Distributed Systems — Fundamental Concepts

Allan Clark

School of Informatics
University of Edinburgh

http://www.inf.ed.ac.uk/teaching/courses/ds
Autumn Term 2012

Fundamental Concepts

- Distributed Systems are first and foremost complex software systems
- Architectural paradigms pertinent to distributed systems:
  - Layers
  - Client-Server

Layers

- The basic idea of a layered approach in general:
  - layer: A group of closely related and highly coherent functionalities
  - service: The functionality provided to the superior layer.

An Example Layer Approach

1. Physical transistors
2. Chip architecture; uses physical transistors and provides a set of (binary encoded) machine instructions for basic operations
3. Assembly code; uses binary codes to provide almost the same instructions in an alphabet incoding.
4. Systems programming language: compiler uses the assembly code to expose a high-level programming language such as C
5. Operating System (kernel): uses the systems programming language to provide a range of services to aid application programming
6. Application programming language: provides servies for the application programmer using the operating system and systems programming language

Layering in Distributed Systems

Typically:

1. Computer and Network
2. Platform: hardware and operating system providing access to network protocols
3. Middleware: Used to achieve transparency of heterogeniety at the platform level
4. Applications and services built on top of the middleware

Client-Server Architecture

- The Client-Server Architecture basic mode:
- Client: A process wishing to access some resource or perform operations on a different computer
- Server: Process which accepts requests from clients and processes those requests eventually providing a response
- The client is often referred to as the "active" player and the server the "passive" since it is the client which initiates communication.
- In common parlance a server is a machine but here it is a process
- In order to satisfy some request the server may become a client and make some request of a different server
- Where this is taken to an extreme we get "Peer Processes" which have largely the same functionality and do not describe a client-server architecture

Client-Server Architecture

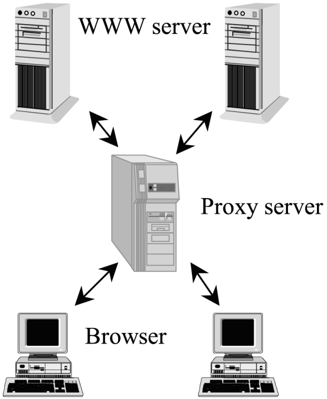
Variants – Multiple Servers

- Service provided by multiple servers
- Many commercial web services implemented by many different physical servers. This is so common now that it is almost the single server that is the variant.
- Motivation:
    - Peformance
    - Reliability
- Servers generally must maintain a replicated or distributed database

Client-Server Architecture

Variants – Proxy Servers

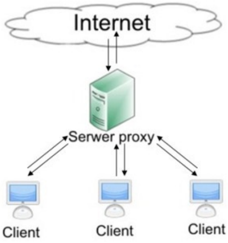
- Proxy server provides transparency of replication/distribution



Client-Server Architecture

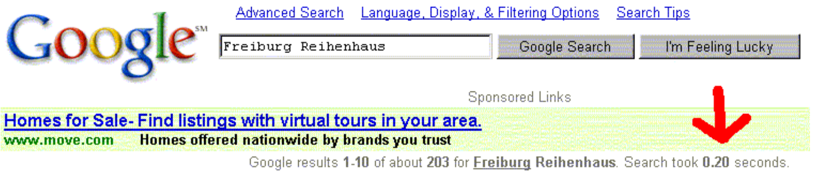
Variants – Proxy Servers

- Proxy server may maintain cache of responses to recent requests
- Requires that identical requests receive identical responses, often this means that the cache store is time bounded
- Frequently used in search engines



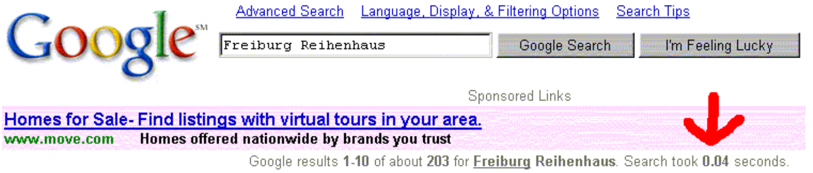
Client-Server Architecture

Variants – Proxy Servers



Homes for Sale- Find listings with virtual tours in your area.
www.move.com Homes offered nationwide by brands you trust

Google results **1-10** of about **203** for **Freiburg Reihenhaus**. Search took **0.20** seconds.



Homes for Sale- Find listings with virtual tours in your area.
www.move.com Homes offered nationwide by brands you trust

Google results **1-10** of about **203** for **Freiburg Reihenhaus**. Search took **0.04** seconds.

Client-Server Architecture

Further Client-Server Variants

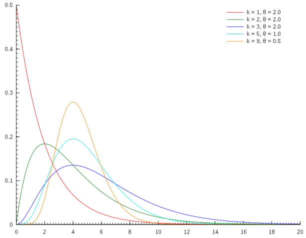
- Mobile Code
  - Code that is sent to the client
  - Java Applets, Flash etc.
- Mobile Agents — really a specific form of mobile code
- Thin Clients
  - Note so much a variant as an extreme example

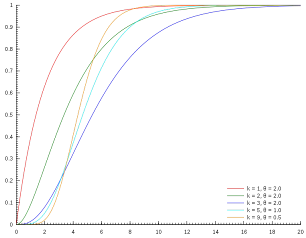
Client-Server Architecture

- Software Implications
  - Use of client-server has impact on the software architecture used
  - What kinds of requests and responses are allowed
  - What are the synchronisation mechanisms between client and server
  - Smaller shorter requests vs. Larger slower requests.

Client-Server Architecture

- Design Challenges
  - Quality of service
    - Performance: Response times





    - Performance: throughput
    - Performance: timeliness
    - Reliability: Server must obviously be generally available
    - Adaptability: For example to high and low demand
    - Dependability: Fault tolerance, not just the server but a client may be faulty (`isup.me`)
    - Security: The server is an obvious point to attack as well as the communication channels of any distributed system

Peer-to-Peer Architecture

- Client-Server approach scales poorly
- As the number of users grows so too do the demands on the centralised resources at the server
- In response Peer-to-Peer architectures arose from the realisation that the resources (computing, data and networking) owned by users of a service could be put to use to support that service
- This has a number of useful consequences but most obviously the shared resources available to users grows with the growth of new users.
- The distributed source code control systems described earlier could be described as peer-to-peer source code control.
- More to say on Peer-to-Peer distributed systems later

Fundamental Interaction Model

- Distributed System
  - Multiple processes
  - Connected by communication channels
- Distributed Algorithm
  - Steps to be taken by each process
  - Defines the communication between processes
  - Does not directly define the sequence of steps globally
- We create models to:
  - Make explicit all relevant assumptions about the distributed system we are modelling/designing
  - Make generalisations about what is possible given those assumptions, for example desirable properties such as no deadlock.
- Model aspects (may or may not be the same model):
  - Interaction model
  - Performance model
  - Failure model
  - Security model

Interaction Model

Synchronous distributed system

- time to execute each step of a computation within a process has known lower and upper bounds
- message delivery times are bound to a known value
- each process has a clock whose drift rate from real time is bounded by a known value

Asynchronous distributed system

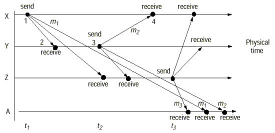
- no bound on process execution times
- no bound on message delivery times
- no bound on clock drift rate

Note

- synchronous distributed systems are easier to handle, but determining realistic bounds can be hard or impossible
- asynchronous distributed systems are more abstract and general: a distributed algorithm executing on one system is likely to also work on another one

Interaction Model

Event Ordering



- As we will see later, in a distributed system it is impossible for any process to have a view on the current global state of the system
- Possible to record timing information locally, and abstract from real time (logical clocks)
- event ordering rules:
  - if e_1 and e_2 happen in the same process and e_1 happens before e_2 then $e_1 \longrightarrow e_2$
  - if e_1 is the sending of a message m and e_2 is the receiving of the same message m then $e_1 \longrightarrow e_2$
  - Hence, $\longrightarrow$ describes a partial ordering relation on the set of events in the distributed system

Performance Model

Performance Characteristics of Communication Channels

- latency delay between sending and receipt of message
  - network access time (e.g. Ethernet transmission delay)
  - time for first bit to travel from sender's network interface to receiver's network interface.
- throughput: number of units (eg packets) delivered per unit of time
- bandwidth: amount of information transmitted per time unit
- delay jitter: variation in delay between different messages of the same type, (e.g., video frames)

Failures

Omission Failures

- process omission failures
  - detection with timeouts
  - crash is fail-stop if other processes can detect with certainty that process has crashed
- communication omission failures: message is not being delivered — dropping of messages
  - possible causes:
    - network transmission error
    - receiver incomming message buffer overflow

Arbitrary Failures

- process: omit intended processing steps or carry out unintended ones
- communication channel: corruption or duplication etc.

Failures

Class of Failure	Affects	Description
Fail-stop	Process	Process halts and remains halted. Other processes may detect this.
Crash	Process	Process halts and remains halted. Other processes may not detect this.
Omission	Channel	A message inserted in one outgoing buffer never arrives at the other end's incoming buffer
Send-omission	Process	A process completes a *send* but the message is never put in its outgoing buffer
Receive-omission	Process	A message is put in a process's incoming buffer but the process never receives it.
Arbitrary (Byzantine)	Process / Channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times.

Failures

Masking/Hiding Failures

- A service may mask an error by hiding it entirely or,
- converting it into a more acceptable type of error
- A reliable protocol can be built upon an unreliable protocol by requesting retransmission of dropped messages
- Message sequence numbers can be used to ensure no message is delivered twice, particularly when used with a guaranteed delivery protocol.
- Parity bits or checksums can be used to detect an error and thereby turn an arbitrary failure into an omission failure.

Security

- Two related problems:
  - We wish to make sure only the intended recipient(s) can receive a message
  - Additionally messages (for example invocation requests) should be authenticated so that we know from whom they originated
- These can be largely mitigated against with the use of modern cryptographic algorithms
- However their use incurs some cost which we may hope to minimise
- Denial of service
  - generating debilitating network or server load so that services become the equivalent of unavailable
- Mobile Code:
  - requires executability priviledges on target machine
  - code may be malicious

Summary — Fundamental Interaction Model

- We have looked at architectural models: Client-Server and Peer-to-Peer.
- These are complemented by fundamental models to aid in reasoning about behaviour:
  - Interaction model
    - Classifies models as synchronous or asynchronous
    - Identify basic components from which distributed systems are built
  - Performance model — sometimes combined with interaction
    - concerned with the efficiency of completing global tasks
    - can be used to compare approaches
  - Failure model
    - Used to analyse how resilient a distributed system is to failures
    - Can be used to classify what can go wrong and how that affects the system including other peers
  - Security model
    - Allows us to keep the costs associated with security measures to a minimum

Networking — Types of Networks

1. Personal Area Networks — generally wireless e.g. bluetooth
2. Local Area Networks
3. Wide Area Networks
4. Wireless local area networks
5. Wireless Wide Area Networks (3G and now 4G)
6. Internetworks — comprising of potentially many kinds of networks linked together by routers and gateways. The Internet being the most obvious example.

Getting Messages to Destinations — Switching

Broadcasting

- Broadcasting is one way of getting the message to its intended recipient
- Simply send it to everyone and have all the receivers filter their messages to receive only the ones intended for them
- A bit like *spam*
- Local area networks are commonly built on this technology (in particular Ethernet is)
- Wireless networks are necessarily broadcast networks
- Cryptography can be used to force filtering on the receivers
- Broadcasting does not scale well with the number of senders

Getting Messages to Destinations — Switching

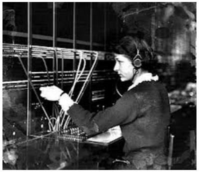
Broadcasting

Photo copyright Kwozie flickr user

Getting Messages to Destinations — Switching

Circuit Switching

- Was used for the telephone system



- Very rarely used for computer networks
- Circuit switching does have some advantages including greater efficiency once the circuit has been initiated
- Long distance networks required several switches in-between end-points.
- However it has several disadvantages including:
  - low adaptability to changing traffic
  - low adaptability to loss of communication channel

Getting Messages to Destinations — Switching

Packet Switching or Store and Forward

- When networks were built with computers so came the possibility to do some processing at each node along the path
- Packet switching is an example of what is called a "store and forward" network
- Each packet is treated separately at each node, it is first stored and then a decision is made about how and where to forward it
- The postal system is an example of a store and forward network, using packet switching

Getting Messages to Destinations — Switching

Packet Switching or Store and Forward

- Packet Switching can adapt to changing network conditions
- Including the loss of a communication channel
- They do incur some disadvantages, in particular packages may arrive out of order
- Packet lengths are restricted in order to:
  - Each computer in the network can allocate sufficient storage to hold the largest possible incoming packet
  - Avoid undue delays in waiting for communication channels to become free (essentially the same reason you don't send an unsegmented thesis to the printer)
- "frame relay" is a compromise between circuit and package switching.

Protocols

- Protocols enable communication between computers
- A protocol specifies:
  1. The sequence of messages that must be exchanged e.g. message - acknowledgement
  2. The format of the data in the messages
- A key idea is that of protocol layering
- Software at the sender and receiver is arranged in modules representing each layer
- Conceptually the software at layer N is communicating with the other computer at layer N
- But in reality is invoking and reacting to the layer below
- In particular one can build a reliable communication layer atop an unreliable communication layer.

Routing

- Routing is required in networks larger than a LAN
- Adaptive routing allows for changes in network traffic and connectivity
- A routing algorithm is implemented by a program in the network layer at each node
- It must:
  1. Determine the route taken by each packet as it travels through the network. A circuit switched network will set up a route for all subsequent packets but a packet switched network will perform the same steps for each packet. The routing algorithm in a packet switched network must therefore be simple and efficient.
  2. Dynamically update its knowledge of the network so as to better route subsequent packets/circuits
- Internet routing is essentially path finding in graphs.

Routing — Example Algorithm

Router Information Protocol (RIP)

1. Maintain a routing table:

Dest	Link	Cost
1	local	0
2	2	1
8	2	4

2. Periodically — and whenever the local routing table changes — send table in summary form to all accessible links
3. If a routing table packet is received from a neighbouring router update your own table accordingly:
   - If there is a new destination add that row to your table
   - If there is a lower cost route to an existing node update the appropriate row
   - If the table was received on link N replace all differing rows with N as the link
4. If a link $\mathcal{L}$ becomes unavailable set cost to ∞ for all entries with $\mathcal{L}$. Since the routing table has changed, send it to all accessible links.

Routing — Example Algorithm

Router Information Protocol (RIP)

Dest	Link	Cost
Allan	local	0
Bob	lBob	1
Alice	lAlice	1
Susan	lAlice	5

Routing — Example Algorithm

Router Information Protocol (RIP)

Bob sends me a new table and it has information about a node I hadn't seen before "Harry" at a cost of 8

Dest	Link	Cost
Allan	local	0
Bob	lBob	1
Alice	lAlice	1
Susan	lAlice	5
Harry	lBob	9

Routing — Example Algorithm

Router Information Protocol (RIP)

Susan now sends me an updated table and it contains information about “Harry” that she can get a packet there within 5 hops.

Dest	Link	Cost
Allan	local	0
Bob	IBob	1
Alice	IAlice	1
Susan	IAlice	5
Harry	IAlice	6

Routing — Example Algorithm

Router Information Protocol (RIP)

- Don't forget that after each of these updates I perform a send to all outgoing links.
- In particular Bob could now have received my table linking to Harry in 6 which would mean he would have a new route to Harry through me at a cost of 7 beating his previous 8.
- I now receive a table from Alice with the "Harry" link set to ∞.

Dest	Link	Cost
Allan	local	0
Bob	lBob	1
Alice	lAlice	1
Susan	lAlice	5
Harry	lAlice	∞

Routing — Example Algorithm

Router Information Protocol (RIP)

I then later detect that the link lAlice has been broken

Dest	Link	Cost
Allan	local	0
Bob	lBob	1
Alice	lAlice	∞
Susan	lAlice	∞
Harry	lAlice	∞

Routing — Example Algorithm

Router Information Protocol (RIP)

Bob then later sends his table which still has a link to Harry at cost 8

Dest	Link	Cost
Allan	local	0
Bob	lBob	1
Alice	lAlice	∞
Susan	lAlice	∞
Harry	lBob	9

Routing — Example Algorithm

Router Information Protocol (RIP)

- This algorithm has been shown to eventually converge on the best routes to each destination whenever the network is changed
- This is a simple version of the algorithm and it may be improved in many ways:
    1. The cost metric can take into account bandwidth
    2. Avoid undesirable intermediate states before convergence, such as loops.
        - Optional home exercise: show an example where there is a looping intermediate state
- Note: this is a distributed algorithm: I promised you that "parts of computer networks are distributed systems"

Networking Issues

- Performance — We are of course most interested in the speed with which individual messages can be transferred between two computers.
  - latency delay after a send is initiated before data begins to arrive at the destination
  - data transfer rate this is the bits per second rate that is quoted.
  - Message transmission time = latency + length/data transfer rate
  - Though longer messages may require segmentation into multiple messages
  - Latency affects small frequent message passing which is common for distributed systems

Networking Issues — Performance

- Time required to transmit a short message and receive a reply on a small local network: about half a millisecond (0.0005s)
- Time required to invoke an operation on an object in local memory: sub-microsecond (0.000001s)
- About a thousand times slower on the network
- However, networks can outperform hard-disks.
- So if you have one large server with a very large amount of system memory this may perform better than several machines with small amounts of system memory
- Over the Internet we might be looking at about 5-500 milliseconds
- Some of this is latency (switching delays at routers) and some is data transfer rate (contention for network circuits)

Networking Issues — Reliability

- Physical transmission media is generally pretty reliabile — though wireless less so
- Message losses are often due to software errors
- Many applications are able to recover and/or tolerate transmission errors.
- A guaranteed communication channel is often therefore needless overhead.
- In particular because the software itself may lose the message it must be designed to account for that — it may then as well cope with transmission failure by the communication channel.
- But it depends on the transmission media
- Must try to reduce the amount of incorrect data that is transmitted as well as the amount of checking done on correct data.

Reliability

$$\text{Left} \longleftrightarrow 1 \longleftrightarrow 2 \longleftrightarrow 3 \longleftrightarrow 4 \longleftrightarrow 5 \longleftrightarrow \text{Right}$$

$$\text{Left} \longleftrightarrow 1 \longleftrightarrow 2 \longleftrightarrow 3 \longleftrightarrow 4 \longleftrightarrow 5 \longleftrightarrow \text{Right}$$

$$\text{Left} \longleftrightarrow 1 \longleftrightarrow 2 \longleftrightarrow 3 \longleftrightarrow 4 \longleftrightarrow 5 \longleftrightarrow \text{Right}$$

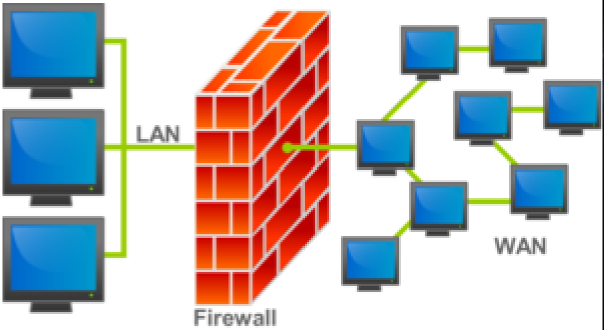
$$\text{Left} \longleftrightarrow 1 \longleftrightarrow 2 \longleftrightarrow 3 \longleftrightarrow 4 \longleftrightarrow 5 \longleftrightarrow \text{Right}$$

$$\text{Left} \longleftrightarrow 1 \longleftrightarrow 2 \longleftrightarrow 3 \longleftrightarrow 4 \longleftrightarrow 5 \longleftrightarrow \text{Right}$$

Red denotes a node at which error detection/correction occurs

- If the probability of a message getting through any channel is 0.5 then completing the trip is $0.5^6 = 0.016$
- Fortunately communication channels are generally more reliable
- $(\frac{9999}{10000})^6 = 0.9994 > \frac{999}{1000}$

Networking Issues — Security



- Security is generally handled more at the application layer
- Generally through cryptographic techniques
- Though the network can provide some level of security
- A firewall is catch all solution with associated inefficiencies
- For some organisations those inefficiencies are deemed appropriate.

Interprocess Communication

Interprocess Communication

UDP and TCP

- Two internet protocols provide two alternative transmission protocols for differing situations with different characteristics
- User Datagram Protocol — UDP
  - Simple and efficient message passing
  - Suffers from possible omission failures
  - Provides error detection but no error correction
- Transmission Control Protocol – TCP
  - Built on top of UDP
  - Provides a guaranteed message delivery service
  - But does so at the cost of additional messages
  - Has a higher latency as a stream must first be set up
  - Provides both error detection and correction

UDP and TCP

- User Datagram Protocol — UDP
  - Is connectionless
  - Used for small requests from possibly large numbers of clients
  - Examples: DNS, RIP and VOIP and online gaming
  - VOIP: The biran prefmros smoe erorr mkasnig for us
  - Sometimes used for larger requests when the application may be able to do its own error correction
- Transmission Control Protocol – TCP
  - Is connection based
  - Used for larger requests
  - Examples: SMTP, HTTP and TELNET

UDP and TCP

Failure Models

- User Datagram Protocol — UDP
  - Sometimes packages are dropped — no guaranteed validity
  - Messages may be delivered out of order — no guaranteed validity
  - Checksums are used to provide near guaranteed integrity
- Transmission Control Protocol – TCP
  - Uses checksums to give near guaranteed integrity
  - Uses sequence numbers, timeouts and retransmissions to provide guaranteed validity
  - If the communication channel is bad enough then timeouts may occur often enough for the connection to be deemed broken
    - Therefore there is no absolute guarantee of reliabile communication
    - Processes cannot distinguish between network failure and failure of the other process
    - Processes cannot be sure that recent messages have succeeded or failed.

Send and Receive

- Communication between to separate hosts is supported by two operations, simply send and receive
- Two modes of communication:
  1. Synchronous
     - communicating processes synchronise on every message
     - Hence both sending and receiving are blocking operations
     - Sort of the "instant messaging" of the data communication world
  2. Asynchronous
     - Only the receive operation is blocking, the send is not
     - The "e-mail" of the data communcation world
     - A form of non-blocking receive can be built, however this is really just a thread which waits and then sends a signal to the parent thread when there is received data to be read

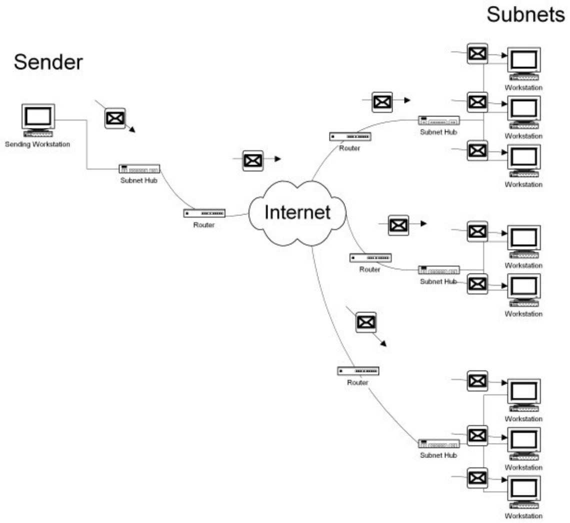
Sockets

- Sockets are a dominant abstraction programmers use for writing synchronous and asynchronous communication
- A process may use the same socket for sending and receiving
- A socket is associated with an Internet Address and a Port number
- Generally servers will advertise on which ports they will receive
- Only a single process can receive on a particular port
- Sockets can be used to send/receive UDP messages
- Sockets can also be used to set up a TCP stream of communication, generally two such streams are initiated to enable two-way communication between the two hosts.

Multicast

- For some applications it is appropriate to send a single message to many recipients
- Multicast is essentially a selective broadcast
  1. unicast
  2. anycast (one of a group)
  3. multicast
  4. broadcast
- The most common reasons for multicast are:
  1. Efficiency
  2. Simplicity/transparency for the sender, in particular the sender need not necessarily know all the recipients
- However there are some issues, in particular we must consider the failure semantics of multi-recipient messages.
- Attempts to provide strict failure semantics for multicast messages unfortunately often negate part or all of these two advantages

Multicast


Subnets
Sender
Sending Workstation
Subnet Hub
Router
Internet
Router
Subnet Hub
Workstation
Workstation
Workstation
Router
Subnet Hub
Workstation
Workstation
Router
Subnet Hub
Workstation
Workstation
Workstation

Multicast

Uses of Multicast

1. Fault tolerance based on replicated services
2. Data replication for increased efficiency
3. Discovery of services in spontaneous networking
4. Propagation of event notifications

Multicast

Plausible Failure Semantics

1. Maybe semantics — The multicast equivalent of UDP, some processes may receive each message some may not. They may receive messages in different orders
2. Either all members receive a message or none do, some may receive a message out of order
3. All members of the group receive every message in the correct order
    - called: *totally ordered multicast*
    - We will see this in more detail in a later part of the course

IP-Multicast

- UDP failure semantics:
- For each message, some members of the group may receive the message some may not
- IP-Multicast is built on top of IP
- The sender is unaware of the identities of the individual recipients of the message
- IP addresses (in IPv4) in the range 224.0.0.0 to 239.255.255.255 are reserved for multicast traffic and are managed globally
- Any socket (that is any port on any computer with an IP address) may join any IP-multicast group
- IGMP (Internet Group Management Protocol) is used both for requesting entry to a group and for communication between adjacent routers

IP-Multicast

- Upon receiving a multicast message a multicast router sends the message on to any links which have members of the group
- To avoid eternally propogating messages, each multicast message has a “Time To Live” variable which is decremented with each propogation
- Groups ownership is not addressed by the IP-multicast protocol
- For small local groups this can be achieved through using a small time to live number
- Over the Internet other solutions are required, for example Multcast Address Allocation Architecture is a client-server based solution, in which the server maintains addresses which are free.

Multicast XCAST Implementation

- An alternative way to implement multicast is to require the sender to attach each recipient address to the message
- This is used by XCAST (Explicit Multi-Unicast), which is implemented on top of IP and places each receiver's address in the IP packet header
- Since IP-packets are limited in size, this places a strict limit on the size of the group
- The group must also be known ahead of time
- However it is appropriate for use when there are a large number of small sessions which have a small number of groups
- Video conferencing for example

External Data and Marshalling

- Ultimately processes/algorithms wish to exchange data
- But messages are restricted to a sequence of bytes
- Hence the communicating processes must agree in advance a suitable format in which the data should be converted to/from a sequence of bytes
- Examples:
  - XML
  - Java serialisation
  - JSON
  - CORBA

External Data and Marshalling

CORBA

- Common Object Request Broker Architecture
- Marshals data for receivers that have prior knowledge of the types of the objects to be communicated
- Type information is defined in an Interface Definition Language (IDL) file
- IDL files can be automatically mapped to programming language type definitions and code to (de)marshall object
- Has the disadvantage that types must be agreed upon in advance
- Has the advantage that there is no overhead in communicating the type

External Data and Marshalling

Java Serialisation

- Includes the full type information in the marshalled data
- Uses reflection in order to obtain that type information
- Is restricted of course to use with the Java programming language (and languages specifically designed to interoperate)
- The .NET framework has a similar approach

XML

- More general than either Java Serialisation or CORBA
- Can be used in both modes, that is either to send type information together with the data or agree on pre-existing types

External Data and Marshalling

JSON

- Javascript Object Notation
- Includes type information, but that type information is basic
- Number, String, Boolean, Array, Null or
- Object — a list of key-value pairs
- Is becoming very popular because it is useful for many languages and requires no parsing by the application programmer
- In particular popular with dynamic languages such as Python

Summary

- UDP provides simple, efficient, connectionless sending of messages with few guarantees
- TCP provides connection-based sending atop UDP with greater guarantees of validity, no omission failures
- Programming APIs built atop these tend to rely on the *Sockets* abstraction to provide synchronous or asynchronous send and receive operations
- Marshalling is used to send complex data structures as one-dimensional sequences of bytes
- Different approaches may require prior agreement as to the types of the marshalled data and may make constraints on programming language used

Any Questions

Any Questions?

Distributed Systems — Questions

Questions

- Question : Are — and if not, why not? — platform layers not generally standardised to reduce/remove the need for middlewares like some kind of distributed POSIX?
- Answer : To some degree, for example the *Sockets* abstraction is widely implemented. Essentially middleware exists either because popular platforms have not agreed upon a common abstraction or because that abstraction more usefully sits outside of the realm of the "platform". Why platform vendors cannot agree upon common abstractions is more of a social and possibly economic question.

Distributed Systems — Questions

Questions

- Question : Proxy servers provide transparency of replication/distribution. Can they be classified as middleware?
- Answer : Middleware is a term used only for software, since a proxy must ultimately be realised in hardware we wouldn't normally say that a proxy is middleware.

Distributed Systems — Questions

Questions

- Question : With regards to synchronous and asynchronous systems, specifically determining realistic bounds, why can one not just specify bounds that covers all possible circumstances? For example, assuming that, say, HTTP GET requests will always return within 10 minutes if the server is available? Obviously this assumption is ridiculous, but at least then the bounds are known.
- Answer : The key point is whether or not one can determine useful bounds. The distinction is more in how we then treat the communication system. All systems can have fairly unreasonable bounds applied — a message may arrive instantaneously or may take 1000 years. Atop which you could attempt to build a reliable communication system using a timeout of 2000 years. Alternatively one could simply assume asynchronicity and build on top of that.

Distributed Systems — Questions

Questions

- Question : Can you give an example of a process omission failure and the difference between process omission and arbitrary failures?
- Answer :
  - A process omission failure is when a message which should have been sent (or received) simply isn't. It might be because the code of the process is erroneous, or it might be some software driver is incorrect. For example perhaps the message was put in the out-going buffer but that buffer was full and the software did not deal with that correct.
  - So for example in the RIP protocol one of the routers may simply fail to send on their updated RIP-table after an update
  - An arbitrary failure is when a message is sent, but the message sent is not the correct message. Generally this is more likely to be incorrect logic in the code of the process itself.

Distributed Systems — Questions

Questions

- Question : *Can you give an example of a process omission failure and the difference between process omission and arbitrary failures?* — continued
- Answer :
  - In the RIP algorithm one process may simply send an incorrect table. Or it may erroneously set all link costs to ∞ and hence — at least temporarily — continuously send incorrect routing tables.
  - The question is, given such an error, how does the distributed algorithm cope with this. Is it detectable? Is the behaviour acceptable even if it is not detected or in the meantime before it is detected?

Introducing PEPA

- PEPA: Performance Evaluation Process Algebra
- Modellers define their model by first describing a set of sequential components and then combining those sequential components together in parallel to form the main system equation.
- Definitions are built using the choice $(+)$, prefix $(.)$ operators.
- The system equation is built using the cooperation operators $\underset{L}{\bowtie}$, $\|$ and hiding $\backslash$.

Service Example

$$
\begin{array}{lcl}
\mathit{Service} & = & (\mathit{request}, \top).\mathit{Service} \\
 & + & (\mathit{service}, r_{\mathit{serve}}).\mathit{Service} \\
 & + & (\mathit{break}, r_{\mathit{break}}).\mathit{Broken} \\
\mathit{Broken} & = & (\mathit{repair}, r_{\mathit{repair}}).\mathit{Service} \\
 & + & (\mathit{request}, \top).\mathit{Broken} \\
 & & \\
\mathit{Client} & = & (\mathit{request}, r_{\mathit{join}}).\mathit{Wait} \\
\mathit{Wait} & = & (\mathit{service}, \top).\mathit{Client} \\
 & & \\
 & & \mathit{Service} \underset{L}{\bowtie} \mathit{Client}[\mathit{clients}] \\
\mathit{where}\ L & = & \{\mathit{request}, \mathit{service}\} \\
\mathit{and}\ \mathit{Client}[3] & = & \mathit{Client} \parallel \mathit{Client} \parallel \mathit{Client}
\end{array}
$$

State Specifications Examples

$\mathit{Broken} == 1$	The/a server is in the state *Broken*
$\mathit{Wait} > 3$	More than three clients waiting
$\mathit{Broken} == 1$ && $\mathit{Wait} > 3$	Both the previous are true
$\mathit{Service} < \mathit{Wait}$	Fewer servers ready than clients waiting

Activity Probe Specifications Examples

$a : start, b : stop$	Any state between the *a* and *b* actions
$P :: (a : start, b : stop)$	. . . as observed by a single *P* process
$(a\|b\|c) : start, (x\|y) : stop$	choice to start and end
$(a, a, a)/b : start, b : stop$	As before, without a *b* interrupting

PEPA

$$\bowtie$$

More information at: `www.dcs.ed.ac.uk/pepa`

Concurrent Finite State Machines

- An example of an interaction modelling framework
- Due to Brand and Zafiropoulo
- Consist of a set of finite state machines which can communicate via a set of communication channels
- Every FSM represents a concurrent, communication process
- One pair of channels (C_{ij} and C_{ji}) for each pair of machines
- Every communication channel is:
    - full-duplex
    - error-free
    - has a first-in-first-out strategy
    - had unbounded capacity
    - So this represents a perfect full-duplex channel

Concurrent Finite State Machines

Formalisation

- N: a positive integer
- i, j = 1, . . . , N indexes over processes
- $\langle Q_i \rangle_{i=1}^{N}$ N disjoint finite sets, Q_i denotes the state of process I.
- $\langle A_{ij} \rangle_{i,j=1}^{N}$ disjoint sets where A_{ij} denotes the message alphabet for the channel $i \longrightarrow j$
- $\forall i A_{ii} = \{\}$
- δ: relation determining for each pair (i,j) the following functions
  - $Q_i \times A_{ij} \rightarrow Q_i$: send from i to j
  - $Q_i \times A_{ji} \rightarrow Q_i$: receive from j at i
- $\langle q_i^0 \rangle$ the initial states such that $\forall i (q_i^0 \in Q_i)$
- AND SO: we call $(\langle Q_i \rangle, \langle q_i^0 \rangle, \langle A_{ij} \rangle, \delta)$ a protocol

Concurrent Finite State Machines

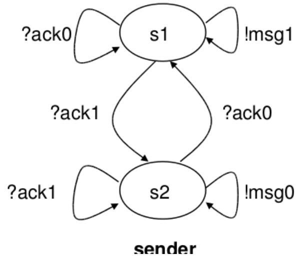
Notation

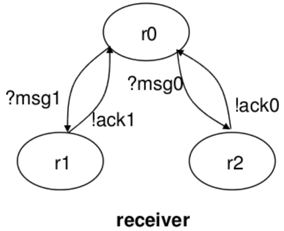
- `si` $\in Q_i$: state of process i
- `xij` $\in A_{ij}$: a message
  - `?xij` reception of a message
  - `!yji` sending of a message
- `f((s1, ..  sn)) = (f(s1), .., f(sn))`
- `x,y`: message
- `X,Y`: sequence of messages
- `x, xy, xY, xXY` : concatenated sequences of messages

Concurrent Finite State Machines

Alternating Bit Protocol

- Simple protocol securing unreliable message channels
- Sender sends message `msgn` with $n \in 0, 1$ a sequence number
- Receiver acknowledges with `ackn`
- Sender sets new sequence number at $1 + \mathrm{n} \bmod 2$
- Retransmission of current message when wrong sequence number receieved





Concurrent Finite State Machines

- Semantics of a protocol:
  - The set of admissable state sequences
- State of a protocol:
  - sum of:
    1. Local state of each of the processes
    2. state of all channels (which is the sequence of all messages along it which have been sent but not received)
  - We call this the global system state.

Concurrent Finite State Machines

Obtaining All Computations:

1. Initially all processes in their initial states and all channels empty
2. System is in a current global system state $\underline{s}$
3. State transition triggered by send and receive events
    - send event:
        - add a message to the tail of the appropriate channel
        - update the local state of the sending process
    - receive event:
        - take the message from the head of the message queue
        - update the local state of the receiving process
4. Leads to a new global system state

Concurrent Finite State Machines

State Transition Relation

- Let P be a protocol and G be the set of all global system states (S,C)
- The $\vdash$ $(G \longrightarrow G)$ is defined as follows: $(S, C) \vdash (S', C')$ iff $\exists i, k, xij$ such that (S, C) and (S', C') are identical other than either
  - si' $= \delta$(si, !xij) and cij' $=$ cij xij
  - or
  - si' $= \delta$(si, ?xij) and cji $=$ xji cji'

Concurrent Finite State Machines

Reachable Global System State

- Let:
  - G_0 be the initial global system state
  - G a global system state of the same protocol
  - $\vdash$ the state transition relation of the same protocol
  - $\vdash^*$ the transitive closure of $\vdash$
- We say that G is reachable if:
  - $G \vdash^* G$

Paths and the language accepted by a protocol can be defined via $\vdash^*$ as would be done for NFAs.

Concurrent Finite State Machines

Expressiveness

- Theorem: CFSMs are Turing complete
  - Many proofs possible (including proof by claiming it is obvious)
  - One such idea:
    - three processes P1, P2, P3
    - Simulate the control of the Turing Machine in P2
    - Use P1 and the channels c12 and c21 to simulate the left of the tape
    - Use P3 and the channels c23 and c32 to simulate the right of the tape
    - since all cij have unbounded capacity we have an inifite tape

Consequences

- global state space has unbounded size
- undecidable problems include:
  - termination
  - will some communication event ever be executed?
  - is some system state reachable?
  - is the protocol deadlock free?