Blockchain (BlCh)

Repetition DSy – part 1

Thomas Bocek

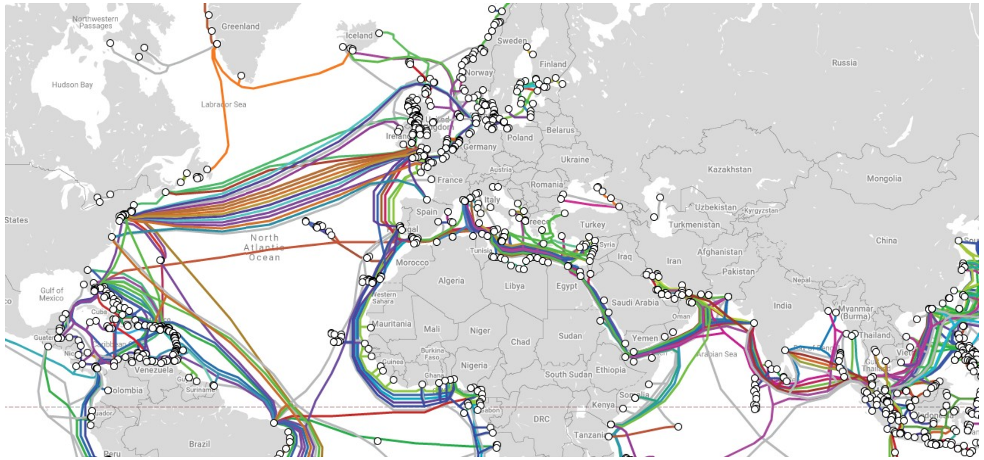
OST Eastern Switzerland University of Applied Sciences

17.09.2023

Lecture 1

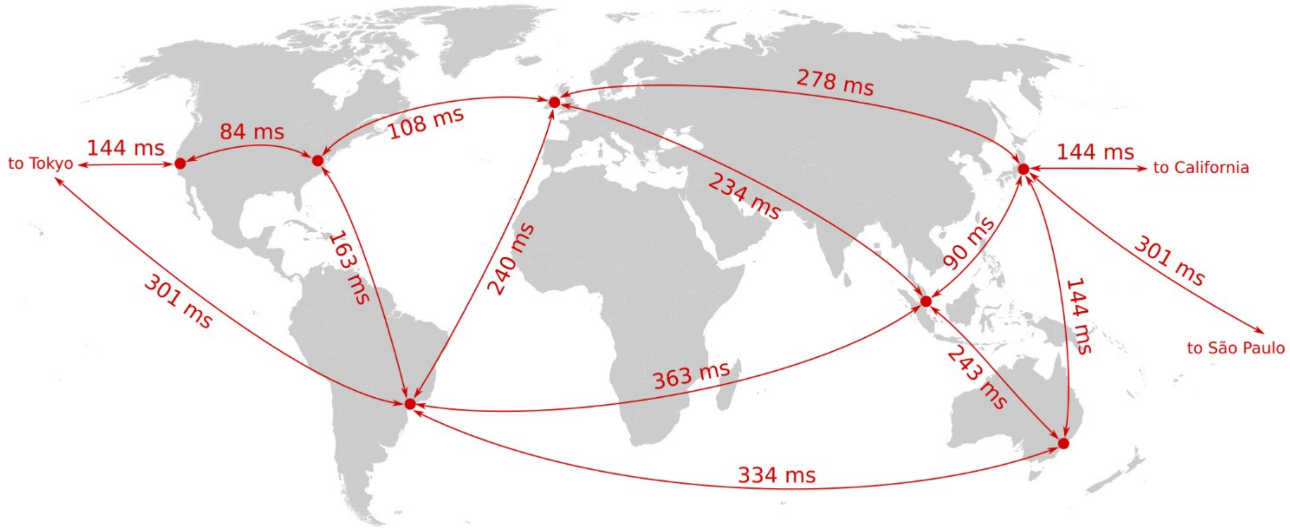
Distributed Systems Motivation

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

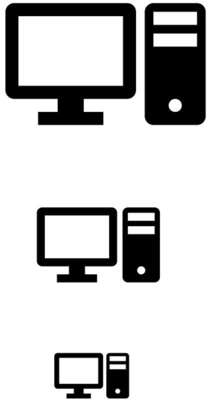


Submarine Cable Map

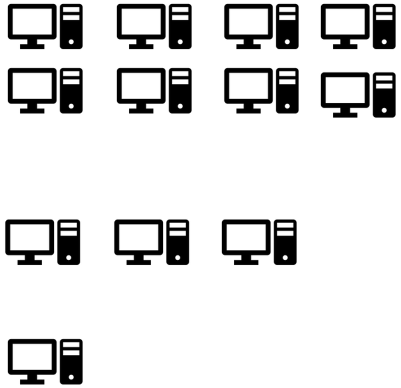
https://www.inkandswitch.com/local-first.html



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

OST

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

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

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

Pro/Cons - Opinion

- Monorepo
  - Tight coupling of projects
    - E.g., generating openapi.yml from backend, generate types for frontend → simply copy
  - Everyone sees all code / commits
  - Encourages code sharing within organization
  - Scaling: large repos, specialized tooling
- Polyrepo
  - Loose coupling of projects
    - If you want to generate openapi.yml, you need access from the backend repository to the frontend (e.g., curl+token)
  - Fine grained access control
  - Encourages code sharing across organizations
  - Scaling: many projects, special coordination

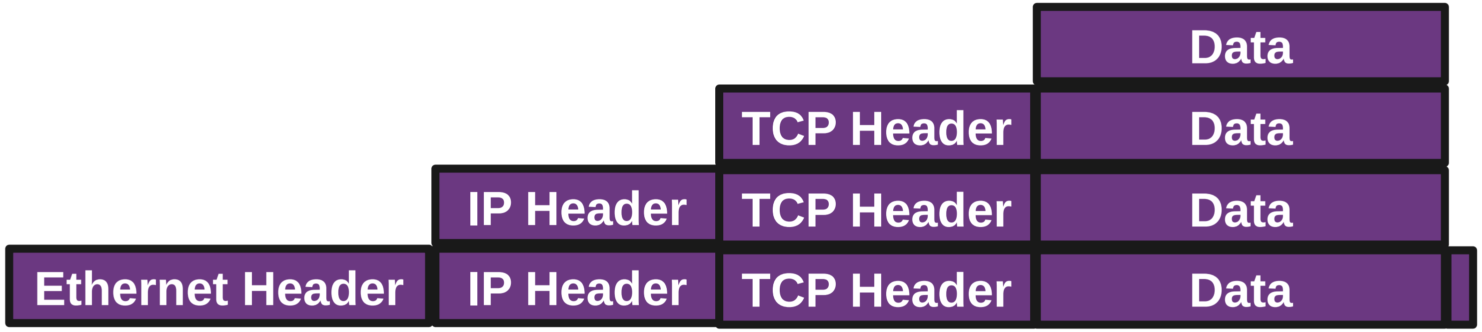
Key Differences

- 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 “independent” 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). If you have a tight coupling between projects (OpenAPI), use monorepos.
- Other opinion (sales pitch): https://monorepo.tools

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

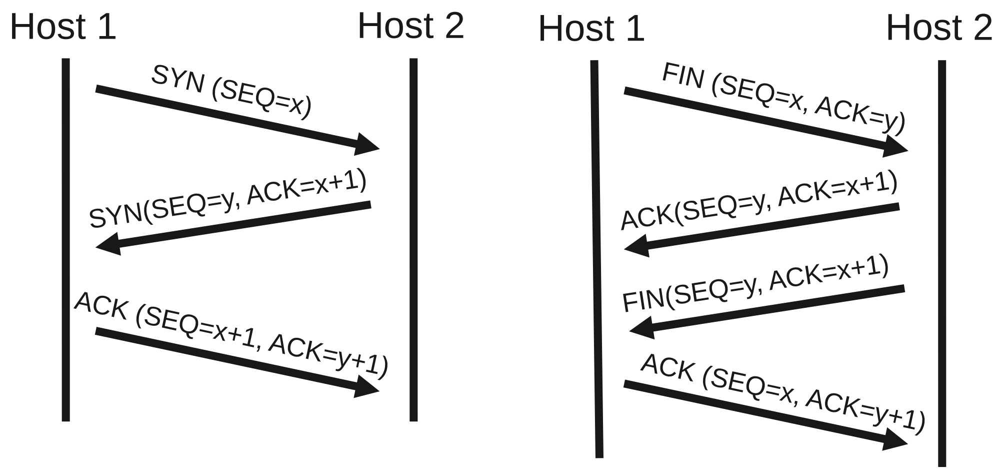
OSI model	"Internet model"
Application	Application
Presentation	
Session	
Transport	Transport
Network	Internet
Data link	Link
Pysical	



OST

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

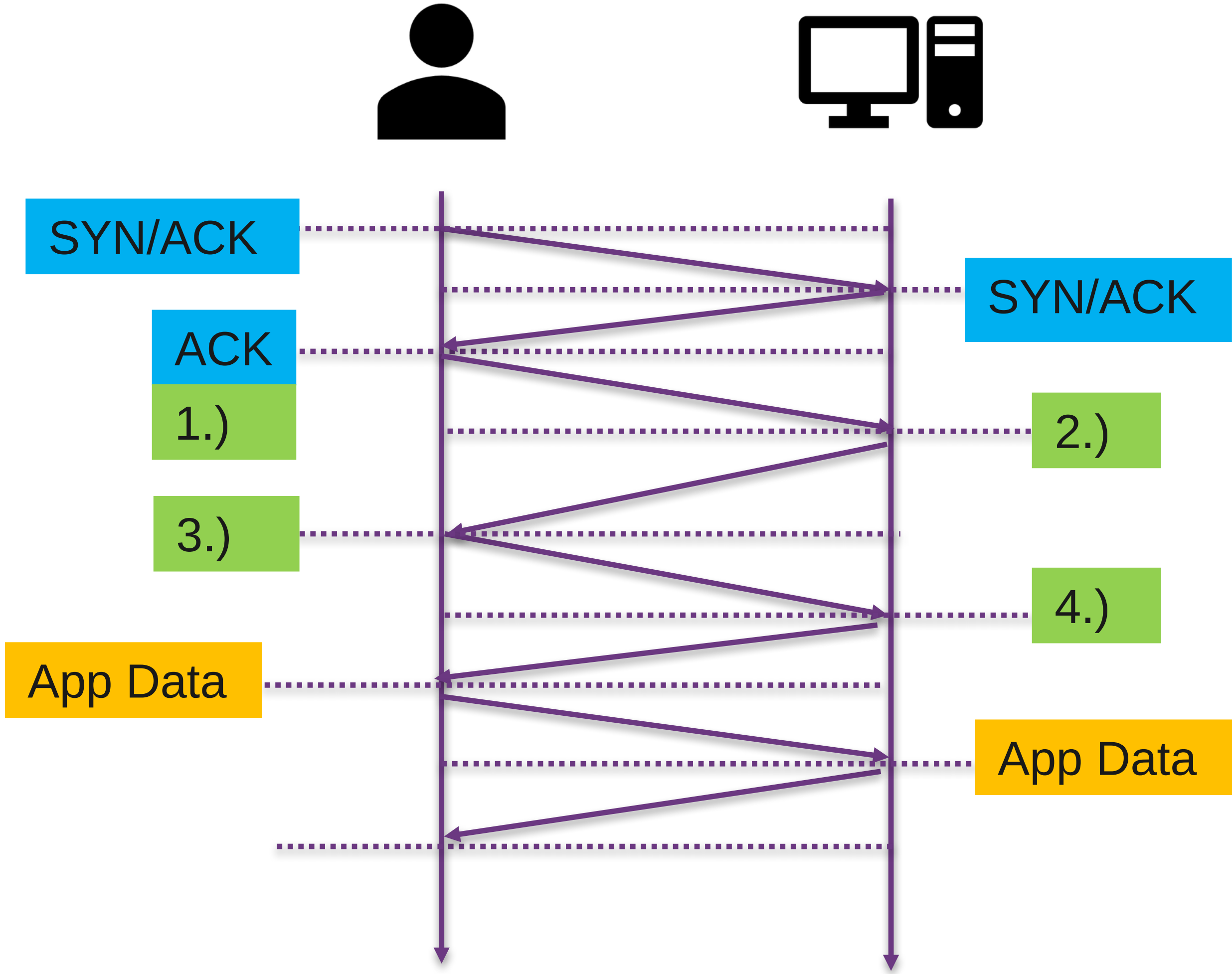
```
const net = require('net');
const server = new net.Server();
server.listen(8081, function() {
  console.log('Launching server...');
});

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



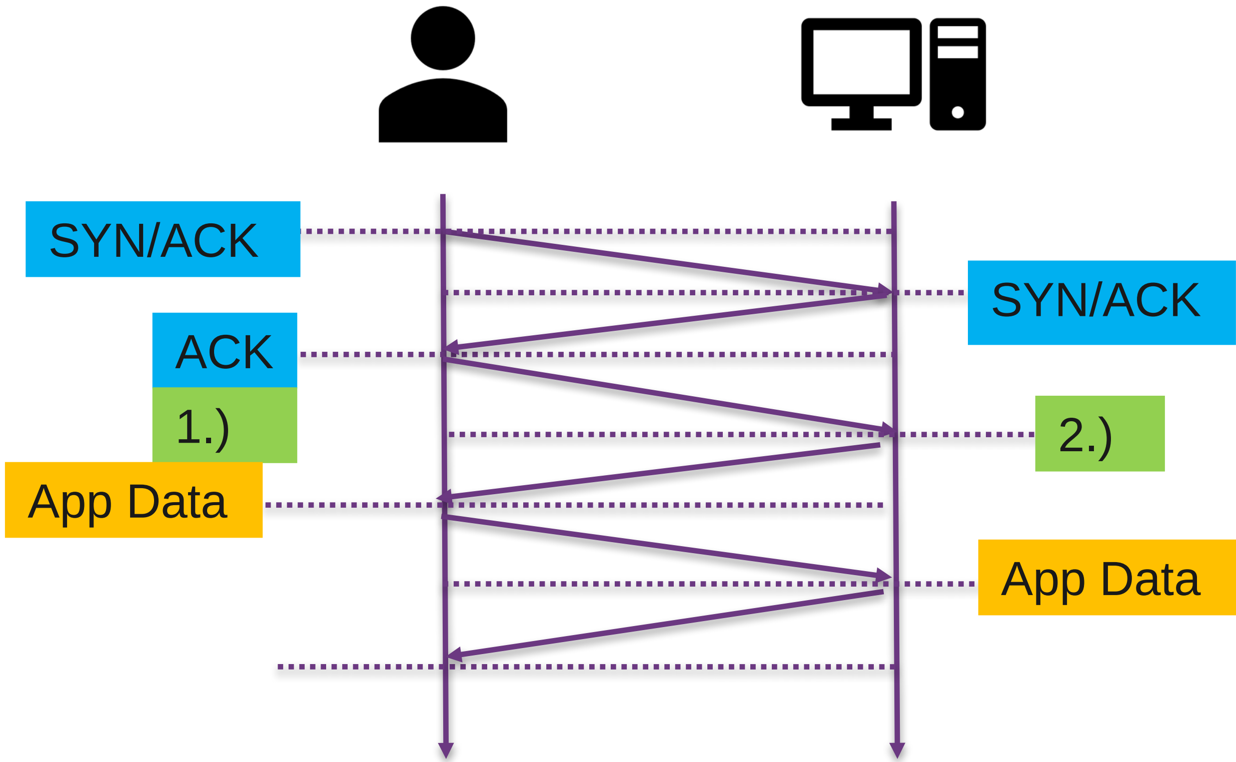
https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_8.0.0/com.ibm.mq.sec.doc/q009930_.htm

OST

```
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

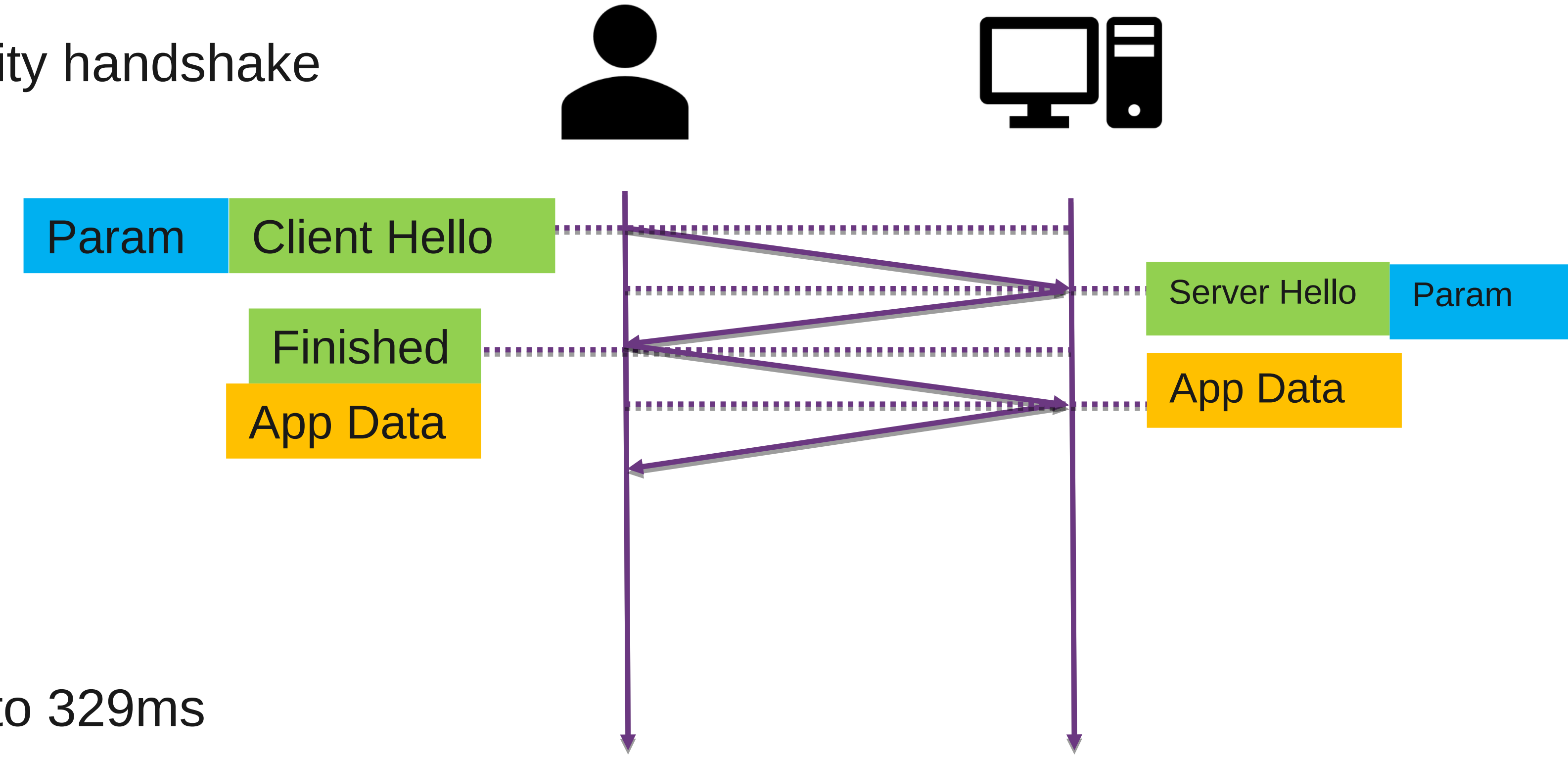
- Ping to Australia: 329ms
  - One way ~ 165ms
- TCP + TLS handshake:
  - 3RTT = 987ms! No data sent yet
- TLS 1.3, finished Aug 2018
  - 1 RTT instead of 2
    - 1.) Client Hello, Key Share
    - 2.) Server Hello, key Share, Verify Certificate, Finished
  - 0 RTT possible, for previous connections,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



OST

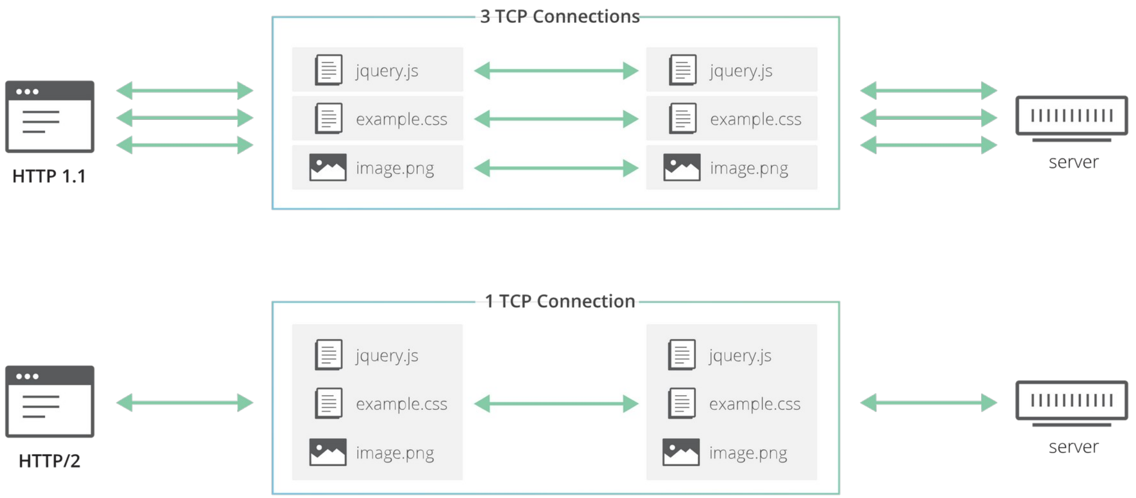
QUIC / HTTP/3

- QUIC: 1RTT connection + security handshake
  - For known connections: 0RTT
  - Built in security
  - “Google's 'QUIC' TCP alternative slow to excite anyone outside Google” [link] (9%, 25%, 75%)
    - Facebook
    - Cloudflare, state of HTTP
- Example Australia: from 987ms to 329ms



QUIC / HTTP3

- Multiplexing in HTTP/2
  - HTTP/1 → HTTP/2
- HTTP/2: Head-of-line blocking
  - One packet loss, TCP needs to be ordered
  - QUIC can multiplex requests: one stream does not affect others
- HTTP/3 is great, but…
  - NAT → SYN, ACK, FIN, conntrack knows when connection ends, not with QUIC, timeouts, new entries, many entries
  - HTTP header compression, referencing previous headers
  - Many TCP optimizations



source: https://blog.cloudflare.com/the-road-to-quic/