Northeastern University - Seattle

CS6650 Building Scalable Distributed Systems

Professor Ian Gorton

Fall 2016

Building Scalable Distributed Systems

Week 3 – Distributed Systems Fundamentals

Outline

- Communications
- Remote Procedure Call Systems
- Partial Failures
- Timing and Ordering of Events
- Performance and scalability revisited

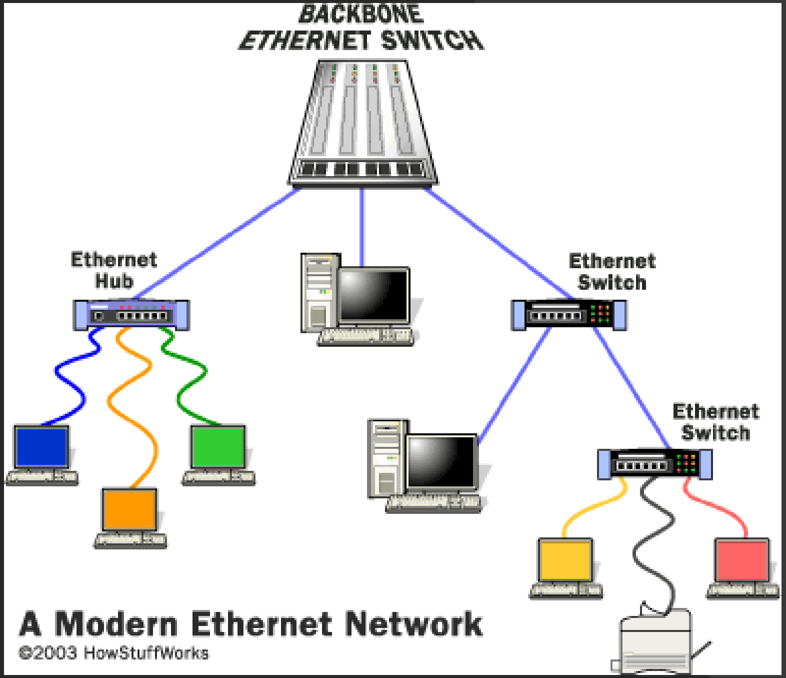
Distributed Systems Communications

Distributed Systems Communicate

- By definition …..
- No shared state, so communications required
    - Shared nothing systems
- This communications has an impact on the extent to which the systems can achieve particular properties such as:
    - Performance
    - Scalability
    - Availability

Shared Network

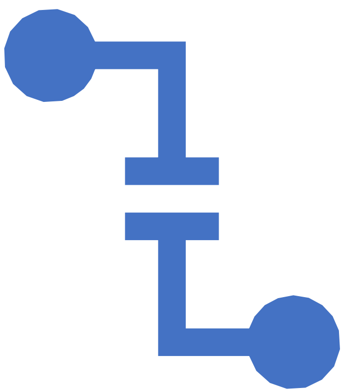
- The network is a shared resource
- You therefore often don’t have control over the load at any given time
    - Moreover you’re often not aware of the load at any given time
- This means that for the most part you can’t count on bandwidth being available



Networks are Fallible

- Things happen
  - Part of your data can be lost
  - Things can be delivered out of order
  - The connection might not be made at all
  - Data can be corrupted
  - Sometimes these conditions are ‘permanent’, sometimes transient

Network Partition



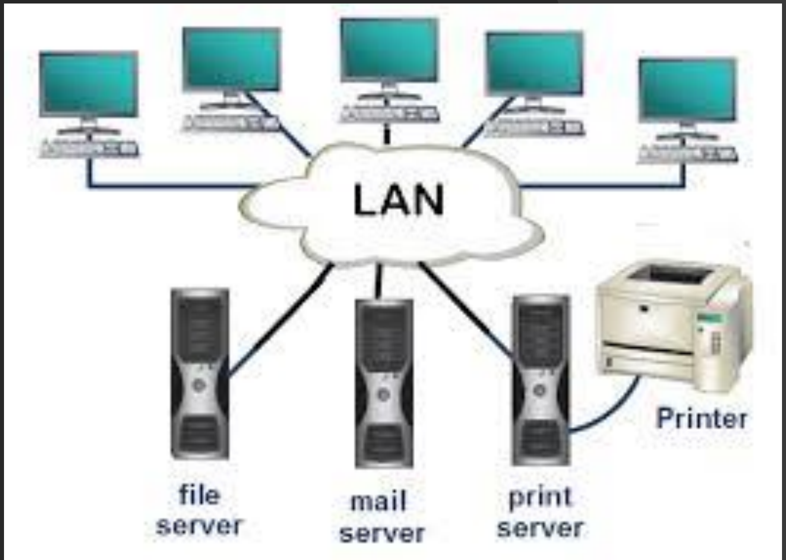
- What is a network partition?
  - A split in the network, such that full *n*-to-*n* connectivity is broken
  - i.e. not all servers can contact each other
- Partitions split the network into one or more disjoint subnetworks
- How can a network partition occur?
  - A switch or a router may fail, or it may receive an incorrect routing rule
  - A cable connecting two racks of servers may develop a fault
- Network partitions are very real, they happen all the time
- Analysis of network failures in Azure Data Centers

Network Characteristics

	Example	Range	Bandwidth (Mbps)	Latency (ms)
Wired:				
LAN	Ethernet	1–2 kms	10–10,000	1–10
WAN	IP routing	worldwide	0.010–600	100–500
MAN	ATM	2–50 kms	1–600	10
Internetwork	Internet	worldwide	0.5–600	100–500
	km			
Wireless:				
WPAN	Bluetooth (IEEE 802.15.1)	10–30m	0.5–2	5–20
WLAN	WiFi (IEEE 802.11)	0.15–1.5 km	11–108	5–20
WMAN	WiMAX (IEEE 802.16)	5–50 km	1.5–20	5–20
WWAN	3G phone	cell: 1—5	348–14.4	100–500

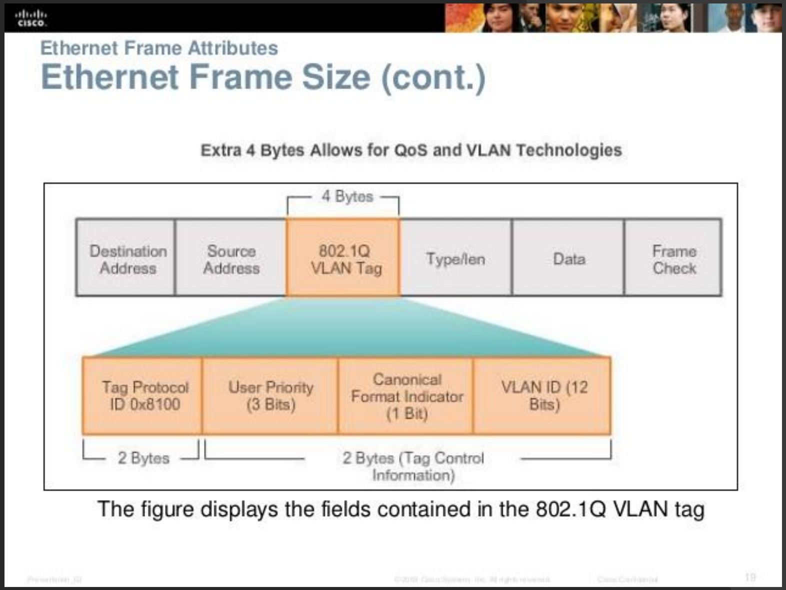
Local Area Networks

- These are networks that cover a limited geographic region
  - Within a building or a campus
- They have direct connection via private (non-leased) coaxial cable, twisted pair, fiber optic cable, wifi
- They typically support a single organization or entity
- Most LANs use Ethernet/wifi connections



How do LANs Route Messages?

- Each Ethernet connection has an address
- Data is sent in “frames”
    - Each frame has a source and destination address
- The frame is sent to every node on the LAN
- The node with an Ethernet address that matches the destination address will respond
    - The other nodes will ignore the frame
- The destination address could also be a “broadcast” address
    - This means that the message is intended for all nodes



Characteristics

Fast (low latency)

See upcoming slide

High data transfer rate

10 – 1000 Mbps

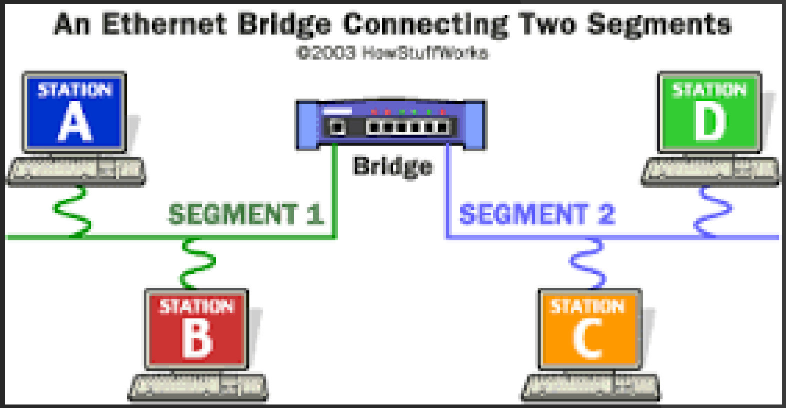
There are limits, however

Physical limitations on the geographic region covered by a single network

Also limits on the number of nodes on a single network

Segmentation

- In order to increase the capability of LANs they can be split into segments
- Segments are essentially LAN partitions that are connected via a bridge
- The bridge will only forward messages intended for a node on that segment



Latency Comparisons*

Activity	Latency
Main memory reference	100 ns (.0001 ms)
Send 1K over 1 Gbps network	1000 ns (0.01 ms)
Read 1 MB sequentially from SSD	1 ms
Read 4K randomly from SSD	15 ms
Read 1 MB sequentially from disk	20 ms
Send packet CA → Netherlands → CA	150 ms

- * dean-keynote-ladis2009_scalable_distributed_google_system

Things to Note

- Reading from cached memory on a fast Ethernet is faster than a disk read
- The latency for long distances governed by physics
    - Data travels at the speed of light
    - Slower actually due to network overheads/switching
- Physical location has a big impact on potential latencies

Provider	Direct Connect Location	Nearest AWS Region	Distance (miles)	Approximate Latency (ms)
CoreSite	New York, NY	N. Virginia	255	~16 ms
CoreSite	Los Angeles, CA	N. California	340	~20 ms
Equinix	Ashburn, VA	N. Virginia	< 20	< 4 ms
Equinix	San Jose, CA	N. California	< 20	< 4 ms
Equinix	Singapore	Singapore	< 20	< 4 ms
Equinix	Tokyo	Tokyo	< 20	< 4 ms
Telecity	London, UK	Ireland	330	~20 ms
Terremark	Sao Paulo	Sao Paulo	< 20	< 4 ms
Equinix	Sydney	Sydney	< 20	< 4 ms

Network Types

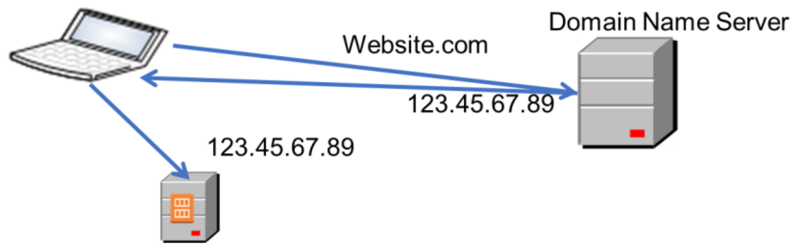
Synchronous networks:

- Transfer data ‘instantaneously’ based on a shared clock
- Full duplex (two way) – upload/download speeds typically the same
- Eg SONET, dedicated fibre optic strands (one each way)

Asynchronous networks:

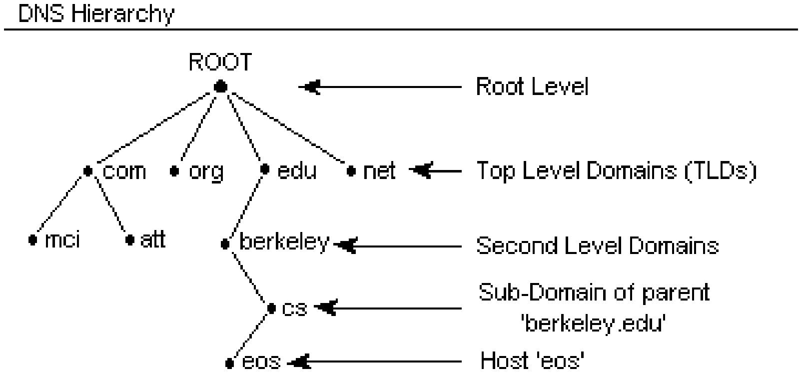
- No shared clock
- Half duplex
- Unbounded transmission delays
- E.g. The Internet we all use ☹

The Internet



- The internet uses IP addresses to route messages and locate hosts
- When you enter a URL it needs to be translated into an IP address before your request is routed to the host
- You will get the IP address from a Domain Name System (DNS)

DNS Hierarchy

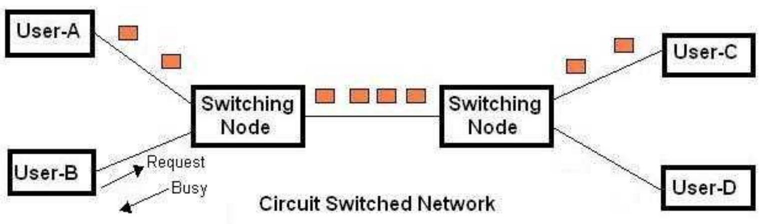


- Consider URL www.yahoo.com.au
    - If one server held all DNS -> IP mappings, it would both get overloaded and hold over 500+ million mappings.
    - Growing rapidly!!
- DNS is arranged as a hierarchy.
- There is an “authoritative” name server that holds all of the final suffixes (e.g. .au, .edu, .com, .co)
- It is replicated for performance reasons
- https://root-servers.org/

Finding www.yahoo.com.au

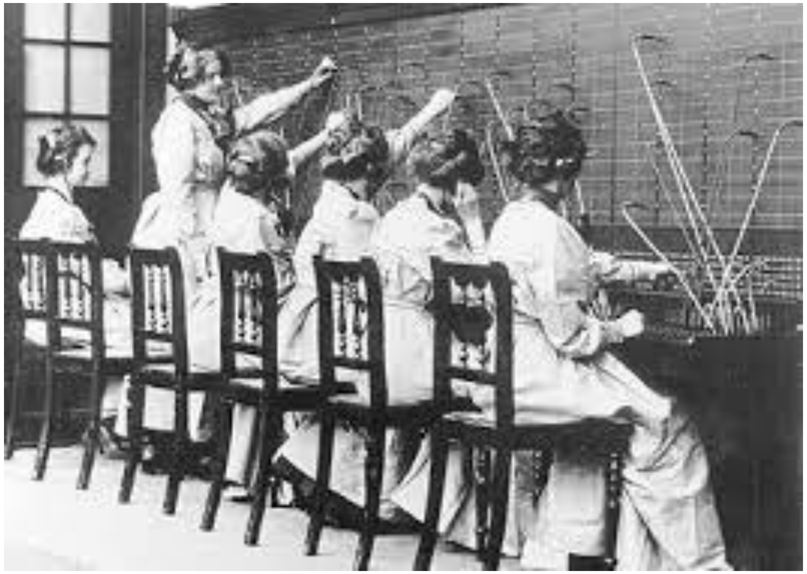
- The final suffix DNS has the IP of the .com DNS
- The .com DNS has the IP of .com.au's DNS
- From there the DNS will return the IP for interim DNS or the host itself
- The www.yahoo.com.au DNS, in turn, has IP for various local DNSs that are under Yahoo!'s control
- This allows Yahoo! to change the IP of the various local DNSs without changing anything up the hierarchy.
- Clients cache DNS lookups to improve performance

Packet Switching



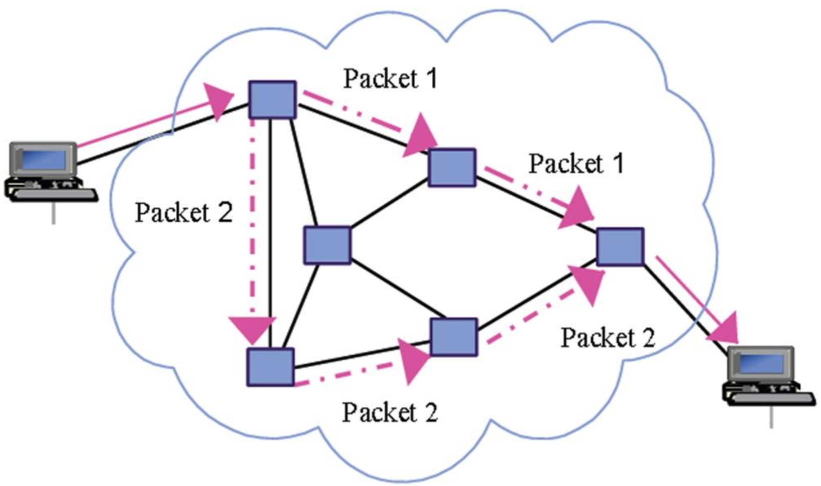
- In the ‘old days’ telephone companies used “**circuit switching**”
    - the network establishes a dedicated communications channel through the network
    - Remember “switchboard operators” (in movies?)?
- **Packet switching** bundles data into “packets” and then routes these packets independently to the destination

Benefits



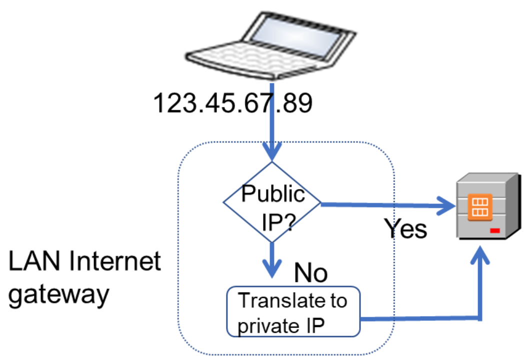
- More efficient
  - With circuit switched networks no one else can use the circuit until free
- Flexibility
  - Packet switched networks are easy to change
  - Packets find their way dynamically to the destination with the aid of routers

Packet Switching



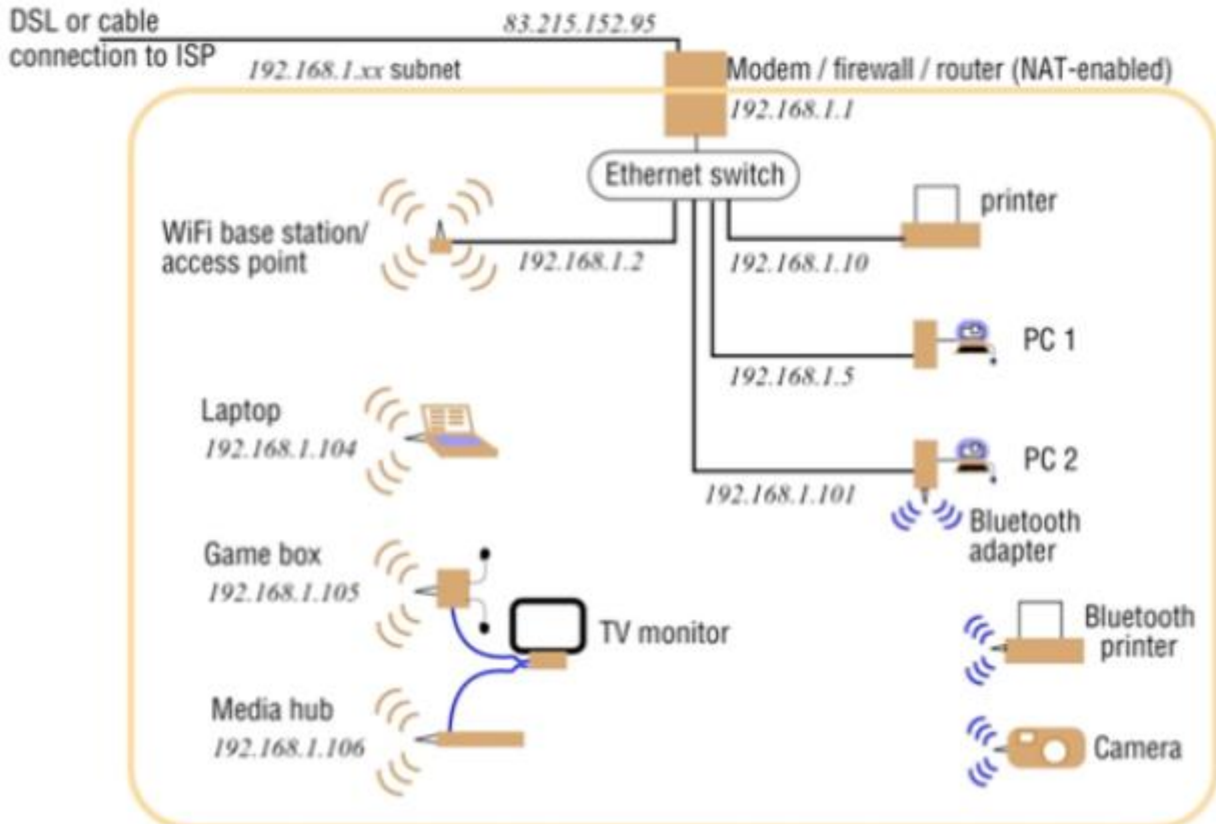
- Communication channels are shared
- Channels are dynamically determined based on availability of routes, load, and other factors
- Packets can travel independent channels
  - Arrival rate for packets can vary
  - Failure rates for packets can vary
- These issues need to be addressed in some way

Assigning IP addresses



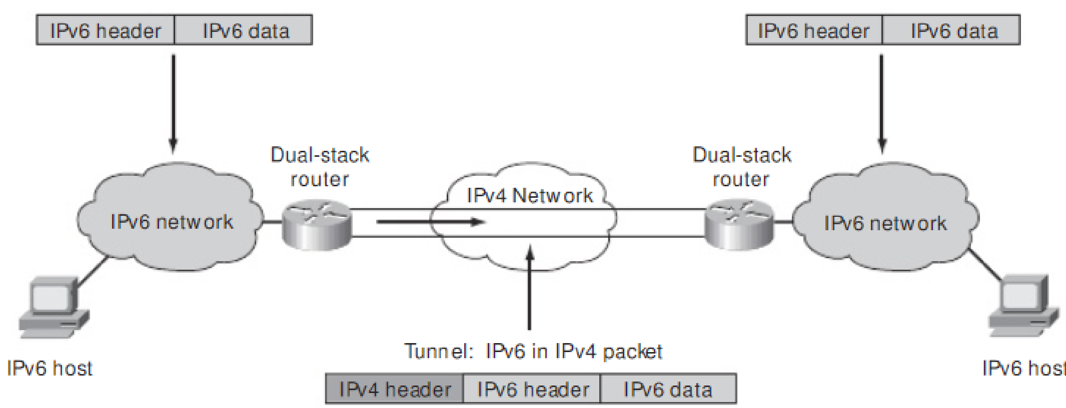
- Every “device” on the internet gets an IP
  - This includes virtual machines in the cloud and mobile devices
- IP address can be
  - Private and not seen outside of the LAN
  - Public and directly addressable from outside of the LAN
- An IP message has a header and a payload. The header includes
  - IP address of the source
  - IP address of the destination

Message Routing



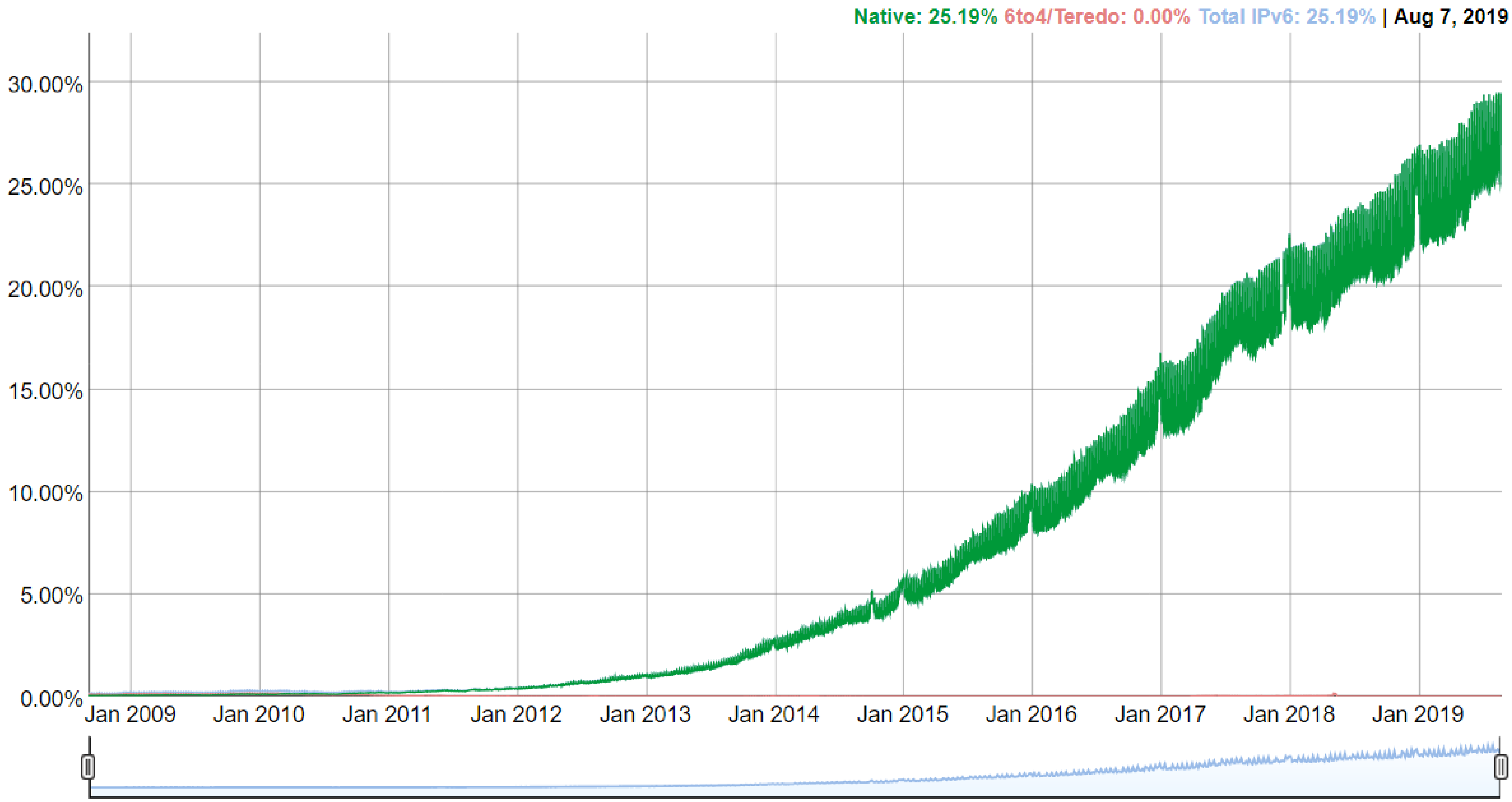
IPv4 and IPv6

- An IP (Internet Protocol) address is a numerical label that identifies a "device" on the internet.
  - IPv4 is 32 bits long - xxx.xxx.xxx.xxx
  - IPv6 was created in 1995 and it has 128 bits = more devices
- June 8, 2011 was designated as world IPv6 day where top websites and internet providers had a 24 hour test of IPv6 infrastructure. This test was successful.
- Google publishes statistics for percentage of users that access Google over IPv6. https://www.google.com/intl/en/ipv6/statistics.html
- As IPv4 and IPv6 networks are not directly interoperable, transition technologies (eg tunneling) permit hosts on either network type to communicate with any other host.



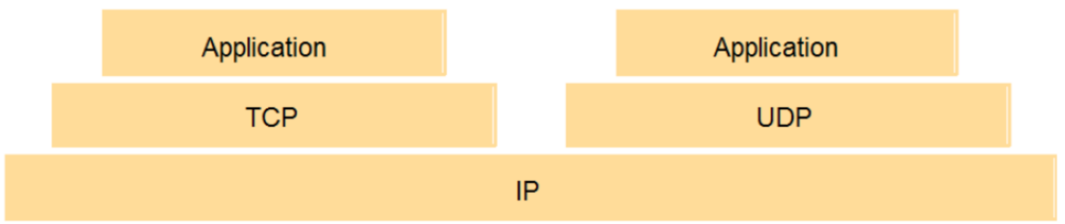
IPv6 Adoption

We are continuously measuring the availability of IPv6 connectivity among Google users. The graph shows the percentage of users that access Google over IPv6.



Internet Protocol Stack

- Layered approach that allows developers to not concern themselves with lower level details
- The internet protocol stack is abstracted to programmers



Application Layer

- Application layer provides a platform independent way to exchange data independent of the internal representation of the data
- Application layer protocols include:
  - HTTP
  - FTP
  - SFTP
  - SSH
  - …

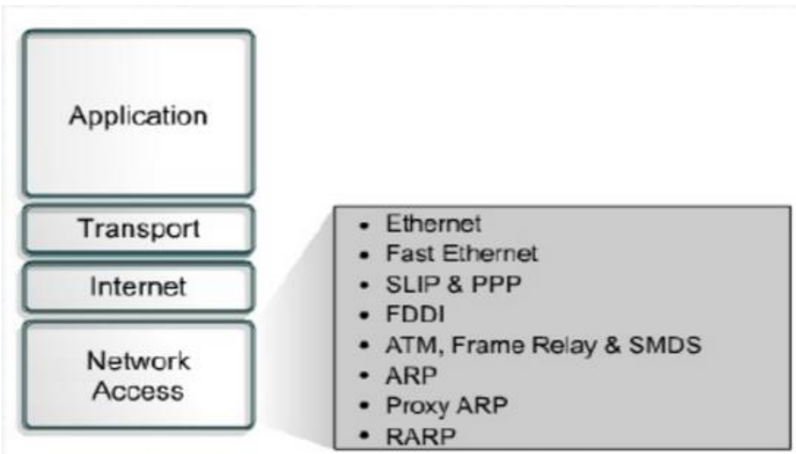
Transport Layer

- Responsible for providing transparent transport of data utilizing the services of the network layer
- Examples include:
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)
- We will talk about these in a minute

Internet Layer

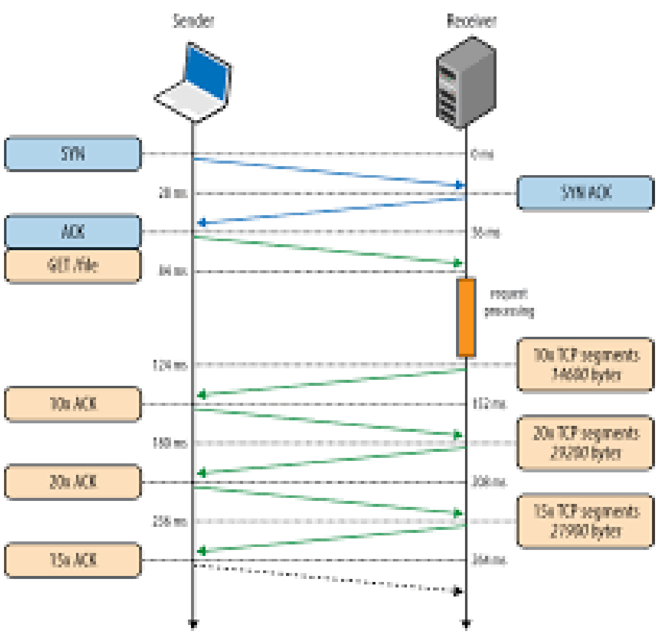
- Internet layer has the responsibility of sending the packets across the network
- This includes the IP addressing/routing that we talked about at the beginning of this lecture

Network Access Layer Lurks Below …



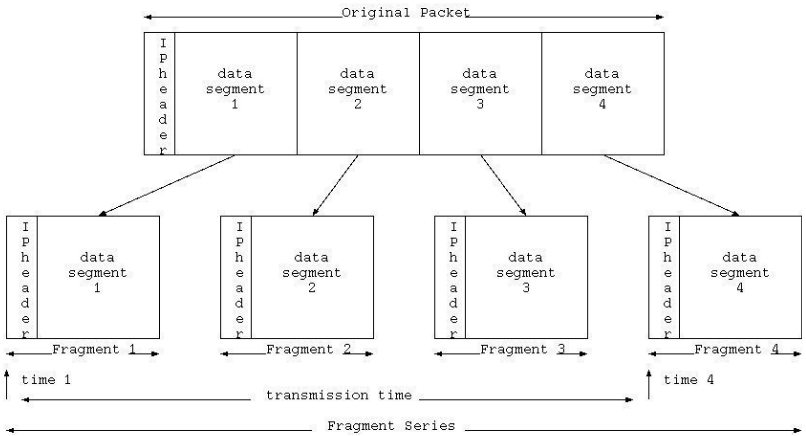
- Responsibility of abstracting the specific network hardware
- Allows transport of the data packets independent of the hardware used to realize the network

Transport Control Protocol



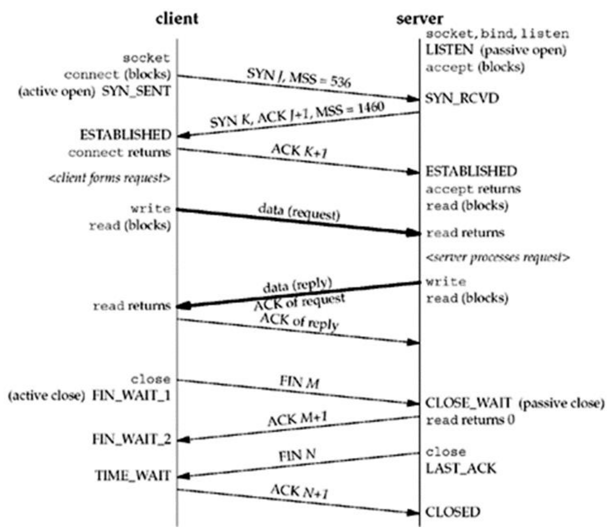
- TCP offers the application layer a way to transport data
  - Reliable
  - Stream oriented
  - Ordered
- IP breaks the data into “packets”
- These packets are routed from the source to the host
  - They are routed independently
  - The individual packets could follow a different route

What TCP Does



- TCP divides data to be transmitted into “segments”
- The segments are sent one by one to the IP layer
  - Wrapped in an IP packet and sent over the wire to the destination
- Segment includes a TCP header that includes
  - Sequence number
  - Source and destination information
  - Acknowledgement number
- TCP guarantees accurate delivery
  - But not necessarily timely delivery
  - Long delays can be experienced with TCP

TCP Communication Phases



- TCP Communication includes:
  - Connection establishment
  - Data transfer
  - Connection termination
- The TCP server sends cumulative acknowledgements
  - Receiver has received all packets < the acknowledged sequence number
- Servers can handle many incoming TCP connections
- The state of each session is stored by session id in a table

User Datagram Protocol (UDP)

- UDP is another transport layer protocol that is:
  - Stateless
  - Unreliable
  - Doesn’t guarantee ordering
  - Relatively lightweight

How UDP Works

- UDP:
  - breaks the data to be transmitted into segments
  - wraps these “datagram” segments in a UDP header containing source port, destination port, length, and checksum
  - transmits these data segments
  - No acknowledgement/retry of packets – fire and forget!!
- Programming example:
  - https://www.youtube.com/watch?v=Emuw71lozdA

Comparison

- UDP is much lighter weight and faster than TCP
- It's good for applications that have small amounts of data
  - Thus the overhead associated with TCP is too great
  - It's more efficient to build reliability mechanisms on top of UDP
- It's also good for data that can withstand packet loss
  - Streaming audio or video is an example of such data

Comparison

- TCP on the other hand is heavy weight
  - There's significant overhead in the TCP segment as well as the protocol itself
- It's reliable, however
  - It will deliver data intact even if there is packet loss
    - Resends unacknowledged messages
  - It will handle ordering
  - It will throttle the rate according the capacity
- This does result in potentially long lags, however
- Most of the internet traffic uses TCP

Lab 3 Exercise

Let’s play with Sockets!!

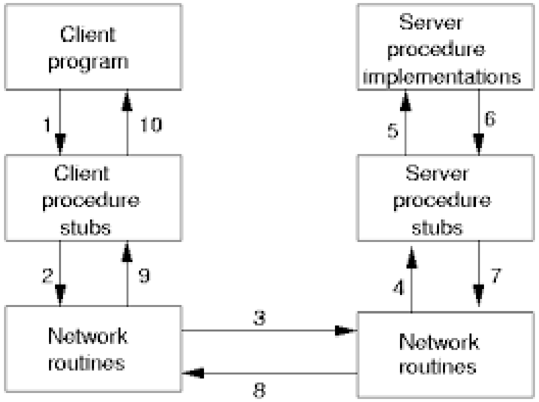
Remote Procedure Call Systems

Sockets

- Primitive mechanism
  - Fast …
- Application code responsible for:
  - Formatting
  - Client-server protocol
  - Error handling
- Lots of gotchas
  - Basically error prone


CP
UDP
TCP
UDP
TCP
UDP

Remote Procedure Call



- RPC allows a process executing on one machine to call a process on a remote machine
  - Built on top of TCP/IP
- Realized by many different technologies
  - Not always compatible
- Examples of RPC
  - CORBA
    - Old …
  - Java RMI
  - DCE
    - Really old!!

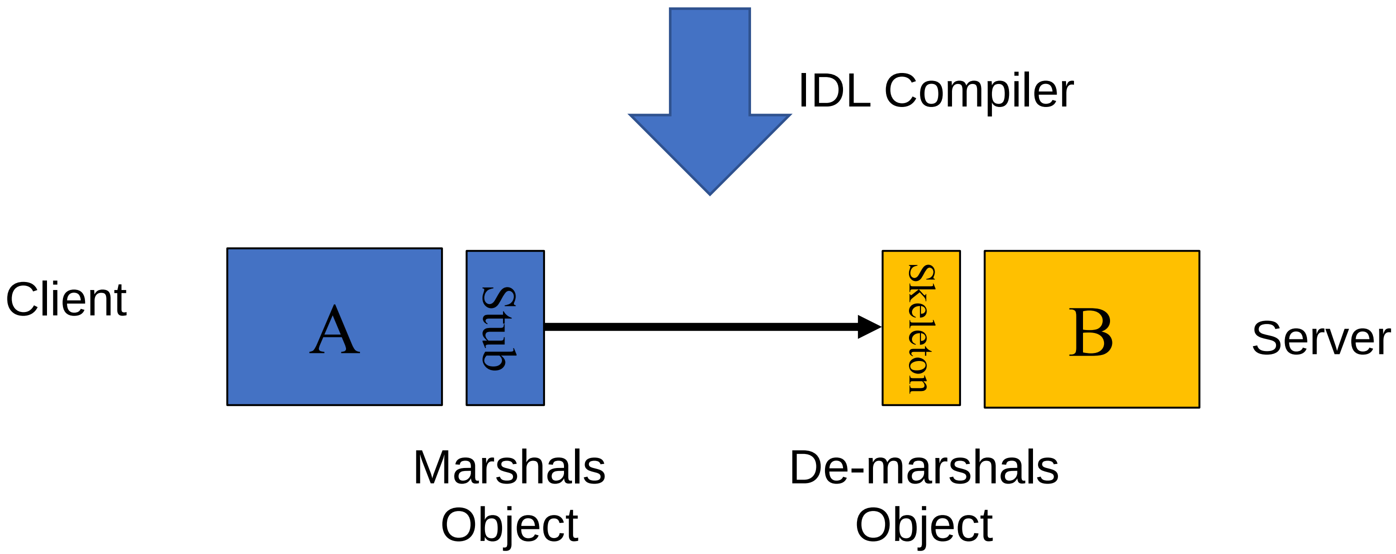
A historical perspective ….
http://ieeexplore.ieee.org/document/4623232/?reload=true&arnumber=4623232&tag=1

CORBA

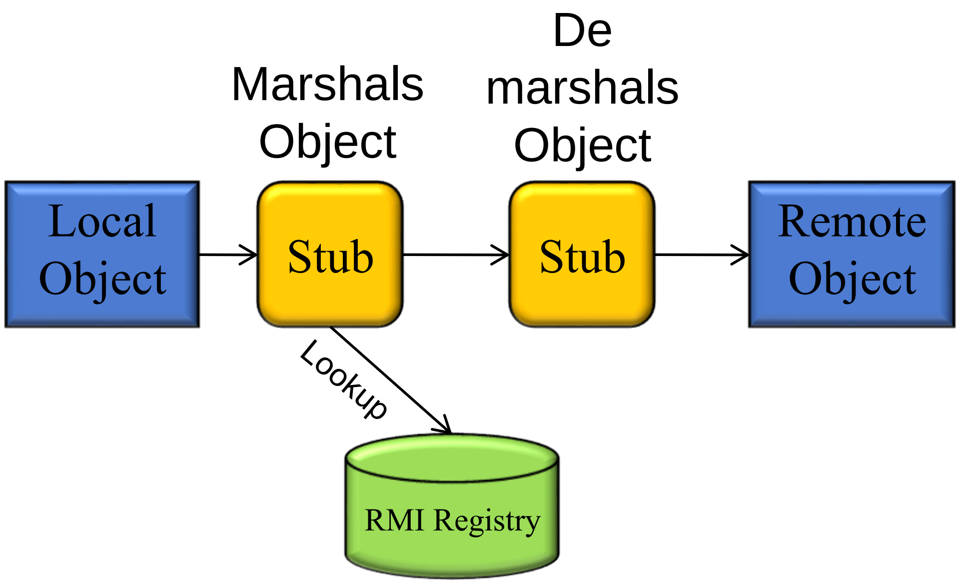
- Common Object Request Broker Architecture
- Standard defined by the Object Management Group
- Designed to support distributed communications
  - Platform independent
- Interface Definition Language specifies the “object”
  - Mapping defines the translation from the object to the end point implementation

CORBA Interaction

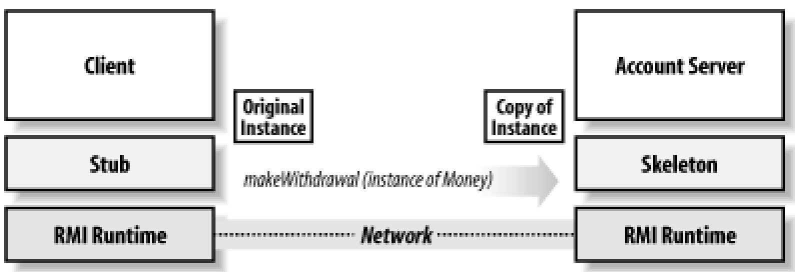
```
// CORBA IDL
Interface myServer{
    // remote method signatures
}
```



Java Remote Method Invocation

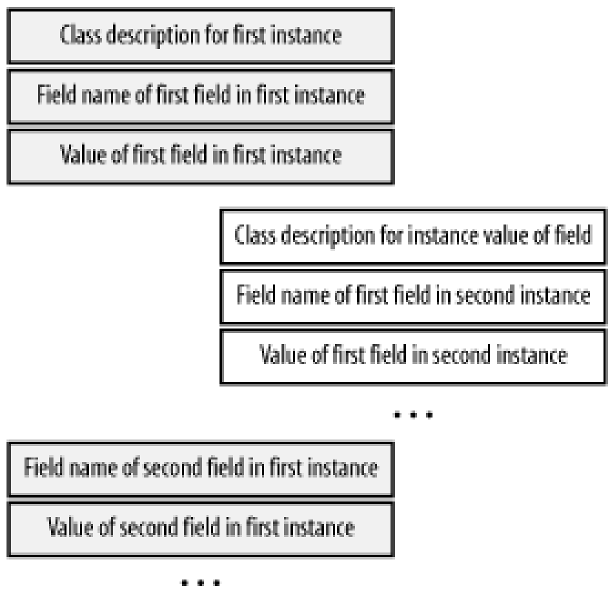


Serialization



- Objects are converted into bytes streams by:
  - Writing the metadata of the class associated with the instance
  - It recursively writes the description of the superclass (to java.lang.object)
  - Then it writes the data
- An object with 2 bytes of data could become more than 50 bytes …

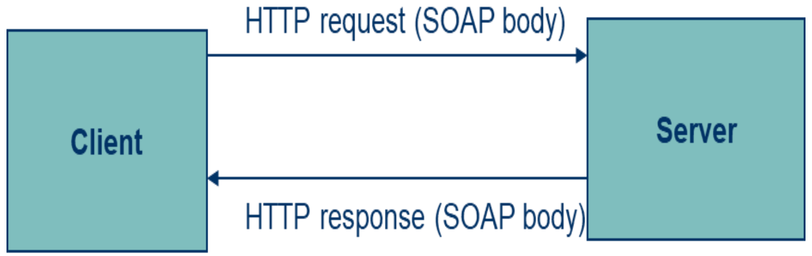
Implications or RPC



- Overhead is added
  - In terms of data being transported
  - As well as the effort to marshal and de-marshal
- We still have syntactic and semantic dependencies
- Location dependencies

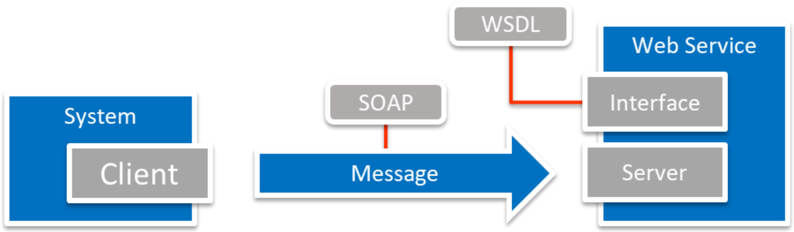

XML
WEB
SERVICES

Web Services Basics



- Web Services provide communications between two applications on the Internet
- Like RPC using HTTP/XML … sort of … as we'll see …

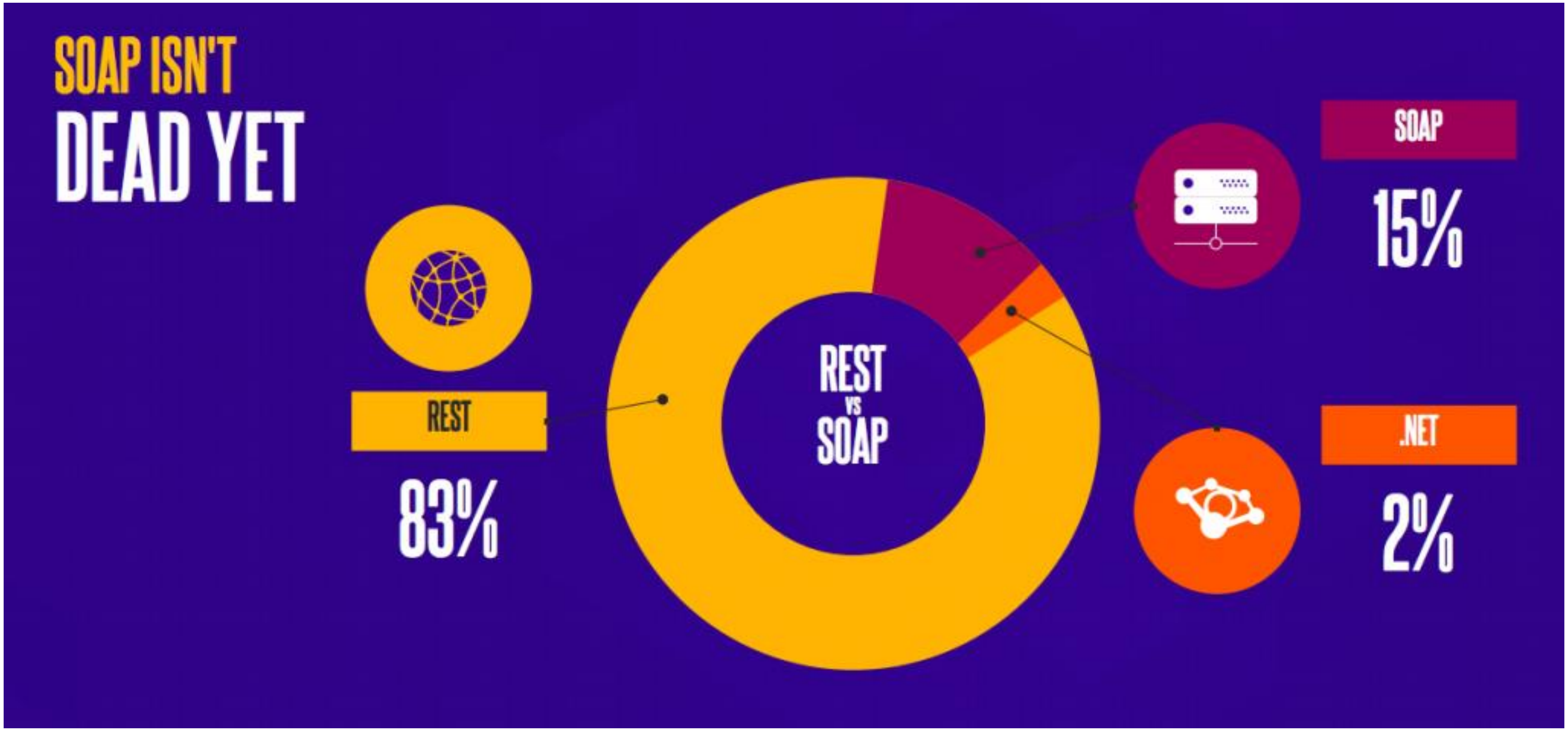
SOAP-based Web Services



- Standard web services technologies:
  - Communication via SOAP XML documents over HTTP
  - Operations of web service defined by Web Services Definition Language (WSDL) XML vocabulary
  - Data within WSDL defined using XML Schema
- Text-based protocol
  - Simple - designed for interoperability
  - Verbose

Which one?

https://jaxenter.com/state-of-api-integration-report-136342.html

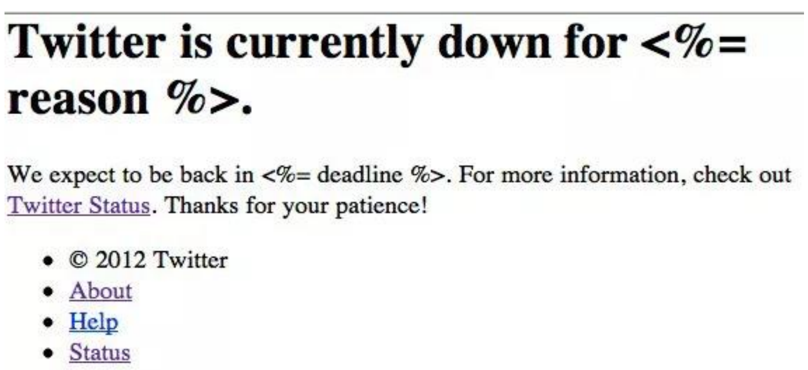


Partial Failures

When things go wrong

- Distributed Systems can fail in lots of weird and wonderful ways
- If it can fail, it will.
  - Especially at scale
- We need to build systems that:
  - Can detect failures
  - Meet business expectations in face of failures

Example Faults



- Disk crash
- Machine crash
- Network failure
- Power failures
- Overheating due to HVAC failure

Partial Failures

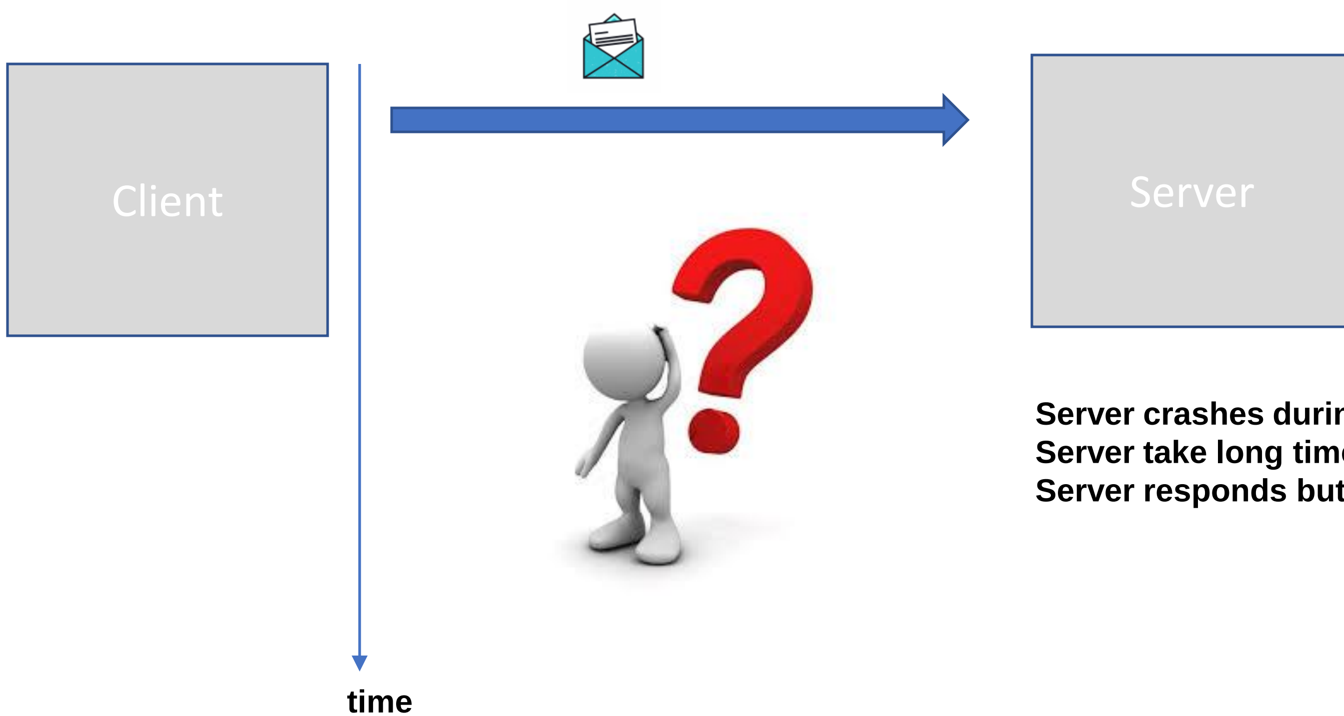
- Some nodes working fine, some failed
  - Eg power failure
- Highly available systems continue to operate with partial failures
  - Little/no customer visibility
  - Include mechanisms to recover from failures and replace/repair broken components
- Different to fault tolerance:
  - Triple redundant hardware components
  - Voting to detect partial failures
  - Expensive!!
  - Eg HP Integrity Non Stop

Build Reliable System from(many) Unreliable Components

Partial Failures

- Client sends a request to the server
  - What happens next?
- Request succeeds – all good
- IP address not found – error response
- IP address found, but server process has failed – error as server not listening
- Server receives request and fails while processing it
- Server is heavily loaded and processes request but takes a long time to respond
- Server processes request but response not delivered due to network failure

Partial Failures

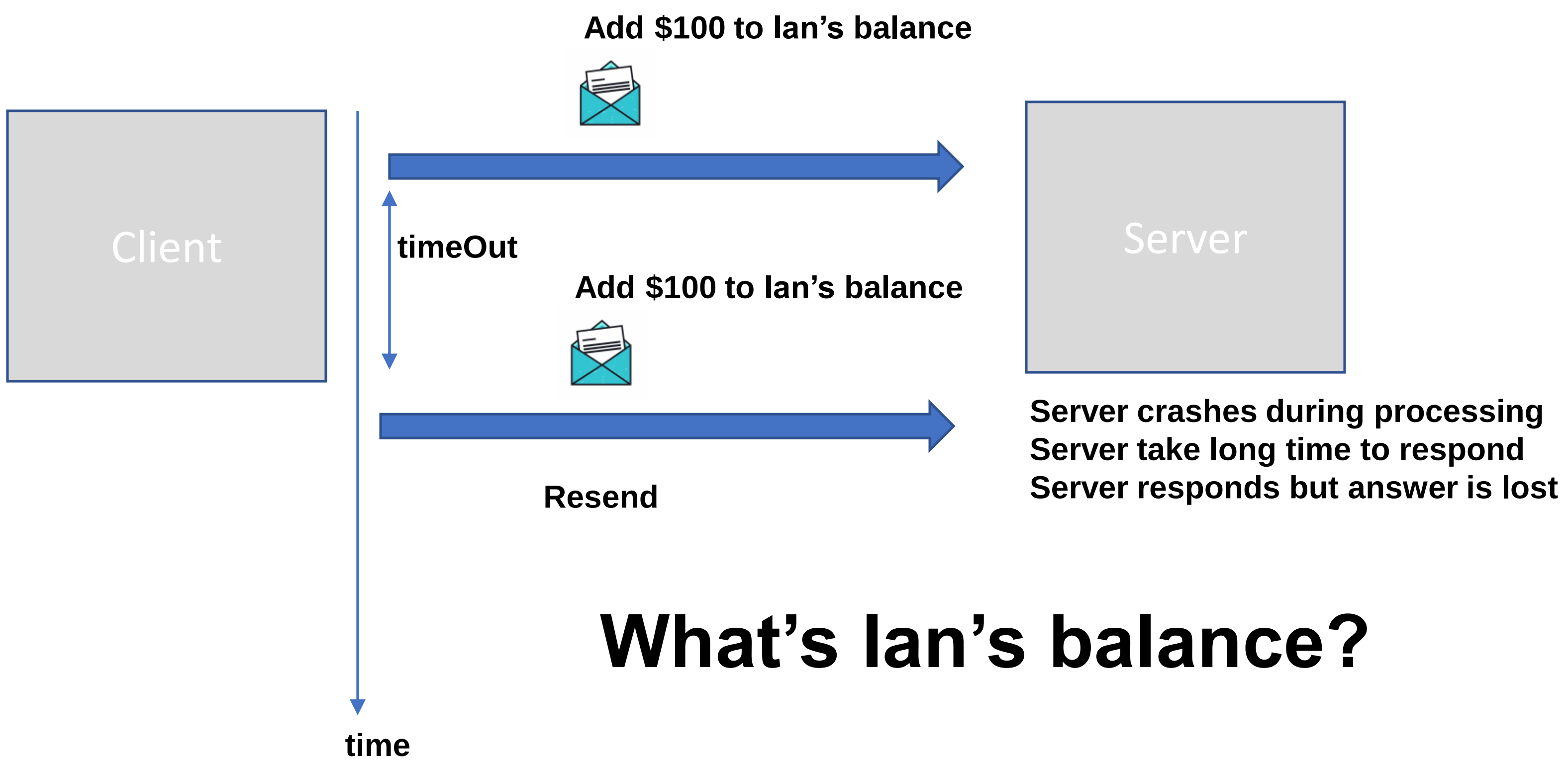


Server crashes during processing
Server take long time to respond
Server responds but answer is lost

Partial Failures

- Client waits a ‘long time’
  - Answer doesn’t arrive
  - Slow response time, long latencies before possible failure and retry
- Client waits a ‘short time’
  - Resends request
  - Faster response time,
  - What if server is just being slow?
- In either case, how does the client know if a request has been processed

Example



What's Ian's balance?

Communications Delivery Guarantees

unreliable

at-most-once delivery
means that for each message sent, that message is delivered zero or one times. ie messages may be lost.

Fire and forget

at-least-once delivery
means that for each message sent, potentially multiple attempts are made at delivering it, such that at least one succeeds; ie messages may be duplicated but not lost.

Retransmission
Acknowledgement

exactly-once delivery
means that for each message sent, exactly one delivery is made to the recipient; the message can neither be lost nor duplicated.

Retransmission
Acknowledgement
Filter out duplicates

Reliable **slow**

Idempotent Requests

- Client can send same request multiple times while producing same result
  - Ian’s balance = 100
  - Send ‘add 100 to Ian’s balance’
  - Resend due to timeout
  - Result is ALWAYS 200
- Needs a mechanism for the server to recognize duplicate requests
  - How?
  - Necessary for correct system operations

Implications for building systems

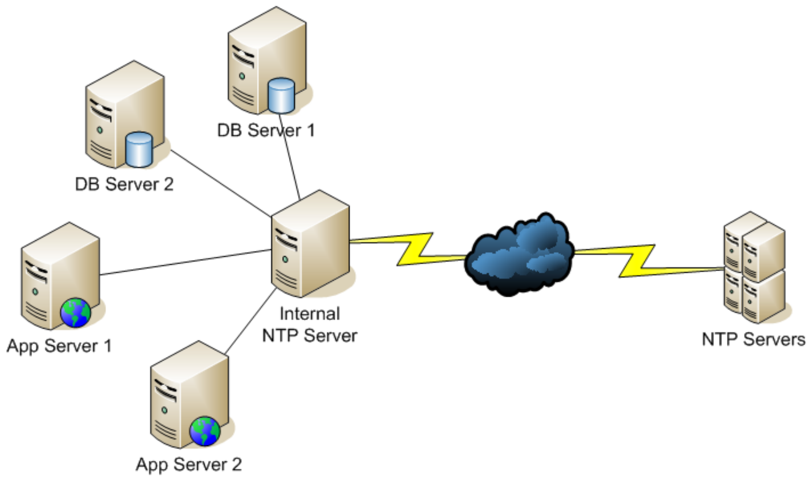
- Using REST, application is responsible for handling message failures and retries/idempotence
- More abstract communications mechanisms may offer certain guarantees, ie:
  - Java Messaging service can guarantee at most once delivery (as well as lower level guarantees)
  - Much more on this later in course …

Time in distributed Systems

Time

- Each node is a system has at least two local system clocks
- Time of day clock, e.g. in Java
    - Number of milliseconds since midnight January 1970
    - Can be synchronized using an NTP server,
    - May move backwards if too far ahead of NTP
- Monotonic Clock, e.g. in Java
    - System.nanoTime()
    - Always moves forward
    - Actual value meaningless
    - Multicore machines cause issues which OS tries to accommodate for threads scheduled on different cores

Network Time Protocol



- Synchronize ToD clocks to within a few millisecs of UTC
    - Hierarchy with highly reliable Atomic/GPS clock at core
    - Mitigates variable network latencies
    - UDP based timestamp exchanges

Accuracy

It depends …

- LAN - small number of milliseconds
- WAN – 10s of milliseconds, congestion may cause longer (eg 1 second)

Clocks also drift (run at different rates)

- Can vary by ~10-20 seconds a day

Higher accuracy possible but expensive:

- GPS clocks
- Precision Time Service (PTP)
- Google TrueTime (proprietary)
- Amazon Time Sync

Time issues

- NTP synks may cause a clock to jump forward or backwards
- Failure to synk with NTP server (eg firewall issue) may create major drift
- Shared virtualized code on a VM is paused while not running and hence sees jumps in clock

Bottom Line

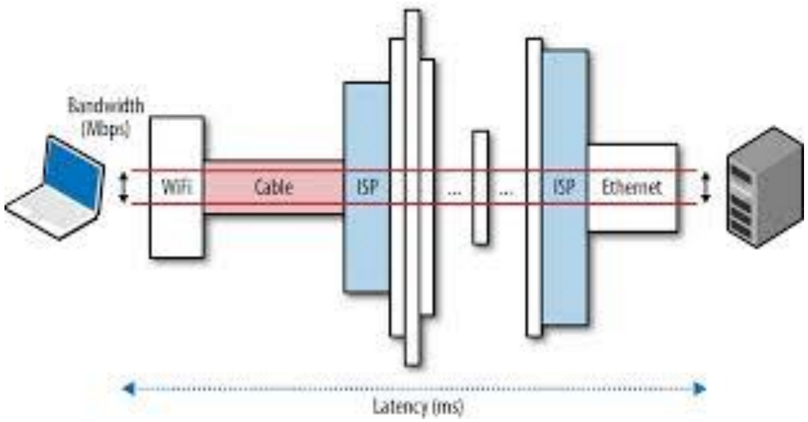
- We can’t rely on a precise central time source in our systems
  - Clock skew, etc
- We can’t rely on central source of knowledge/state being always available
  - Partial failures, etc
- Next week we’ll learn about algorithms that address these issues
  - Their strengths
  - Their weaknesses and limits

Performance and Scalability Revisited

Performance Revisited

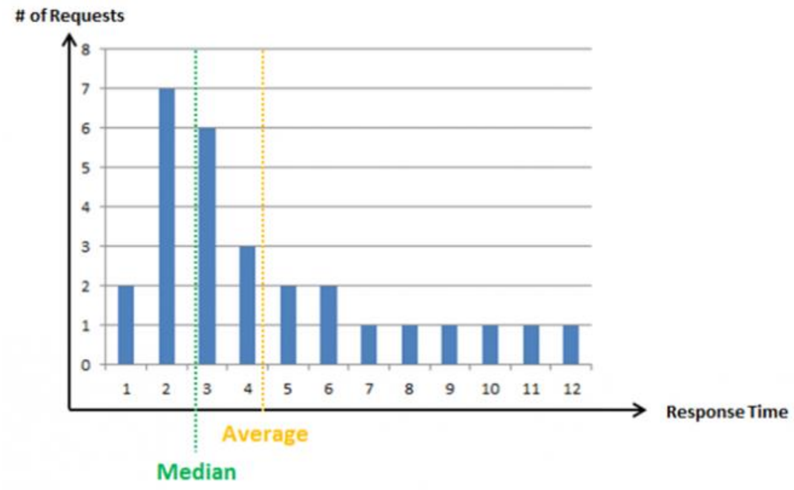
- Performance can mean different things depending on the context
- Typically people mean one or more of the following:
  - Latency
  - Predictability
  - Utilization
  - Throughput

Latency



- Latency is the time that elapses between a stimulus and a response
- the time that it takes an element to react to a given input

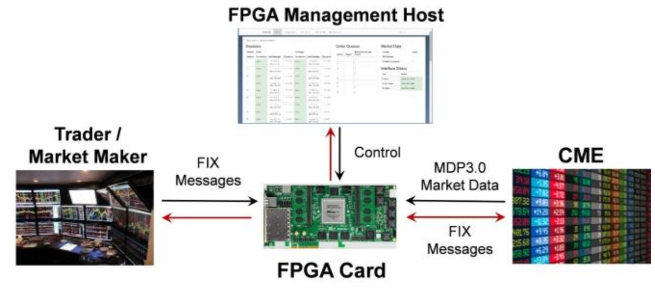
Worst Case vs Average Case



- Major difference conceptually between average case and worst case
  - Big difference in the mechanisms needed to achieve these
- In ‘enterprise scale’ systems, typically worried about average case
- In massively scalable Internet systems, average is not adequate.

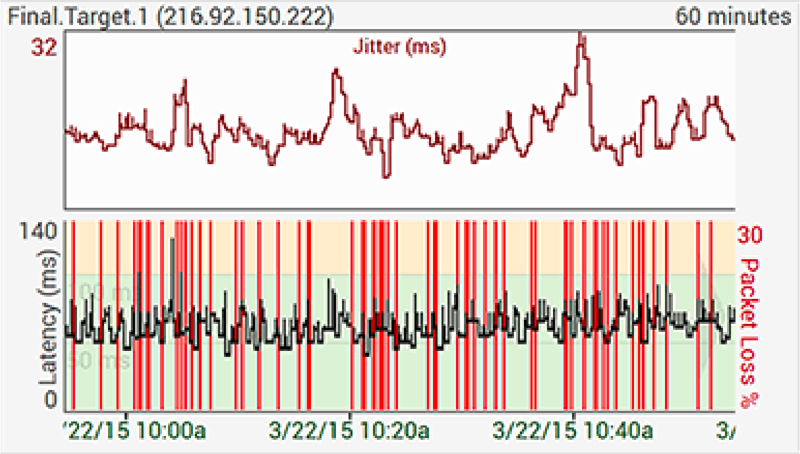
Hard vs. Soft Real Time

- Hard real time systems have deadlines for events that must be met
  - If they are not, bad things can happen
- Examples:
  - Driverless car: process imagery in N ms or it is too late to apply brakes to avoid hitting someone
  - Trading systems: Enact sales within N ms or prices drop and you'll make less money
- Hard real time systems are a field of study in their own right
  - Specialized languages, OSs, hardware
  - Not a topic in this course



Sub-microsecond (μs) latency trading system

Predictability

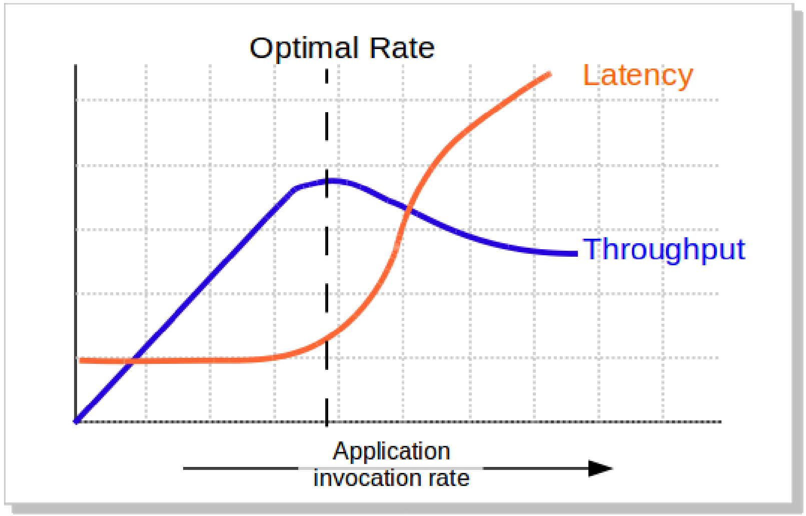


- Predictability is the variation between executions
  - This is sometimes called "jitter"
- In scalable systems, predictability gives a consistent user experience
  - Watching Netflix
  - Google Search
  - Massive online games
- Typically viewed as latency against time

Resource Utilization

- Predictable performance requires efficient resource utilization
  - CPU, Network, Disk, Memory
- If resources run low or become exhausted
  - Bad things happen …
  - Examples?
- Optimizing resource utilization is crucial for performance (and scalability)
  - CPU optimization?
  - Network optimization?
  - Disk optimization
  - Memory optimization?

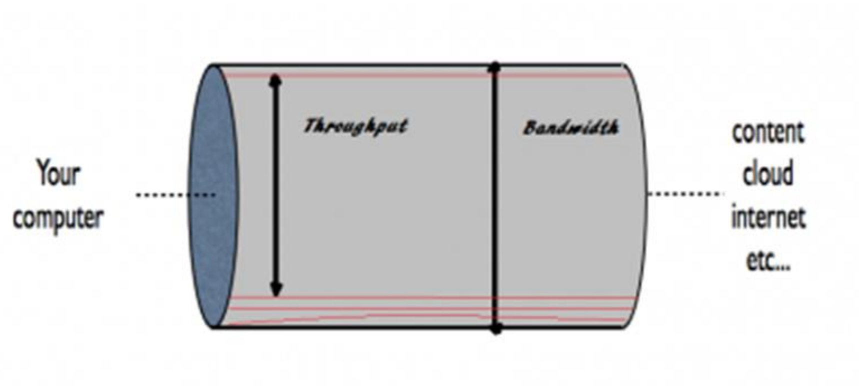
Throughput



- Throughput is concerned with the average rate of task execution per unit time
  - Eg 1000 requests/sec
  - Maximum throughput of 250 credit card approvals each minute
- Measures aggregate behavior under various loads
- This has to do with scalability

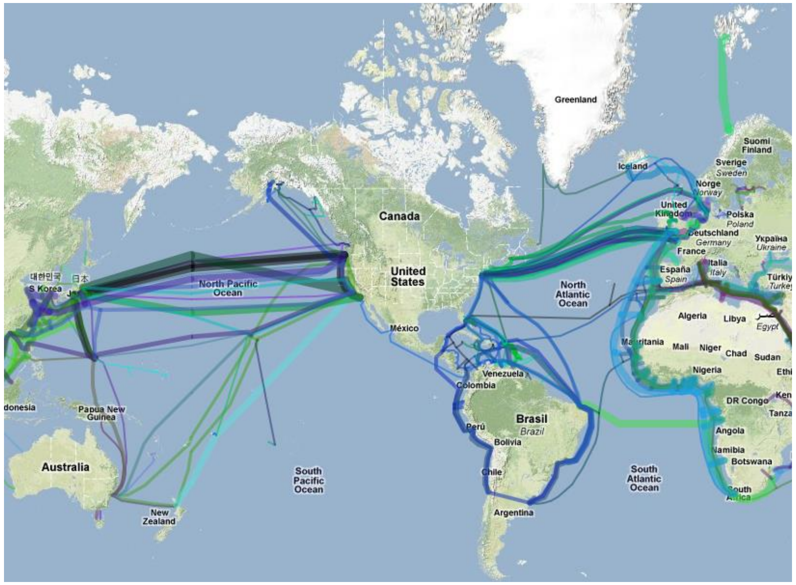
Throughput and Bandwidth

http://www.stepbystep.com/difference-between-throughput-and-bandwidth-102349/



- Bandwidth is maximum capacity
- Throughput is actual capacity under various loads

Throughput and Bandwidth



- If bandwidth not sufficient to support required throughput, what can we do?
- Increasing capacity allows us to scale systems

How do we scale this system?

Scalability

- Scalability is the ability of a system to support a growing amount of work.
  - additional users
  - additional requests from current users
  - increased volume of data
- Or a decreasing amount of work
  - Not many people watch Netflix at 3am compared to 3pm
  - After Xmas sales on Amazon decrease significantly
- Why is scaling down important?

Scalability - Analogy

Scalability - Throughput

Scalability – reduced throughput

Scalability – increased capacity

- Scaling software systems follows similar principles
- Uses very different mechanisms

Performance and Scalability

- Designing for performance is **about delivering more work using less resources** eg
  - Caching results of calculations
  - More efficient algorithms (eg O(n log n) instead of O(n2))
- Designing for scalability **is about the ability to exploit more resources**
  - Parallelize the solution to take advantage of additional resources
- Sometimes increase amount of work to process a task so we can do parts in parallel
  - Design trade-offs ….

Class Exercise

- Experiment with your multithreaded socket client-server and see what is the best throughput you can achieve?
- For example:
  - Start 100 threads, measure time/throughput
  - Start 500 threads, measure time/throughput
  - Start client and server on different machines
  - Experiment and see what throughput you can achieve

Summary

- An understanding of communications and protolcols is fundamental to distributed systems.
- Higher level RPC-style abstractions build on transport layer protocols to provide ease of development
- Partial failures need to be considered when designing communications mechanisms
- There is no global clock in distributed systems. This makes timing and ordering difficult
- Systems scale by increasing capacity