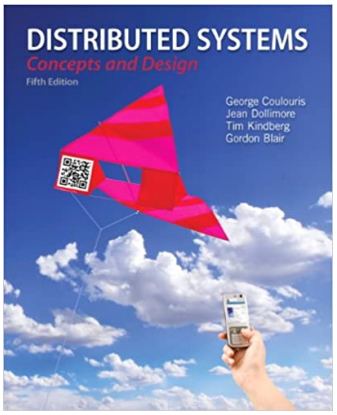
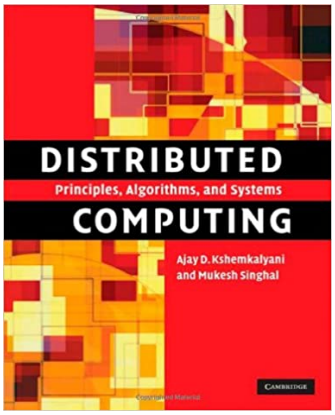
An introduction

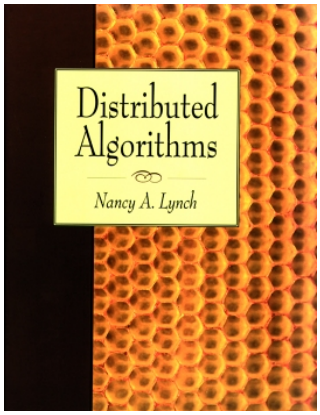
Course material

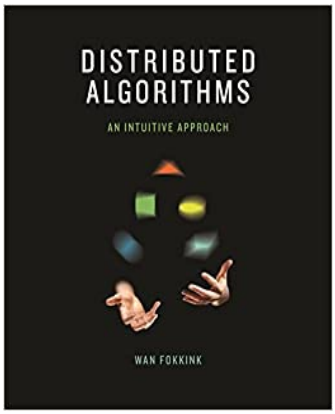
Some textbooks:

- **Distributed Systems: Concepts and Design** by Coulouris, Dollimore, Kindberg, Blair
- **Distributed Computing: Principles, Algorithms, and Systems** by Kshemkalyani and Singhal
- **Distributed Algorithms** + **Lecture notes** by Nancy Lynch
- **Distributed Algorithms: An Intuitive Approach** by Wan Fokkink









A distributed system is..

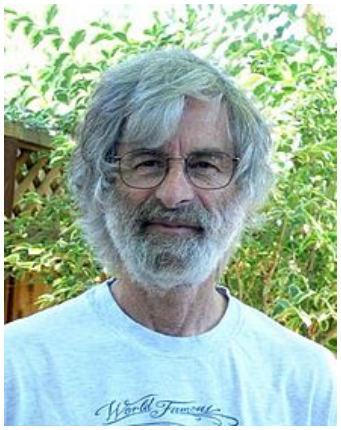
... *"a collection of independent computers that appear to the user as one computer."* — Andrew Tannenbaum

- multiple nodes communicating through a network
- trying to achieve some task together
- nodes being generic computing devices (computer, mobile phone, robot, car, ...)

A distributed system is..

... "a system in which the failure of a computer you didn't even know existed can render your own computer unusable." — Leslie Lamport

- multiple nodes communicating through a network
- trying to achieve some task together
- nodes being generic computing devices (computer, mobile phone, robot, car, ...)



Why make a system distributed?

1. *It's inherently distributed:*
   e.g. sending a message from your phone to your friend's phone

2. *For better reliability:*
   even if one node fails, the system as a whole keeps functioning

3. *For better performance:*
   get data from a nearby node, rather than from other side of world

4. *To solve bigger problems:*
   e.g. huge amounts of data can't fit on one machine

Challenges & troubles

Heterogeneity

Opennnes

Transparency

Concurrency

Consistency

Scalability

Security

Availability

Fault tolerance

Heterogeneity

There's a variety of

- hardware (e.g. in IoT)
- operating systems
- virtualization environments (Microsoft Hyper-V, OracleVM, docker ecosystem, ...)
- programming languages (Java, javascript, erlang, ...)
- mobile code
- networks
- data representation
- middleware (CORBA, XML/RPC, SOAP, Java RMI, ...)

It is very hard to guarantee correct interactions between all these components.

Openness

Systems can be extended and their components gets re-implemented in various ways.

This requires:

- uniform communication mechanisms
- published interfaces
- open standards
- documentation

Transparency

The system should be perceived as a whole, which requires *concealment of separation of components* (this indeed distinguishes a distributed system from a networked system).
The more transparencies are implemented, the more "distributed" is the system!

ISO Reference Model of Open Distributed Processing has defined several forms of transparencies, e.g.:

Transparency	*Main concern*	*Result*
Access	Methods used to access and manipulate resources/services.	Clients use same operations to access local and remote objects.
Concurrency	Concurrent access to resouces/services, race conditions.	Clients are masked from effects of concurrency (e.g. linearizability).
Location	Location of resouces/services in the distributed system.	Clients are unaware of the physical location of objects (e.g. IP).
Migration	Dynamic relocation of resources/services.	Clients are unaware of the dynamic relocation of resource/seervice.
Replication	Access to replicated objects, multiple responses, consistency.	Clients access replicated objects as if they were a single object. Distribution of requests, robustness to failures.
Resource/liveness	Resource management (e.g. deactivation/reactivation).	Clients are unaware of the deactivation/reactivation of a resource.
Failure	Partial failure of resource/service in a node.	Clients are unaware of a failure in a node.
Federation	Pan-organizational boundaries.	Clients are unaware of administrative and technology boundaries.

Concurrency

Services and shared resources should be accessed in a concurrent way.

This poses problems w.r.t.

- **ordering of events**
  when can we say that an event has happened before another one?
- **concept of global state**
  there is no such concept in the presence of concurrent non-commutative events
- **consistency of local states**
  e.g. use of distributed commit protocols, conflict resolution, ...
- **mutual exclusion**
  e.g. locking mechanisms required
- **reaching consensus**
  e.g. use of quorums and gossiping algorithms, broadcasts, ...

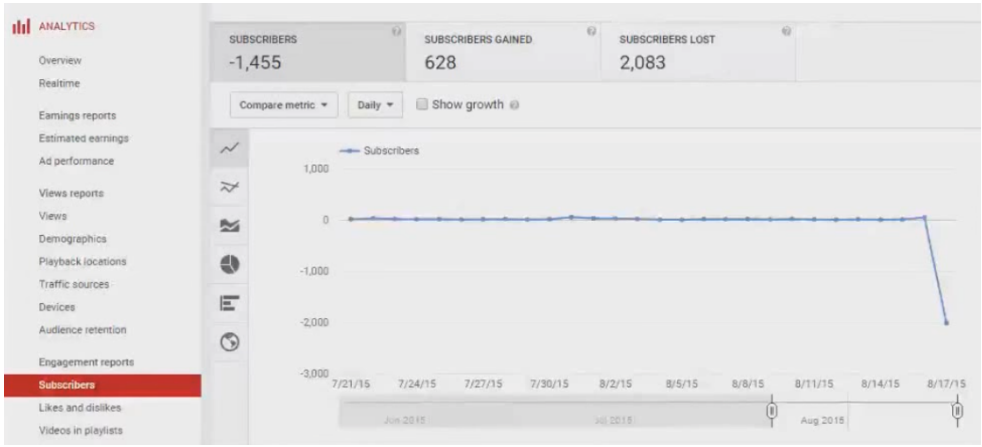
Consistency

Nodes in the systems should have a *consistent view* of the data.

In this respect challenges are posed by
- data being **distributed** or **replicated** over several nodes
- **concurrent operations** on data (e.g. accesses and updates)
- **network failures** / **inefficiencies** (e.g. retry).

Several *models of consistency*, e.g.
- serializability
- read-after-write consistency
- eventual consistency
- ...

Several *techniques*, e.g.
- idempotency & commutativity
- quorums
- locking mechanisms
- optimistic transactions
- ...



Scalability

As the number N of resources (e.g. files) becomes larger and larger, one requires:

- **scalability of performance**
  e.g. time to access/update/process/add/remove one resource should be *sublinear* in N (e.g. O(log N))
- **scalability of storage**
  e.g. size of total storage (including indexing data) should be *almost linear* in N (e.g. O(N log N))
- **scalability of costs**
  costs for development, deployment, maintenance, etc. should be *almost linear in* N

Growth of the Internet (computers vs web servers)

Date	*Computers*	*Web servers*	*Percentage*
1993, July	1 777 600	130	0.008%
1995, July	6 642 000	23 500	0.4%
1997, July	19 540 000	1 203 096	6%
1999, July	56 218 000	6 598 697	12%
2001, July	125 888 197	31 299 592	25%
2003, July	~ 200 000 000	42 298 371	21%
2005, July	353 284 187	67 571 581	19%
2014, July	996 106 380	178 558 568	18%

Scalability

As the number N of resources (e.g. files) becomes larger and larger, one requires:

- **scalability of performance**
  e.g. time to access/update/process/add/remove one resource should be *sublinear* in N (e.g. O(log N))
- **scalability of storage**
  e.g. size of total storage (including indexing data) should be *almost linear* in N (e.g. O(N log N))
- **scalability of costs**
  costs for development, deployment, maintenance, etc. should be *almost linear in* N

Growth of the Internet (computers vs web servers)

Date	*Computers*	*Web servers*	*Percentage*
1993, July	1 777 600	130	0.0007%
1995, July	6 642 000	23 500	0.4%
1997, July	19 540 000	1 203 096	6%
1999, July	56 218 000	6 598 697	12%
2001, July	125 888 197	31 299 592	25%
2003, July	> 200 000 000	42 298 371	21%
2005, July	353 284 187	67 571 581	19%
2014, July	996 106 380	179 558 568	18%

Question:

Consider two different network topologies, e.g. a *ring* and a *mesh*.
Which one is better in terms of scalability of performance / storage?

Security

The system is likely to be exposed to an *adversarial environment*.

It should then guarantee, for instance:

- **confidentiality** (e.g. against sniffing attacks)
- **integrity** (e.g. against forgery of messages and data)
- **security of mobile code** (e.g. cross-origin access policy)

Availability

The system should always be accessible (e.g. tolerant against DOS attacks), and should function correctly.

Example: online shop wants to sell stuff 24/7 (downtime = money loss).

In practice, this requirement can only be approximated:

uptime = fraction of time that a service is accessible and functioning correctly

- "Two nines" = 99% up = down 3.7 days/year (e.g. batch processing, data extraction)
- "Three nines" = 99.9% up = down 8.8 hours/year (e.g. project management tools)
- "Four nines" = 99.99% up = down 53 minutes/year (e.g. video streaming, online gaming)
- "Five nines" = 99.999% up = down 5.3 minutes/year (e.g. ATM, telecommunications)

Service-Level Objective = uptime combined with latency constraints
e.g. "99.9% of requests get a response within 200ms"

Fault tolerance

We want the system as a whole to continue working *even when a fraction of its components is faulty*, e.g. *

- messages may be lost
- nodes may crash, or temporarily freeze, while others keep running
- network may be temporarily partitioned

Handling principles:

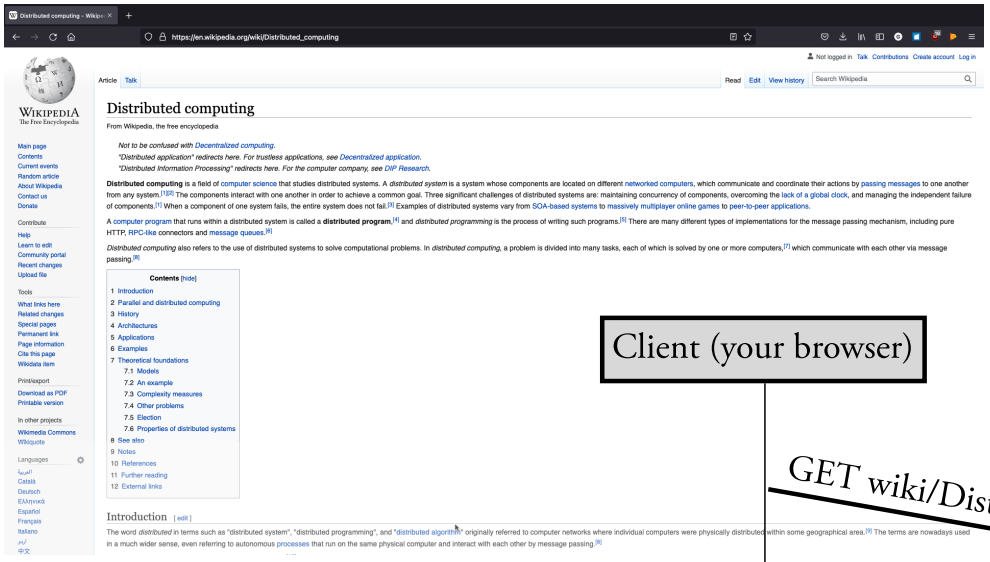
- *detecting*
  not easy, often impossible, but if detected a failure can be managed to some extent
- *masking*
  e.g. by hiding the crashed node, or retransmitting a lost message (e.g. TCP)
- *tolerating*
  e.g. by ignoring and reporting an abnormal behaviours (e.g. delivery failure in email servers)
- *recovery*
  e.g. by logs and rollbacks (e.g. DBMS)
- *redundancy*
  e.g. by keeping replicas (but this raises a consistency problem...)

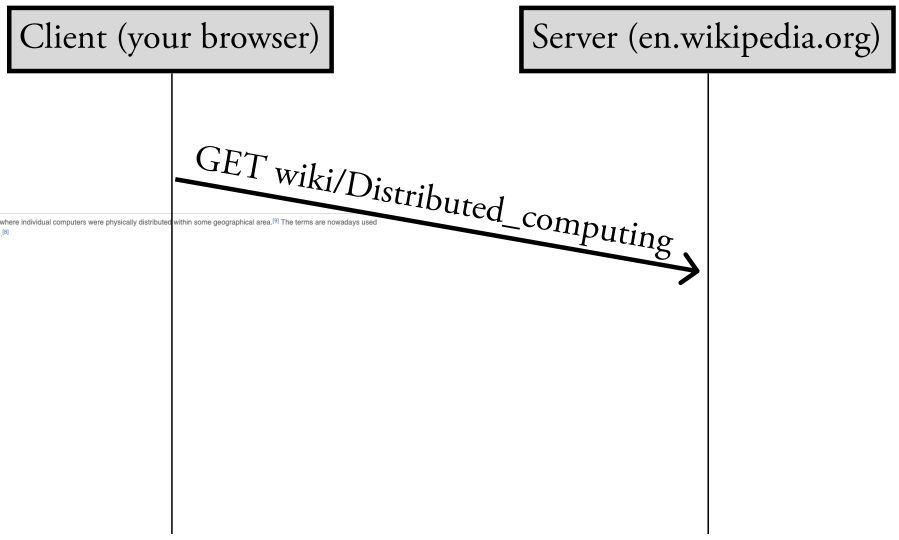
* "*All these problems will happen, simultaneously and non-deterministically*" — Murphy

Credits to Denise Yu http://deniseyu.io

Examples

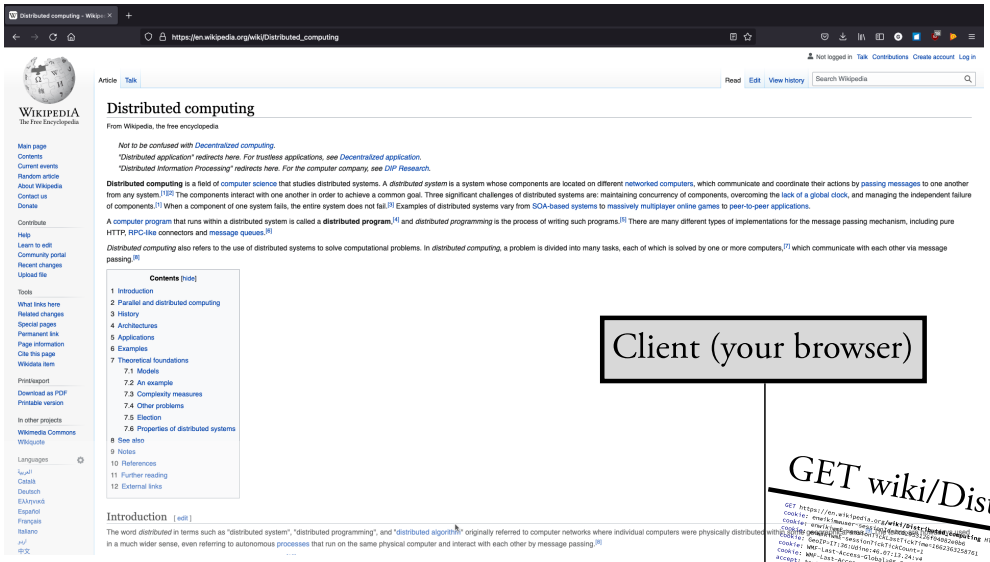
The web as a distributed system

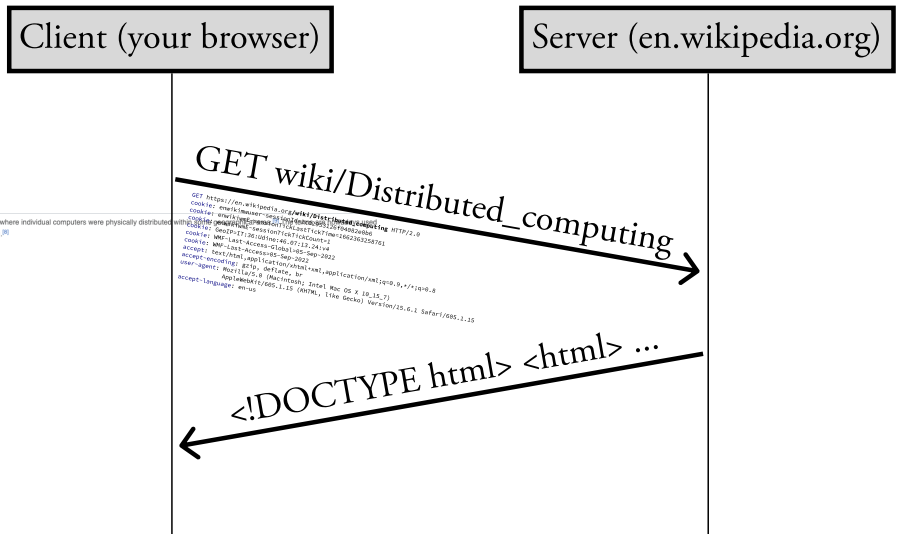


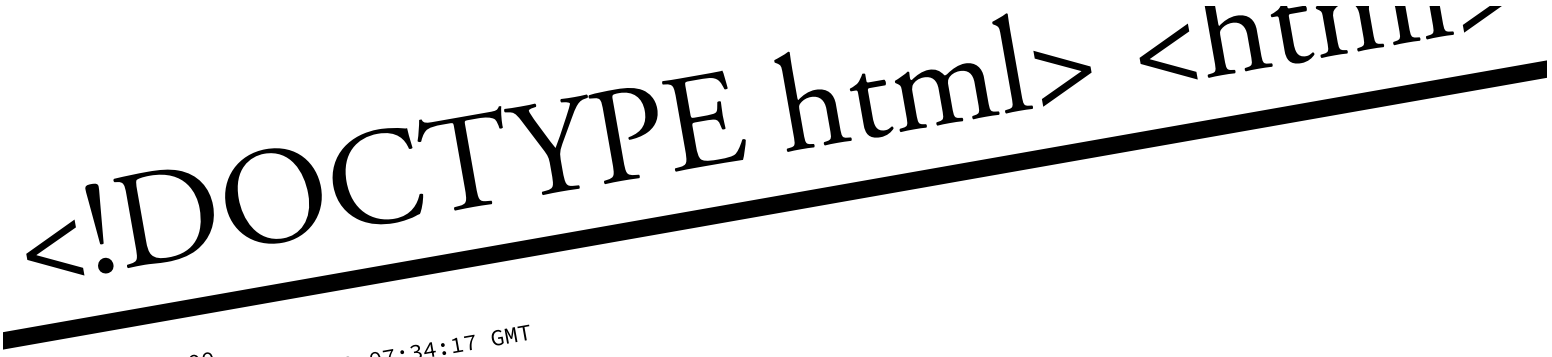


Examples

The web as a distributed system



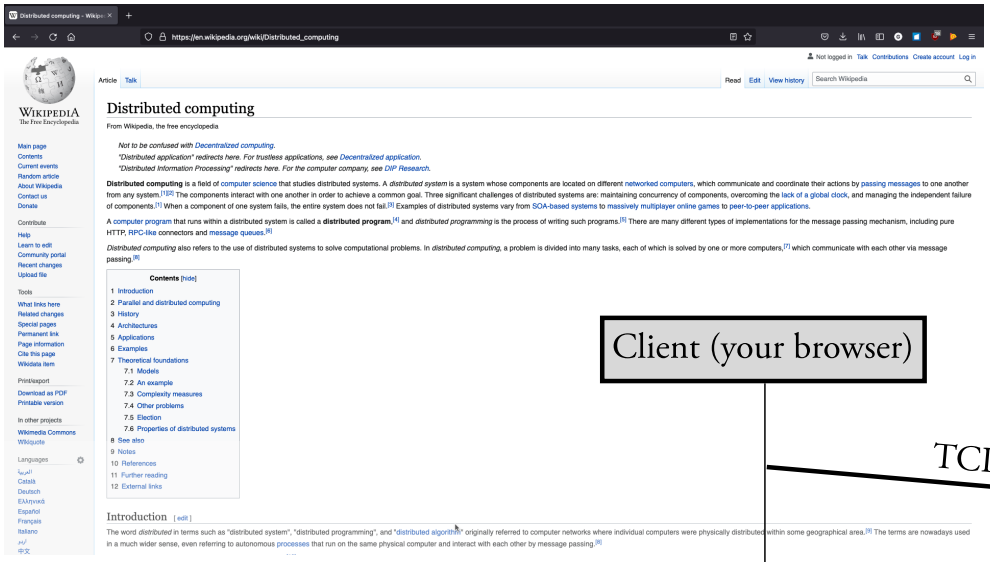


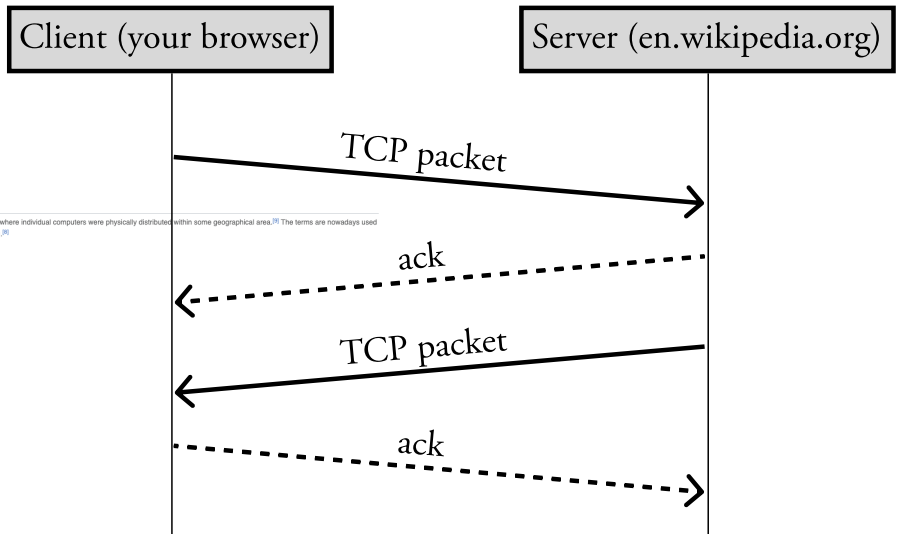


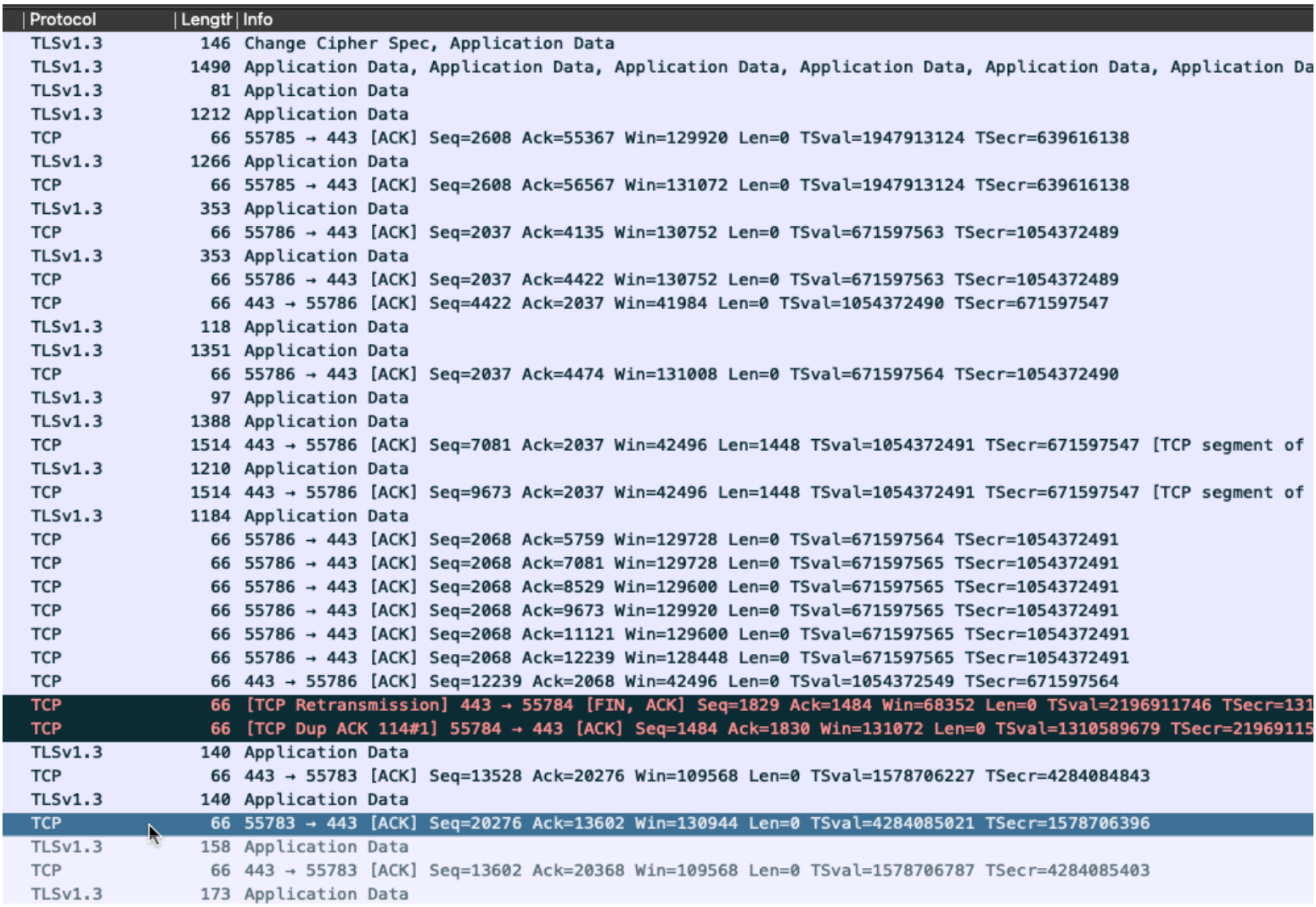

```
HTTP/2.0 200
date: Mon, 05 Sep 2022 07:34:17 GMT
server: mw1351.eqiad.wmnet
x-content-type-options: nosniff
content-language: en
vary: Accept-Encoding,Cookie,Authorization
last-modified: Sun, 04 Sep 2022 14:38:04 GMT
content-type: text/html; charset=UTF-8
content-encoding: gzip
age: 1008
x-cache: cp6014 miss, cp6015 hit/6
x-cache-status: hit-front
...
content-length: 44497

decoded gzip:

<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
  <meta charset="UTF-8"/>
  <title>Distributed computing - Wikipedia</title>
  ...
</head>
<body class="...">
  ...
</body>
</html>
```

Examples

The web as a distributed system







Protocol	Length	Info
TLSv1.3	146	Change Cipher Spec, Application Data
TLSv1.3	1490	Application Data, Application Data, Application Data, Application Data, Application Data, Application Da
TLSv1.3	81	Application Data
TLSv1.3	1212	Application Data
TCP	66	55785 → 443 [ACK] Seq=2608 Ack=55367 Win=129920 Len=0 TSval=1947913124 TSecr=639616138
TLSv1.3	1266	Application Data
TCP	66	55785 → 443 [ACK] Seq=2608 Ack=56567 Win=131072 Len=0 TSval=1947913124 TSecr=639616138
TLSv1.3	353	Application Data
TCP	66	55786 → 443 [ACK] Seq=2037 Ack=4135 Win=130752 Len=0 TSval=671597563 TSecr=1054372489
TLSv1.3	353	Application Data
TCP	66	55786 → 443 [ACK] Seq=2037 Ack=4422 Win=130752 Len=0 TSval=671597563 TSecr=1054372489
TCP	66	443 → 55786 [ACK] Seq=4422 Ack=2037 Win=41984 Len=0 TSval=1054372490 TSecr=671597547
TLSv1.3	118	Application Data
TLSv1.3	1351	Application Data
TCP	66	55786 → 443 [ACK] Seq=2037 Ack=4474 Win=131008 Len=0 TSval=671597564 TSecr=1054372490
TLSv1.3	97	Application Data
TLSv1.3	1388	Application Data
TCP	1514	443 → 55786 [ACK] Seq=7081 Ack=2037 Win=42496 Len=1448 TSval=1054372491 TSecr=671597547 [TCP segment of
TLSv1.3	1210	Application Data
TCP	1514	443 → 55786 [ACK] Seq=9673 Ack=2037 Win=42496 Len=1448 TSval=1054372491 TSecr=671597547 [TCP segment of
TLSv1.3	1184	Application Data
TCP	66	55786 → 443 [ACK] Seq=2068 Ack=5759 Win=129728 Len=0 TSval=671597564 TSecr=1054372491
TCP	66	55786 → 443 [ACK] Seq=2068 Ack=7081 Win=129728 Len=0 TSval=671597565 TSecr=1054372491
TCP	66	55786 → 443 [ACK] Seq=2068 Ack=8529 Win=129600 Len=0 TSval=671597565 TSecr=1054372491
TCP	66	55786 → 443 [ACK] Seq=2068 Ack=9673 Win=129920 Len=0 TSval=671597565 TSecr=1054372491
TCP	66	55786 → 443 [ACK] Seq=2068 Ack=11121 Win=129600 Len=0 TSval=671597565 TSecr=1054372491
TCP	66	55786 → 443 [ACK] Seq=2068 Ack=12239 Win=128448 Len=0 TSval=671597565 TSecr=1054372491
TCP	66	443 → 55786 [ACK] Seq=12239 Ack=2068 Win=42496 Len=0 TSval=1054372549 TSecr=671597564
TCP	66	[TCP Retransmission] 443 → 55784 [FIN, ACK] Seq=1829 Ack=1484 Win=68352 Len=0 TSval=2196911746 TSecr=131
TCP	66	[TCP Dup ACK 114#1] 55784 → 443 [ACK] Seq=1484 Ack=1830 Win=131072 Len=0 TSval=1310589679 TSecr=21969115
TLSv1.3	140	Application Data
TCP	66	443 → 55783 [ACK] Seq=13528 Ack=20276 Win=109568 Len=0 TSval=1578706227 TSecr=4284084843
TLSv1.3	140	Application Data
TCP	66	55783 → 443 [ACK] Seq=20276 Ack=13602 Win=130944 Len=0 TSval=4284085021 TSecr=1578706396
TLSv1.3	158	Application Data
TCP	66	443 → 55783 [ACK] Seq=13602 Ack=20368 Win=109568 Len=0 TSval=1578706787 TSecr=4284085403
TLSv1.3	173	Application Data

Other examples of distributed systems

- The information society

  WWW, search engines (Google), cloud computing (AWS), Wikipedia, social networking (e.g. Facebook, MySpace)

Google web search engine indexes the whole content of the internet:
- more than $16*10^9$ web pages

It includes:

- a physical infrastructure of *thousands of computers*, located all around the world
- a *distributed file system*, heavily optimized
- a service implementing *distributed locking, consensus*, and *conflict resolution*
- a programming model supporting *parallel and distributed computations*

(Google)

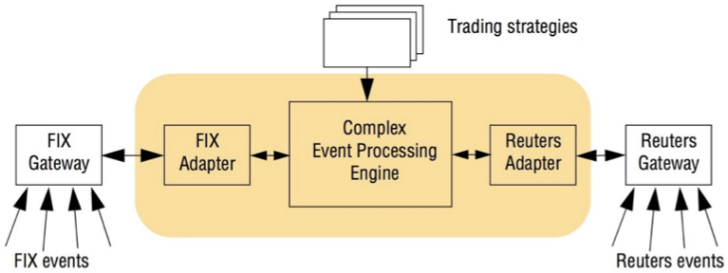
Other examples of distributed systems

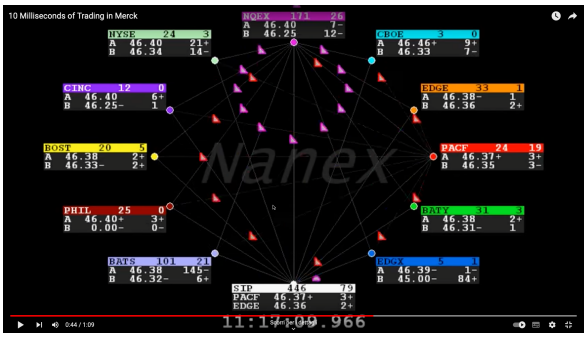
• The information society	WWW, search engines (Google), cloud computing (AWS), Wikipedia, social networking (e.g. Facebook, MySpace)
• Finance & commerce	e-commerce (e.g. Amazon, eBay), PayPal, trading, online banking

trading,

Distributed *event-based* system, programmed by Event Processing Languages

- events are exchanged in a *standard format and protocol* (Financial Information eXchange, Reuter events)
- *real-time access* to wide range of information sources (e.g. share prices, trends, economics, political developments)
- *automated monitoring* of events and trading applications





https://www.youtube.com/watch?v=L5cZaIZ5bWc

Other examples of distributed systems

• The information society	WWW, search engines (Google), cloud computing (AWS), Wikipedia, social networking (e.g. Facebook, MySpace)
• Finance & commerce	e-commerce (e.g. Amazon, eBay), PayPal, trading, online banking
• Creative industries & entertainment	online gaming, audio and video streaming (Netflix), user-generated content (e.g. YouTube, Flickr)
• Healthcare	health informatics, clinic data, monitoring (e.g. Immuni)
• Education	e-learning, virtual learning environments, distance learning
• Transport & logistics	GPS in-route finding systems, map services (e.g. Google Maps)
• Science	distributed databases (e.g. GRID - Global Research Identifier Database, ROR, ...)
• Environmental management	sensor technology to monitor earthquakes, floods or tsunamis

Generations of distributed systems

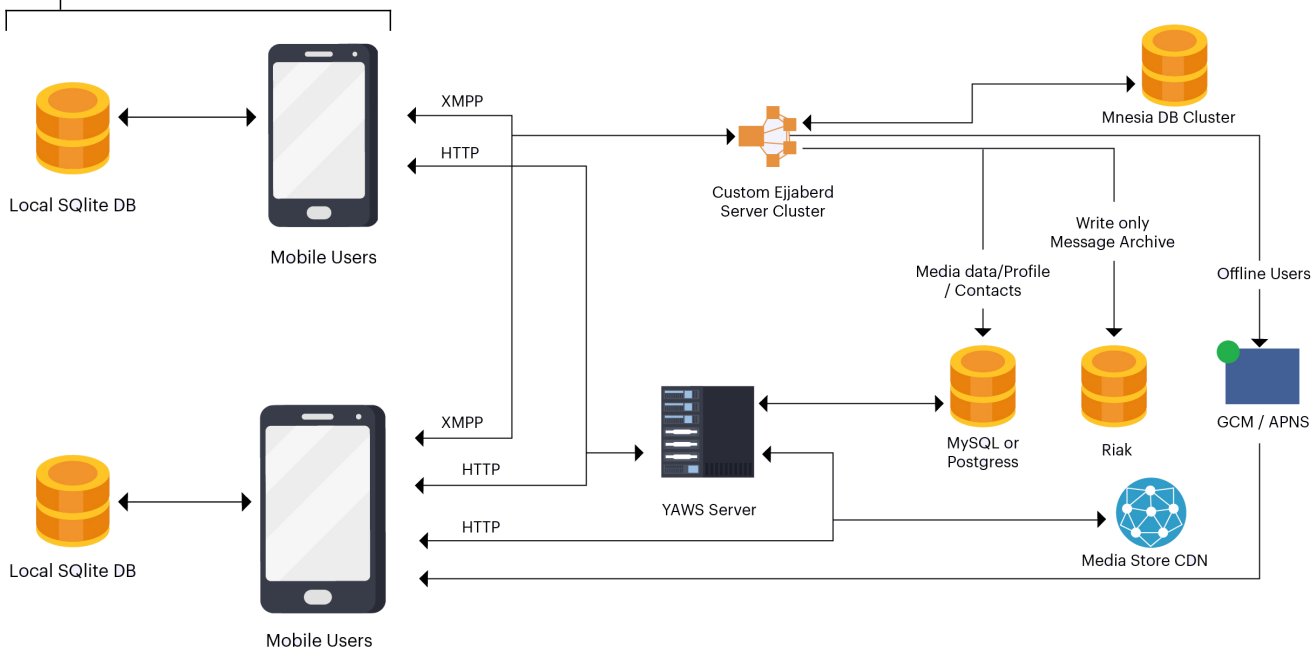
Generation	*Scale*	*Heterogeneity*	*Openness*	*Quality of service*
Early generation LANs, shared printers, file servers, emails	Small	Limited (typically homogeneous configurations)	Not a priority	In its infancy
Internet era WWW, networks of networks	Large	Significant in terms of platforms, languages, and middleware	Significant, with a range of standards introduced	Significant, with a range of services introduced
Contemporary D.S. mobiles systems, cloud computing, virtualised systems	Huge	High (radically different types of components)	Major research challenge, with existing standards not able to embrace the complexity of the systems	Major research challenge, with existing services not able to embrace the complexity of the systems

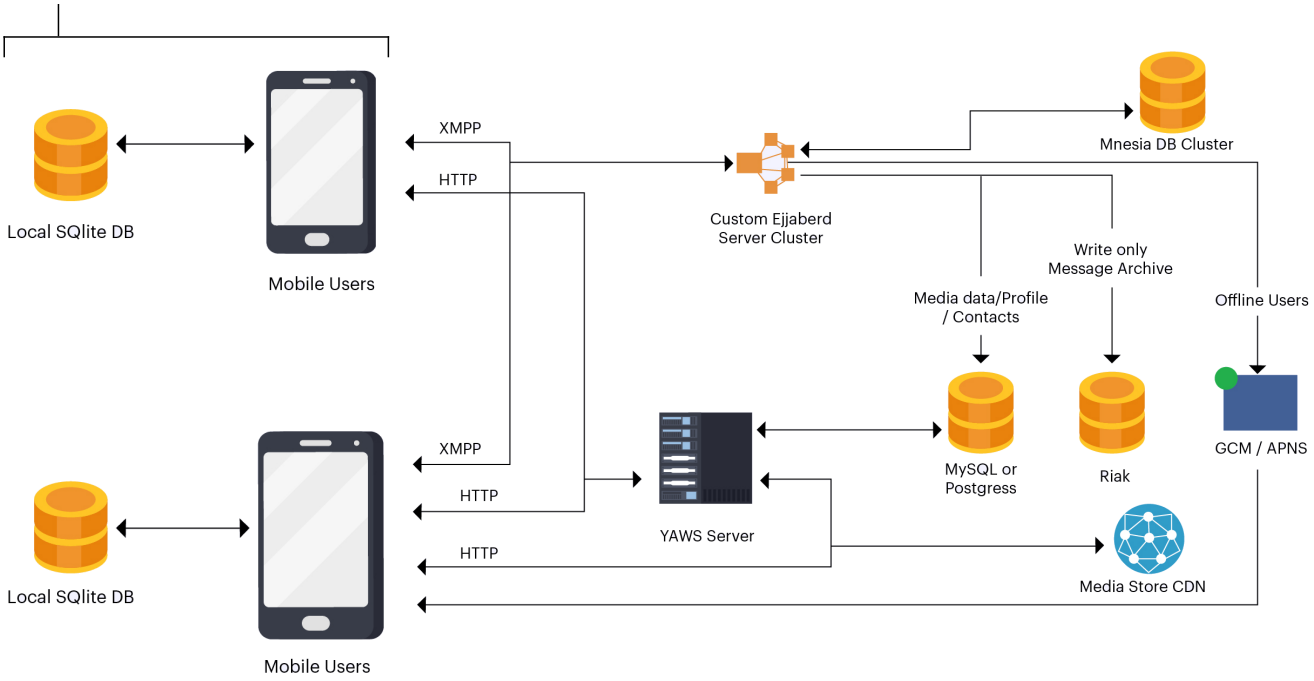
A glimpse at Whatsapp architecture

2.5 billion active users, 100 billion messages sent every day ...with just 50 engineers!

frontend:

- *mobile code* on Android (Java), iOS (Swift), Windows phones (C#)
- *web app code* with Javascript / HTML / CSS
- *desktop app code* on MacOS (Swift / Objective-C), Windows (C / C# / Java)
- SQLlite DB for storing chats





backend:

- Erlang (main programming language on servers)
- XMPP (open standard chat protocol for exchaging small XML data)
- Ejjaberd (open-source XMPP server, written in Erlang)
- Mnesia (DBMS, written in Erlang)
- Riak (distributed NoSQL key-value DBMS)
- YAWS (web server in Erlang, used for managing multimedia data)
- GCM (Google Cloud Messaging) /
  APNS (Apple Push Notification Service)

An introduction

[back]

Course material

Some textbooks:

- **Distributed Systems: Concepts and Design** by Coulouris, Dollimore, Kindberg, Blair
- **Distributed Computing: Principles, Algorithms, and Systems** by Kshemkalyani and Singhal
- **Distributed Algorithms** + **Lecture notes** by Nancy Lynch
- **Distributed Algorithms: An Intuitive Approach** by Wan Fokkink

A distributed system is...

..."a system in which the failure of a computer you didn't even know existed can render your own computer unusable." — Leslie Lamport

- *multiple* nodes communicating through a network
- trying to achieve some task together
- nodes being generic computing devices (computer, mobile phone, robot, car, ...)

Why make a system distributed?

1. *It's inherently distributed:*
   e.g. sending a message from your phone to your friend's phone
2. *For better reliability:*
   even if one node fails, the system as a whole keeps functioning
3. *For better performance:*
   get data from a nearby node, rather than from other side of world
4. *To solve bigger problems:*
   e.g. huge amounts of data can't fit on one machine

Challenges & troubles

Heterogeneity

Openness

Transparency

Concurrency

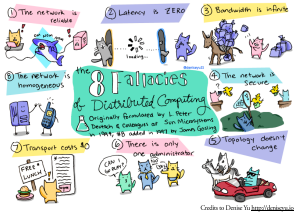
Consistency

Scalability

Security

Availability

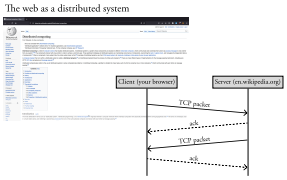
Fault tolerance



Credits to Denise Yu https://denizeyu.io

Examples

The web as a distributed system



Other examples of distributed systems

• The information society	WWW, search engines (Google), cloud computing (AWS), Wikipedia, social networking (e.g. Facebook, MySpace)
• Finance & commerce	e-commerce (e.g. Amazon, eBay), PayPal, trading, online banking
• Creative industries & entertainment	online gaming, audio and video streaming (Netflix), user-generated content (e.g. YouTube, Flickr)
• Healthcare	health informatics, clinic data, monitoring (e.g. Immuni)
• Education	e-learning, virtual learning environments, distance learning
• Transport & logistics	GPS in-route finding systems, map services (e.g. Google Maps)
• Science	distributed databases (e.g. GRID - Global Research Identifier Database, ROR, ...)
• Environmental management	sensor technology to monitor earthquakes, floods or tsunamis

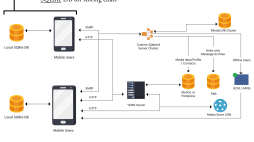
Generations of distributed systems

Generation	*Scale*	*Heterogeneity*	*Openness*	*Quality of service*
Early generation LANs, shared printers, file servers, emails	Small	Limited (typically homogeneous configurations)	Not a priority	Is in infancy
Internet era WWW, networks of networks	Large	Significant in terms of platforms, languages, and middleware	Significant, with a range of standards introduced	Significant, with a range of services introduced
Contemporary D.S. mobiles systems, cloud computing, virtualised systems	Huge	High (radically different types of components)	Major research challenge, with existing standards not able to embrace the complexity of the systems	Major research challenge, with existing services not able to embrace the complexity of the systems

A glimpse at Whatsapp architecture

2.5 billion active users, 100 billion messages sent every day ...with just 50 engineers!

frontend:
- *mobile code* on Android (Java), iOS (Swift), Windows phones (C#)
- *web app code* with Javascript / HTML / CSS
- *desktop app code* on MacOS (Swift / Objective-C), Windows (C / C# / Java)
- SQLite DB for storing chats



backend:
- Erlang (main programming language on servers)
- XMPP (open standard chat protocol for exchaging small XML data)
- Ejabberd (open-source XMPP server, written in Erlang)
- Mnesia (DBMS, written in Erlang)
- Riak (distributed NoSQL key-value DBMS)
- YAWS (web server in Erlang, used for managing multimedia data)
- GCM (Google Cloud Messaging) / APNS (Apple Push Notification Service)