Advanced Network Programming (ANP)
XB_0048

Introduction

Animesh Trivedi
Autumn 2020, Period 1

Expectations...

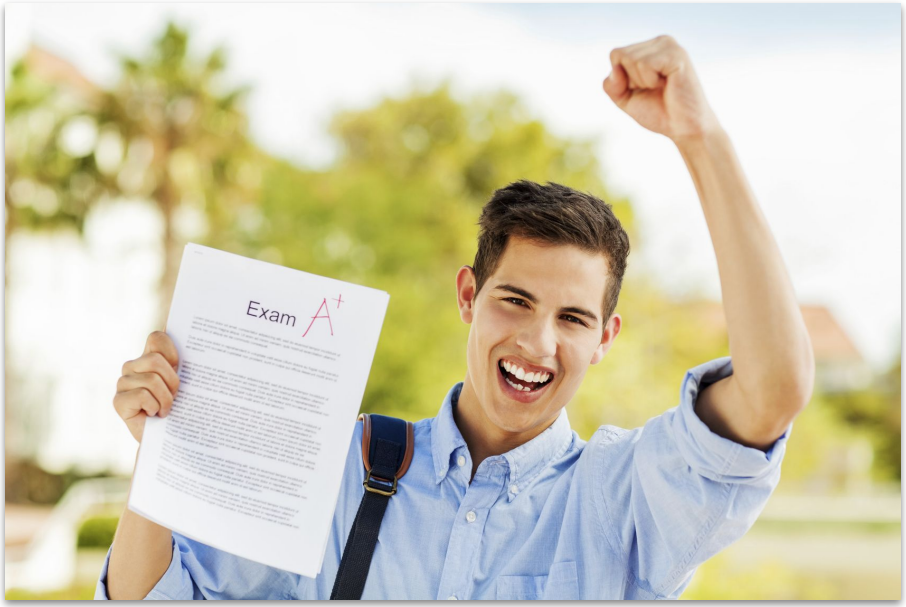
This course builds on prior knowledge from multiple courses. So please refresh your knowledge of

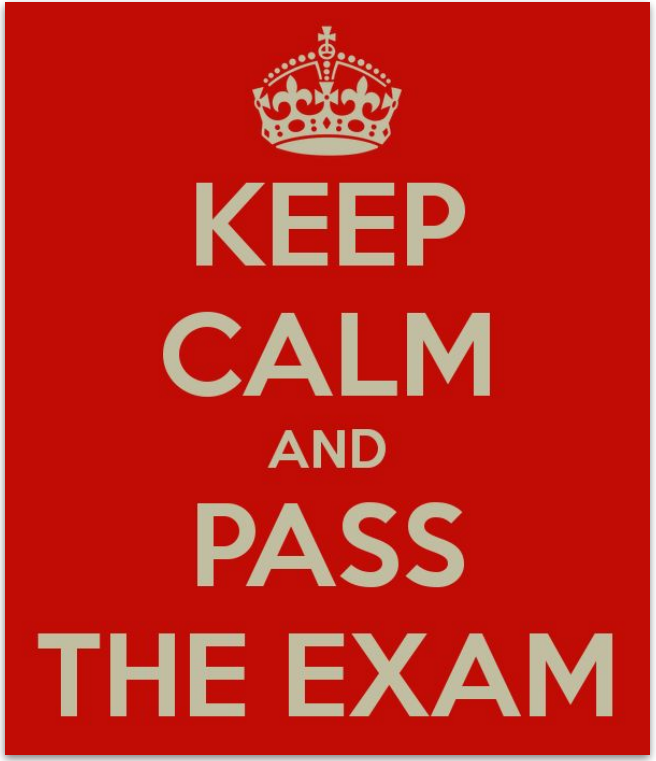
- **Computer Organization (XB_40009) :** CPU, devices, interrupts, memory architecture
- **Operating Systems (X_405067):** Kernel and userspace, processes, synchronization
- **Computer Networks (X_400487):** Protocols, Layer models, TCP/IP basic
- **Programming (XB_40011):** knowledge of C/C++

Please refresh your knowledge of these topics, or consult course slides, and online resources.

Why you should care about networking

Obviously, you need to pass your exams !
but...

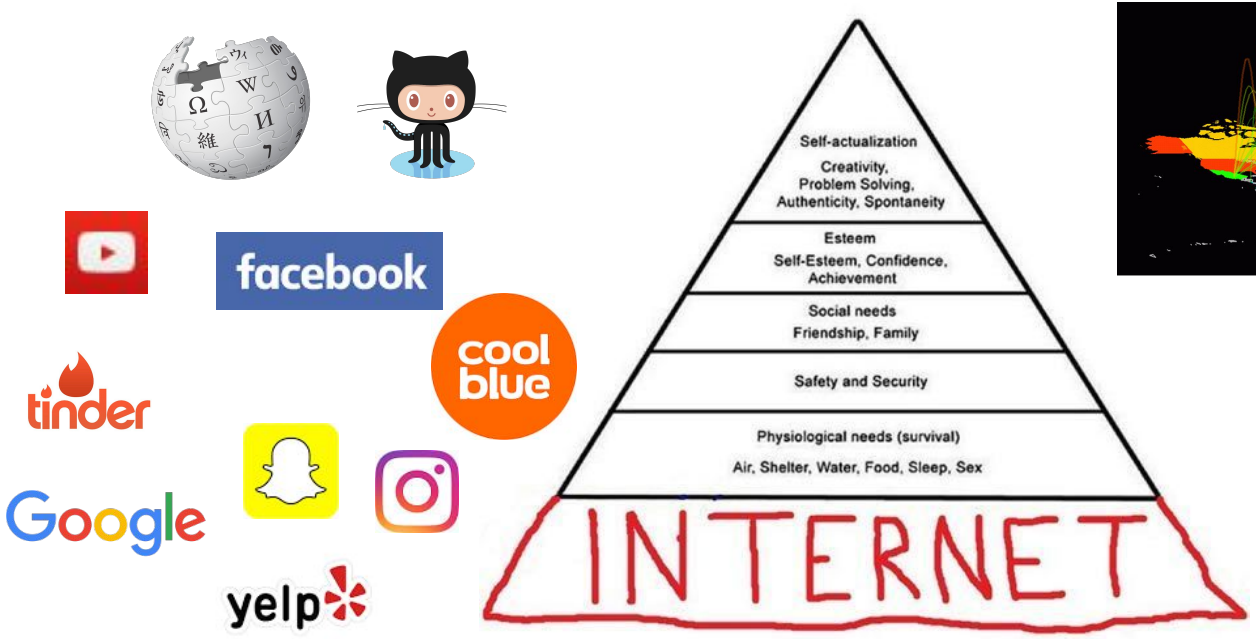


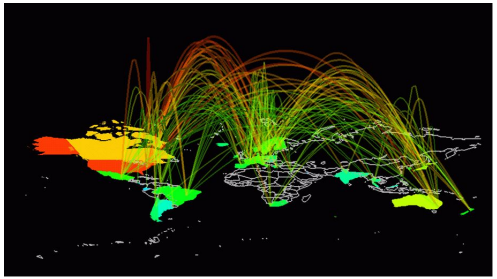


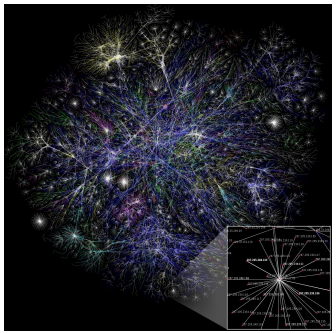
Why you should care about networking

1. Printing press, 1430s
2. Electricity, late 19th century
3. Penicillin, 1928
4. Semiconductor electronics, 1950s
5. Optical lenses, 13th century
6. Paper, second century
7. Internal combustion engine, ~1860
8. Vaccination, 1796
9. **The Internet, 1960s**
10. Steam engine, 1712

Why care about networking - Personal

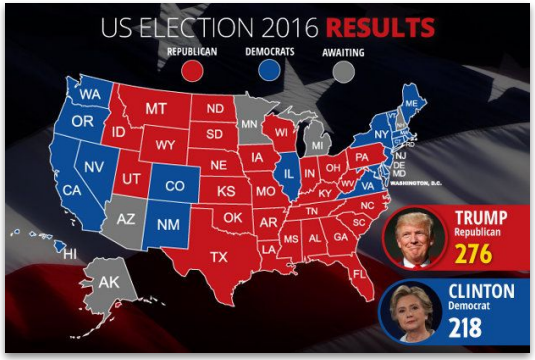


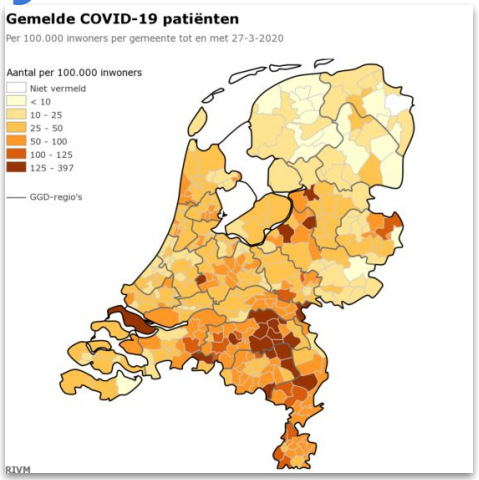




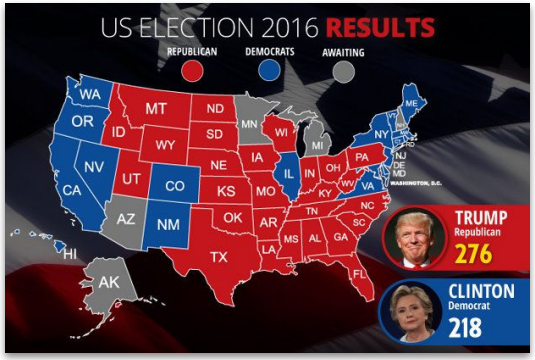
We live in an interconnected world - essential for survival !

Why care about networking - Society



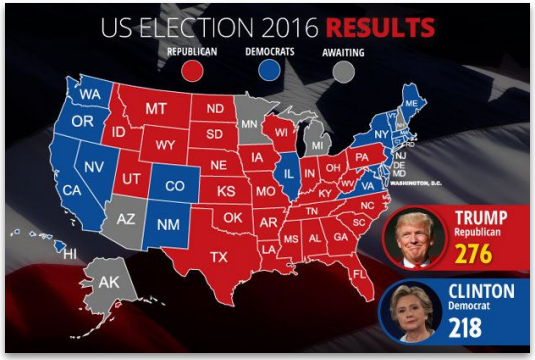


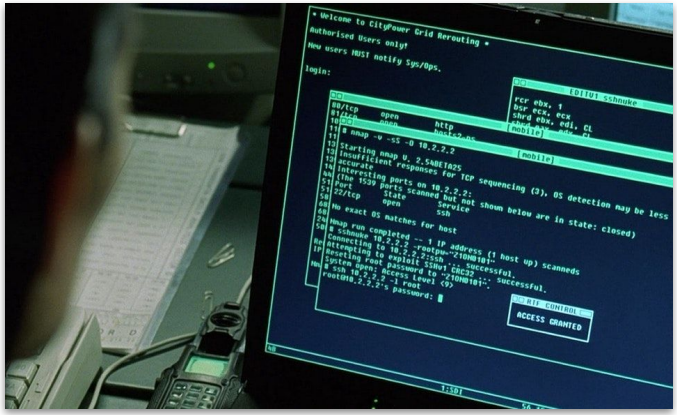
Why care about networking - Society



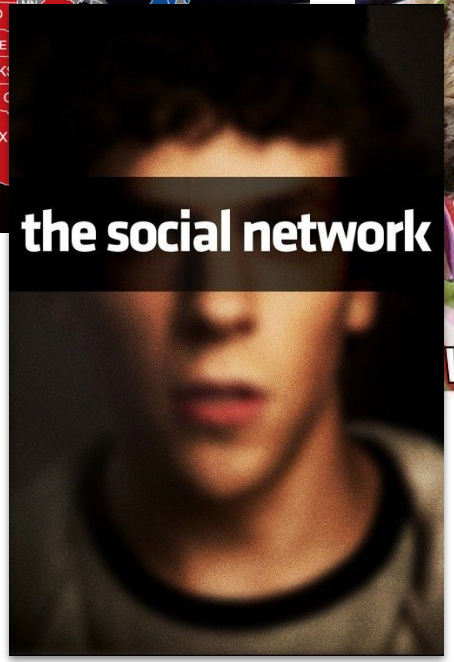
https://cheezburger.com/8763535360

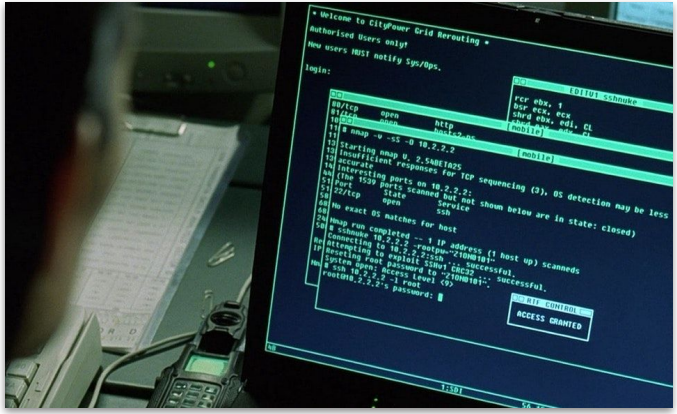
Why care about networking - Society





Why care about networking - Society





nicira

BAREFOOT NETWORKS

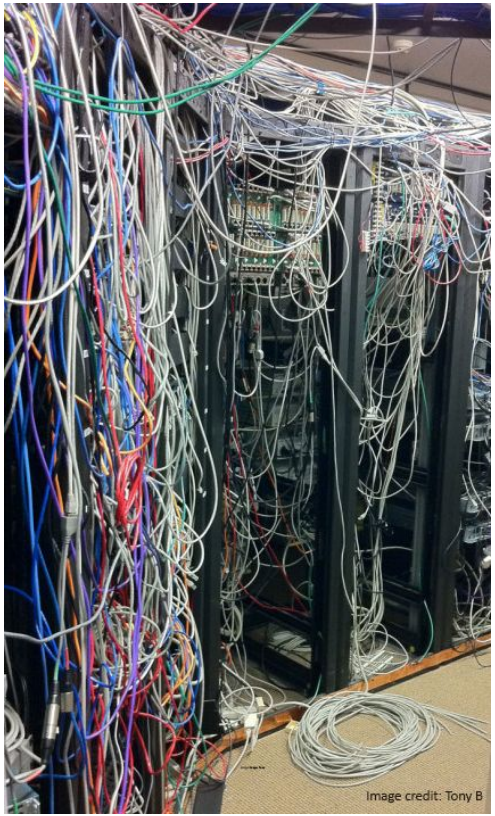
$$$

What is this course about

- Learn about **low-level** networking internals
  - What happens when you call `send(data)`
  - Design and code a “real” stack

- **Part 1:** Challenges with end host networking - Animesh
  - Stalled CPU, 100+ Gbps networking
  - Cutting-edge research

- **Part 2:** Challenges inside data centers - Lin
  - How to manage a network of 1M servers

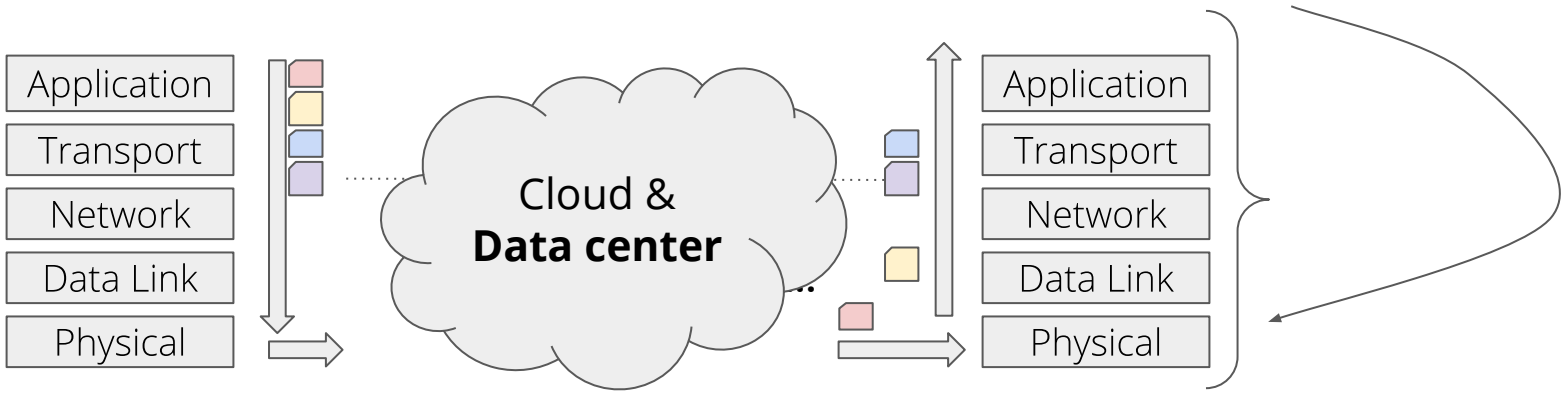


https://www.techrepublic.com/blog/10-things/10-stupid-things-people-do-in-their-data-centers/

Part 1 and part 2

Part 1: End-host networking Stack

How do you build and process network data?

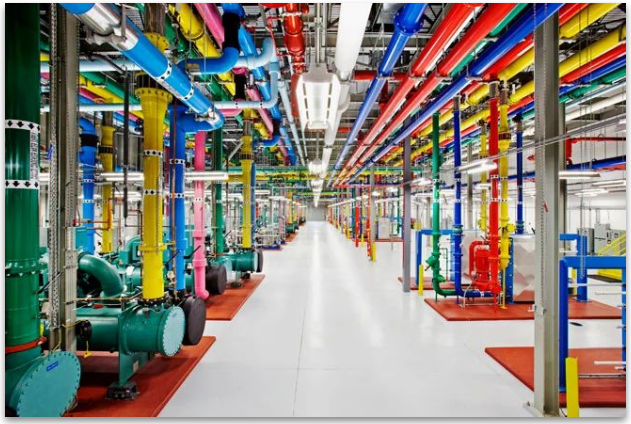


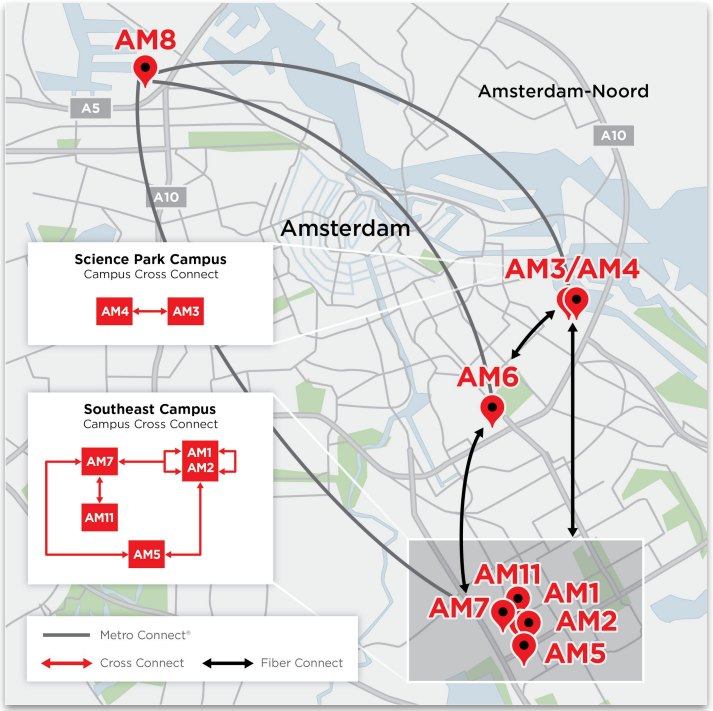
Part 2: Network Infrastructure

What happens when a packet leaves your computer?

What are data centers?

- Large installation of servers in one place
- Connected with high-performance networks
- Efficient cooling and power delivery

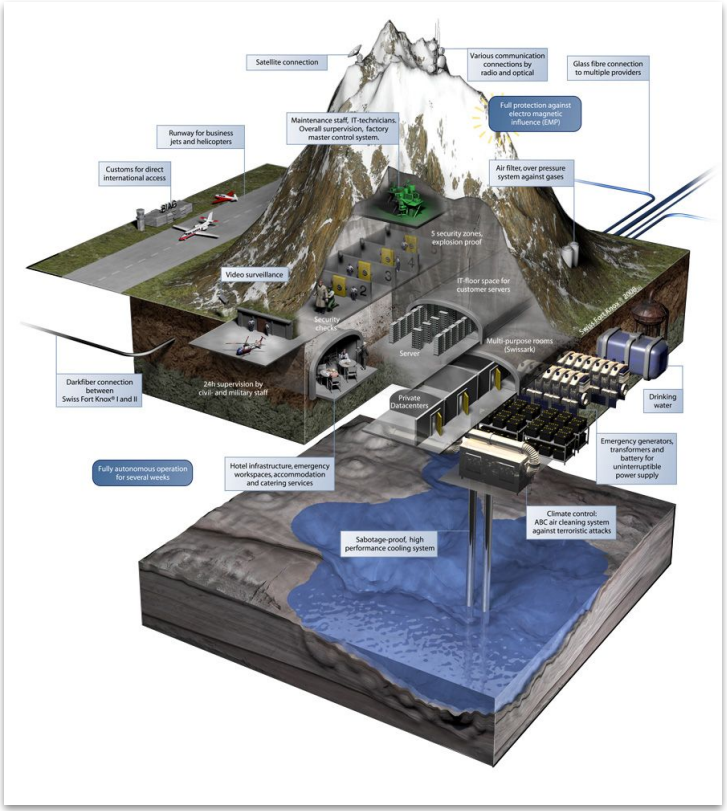




https://www.google.com/about/datacenters/

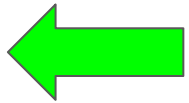
https://www.equinix.nl/locations/netherlands-colocation/amsterdam-data-centers/

There is one in the mountains in Switzerland ;)



https://www.swissdatabackup.ch/en/mount10/swiss-fort-knox/
https://websitehostreview.com/10-most-incredible-data-centers-on-earth/

Layout of upcoming lectures - Part 1

Sep 1st, 2020 (today): *Introduction and networking concepts* 

Sep 3rd, 2020 (this Tuesday): *Networking concepts (continued)*

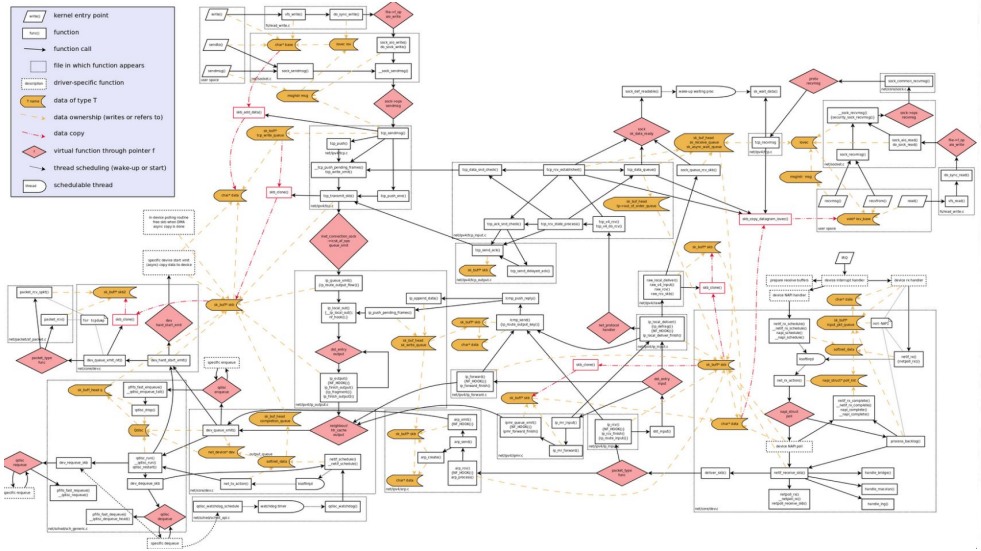
Sep 8th, 2020 : *Linux networking internals*

Sep 10th 2020: *Multicore scalability*

Sep 15th 2020: *Userspace networking stacks*

Sep 17th 2020: *Introduction to RDMA networking*

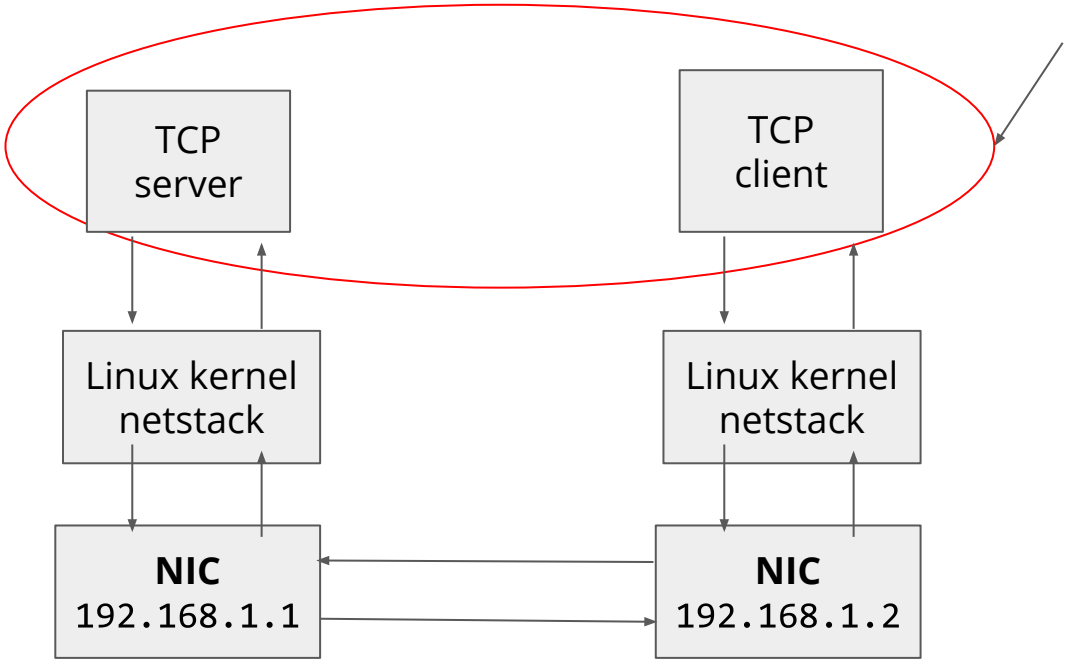
Project: Build your own networking stack!



https://i.pinimg.com/originals/30/5b/fe/305bfea090b95b94218d9892aefc7e88.png
https://wiki.nix-pro.com/view/Packet_journey_through_Linux_kernel

https://www.networkcomputing.com/data-centers/moving-stack

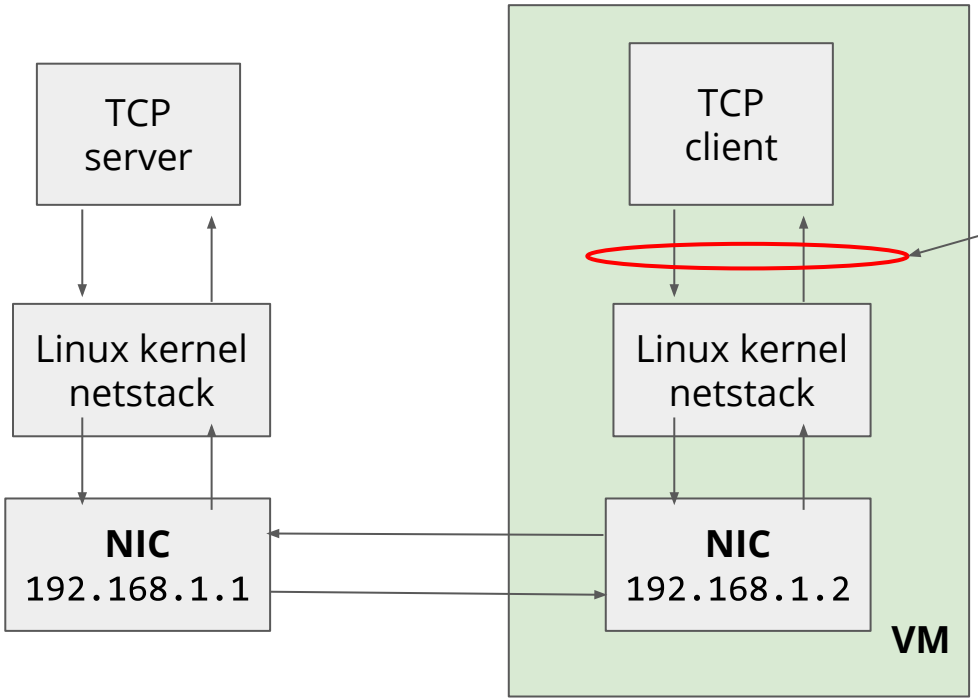
ANP netstack project overview



Basic server client example given

1. Client connect to the TCP server
2. Sends a buffer with a predefined pattern
3. The server receives the buffer and checks the pattern
4. The servers sends the same buffer back to the client
5. The client receives the buffer and checks the pattern - they _must_ match
6. Close the connection from the client side

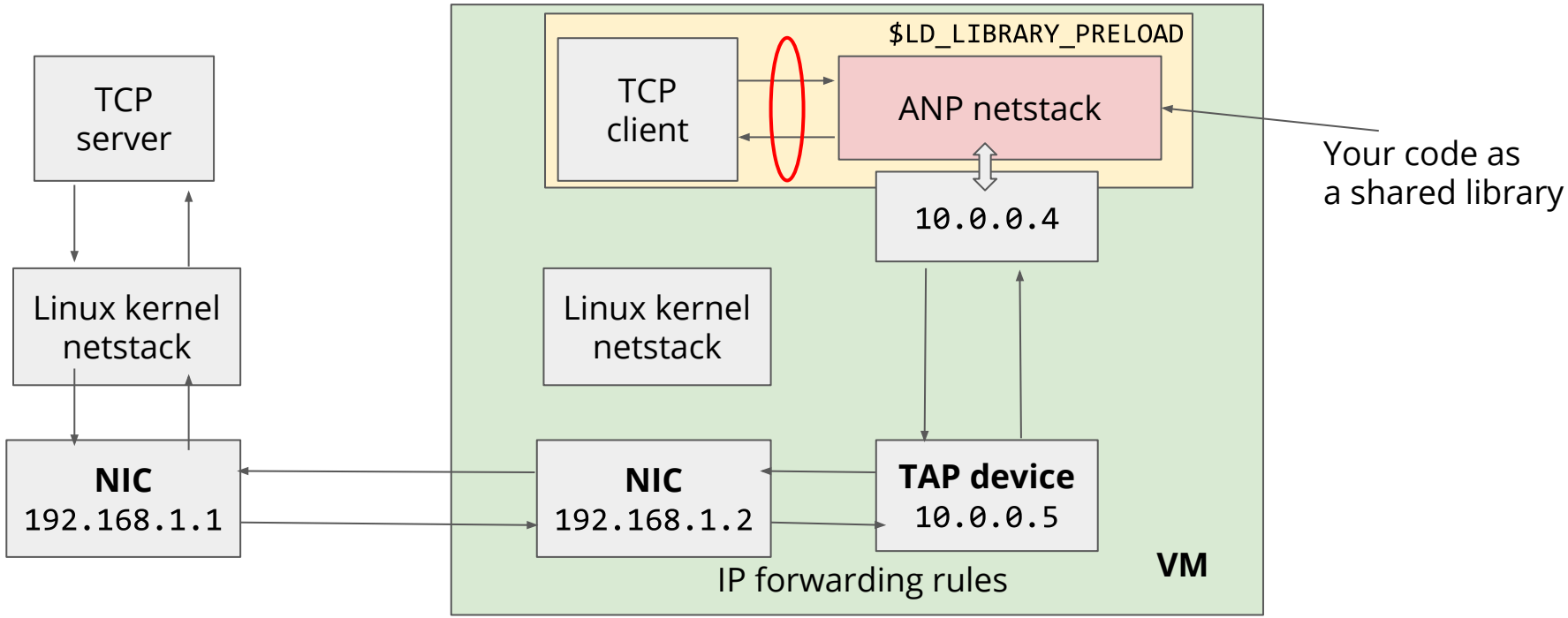
ANP netstack project overview



The BSD socket API

1. `socket()` : expects a file descriptor
2. `connect()` : TCP 3-way handshake
3. `send()` : TCP transmission, ACKs
4. `recv()` : TCP reception, ACKs
5. `close()` : 3-way shutdown

ANP netstack project overview



Why we chose to build this way?

- Develop assignment in the Linux kernel networking stack
  - Mature, battle tested over 30+ years
  - However, extremely complex and steep learning curve
  - You are encouraged to have a look whenever in doubt ;)

Why we chose to build this way?

- Develop assignment in the Linux kernel networking stack
  - Mature, battle tested over 30+ years
  - However, extremely complex and steep learning curve
  - You are encouraged to have a look whenever in doubt ;)

- Develop assignment in the userspace
  - Easy to develop, full flexibility (just another userspace program)
  - Needs boilerplate code, which we provide
  - **Not just a toy example -** *userspace networking stack, co-developed with an application is the way current networking research is conducted*
    - Completely customizable and can be co-developed (what does this mean becomes clear later)
  - Run an unmodified TCP server client application ← very important !

ANP project milestones

1. ***Welcome to the machine*** **(Tue, 8 Sep 2020 before the lecture) : canvas quiz**
   Individual: Get the given infrastructure up and run the arping command **(5 points)**

2. ***Hey you*** **(Tue, 15th Sep 2020) : canvas quiz**
   Individual: Implement the ICMP protocol - get the "ping" command working **(5 points)**

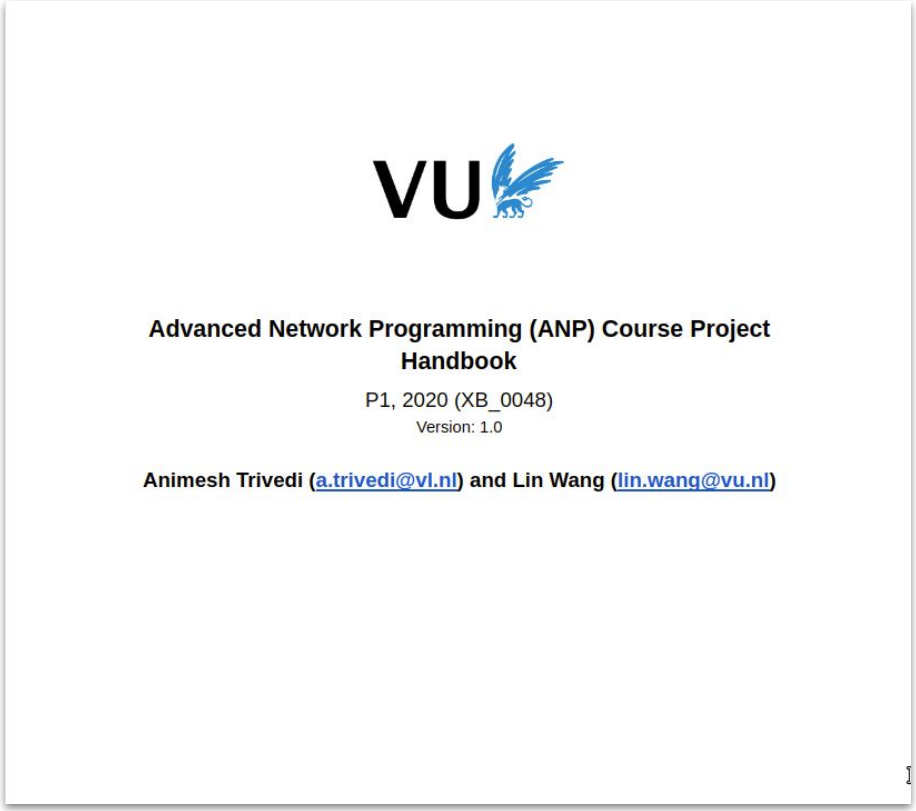
Group formation

3. ***Is anyone out there?*** **(Tue, 29th Sep 2020) : interview**
   Group: Establish the 3-way handshake with the TCP server **(15 points)**

4. ***Careful With That Data, Eugene*** **(Tue 13th Oct 2020) : interview**
   Group: Transmit data, receive data, and close the connection **(15 points)**

5. ***Another Graph in the Wall*** **(Tue Oct 20th 2020) : canvas submission**
   Group: design and run an experiment to measure latency profile **(10 points)**

On Canvas - read the project handbook



VU

Advanced Network Programming (ANP) Course Project Handbook

P1, 2020 (XB_0048)

Version: 1.0

Animesh Trivedi (a.trivedi@vl.nl) and Lin Wang (lin.wang@vu.nl)

Warning

This is an **experimental** course

- You are a part of the experiment
- We will build upon your feedback

This is a **coding-heavy** course

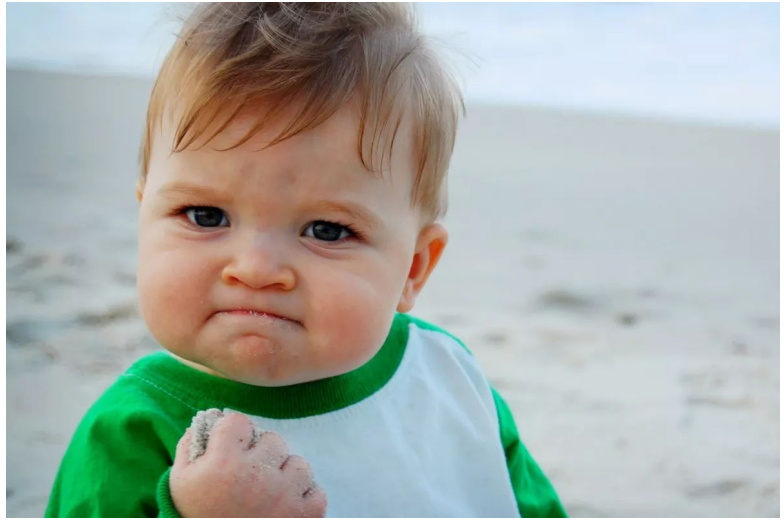
- If you have not done C/C++ programming this may be a tough course
  - *Use of structs, pointers, file I/O, thread synchronization, locks*
- Start coding early, there will be plenty of surprises

There's a relatively little flexibility with the deadlines (as they all dependent on each other)

https://www.redbubble.com/i/art-board-print/Coding-in-Progress-by-D3mon98/36978034.7Q6GI

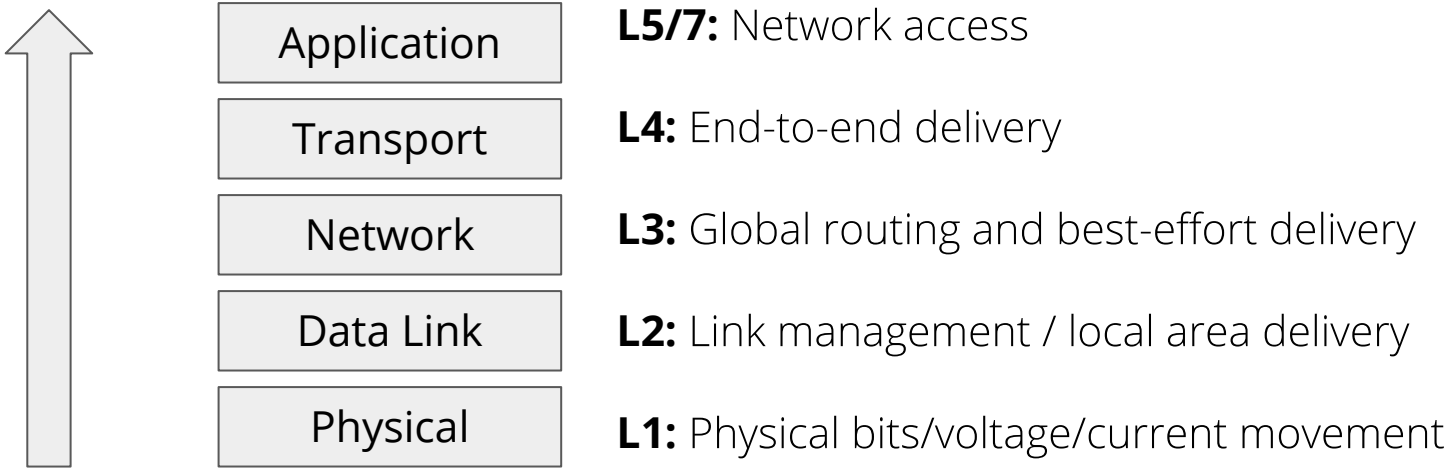
But equally rewarding

- See your stack in action when communicating with the standard Linux networking stack
- Build your own networking protocol
- Unlimited number of customization and bonuses possible
- Learn everything about (in)famous TCP/IP stack
- Solve crazy small challenges

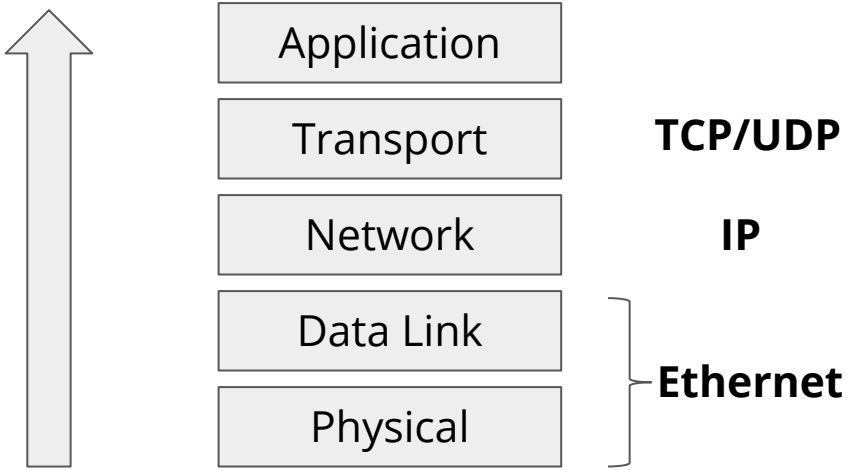


Recap: The layered model

Modularity: Layer by layer architecture where one layer provides service to the next one

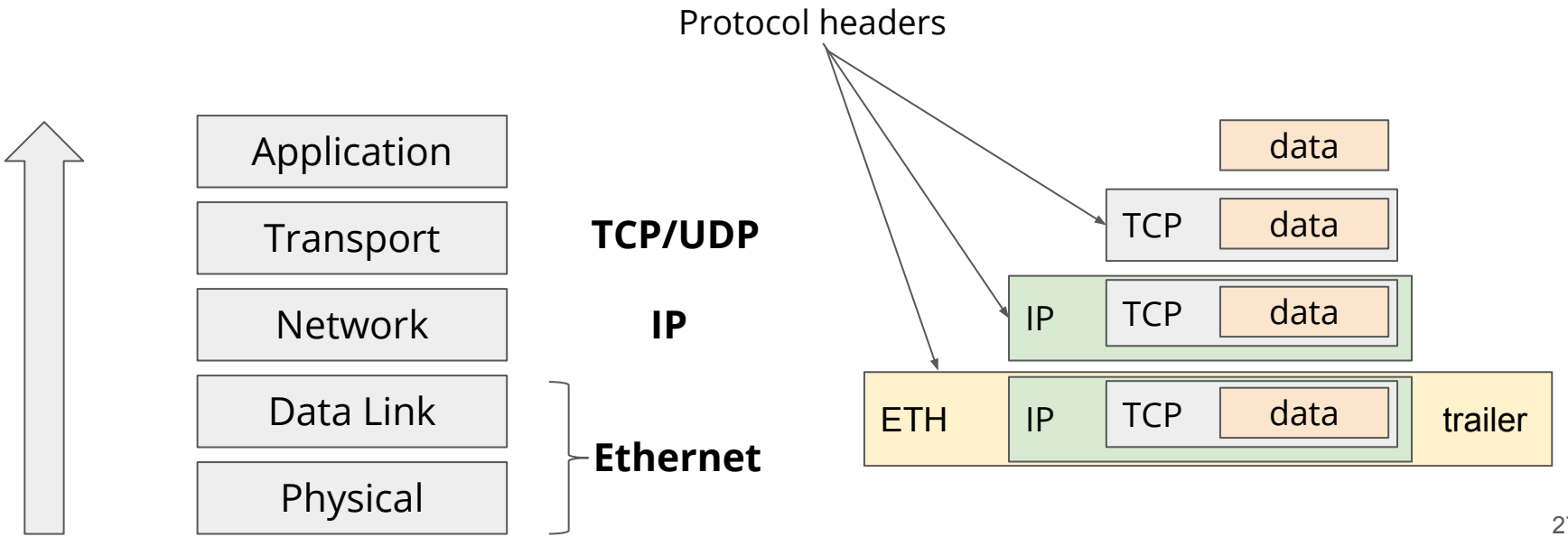


Recap: The layered model - Protocols



Some examples, but they are not the only one

The layered model - Protocols headers and encapsulation



How do applications use the network?

We use a network application programming interface (or API)

One example is a **Socket interface**

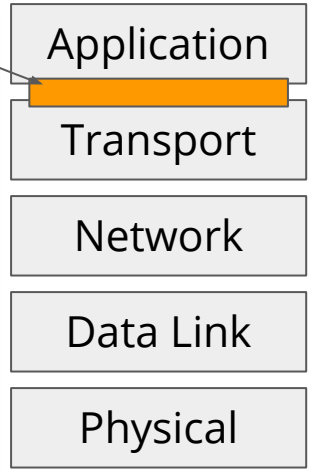
```
int sock = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in serv_addr;
[...]
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);
[...]
connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr);
// send data
send(sock , “Hello World!”, 12, …);
//recv data
recv(sock, buffer, 1024);
```

How do applications use the network?

We use a network application programming interface (or API)

One example is a **Socket interface**

```
int sock = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in serv_addr;
[...]
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);
[...]
connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr);
// send data
send(sock , “Hello World!”, 12, …);
//recv data
recv(sock, buffer, 1024);
```

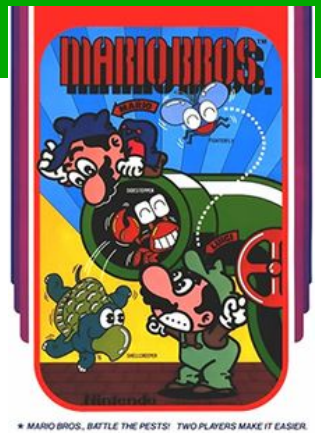


History : Socket interface

- One of the first implementations in the 4.2BSD Unix (1983)

1983!


MICROSOFT®
Microsoft Word
Version 1.15
Microsoft Word:

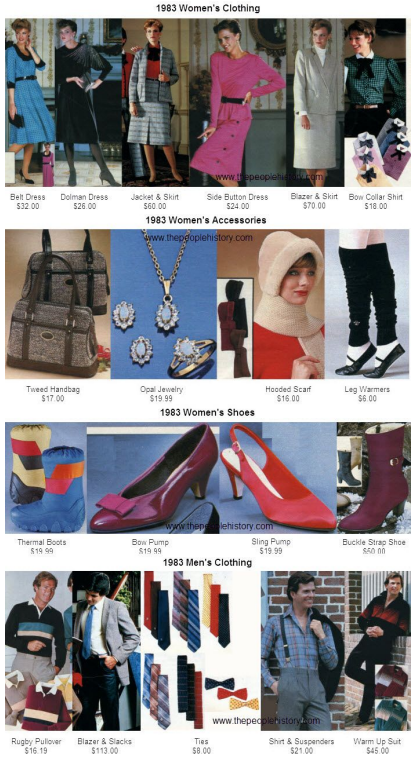

MARIO BROS.
Nintendo
★ MARIO BROS., BATTLE THE PESTS! TWO PLAYERS MAKE IT EASIER.


1983 Women's Clothing
www.thepeoplehistory.com
Belt Dress $32.00
Dolman Dress $26.00
Jacket & Skirt $60.00
Side Button Dress $24.00
Blazer & Skirt $70.00
Bow Collar Shirt $18.00
1983 Women's Accessories
www.thepeoplehistory.com
Tweed Handbag $17.00
Opal Jewelry $19.99
Hooded Scarf $16.00
Leg Warmers $6.00
1983 Women's Shoes
www.thepeoplehistory.com
Thermal Boots $19.99
Bow Pump $19.99
Sling Pump $19.99
Buckle Strap Shoe $60.00
1983 Men's Clothing
www.thepeoplehistory.com
Rugby Pullover $16.19
Blazer & Slacks $113.00
Ties $8.00
Shirt & Suspenders $21.00
Warm Up Suit $45.00

History : Socket interface

- One of the first implementations in the 4.2BSD Unix (1983)
  - It is 36 years old
  - In 1983: Microsoft Word is first released
  - In 1983: Mario Bros. was first released as a Nintendo arcade game
  - In 1983: First mobile phones from Motorola

Modern derivatives: WinSock, BSD socket, POSIX socket, more



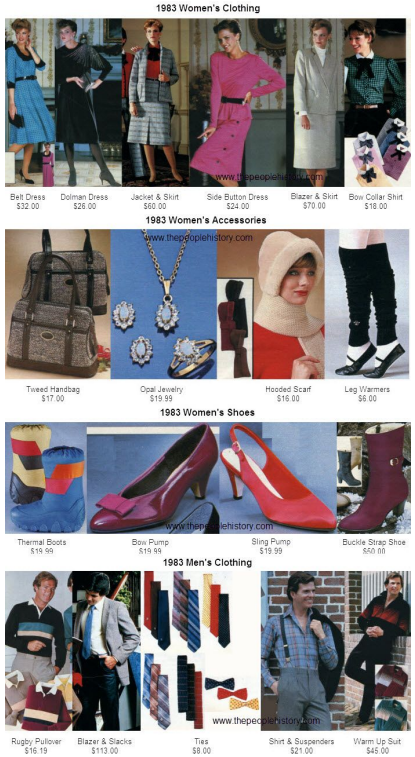
History : Socket interface

- One of the first implementations in the 4.2BSD Unix (1983)
  - It is 36 years old
  - In 1983: Microsoft Word is first released
  - In 1983: Mario Bros. was first released as a Nintendo arcade game
  - In 1983: First mobile phones from Motorola

  Modern derivatives: WinSock, BSD socket, POSIX socket, more

- First reference RFC #147 (`The Definition of a Socket, 1971`) https://tools.ietf.org/html/rfc147 (only 2 pages)

- Follows the UNIX philosophy
  - *Everything is a file* (a socket is a file descriptor)



History : Socket interface

- One of the first implementations in the 4.2BSD
  - It is 36 years old
  - In 1983: Microsoft Word is first released
  - In 1983: Mario Bros. was first released as a N
  - In 1983: First mobile phones from Motor

  Modern derivatives: WinSock, BSD
- First reference RFC #147 (`The Definition of a Soc`
  `https://tools.ietf.org/html/rfc147` `(only 2 pages)`
- Follows the UNIX philosophy
  - *Everything is a file* (a socket is a file descriptor)

What is an RFC? Request for Comments

is a publication from the Internet Society (ISOC)/ the Internet Engineering Task Force (IETF)

About any number of topics: protocols, behavior, semantics, tutorials, do and don't...and poems (968), bizarre protocols (see: The Infinite Monkey Protocol Suite (IMPS), 2795)

They are identified by numbers

TCP (**793**), ICMP (792), IP (791), UDP (768), ARP (826), DNS (1034), HTTP(2068)

https://en.wikipedia.org/wiki/List_of_RFCs
https://tangentsoft.net/rfcs/humorous.html

Quick recap: Socket API (see tutorial in Files in Canvas)

1. `int fd = socket(AF_INET, SOCK_STREAM, 0);`
    a. Returns a file descriptor as an integer
    b. File descriptor, a process-local integer to identify any open file or socket
    c. Unique within a process
2. `int ret = connect(fd, (struct sockaddr*)&server_addr, sizeof(server_addr));`
    a. Proceed to setup a connection with a server_addr and attach to “fd”
    b. For TCP, here runs the 3-way handshake protocol
3. `ret = send(fd, (void*) tx_buffer, (int) size, (int) flags);`
    a. Does data transmission operation
    b. Return values
        i. Less than zero: then there was error, check errno
        ii. More than zero, but less than size: only some part of data was accepted for TX
        iii. Equal to size: all of it was transmitted
4. Similarly for `recv`()
5. `Close`(fd): close the connection

Socket interface - The unofficial standard

- Setting up and managing connections
  - `socket()`, `bind()`, `listen()`, `connect()`, `accept()`, `close()`
- Network operations
  - `send()`, `recv()`, `sendto()`, and `recvfrom()` (or `write()` and `read()` may also be used)
- Address/hostname management
  - `gethostbyname()` and `gethostbyaddr()` to resolve IPv4 host names and addresses
- Select activity and readiness of a socket for I/O
  - `select()`, `poll()`
- Setting up extra options
  - `getsockopt()` and `setsockopt()`

Not a complete list. There are OS specific (e.g., Linux) specific extensions.

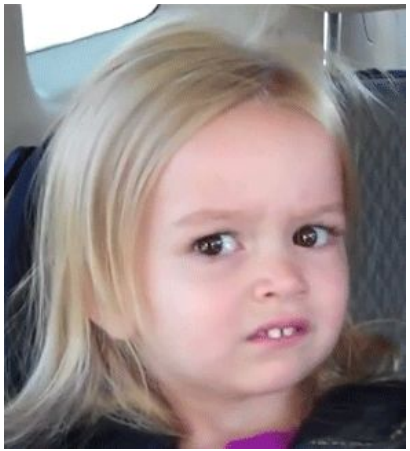
Socket - A highly successful abstraction

- Socket is a **very successful abstraction**
  - A UNIX file with a bunch of basic functions
  - Applications are shielded away from managing anything but just "*what to send*" and "*where to receive*"
    - `send(int socketfd, void* buffer_address, size_t length, int flags);`
    - `recv(int socketfd, void* buffer_address, size_t length, int flags);`

- Worked **extremely well** all these years supporting different classes of applications
  - Web servers, video streaming, messaging applications, _your_favorite_application

But wait...

- where are the rest of the networking layers?
- what happens after calling send / recv functions?
- who is running the TCP state machine?
- who is managing the TCP window and retransmission?
- who is doing IP routing?
- who is doing the MAC layer management?
- ...and so many more question



The answer is ...

The Operating System

- Linux, Windows, Open/Free/NetBSD, Minix - whatever you are running

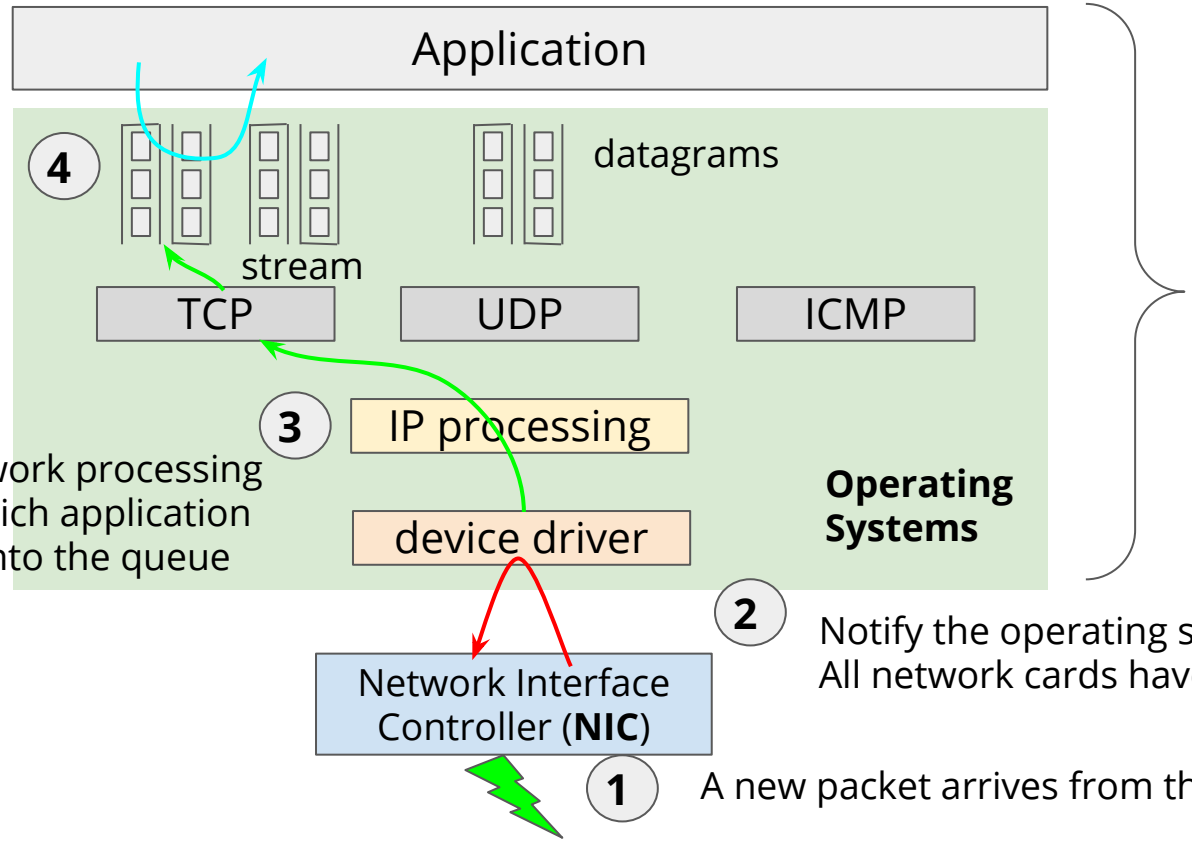


Why?

- Network connectivity is an important shared resource for all
- Every networking application benefits from a common implementation

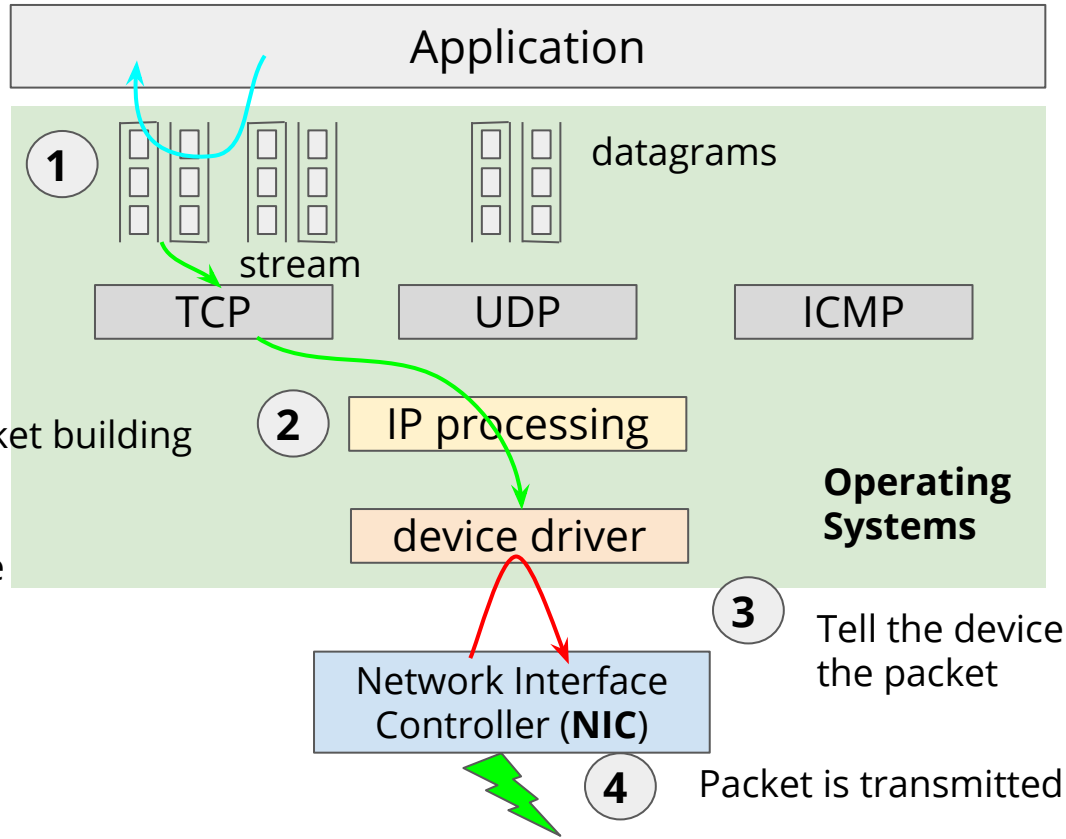
```
As we will see later, this is _NOT_ THE only way to arrange things
```

A packet's journey - (simplified) Receiving path



A packet's journey - (simplified) Sending path

Queue data when application calls send()



Perform data packet building

- TCP header
- IP header
- which device

Tell the device driver to transmit the packet

Packet is transmitted on the network

Still many unanswered questions here

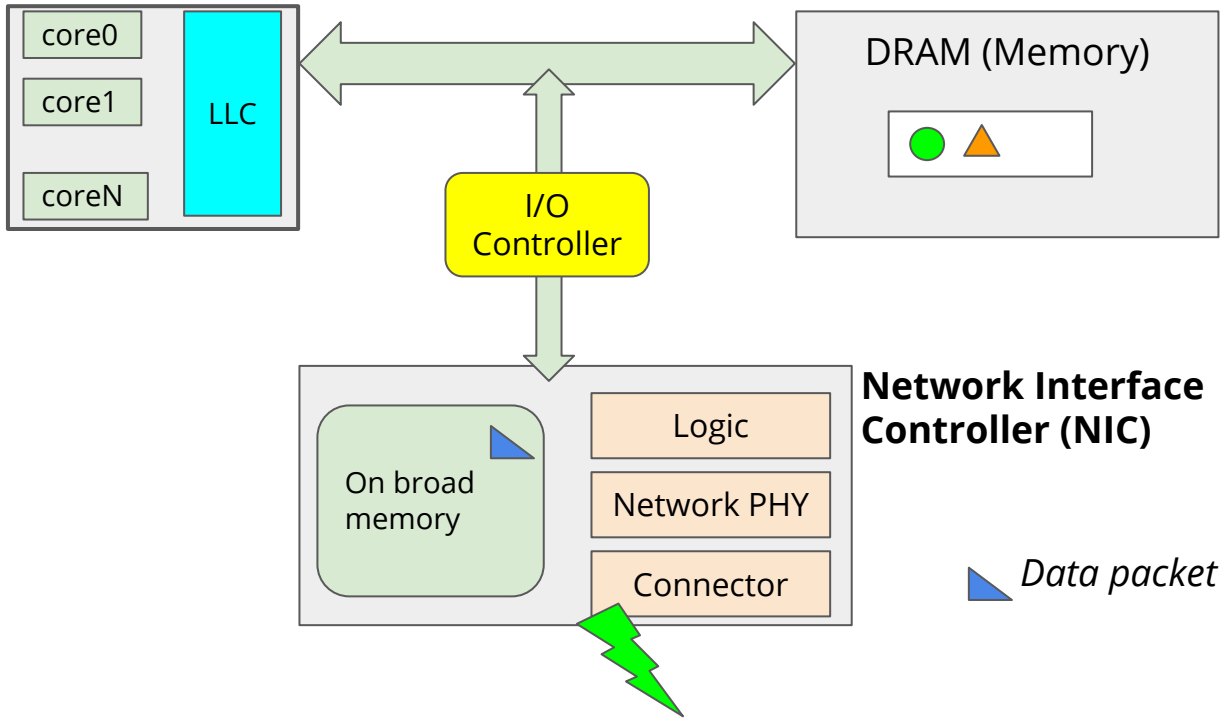
Think of the receive path. This is more complicated than the sending path (can you think of why?)

1. How to transfer data between a network controller and the end host
2. How to notify the end host about network packet reception
    a. Do you need to tell the end host about a packet transmission?
3. How to build a packet with multiple protocols and headers
4. How much time/steps it takes to receive data? 1 bytes, 1 kB, 1 MB, or 1 GB?
5. ...and many many many more questions.

Lets answer some of them, one by one and introduce the key ideas

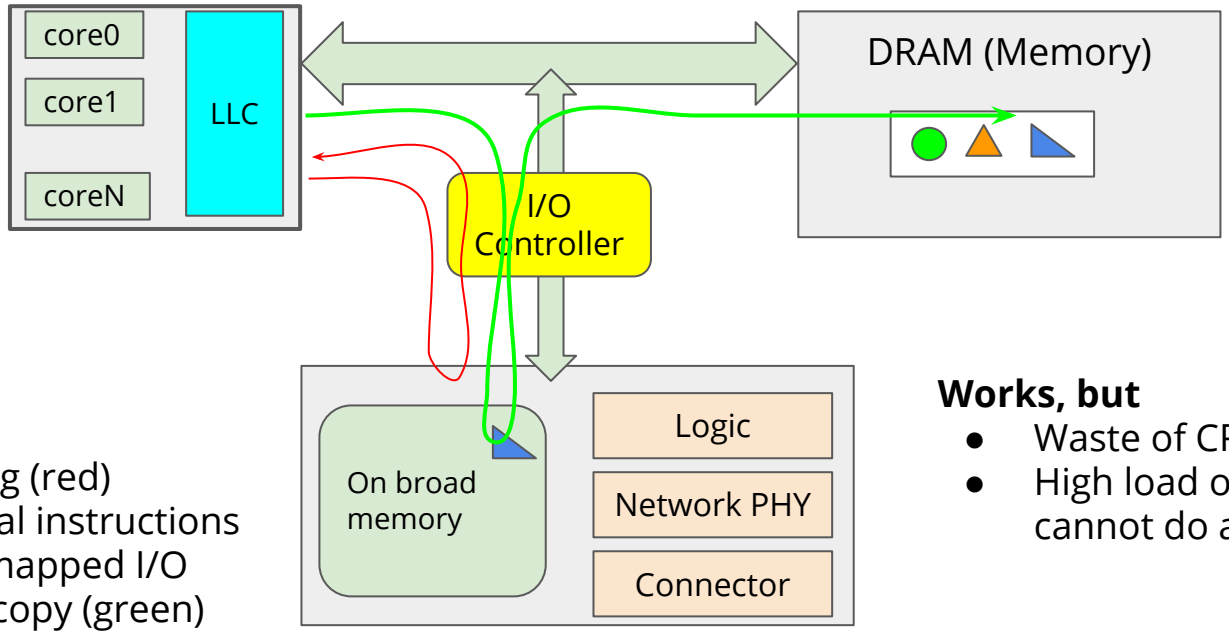
Transferring data between the end host and the NIC

Transferring data between NIC and end-Host



ideas?

Transferring data between NIC and end-Host



Strategy 1:

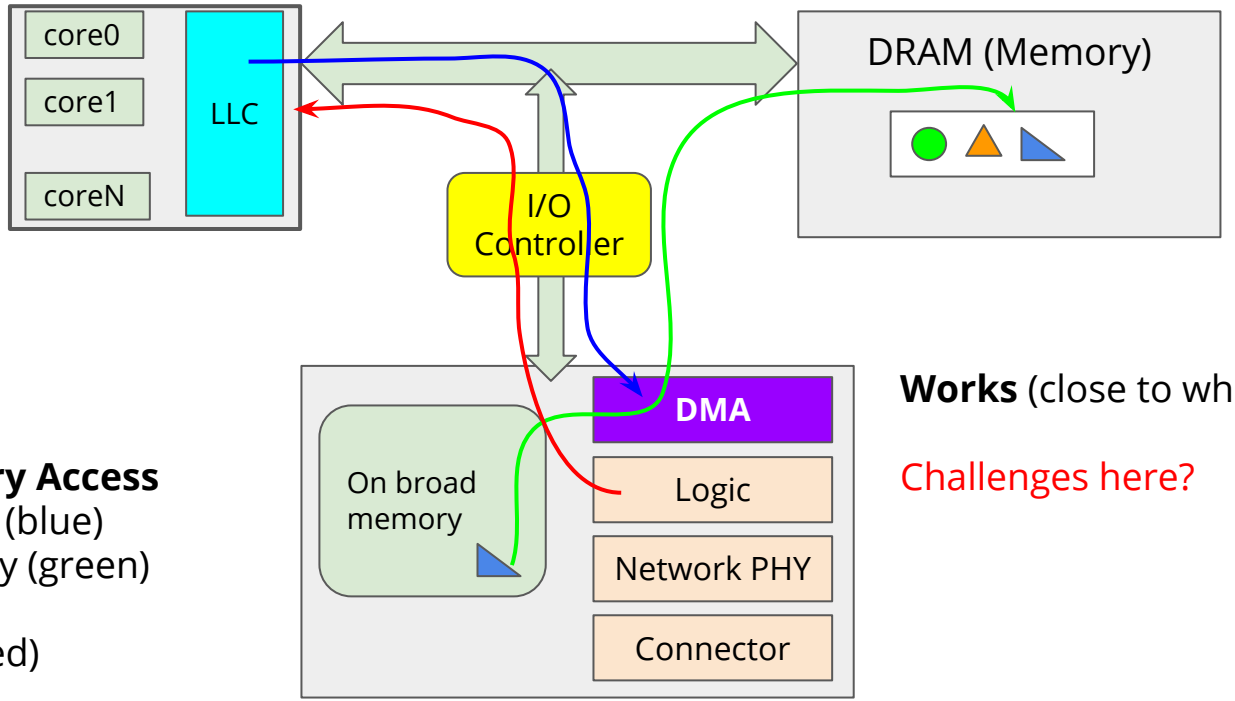
- CPU does polling (red)
  - PIO: Special instructions
  - Memory-mapped I/O
- CPU does data copy (green)

Works, but

- Waste of CPU cycles
- High load on the CPU cannot do anything else

Transferring data between NIC and end-Host

program the DMA engine, tell where to deposit data (addr, length)



Strategy 2:

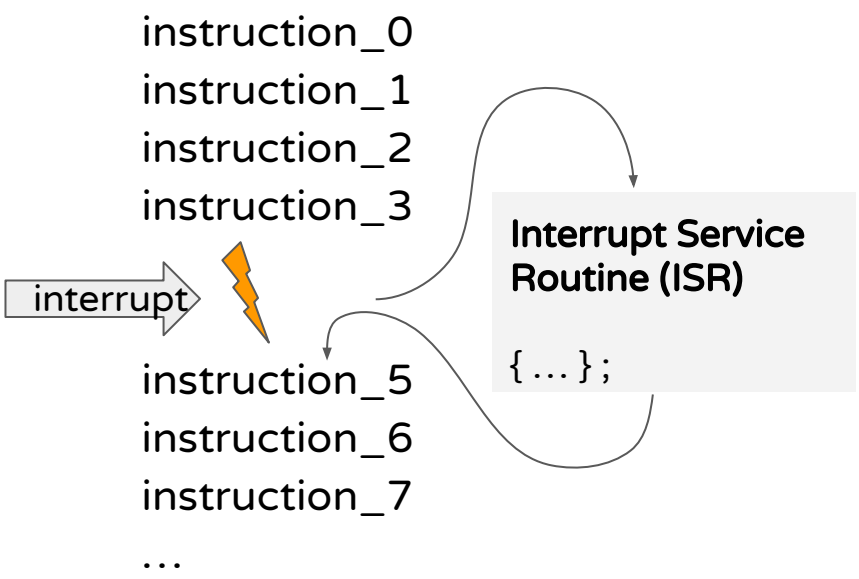
- **Direct Memory Access**
  - Program (blue)
  - Data copy (green)
- Interrupts
  - Notify (red)

Works (close to what we have)

Challenges here?

When should a NIC interrupt about packet reception or transmission?

What happens when there is an interrupt

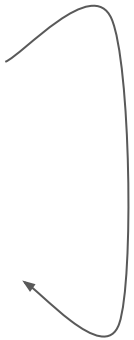


1. Device raises interrupt request
2. Processor interrupts program in execution
3. Interrupts are disabled
4. Device is informed of acceptance and, as a consequence, lowers interrupt
5. Interrupt is handled by **service routine**
6. Interrupts are enabled
7. Execution of interrupted program is resumed

Interrupt storm (or Interrupt livelocks)

Imagine a situation where a CPU is constantly receiving interrupts:

1. The CPU gets an interrupt
2. It processes interrupts by executing ISR
3. Start normal processing ...
4. Interrupt again



No “actual” work progress can be made.

- The system is alive, but is “**locked**” and cannot do any actual work : **livelock**
- In comparison: “deadlock” - just waiting for some resource

Interrupt storms often happens on **the receive path** because a NIC/system cannot control when to receive the packet (*but it controls when to transmit*)

Apollo 11 : The Moon Mission - The First Interrupt Storm

https://www.discovermagazine.com/the-sciences/apollo-11s-1202-alarm-explained

With network interrupts

If there is interrupt every time a packet is received, how frequently there might be an in interrupt for small packets :

64 bytes (+20 headers, min packet size on ETH) of data :

1 Gbps = 84 * 8 / 100 * 10^6 = 0.6 microseconds (barely manageable)

10 Gbps = 84 * 8 / 10 * 10^9 = 67.2 nanoseconds (close to a DRAM access)

100 Gbps = 84 * 8 / 100 * 10^9 = 6.72 nanosecond ! (less than a DRAM access)

At these rates the CPU will just take interrupts, and do nothing else
If it cannot keep up, then packets will be dropped

Interrupt storm mitigations

1. **Interrupt coalescing**
   a. Don’t generate interrupt on every packet, but “n” packets to amortize the cost of taking interrupt
   b. A typical value depends upon (a) NIC buffering capacity; (b) network speed; and (c) accepted delay due to batching of “n” packets

2. **Polling**
   a. Disable interrupts all together, and use CPU polling to check for new packet arrivals

3. **Hybrid : a mix of these two**
   a. In practice, a hybrid strategy of these two are used
   b. Interrupts -> Polling -> Interrupts
   c. There is a threshold, when the rate exceed then switch to polling, then to interrupts

Linux Tools - ethtool -c

```
ETHTOOL(8)                                  System Manager's Manual                                  ETHTOOL(8)

NAME
       ethtool - query or control network driver and hardware settings

SYNOPSIS
       ethtool devname

       ethtool -h|--help

       ethtool --version

       ethtool -a|--show-pause devname

       ethtool -A|--pause devname [autoneg on|off] [rx on|off] [tx on|off]

       ethtool -c|--show-coalesce devname

       ethtool -C|--coalesce devname [adaptive-rx on|off] [adaptive-tx on|off] [rx-usecs N] [rx-frames N] [rx-usecs-irq N]
              [tx-usecs-irq N] [tx-frames-irq N] [stats-block-usecs N] [pkt-rate-low N] [rx-usecs-low N] [rx-frames-low N]
              [rx-usecs-high N] [rx-frames-high N] [tx-usecs-high N] [tx-frames-high N] [sample-interval N]
```

```
atr@atr:~$ ethtool -c enp0s3
Coalesce parameters for enp0s3:
Adaptive RX: off  TX: off
stats-block-usecs: 0
sample-interval: 0
pkt-rate-low: 0
pkt-rate-high: 0

rx-usecs: 0
rx-frames: 0
rx-usecs-irq: 0
rx-frames-irq: 0

tx-usecs: 0
tx-frames: 0
tx-usecs-irq: 0
tx-frames-irq: 0

rx-usecs-low: 0
rx-frame-low: 0
tx-usecs-low: 0
tx-frame-low: 0

rx-usecs-high: 0
rx-frame-high: 0
tx-usecs-high: 0
tx-frame-high: 0

atr@atr:~$
```

Important tool - gives you a lot of information about a network device

- -c and -C are the flags to check for coleasing setting
- Can set threshold when to generate interrupt
  - Timeout
  - Number of packets
  - Adaptive, high and low threshold

How to build a packet with data, header, and trailer?

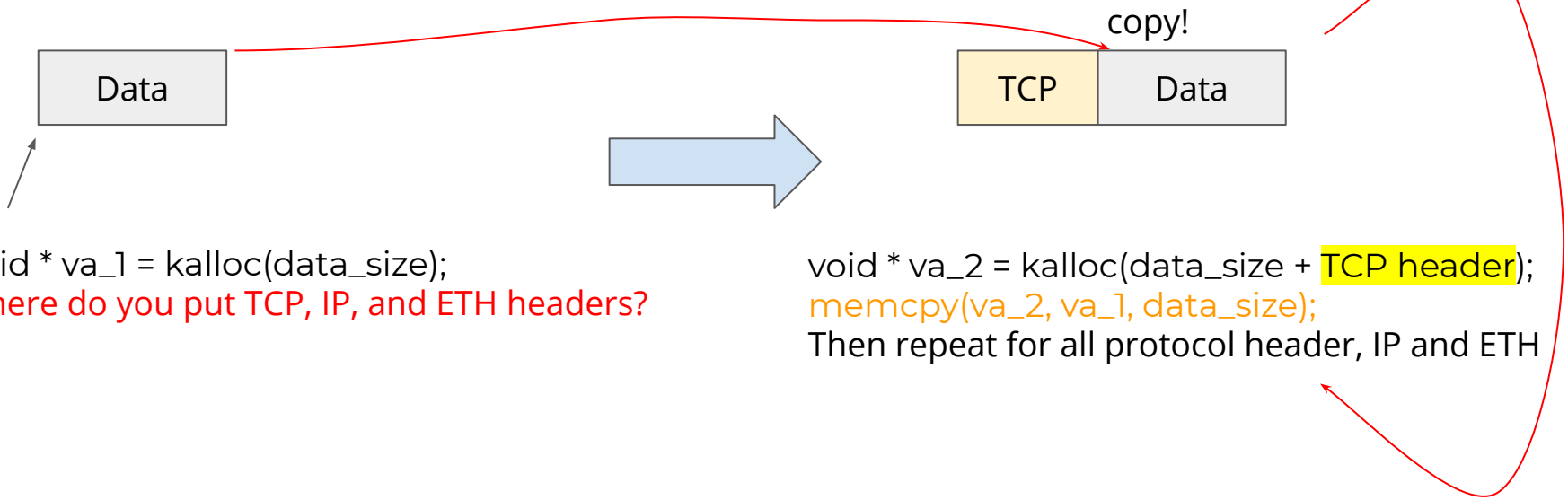
Building Packets with Headers and Trailers

Typically you tell the NIC, please transmit or receive data from (address, length)
Problem: needs that data is contiguous in physical memory

void * va_1 = kalloc(data_size);
where do you put TCP, IP, and ETH headers?

Building Packets with Headers and Trailers

Typically you tell the NIC, please transmit or receive data from (address, length)
Problem: needs that data is contiguous in physical memory

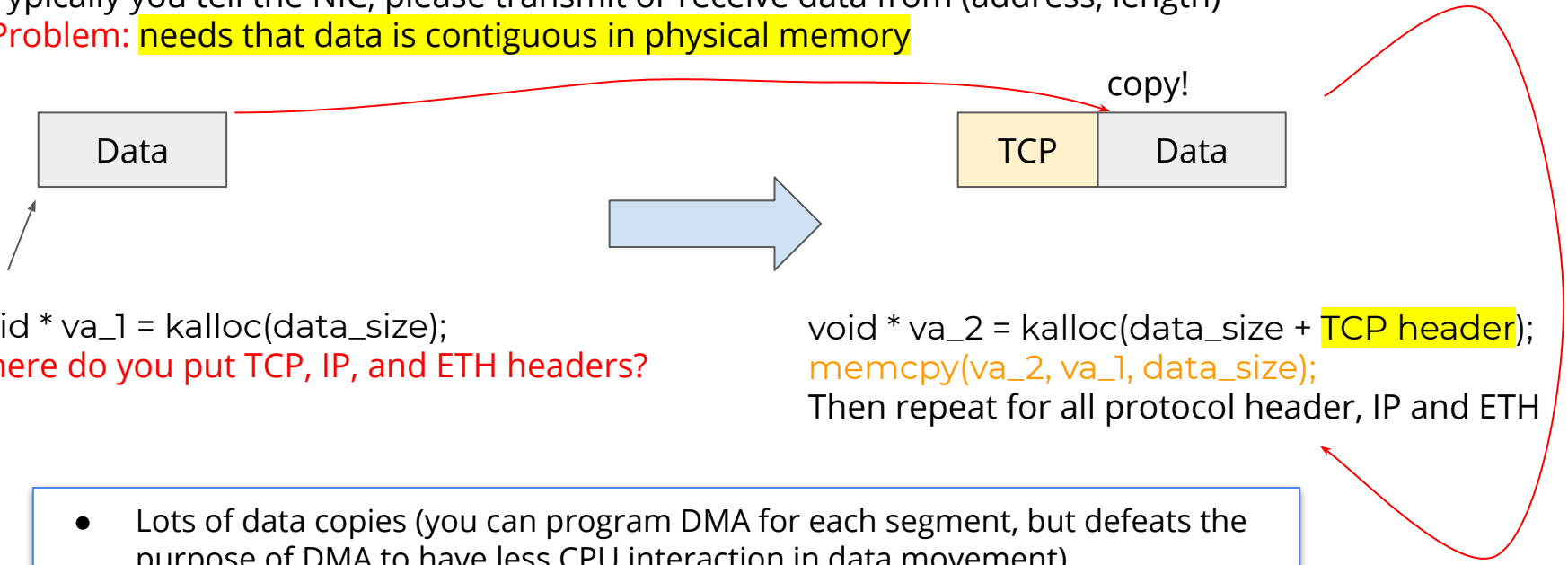


void * va_1 = kalloc(data_size);
where do you put TCP, IP, and ETH headers?

void * va_2 = kalloc(data_size + TCP header);
memcpy(va_2, va_1, data_size);
Then repeat for all protocol header, IP and ETH

Building Packets with Headers and Trailers

Typically you tell the NIC, please transmit or receive data from (address, length)
Problem: needs that data is contiguous in physical memory



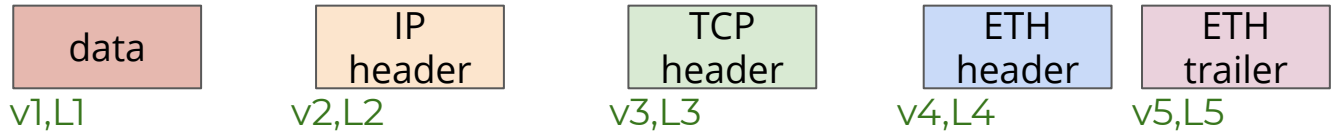
void * va_1 = kalloc(data_size);
where do you put TCP, IP, and ETH headers?

void * va_2 = kalloc(data_size + TCP header);
memcpy(va_2, va_1, data_size);
Then repeat for all protocol header, IP and ETH

- Lots of data copies (you can program DMA for each segment, but defeats the purpose of DMA to have less CPU interaction in data movement)
- CPU occiped, and waste of CPU
- Poor performance (more copies ⇒ low bandwidth, high latency, low ops/secs)
- Similarly think about on the receive path : you need to strip headers

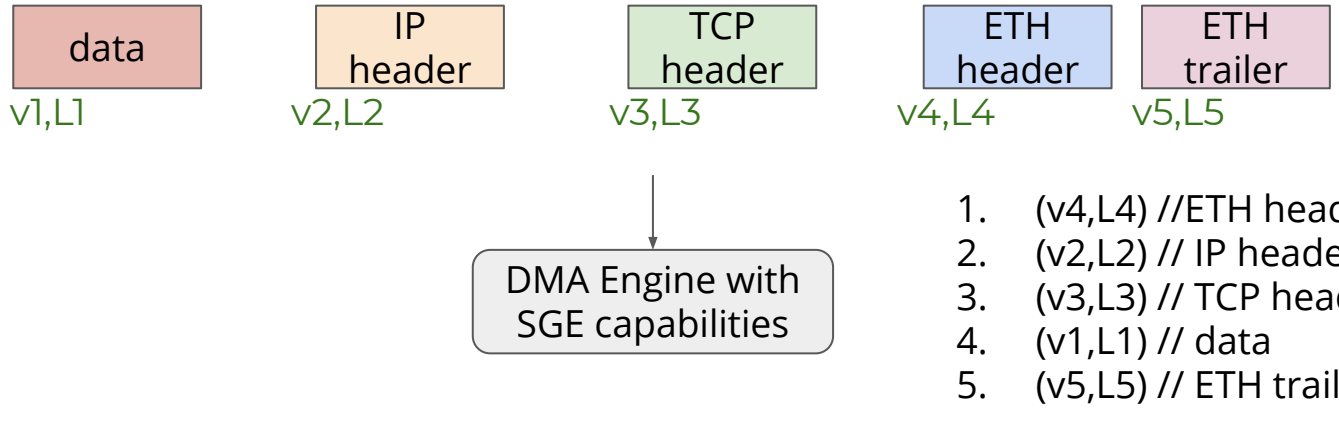
Scatter-Gather I/O Capabilities

Instead of one address, one length, pass a list of address if length to the DMA engine



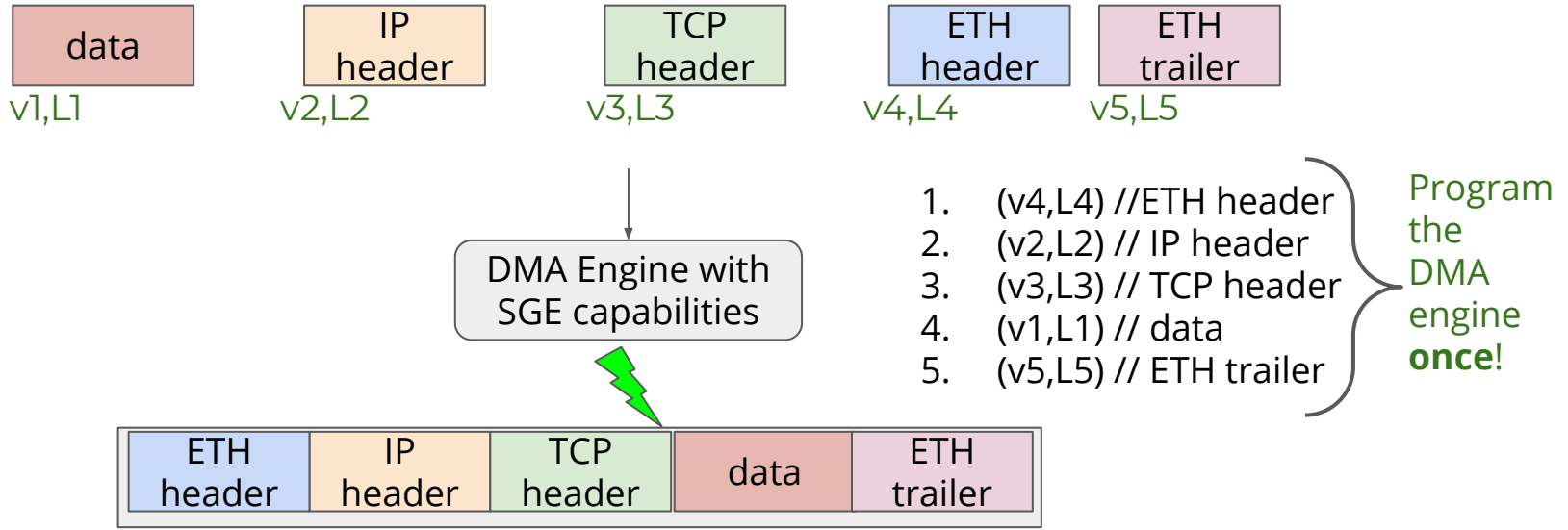
Scatter-Gather I/O Capabilities

Instead of one address, one length, pass a list of address if length to the DMA engine



Scatter-Gather I/O Capabilities

Instead of one address, one length, pass a list of address if length to the DMA engine



A single packet is built and transmitted from multiple disjoint locations.

Linux Tool: ethtool -k

```
atr@evelyn:~$ ethtool -k enp0s25
Features for enp0s25:
rx-checksumming: on
tx-checksumming: on
        tx-checksum-ipv4: off [fixed]
        tx-checksum-ip-generic: on
        tx-checksum-ipv6: off [fixed]
        tx-checksum-fcoe-crc: off [fixed]
        tx-checksum-sctp: off [fixed]
scatter-gather: on
        tx-scatter-gather: on
        tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: on
        tx-tcp-segmentation: on
        tx-tcp-ecn-segmentation: off [fixed]
        tx-tcp-mangleid-segmentation: off
        tx-tcp6-segmentation: on
udp-fragmentation-offload: off
generic-segmentation-offload: on
generic-receive-offload: on
large-receive-offload: off [fixed]
rx-vlan-offload: on
tx-vlan-offload: on
```

Recap - Lecture 1

From this lecture you should know

1. Basic course administrative information
2. Refresh the idea of socket networking
3. What happens when you send or receive a data packet
4. Where is networking stack implemented
5. What is a interrupt storm
6. What is a scatter-gather I/O

Next lecture, we will continue with the basics ...