Blockchain (BlCh)

Repetition DSy – part 1

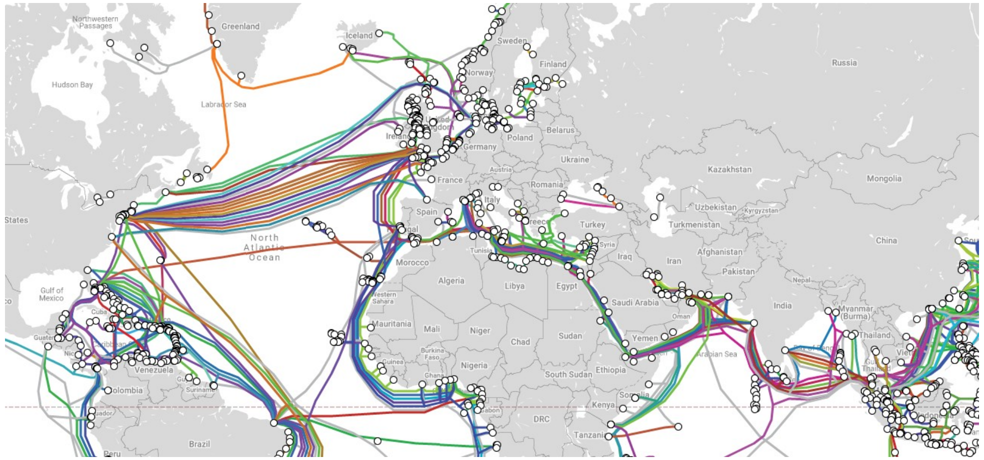
Thomas Bocek

18.09.2022

Lecture 1

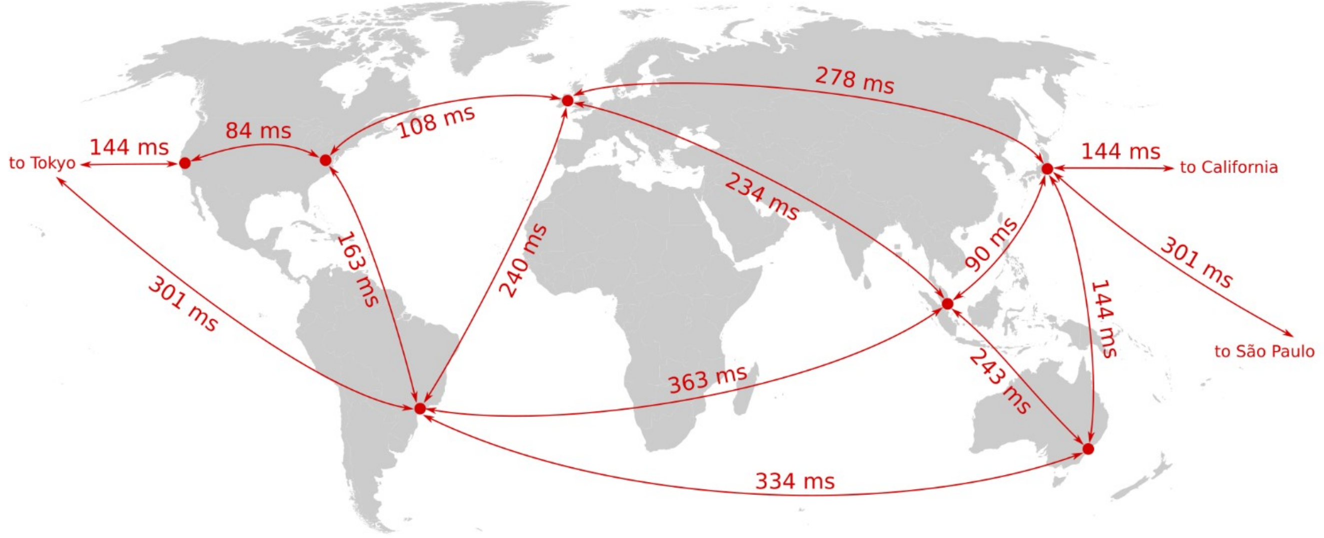
Distributed Systems Motivation

- Why Distributed Systems
  - Scaling
  - Location
  - Fault-tolerance (bitflips, outages)

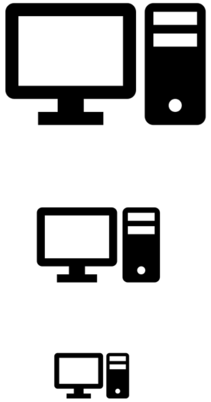


Submarine Cable Map

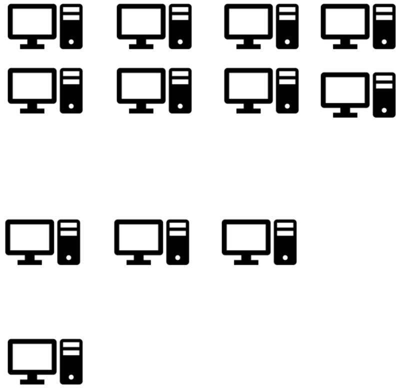
https://www.inkandswitch.com/local-first.html



vertical



horizontal



Lecture 2

Distributed Systems Categorization

“Controlled” Distributed Systems

- 1 responsible organization
- Low churn
- Examples:
  - Amazon DynamoDB
  - Client/server
- “Secure environment”
- High availability
- Can be homogeneous / heterogeneous

“Fully” Decentralized Systems

- N responsible organizations
- High churn
- Examples:
  - BitTorrent
  - Blockchain
- “Hostile environment”
- Unpredictable availability
- Is heterogeneous

Distributed Systems Categorization

"Controlled" Distributed Systems

- Mechanisms that work well:
  - Consistent hashing (DynamoDB, Cassandra)
  - Master nodes, central coordinator
- Network is under control or client/server → no NAT issues

"Fully" Decentralized Systems

- Mechanisms that work well:
  - Consistent hashing (DHTs)
  - Flooding/broadcasting - Bitcoin
- NAT and direct connectivity huge problem

OST

Distributed Systems Categorization

"Controlled" Distributed Systems

- Consistency
  - Leader election (Zookeeper, Paxos, Raft)
- Replication principles
  - More replicas: higher availability, higher reliability, higher performance, better scalability, but: requires maintaining consistency in replicas
- Transparency principles apply

"Fully" Decentralized Systems

- Consistency
  - Weak consistency: DHTs
  - Nakamoto consensus (aka proof of work)
  - Proof of stake – Leader election, PBFT protocols
    Is Bitcoin eventually consistent?
    - Some argue no, some argue it has even stronger guarantees [link]
- Replication principles apply to fully decentralized systems as well
- Transparency principles apply

OST

Distributed Systems Categorization

- Spring Term – Distributed Systems (DSy)
  - Tightly/loosely coupled
  - Heterogeneous systems
  - Small-scale systems
  - Distributed systems

  (we will also talk about blockchains in this lecture)

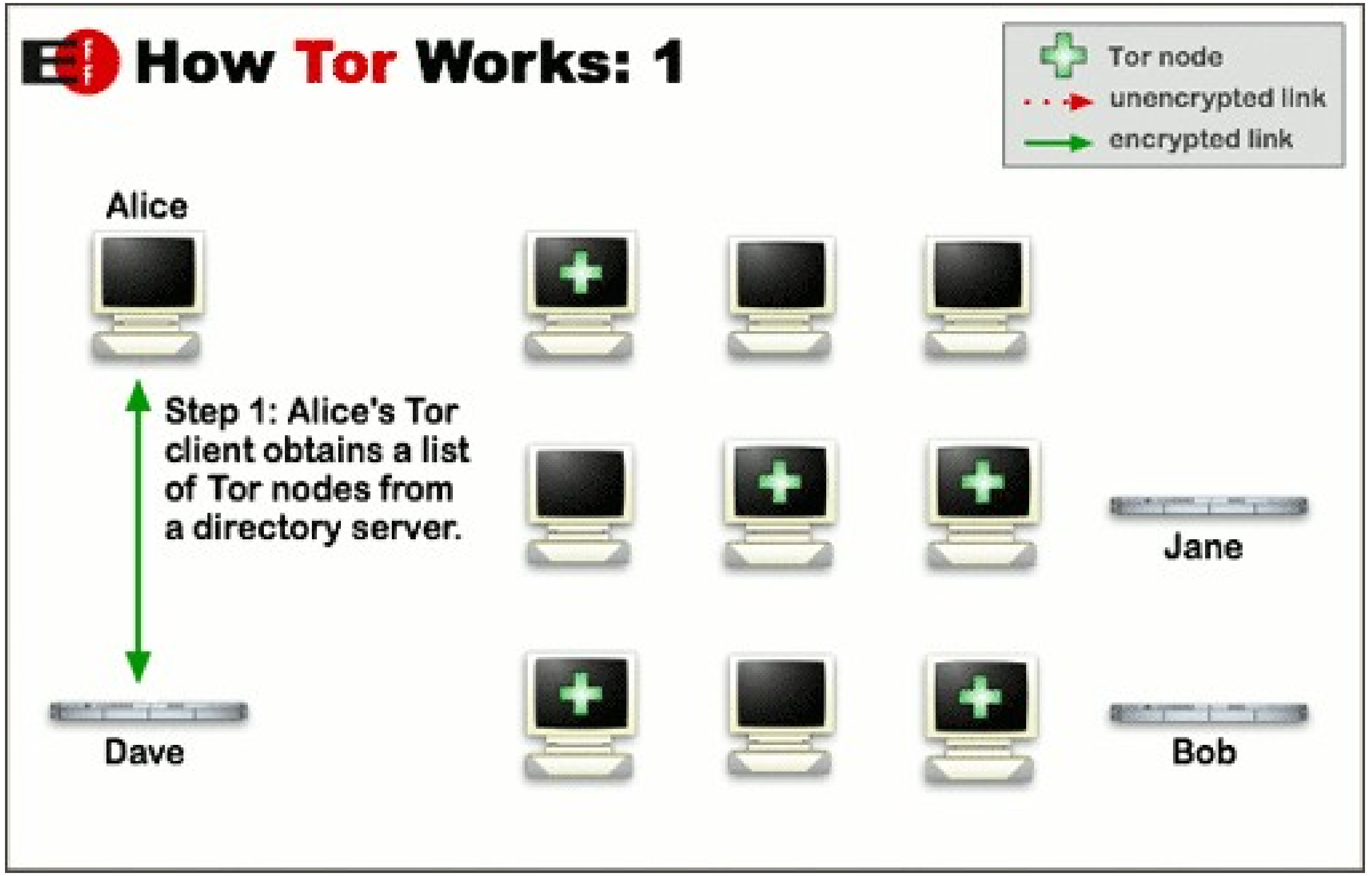
- Fall Term – Blockchain (BlCh)
  - Loosely coupled
  - Heterogeneous systems
  - Large-scale systems
  - Decentralized systems

  (we will also talk about distributed systems in this lecture, but DSy is highly recommended)

Lecture 3

Tor

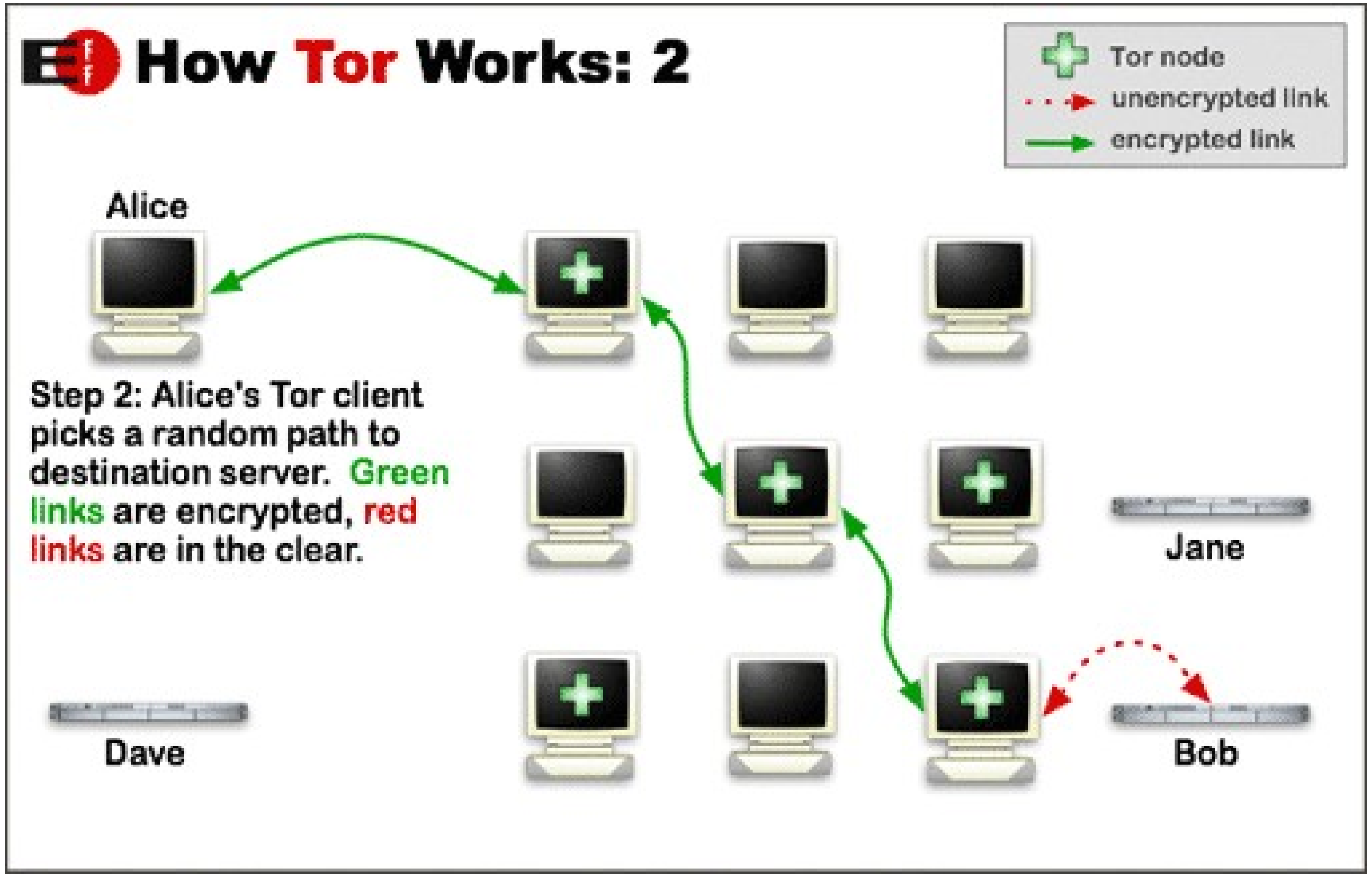
- How it works



OST

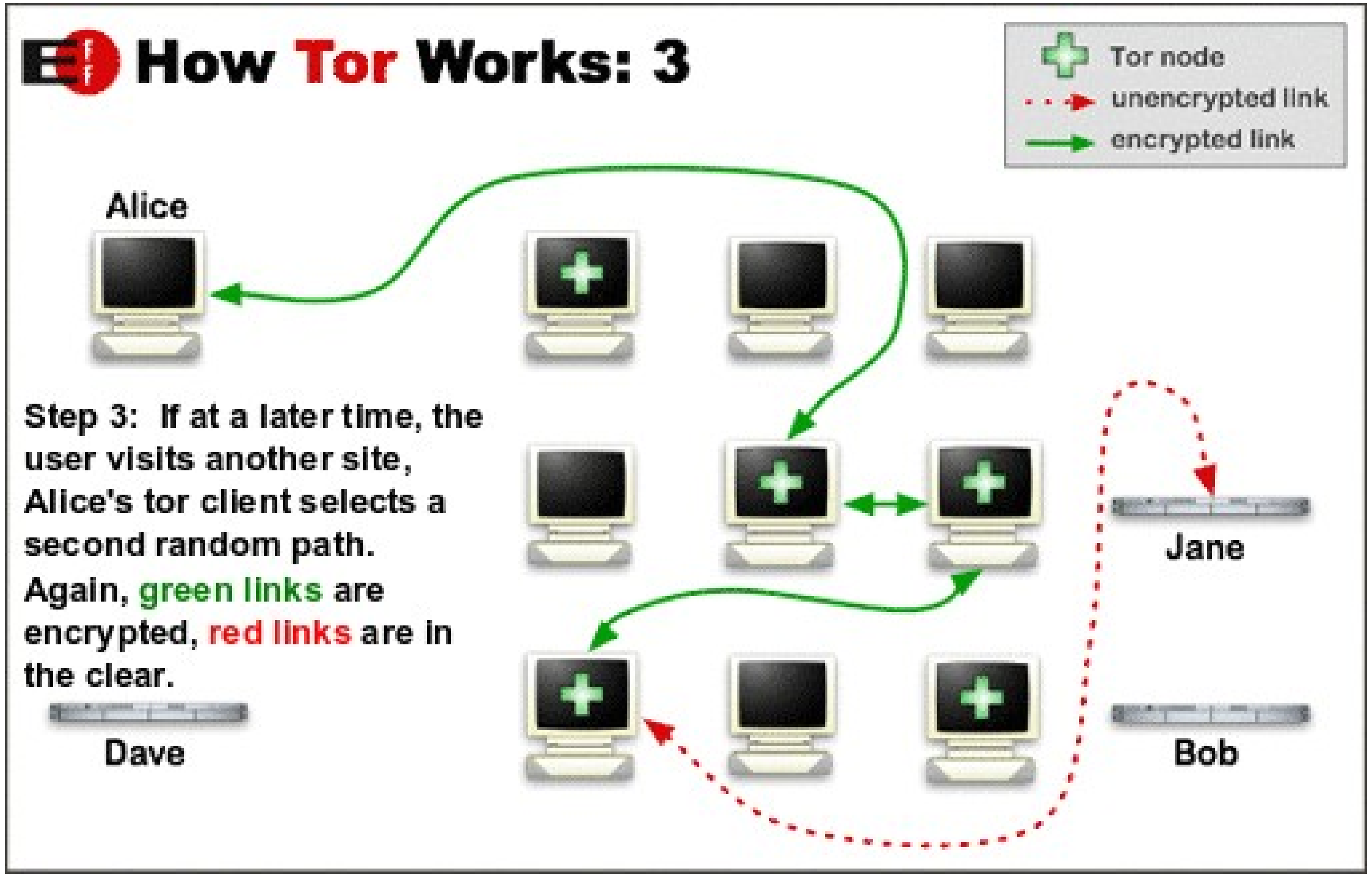
Tor

- Alice to Bob



Tor

- Alice to Jane



WebSockets

- Full-duplex communication over TCP [overview]
  - REST / JSON is in one direction
- How can the server notify the browser (client?)
  - Polling
    - Short: request e.g. every 0.5s
    - Long: request until timeout or reply
  - Server Sent Events (alternative) SSE
    - One way communication from server to browser (client)
    - Server receives a regular HTTP request, keeps connection open, server can now push data to the client
  - WebSockets

- HTTP handshake, then upgrade to communication channel

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Server response:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

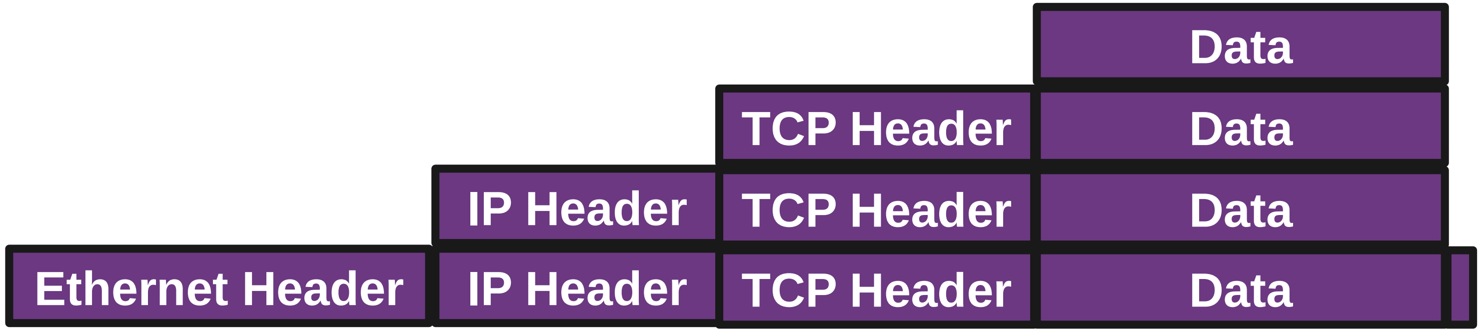
  - Data can be text or binary
- With SSL/TLS → wss://
  - Some configuration required on LBs / RRs

OST

Networking: Layers

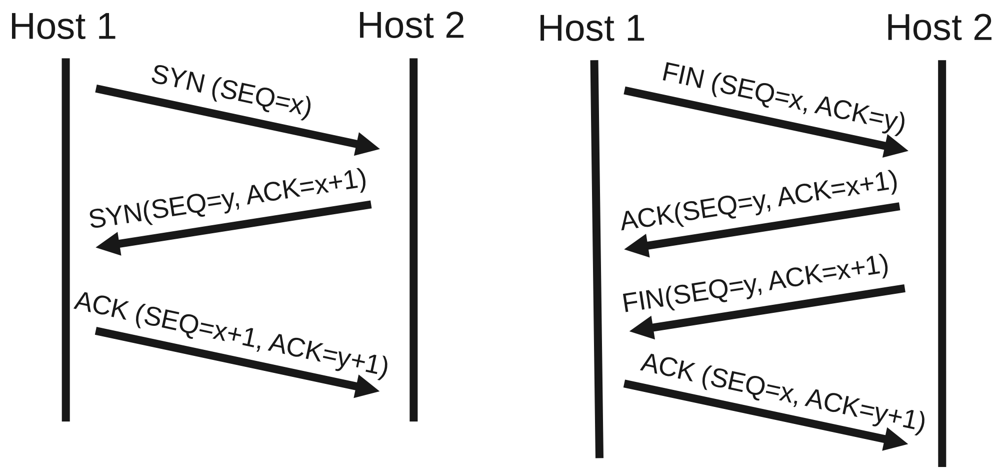
- Networking: Each vendor had its own proprietary solution - not compatible with another solution
  - IPX/SPX – 1983, AppleTalk 1985, DECnet 1975, XNS 1977
- Nowadays most vendors build compatible networks hardware/software from different vendors
  - Cisco, Dell, HP, Huawei, Juniper, Lenovo, Linksys, Netgear, MicroTik, Siemens, Ubiquiti, etc.
- Goal of layers: interoperability
  - 1984: ISO 7498 - The Basic Reference Model for Open Systems Interconnection

OSI model	"Internet model"
Application	Application
Presentation	
Session	
Transport	Transport
Network	Internet
Data link	Link
Pysical	



Layer 4 - TCP

- Connection establishment
  - SYN, SYN-ACK, ACK (three way)
  - Initiates TCP session: initial sequence number is ~ random
- Connection termination
  - FIN, ACK + FIN, ACK (three/four way)
  - 3-way handshake, when host 1 sends a FIN and host 2 replies with a FIN & ACK
- Sequences and ACKs
  - Identification each byte of data
  - Order of the bytes → reconstruction
  - Detecting lost data: RTO, DupACK:



- Retransmission timeout
  - If no ACK is received aftert timout (e.g. 2xRTT), resend.
- Duplicate cumulative acknowledgements, selective ACK [link]
  - ACKs for last consecutive packets
  - 3 times same ACK → retransmit missing packets (fast retransmit)

TCP/IP from an Application Developer View

- Server in golang (repo)
  - git clone https://github.com/tbocek/DSy
  - Download GoLand, or others
  - go run server.go → server
- Listening on TCP port 8081
  - Return string in uppercase
- Node.js version
  - Download WebStorm, or other
- Client:
  - nc localhost 8081

```
const net = require('net');
const server = new net.Server();
server.listen(8081, function() {
   console.log('Launching server...');
});

server.on('connection', function(socket) {
   socket.on('data', function(chunk) {
      console.log(`Data received from client: ${chunk.toString()}`);

socket.write(chunk.toString().toUpperCase() + "\n");
   });
});
```

```
package main
import ("bufio"
    "fmt"
    "net"
    "strings")
func main() {
    fmt.Println("Launching server...")
    ln, _ := net.Listen("tcp", ":8081") // listen on all interfaces
    for {
        conn, _ := ln.Accept() // accept connection on port
        message, _ := bufio.NewReader(conn).ReadString('\n') //read line
        fmt.Print("Message Received:", string(message))
        newMessage := strings.ToUpper(message) //change to upper
        conn.Write([]byte(newMessage + "\n")) //send upper string back
    }
}
```

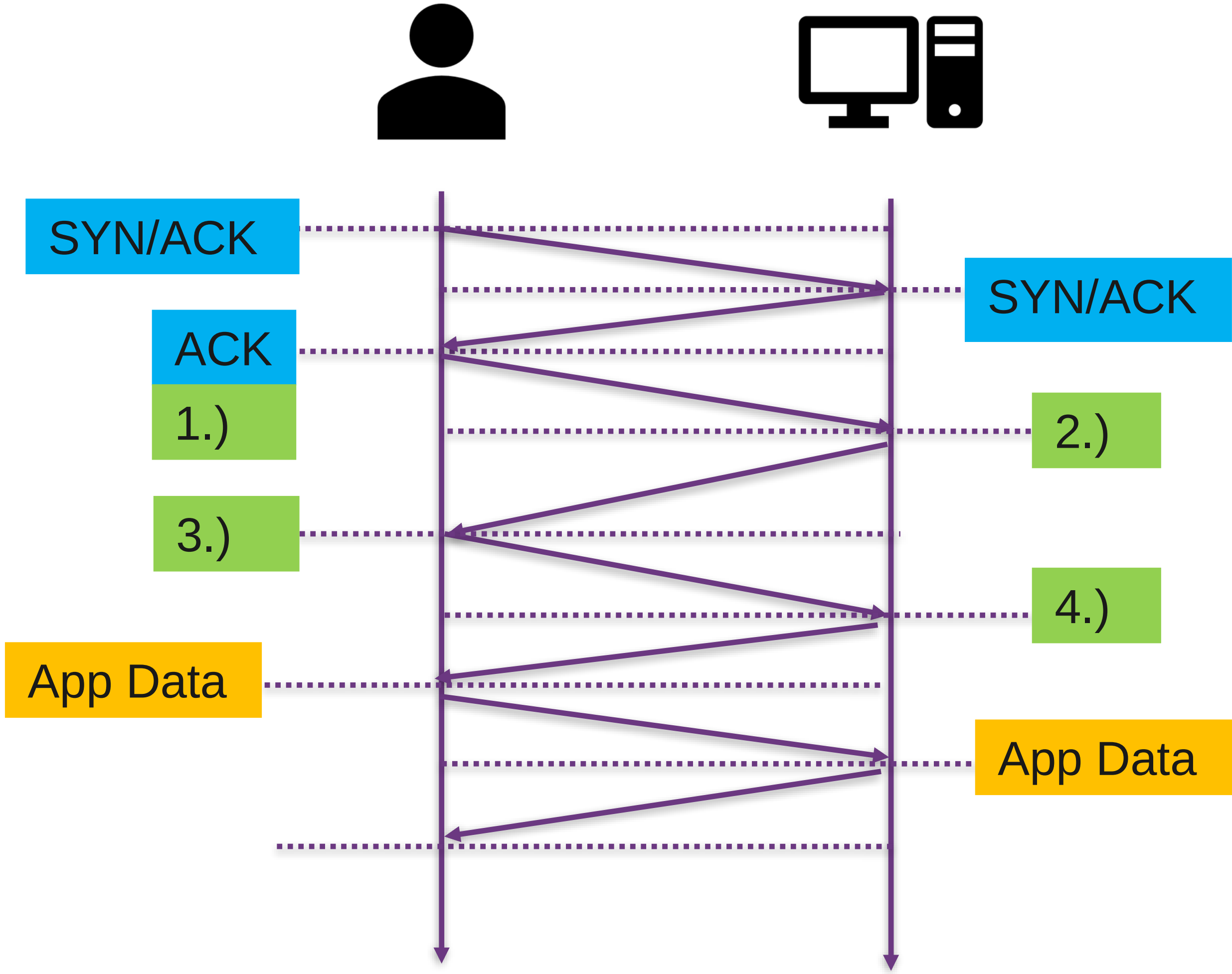
Lecture 4

Layer 4 – TCP + TLS

- Security: Transport Layer Security (TLS)

  1. "client hello" lists cryptographic information, TLS version, ciphers/keys
  2. "server hello" chosen cipher, the session ID, random bytes, digital certificate (checked by client), optional: "client certificate request"
  3. Key exchange using random bytes, now server and client can calc secret key
  4. "finished" message, encrypted with the secret key

- 3 RTT to send first byte, 4RTT to receive first byte

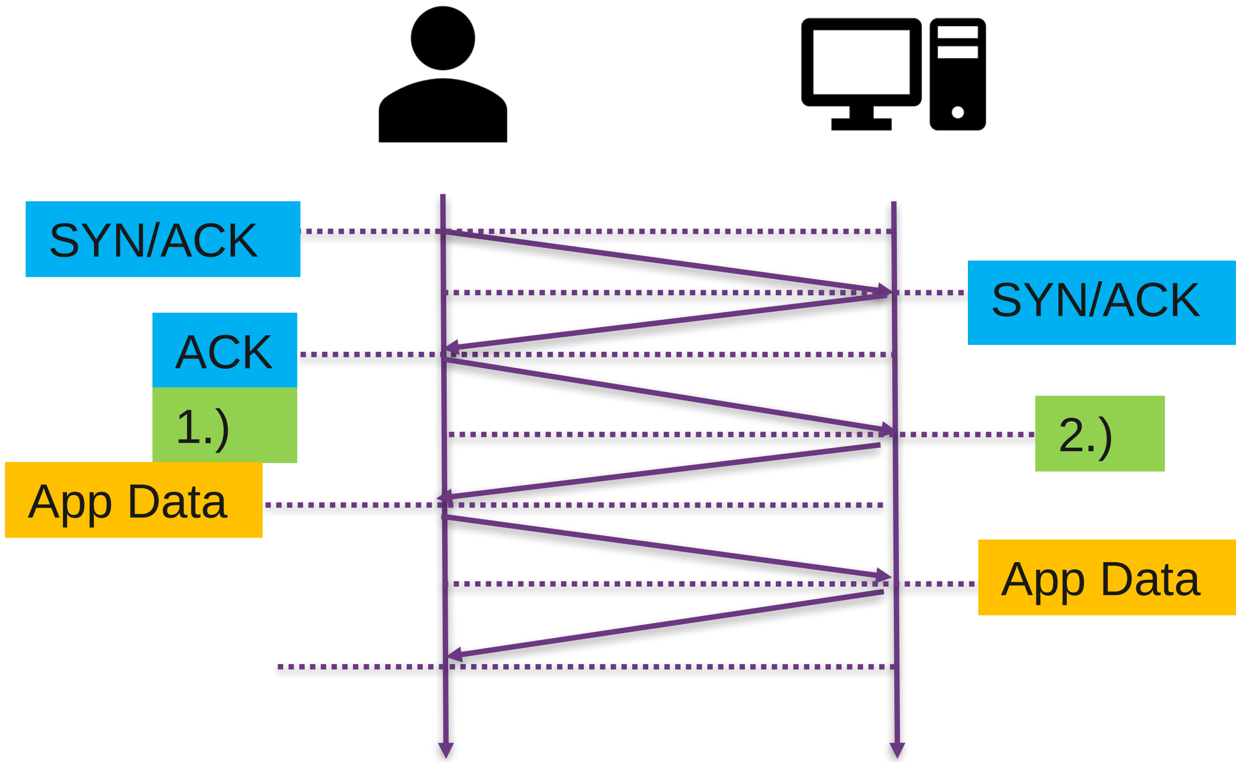


https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_8.0.0/com.ibm.mq.sec.doc/q009930_.htm

```
PING sydney.edu.au (129.78.5.8) 56(84) bytes of data.
64 bytes from scilearn.sydney.edu.au (129.78.5.8): icmp_seq=1 ttl=233 time=307 ms
64 bytes from scilearn.sydney.edu.au (129.78.5.8): icmp_seq=2 ttl=233 time=305 ms
64 bytes from scilearn.sydney.edu.au (129.78.5.8): icmp_seq=3 ttl=233 time=305 ms
```

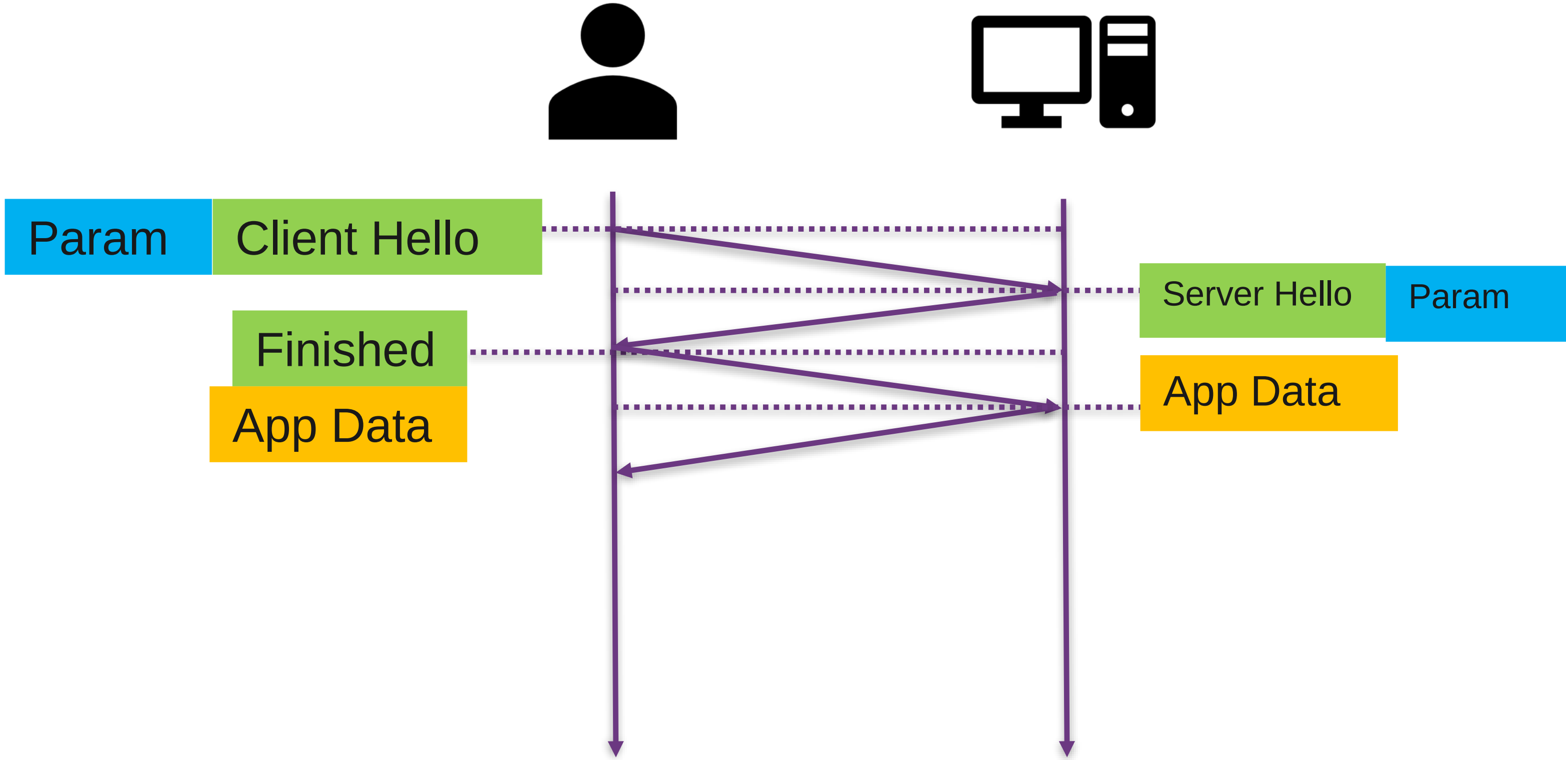
Layer 4 – TCP + TLS

- Ping to Australia: 329ms
  - One way ~ 165ms
- TCP + TLS handshake:
  - 3RTT = 987ms! No data sent yet
- TLS 1.3, finished Aug 2018
  - 1 RTT instead of 2
    - 1.) Client Hello, Key Share
    - 2.) Server Hello, key Share, Verify Certificate, Finished
  - 0 RTT possible, for previous connections, loosing perfect forward secrecy
  - 90% of browsers used already support it



QUIC / HTTP3

- QUIC: 1RTT (chrome example)
  - For known connections: 0RTT
  - Built in security
  - "Google's 'QUIC' TCP alternative slow to excite anyone outside Google" [link] (7%, 25%)
    - Facebook
    - Cloudflare
  - Can I use (72.5%)
- Example Australia: from 987ms to 329ms



OST

Pro/Cons - Opinion

- **Monorepo**
  - Tight coupling of projects
  - Everyone sees all code / commits
  - Encourages code sharing within organization
  - Scaling: large repos, specialized tooling
- **Polyrepo**
  - Loose coupling of projects
  - Fine grained access control
  - Encourages code sharing across organizations
  - Scaling: many projects, special coordination

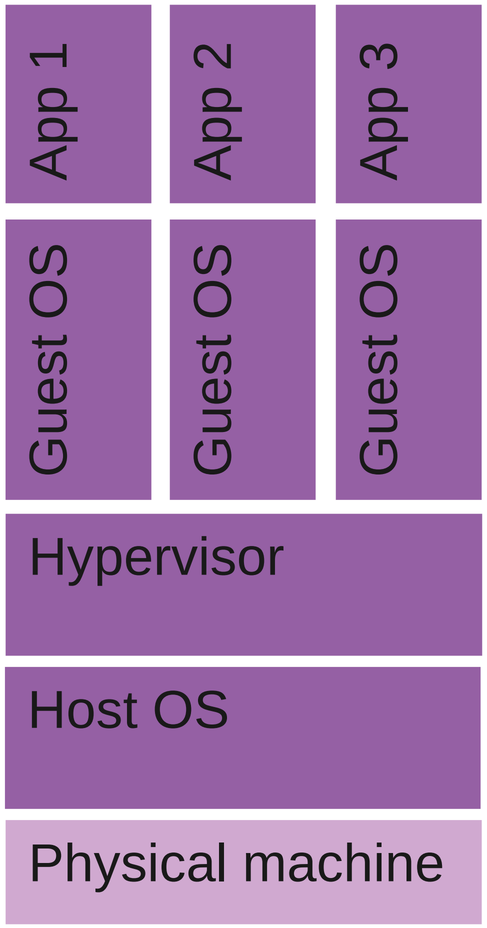
- Opinion: Accenture - “From my experience, for a smaller team, starting with mono-repo is always safe and easy to start. Large and distributed teams would benefit more from poly-repo”
- My opinion: for small teams and project, use polyrepo. (I worked with small teams with mono and polyrepo, I have worked in big projects with polyrepos, but never in a big project with monorepos)
- Other opinion (sales pitch): https://monorepo.tools

OST

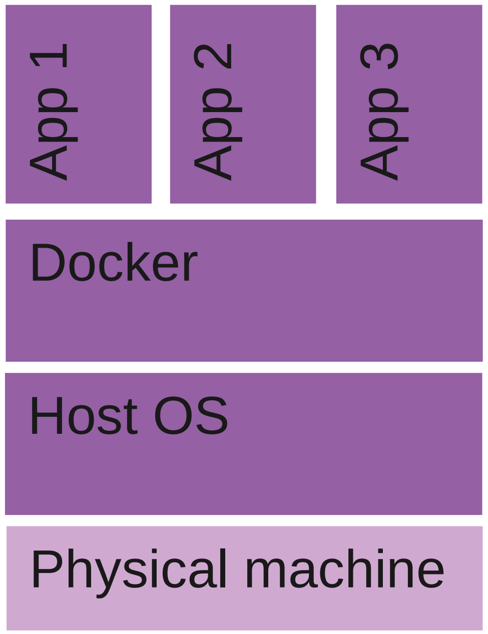
Lecture 5

OST

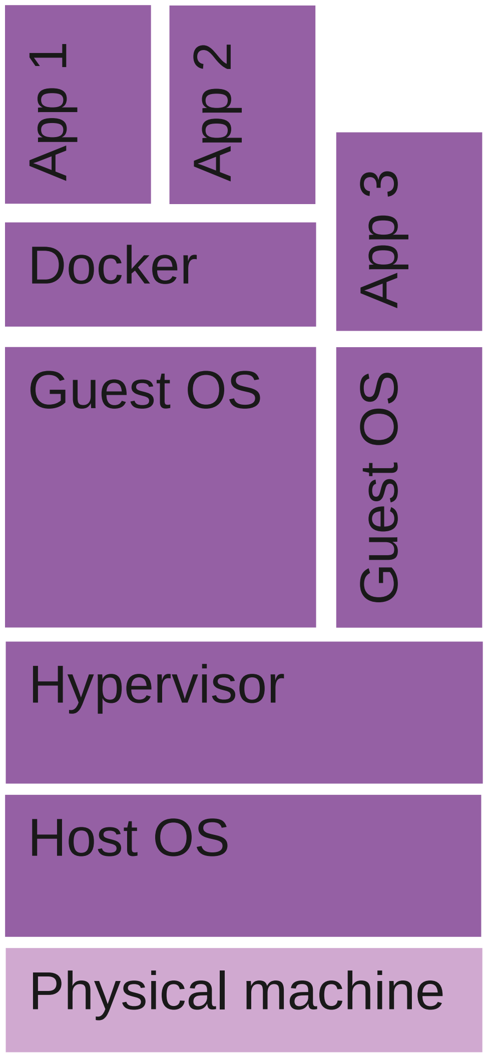
Introduction



- Virtual machines



- Container



- Both

Comparison

Container

+ Reduced size of snapshots 2MB vs 45MB

+ Quicker spinning up apps

+ / - Available memory is shared

+ / - Process-based isolation (share same kernel)

Use case: complex application setup, with container less complex configuration

Providers: ECS, Kubernetes Engine, Docker on Azure (or Kubernetes)

Virtual Machine

+ App can access all OS resources

+ Live migrations

+ / - Pre allocates memory

+ / - Full isolation

Use case: better hardware utilization / resource sharing

EC2, Virtual Machines, Compute Engine, Droplets

Market shares, market hares, other views

OverlayFS

- Example
  - The lower directory can be read-only or could be an overlay itself
  - The upper directory is normally writable
  - The workdir is used to prepare files as they are switched between the layers.

```
cd /tmp
mkdir lower upper workdir overlay

sudo mount -t overlay -o \
lowerdir=/tmp/lower,\
upperdir=/tmp/upper,\
workdir=/tmp/workdir \
none /tmp/overlay
```

- Read only
- How to remove data in read-only lowerdir
  - Mark as deleted in upperdir

```
cd /tmp
mkdir lower upper workdir overlay

sudo mount -t overlay -o
lowerdir=/tmp/lower1:/tmp/lower2 /tmp/overlay
```

```
cd /tmp
mkdir lower upper workdir overlay

sudo mount -t overlay -o \
lowerdir=/tmp/lower1:/tmp/lower2,\
upperdir=/tmp/upper,\
workdir=/tmp/workdir \
none /tmp/overlay
```

Cgroups

- control groups: limits, isolates, prioritization of CPU, memory, disk I/O, network

```
ls /sys/fs/cgroup

sudo apt install cgroup-tools / yay -S libcgroup

cgcreate -g cpu:red
cgcreate -g cpu:blue

echo -n "20" > /sys/fs/cgroup/blue/cpu.weight
echo -n "80" > /sys/fs/cgroup/red/cpu.weight

cgexec -g cpu:blue bash
cgexec -g cpu:red bash

sha256sum /dev/urandom #does not work?
taskset -c 0 sha256sum /dev/urandom
```

- Install tools
- Create two groups
  - Assign 20% of CPU and 80% of CPU
- Execute bash → test CPU
- Resource control with docker

```
docker run \
--name=low_prio \
--cpuset-cpus=0 \
--cpu-shares=20 \
alpine sha256sum /dev/urandom

docker run \
--name=high_prio \
--cpuset-cpus=0 \
--cpu-shares=80 \
alpine sah256sum /dev/urandom
```

Separate Networks

- Linux Network Namespaces
  - provide isolation of the system resources associated with networking [source]

```
ip netns add testnet
ip netns list
```

  - Create virtual ethernet connection

```
ip link add veth0 type veth peer name veth1 netns testnet
ip link list #?
ip netns exec testnet <cmd>
```

  - Configure network

```
ip addr add 10.1.1.1/24 dev veth0
ip netns exec testnet ip addr add 10.1.1.2/24 dev veth1
ip netns exec testnet ip link set dev veth1 up
```

- Run server

```
ip netns exec blue nc –l 8000
```

- Server can be contacted
- How to connect to outside?
  - E.g. layer 3

```
iptables -t nat -A POSTROUTING -s 10.1.1.0/24 -o enp9s0 -j MASQUERADE
iptables -A FORWARD -j ACCEPT #open up wide…
```