

Computer Organization and Networks

Chapter 7: Networking II

Winter 2022/2023



Jakob Heher, www.iaik.tugraz.at
he/his

IPv4 address space

- IPv4 addresses are 32 bits long
 - How many different IP addresses can exist?

IPv4 address exhaustion

- IPv4 addresses are 32 bits long
 - There can be at most 2^{32} different IPv4 addresses
 - $2^{32} = 4$ billion, 294 million, 967 thousand, two hundred and ninety-six
 - Global population ≈ 7.9 billion (September 2021)
- How many devices do you own that use IPv4?
 - Your home PC
 - Your phone
 - Your ISP router (twice!)
 - Laptops? Game consoles? Cars? Fridges? Doorbells?

IPv4 address exhaustion

<https://www.arin.net/vault/announcements/2015/20150924.html>

ARIN IPv4 Free Pool Reaches Zero

Posted: Thursday, 24 September 2015

On 24 September 2015, ARIN issued the final IPv4 addresses in its free pool. ARIN will

<https://www.ripe.net/publications/news/about-ripe-ncc-and-ripe/the-ripe-ncc-has-run-out-of-ipv4-addresses>

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.

<https://www.lacnic.net/4848/2/lacnic/ipv4-exhaustion:-lacnic-has-assigned-the-last-remaining-address-block>

IPv4 Exhaustion: LACNIC Has Assigned the Last Remaining Address Block

19 August 2020

The Latin American and Caribbean Internet Address Registry (LACNIC) announces that the last available IPv4 address block has been reserved.

IPv4 address exhaustion

- The internet is out of IPv4 addresses...
- Somehow, your new phone still works?
- There are ways around address exhaustion
 - We'll talk about this later!

IPv6

- Internet Protocol, version 6
- Successor to IPv4

- Not natively interoperable with IPv4
 - IPv4-only devices cannot communicate with IPv6-only devices
 - Most modern devices implement *both* IPv4 and IPv6
 - Eventually, IPv4 will be phased out...

IPv6 addressing

```
Ethernet adapter Ethernet:  
  
Link-local IPv6 Address . . . . . : fe80::10e5:f700:f6ab:7afc
```

- 128-bit address
 - Notation: 16-bit hexadecimal blocks separated by colons (:)
 - Zero blocks can be omitted using double colon (::)
 - **fe80::10e5:f700:f6ab:7afc** is the same as
fe 80 00 00 00 00 00 00 10 e5 f7 00 f6 ab 7a fc

IPv6 addressing

```
Ethernet adapter Ethernet:  
  
Link-local IPv6 Address . . . . . : fe80::10e5:f700:f6ab:7afc  
IPv6 Address . . . . . :  
IPv6 Address . . . . . :  
IPv6 Address . . . . . :
```

- 64-bit *network prefix*, 64-bit *interface identifier*
- A single interface (e.g.: a network card) may have multiple addresses
 - Addresses share the *interface identifier*
- Addresses have a *scope* in which they are valid

IPv6 scoping

- Global addresses
 - Valid in any network connected to the internet
 - May be routed on the public internet
- Unique-local addresses (in **fc00::/7**)
 - Same idea as IPv4 private networks
 - No assignment/registration needed
 - Routed only in local networks, but *not* on the public internet
- Link-local addresses (in **fe80::/64**)
 - Only valid within the Link Layer network

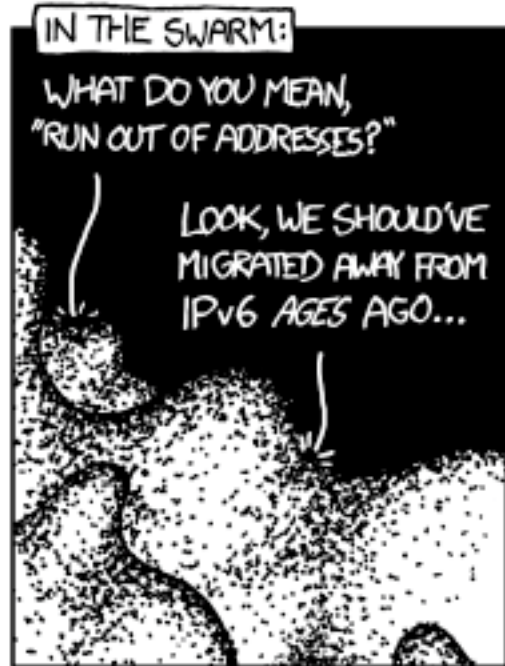
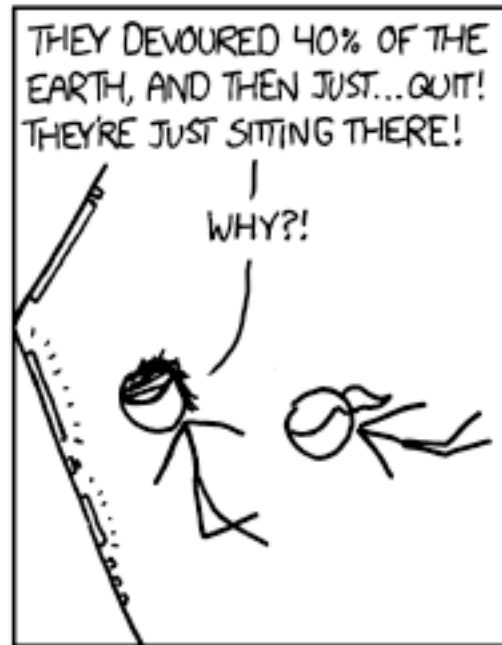
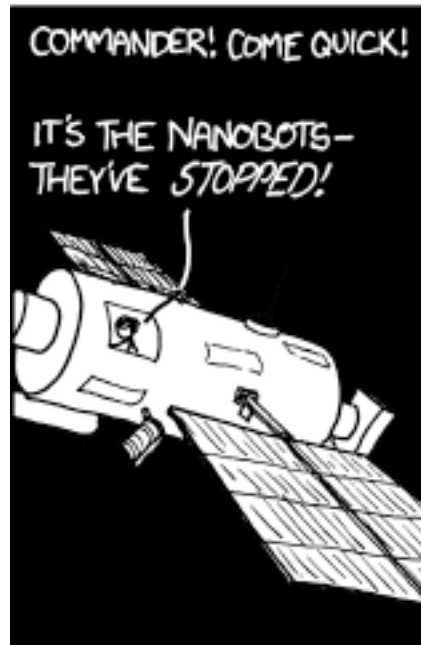
IPv6 packet overview

- Similar fields to IPv4 packets
 - Version is always **0110** (version 6)
 - Length, Source and Destination fields
 - Optional extension header blocks
- Header checksum removed
 - Relies on Link Layer to provide error detection
- Fragmentation (mostly) removed
 - No fragmentation by routers
 - Fragmentation by hosts only as an extension
 - Transport Layer is expected to perform fragmentation

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																																
28	224	Destination address																															
32	256																																
36	288																																

IPv6 recap

- Successor to IPv4
 - “Permanent” solution to IP address exhaustion
 - We’ll talk about IPv4 workarounds in a bit!
 - Some protocol-level improvements
 - Not interoperable with IPv4
- Supported by most modern end-user devices
 - Server-side support is... still lacking [<https://ipv6.watch>]
- 128-bit addresses (64-bit network part, 64-bit interface identifier)
 - 2^{64} networks, each consisting of 2^{64} hosts





The Transport Layer

The Transport Layer

- Computers A and B are capable of sending data to each other
- **Goal:** Allow multiple applications to communicate reliably
- **Concerns:**
 - How to distinguish which application data is meant for? (multiplexing)
 - What if data is lost on the lower layers? (reliability)
 - How much data can the network handle? (congestion control)
 - How much data can the receiver handle? (flow control)

The Transport Layer

- The internet has two widely-used protocols at the Transport Layer:
 - **Transmission Control Protocol**
 - Focused on reliable delivery
 - Connection-based
 - **User Datagram Protocol**
 - Focused on speed
 - Connectionless

The Transport Layer: Ports

- Concept used for both TCP and UDP
- Source and destination identified by *port number*
 - 16 bits (65536 available ports)
 - TCP and UDP ports are *separate*
 - The protocols implement the same idea, but each only cares about its own ports...
- Common notation: Port number after IP address
 - **127.0.0.1:8000** is port 8000 at host 127.0.0.1
 - **[:, :, 1]:8000** is port 8000 at host ::1

UDP

- Fire-and-forget transmission of single datagrams
 - Useful for real-time applications
- Data may never arrive, may arrive out of order, ...
 - Data loss must be tolerable for the upper-layer application
- Extremely simple and straightforward

UDP datagram 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	Length																Checksum															

TCP

- Highly reliable transmission of a byte stream
 - Acknowledgments and re-transmission
 - Guaranteed to maintain data ordering
- Non-trivial protocol overhead
 - Still better than re-inventing the wheel if you need it!

TCP

- TCP connections have two sides: *server* and *client*
- *Server* listens on a specific port
 - Server port is fixed for all connections
- *Client* connects to that port on the server
 - Client uses a “random” ephemeral port, different for each connection
 - See for yourself: **netstat -onb** (Win) or **netstat -tnap** (Linux, Mac)
- Connections are uniquely identified by *client IP + client port*

The Transport Layer: Ports

- Two applications can't use the same port number
- Client needs to know which port number to connect to
- Port numbers are standardized by IANA
 - 0–1023: *well-known* ports
 - Examples: 22 (SSH), 80 (HTTP), 123 (NTP), 194 (IRC), 443 (HTTPS), ...
 - 1024–49151: *registered* ports
 - Most server applications will use this range (even unregistered ones...)
 - 49152–65535: *dynamic* ports
 - Most OS will use this range for ephemeral (client) ports

TCP packet overview

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset																Window size															
16	128	Checksum																Urgent pointer (if URG set)															

- Source + destination ports allow identification of connection
- Checksum over entire header + data

TCP data ordering

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset								Reserved 000								ACK								Window size							
16	128	Checksum																Urgent pointer (if ACK set)															

- TCP maintains a *sequence number* across the entire connection
 - Separate number for each end's packets
- Receipt of contiguous data confirmed via *acknowledgment number*
 - Acknowledgement number := next expected sequence number
- This allows ordering of data and re-sending of lost packets!

TCP data ordering

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset								Reserved 000								ACK		SYN		Window size											
16	128	Checksum																Urgent pointer (if URG set)															

- Connection establishment: Three-way handshake
 - Client -> Server: **SYN**
 - Server -> Client: **SYN + ACK**
 - Client -> Server: **ACK**

TCP data ordering

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset								Reserved 000								SYN								Window size							
16	128	Checksum																Urgent pointer (if URG set)															

- Client -> Server: **SYN**
 - Sequence number: **seq_c**, chosen randomly

TCP data ordering

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset								Reserved 000								ACK		SYN		Window size											
16	128	Checksum																Urgent pointer (if URG set)															

- Server -> Client: **SYN + ACK**
 - Sequence number: **seq_s**, chosen randomly
 - Acknowledgement: **seq_c+1**

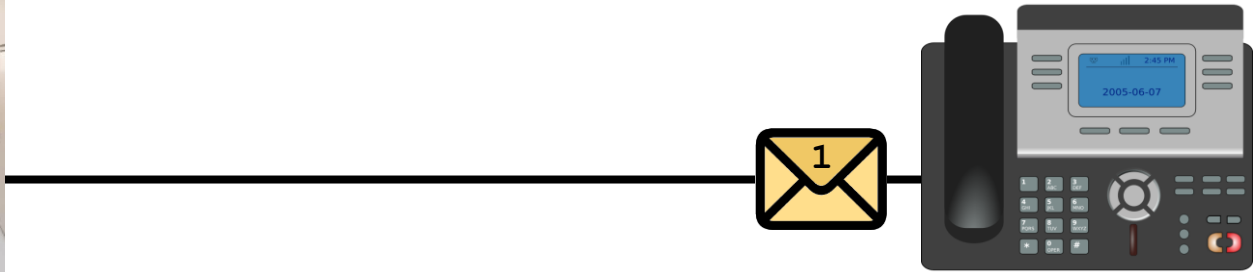
TCP data ordering

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset								Reserved 000								ACK								Window size							
16	128	Checksum																Urgent pointer (if URG set)															

- Client -> Server: **ACK**
 - Sequence number: **seq_c+1**
 - Acknowledgement: **seq_s+1**
- Now both sides know that the other side has their sequence number
 - Ready to communicate!

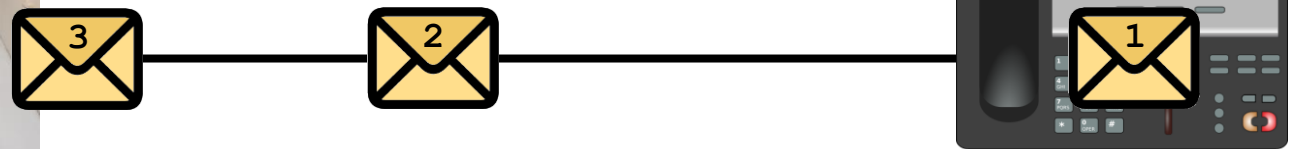
TCP flow control

- Imagine: a supercomputer talking to a desk phone via a 100Gbps link



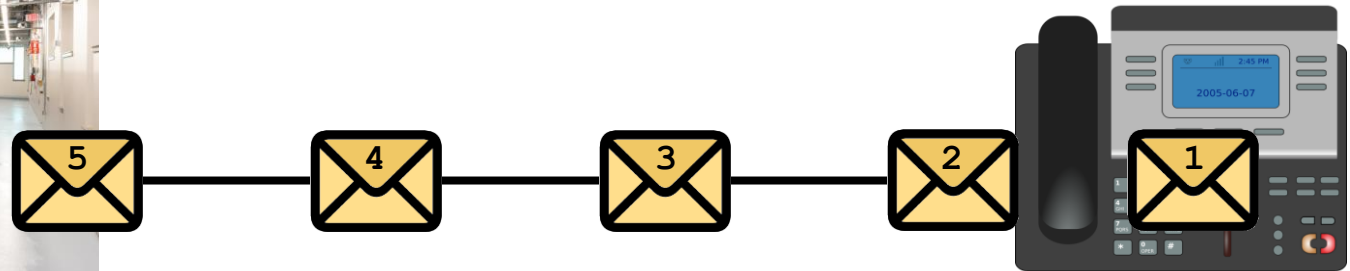
TCP flow control

- Imagine: a supercomputer talking to a desk phone via a 100Gbps link



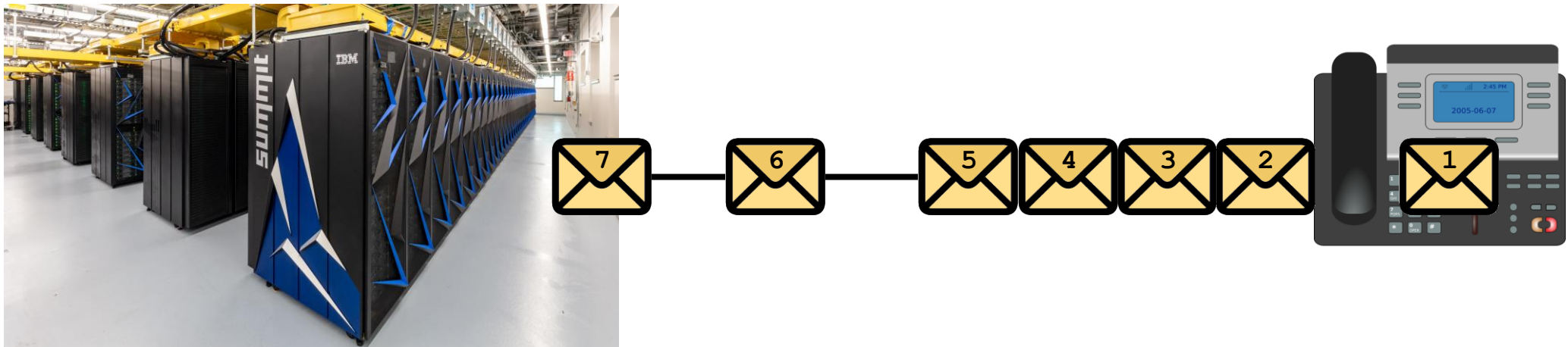
TCP flow control

- Imagine: a supercomputer talking to a desk phone via a 100Gbps link



TCP flow control

- Imagine: a supercomputer talking to a desk phone via a 100Gbps link



- The desk phone doesn't stand a chance to keep up!

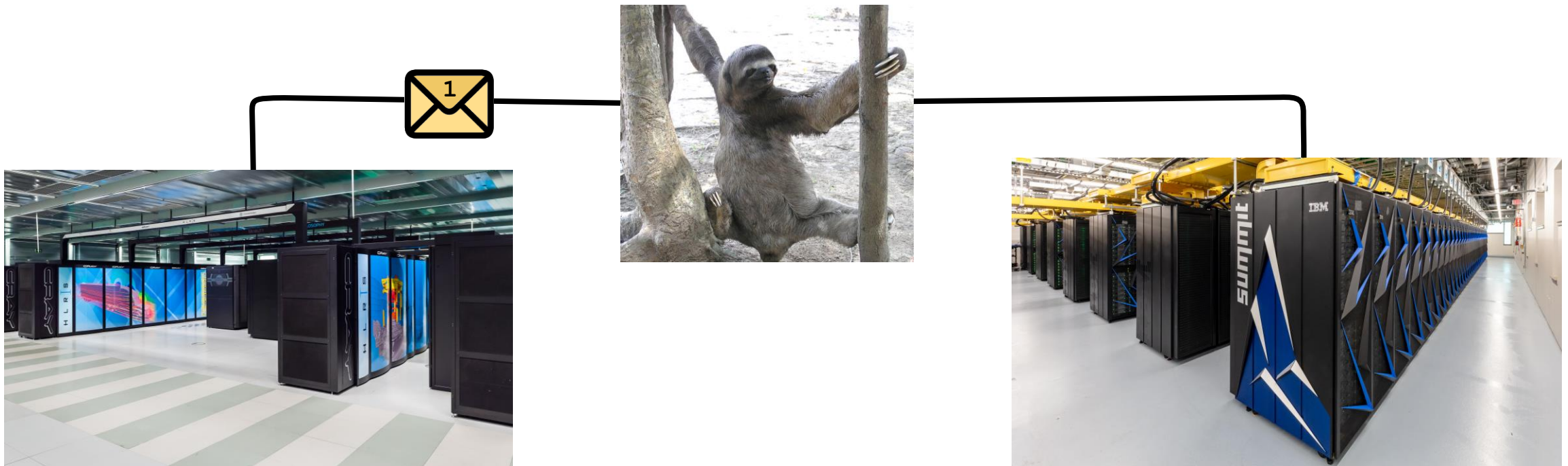
TCP flow control

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset																Window Size															
16	128	Checksum																Urgent pointer (if URG set)															

- Imagine: a supercomputer talking to a desk phone via a 100Gbps link
- *Window size* indicates how much more data the host can handle
- The other end must throttle its transmission rate to accommodate
 - Window size is relative to the last ACK'd packet

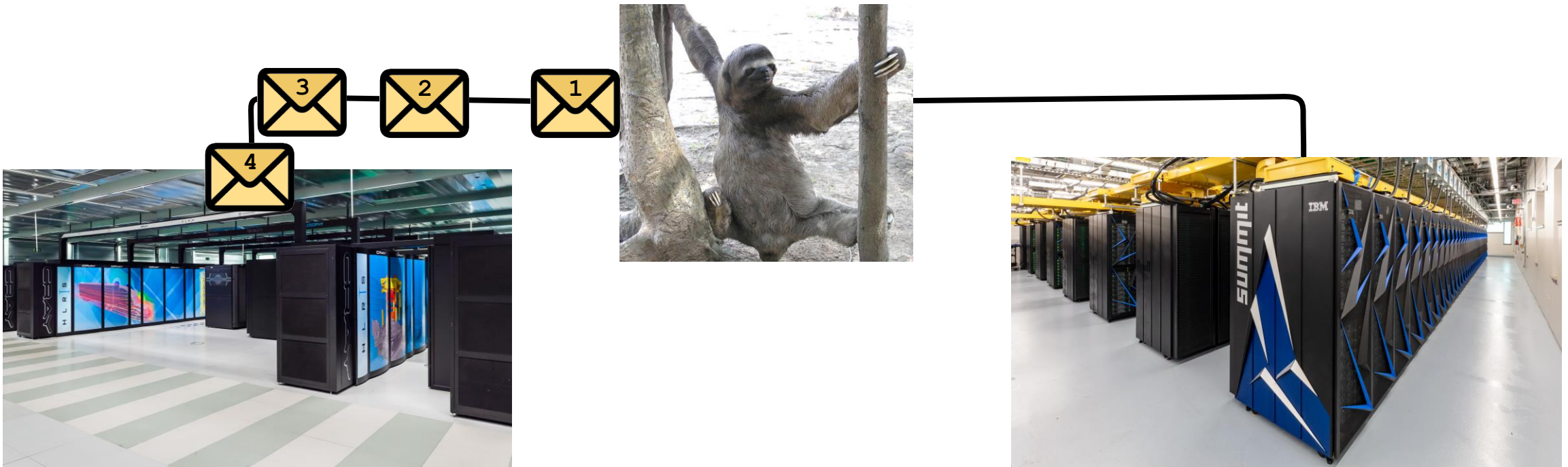
TCP congestion control

- Imagine: two supercomputers talking via a dial-up connection
 - Keep in mind: the “dial-up connection” could be some intermediate network



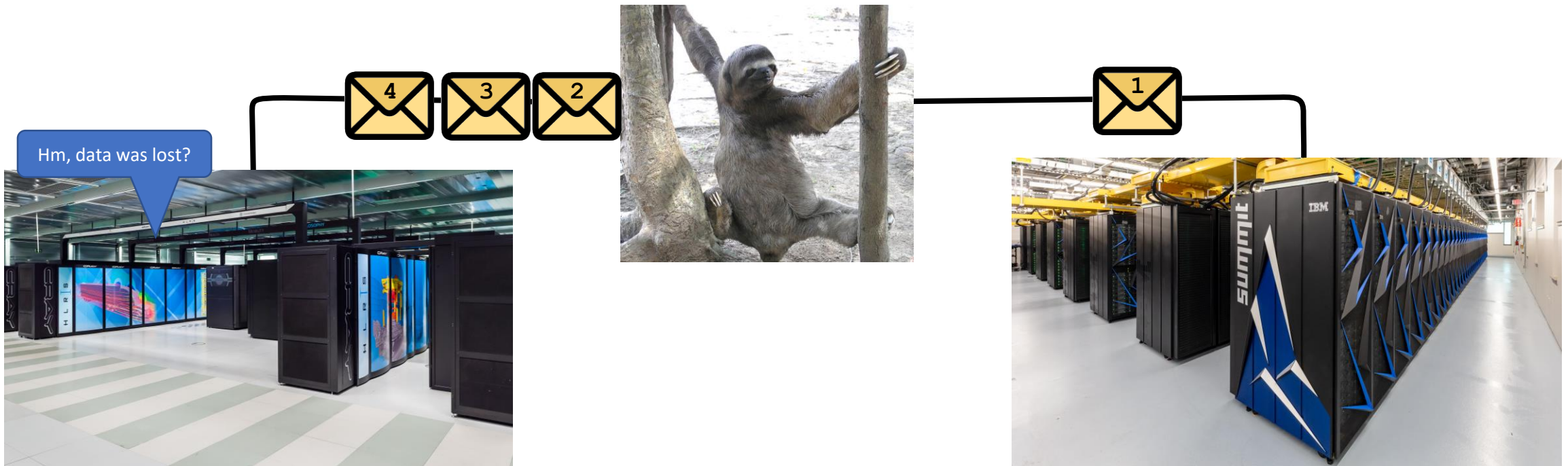
TCP congestion control

- Imagine: two supercomputers talking via a dial-up connection
 - Keep in mind: the “dial-up connection” could be some intermediate network



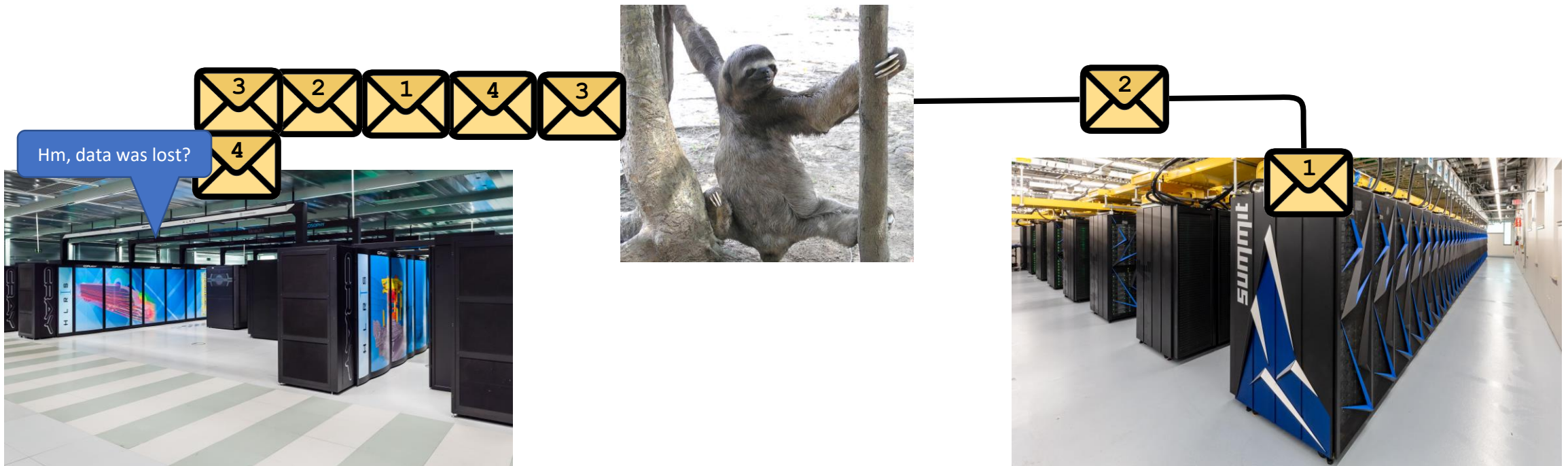
TCP congestion control

- Imagine: two supercomputers talking via a dial-up connection
 - Keep in mind: the “dial-up connection” could be some intermediate network



TCP congestion control

- Imagine: two supercomputers talking via a dial-up connection
 - Keep in mind: the “dial-up connection” could be some intermediate network



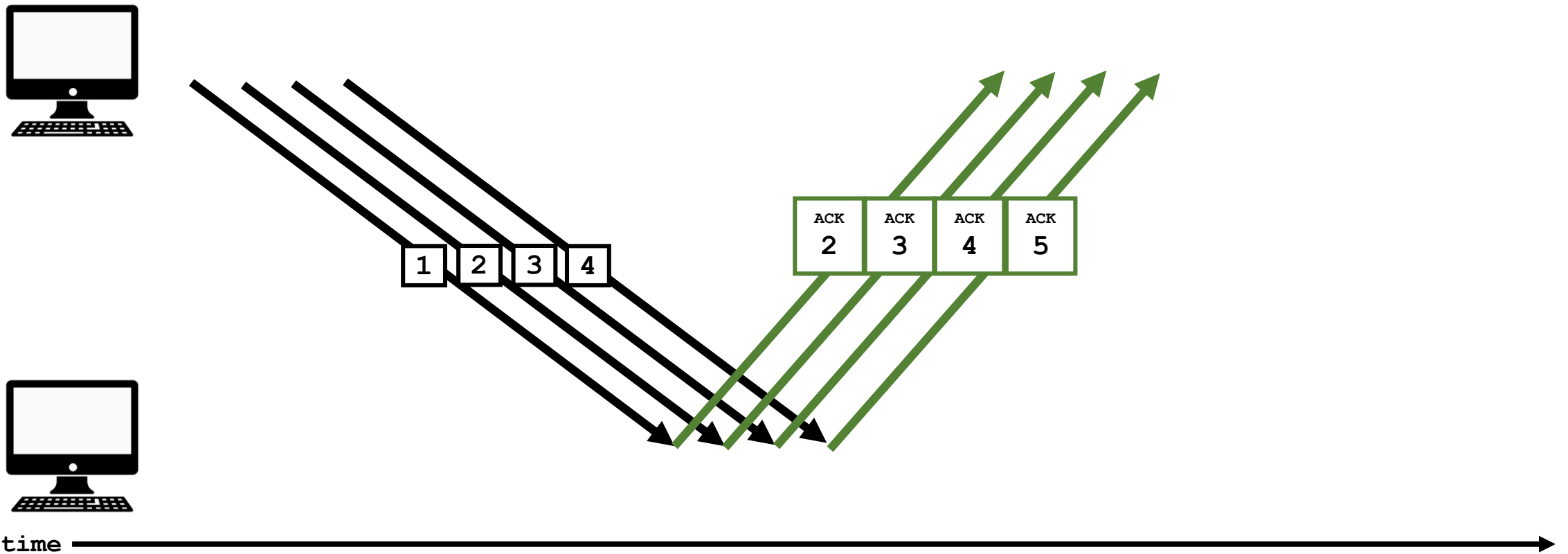
TCP congestion control

- Imagine: two supercomputers talking via a dial-up connection
 - Keep in mind: the “dial-up connection” could be some intermediate network
- If you just keep shoving data...
 - ... it will get stuck in a queue somewhere ...
 - ... so you think it was lost and send it again ...
 - ... now your queue is twice the size ...
 - ... and nothing useful gets done.
- How do we avoid that?

TCP congestion control

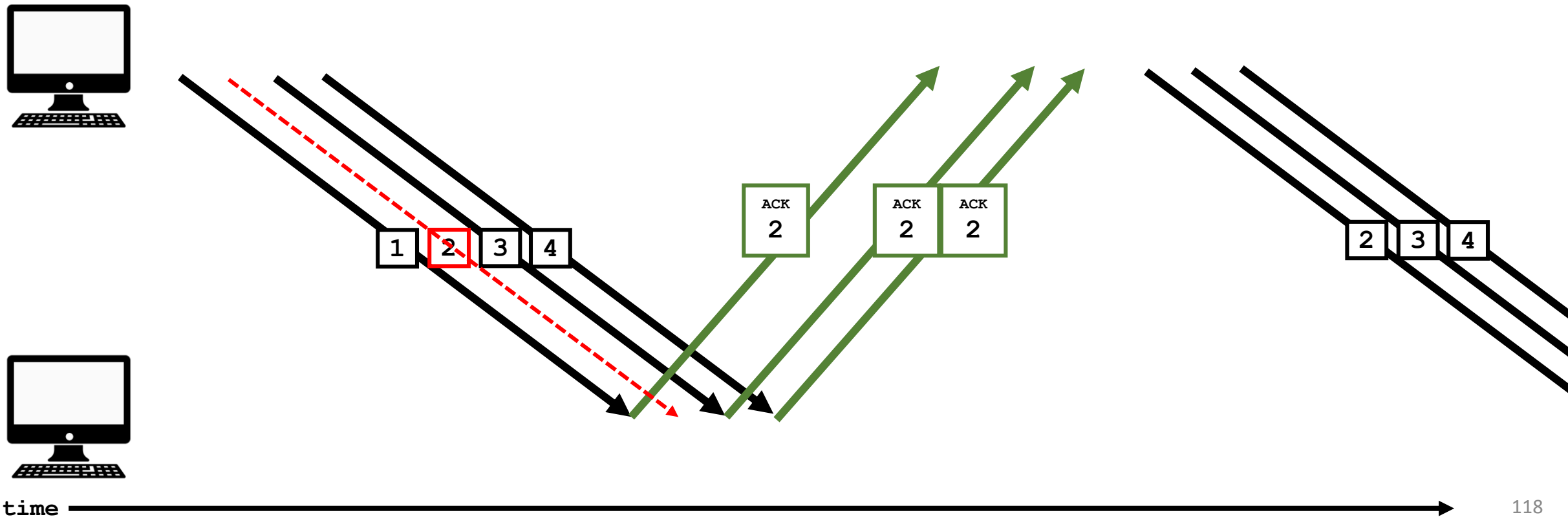
- Each side throttles its data transmission rate independently
 - No cooperation required
 - Different OS have different algorithms
- Basic concept:
 - Start at a relatively slow rate, then increase speed until data gets lost
 - Once data is lost, assume we overloaded the connection and slow down again
- Details differ from OS to OS

TCP: Selective Acknowledgment



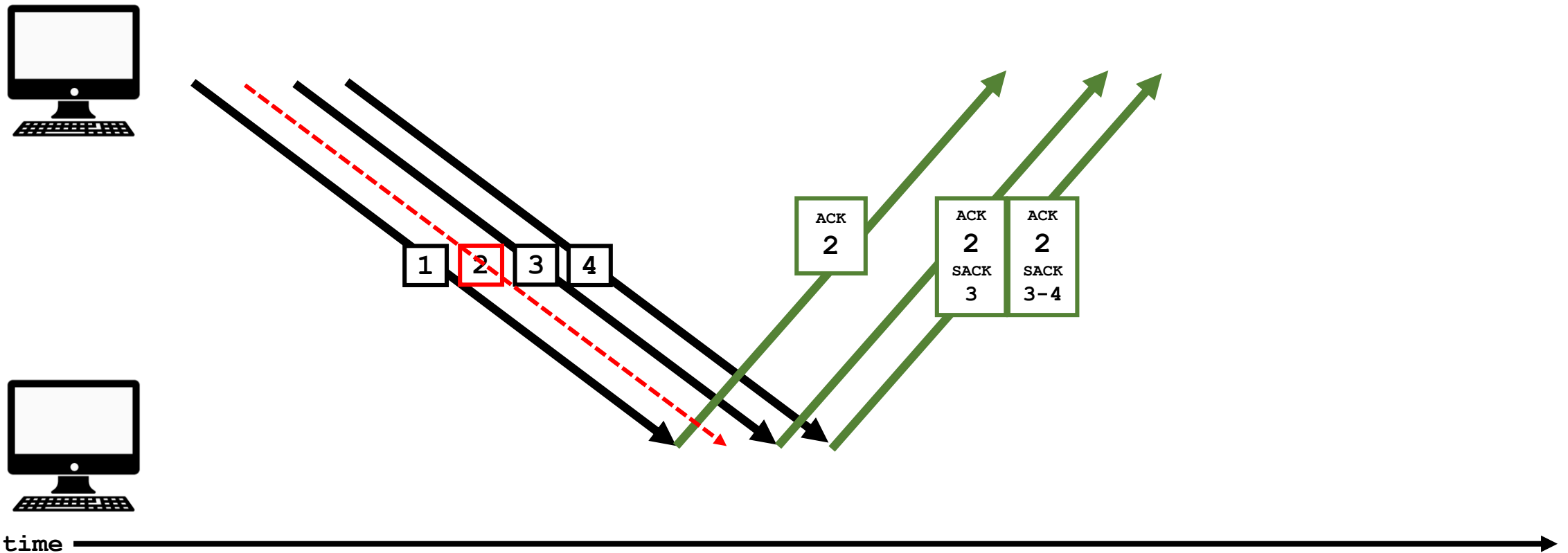
TCP: Selective Acknowledgment

- “Standard” TCP does not deal with packet loss efficiently
 - Superfluous data is re-sent, wasting time and bandwidth!



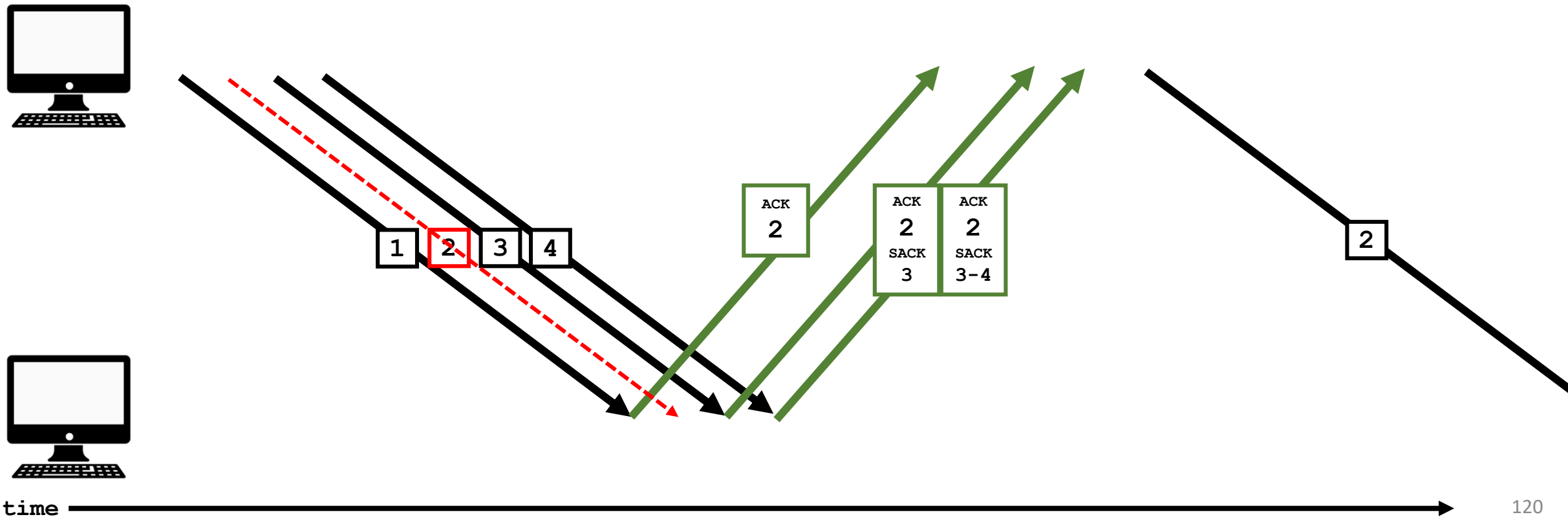
TCP: Selective Acknowledgment

- The SACK extension lets the recipient acknowledge further ranges



TCP: Selective Acknowledgment

- The SACK extension lets the recipient acknowledge further ranges
 - These ranges do not need to be re-sent!



Transport Layer recap

- Two main protocols: TCP and UDP
 - TCP: highly reliable, but comes with overhead
 - UDP: low overhead, but no reliability guarantees
- *Port numbers* identify target application
 - By convention, low port numbers (0–1023) are reserved for specific services
 - 1024–49151 are used by other servers
 - 49152–65535 are used for ephemeral ports

TCP recap

- *Client* establishes connection to *Server*
 - Server listens on a pre-agreed port
 - Client uses a “random” port (49152–65535)
- *Sequence numbers* and *acknowledgement numbers*
 - Client and server have separate counters
 - Acknowledgement of received data using the other side’s counter
 - Re-ordering and re-sending if necessary

TCP recap

- ***Flow Control*** protects the ***recipient***
 - Recipient advertises its capacity
 - Sender has to abide by it
- ***Congestion Control*** protects the ***network***
 - Transmission rate is gradually increased
 - Throttled back if packet loss is detected
 - Each side handles this independently
 - Details differ from OS to OS



Image used under Pixabay License

The Network Layer

(again?)

Recap: IPv4 address exhaustion

- IPv4 addresses are 32 bits long
 - 2^{32} is about 4 billion
- Every Internet-enabled device needs an address to communicate
 - There are a lot of devices

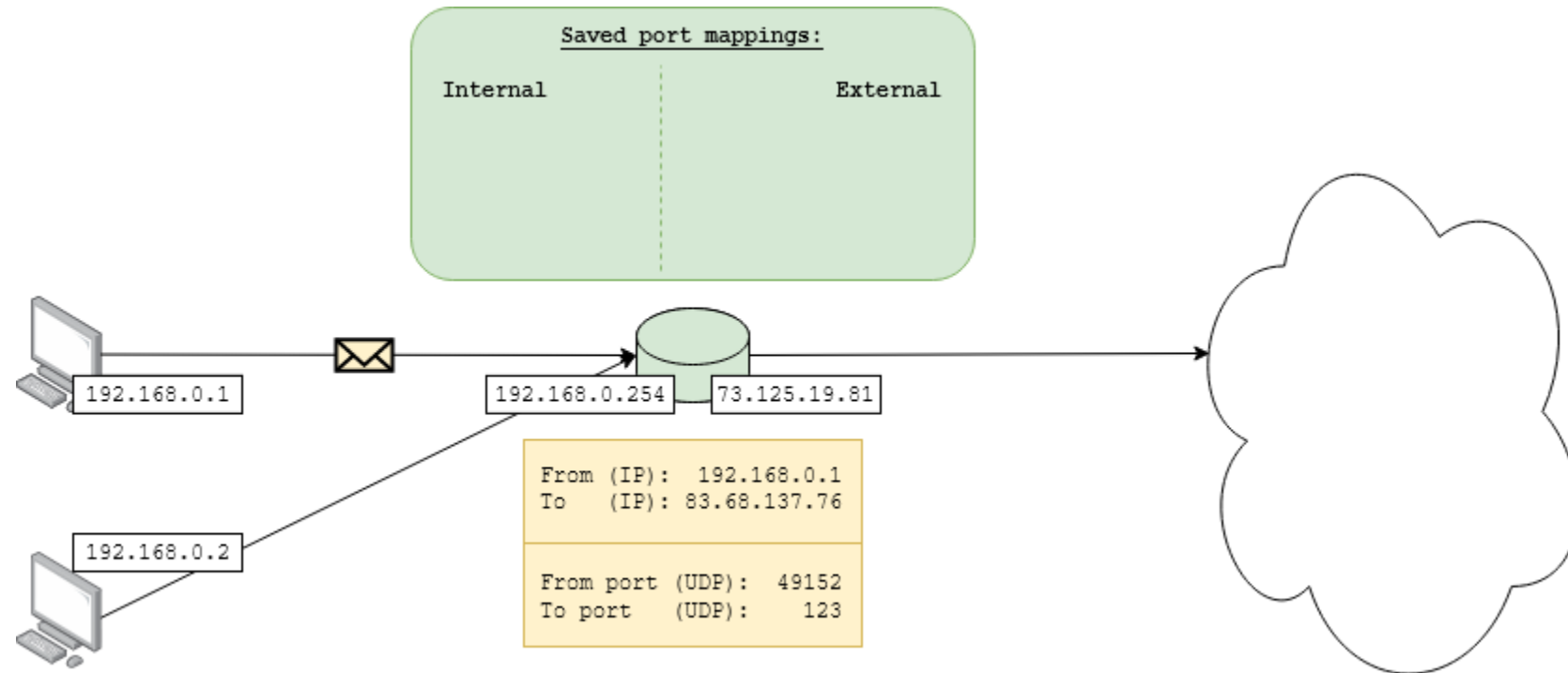
- The internet is (mostly) out of IPv4 addresses!

Network Address Translation

- “Hide” an entire private network behind a single public IP
 - Rewrite IP packets at the boundary
- Also known as:
 - “NAT”
 - Port Address Translation (“PAT”)
 - Network Address and Port Translation (“NAPT”)
 - NAT overloading
 - IP masquerading

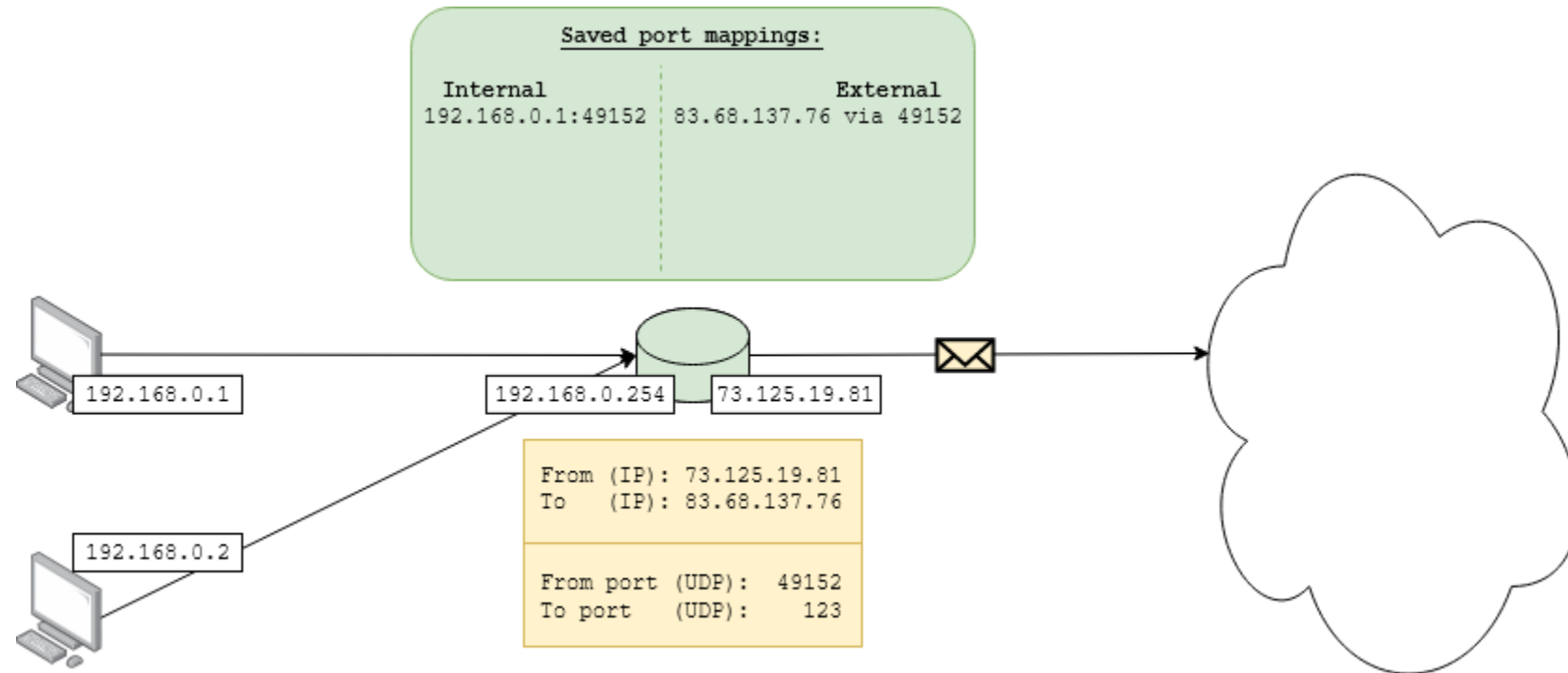
Network Address Translation

- “Hide” an entire private network behind a single public IP
 - Rewrite IP packets at the boundary



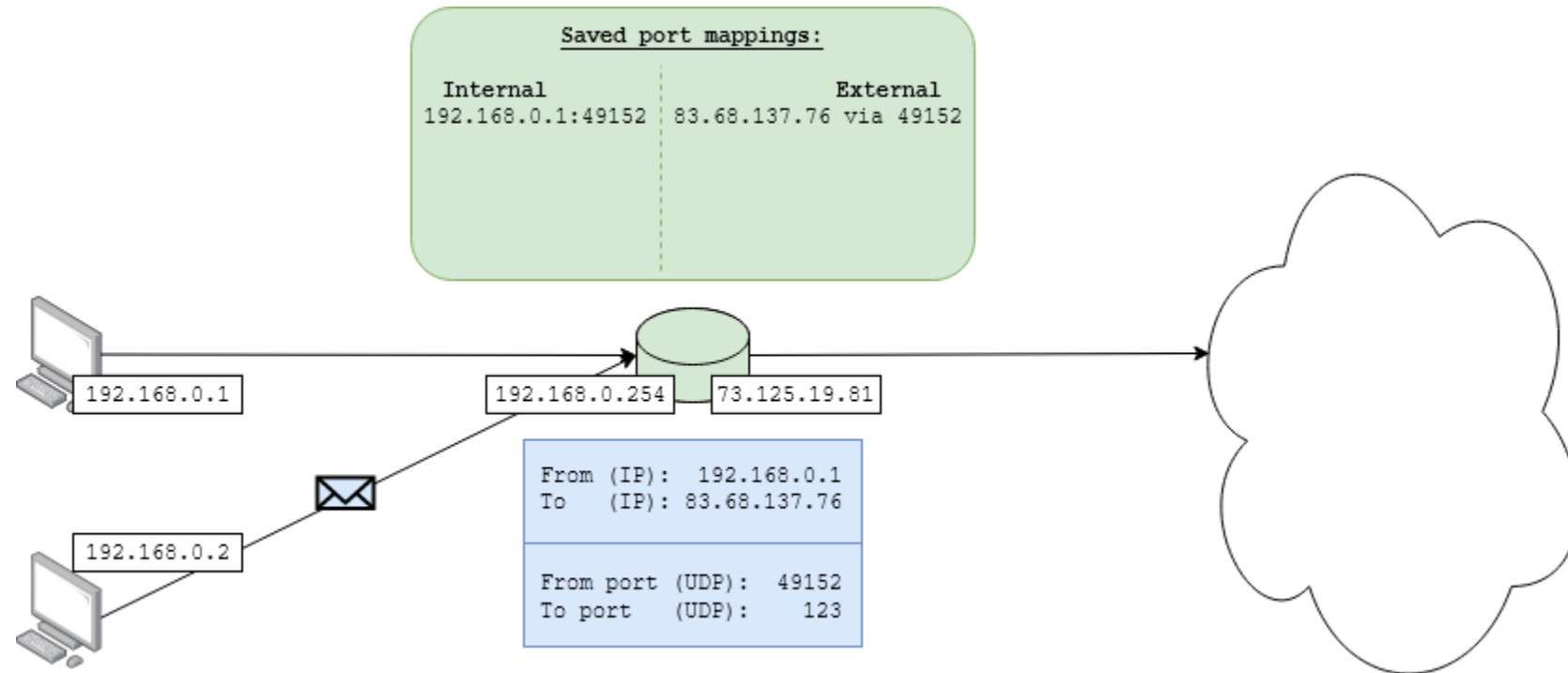
Network Address Translation

- “Hide” an entire private network behind a single public IP
 - Rewrite IP packets at the boundary



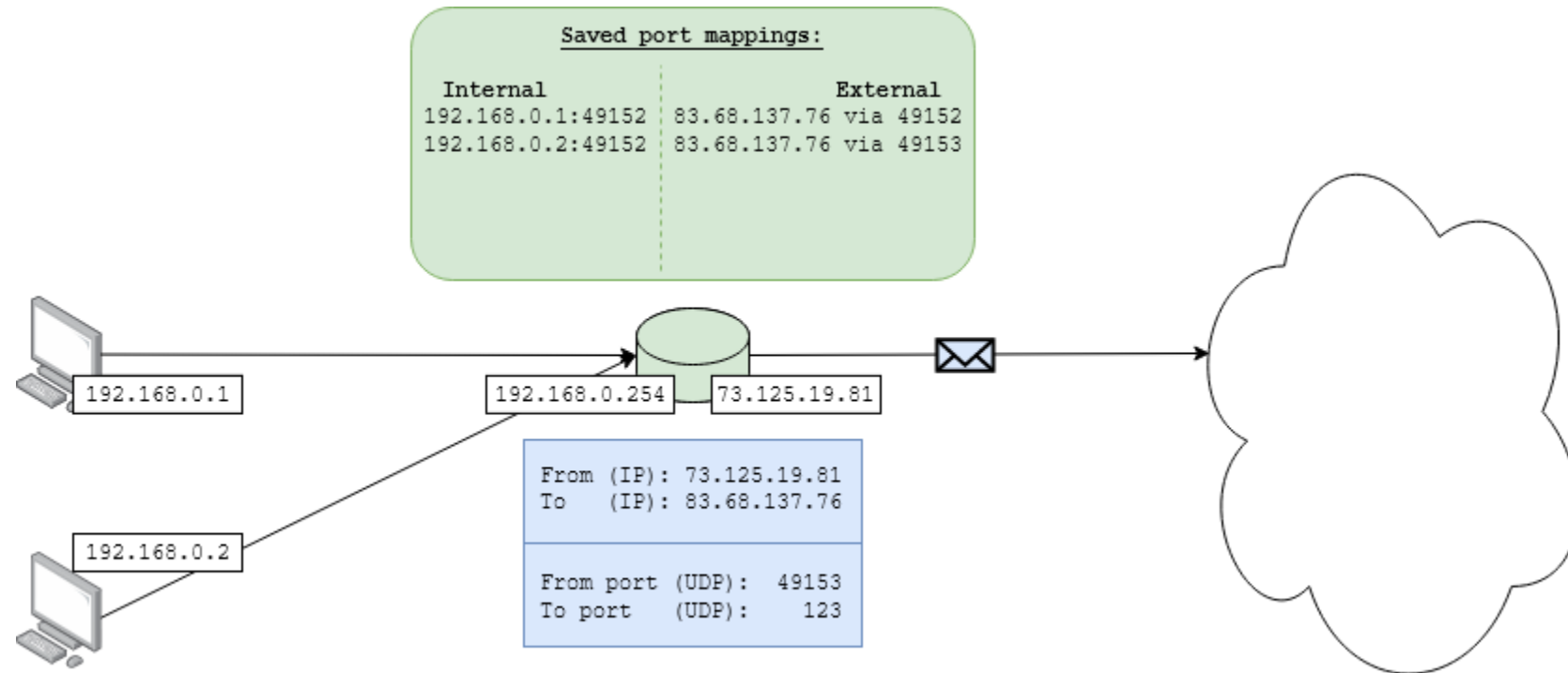
Network Address Translation

- “Hide” an entire private network behind a single public IP
 - Rewrite IP packets at the boundary



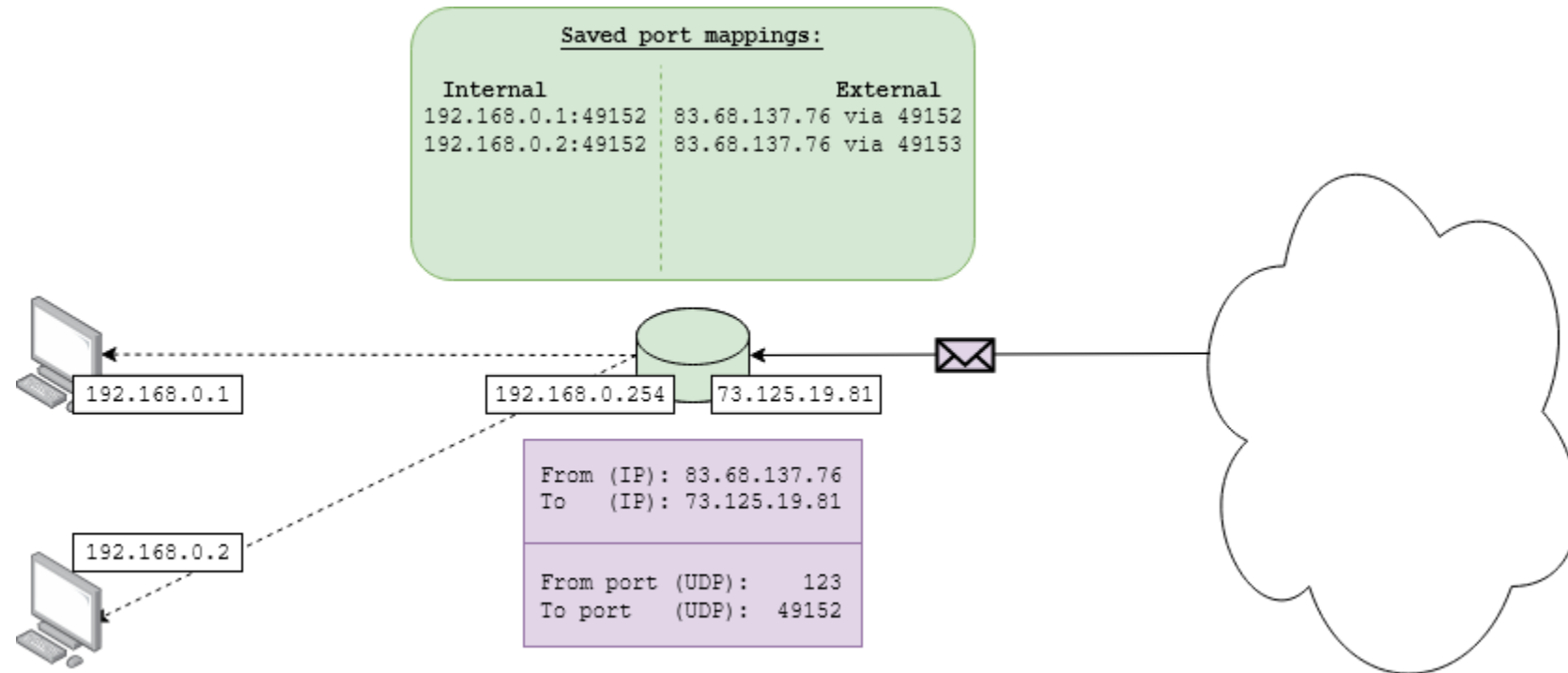
Network Address Translation

- “Hide” an entire private network behind a single public IP
 - Rewrite IP packets at the boundary
 - Rewrite TCP/UDP ports to disambiguate



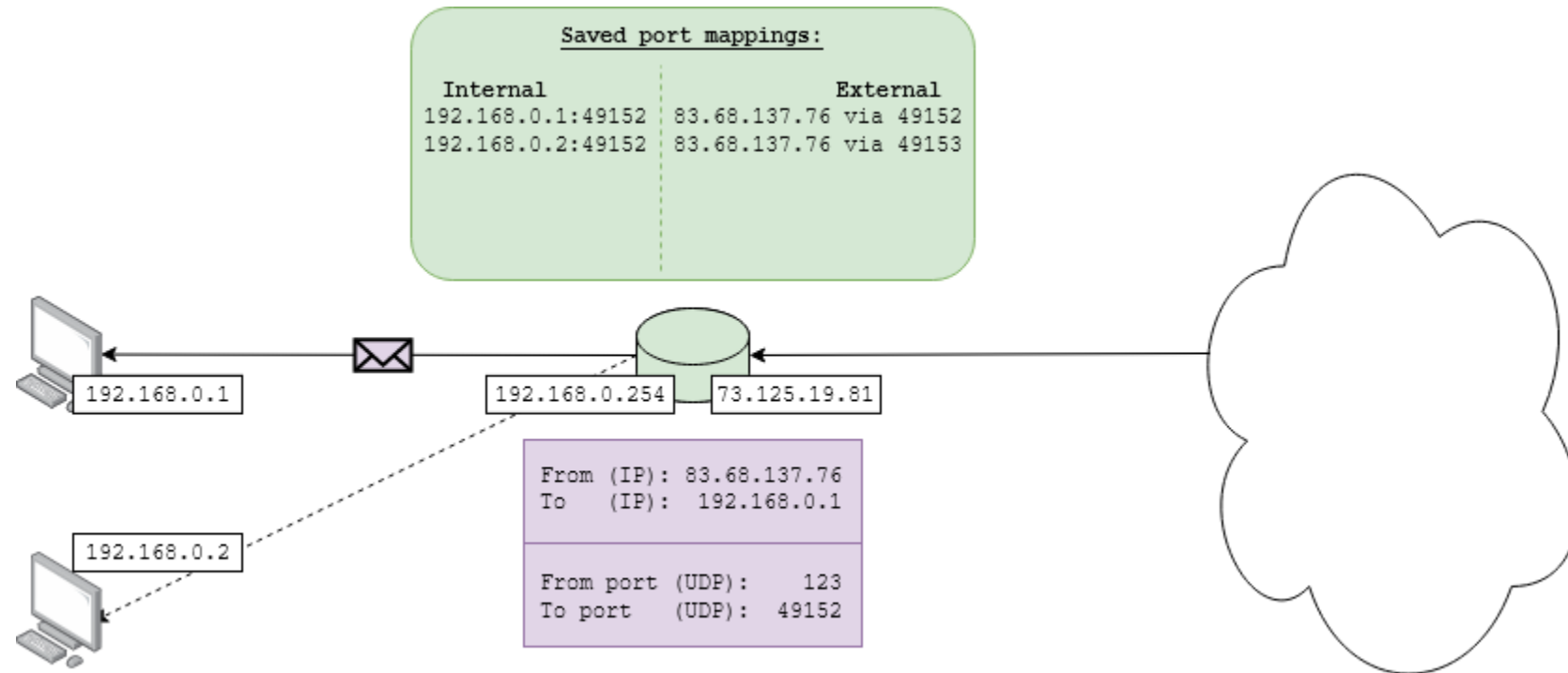
Network Address Translation

- “Hide” an entire private network behind a single public IP
 - Rewrite IP packets at the boundary
 - Rewrite TCP/UDP ports to disambiguate



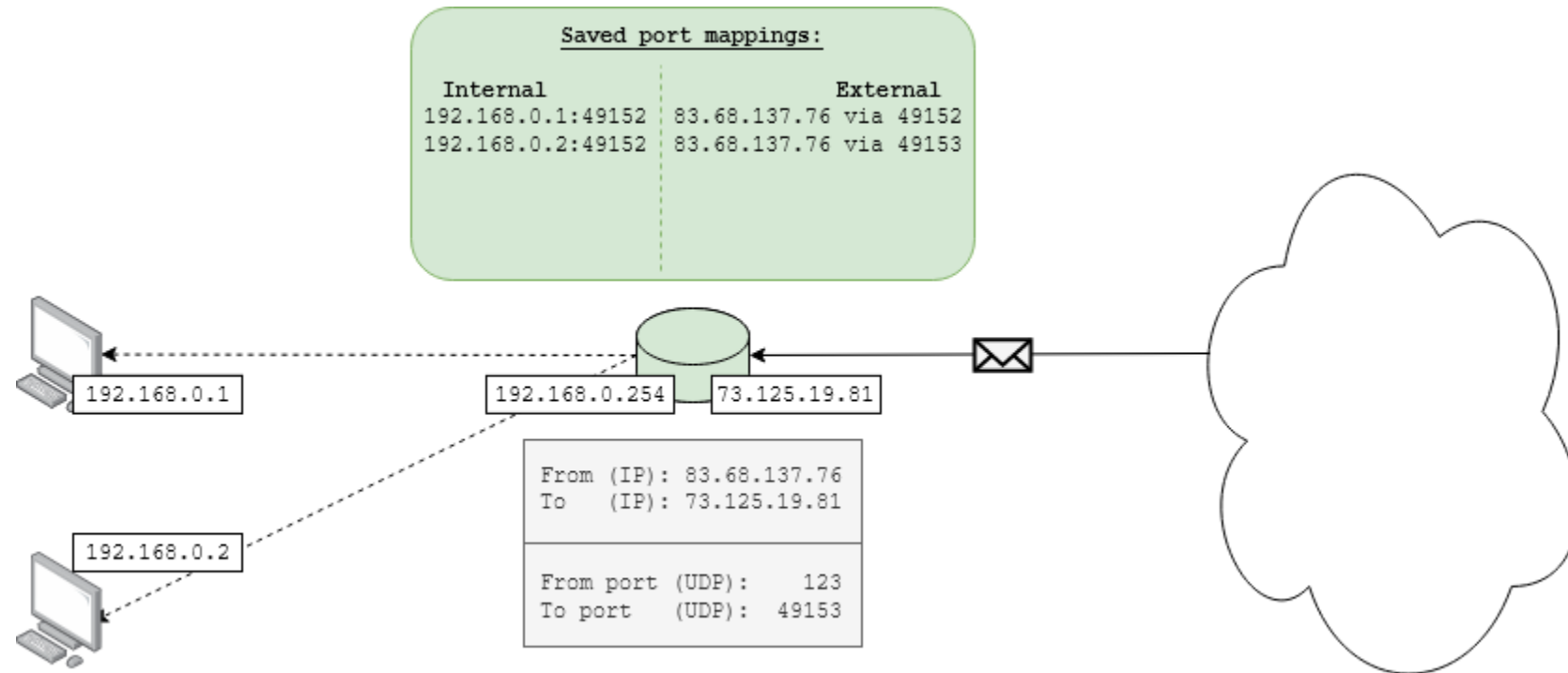
Network Address Translation

- “Hide” an entire private network behind a single public IP
 - Rewrite IP packets at the boundary
 - Rewrite TCP/UDP ports to disambiguate



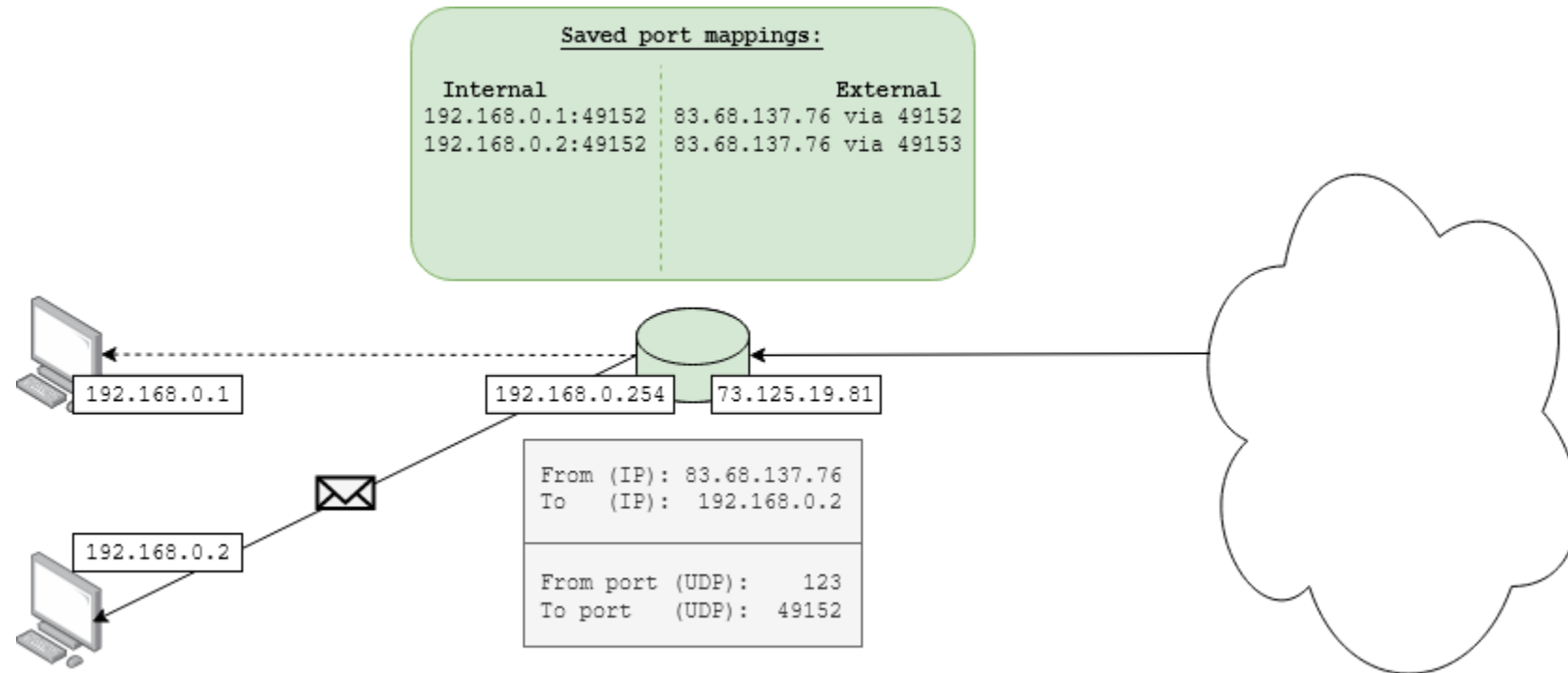
Network Address Translation

- “Hide” an entire private network behind a single public IP
 - Rewrite IP packets at the boundary
 - Rewrite TCP/UDP ports to disambiguate



Network Address Translation

- “Hide” an entire private network behind a single public IP
 - Rewrite IP packets at the boundary
 - Rewrite TCP/UDP ports to disambiguate



Network Address Translation

- “Hide” an entire private network behind a single public IP
 - Rewrite IP packets at the boundary
 - Rewrite TCP/UDP ports to disambiguate
- Transparent if a client “inside” connects to a server “outside”
 - The reverse will not work (by default)
- You can have PAT networks nested within PAT networks
 - Entire ISPs can connect all their clients using one publicly-routable IP address!
- Your home ISP router almost definitely does this!
 - Compare your **ipconfig/ifconfig** address with “what’s my ip” (google)

IPv4 fragmentation – Issues

- 16-bit packet ID is insufficient for high transmission rates
 - 16 bit packet ID \triangleq 65536 packets “in flight”
 - No acknowledgments \Rightarrow ID can't be reused until TTL expires
 - 65536 packets \div 128 seconds = 512 packets per second
- Also: other issues

IPv4 fragmentation – Issues

- Fragmentation splits TCP/UDP transmission units across IP packets
 - Only the first fragment has the transport layer header!
 - If it even does...? There's no minimum fragment size...
 - Which host should a PAT firewall forward the data to?
 - Firewalls can't effectively filter these packets either...
- Reassembly is very fragile
 - How long should fragments be kept around for?
 - Denial-of-service attacks!
 - How do you handle overlapping fragments?
 - They are valid as per the protocol spec...

Recap

- **Data Link Layer**: send data to locally connected devices
 - Ethernet, Wi-Fi, Bluetooth, ...
- **Network Layer**: send data to devices over the internet
 - IPv4, IPv6, ...
- **Transport Layer**: structure the data into individual connections
 - TCP, UDP, ...
- What's left?
 - Actually send useful data!