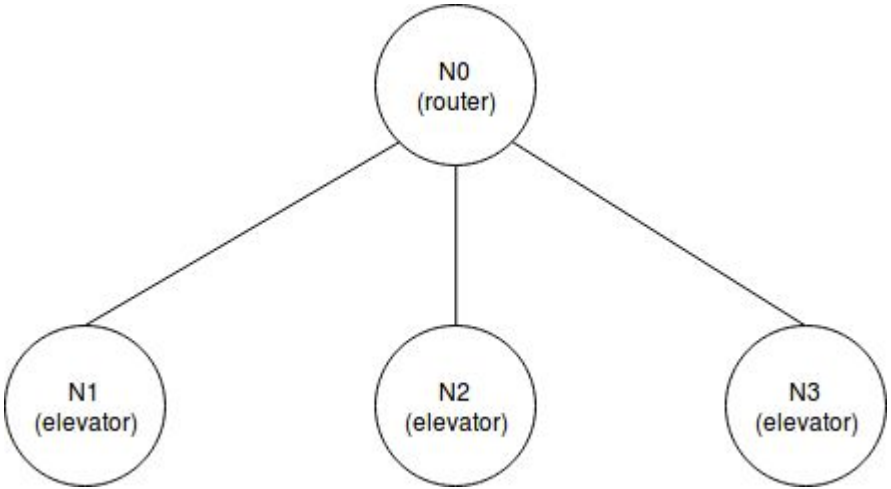
Network

(for the elevator project)

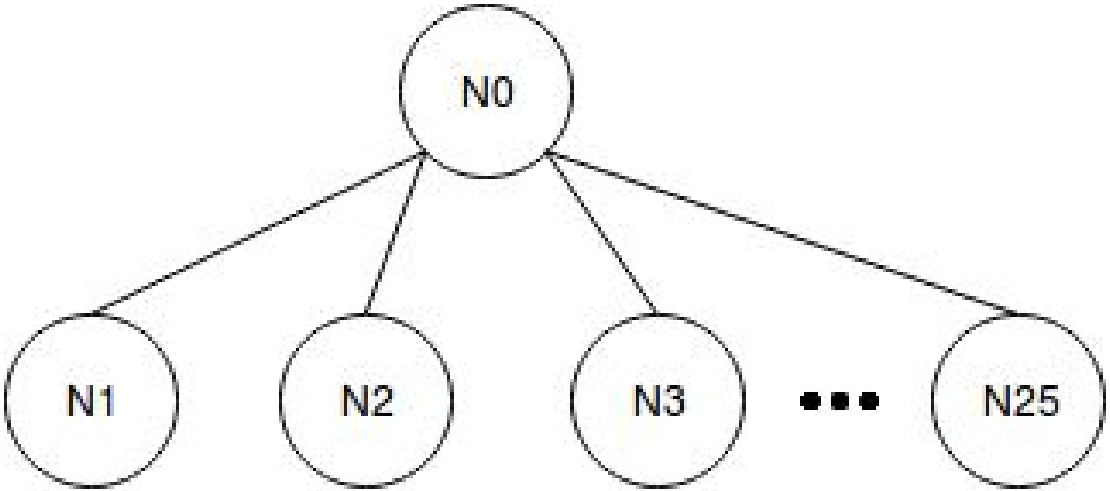
Network basics

- A collection of links and nodes.
- Nodes can send data to other nodes through links.
- A node can source, sink or relay data.

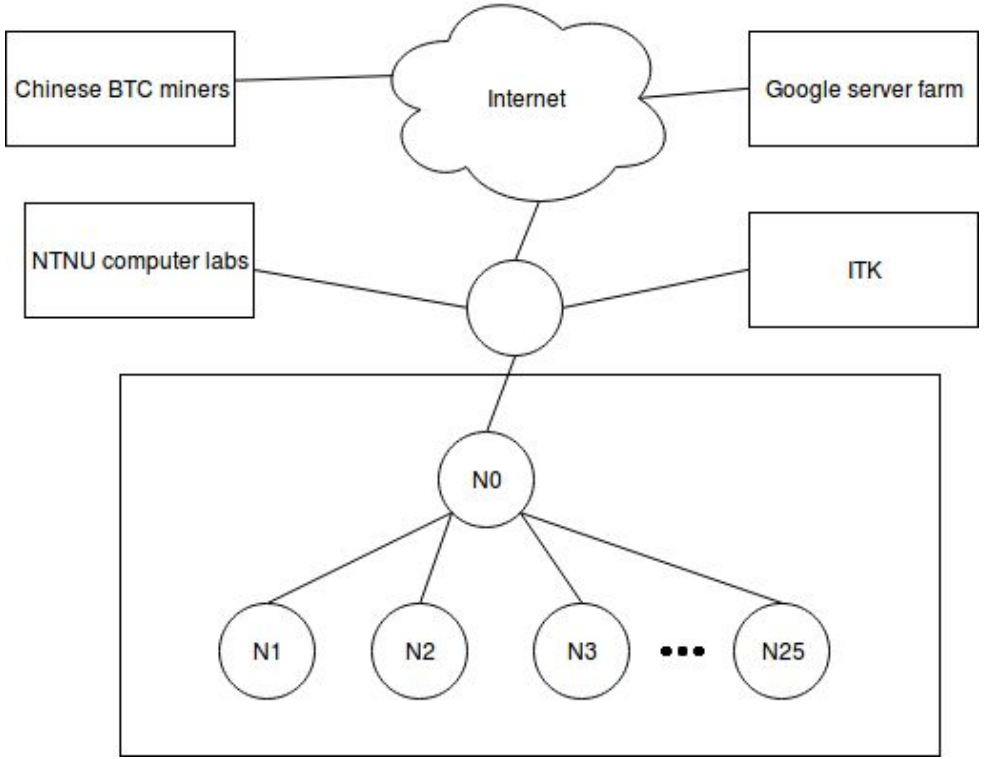
Elevator network



Elevator network

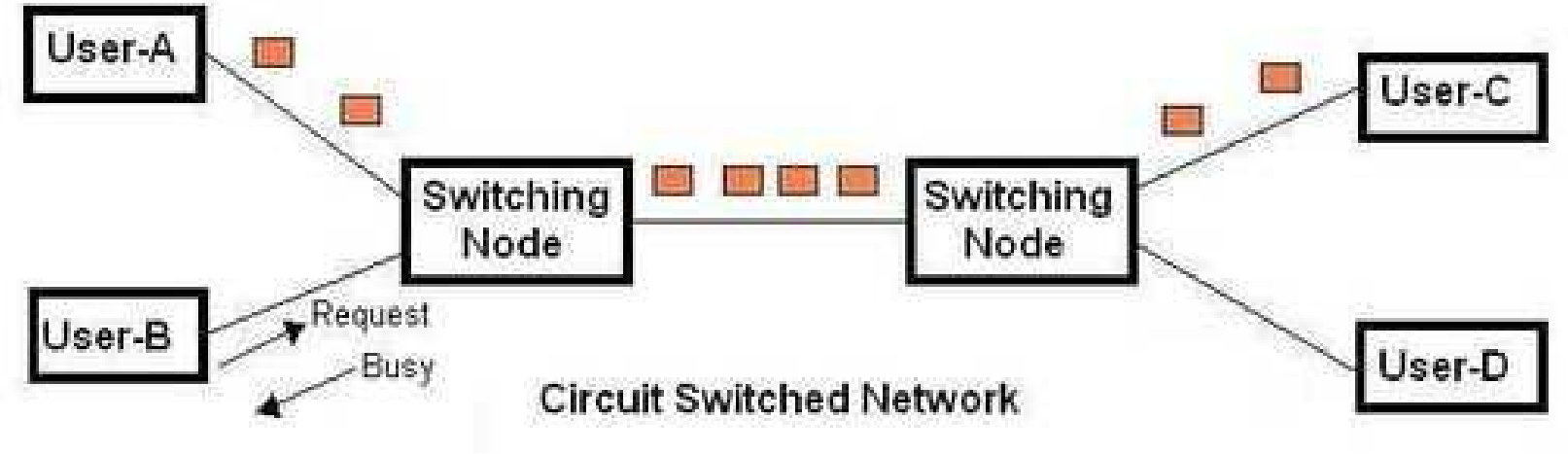


Elevator network



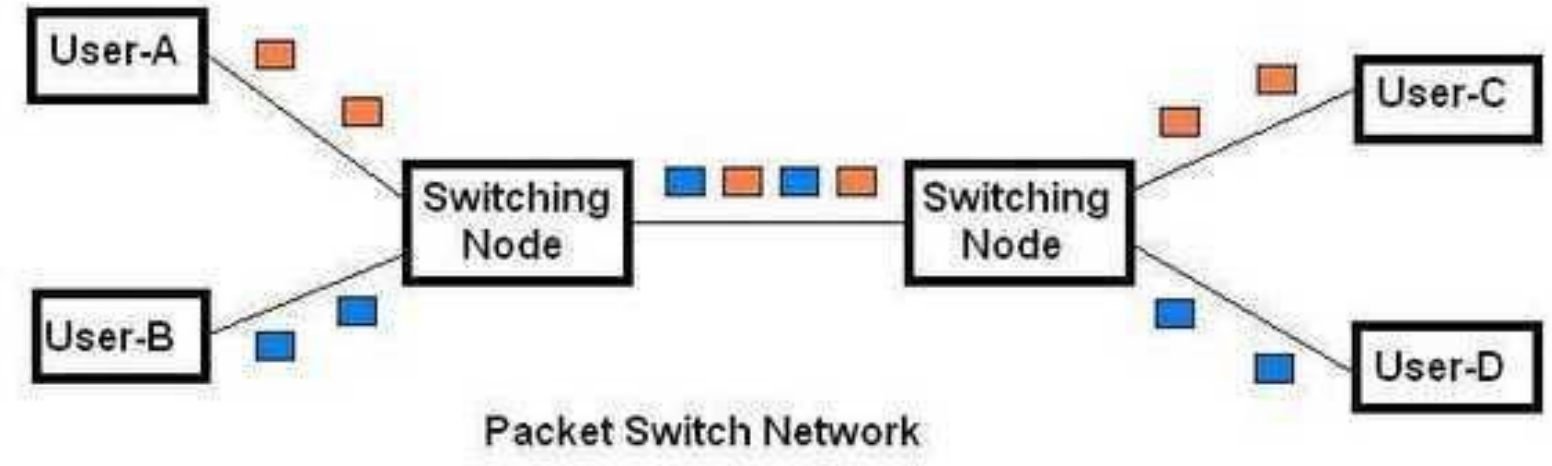
Network basics

- A network can be circuit switched (CS) or packet switched (PS).



Network basics

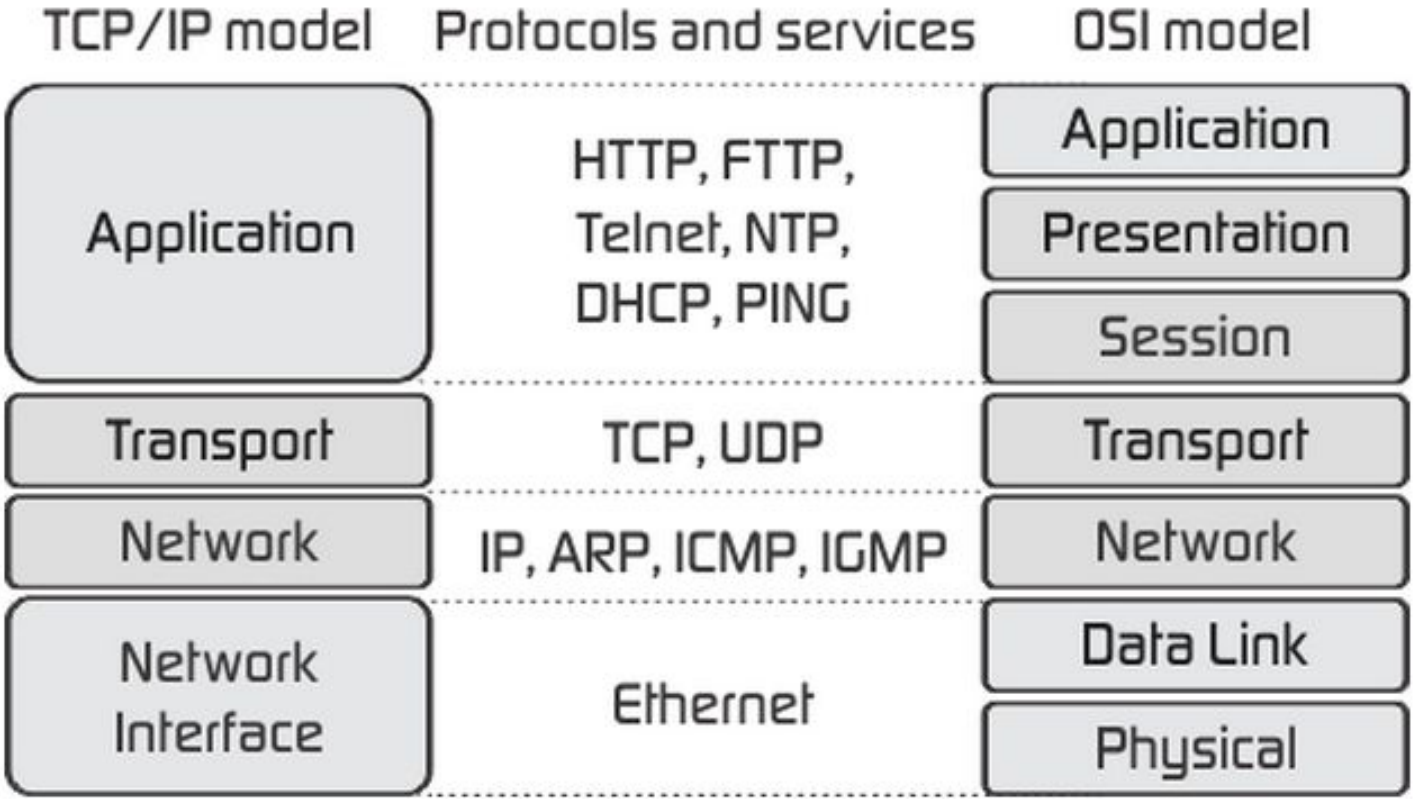
- A network can be circuit switched (CS) or packet switched (PS).



Network basics

- A network can be circuit switched (CS) or packet switched (PS).
- Circuit switched
  - Dedicated path.
  - Once a path is established, bandwidth can be guaranteed.
  - Bandwidth is fixed (need to be dimensioned for worst case).
  - May refuse to setup connection if resource allocation fails.
- Packet switched
  - No dedicated path (can follow any route).
  - No bandwidth guarantees.
  - Extra bandwidth will used if available.
  - No initial resource allocation is required.

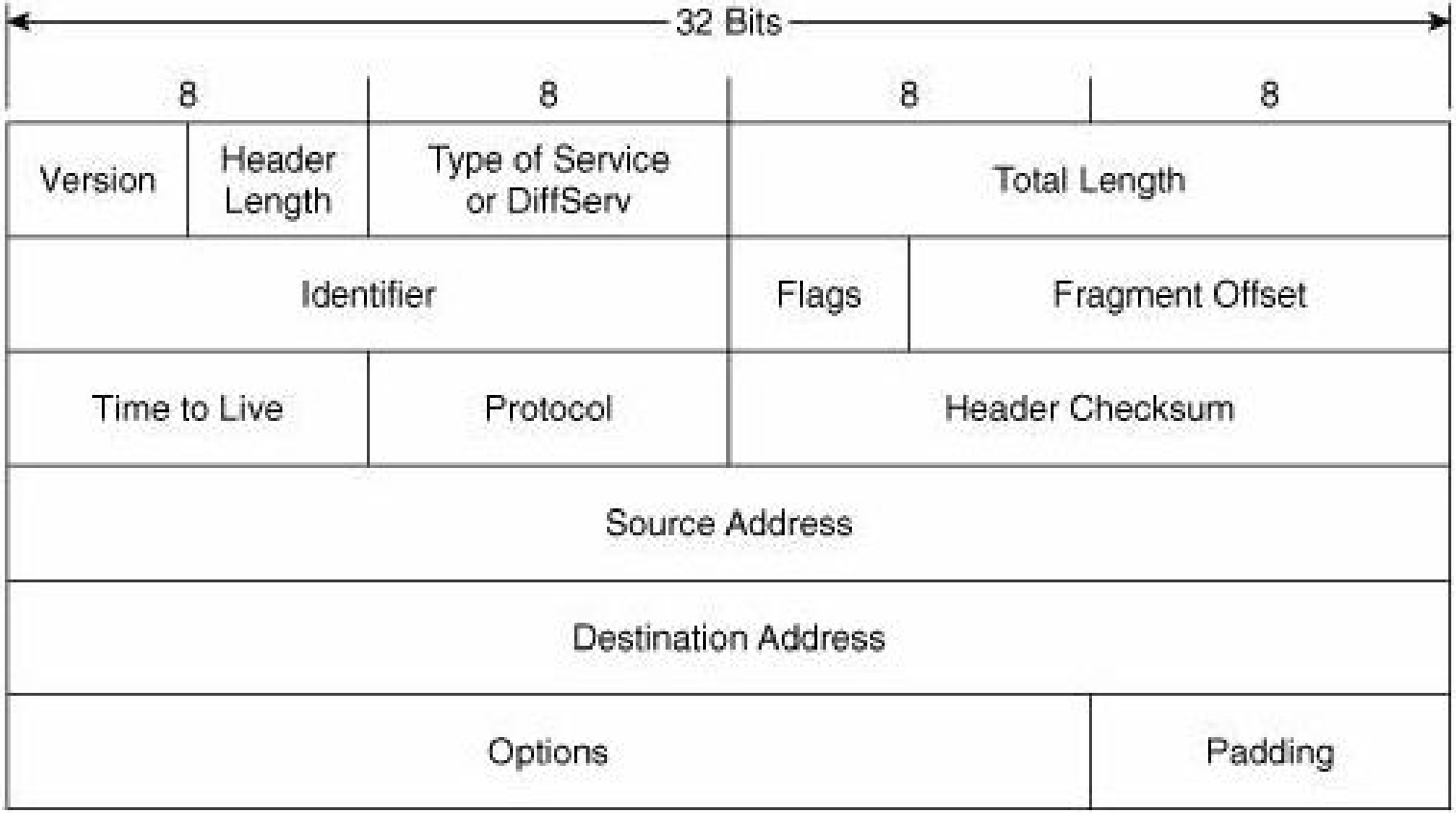
Network stack



Internet Protocol (IP)

- Packet switched.
- Every node (computer) have a 4 byte (32 bits) address (for IPv4).
  - A number of bits in the IP address represent the Subnet.
  - IPv6 uses 16 bytes addresses.
- Packets are routed based on the IP address.
  - Both between subnets and within subnets (in different ways)
- Internet operates after a best effort principle.
  - No guarantees!

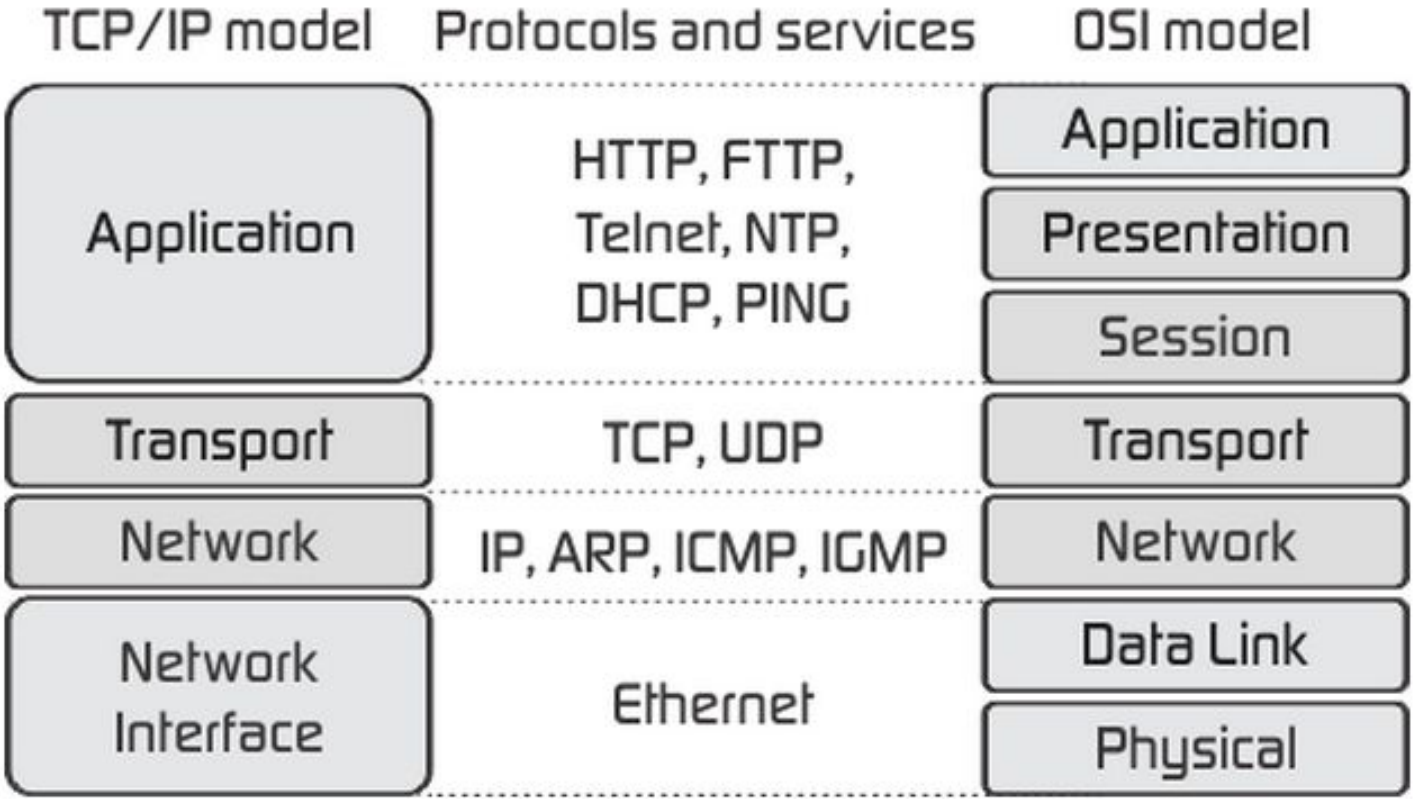
Internet Protocol (IP)



Transport Layer

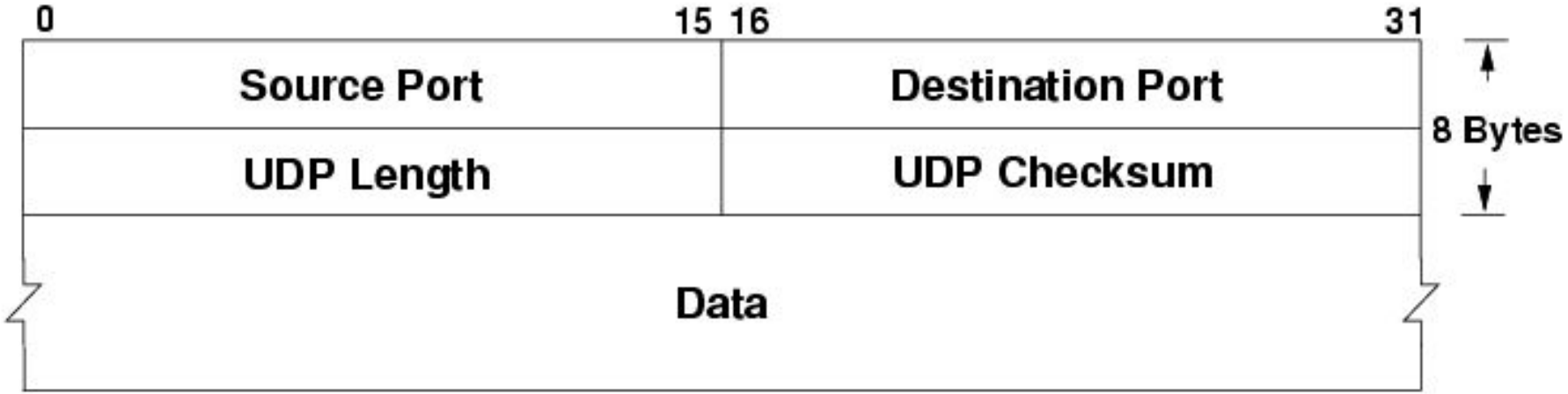
- We could put the data in an IP packet and send it to another node but:
  - What if we run many applications on the same node:
    - We need a way of multiplexing packets between applications
  - Sometimes it’s better to think of data in streams instead of packets
    - Video streaming!
    - Web page (what if your web page doesn’t fit in an IP packet)
  - Flow and congestion control
    - We can often improve performance by not congesting the network.
  - Ordered delivery and resending
    - If you’re doing a file transfer, receiving the packets in order is a nice feature.

Network stack



UDP protocol (user datagram protocol)

- Minimal layer on top of IP
- Contains multiplexing in the form of port numbers.
- Contains a checksum that will detect corrupt packets.



UDP - Failure modes

- Sent messages never arrives at destination.
  - This is within specs.
- What can cause a packet to drop?

UDP - Failure modes

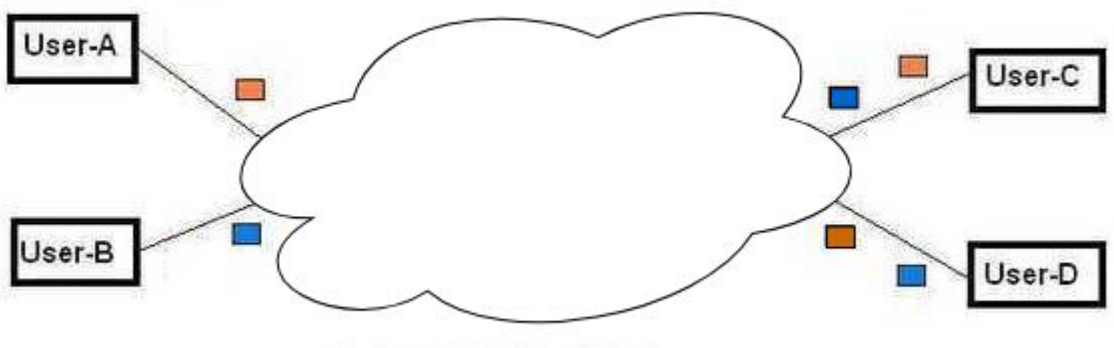
- Sent messages never arrives at destination.
  - This is within specs.
- What can cause a packet to drop?
  - A router may decide to drop a packet due to full receive buffer.
  - Bit flip (checksum will cause drop).
  - Cut cable.
- How will this influence the elevator system?

UDP - Failure modes

- Sent messages arrives in “weird” order.
  - This is also within specs.
  - Two packets with same src and dst can be routed differently.
- How will this influence the elevator system?

UDP - Failure modes

- Sent messages arrives in “weird” order.
  - This is also within specs.
  - Two packets with same src and dst can be routed differently.
- How will this influence the elevator system?



TCP protocol (Transmission Control Protocol)

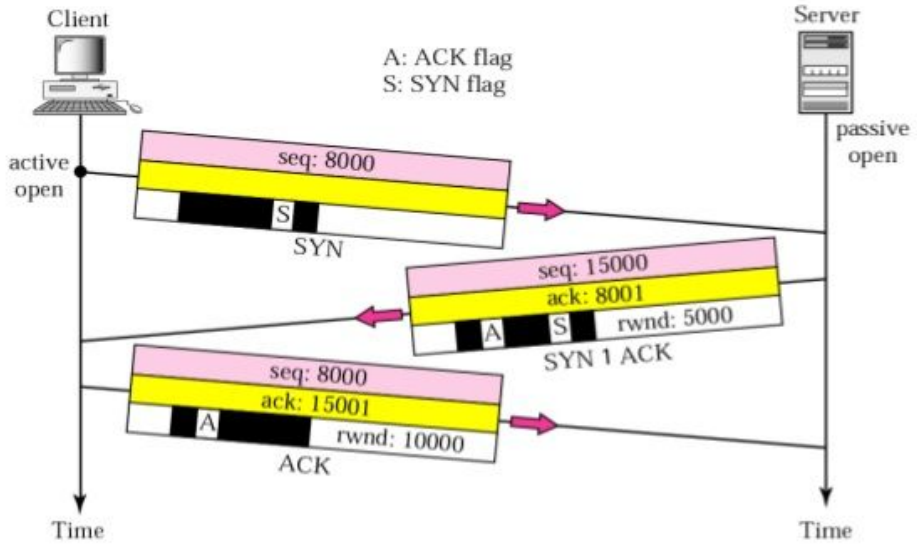
- Multiplexing of stateful connections with port numbers
- Checksum to detect corrupt packets and resending of lost packets.
- Guarantee reception in transmitted order (or no reception).
- Flow control and congestion control.

TCP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				N S	C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I N	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

Three way handshake

Figure 12.9 *Connection establishment using three-way handshaking*

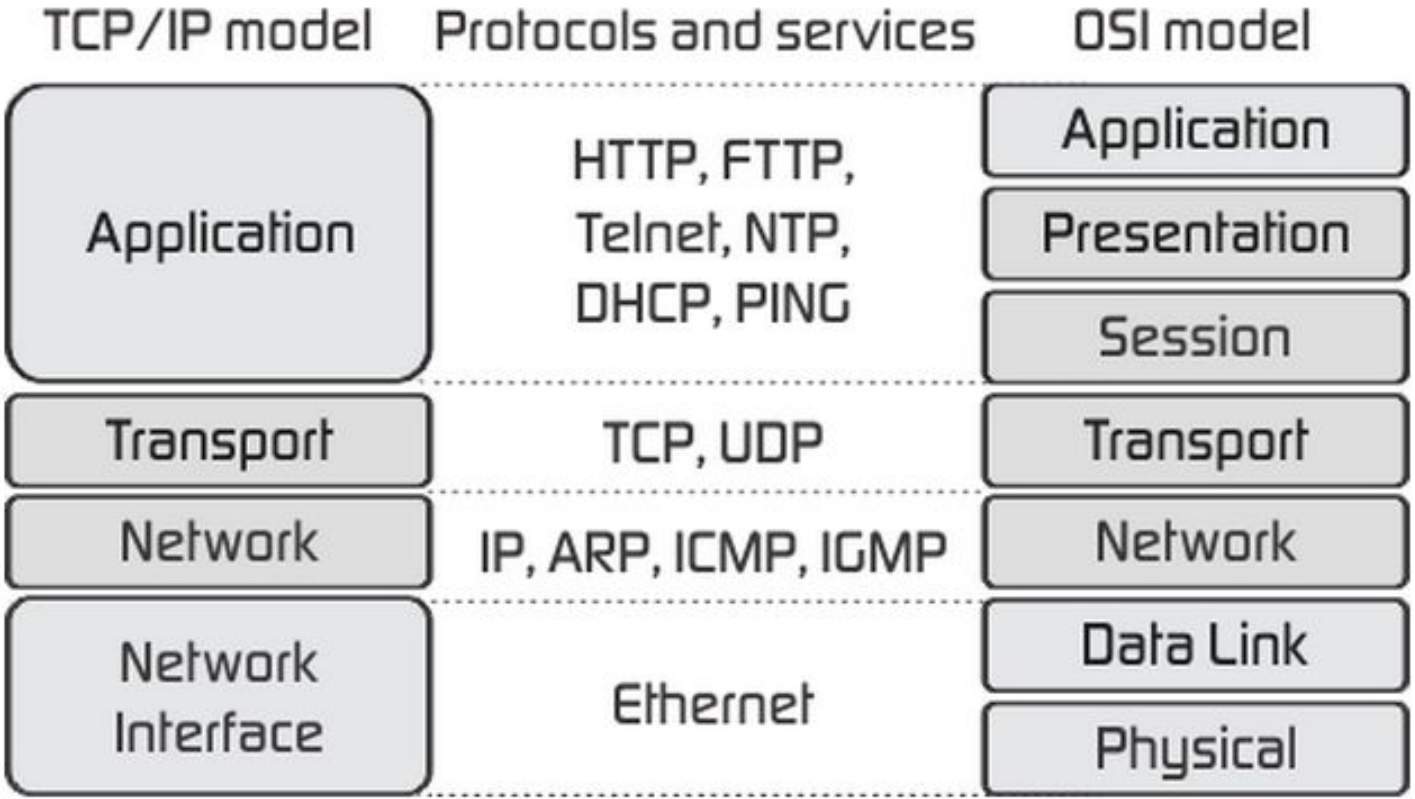


"Hi, I'd like to hear a TCP joke."
"Hello, would you like to hear a TCP joke?"
"Yes, I'd like to hear a TCP joke."
"OK, I'll tell you a TCP joke."
"Ok, I will hear a TCP joke."
"Are you ready to hear a TCP joke?"
"Yes, I am ready to hear a TCP joke."
"Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline."
"Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline."
"I'm sorry, your connection has timed out.
...Hello, would you like to hear a TCP joke?"

TCP - Failure modes

- Sent messages never arrives at destination.
- There’s two upsides.
  - Retransmission.
  - You will be notified when the connection time out.

Network stack

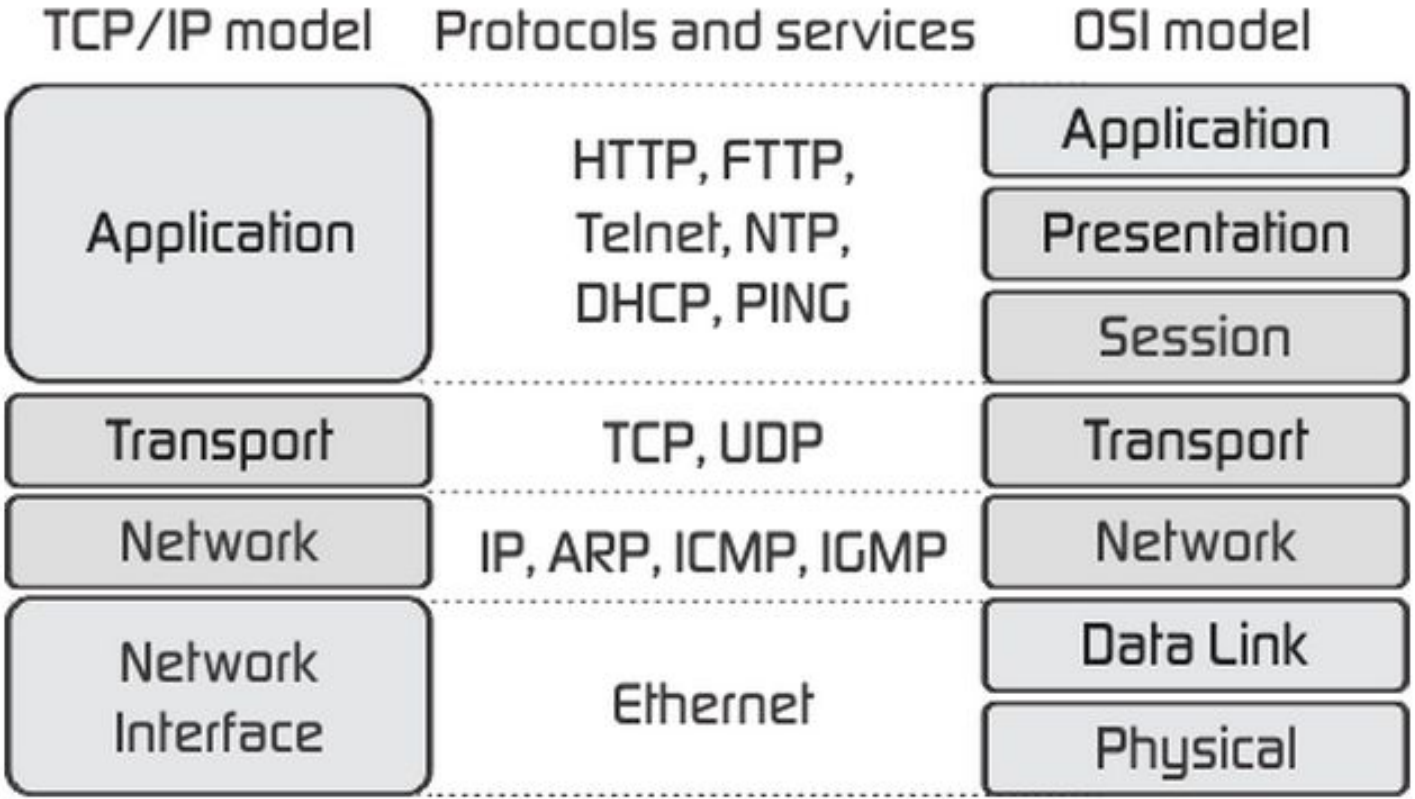


JSON (JavaScript Object Notation)

- Text based representation of data.
- Not only for JavaScript!
- Support the most common data types
- Allows serializing and transmitting composite data types over the network easily.

```
{
    "Rail Booking": {
        "reservation": {
            "ref_no": 1234567,
            "time_stamp": "2016-06-24T14:26:59.125",
            "confirmed": true
        },
        "train": {
            "date": "07/04/2016",
            "time": "09:30",
            "from": "New York",
            "to": "Chicago",
            "seat": "57B"
        },
        "passenger": {
            "name": "John Smith"
        },
        "price": 1234.25,
        "comments": ["Lunch & dinner incl.", "\"Have a nice day!\""]
    }
}
```

Network stack

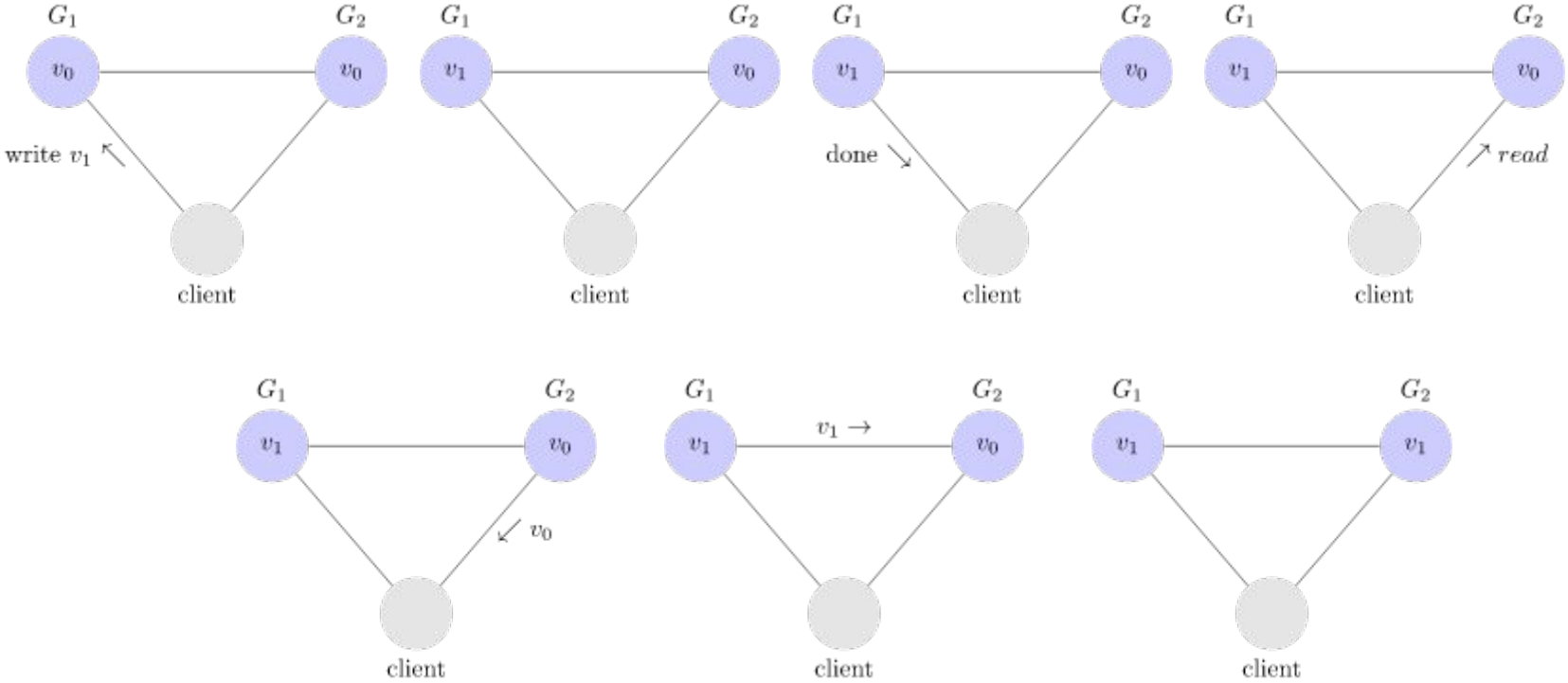



And now for something completely different

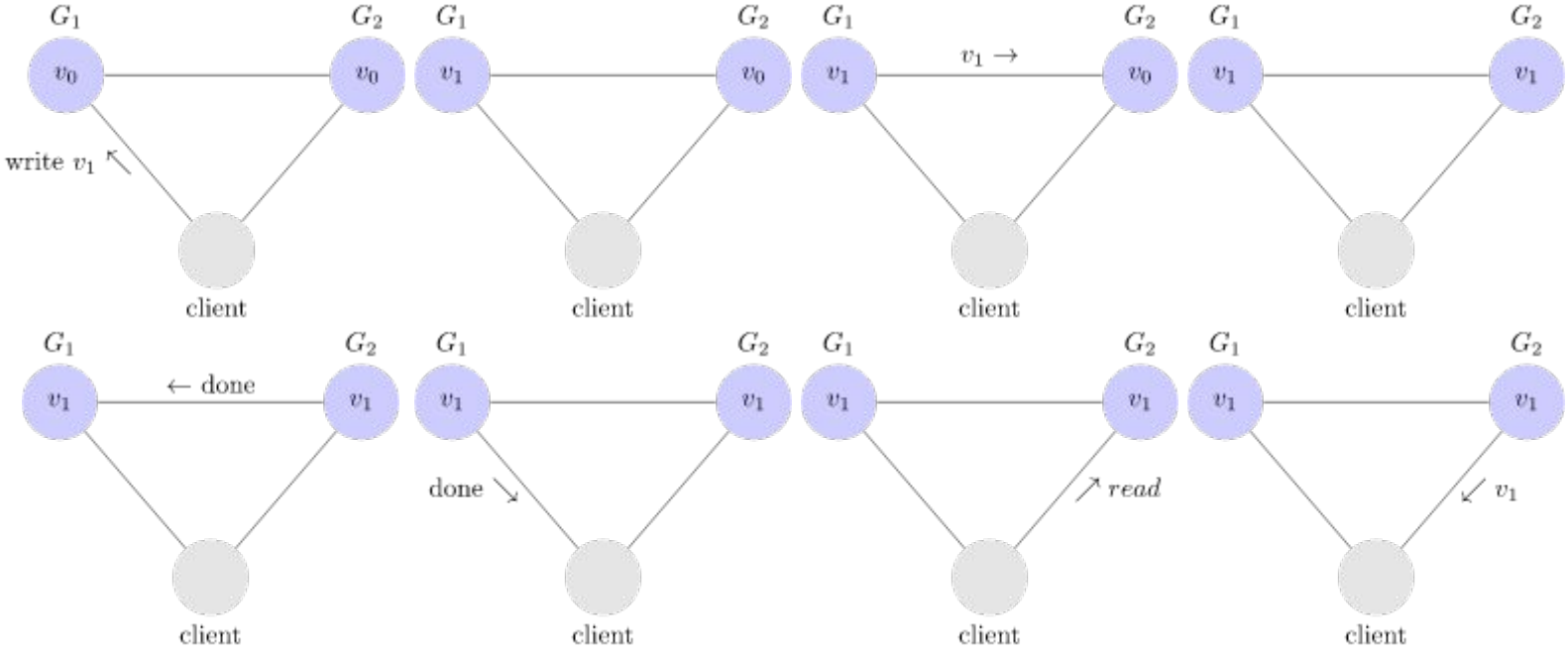
CAP Theorem

- For a distributed system (distributed data store)
  - Consistency:
    - Any read operation that begins after a write operation completes must return that value, or the result of a later write operation.

Inconsistency Example



Consistency Example

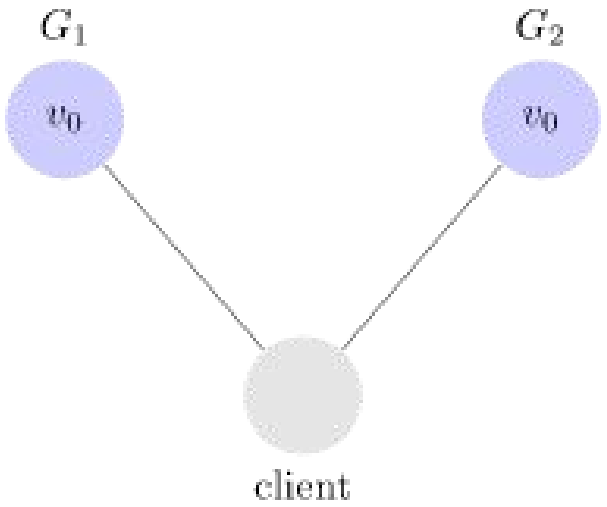


CAP Theorem

- For a distributed system (distributed data store)
  - Availability:
    - Every request received by a non-failing node in the system must result in a response.

CAP Theorem

- For a distributed system (distributed data store)
  - Partition Tolerance:
    - The network will be allowed to lose arbitrarily many messages sent from one node to another.



CAP Theorem

- For a distributed system (distributed data store)
  - Consistency:
    - Any read operation that begins after a write operation completes must return that value, or the result of a later write operation.
  - Availability:
    - Every request received by a non-failing node in the system must result in a response.
  - Partition Tolerance:
    - The network will be allowed to lose arbitrarily many messages sent from one node to another.
- And we can have maximum two of these at the same time.

Elevator project network module

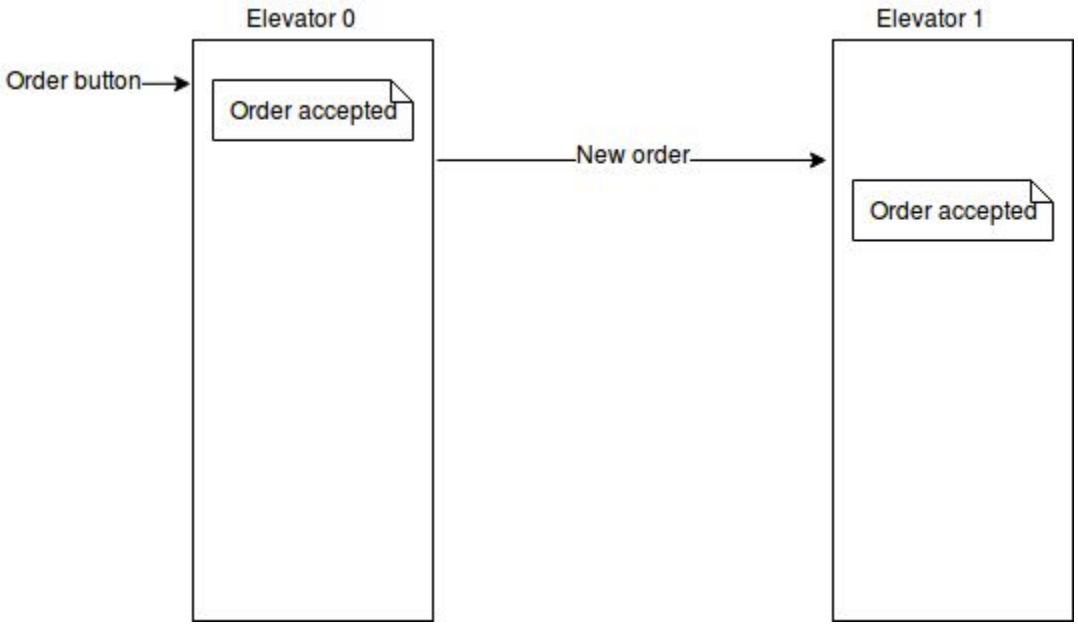
- In the elevator project we would a like to have a module that:
  - Synchronize data between elevators.
  - We can either send worldview or changes.
  - We need to know if the data has been received by others.

Elevator specification

- No network partitioning => consistency and availability.
- Network partitioning => availability or consistency?
- We don’t need consistency. We only need redundancy in the orders we accept.
- If it’s impossible to have redundancy => no availability

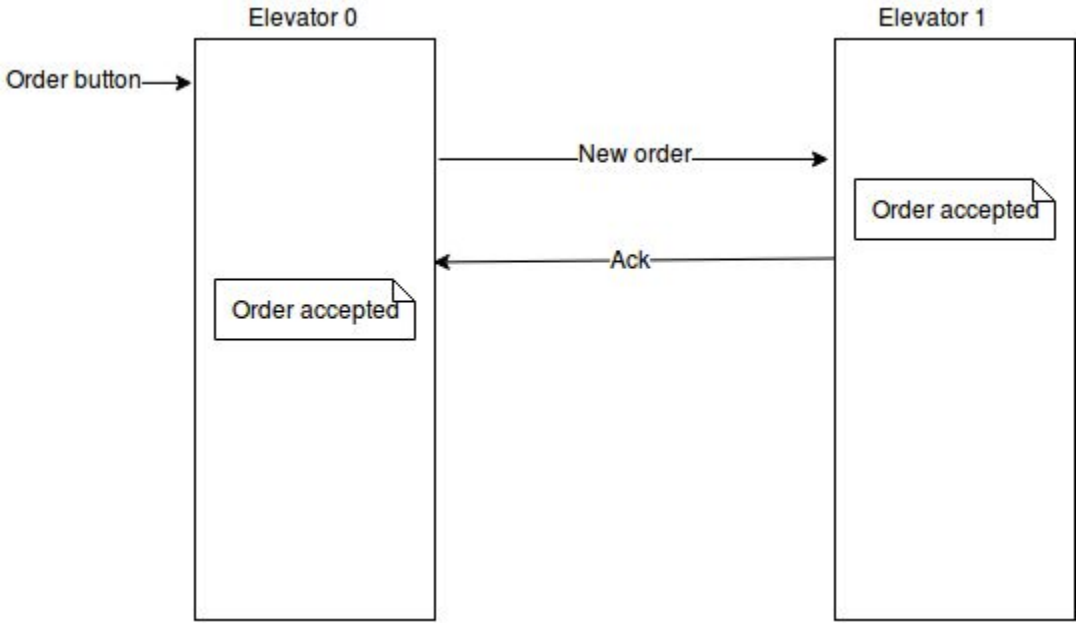
Redundancy in the elevator project.

We need to know that any order will be executed even if a single fault occur.



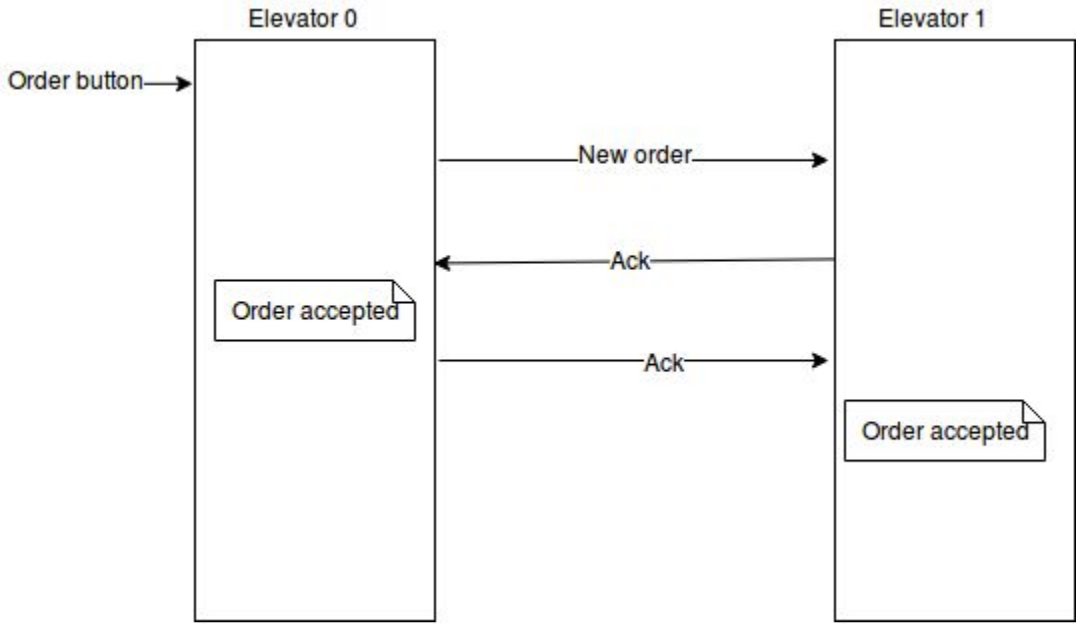
Redundancy in the elevator project.

We need to know that any order will be executed even if a single fault occur.



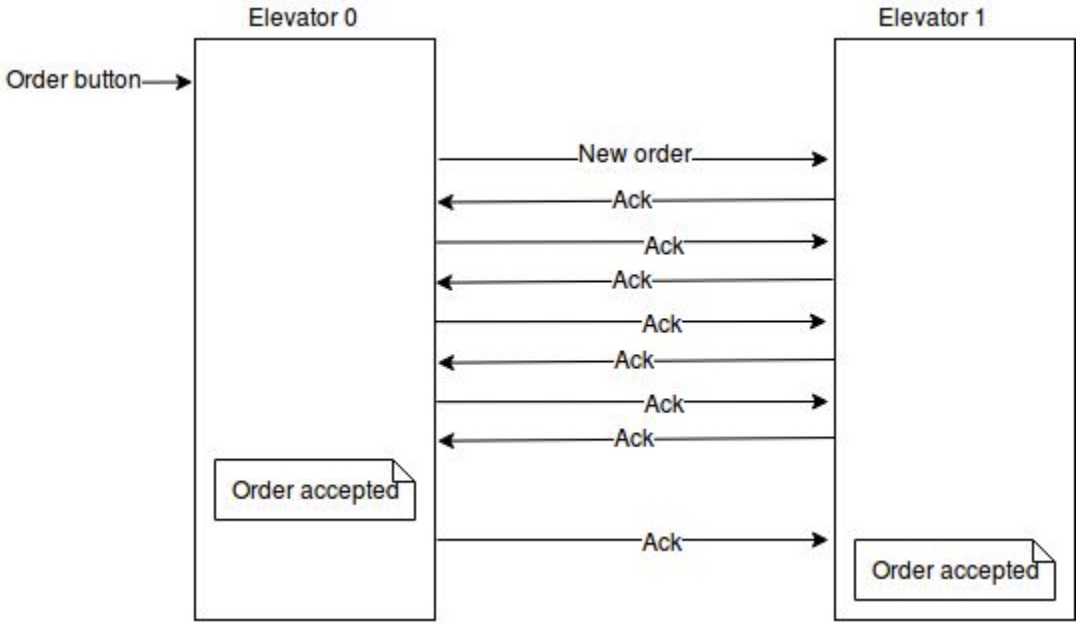
Redundancy in the elevator project.

We need to know that any order will be executed even if a single fault occur.



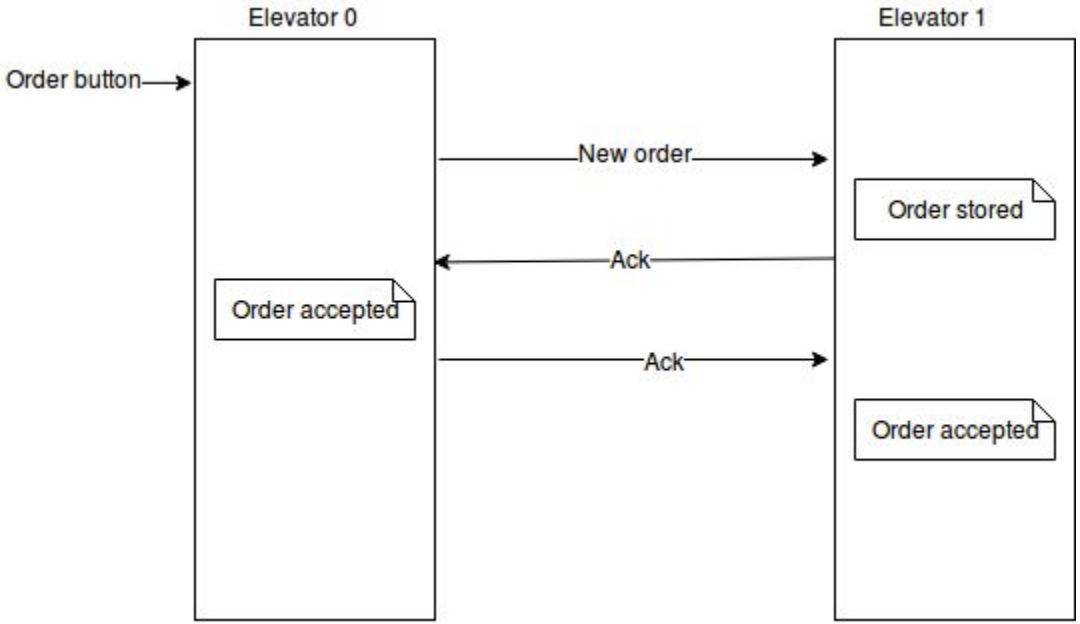
Redundancy in the elevator project.

We need to know that any order will be executed even if a single fault occur.



Redundancy in the elevator project.

We need to know that any order will be executed even if a single fault occur.



Redundancy in the elevator project.

We need to know that any order will be executed even if a single fault occur.

3.3. Assume that multiple errors will not happen simultaneously. More specifically, assume that you have sufficient time to do necessary preparation before the next error will occur as long as you detect the first error and recover from it within reasonable time.

3.4. Even though errors will not happen simultaneously they can be "active" simultaneously. For instance when the second error happens before the first one is resolved.

3.5. Loss of packets in UDP is not regarded as an error. UDP is in nature unreliable and can rightfully drop arbitrary packets. As a consequence of this, you may encounter an error at the same time as UDP packets are dropped.

Consensus without “ack”

- We have three actors.
- They want to keep track of one elevator order.
- They constantly send their own order status.
- And they read the other actors order status.

Ack-less consensus (take 1)

A	0																
B	0																
C	0																

- Everyone starts without the order

Ack-less consensus (take 1)

A	0	1															
B	0	0															
C	0	0															

- A receives an order (but does not accept it)

Ack-less consensus (take 1)

A	0	1	1	1													
B	0	0	0	1													
C	0	0	0	0													

- B notices the order and adds it to its own list

Ack-less consensus (take 1)

A	0	1	1	1	1													
B	0	0	0	1	1													
C	0	0	0	0	1													

- C also notice the order.
- All nodes notice that all other nodes know about the order => accepted

Ack-less consensus (take 1)

A	0	1	1	1	1	1												
B	0	0	0	1	1	1												
C	0	0	0	0	1	0												

- Node C finishes the order

Ack-less consensus (take 1)

A	0	1	1	1	1	1	1	1	1	1	1							
B	0	0	0	1	1	1	1	1	1	1	1							
C	0	0	0	0	1	0	0	0	0	0	0							

- Node A and Node B never received the frame where Node C accepted
- They are forever de-synced
  - Node C is waiting for Node A and Node B to become 0
  - Node A and Node B is waiting for Node C to become 1

Ack-less consensus (take 2)

- Introduce a new state:
  - 0 means no order
  - 1 means order exists
  - 2 means order accepted
- Nodes can only progress if all other nodes have reached the last state

Ack-less consensus (take 2)

A	0	1	1	1													
B	0	0	1	1													
C	0	0	0	1													

- Node A receives an order
- It’s propagated to Node B and Node C

Ack-less consensus (take 2)

A	0	1	1	1	1	2												
B	0	0	1	1	1	2												
C	0	0	0	1	2	2												

- Node C progress to 2, A and B follows

Ack-less consensus (take 2)

A	0	1	1	1	1	2	0	0										
B	0	0	1	1	1	2	2	0										
C	0	0	0	1	2	2	2	0										

- When the order is served, the nodes progress to 0.

Ack-less consensus (take 2)

A	0	1	1														
B	0	0	1														

- Node A receives an order

Ack-less consensus (take 2)

A	0	1	1	1	1	2	2										
B	0	0	X	X	X	T	T										

- Before progressing to 2, a node crashes.
- The state message can be multi purpose
  - Tell other nodes about state
  - Heartbeat

Ack-less consensus (take 2)

A	0	1	1	1	1	2	2	2	0									
B	0	0	X	X	X	T	T	0	0									

- B reconnects with state 0
- A thinks that B have taken the order
- The order is lost

Ack-less consensus (take 3)

- 0: no order
- 1: order exists
  - Order is accepted when all elevators is in this state
- 2: ready to clear order

Ack-less consensus (take 3)

A	0	1															
B	0	0															

- Node A receives an order

Ack-less consensus (take 3)

A	0	1	1	1	1												
B	0	0	X	X	X												

- Before progressing to 2, a node crashes.

Ack-less consensus (take 3)

A	0	1	1	1	1	1	1											
B	0	0	X	X	X	T	T											

- When timeout is detected, the order is accepted

Ack-less consensus (take 3)

A	0	1	1	1	1	1	1	1	1									
B	0	0	X	X	X	T	T	0	1									

- If the crashed node returns, it will learn about the new order.

Ack-less consensus (take 3)

A	0	1	1	1	1													
B	0	0	1	1	2													
C	T	T	T	T	0													

- What if something like this happens?

Consensus without “ack” sum up

- It can solve a lot of problems.
- You need more than 2 states.
- A new node appearing will cause problems that you will need to handle.

Q&A design review

Ask me some questions...