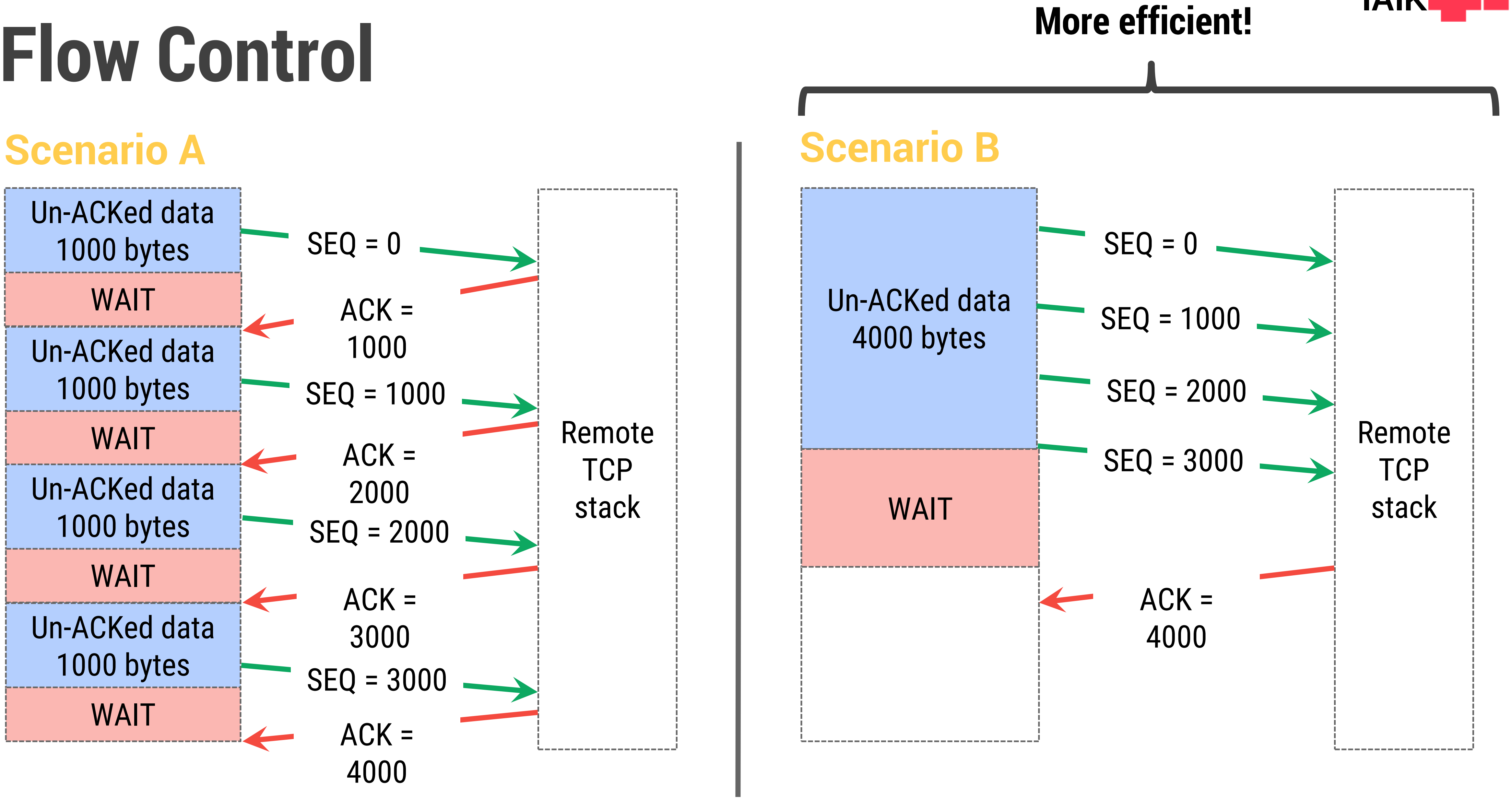
Status quo

- Ordered delivery → sequence number identifies sent byte range
- Received data acknowledged by ACK no.

→ *How much data can be sent before ACK is required?*

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

Flow Control



How can the sender determine the amount of bytes it can send before an ACK must come back? → Window size!

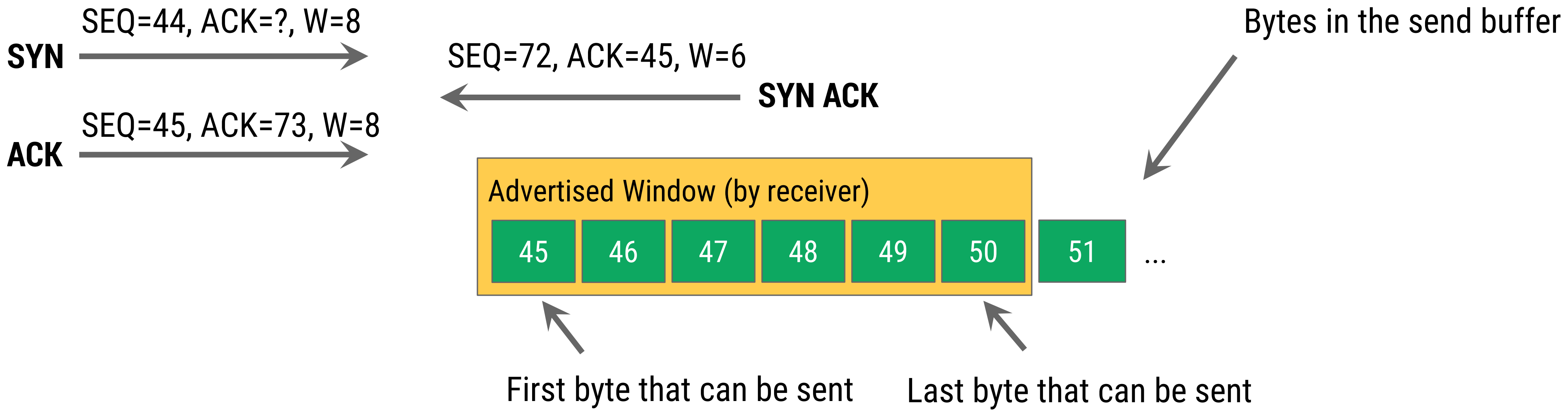
Sliding Window

Idea

Adjust amount of sendable data (= advertised window) before ACK is inevitable

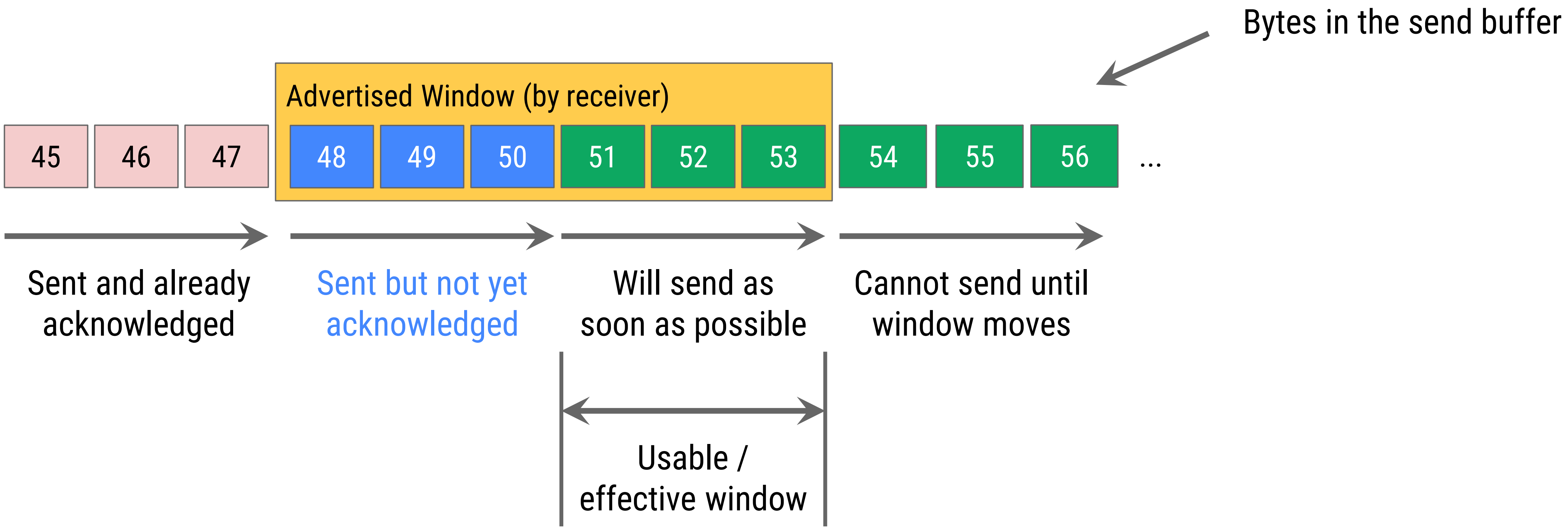
Principle

- Receiver advertises amount of bytes it is able to receive
- Starting size of window negotiated during handshake



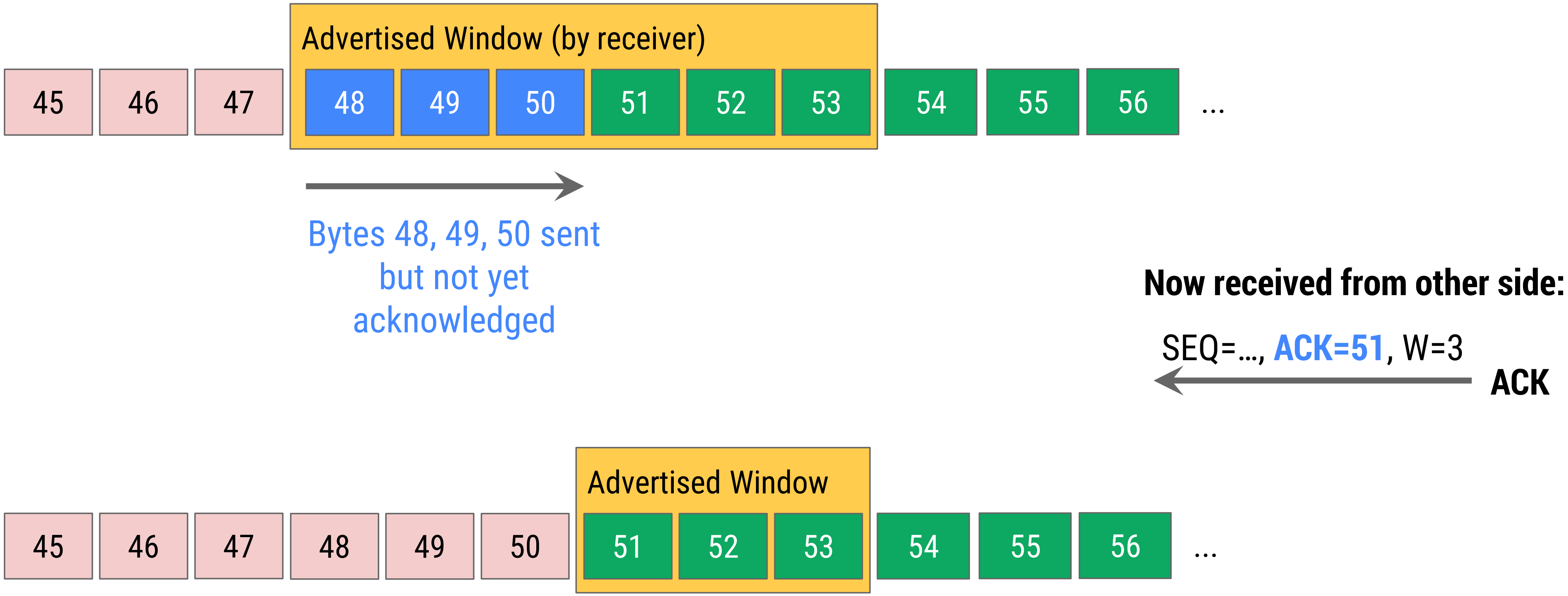
Sliding Window

Sender's point of view after sender got ACK=48, WIN=6 from receiver



Sliding Window

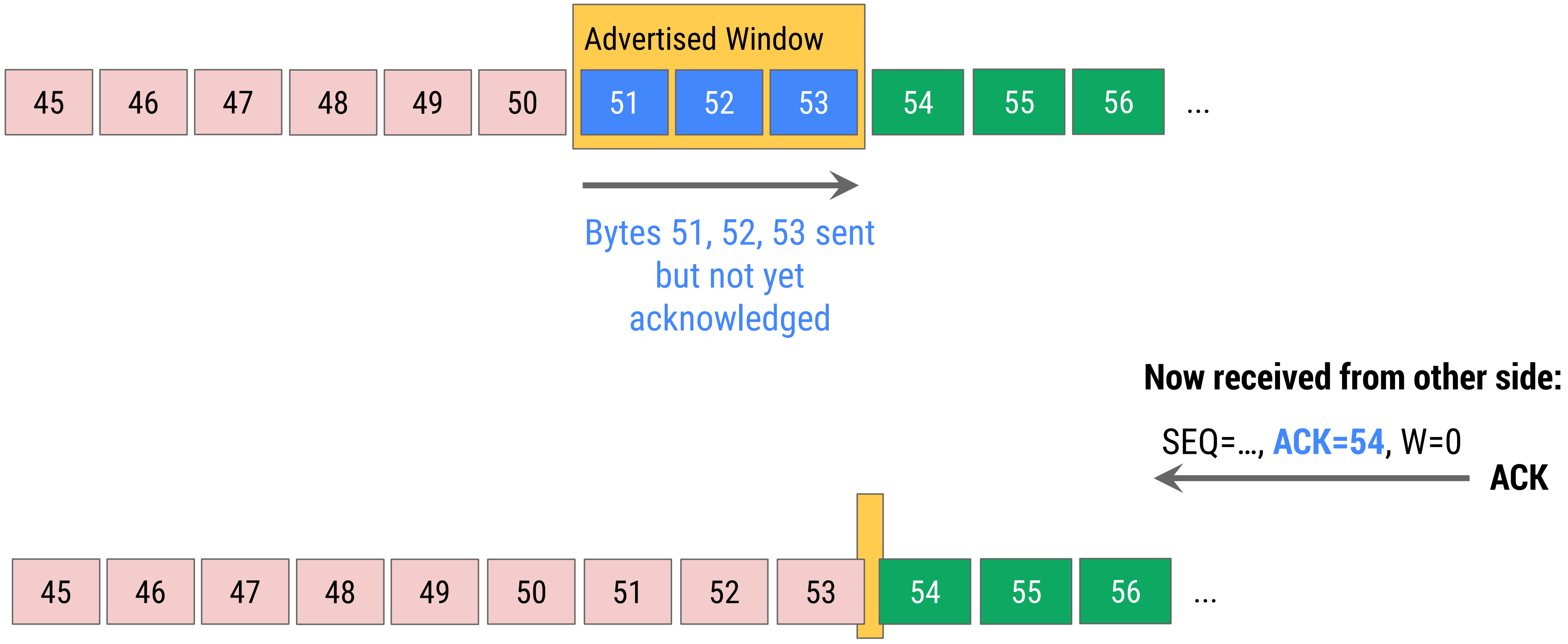
Closing the window, e.g. due to congestion…



- Sender can now send bytes 51, 52, 53 which were already granted previously
- Receiver didn‘t open the window (right edge still 53) → Congestion

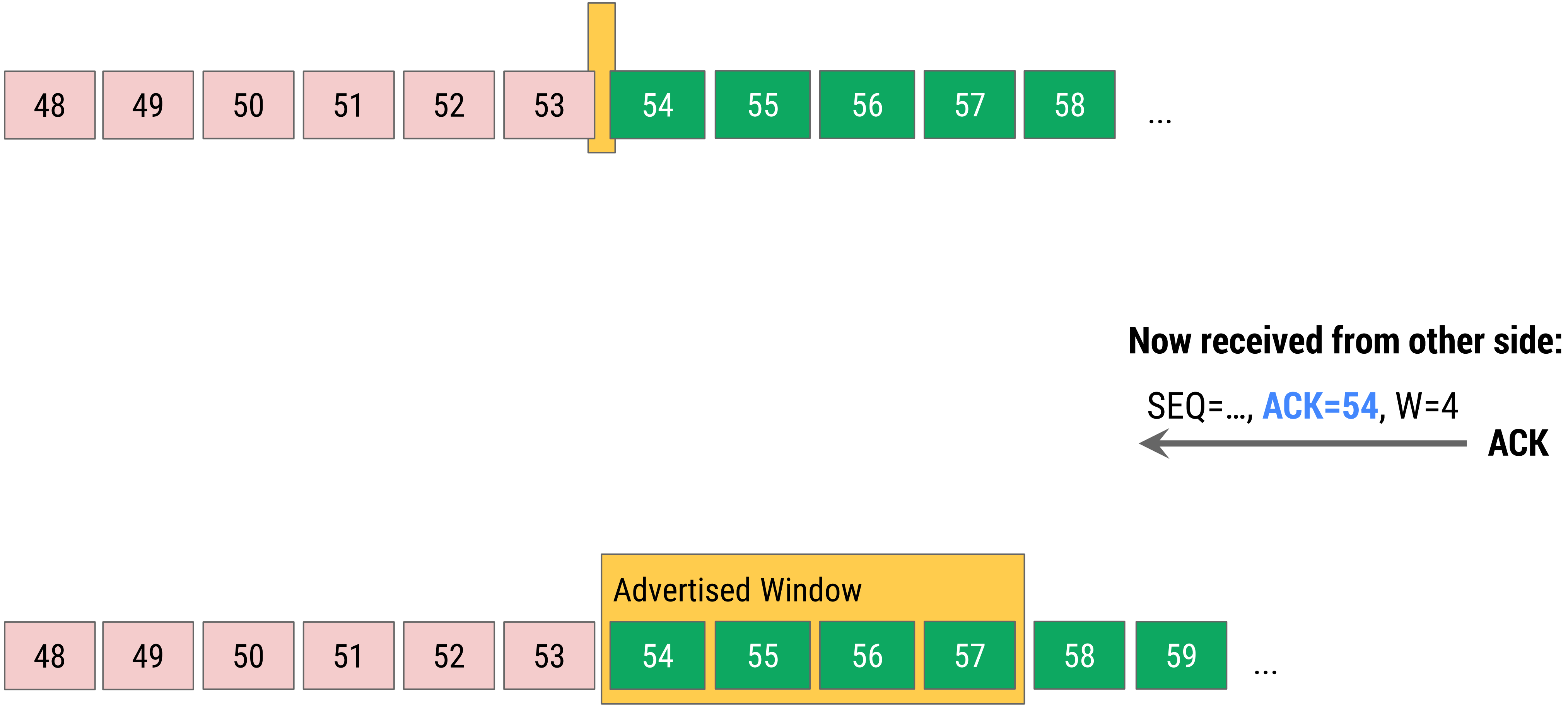
Sliding Window

Window closed...



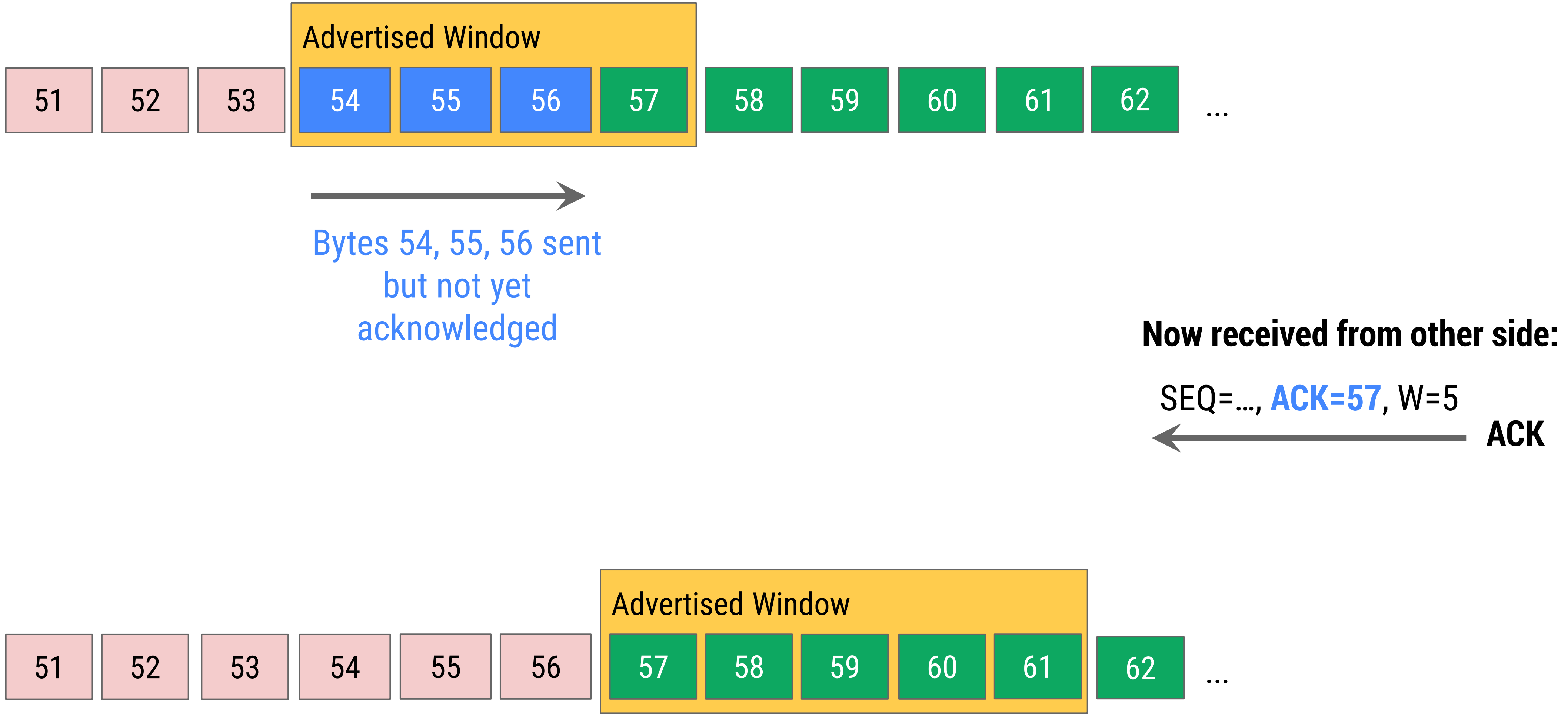
Sliding Window

Opening the window…



Sliding Window

Increasing the window…



Congestion Control

Congestion

- **Network** is overloaded → Routers / Switches cannot handle amount of traffic
- Packets are dropped causing timeouts and re-transmissons

Why Control?

- Keep data flow rate below collapse
- Achieve high performance without re-transmissions and packet drops

Approach

Maintain a „Congestion Window“ that tells sender how much data can be sent
→ Dynamic: Increased when packets are ACKed, decreased if lost (not ACKed)

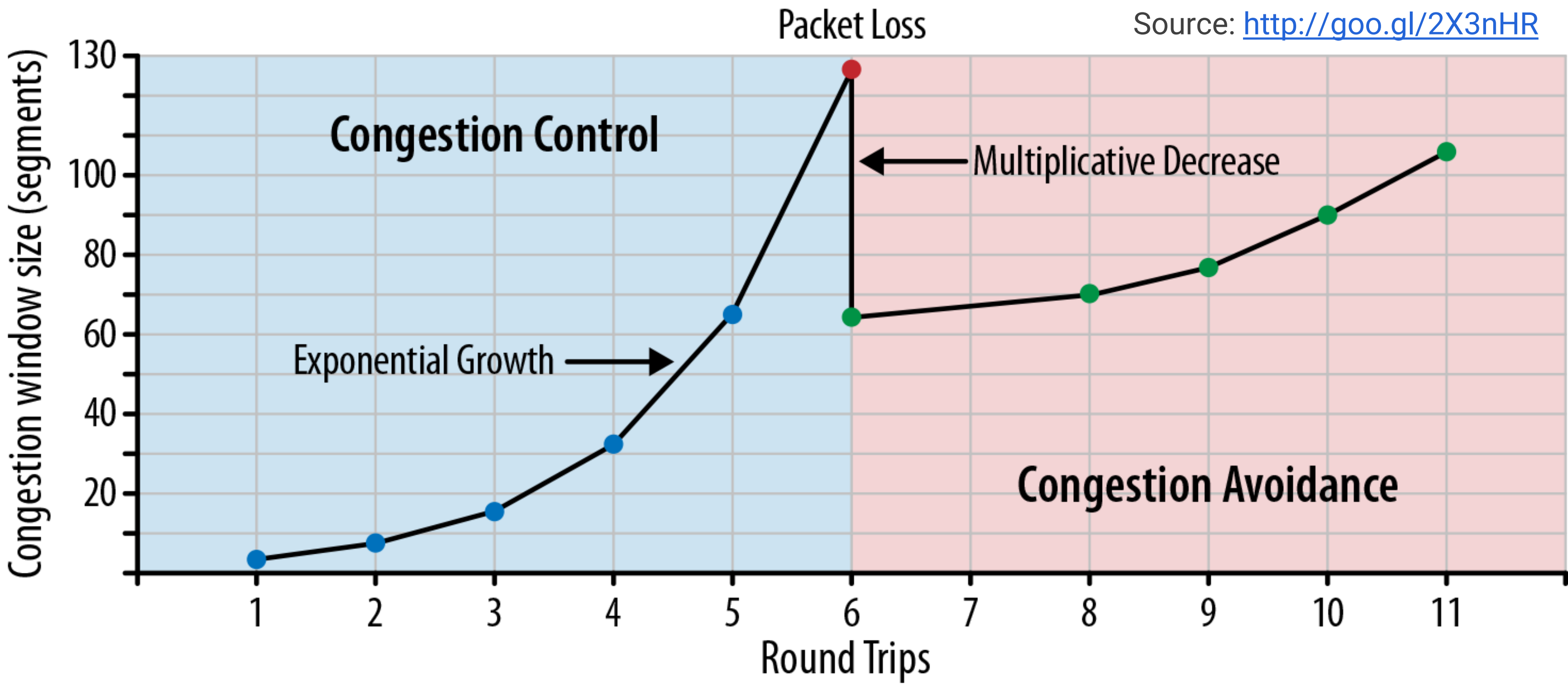
Congestion Window

Window maintained by TCP stack of sender → not part of TCP header!

Idea

- Based on a technique called additive increase / multiplicate decrease
  - Start sending small amount of data (small congestion window)
  - Increase amount of data in a linear way (additive increase)
  - When packet loss occurs, set congestion window to half (multiplicate decrease)

Source: http://goo.gl/2X3nHR



Outlook

- 16.12.2020
  - Application Layer: HTTP
    HTTP/2, AJAX, WebSockets
  - Application Layer: DNS