Networking music and computations

From MIDI to OSC and UDP to HPC

Master ATIAM - Informatique
Philippe Esling (esling@ircam.fr)
Maître de conférences – UPMC
Equipe représentations musicales (IRCAM, Paris)

Towards network music

We live in the (strange) future …

- Laptop orchestras (cf. Stanford, Princeton, …)
- Robotic bands and drummers
- Hyper-instruments

But communication / network is still the center of music groups

Towards network music

Towards network music

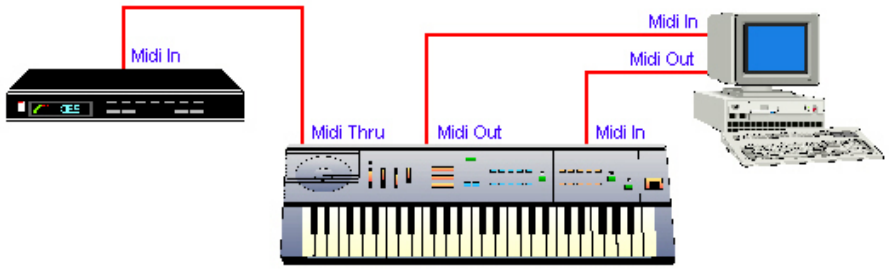
Hey granpa' MIDI

Musical **I**nstrument **D**igital **I**nterface

The MIDI protocol — a "language" that lets synthesizers, computers and other devices talk to each other.

More precisely …

- MIDI doesn't directly describe musical sound
- MIDI is not a language
- It is a data communications protocol



Good ol' days

- **1900s**: First electronic synthesizers developed
- **1970s**: digital synthesizers developed
  - Each manufacturer used different design scheme
  - Synthesizers were monophonic
  - With a single input device, each player can only run single synthesis
  - To use a wide range of synthesized sounds, many players were needed
- 1981: 3 synthesizer companies decided to do something about it.
  - Sequential Circuits, Roland, Oberheim Electronics
- 1982: other companies such as Yamaha, Korg, Kawai joined.
- 1983: full MIDI 1.0 Detailed Specification released

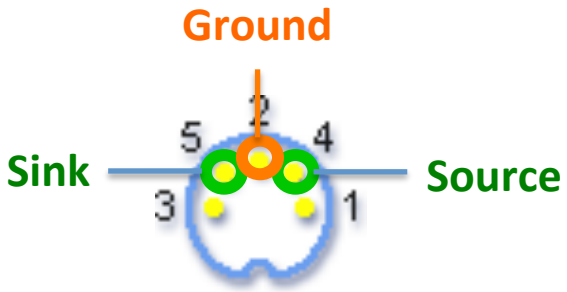
- It standardized the control signal and inter-machine communication between synthesizer devices

MIDI Ports and transmission

- MIDI ports use a five-pin DIN connector
  - Inexpensive and readily available
  - Only 3 pins among 5 are used until now
  - Both ends of MIDI line are the same.

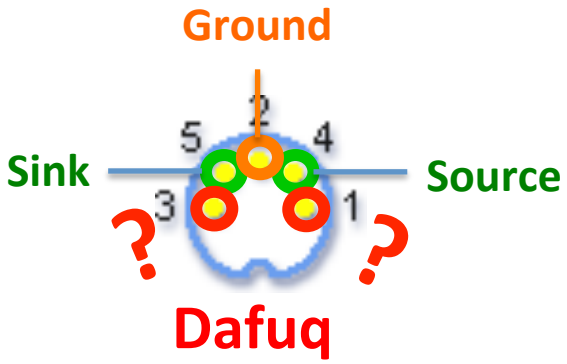
MIDI Ports and transmission

- MIDI ports use a five-pin DIN connector
  - Inexpensive and readily available
  - Only 3 pins among 5 are used until now
  - Both ends of MIDI line are the same.



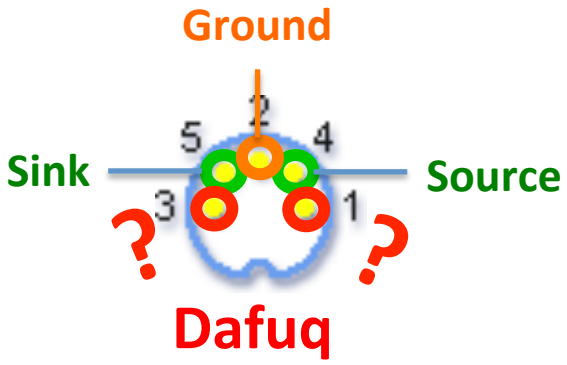
MIDI Ports and transmission

- MIDI ports use a five-pin DIN connector
  - Inexpensive and readily available
  - Only 3 pins among 5 are used until now
  - Both ends of MIDI line are the same.

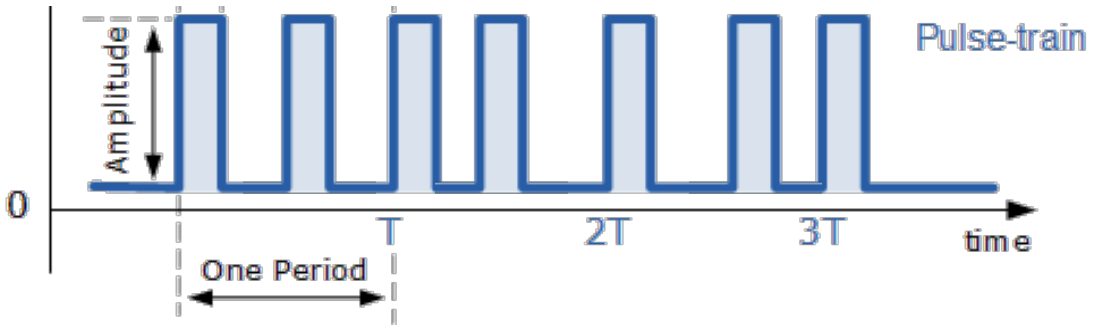


MIDI Ports and transmission

- MIDI ports use a five-pin DIN connector
  - Inexpensive and readily available
  - Only 3 pins among 5 are used until now
  - Both ends of MIDI line are the same.



- Serial transfer, data are sent bit by bit
  - Transmission rate is slow at only 31,250 bits/sec
  - Too slow to transmit samples in real-time
  - Have to do off-line sample dump



MIDI Architecture

- Consists of channels, messages and programs mapped to ‘voices’
- A voice is a sound featured in a sound module
- MIDI can be considered to have three distinct **layers**
  - **The device layer** - hardware, cables, foot switches
  - **The communication layer** - channels and messages
  - **The music layer** - actual voices and sound quality of the voices
- MIDI messages transmitted only on channels 1..16 (so only 16 devices)
- MIDI can play several sounds ‘voices’ on a sound module where the sound module is **multi-timbral** - able to play more than one sound

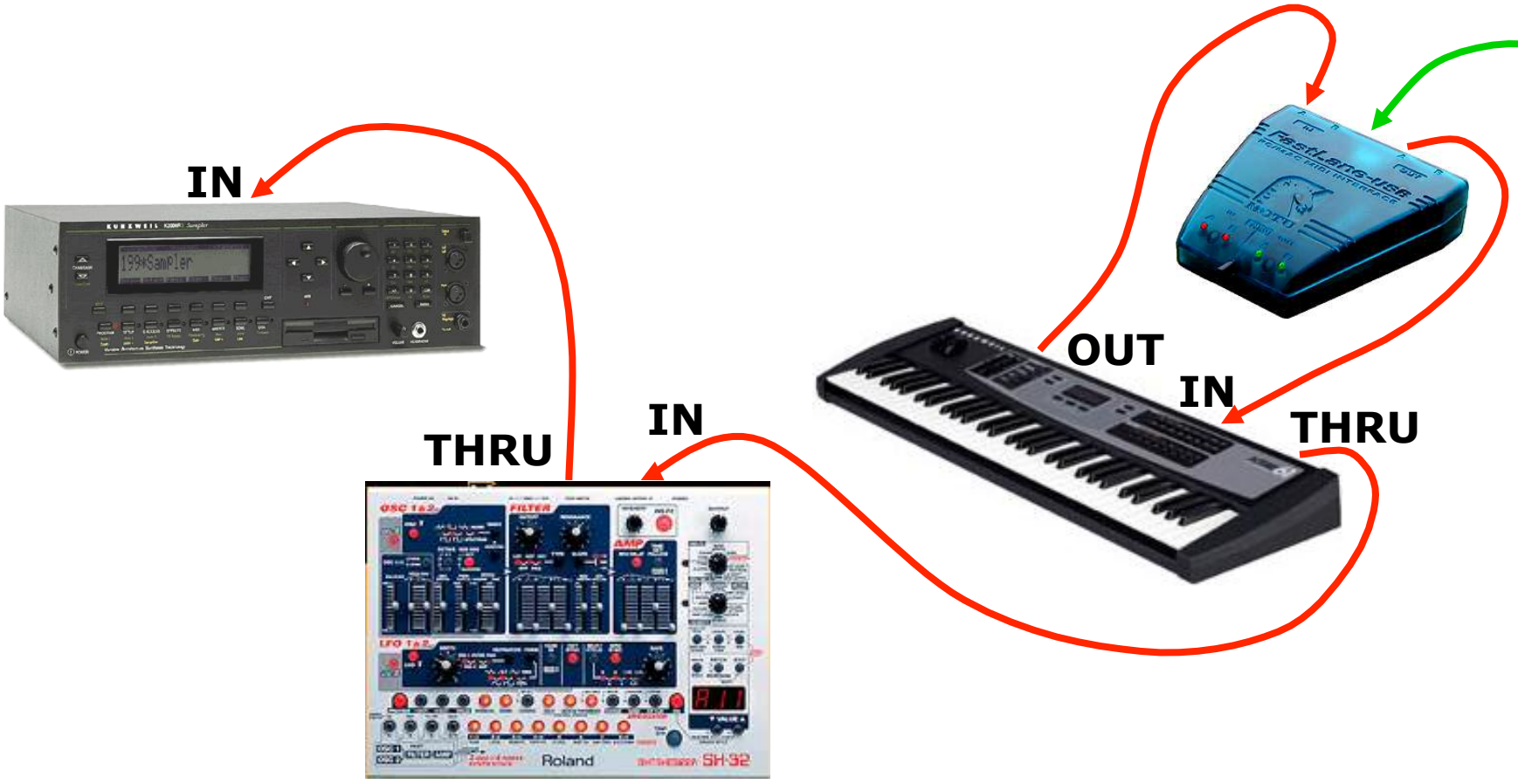
MIDI In : data enters each item through the MIDI In port.

MIDI Out : data generated are sent out through the MIDI Out port.

MIDI Thru : Used to re-transmit all information received at the MIDI In.

- Often these ports are used to create a chain of connected devices in a single MIDI data path, called a 'daisy chain'.

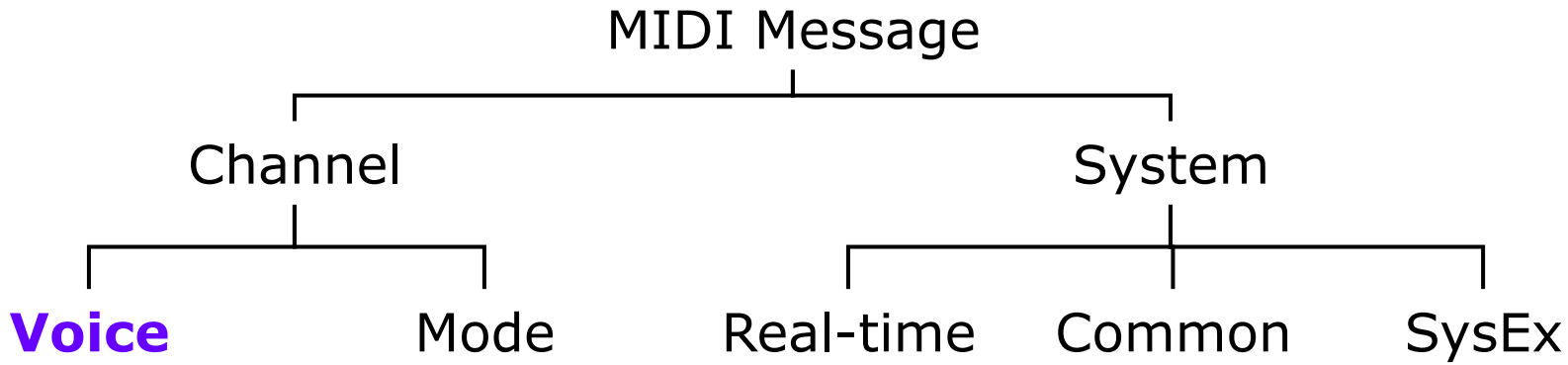
MIDI Daisy-chain network



The 3 devices must **share 16 channels**.

MIDI Messages

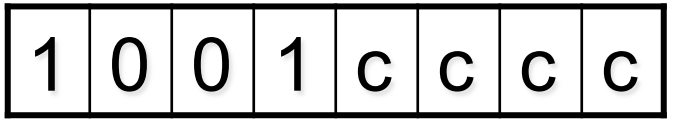
- **Pre-determined set of messages**
- Note on / Note off
  - Pitches (**only 127 possible and only semitones**) and velocity
- Pedal and foot switch information
- Program change (patch change)
- Note/pitch bend
- Polyphonic pressure (aftertouch)
- Monophonic pressure (aftertouch)
- System Common Messages (same data to number of devices)
- System Exclusive Messages (manufacturer specific messages)



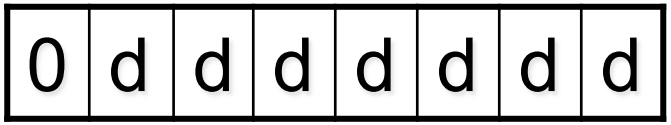
MIDI Transmission protocol

- Each message begin with ONE start bit (logical 0)
- Then followed by EIGHT message bits
- End with ONE stop bit (logical 1)

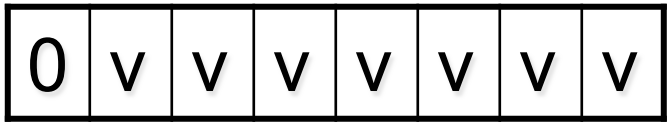
  - 1st byte: Status byte
    - 1001 means **note on**
    - cccc is the binary representation of the message channel



  - 2nd byte: Pitch Data byte
    - 0 means "it is a data byte"
    - ddddddd is the binary representation of the pitch. (decimal 0-127).



  - 3rd byte : Velocity byte
    - vvvvvvv is the value of velocity



Other MIDI messages examples

Program Change

- Assign particular patch (instrument) to a channel
  - 1st byte: Status byte→ 1100cccc
  - 2nd byte: program number data byte→ 0ddddddd

Control Change

- Assigns some effect to the sound in the channel
  - 1st byte: Status byte→ 1011cccc
  - 2nd byte: control change type → 0ddddddd
  - 3rd/4th byte: control change value→ 0ddddddd

Pitch Bend

  - 1st byte: Status byte → 1110cccc
  - 2nd byte: pitch bend value (least significant 7 bits) → 0ddddddd
  - 3nd byte: pitch bend value (most significant 7 bits) → 0ddddddd

And the set of system messages

Timing clocks messages

System reset messages (11111111)

General MIDI (GM)

- An agreed set of protocols manufactures must implement
- The mapping of program numbers to voices (patches)
- Establishes 128 (0..127) mappings
  - program number
  - voice type
  - (e.g. program 0 = acoustic piano)
- The GM percussion map :
  - implements a standard drum kit
  - mapped to keys ranging from B0 through to A4

MIDI Hardwares

a. Pure Musical Input Devices

- Most common: Keyboard
- Usual features : Polyphony and Touch response
- A lot of MIDI drums out there as well …
- But also weirder : Saxophones, flutes, etc …

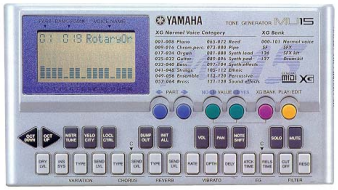
b. Controllers

- Numbered controllers (set of values)
- Continuous Controllers (CC values)
- On/Off controllers [e.g. foot pedal (sustain pedal)]
- Universal MIDI controllers (can control any event)

c. Synthesizer

- Generates sound from scratch
- Wavetable, FM, additive, granular, etc… synthesis.

d. Sequencer
e. MIDI interface
f. sound sampler
[…]



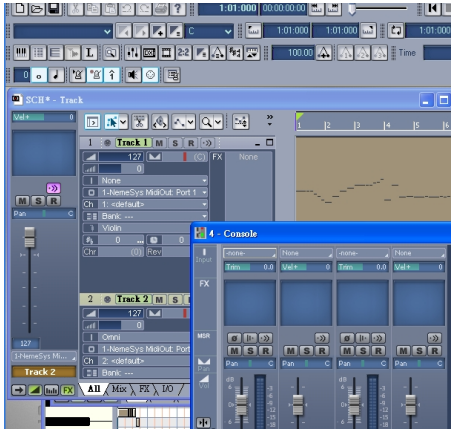
MIDI Softwares

a. Software samplers
- Kontakt, Gigastudio, etc…

b. Production studios
- Ableton live, Logic, Reason, Fruity loops …

c. Recording softwares
- Cakewalk, Cubase, Protools

d. Score editors
- Finale, ouverture, …



Limitations of MIDI

1. Slow -- Serial transfer
- If too much continuous data transfer (e.g. a lot of control data) = MIDI choke
  - Can be solved by EVENT FILTERING (discard less important messages)

2. Slow – MIDI only controls information, time is needed to synthesize the sound
- Too much computation time = MIDI lag
  - Solution: users have to avoid using patch (instrument) which uses a lot of memory (snif)
  - e.g. Cymbal in channel 10 of Nokia Cellular phone

3. Sound quality varies
- Depends on the synthesizer you use

  Solution: users have to judge by ear, to see which sound is good (+ General MIDI (GM))

  Advantage ?

- The size of MIDI file is very small!
  - 3 minutes WAV file, 48kHz, stereo = 40Mb
  - 3 minutes MIDI file, with 10 channels = 40Kb

Going beyond MIDI

- MIDI is limited in features, message type, speed, data size
- … In fact almost everything !
- (But it offers the power of being a standard …
- … so everybody speaks the same language)
- A lot of nowadays interactions **require lots of data**
- And also a lot of **different ways to interact** between mediums
- So why not using modern technology for communication
- Something that can rely on **modern network technology**
- Lots of work has been performed for **internet protocols**
- Open Sound Control (OSC) provides an answer !
- But to get there …

The biggest question

The biggest question

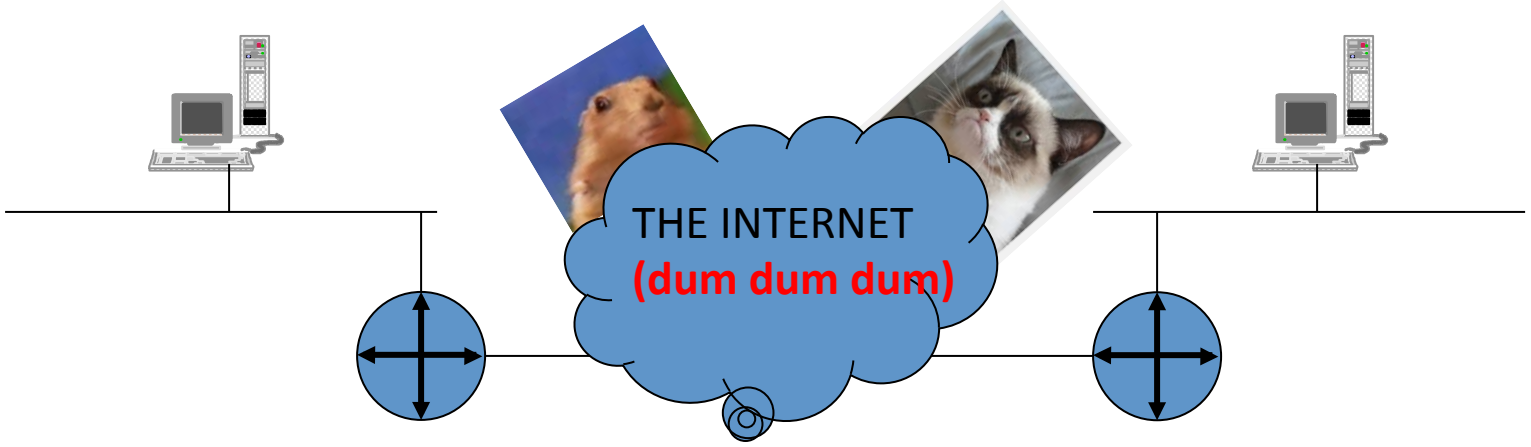
WHAT IS THE INTERNET ?

The biggest question

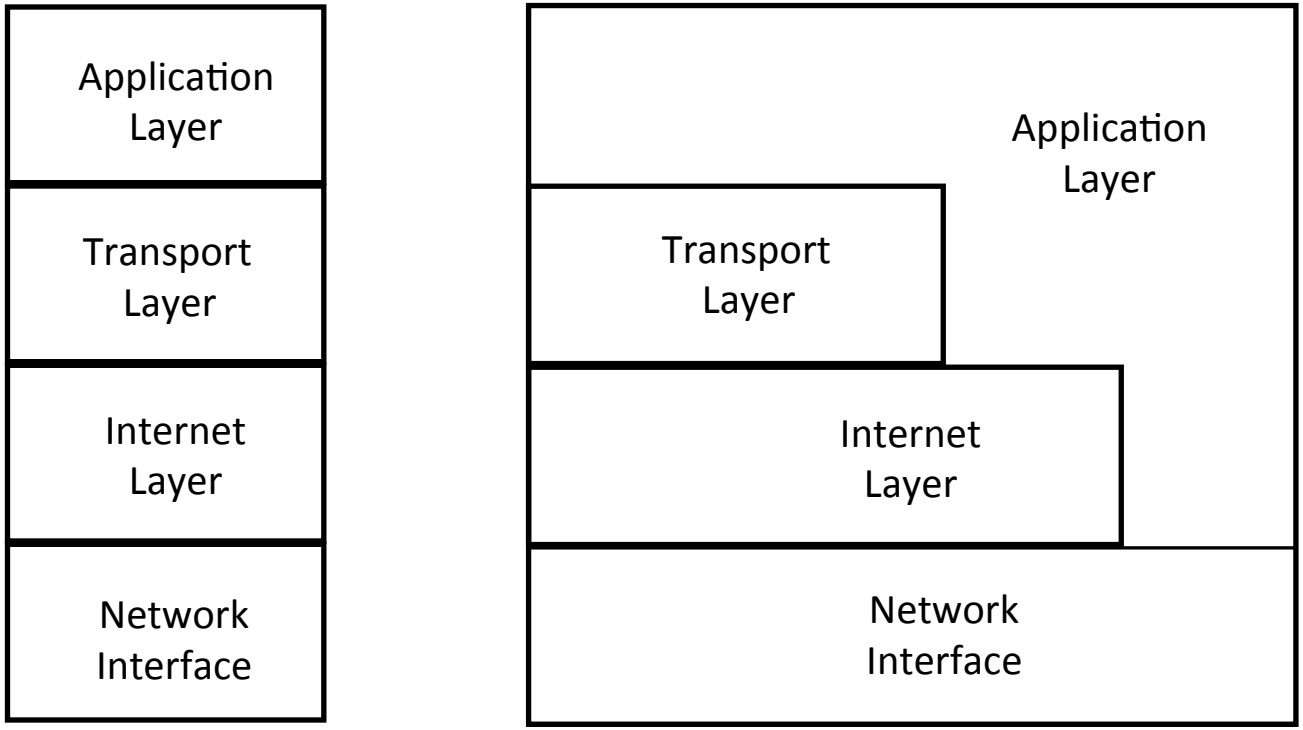
WHAT IS THE INTERNET ?

What is the internet ?

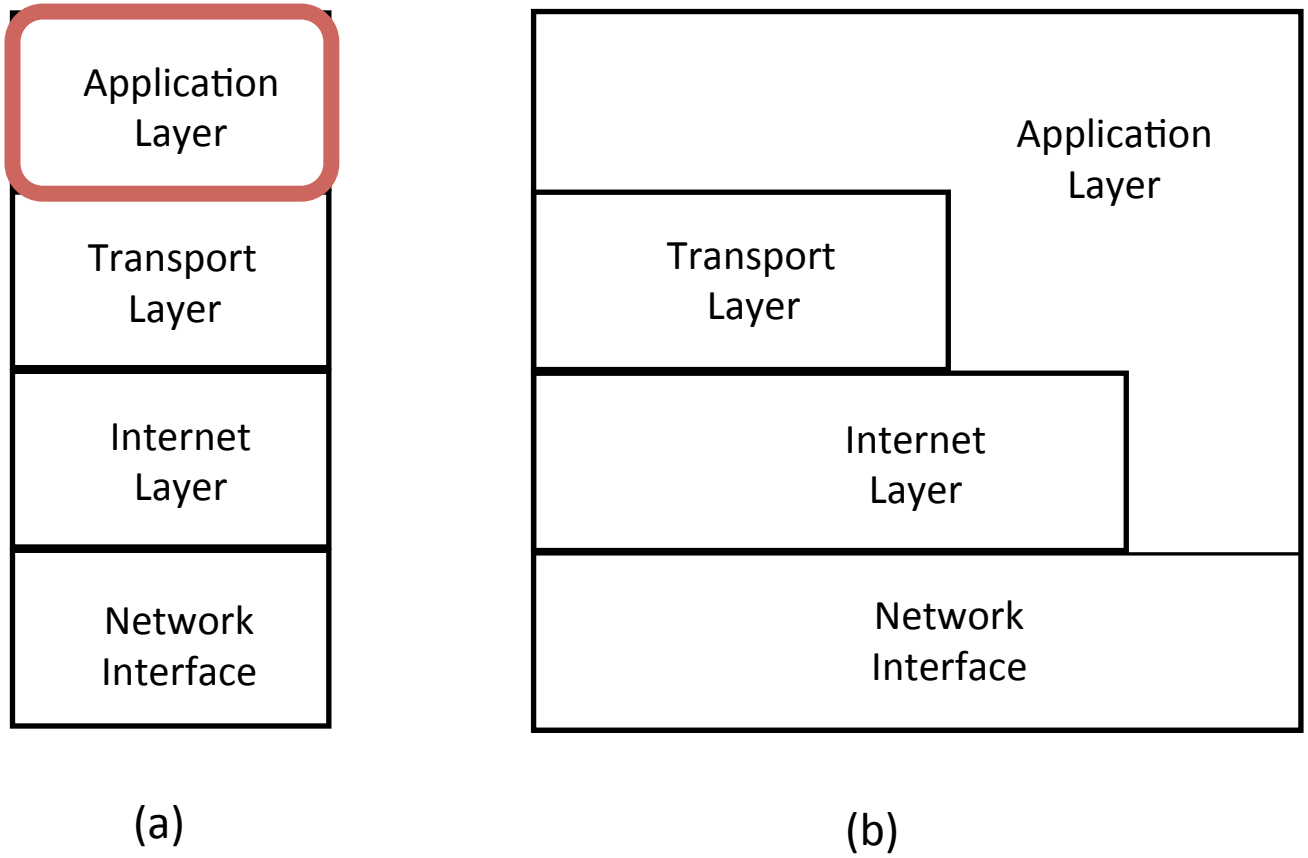
- A set of *inter*connected *net*works
- The **I**nternet is the most famous example
- Networks can be completely different
  - Ethernet, ATM, modem, …
  - TCP/IP is what links them
- *Routers* are devices on multiple networks that pass traffic between them
- Individual networks pass traffic from one router or endpoint to another
- **TCP/IP hides the details** as much as possible



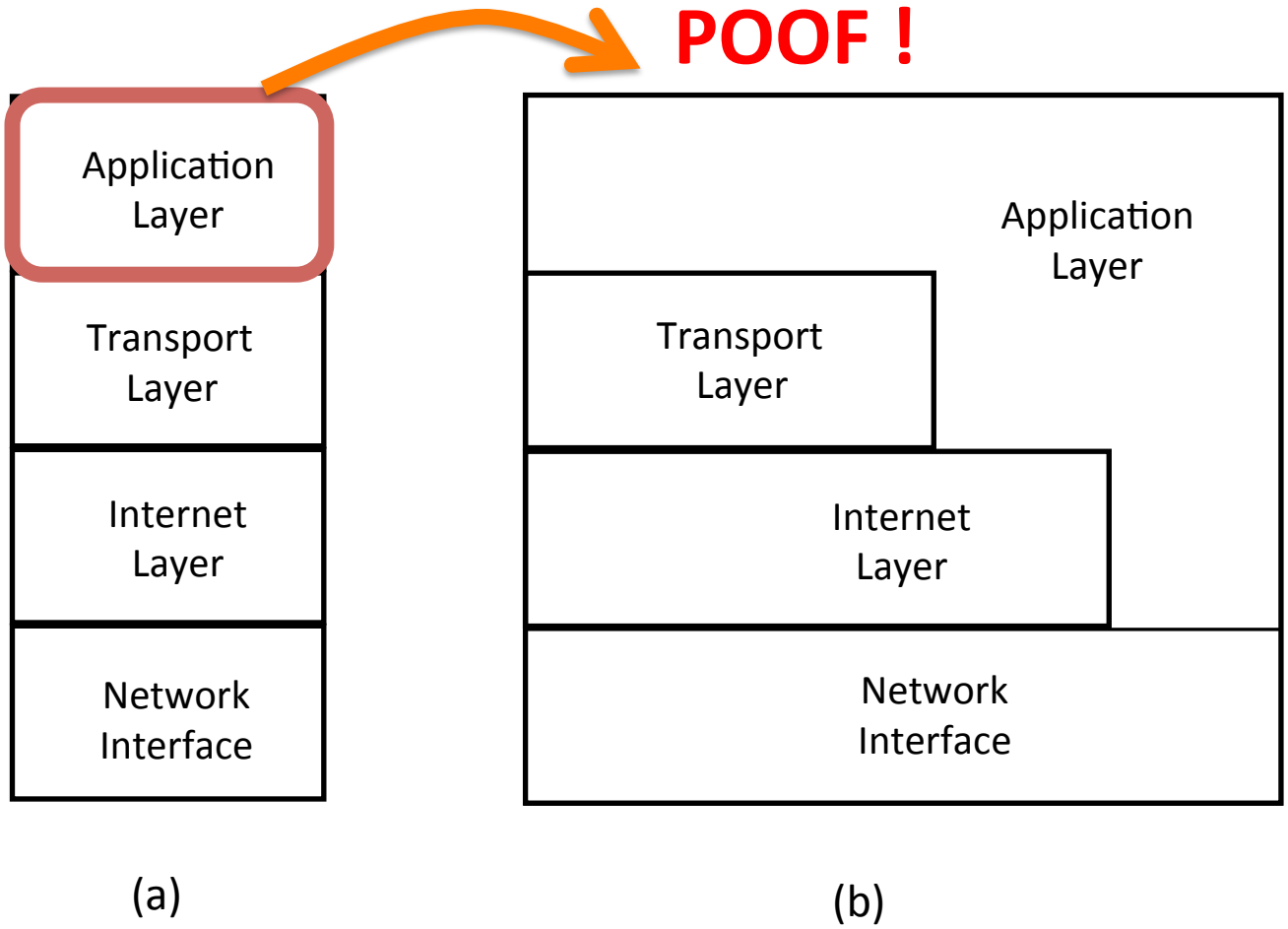
What is TCP/IP hiding ?



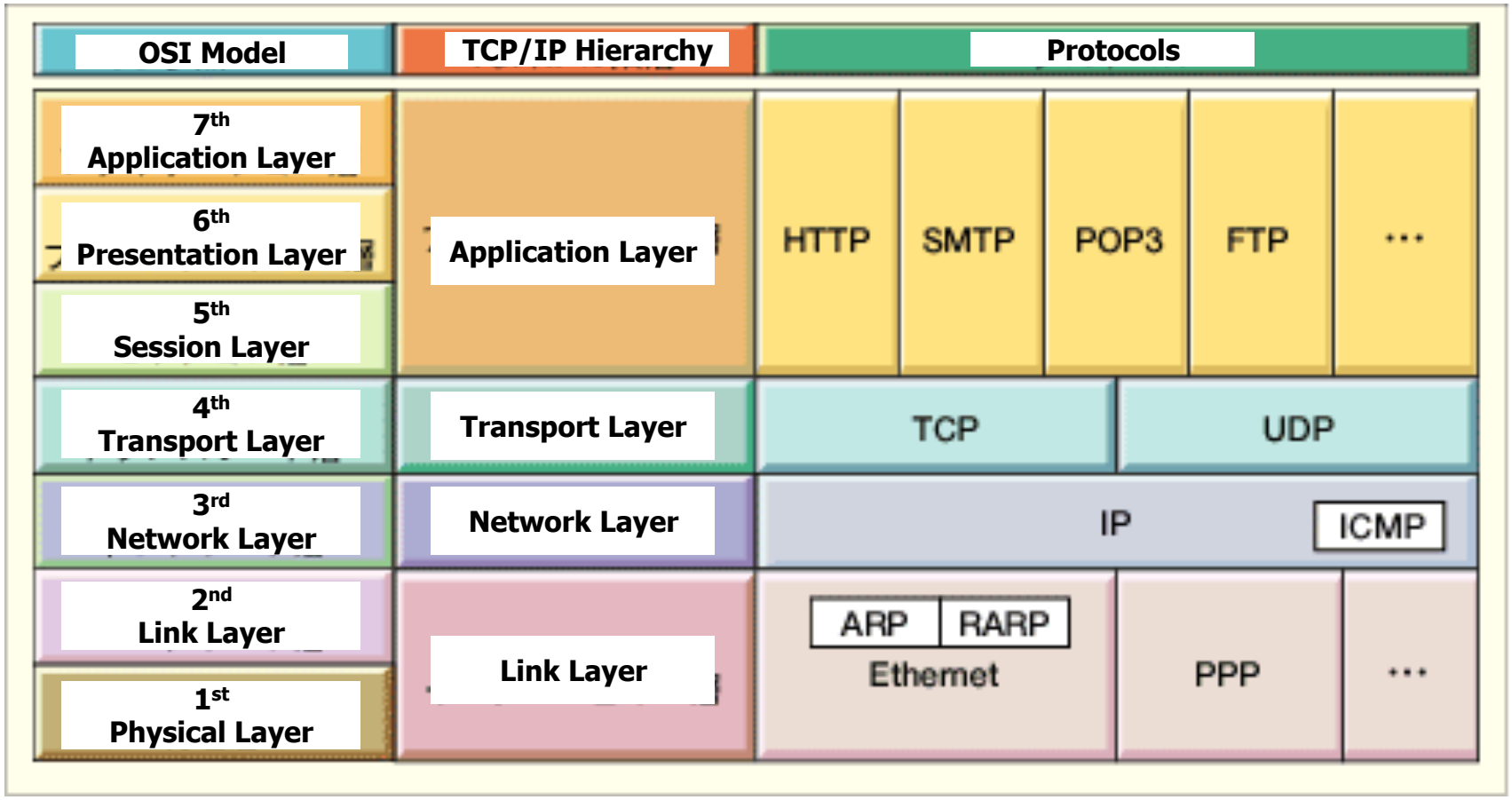
What is TCP/IP hiding ?



What is TCP/IP hiding ?



What is TCP/IP hiding ?



What is TCP/IP hiding ?

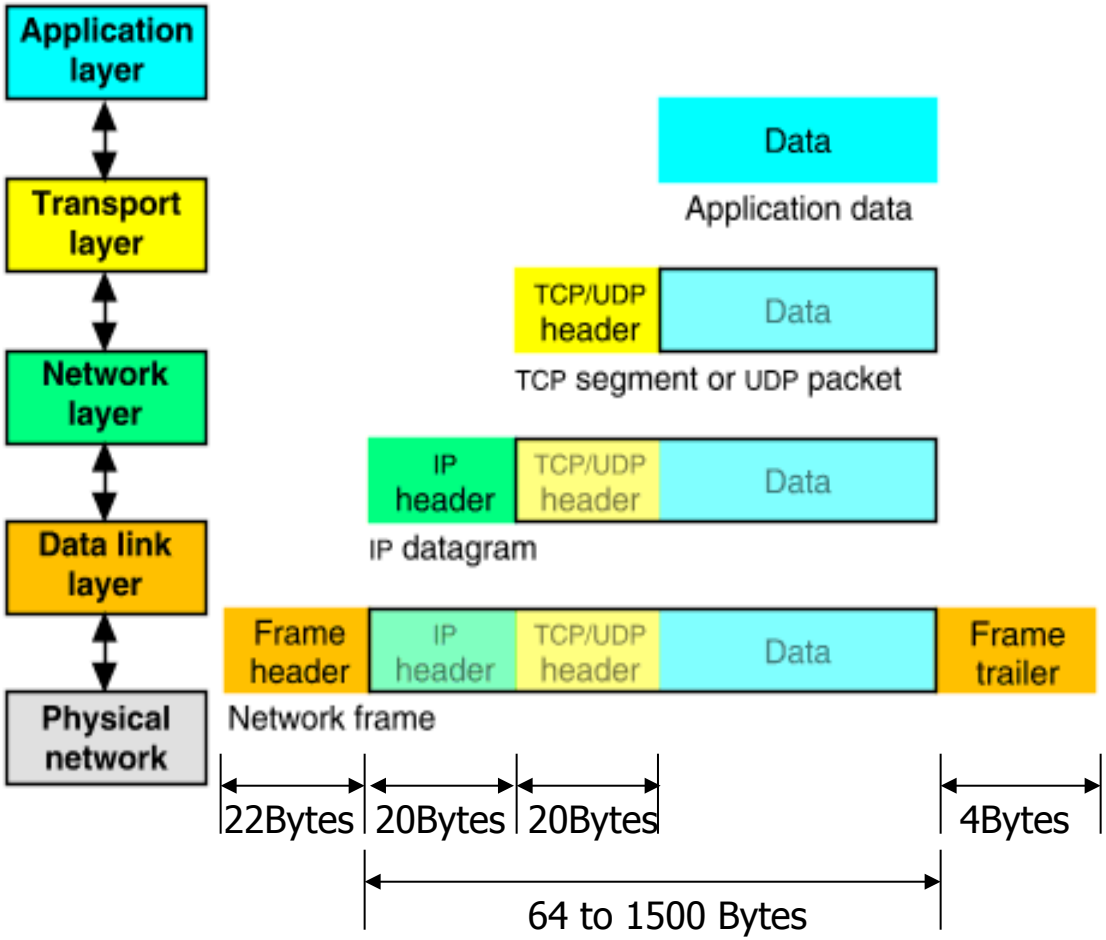
- Seven network “layers”
  - Layer 1 : Physical – cables
  - Layer 2 : Data Link – ethernet
  - Layer 3 : Network – IP
  - Layer 4 : Transport – TCP/UDP
  - Layer 5 : Session
  - Layer 6 : Presentation
  - Layer 7 : Application

TCP/IP Protocol family

- IP : Internet Protocol
  - UDP : User Datagram Protocol
    - RTP, traceroute
  - TCP : Transmission Control Protocol
    - HTTP, FTP, ssh
- Different view – 4 layers
  - Layer 1 : Link
  - Layer 2 : Network
  - Layer 3 : Transport
  - Layer 4 : Application

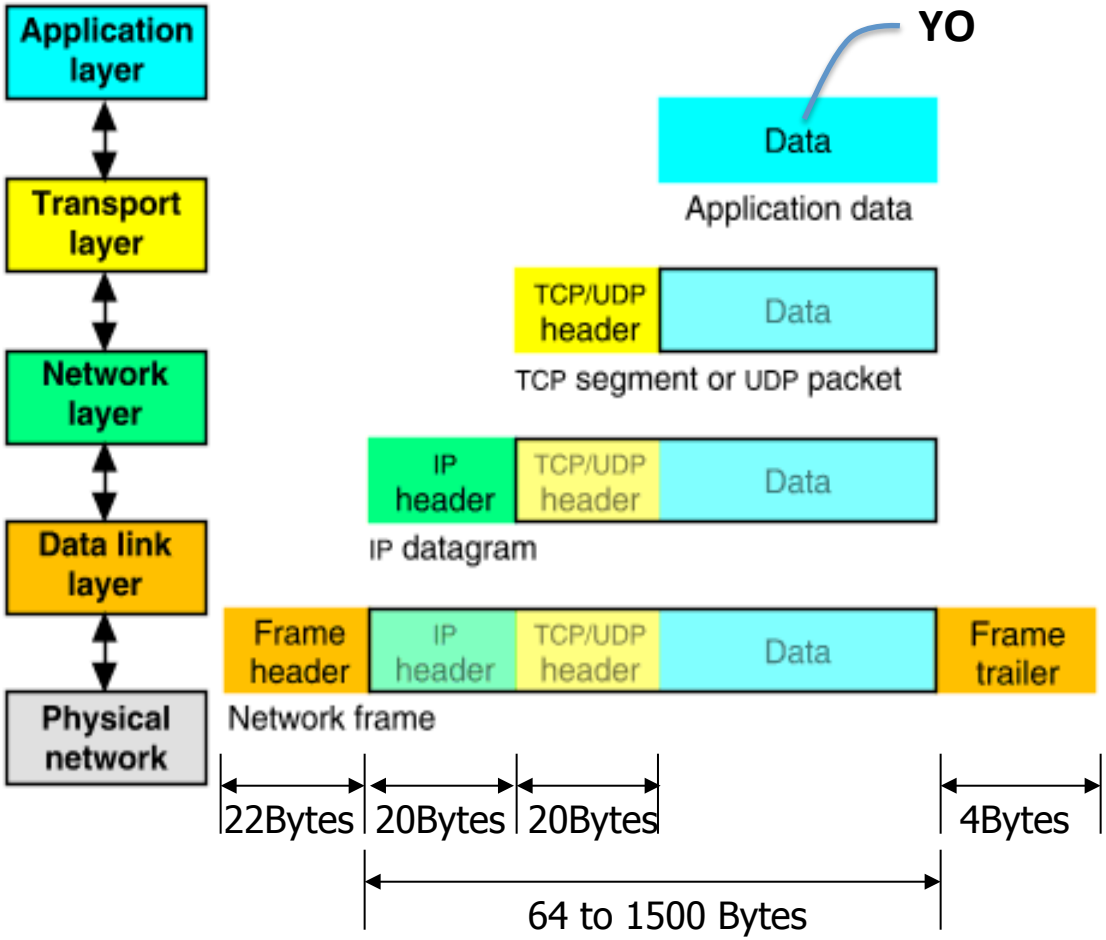
Packet encapsulation

- The data is sent down the protocol stack
- Each layer adds to the data by prepending headers

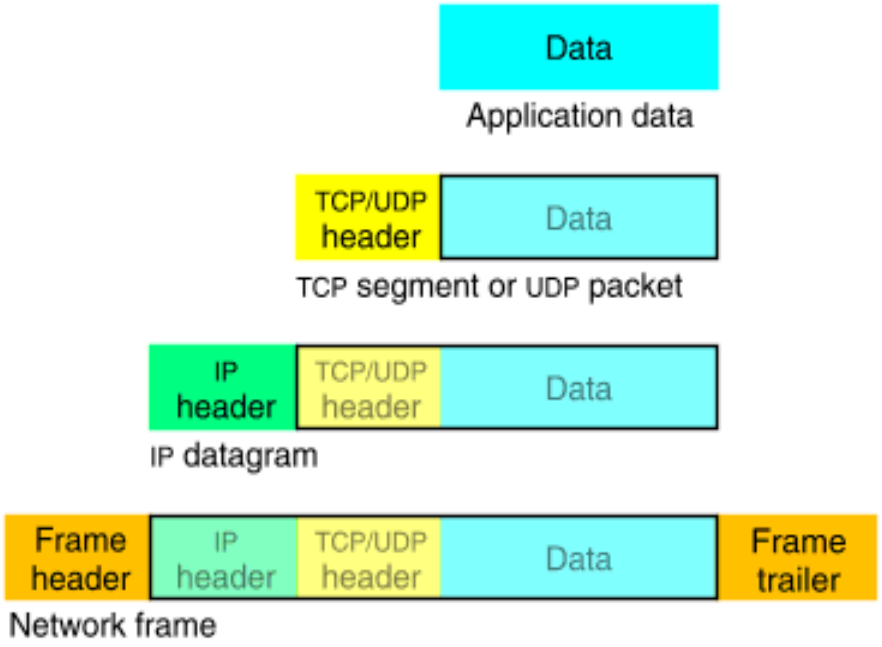


Packet encapsulation

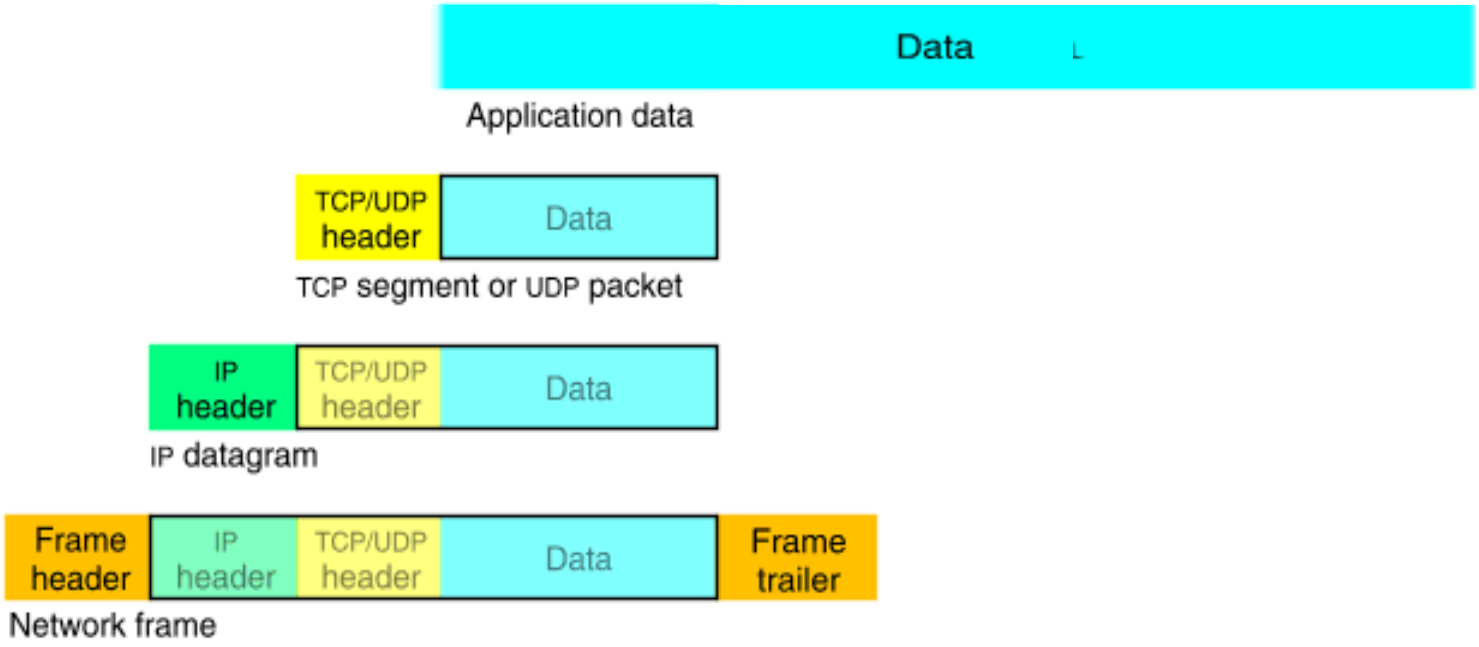
- The data is sent down the protocol stack
- Each layer adds to the data by prepending headers



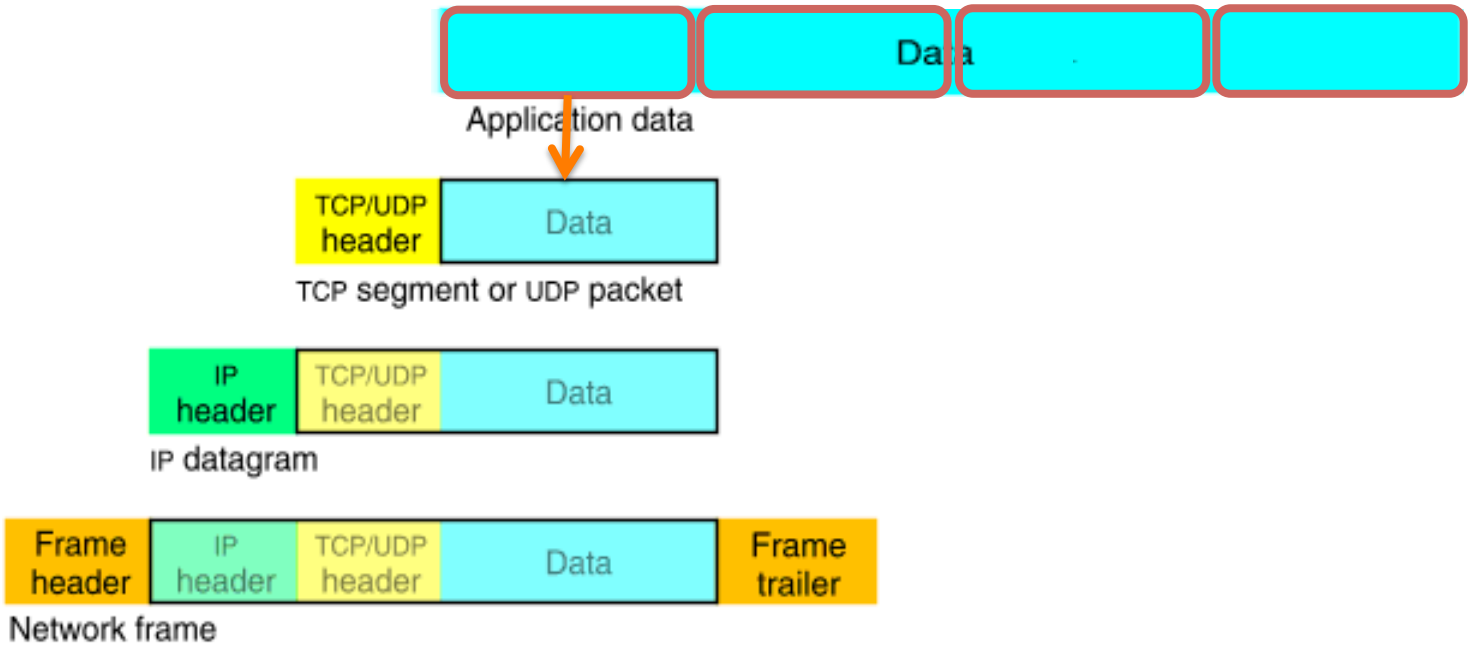
Packet encapsulation



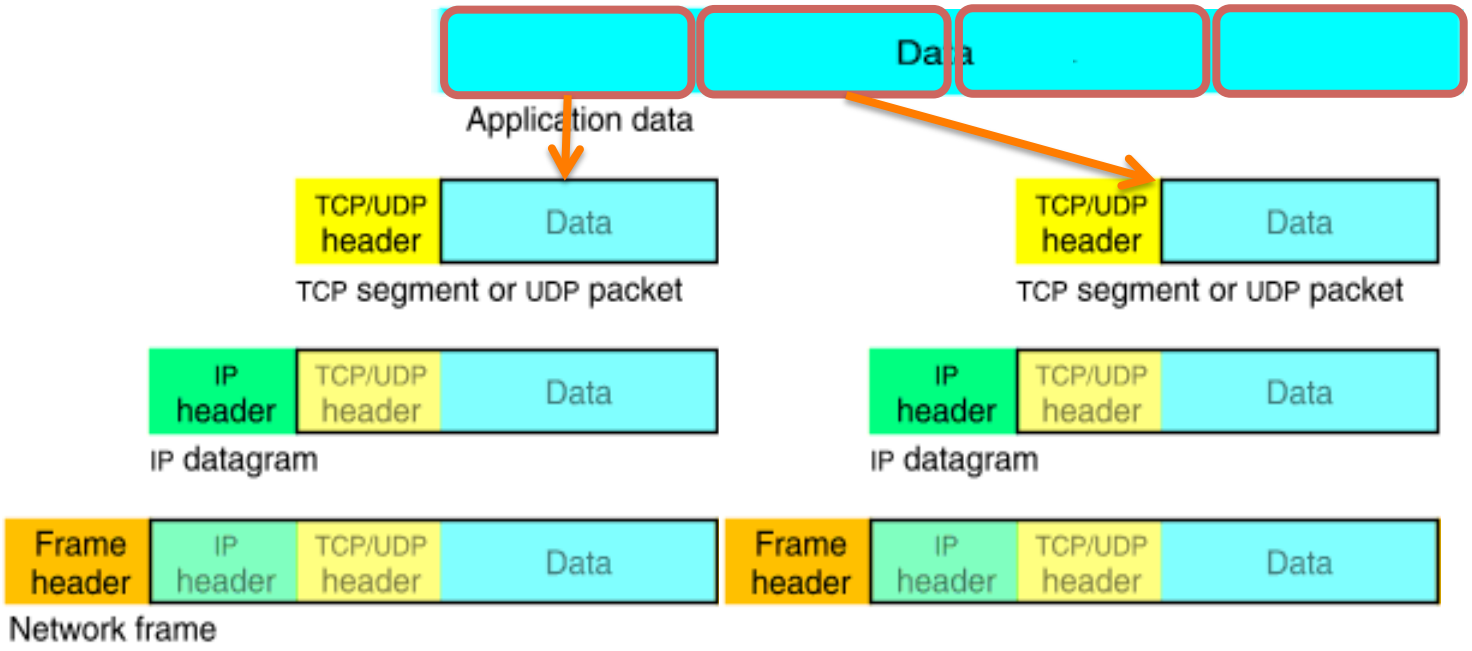
Packet encapsulation



Packet **fragmentation**



Packet **fragmentation**



Under TCP = IP (**Internet Protocol**)

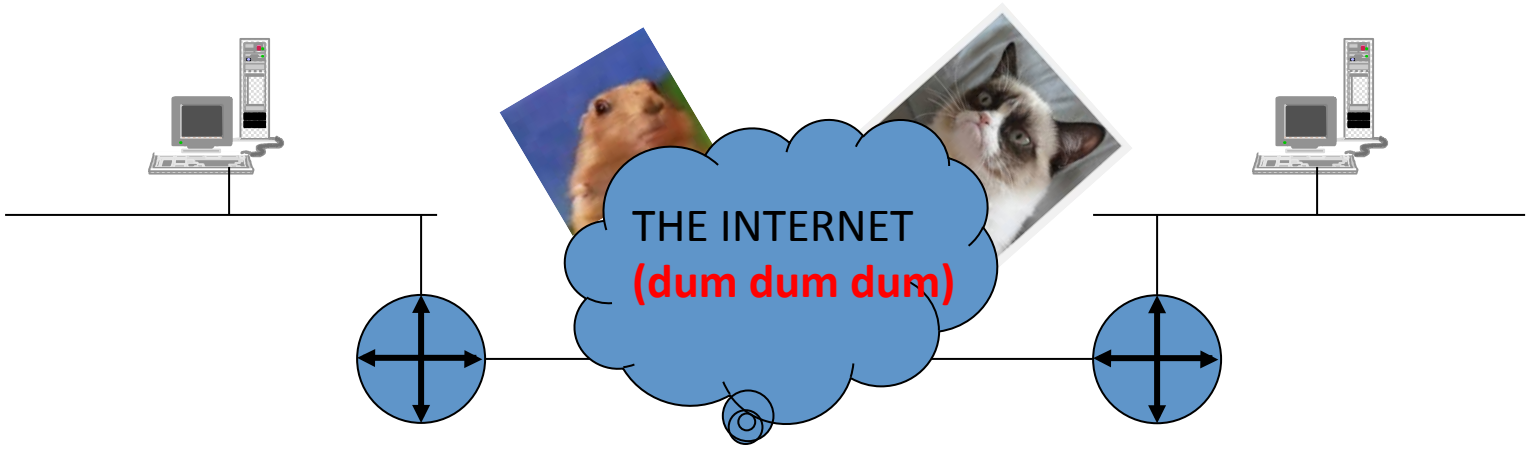
- Responsible for end to end transmission
- Sends data in individual packets
- Maximum size of packet is determined by the networks
  - Fragmented if too large
- Unreliable
  - Packets **might be lost, corrupted, duplicated, out of order**
- IP Addresses = 4 bytes
  - e.g. 163.1.125.
  - Each device normally gets one (or more)
  - In theory there are about 4 billion available

Under IP = Routing !

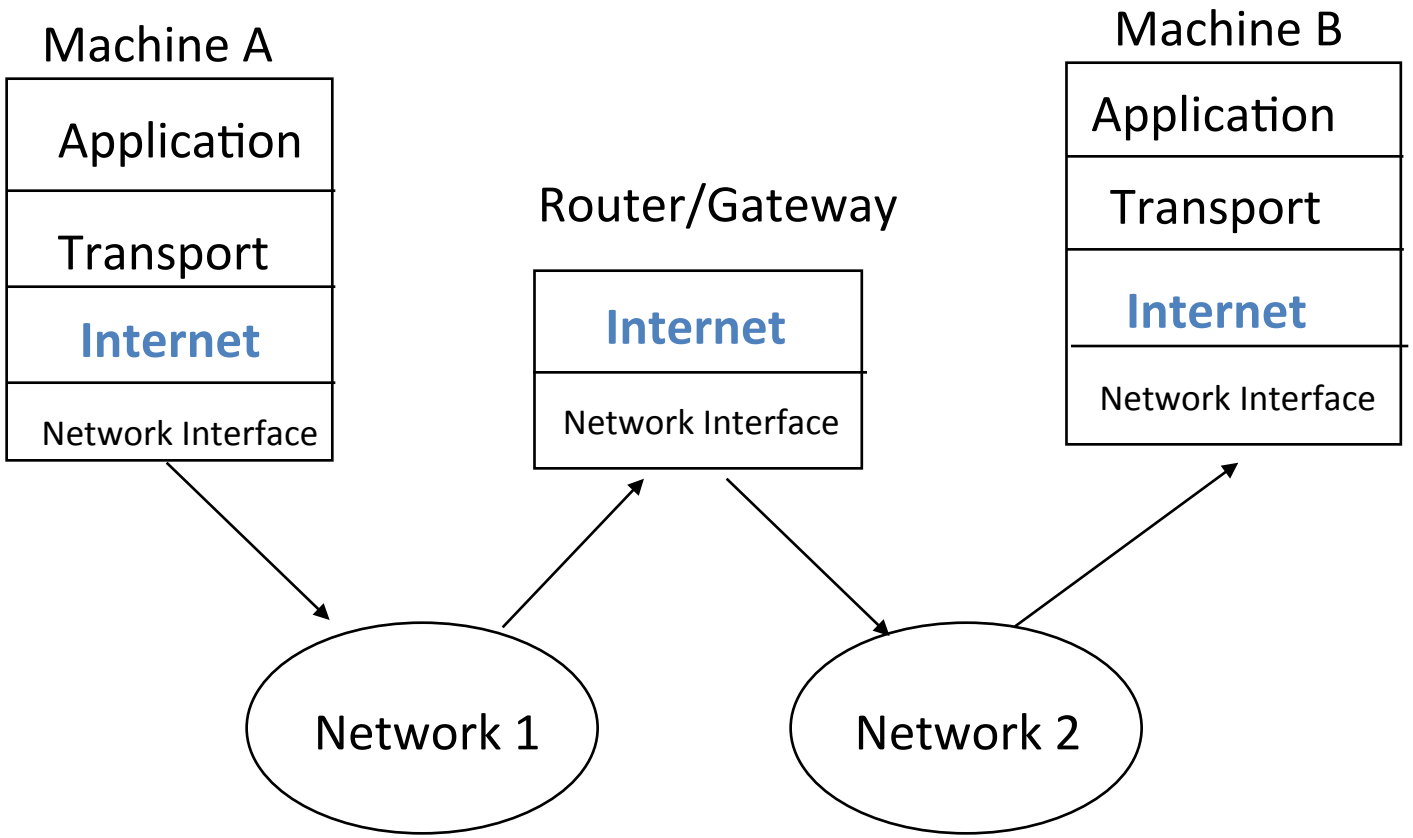
- How does a device know where to send a packet?
  - Need to know IP addresses **on directly attached networks**
  - If destination is on a local network, send it directly there
- If the destination address isn't local
  - **Send everything to a single local router**
  - Routers need to know which network **corresponds to each possible** IP address

Under IP = Routing !

- How does a device know where to send a packet?
  - Need to know IP addresses **on directly attached networks**
  - If destination is on a local network, send it directly there
- If the destination address isn't local
  - **Send everything to a single local router**
  - Routers need to know which network **corresponds to each possible** IP address

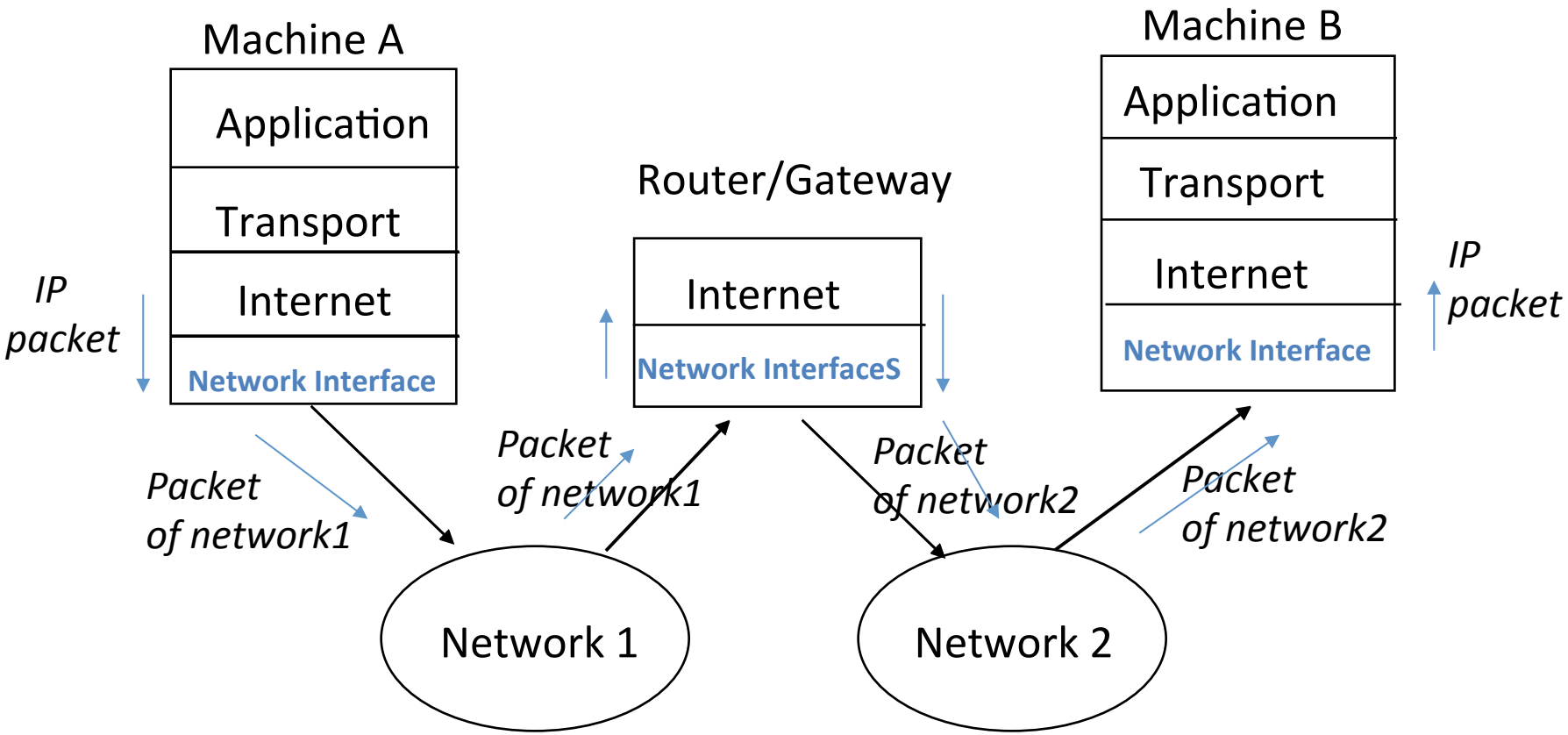


Routing : **single-point** example



1. Transfer of information across networks through gateways/routers
2. Corresponding to OSI network layer: routing and congestion control

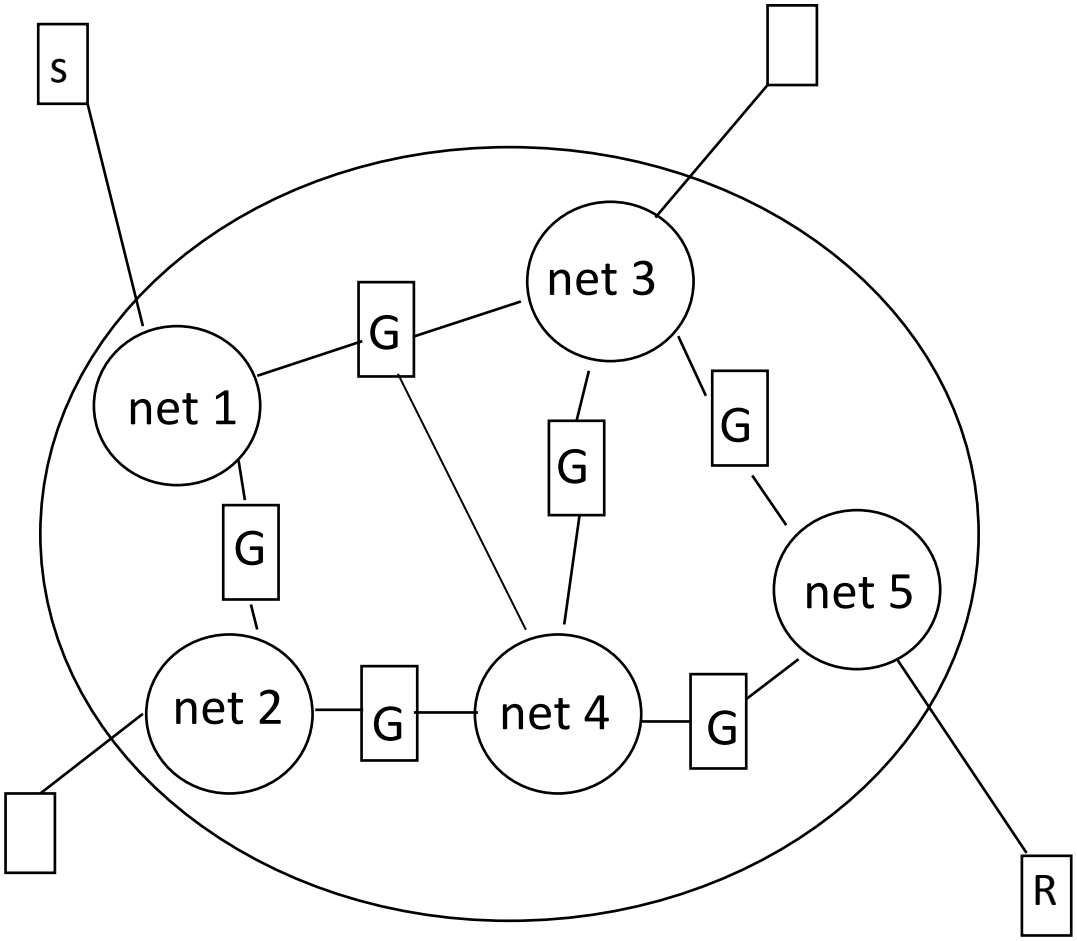
Routing : **single-point** example



1. Transfer of information across networks through gateways/routers
2. Corresponding to OSI network layer: routing and congestion control

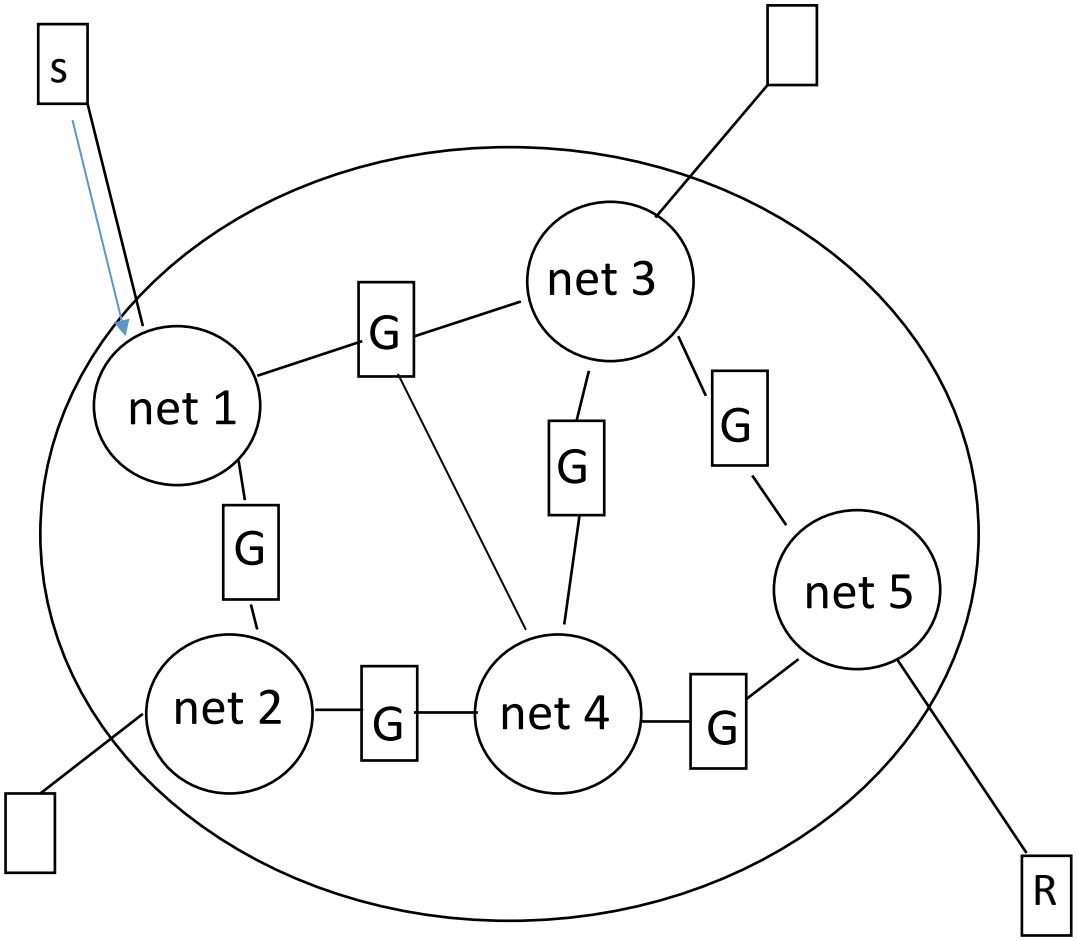
Routing : **generalization**

S sends a packet to R:



Routing : **generalization**

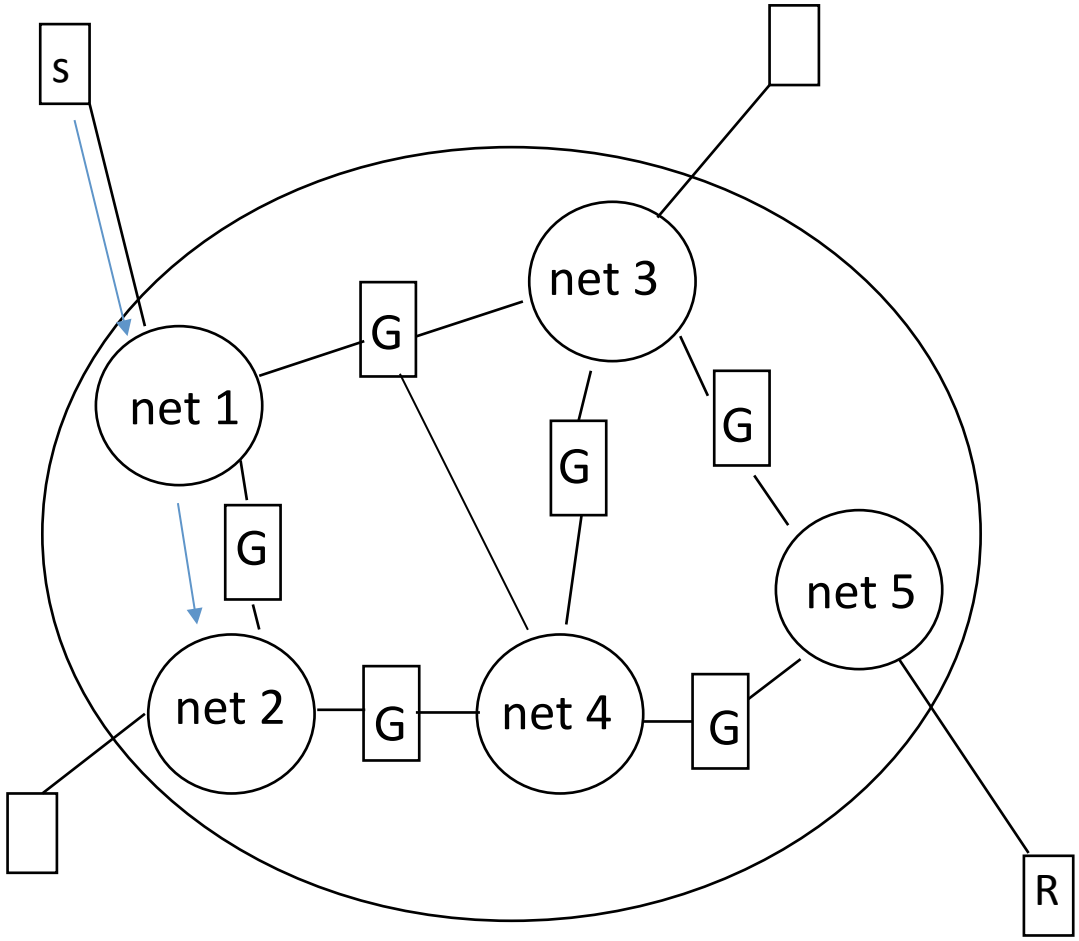
S sends a packet to R:



Routing : **generalization**

S sends a packet to R:

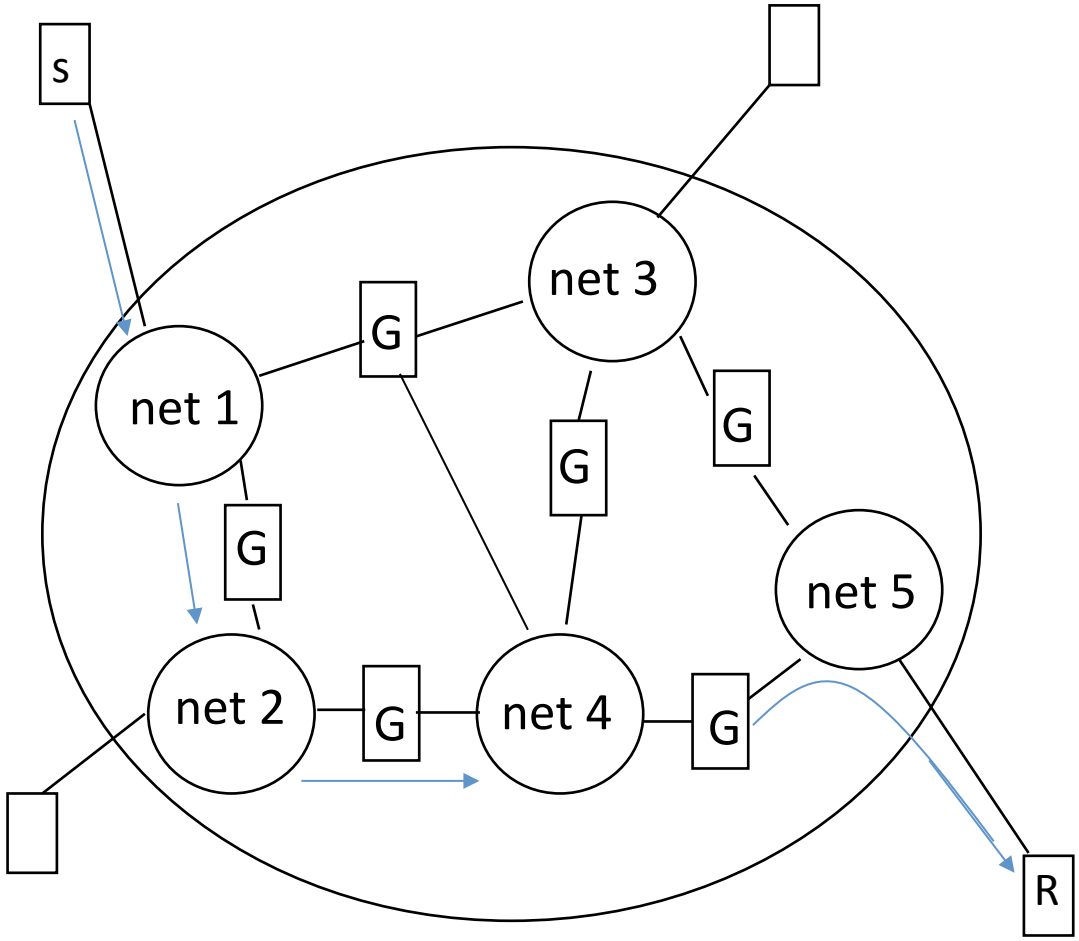
1. Find R's IP address by DNS.
2. Check its routing table for R, if find (next hop), send to it.
3. Otherwise, send to default router
4. Needs to find the physical address of the next hop router.
5. The router checks its routing table for the next hop and send to it.



Routing : **generalization**

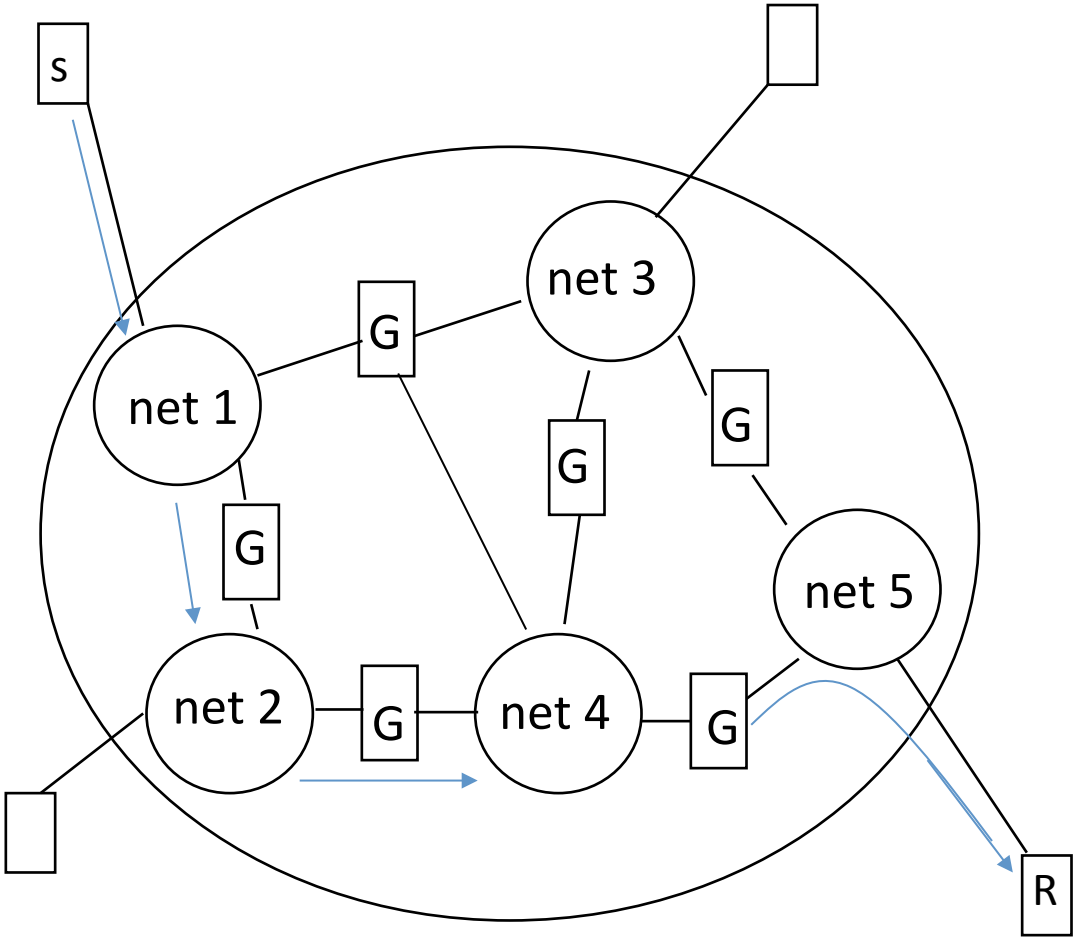
S sends a packet to R:

1. Find R's IP address by DNS.
2. Check its routing table for R, if find (next hop), send to it.
3. Otherwise, send to default router
4. Needs to find the physical address of the next hop router.
5. The router checks its routing table for the next hop and send to it.

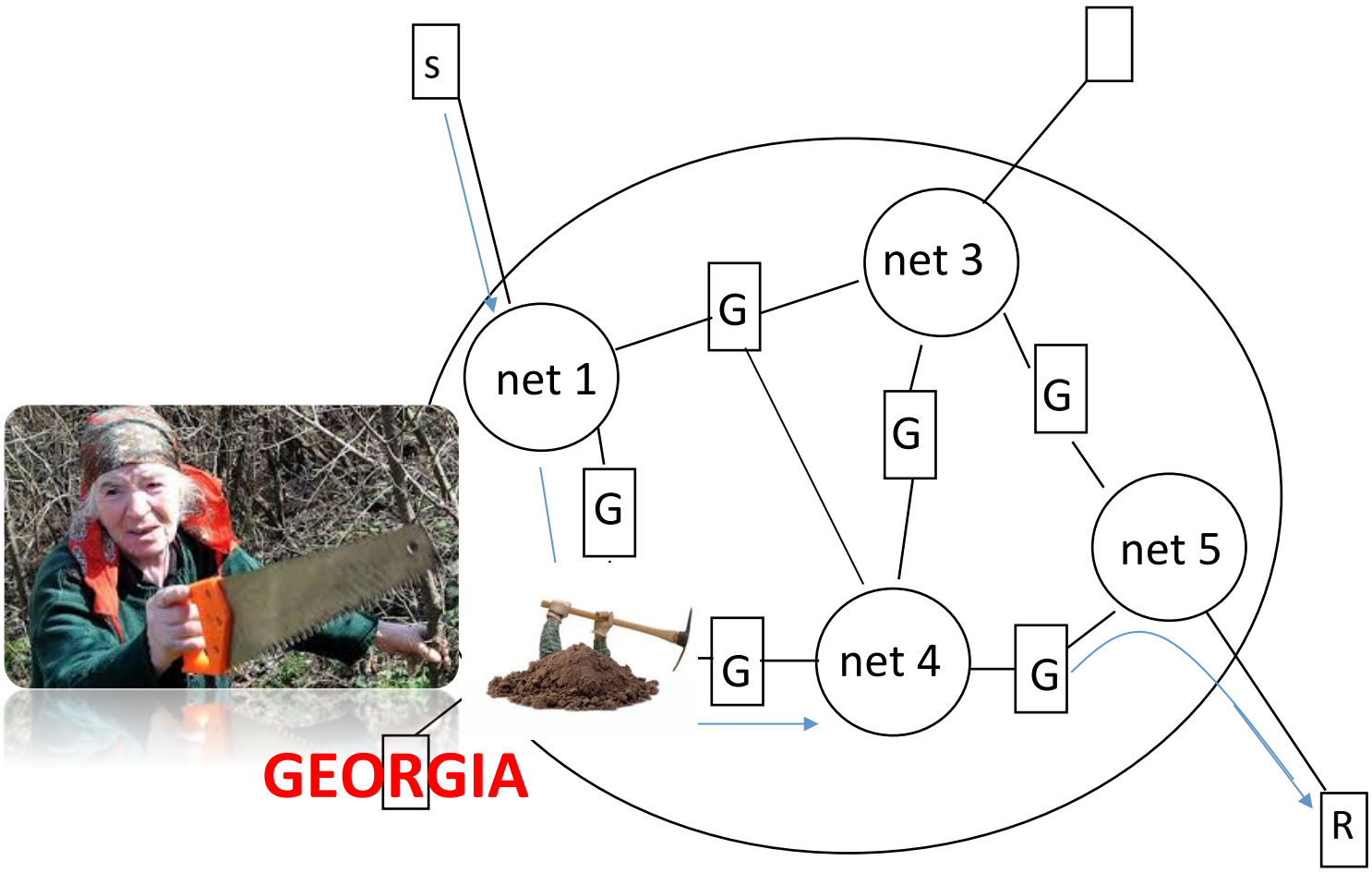


6. continue until the packet reaches the router in the same LAN with R.
7. The router finds R's physical address and sends to it.

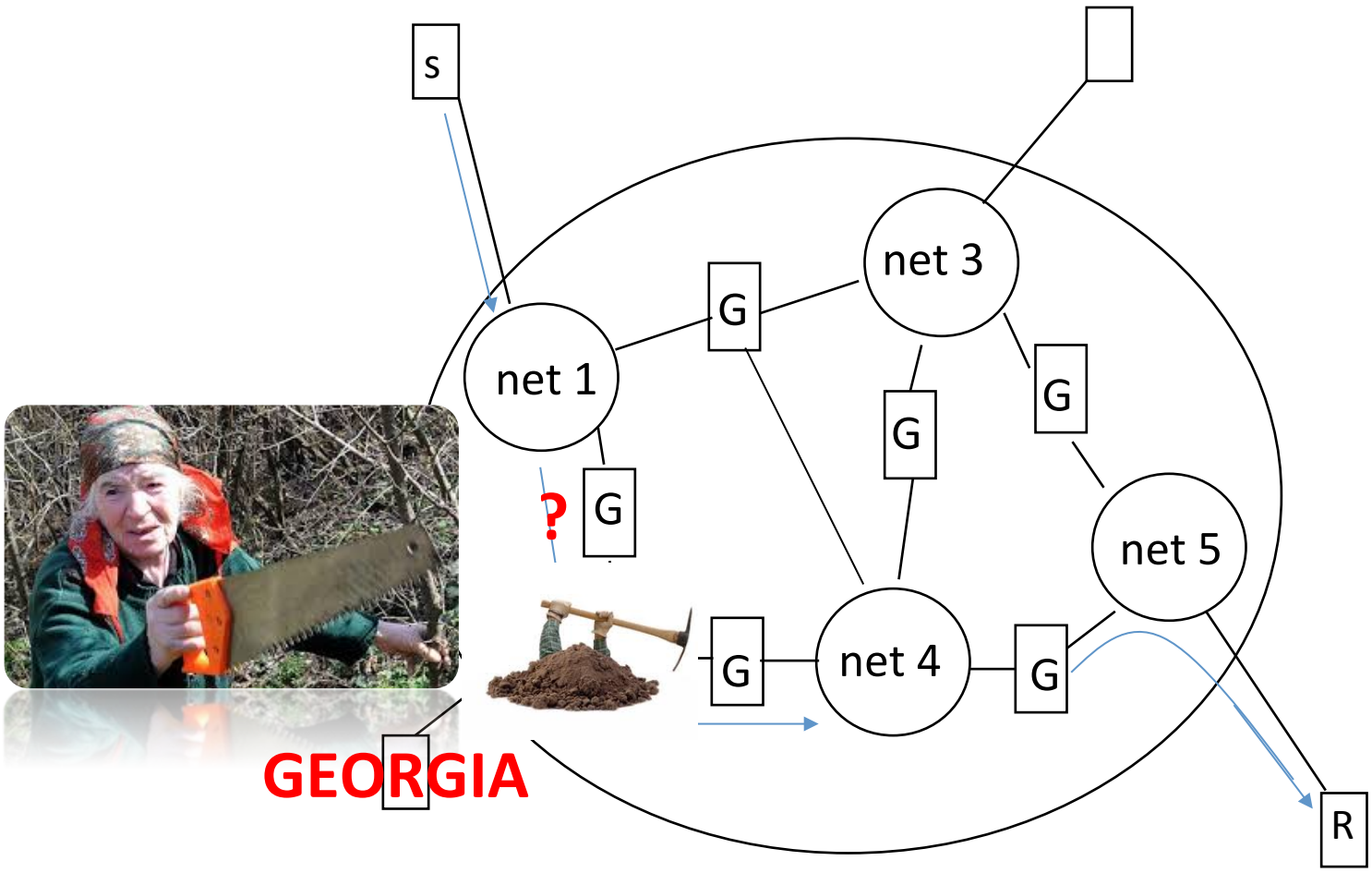
Routing : **generalization**



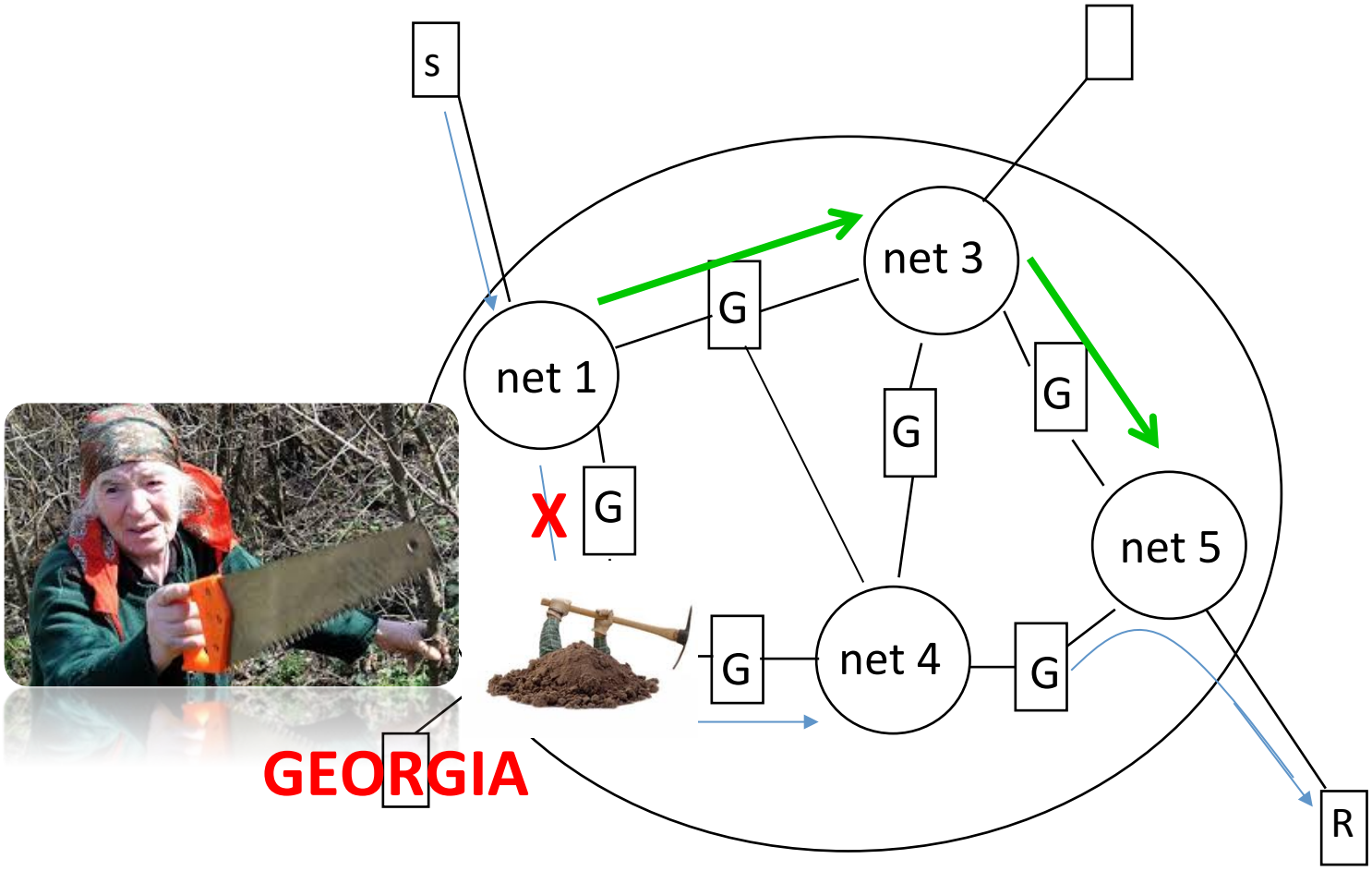
Routing : **generalization**



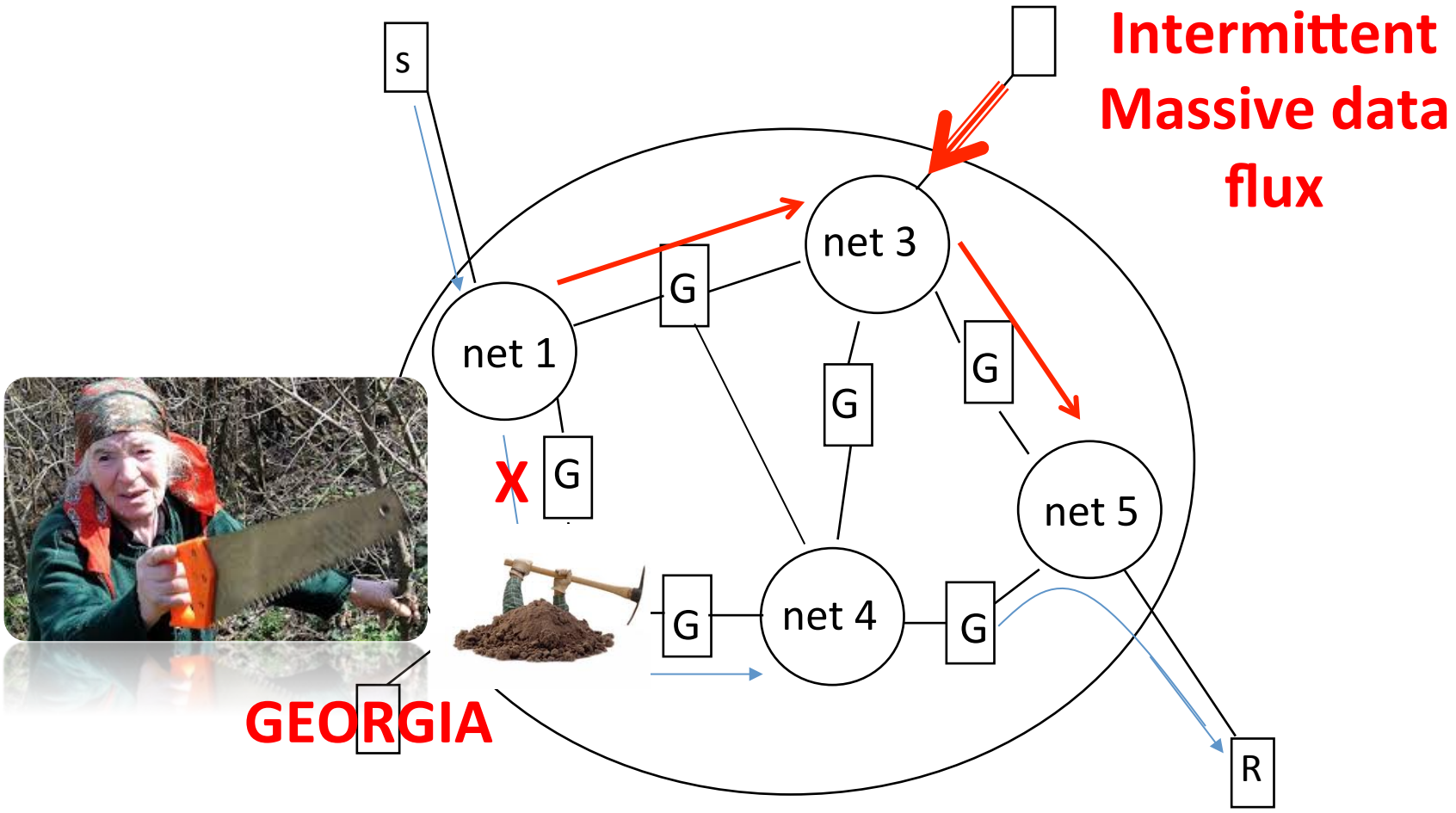
Routing : **generalization**



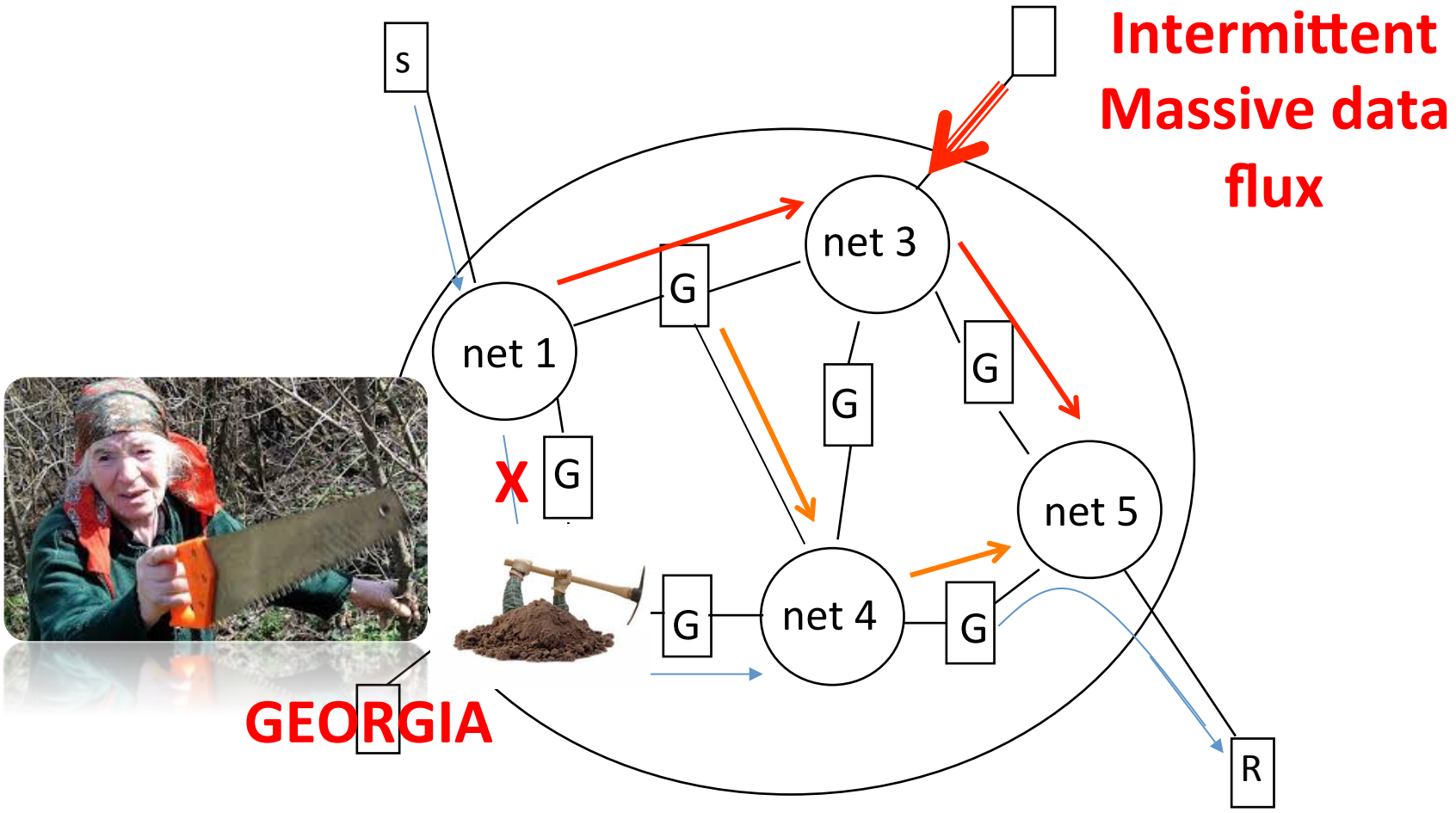
Routing : **generalization**



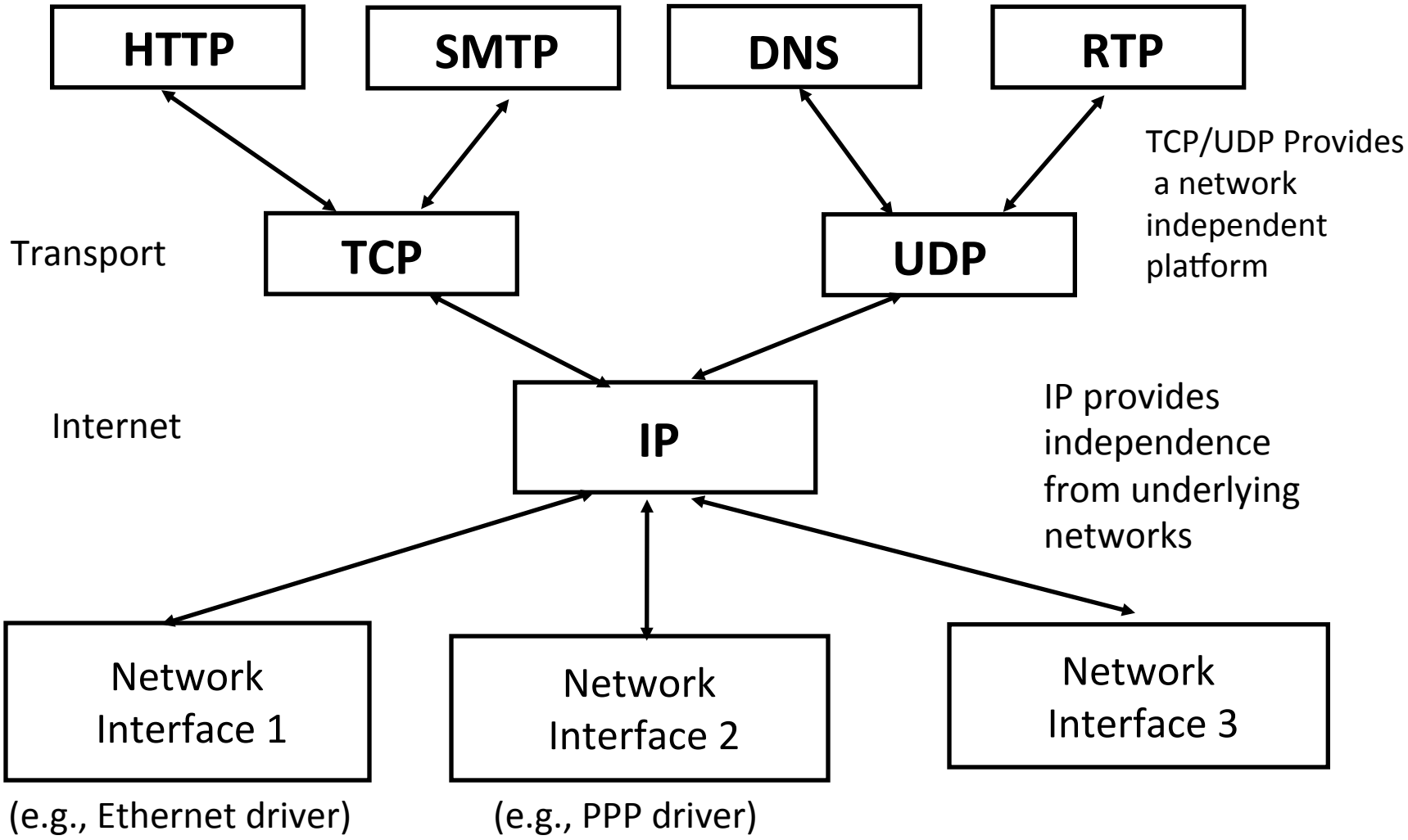
Routing : **generalization**



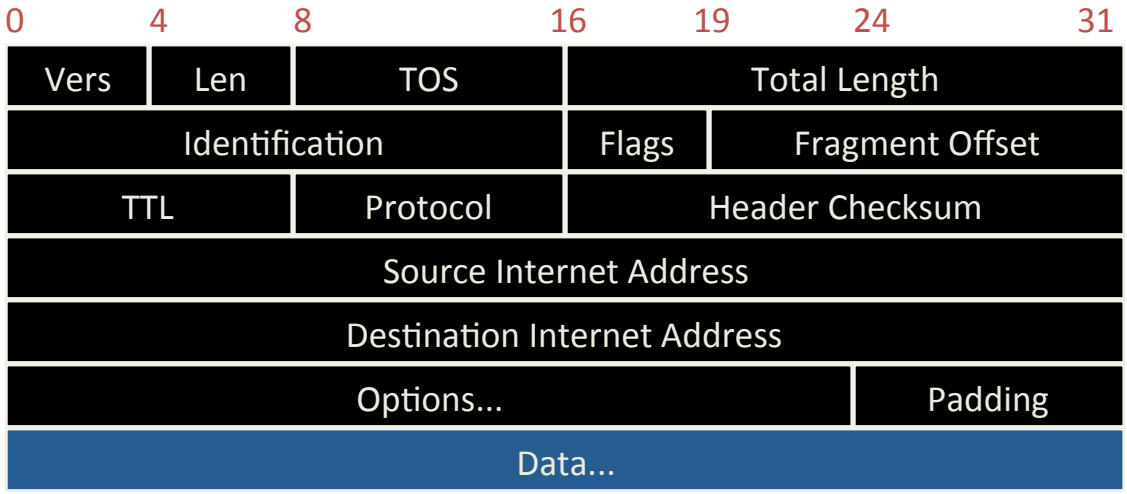
Routing : **generalization**



The acronym party



Structure of IP datagrams



Field	Purpose
Vers	IP version number
Len	Length of IP header (4 octet units)
TOS	Type of Service
T. Length	Length of entire datagram (octets)
Ident.	IP datagram ID (for frag/reassembly)
Flags	Don't/More fragments
Frag Off	Fragment Offset

Field	Purpose
TTL	Time To Live - Max # of hops
Protocol	Higher level protocol (1=ICMP, 6=TCP, 17=UDP)
Checksum	Checksum for the IP header
Source IA	Originator's Internet Address
Dest. IA	Final Destination Internet Address
Options	Source route, time stamp, etc.
Data...	Higher level protocol data

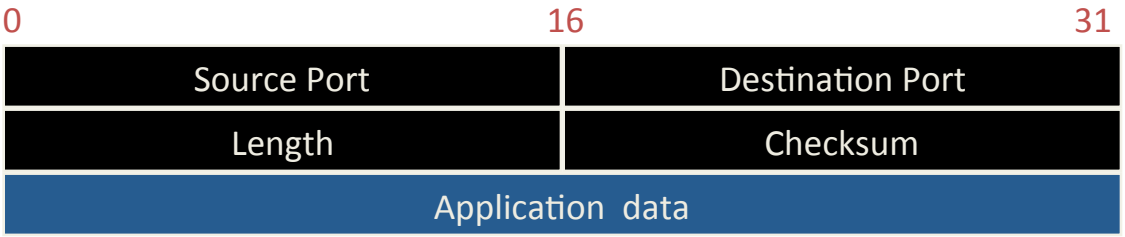
You just need to know the IP addresses, TTL and protocol #

IP Datagrams

- Vers (4 bits): version of IP protocol (IPv4=4)
- Type of Service – TOS (8 bits): little used in past, now QoS
- Total length (16 bits): length of datagram in bytes
- **Time to live – TTL** (8bits): specifies how long datagram is allowed to remain in internet
  - Routers decrement by 1
  - When TTL = 0 router discards datagram
  - **Prevents infinite loops**
- **Protocol** (8 bits): specifies the format of the data area
  - Protocol numbers administered by central authority (TCP=6, UDP=17)
- **Source & destination IP address** (32 bits each): contain IP address of sender and intended recipient
- Options (variable length): Mainly used to record a route
- All these can allow **packet sniffing (#hack ^^)**
- **Can allow WEP / WiFi security key hacking**

On top of IP : UDP

- Thin layer on top of IP
- Adds packet length + checksum (guard against *corrupted packets*)
- Also source and destination *ports (*used to associate with an application)
- **Still unreliable:**
  - Duplication, loss, out-of-orderness possible



Field	Purpose
Source Port	16-bit port number identifying originating application
Destination Port	16-bit port number identifying destination application
Length	Length of UDP datagram (UDP header + data)
Checksum	Checksum of IP pseudo header, UDP header, and data

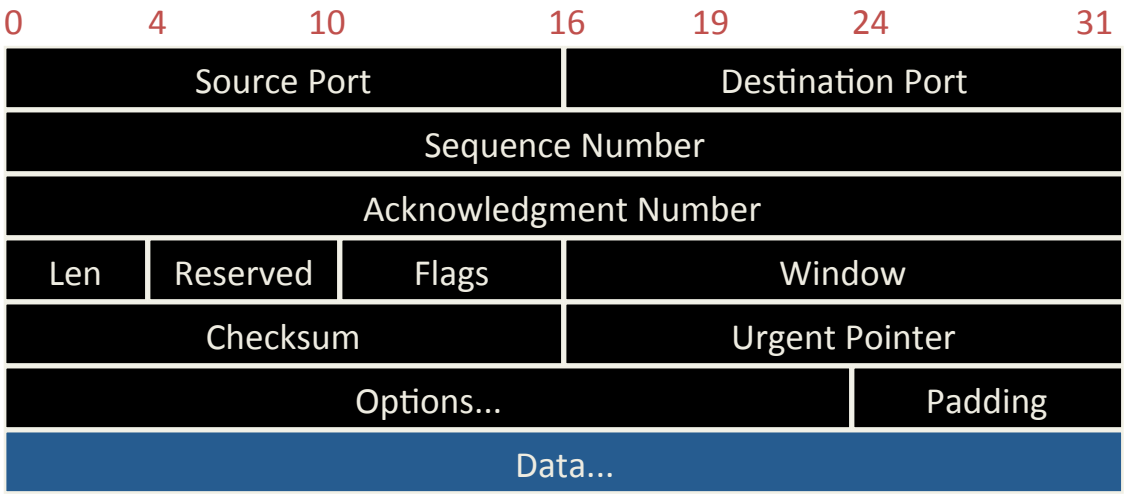
Synchronous version = TCP

- Reliable, *full-duplex*, *connection-oriented*, *stream* delivery
  - Interface presented to the application doesn't require data in individual packets
  - Data is guaranteed to arrive, and in the correct order without duplications
    - Or the connection will be dropped
  - Imposes significant overheads
- Used almost everywhere …
  - HTTP, FTP, …
- Saves the application a lot of work, so used unless there's a good reason not to

So why using UDP at all ?

- Where packet loss is better handled by the application than the network stack
- Where the overhead of setting up a connection isn't wanted
- Asynchronous behavior
- Avoids TCP hanging (packet waiting)
- **VOIP**
- **NFS – Network File System**
- **Most games … and music also !**

TCP Packets



Field	Purpose
Source Port	Identifies originating application
Destination Port	Identifies destination application
Sequence Number	**Sequence number of first octet in the segment**
Acknowledgment #	**Sequence number of the next expected octet (if ACK flag set)**
Len	Length of TCP header in 4 octet units
Flags	TCP flags: SYN, FIN, RST, PSH, ACK, URG
Window	Number of octets from ACK that sender will accept
Checksum	Checksum of IP pseudo-header + TCP header + data
Urgent Pointer	Pointer to end of “urgent data”
Options	Special TCP options such as MSS and Window Scale

So to summarize

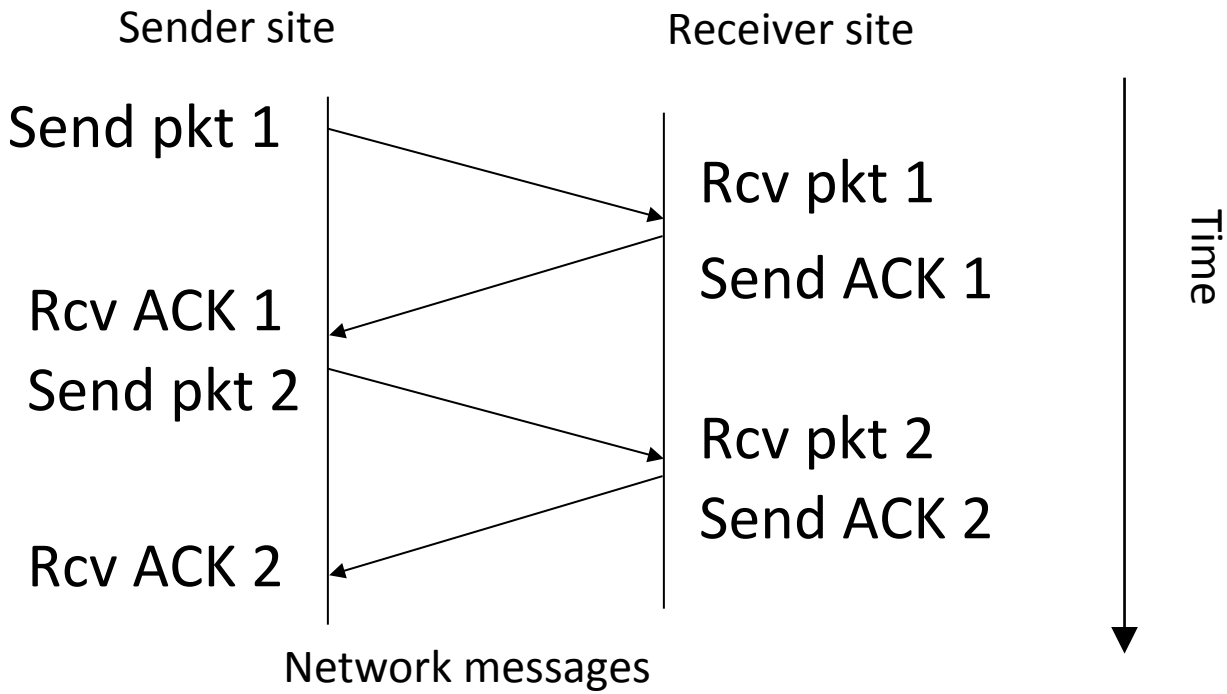
- Application layer directly run over the transport layer, corresponding to OSI transport layer.
- Two kinds of services: TCP & UDP.
- TCP—Transmission Control Protocol, reliable connect-oriented transfer of a byte stream.
- UDP—User Datagram Protocol, best-effort connectionless transfer of individual messages.
- Choose the best protocol depending on your needs

TCP : Providing reliability

- Positive acknowledgement (ACK) with retransmission
  - Sender keeps record of each packet sent
  - Sender awaits an ACK
  - Sender starts timer when sends packet

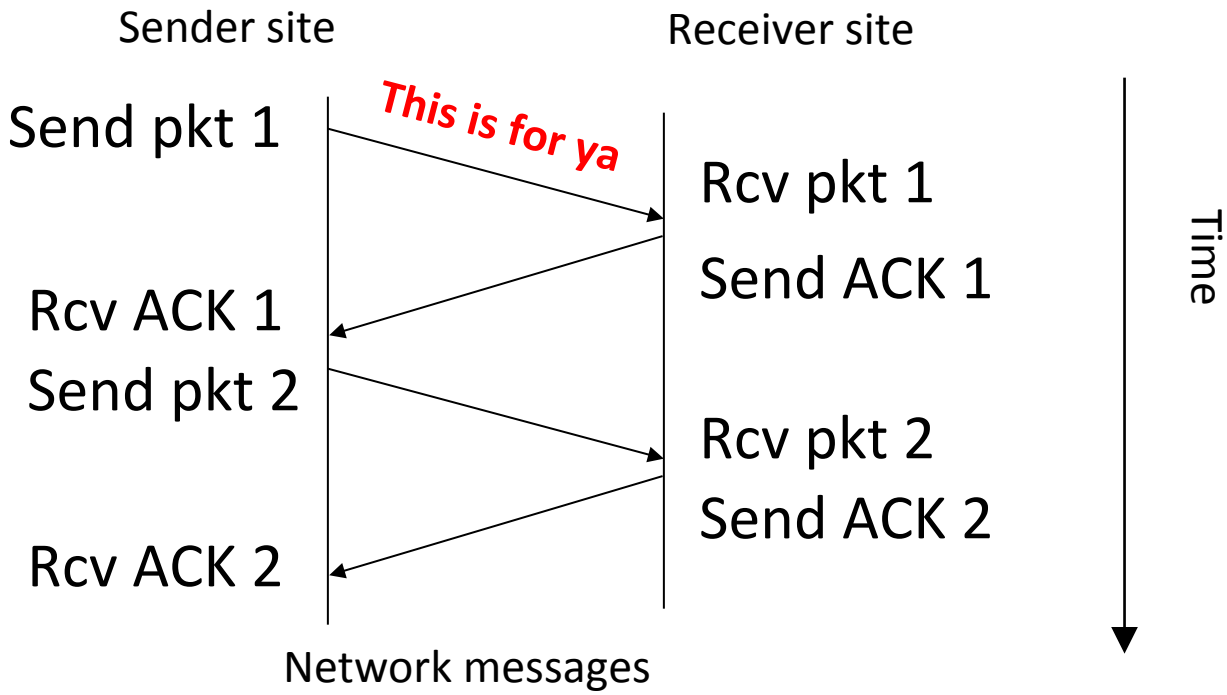
TCP : Providing reliability

- Positive acknowledgement (ACK) with retransmission
  - Sender keeps record of each packet sent
  - Sender awaits an ACK
  - Sender starts timer when sends packet



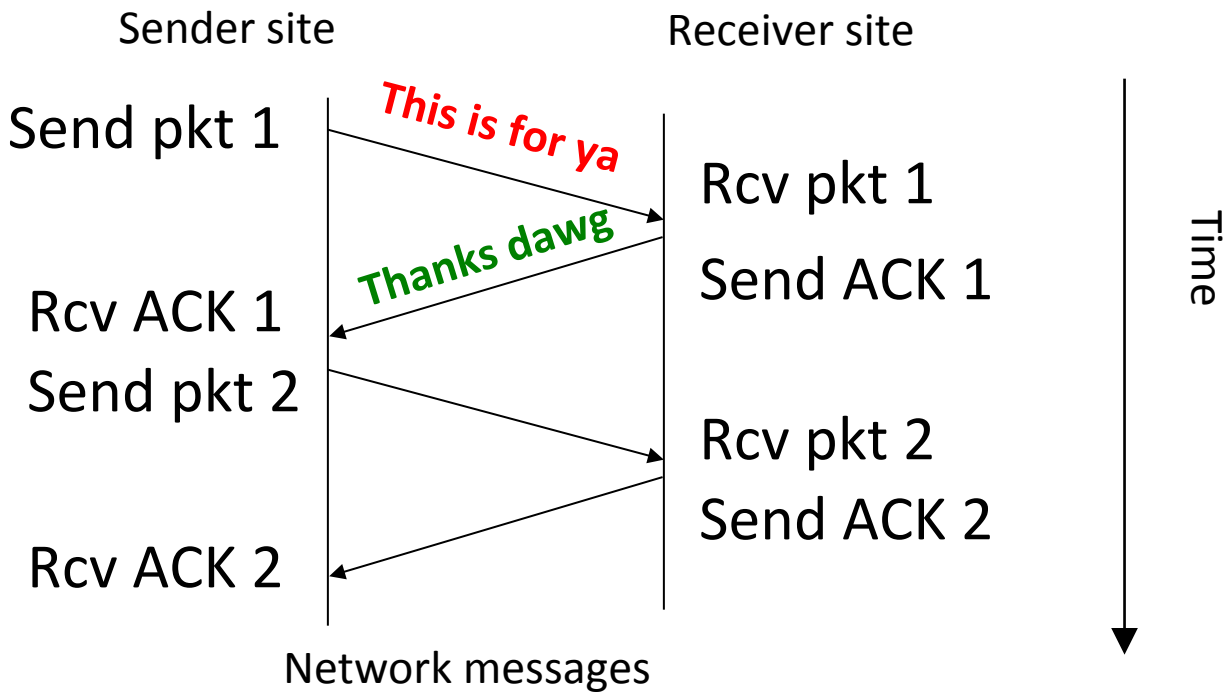
TCP : Providing reliability

- Positive acknowledgement (ACK) with retransmission
  - Sender keeps record of each packet sent
  - Sender awaits an ACK
  - Sender starts timer when sends packet



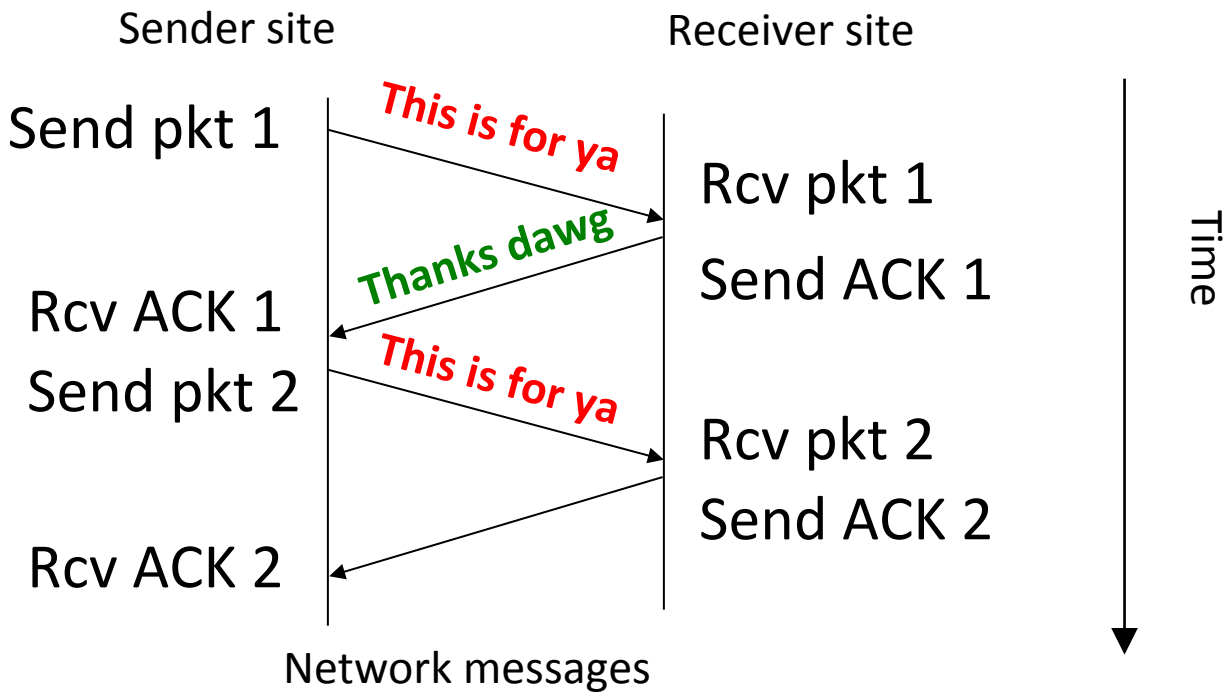
TCP : Providing reliability

- Positive acknowledgement (ACK) with retransmission
  - Sender keeps record of each packet sent
  - Sender awaits an ACK
  - Sender starts timer when sends packet



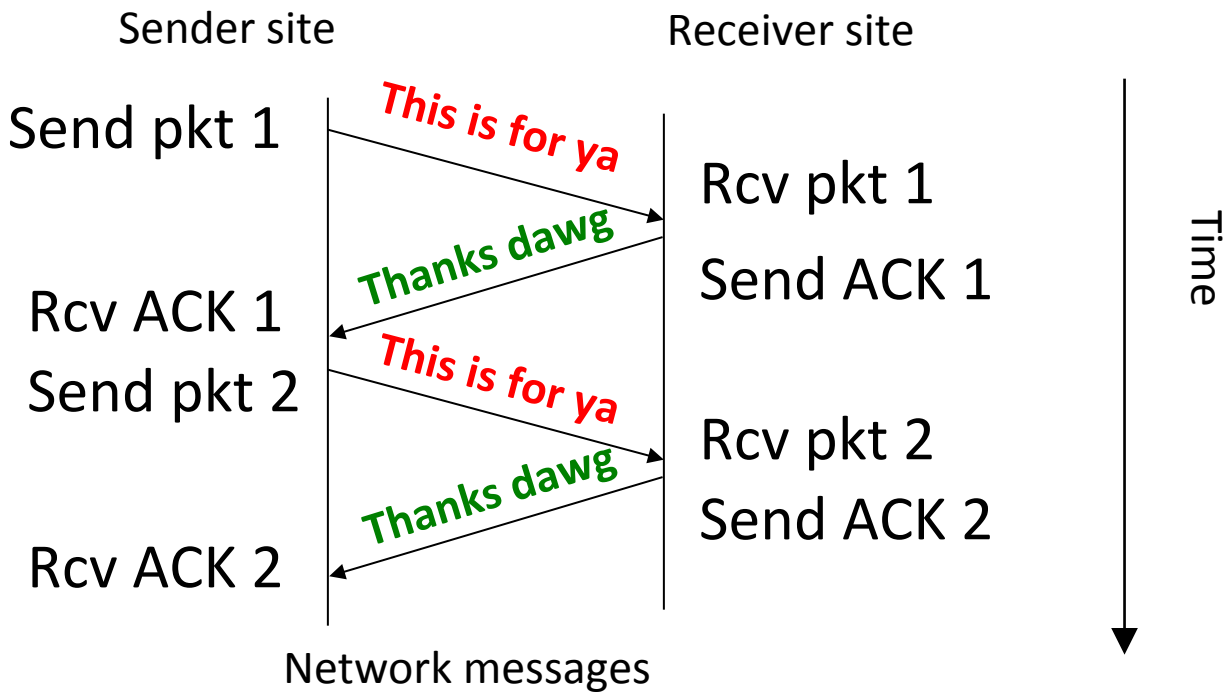
TCP : Providing reliability

- Positive acknowledgement (ACK) with retransmission
  - Sender keeps record of each packet sent
  - Sender awaits an ACK
  - Sender starts timer when sends packet



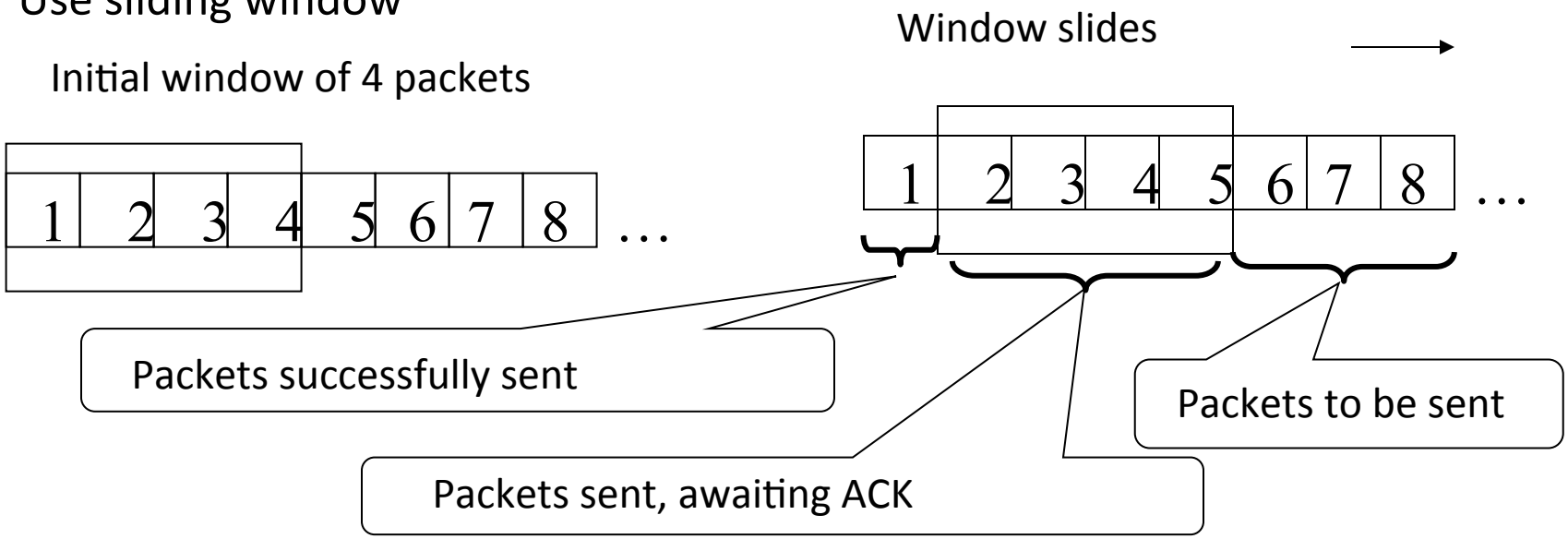
TCP : Providing reliability

- Positive acknowledgement (ACK) with retransmission
  - Sender keeps record of each packet sent
  - Sender awaits an ACK
  - Sender starts timer when sends packet



TCP Sliding window approach

- BUT simple ACK protocol wastes bandwidth since it must delay sending next packet until it gets ACK
- Use sliding window



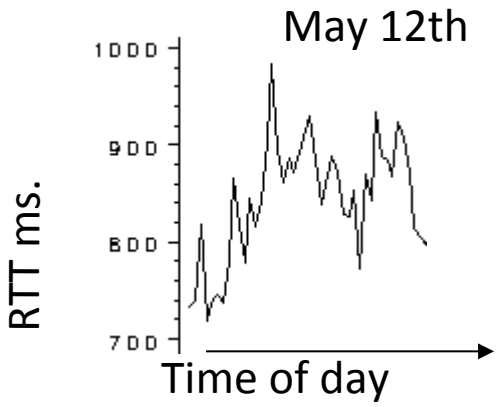
- Sender can send 4 packets of data without ACK
  - When sender gets ACK then can send another packet
  - Window = unacknowledged packets/bytes
  - Keeps timer for each packet

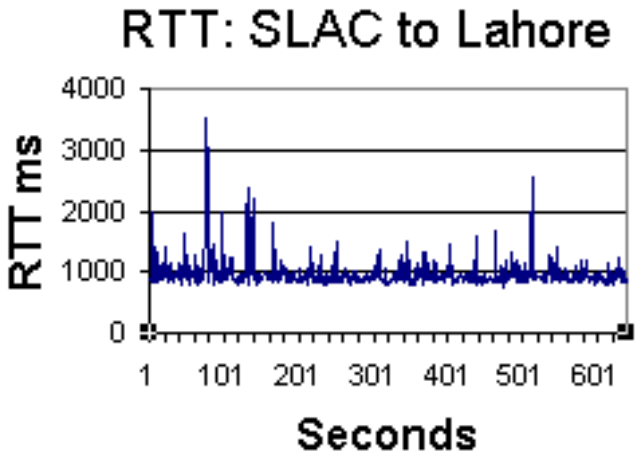
TCP Sliding window

- Windows vary over time
  - Receiver advertises (in ACKs) how many it can receive
    - Based on buffers etc. available
  - Sender adjusts its window to match advertisement
  - If receiver buffers fill, it sends smaller adverts
- Used to match buffer requirements of receiver
- Also used to address congestion control (e.g. in intermediate routers)

TCP Timeout

- Need a timeout estimate that will work
  - for LANs (RTT < msec.)
  - to satellite WANs (~100 msec. to secs)
  - RTT can vary a lot with time of day
  - … or weeks, or even inside a second
- TCP records time segment sent
  - and time ACK received
  - Then calculates RTT sample
- Smooth & use to estimate timeout,
  - Timeout=beta * RTT_s
    - Timeout = RTT_s + eta{=4}*f(dev(RTT_s))
  - Needs to take account of losses, e.g.
    - New_timeout=gamma{2} * timeout



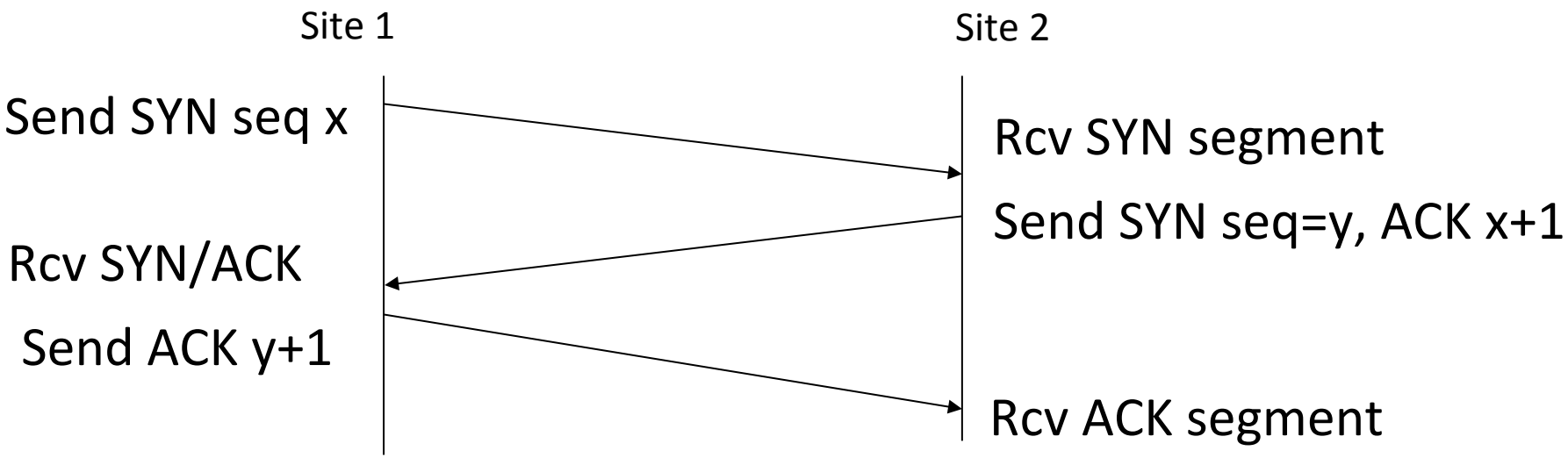


TCP Implementation

- Connections are established using a *three-way handshake*
- Data is divided up into packets by the operating system
- Packets are numbered, and received packets are acknowledged
- Connections are explicitly closed
  - (or may abnormally terminate)

TCP Connection establishment

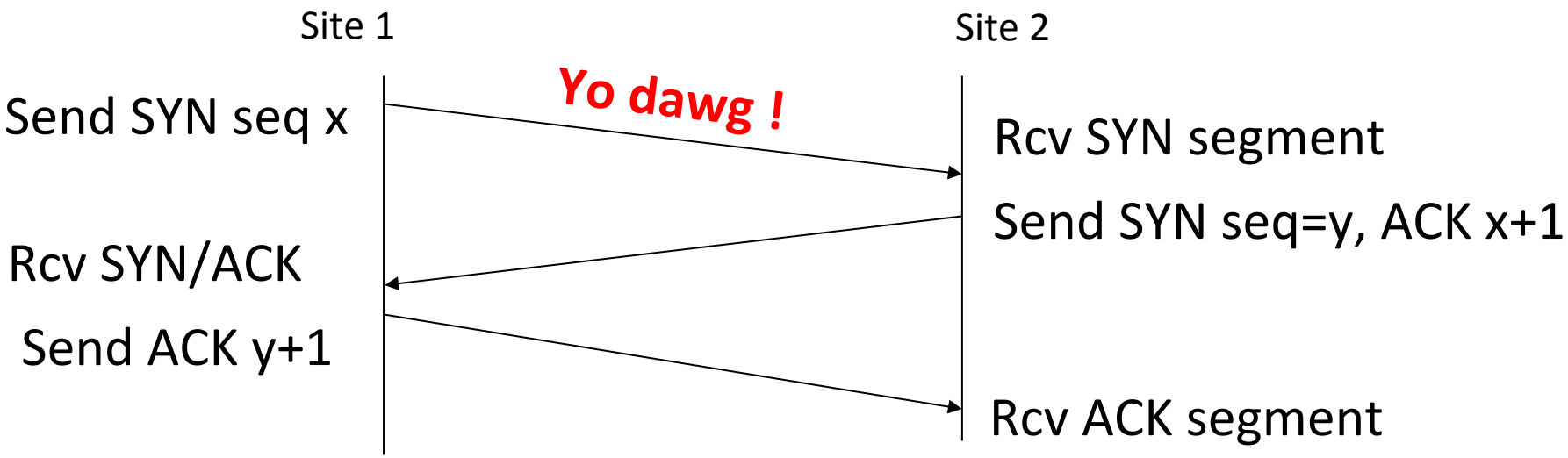
- 3 way handshake



- Initial sequence numbers (x, y) are chosen randomly
- Guarantees both sides ready & know it, and sets initial sequence numbers, also sets window & mss
- Once connection established, data can flow in both directions, equally well, there is no master or slave

TCP Connection establishment

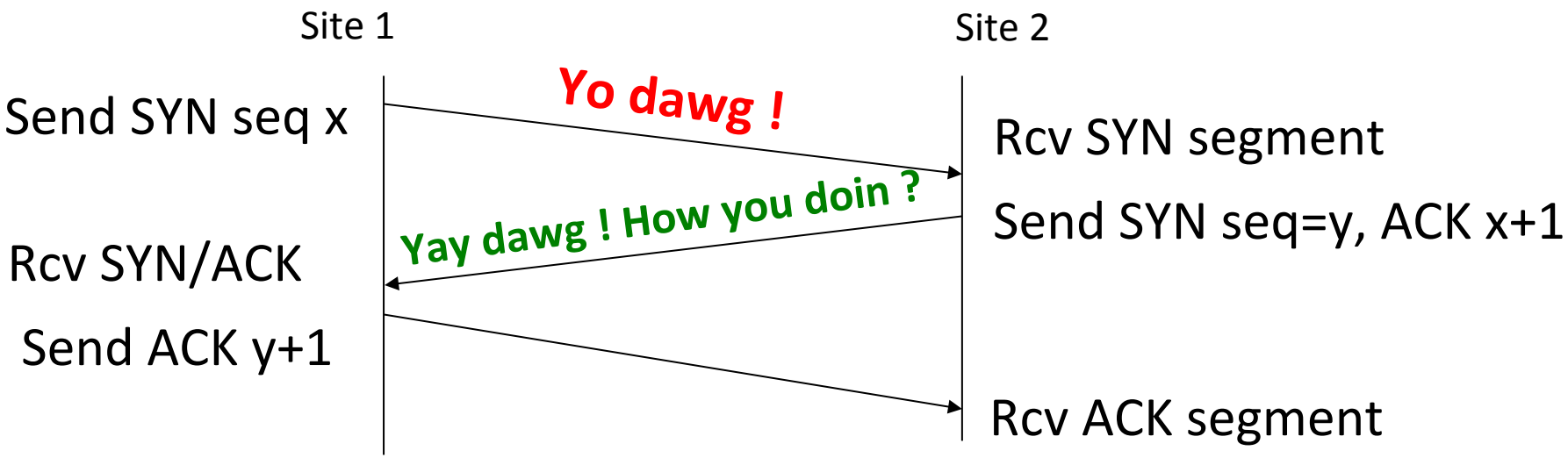
- 3 way handshake



- Initial sequence numbers (x, y) are chosen randomly
- Guarantees both sides ready & know it, and sets initial sequence numbers, also sets window & mss
- Once connection established, data can flow in both directions, equally well, there is no master or slave

TCP Connection establishment

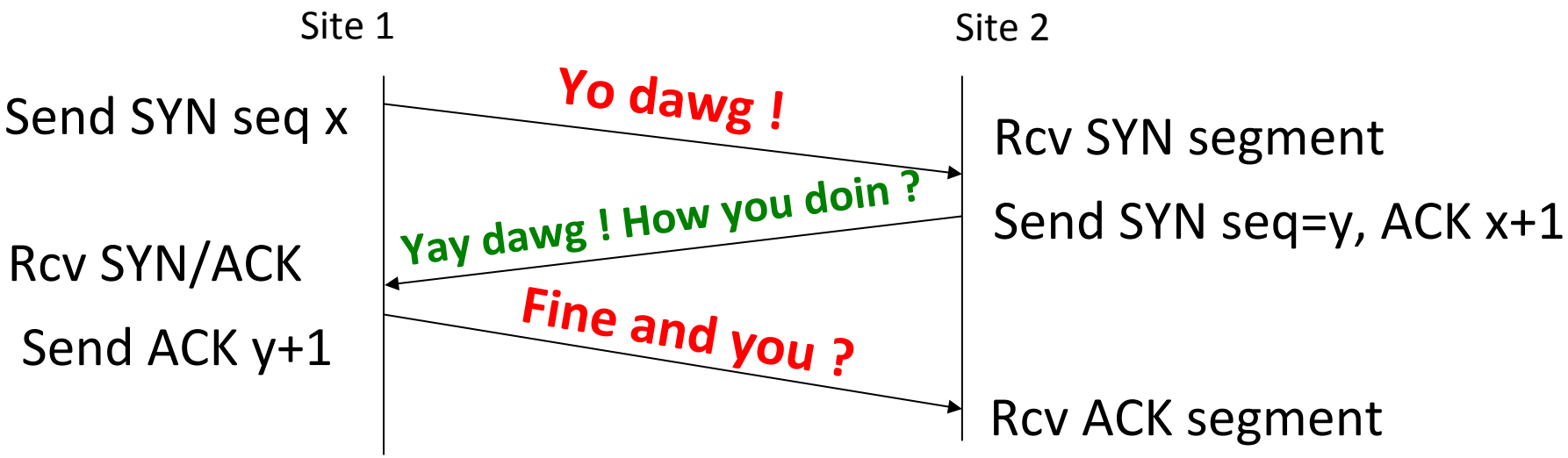
- 3 way handshake



- Initial sequence numbers (x, y) are chosen randomly
- Guarantees both sides ready & know it, and sets initial sequence numbers, also sets window & mss
- Once connection established, data can flow in both directions, equally well, there is no master or slave

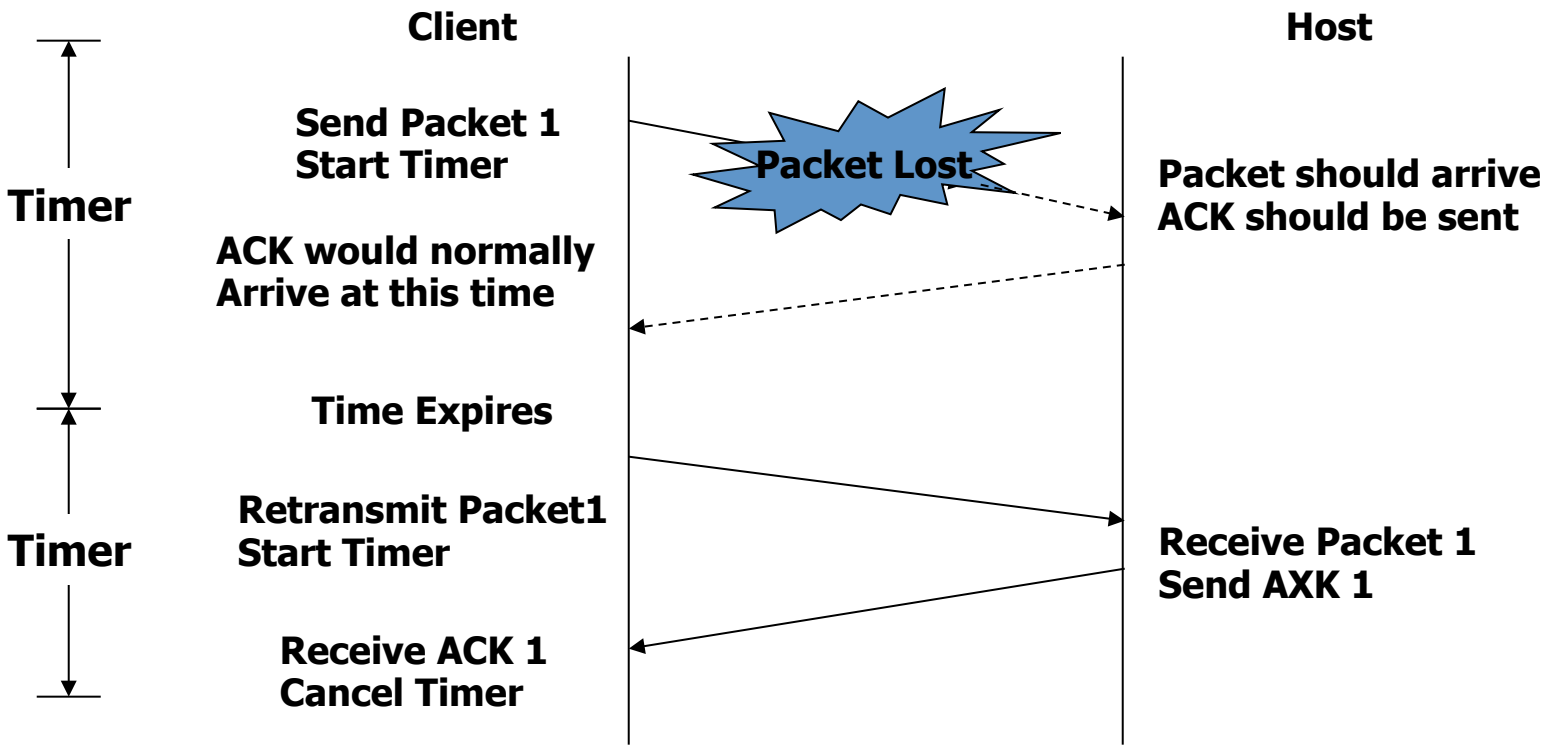
TCP Connection establishment

- 3 way handshake



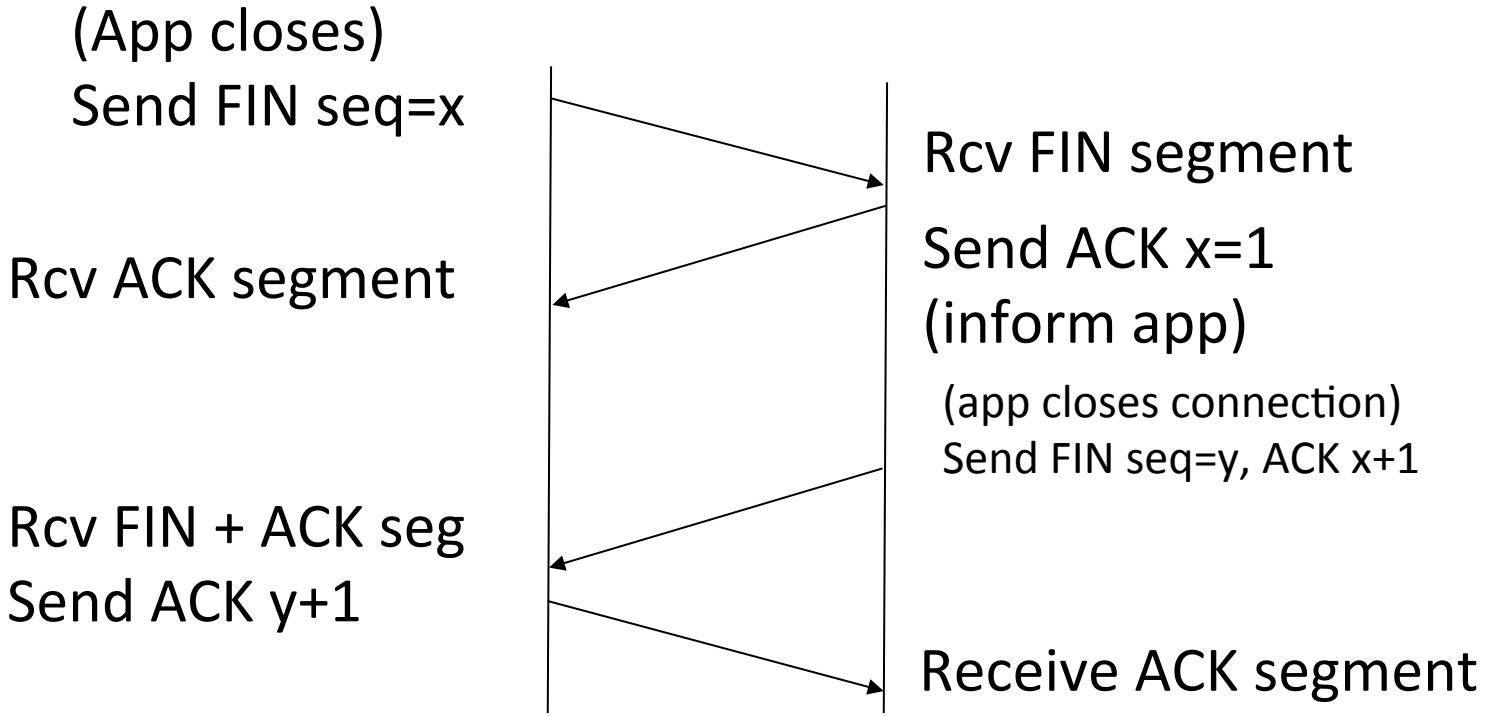
- Initial sequence numbers (x, y) are chosen randomly
- Guarantees both sides ready & know it, and sets initial sequence numbers, also sets window & mss
- Once connection established, data can flow in both directions, equally well, there is no master or slave

TCP Packet recovery !



TCP Connection termination

- Modified 3 way handshake (or 4 way termination)

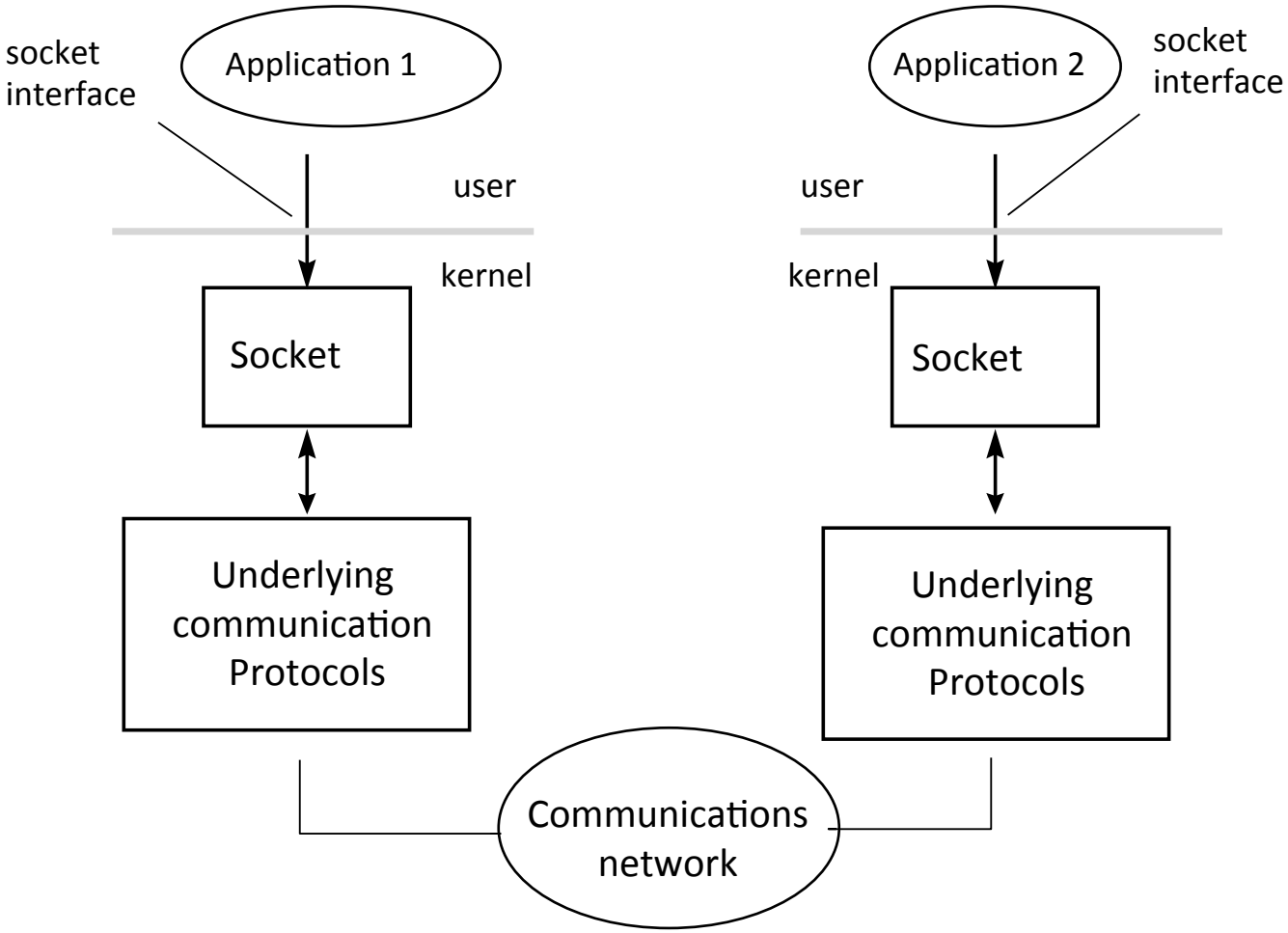


- App tells TCP to close, TCP sends remaining data & waits for ACK, then sends FIN
- Site 2 TCP ACKs FIN, tells its application "end of data"
- Site 2 sends FIN when its app closes connection (may be long delay (e.g. require human interaction).

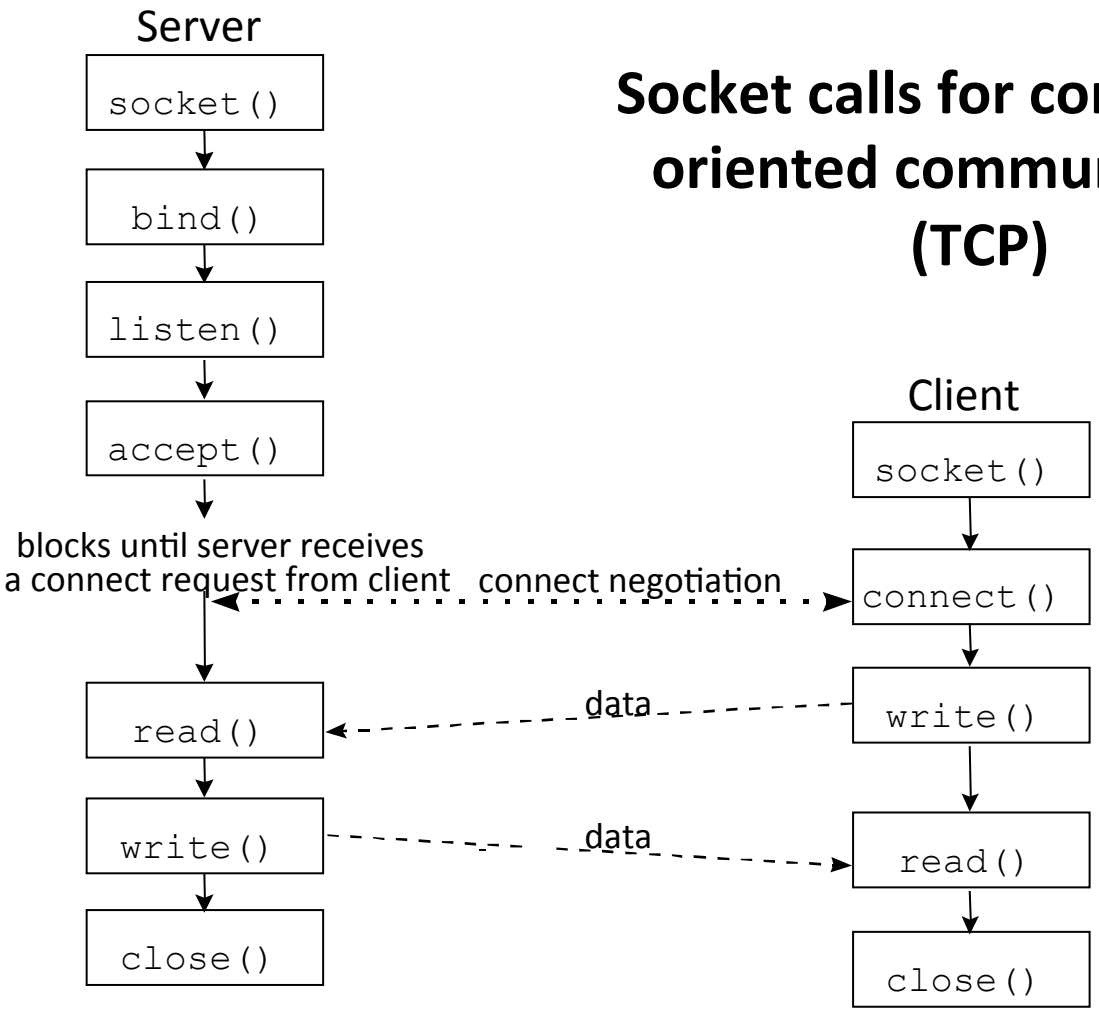
Berkeley **Socket** interface

- The most popular interface to access network resources
- Write applications without worry about underlying networking detail
- Connection-oriented service (TCP connection and transfer) and connectionless service (UDP datagram delivery)
- Socket is physically a handle on which other functions can be called and finish access tasks.
- Used in Open Sound Control (OSC)
- … with UDP or TCP ☺

Berkeley **Socket** interface

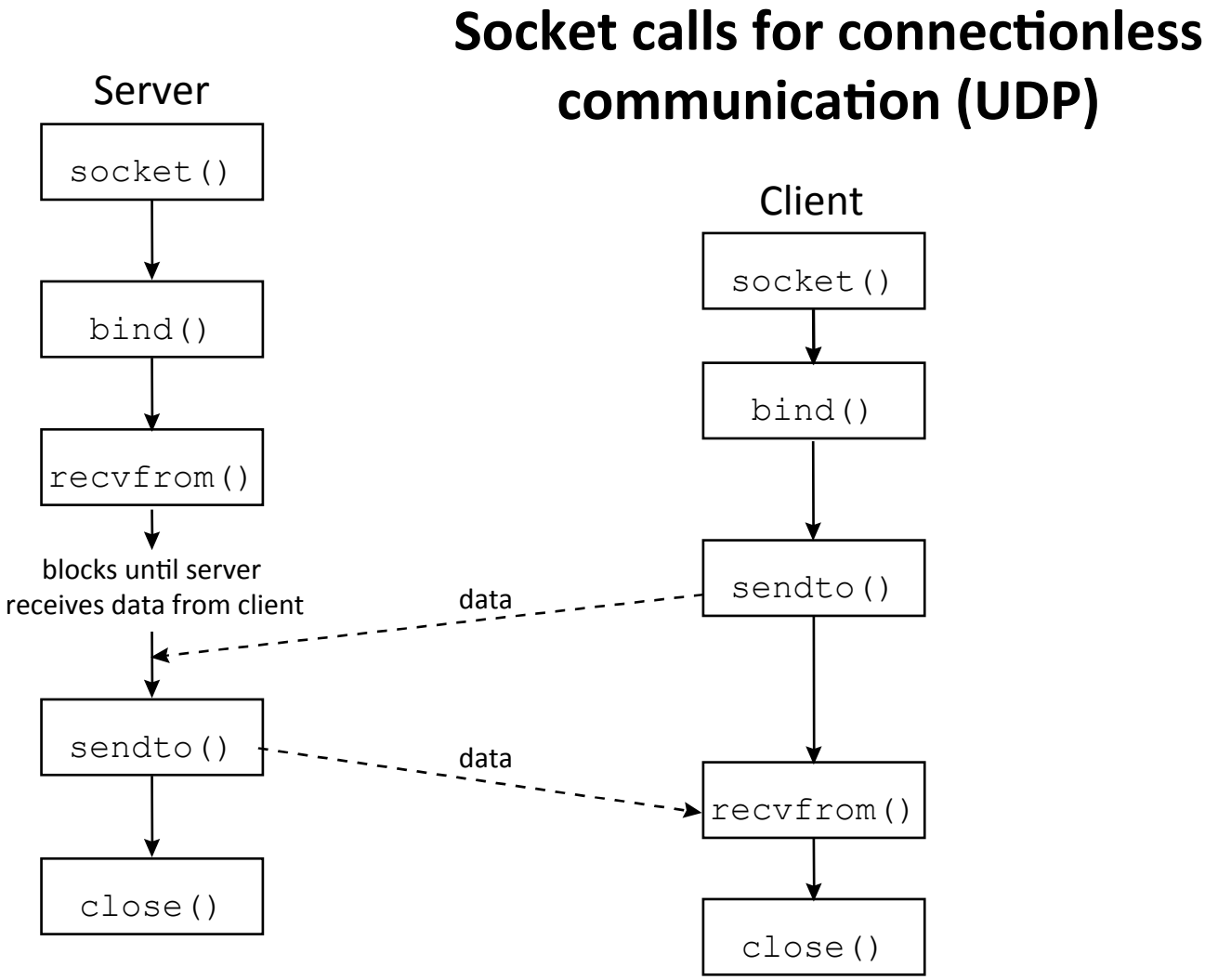


Berkeley **Socket** interface



Socket calls for connection-oriented communication (TCP)

Berkeley **Socket** interface



Internet addressing

- IP address is a 32 bit integer
  - Refers to interface rather than host
  - Consists of network and host portions
    - Enables routers to keep 1 entry/network instead of 1/host
  - Class A, B, C for unicast
  - Class D for multicast
  - Class E reserved
  - Classless addresses
- Written as 4 octets/bytes in decimal format
  - E.g. 134.79.16.1, 127.0.0.1

Class-based internet addressing

- Class A: large number of hosts, few networks
  - 0nnnnnnn hhhhhhhh hhhhhhhh hhhhhhhh
    - 7 network bits (0 and 127 reserved, so 126 networks)
    - 24 host bits (> 16M hosts/net)
    - Initial byte 1-127 (decimal)
- Class B: medium number of hosts and networks
  - 10nnnnnn nnnnnnnn hhhhhhhh hhhhhhhh
    - 16,384 class B networks, 65,534 hosts/network
    - Initial byte 128-191 (decimal)
- Class C: large number of small networks
  - 110nnnnn nnnnnnnn nnnnnnnn hhhhhhhh
    - 2,097,152 networks, 254 hosts/network
    - Initial byte 192-223 (decimal)
- Class D: 224-239 (decimal) Multicast [RFC1112]
- Class E: 240-255 (decimal) Reserved

Address depletion ?

- In 1991 IAB identified 3 dangers
  - Running out of class B addresses
  - Increase in nets has resulted in routing table explosion
  - Increase in net/hosts exhausting 32 bit address space
- Four strategies to address
  - Creative address space allocation {RFC 2050}
  - Private addresses {RFC 1918}, Network Address Translation (NAT) {RFC 1631}
  - Classless InterDomain Routing (CIDR) {RFC 1519}
  - IP version 6 (IPv6) {RFC 1883}

Creative IP allocation

- Class A addresses 64 – 127 reserved
  - Handle on individual basis
- Class B only assigned given a demonstrated need
- Class C
  - divided up into 8 blocks allocated to regional authorities
  - 208-223 remains unassigned and unallocated
- Three main registries handle assignments
  - APNIC – Asia & Pacific www.apnic.net
  - ARIN – N. & S. America, Caribbean & sub-Saharan Africa www.arin.net
  - RIPE – Europe and surrounding areas www.ripe.net

Private IP address

- What **you will really be using**
- IP addresses that are not globally unique
- But used exclusively in an organization
- Three ranges:
  - 10.0.0.0 - 10.255.255.255 a single class A net
  - 172.16.0.0 - 172.31.255.255 16 contiguous class Bs
  - 192.168.0.0 – 192.168.255.255 256 contiguous class Cs
- Connectivity provided by Network Address Translator (NAT)
  - translates outgoing private IP address to Internet IP address, and a return Internet IP address to a private address
  - Only for TCP/UDP packets

Open Sound Control (finally …)

- Transport-independent and message-based lightware protocol to share data between multimedia devices
- Similar to MIDI but more powerful
- Free implementations available for all kind of systems and languages
- Usually implemented using a set of knowledge
  - **Client / Server** design (star network)
  - Based on **socket interface** communication
  - Requires the **IP address** of the server to communicate
  - Exchange data following **MOSTLY UDP protocol !** (but can be tuned to **TCP**) **(why?)**
  - **Can be used in a localhost fashion !** (allows internal communication inside your laptop but between different sound programs)
  - **Extremely useful** in music, but also in sensor arrays, image processing, video, etc…
- Website: http://opensoundcontrol.org/

And the fun just goes on !

- Oppositely to MIDI there is …
  - **No fixed hardware wire transmission format required**
  - Can be used over WiFi, internet, USB, 4G, etc…
  - Unlimited extensibility …
  - … And unlimited bandwidth ! (just **your physical support eg. Ethernet = 100Mb/s**)

And the fun just goes on !

- Oppositely to MIDI there is …
  - **No fixed hardware wire transmission format required**
  - Can be used over WiFi, internet, USB, 4G, etc…
  - Unlimited extensibility …
  - … And unlimited bandwidth ! (just **your physical support eg. Ethernet = 100Mb/s**)
- And regarding the data types you can transport … well …

And the fun just goes on !

- Oppositely to MIDI there is …
  - **No fixed hardware wire transmission format required**
  - Can be used over WiFi, internet, USB, 4G, etc…
  - Unlimited extensibility …
  - … And unlimited bandwidth ! (just **your physical support eg. Ethernet = 1 Gb/s**)
- And regarding the data types you can transport … well …

Type	Example	Category	Frequency
Raw sensor data	RGB video image	Continuous at fixed sample rate and dimension	Very high
Raw audio data	Speech streaming		
Low-level feature	Hand position		
Event detection	Gesture	Event based with varying duration and dimension	Low - high
High-level feature	Gesture expressivity		
Classification	Gesture type		
Control message	Start/stop component	Text message	Seldom

OSC vs. MIDI

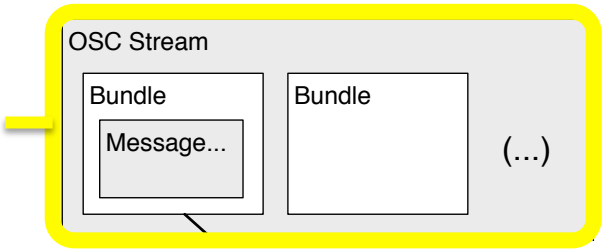
	OSC	MIDI

Examples of Messages	`/wii/ir/x 0.1503`
	`/stylus/pressure 0.014`
	`/camera/look-at 5. 12. 17.`
	`/play-note 15 0.9`
		`144 60 64 (MIDI Note-on)`
		`128 60 64 (MIDI Note-off)`

Message types	User-defined Human-readable	Pre-determined Byte-encoded
Atomic Update of Multiple Parameters	✓ via Bundles	☹
Time-tagging	✓ via Bundles	☹
Hardware Transport Independent	✓	☹
Number of channels	Unlimited	16
Data formats	Integer, Double Precision Floating Point, Strings, and more	1-byte integers 0-255
Message Structure	User-defined	Pre-determined
Microcontroller-friendly	✓	✓
State-machine independent	✓ * (Unless user-imposed)	☹ (e.g. “The Note-off problem”)
Application Areas	Music,, Video, Robotics and more	Music
Clock-sync accuracy, theoretical limit	picosecond (via NTP / IEEE 1588 Sync)	20833 microseconds
Data Rate	Gigabit Speed (> 800M bits / sec)	31,250 bits / second

OSC General structure

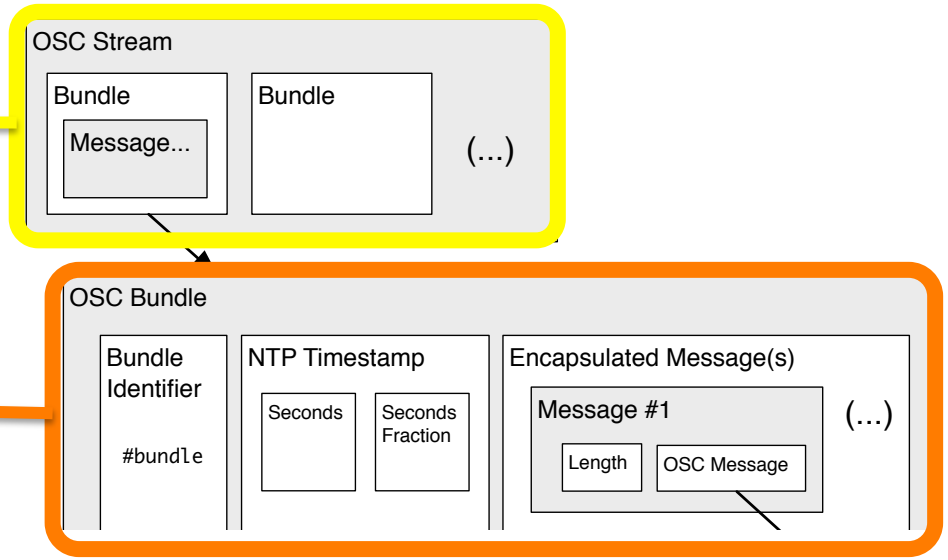
Generally OSC is seen as a stream of information bundles



OSC General structure

Generally OSC is seen as a stream of information bundles

An OSC Bundle is simply a « package » containing a timestamp and one to many messages

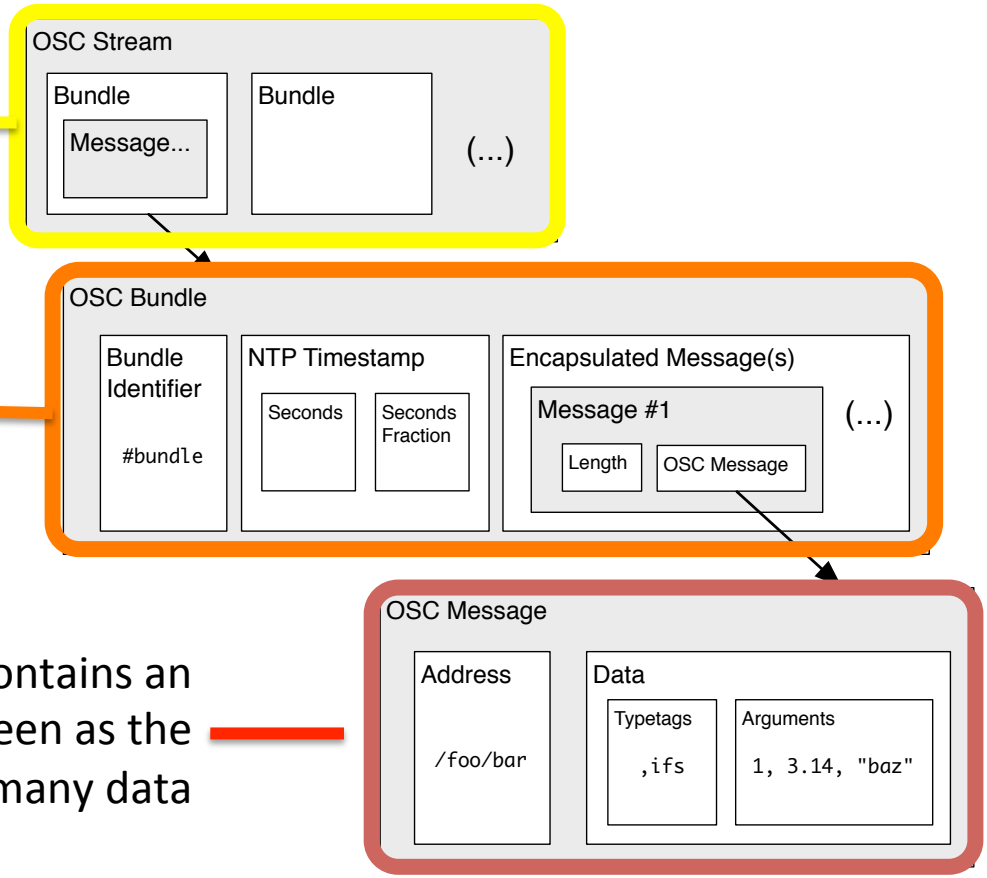


OSC General structure

Generally OSC is seen as a stream of information bundles

An OSC Bundle is simply a « package » containing a timestamp and one to many messages

A message contains an « address » (should be seen as the type of message) and many data

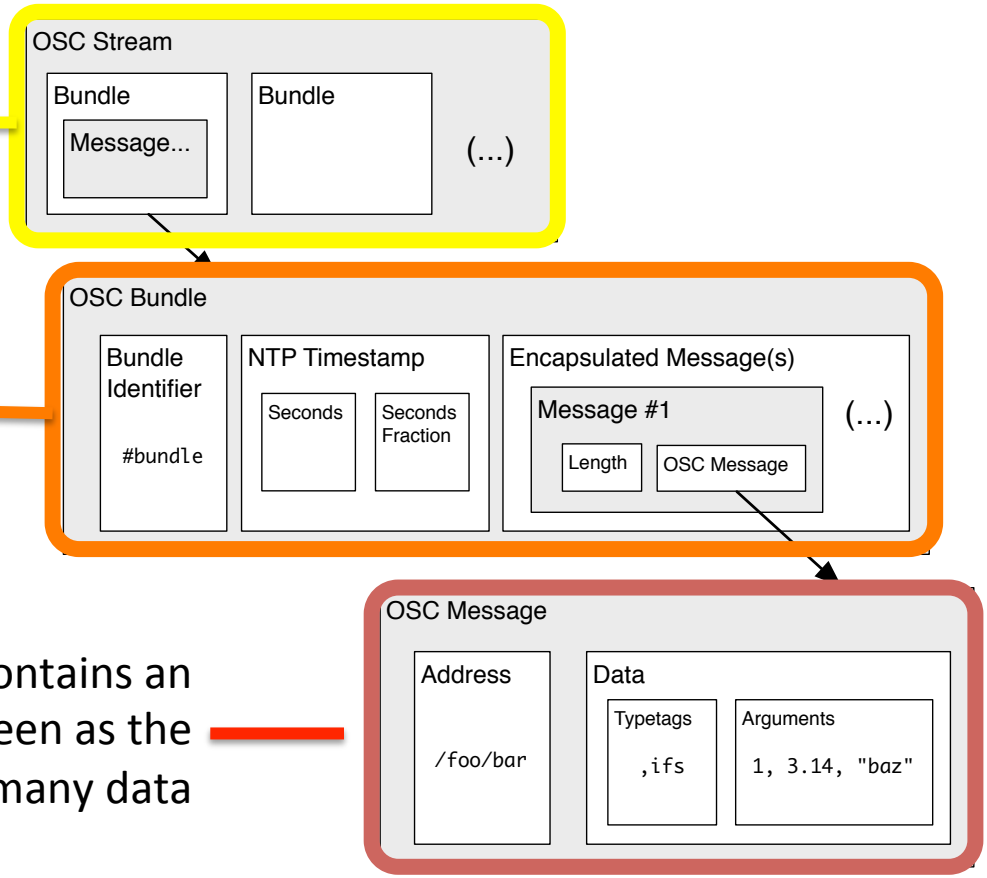


OSC General structure

Generally OSC is seen as a stream of information bundles

An OSC Bundle is simply a « package » containing a timestamp and one to many messages

A message contains an « address » (should be seen as the type of message) and many data



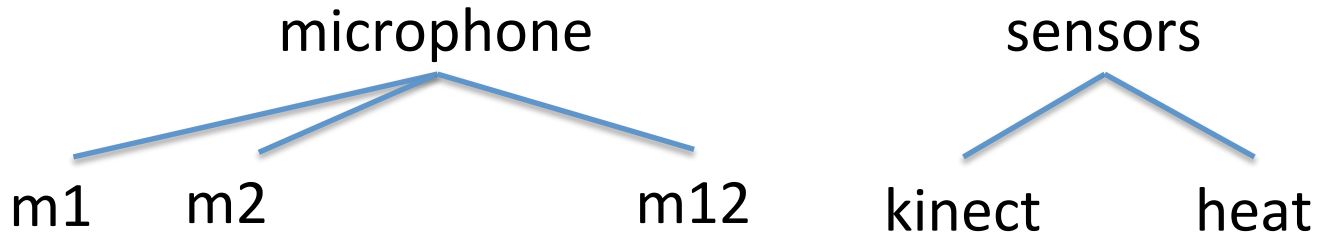
For simple systems you can directly send messages only

OSC Messages

- **You have to write your own protocol (message types)**
- Unless you follow someone else's protocol
- A typical OSC message
  - Starts with an OSC Address Pattern
  - followed by an OSC Type Tag String
  - and a corresponding number of OSC Arguments
- Types of messages are
  - **Int32:** 32-bit big-endian two's complement integer
  - **Float32:** 32-bit big-endian IEEE 754 floating point number
  - **OSC-string:** Sequence of ASCII characters finished by a null
  - **OSC-blob** An int32 size count, followed by that many 8-bit bytes of arbitrary binary data

OSC Messages

- **Can follow a taxonomic tree !**
- /sensors/kinect/head/pos
- /sensors/kinect/legs/axis
- /sensors/heat/temperature
- /microphone_array/m7/receive



Creating our own protocol

- Based on UDP and OSC
- Let's control a **3D rendering system based on kinect** live tracking (stream) **and microphone** events
- Three type of messages:
  `/stream` continuously data stream
  `/event` event-related data including tags
  `/text` text based control messages
- Each message has sender id and timestamp in order to synchronize messages
- Semantic Items are wrapped as `/evnt` messages to interact with the Framework

Specification

Pattern	Arguments	Type	
/stream	Id	String	Identification
	Time	Float32	Timestamp in seconds
	Sr	Float32	Sample rate
	Dim	Int32	Number of parallel streams
	Data	Blob	Byte array with data (rowwise)
/event	Id	String	Identification
	Time	Float32	Timestamp in seconds
	Duration	Float32	Duration
	Num	Int32	Number of events
	EventTag_1	String	Tag of first event
	EventValue_1 …	Float32	Value of first event
	EventTag_num	String	Tag of last event
	EventValue_num	Float32	Value of last event
/text	Id	String	Identification
	Time	Float32	Timestamp in seconds
	Message	String	Message content

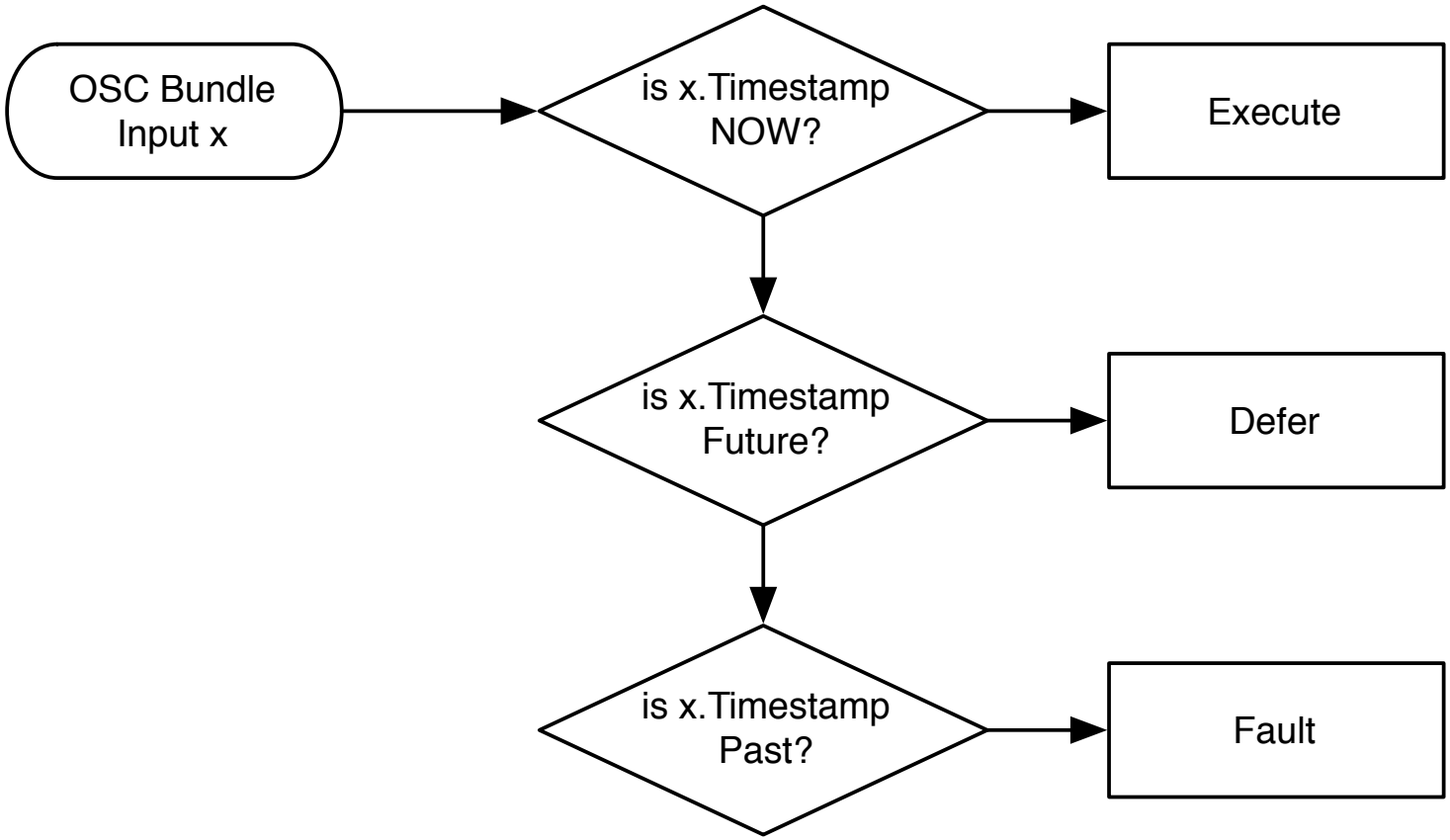
Stream example

- Sending head position as pair of float32-values sampled at 10 fps, e.g. 0.5s of data (= 5 frames): `X 20 20 14 11 10`
  `Y 10 10 11 9 9`
- Message:
  `/stream "head_track" 2.0 10.0 2 <data>`
- Where `<data>` has 2x4x5 bytes:
  `{40, [20 10 20 10 14 11 11 9 10 9]}`
- Receiver has to know data type (here: int)!

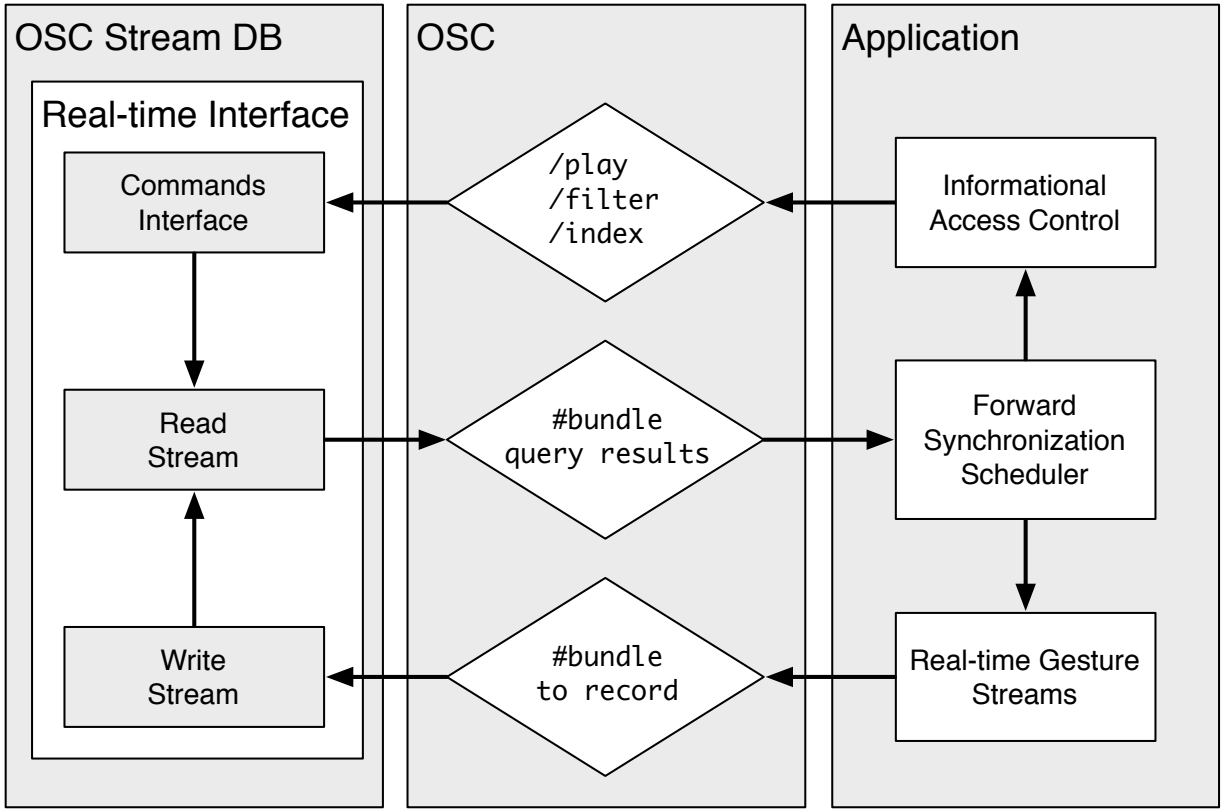
Event example

- Sending speech event in interval [3.0 5.5]s
- Message:
  `/event "speech" 3.0 2.5 0`
- Sending detected key words and detected emotion in same interval
- Message:
  `/event "key_words" 3.0 2.5 3 "I" 0.7 "feel" 0.4 "good" 0.9`
  `/event "emo_voice" 3.0 2.5 1 "happy", 1.0`

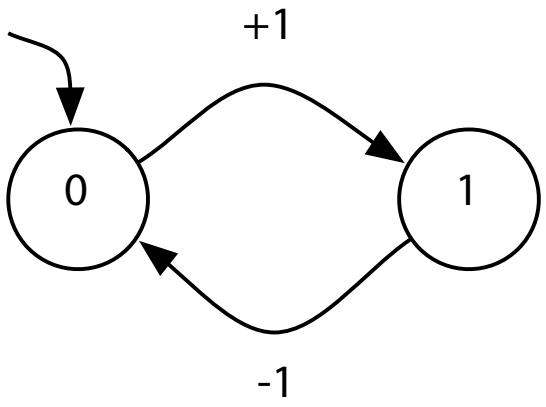
OSC vs. MIDI


OSC Bundle
Input x
is x.Timestamp
NOW?
Execute
is x.Timestamp
Future?
Defer
is x.Timestamp
Past?
Fault

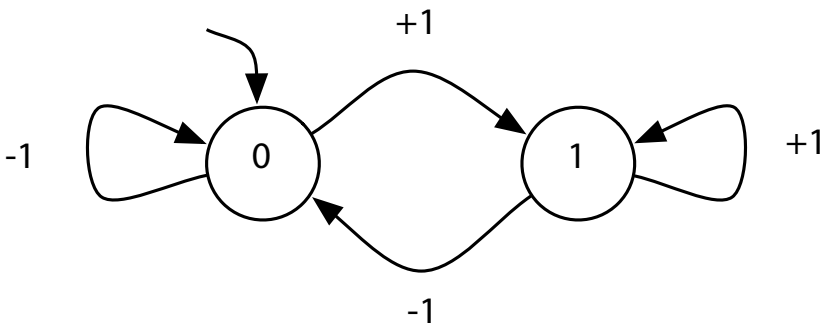
OSC example of DB application



OSC vs. MIDI

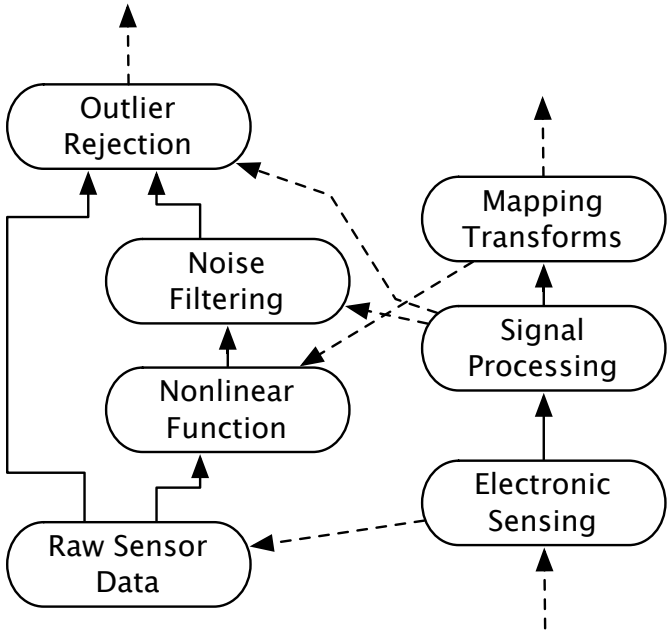


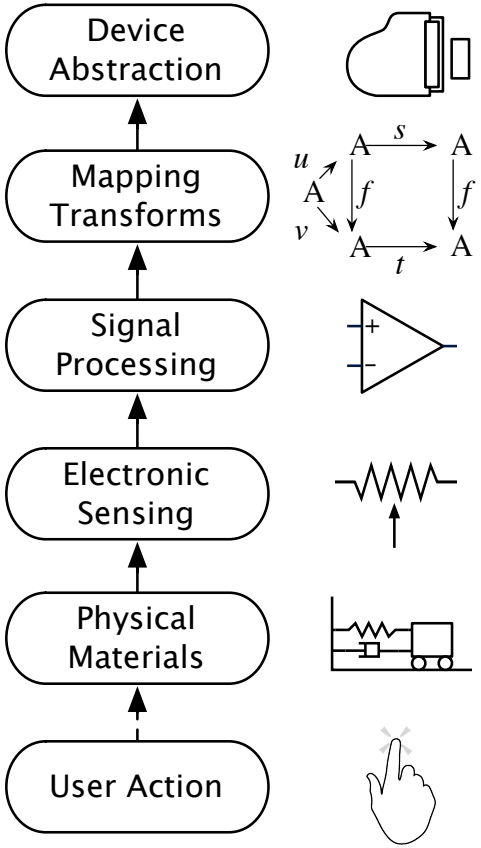
- MIDI can be modeled as a **state-based machine**
- The typical Note OFF problem
- Makes it an automata



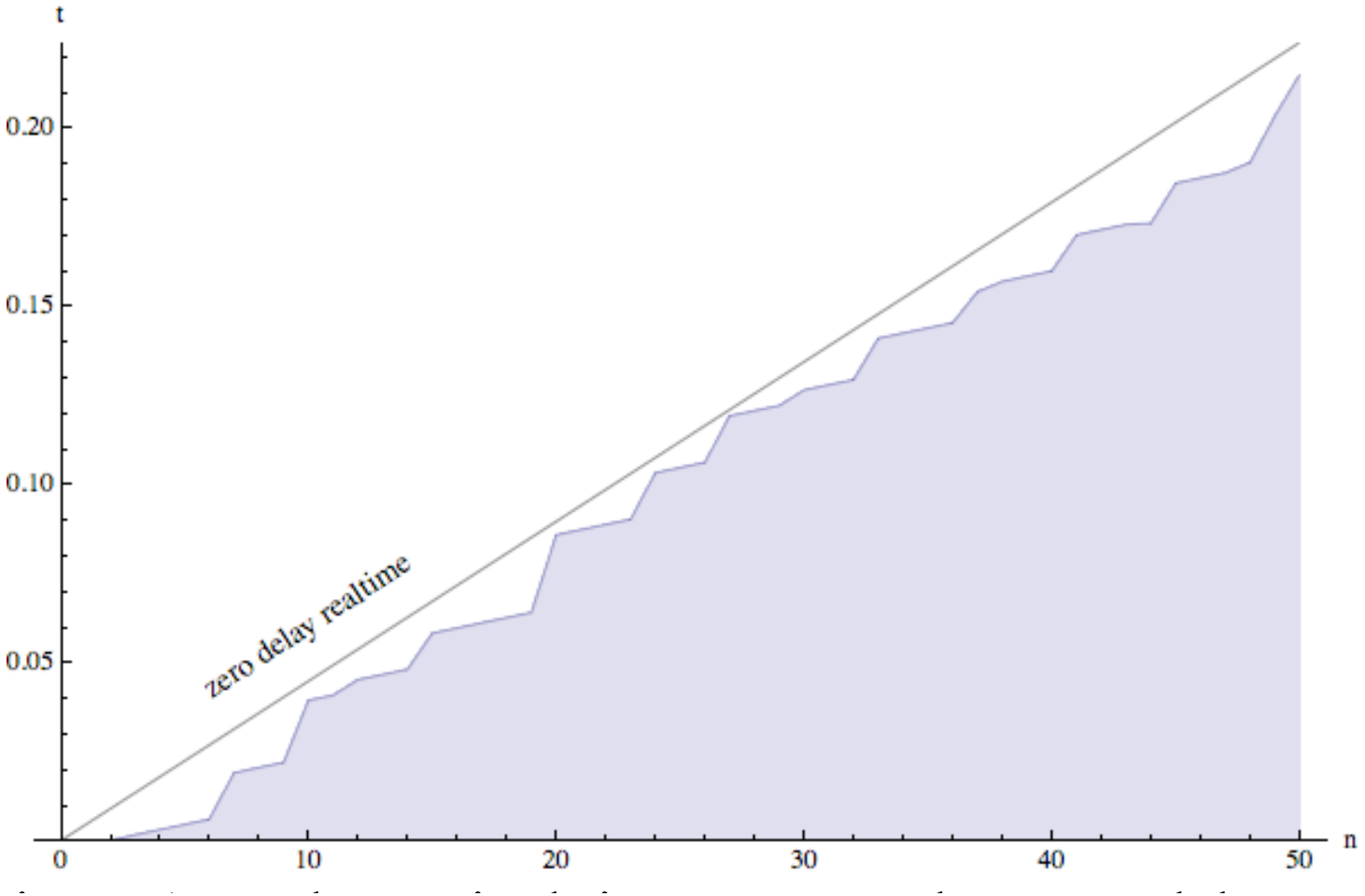
- OSC implicitly solves this problem by being independent message-based
- … But state-based machine can be imposed at will

OSC complete control

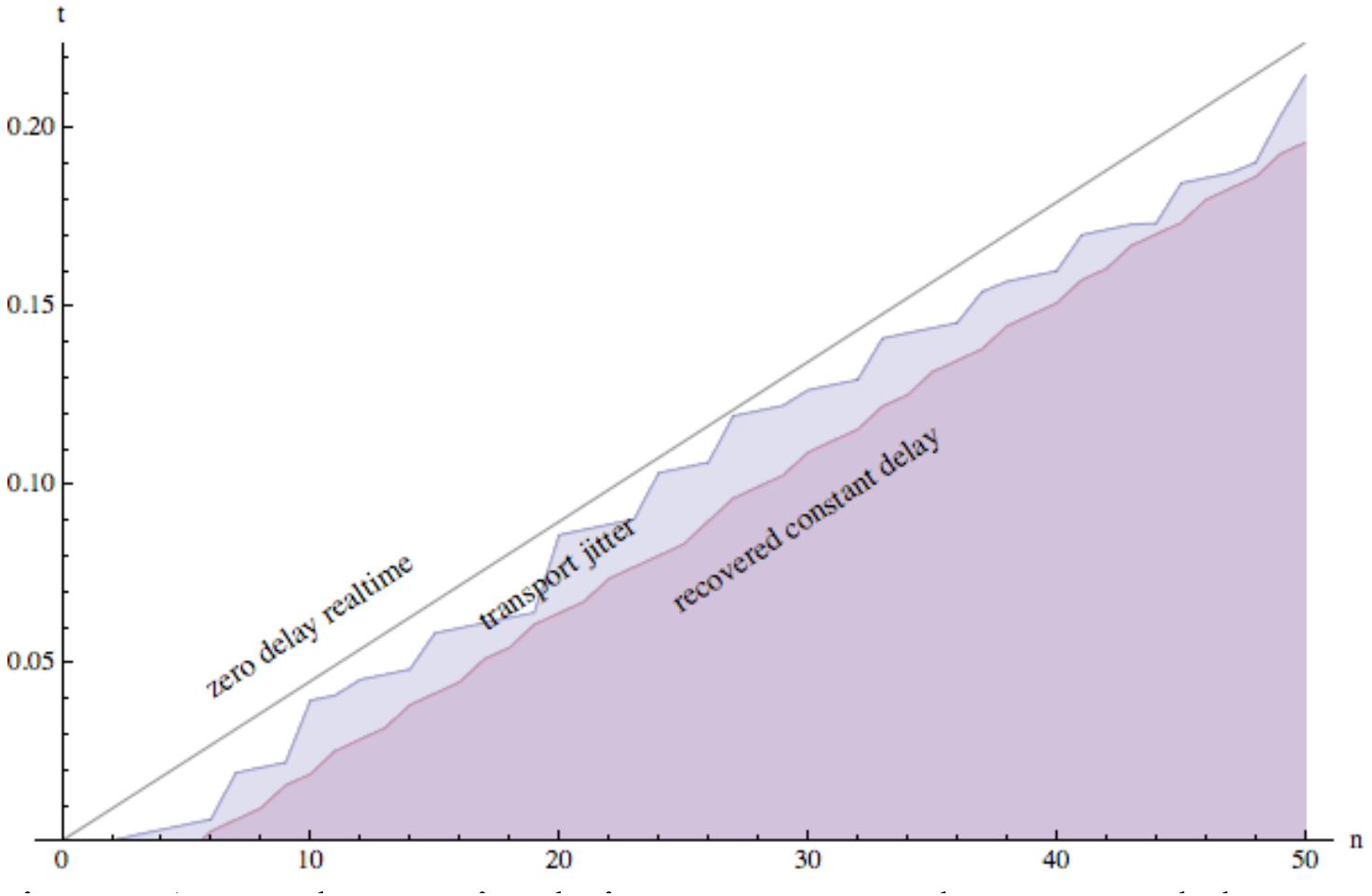




OSC clock problems



OSC clock problems … solved !



Using OSC in real-life

- OSC is already pre-implemented for lots of languages (and hardware devices as well)
  - Csound
  - **Matlab : http://sourceforge.net/projects/oscmex/**
  - PhP
  - Java / Swing
  - Arduino / UDOO
  - iPhone
  - Monome
  - Chuck
  - Python
  - **PureData : http://puredata.info/Members/martinrp/OSCobjects/**
  - **MaxMsp : http://cnmat.berkeley.edu/patch/4059**
  - Scala
  - Delphi
  - Ruby
  - […]

Using OSC in real-life

- OSC is already pre-implemented for lots of languages (and hardware devices as well)
  - Csound
  - **Matlab : http://sourceforge.net/projects/oscmex/**
  - PhP
  - Java / Swing
  - Arduino / UDOO
  - iPhone
  - Monome
  - Chuck
  - Python
  - **PureData : http://puredata.info/Members/martinrp/OSCobjects/**
  - **MaxMsp : http://cnmat.berkeley.edu/patch/4059**
  - Scala
  - Delphi
  - Ruby
  - […]

Pragmatic examples that **we will see now**

The Matlab implementation

Everything can be done … with only 6 functions ☺

- **osc_new_server** => Create an OSC server (create listen stream)
- **osc_free_server** => Free the server
- **osc_new_address** => Create an OSC client (connects to server)
- **osc_free_address** => Free the client
- **osc_recv** => Receive a message from the stream
- **osc_send** => Send a message to the stream

As always in OSC, need to think of two parts of the implementation

- **Server side** : Awaits for client by listening and responds to requests
- **Client side** : Connects to a server and then communicate

The Matlab server

```
% creating a server -- just specify what port to listen on
s = osc_new_server(3334)

% when receiving, you can specify a timeout in seconds
% a timeout of zero can be used for non-blocking operation
m = osc_recv(s, 10.0)

 % check to see if anything is there...
  if length(m) > 0
    m{1}.path          % the address of the first message..
    m{1}.data          % and its data part
    m{length(m)}       % the last message....
  end
end

osc_free_server(s);
osc_free_address(a);
```

The Matlab client

```
% an osc_address is a destination for packets -- IP or domain-name and port
a = osc_new_address('127.0.0.1', 3334)

% an osc message is a struct in matlab with two fields, path and data.
% and types are inferred automatically from the native matlab type !
m = struct( ...
      'path', '/foobar', ...
      'tt', 'ifLs', ...
      'data', {{1, 3.14159, 1, 'hello world'}})

% send the message...
osc_send(a, m)

% multiple messages can be sent at the same time (as a "#bundle")
err = osc_send(a, {m,m,m})

% when we are done with the address it can be freed
osc_free_address(a)
```

The MaxMSP … even simpler !