Distributed Systems Architecture

15 January 2018 20:35

Distributed System

Processes communicate over network with **messages**

Common Objective of all types:

- Treat the system as one - hide the distribution from user

Motivation:

- Scalability
- Deployment
- Diversity of Resources
  - sharing
- Concurrency
  - reduce response time by local processing
  - increase throughout via parallelism

Types:

- Enterprise
  - client and server side applications
  - remote procedure calls - RPC
- Peer-to-Peer
  - publish and query to centralised directory
  - get data from peers
- Computing Clusters
  - parallel programming
- Grid Computing
  - many computing clusters spread over globe
    - working together
- Cloud Computing
  - virtual machines - e.g. AWS

Interaction Primitives

18 January 2018 14:07

Message Passing: (Inter Process Comms)

- Asynchronous Send: unblocked send: sender continues processing once message has been copied out of its address space
  - mostly used with **blocking receive**
  - MUST have buffering at receiver end
  - loose coupling - message can be processed later than receiving
  - problem - buffer overflow - no error reporting
  - maps closely to connection less
  - back and forth to synchronise

- Synchronous send: blocked send - pause until confirmation of message being received
  - to send async, use intermediary mailbox
  - problems - indefinite delays - never sent - use timeout - more complex implementation

Blocked Receive Primitive
- block is no message available
- else store into target variable if message available

Timeout Primitive (Variation of BR)
- specify a limit on blocking time
- if limit exceeded, take another action

Java API for UDP Datagrams

Two Java classes:

– `DatagramPacket` provides a constructor to make a UDP packet from an array of bytes

Bytes in Message	Length of message	Internet address	Port number

– Another constructor is used when receiving a message. Methods `getData, getPort, getAddress` can be used to retrieve fields of `DatagramPacket`

– DatagramSocket. Methods `send` and `receive` for transmitting `datagramPacket`. `setSoTimeout` for a receive to limit how long it blocks. If the timeout expires it throws an `interruptedIOException`.

Remote Procedure Calls

- make a procedure call looks as similar as to local call to the procedure
- **must provide parameters by value**
- encode parameters by value and method-name - then decode when receiving
- Stub:
  - client has *local stub* for **every** remote procedure it can call
  - server has local stub (skeleton) for every procedure which can be called
  - stub formed from method signature
  - Stub procedure:
    - assemble parameters into messages
    - unpack received parameters and assign values
    - access primitives to send/receive messages

Interface Definition Language
- takes code and generates stub and skeleton (server stub)
- link this compiled stub with user/server application
- e.g.
  - Pseudo code example

```
interface calc {
        void mult ( [in] float a, [in] float b, [out] float Res );
        void square ( [in] float a, [out] float Res );
}
```

Binding
- Server *exports* reference to interface to a nameserver
- Client *imports* reference from the nameserver
  - calls methods on this object reference

- First-part binding - client decides which sever to connect to
- Second-party - server decides which clients to serve
- Third-party - decides which clients connect to which server

RPC failure solutions
- best-efforts - Boolean call - return false if timeout - *maybe call semantics*
- at-lest-once - retry up to n times before fail - if received - immediately exit funciton with true - if call occurs multiple times - then this method is ok - not good for when multiple calls effect final output
- at-most-once - RPC guaranteed to not execute more than once - 0 or 1 times
  - keep track of received requests
  - remember reply until confirmation of receipt
  - fails when server crashes
    - received request buffer lost
  - but still most common in RPC systems
- transactional call
  - guarantee that either executed or not
  - atomic transactions
  - database - two phase commit

Java RMI

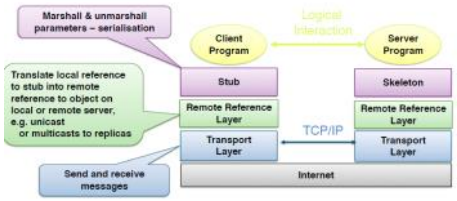
OOP in a distributed system
- more portable
- provides encapsulation without OS
- use inheritance with interfaces
  - specialise behaviour of interfaces
- create/destroy objects vs procedures

Java RMI Architecture

RMI Registry
- must run on every server running remote objects
- advertises availability of objects references

Dynamic Invocation
- bypass skeleton - query registry - send as byte array
- parameters
  - remote object ref
  - method to use
  - method's parameters

Java RMI Architecture



- more simple to program
- uses some Java specific stuff
  - definition of interfaces - for each object
  - reflection - query an object at runtime - dynamically assemble object functionality
  - serialisation - write an object to a byte stream
  - garbage collection - helpful for managing clients collected
    - server counts references for an object
    - clients increment ref. count
      - decrement ref. count when done
    - remote object can be removed when ref. count = 0

- remote **interface** - **extends Remote**
  - must be public
  - all methods must ___ **throws RemoteException**

- remote object - **class** that implement remote interfaces
    - extends **UnicastRemoteObject**
      - export object as single server
      - call "super()" in constructor - URO RMI linking etc
    - implements corresponding interface

  - implements methods defined in remote interface
  - invoked on remote object ref. from client - not server objects

RMI Server main:

1. Security Manager

```
if System.getSecurityManager() == null {
        System.setSecurityManager (new RMIsecurityManager ());
}
```

Create security manager

2. RMI registry

```
Registry r = Locateregistry.getRegistry();
r.rebind ("myname", this)
```

can be in constructor

3. register remote object

```
try {
Calculator c = new CalculatorImpl();
Naming.rebind("rmi://localhost/CalcService", c);
    }
catch (Exception e) {
```

Create server object

Register it with the local registry: URL-reference binding

RMI Client main:

```
Registry r =
   LocateRegistry.getRegistry("thehost.site.ac.uk");
```

need to get registry before trying to obtain remote reference

```
try {
        if System.getSecurityManager() == null {
          System.setSecurityManager (new RMIsecurityManager ());
          Calculator c = (Calculator) Naming.lookup(
                                "rmi://remotehost/CalcService");
```

Get ref to CalcServer stub from remote registry

```
   System.out.println( c.sub(4, 3) );
```

Invoke sub operation on remote calculator

```
   ... other calls ;
   } catch (RemoteException e) {
```

Have to cast remote object as Naming.lookup returns a reference to stub of remote object

- parameters
  - remote object ref
  - method to use
  - method's parameters

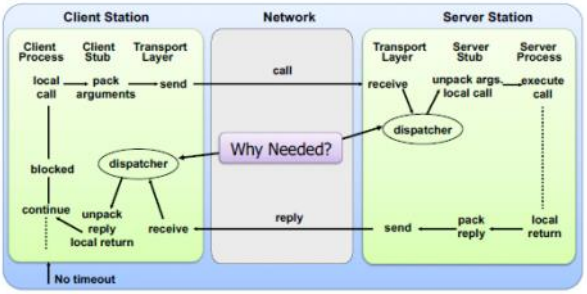
Dispatcher receives request,

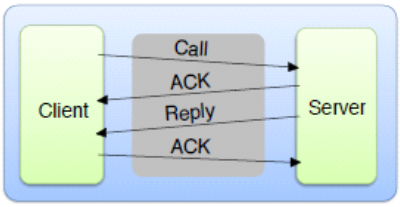
- unmarshalls method object,
- uses method information to unmarshall arguments
- converts remote object reference to local object reference
- calls method object's invoke method supplying local object reference and arguments
- when method executed, marshalls result or exceptions into reply message and sends it back to client

Interaction Implementation

01 February 2018 00:24

Remote Procedure Call





Error Control

- After sending message set timeout
- Retransmit if no ACK
- Save reply until ACK received in case call repeated.

RPC Binding

A name server registers exported interfaces and is queried to locate a server when an interface is imported.

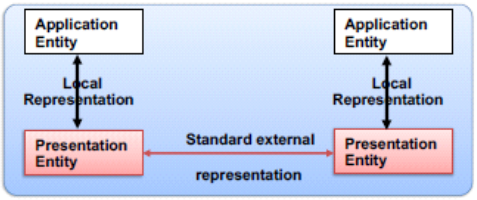
Server

- Calls export(interface type, server name, nameserver)
- Dispatcher address added by stub and passed to Transport
- Server's transport generates unique **exportid** & sends a **register** message to name server containing **type, name, exportid**.

Client

- Calls import(interface type, server name, nameserver)
- Dispatcher address added by stub and passed to Transport
- Client Transport:
  - Send query message with **type & name** to nameserver; Reply contains **type and address of server instance**;
  - Query server to check validity of **type, name and exportid**; Return interface reference (address) or error

Standard External Data Representation (XDR)



Can use:

- fixed length
- variable length
  - needs length field - overhead
- implicit type - know in advance
- explicit type ID field

Time Services

01 February 2018 00:24

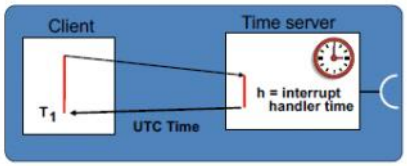
Clock Compensation

Assume 2 clocks can each drift at rate of r msec/s.
Max difference = 2r msec/s
To guarantee accuracy between 2 clocks to within d msecs requires resynch every d/2r secs.

Get UTC and correct software clocks. What happens if local clock is 5 secs fast and you set it right?
Time must never run backward! Rather slow clock down.

Clock register normally set to generate interrupts every 10msec and interrupt handler adds 10msec to software clock. Instead add 9 until correction is made or add 11 to advance clock.

Cristian's Algorithm



Time Server with UTC receiver gives accurate current time

Estimate of message propagation time p = (T1 - T0 - h)/2. Set clock to UTC + p

Measure T1 - T0 over a number of transactions but remove outliers and / or take minimum values as being most accurate

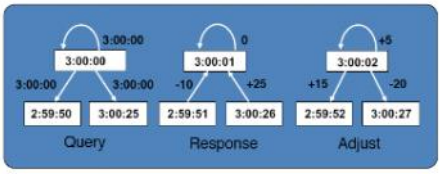
Berkley Algorithm

Co-ordinator chosen as master & periodically polls slaves to query clocks.

Master estimates local times with compensation for propagation delay

Calculate average time, but ignore occasional readings with propagation delay greater than a cut off value or whose current clock is badly out of synch.

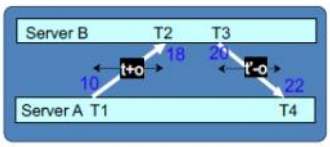
Sends message to each slave indicating clock adjustment



Synchronisation feasible to within 20-25 msec for 15 computers, with drift rate of 2×10^{-6} and max round trip propagation time of 10 msec.

Network Time Protocol:

- primary severs connected to UTC receivers
  - secondary connects to primary
    - tertiary connects to secondary
- Synchronisation modes
  - Multicast
  - Procedure Call Mode - Similar to Cristian's algorithm
    - used when higher accuracy needed
  - Symmetric protocol
    - highest accuracy - used by masters and layers closest to primaries



t = transmission delay (e.g. 5 ms)
o = clock offset of B relative to A (e.g. 3 ms)
Let a = T2 - T1 = t + o, Let b = T4 - T3 = t' - o
RTT = t + t' = a + b = (T2 - T1) + (T4 - T3)
If T1 = 10, T2 = 18, T3 = 20 and T4 = 22 then RTT = 10
2o = a - b = (T2-T1) - (T4 - T3) + (t - t') = (T2-T1) - (T4 - T3) = 8-2 = 6
Clock offset o = (a-b)/2 = ((T2-T1) - (T4 - T3))/2 = 3
(assuming t ≈ t)

Screen clipping taken: 08/04/2018 18:38

- T1 and T3 send current message send time
- T2 and T4 have current message receive time

Logical Time:

- ordering of events, $x \rightarrow y$
  - x occurs before y
- **Lamport's Logical Clocks**
  - process keeps its own logical clock
  - increment before assigning stamp to an event
  - timestamp messages when sending - after increment
  - on receiving a message, process choses max of its own counter and the message's timestamp
- include process id with event id for total ordering
- Vector clocks
  - each process has a vector of ints - size N - N num. of processes
  - increment your own element
  - can look at count for other elements
  - update on message being recieved

Name and Directory Services

01 February 2018 00:26

Naming Concepts

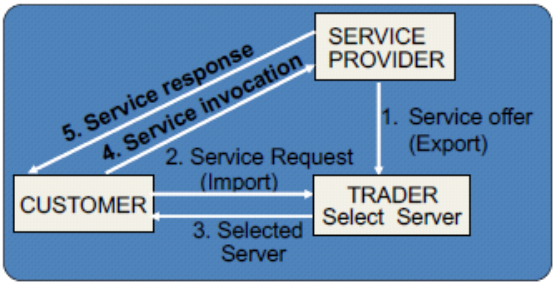
- string
  - names, address routes etc
  - interpret it as defined
- Identifier - unique
  - tied to a specific object
- Name Space
  - syntax and rules
  - how to interpret the string
  - name context
    - can have same name but in different location
- Naming authority
  - controller of assigning names
- alias
  - alternate name to the same object
- Flat Names
  - single global context
  - single global authority
- Name Binding
  - association between name and an object

Directory Service

- job - translate names into attributes
- Distributed Name servers
  - hierarchy
  - maintain reference to server above
  - recursive lookup
    - ask server above until root
    - answer propagates back down
    - or sent directly back to origin of request
  - iterative lookup
    - refer to server for address to next one up
    - then ask new server until destination reached
    - removes load from final server
  - replication
    - update needs to be resolved
    - consistency
      - strong - block until matching
      - weak - eventually propagate update
  - e.g. DNS - Domain Name Servers and X500
- Directory Information Tree
  - tree for the hierarchy
- Directory Access Protocol - DAP
  - DUA - User Agent
  - DSA - System Agent
  - DUA's connect via DSA's
- LDAP - lightweight DAP
  - TCP/IP rather than OSI

Trading

- Service Provider - Server exports offer
- Customer - requests - import service request
- TRADE Server - connect customer to a provider



Public keys over names:

- for
  - simple and mostly likely unique - ID
  - anonymity
  - simplify verification
- against
  - hard to remember
  - human error when typing

Security

01 February 2018 00:05

Threats:

- Attack
  - Information
  - Service
  - Users
- Types
  - Passive
    - observe without interfering
  - Active
    - modify messages
    - masquerade as authorised user
    - denial of service

Security Goals:

- Confidentiality
- Integrity of information - not modifications
- Availability of service
- Requires
  - ID
  - Authentication
    - Cryptography
  - Control access
    - Network
    - OS

Firewall Architectures:

- One Bastion Host
  - Bastion as filter and gateway
- One Packet filter - basic filter
  - use more - performance - reliability
- Screen Host - filter screens for Bastion
  - Packet filter - filter packets to Bastion
    - source or dest must be Bastion Host
    - block packets for internal network
    - block packets with IP Source routing
  - Bastion Host - run Gateway on packet filtered connections
- Screened Host - Dual-homed
  - Bastion Host also filters packets
    - what if the filter gets compromised?
- Screened subnet
  - Screen Host with another filter between Bastion Host and network
    - prevents attack from compromised Bastion
    - block packets with IP Source routing, and unwanted packets
    - only pass packets that have source or dest as Bastion Host
      - and for defined proxies on the gateway
    - block everything else

Cryptography

- Encryption
  - transformation of information
- Decryption
  - use a **key** to retrieve original information

Symmetric Key

Same key used for both encryption and decryption

- how to safely share a key?
  - via Asymmetric keys
- send large data over symmetric keys

Key Management

- Hierarchical
  - master keys used to generate new ones periodically
- Session key distribution server - e.g. Kerberos

Firewalls

- Filter packets from external networks going into and from internal network
- Criteria
  - address - permit certain IP's
  - message type - HTMAL, email, FTP etc

- Components
  - Filters - packets - outgoing or incoming

Rule	Dir	Action	Inside Addr	Inside Port	Outside Addr	Outside Port	Description

    - based on source or destination address
    - and / or ports etc
    - **last rule - block anything else**
    - rules interpreted in order
      - first rule that matches applies

- Block Spoofing
  - in, source your IP, block
  - out, source not Bastion, block
- TCP rules (2)
  - add ACK reply rule
  - web server >1023 d port reply
  - mail server - any d port
- UDP - basic (1)
- FTP (4 rules)
  - server listens on 21 (TCP)
  - send on 20 (TCP)
  - user port >1023
    - passive - client opens both
    - active - open one each
- DNS
  - **tcp or udp**
- proxy - flip - act as client
- block all else

In TCP, an initial open request packet does not have the ACK bit set, subsequent packets do

Rule	Dir	Action	Src Addr	Src Port	Dest Addr	Dest Port	TCP Flags	Description
1.	Out	Allow	*	*	*	25		Mail out
2.	In	Allow	*	25	*	*	ACK	Only replies allowed

Usually ACK done with D Port >1024

- Gateways - higher level-processing e.g. authenticating users
  - Bastion host
    - runs CL or AL GW's
    - runs a trusted, minimal OS
    - admin via terminal
    - read-only is possible
  - Circuit-Level
    - monitor TCP handshaking
      - determine if session is legitimate
    - good for auditing
  - Application-Level
    - like CL-GW - but specific to an application
    - look at header AND contents of a packet
      - higher layer in OSI or TCP/IP model filtering
-
- End-to-End encryption between firewalls
  - creates a VPN

Symmetric Key Distribution and Authentication

KDC: - Key Distribution Centres

- Setup a Session

A -> KDC: $E_A\{\text{request } K_S, B\}$

KDC -> A: $E_A\{K_S\}, E_B\{K_S\}$

A->B: $E_B\{K_S\}$

- A requests a session key - Ks
- KDC gives Ks encrypted for A and B - both sent to A
  - A passes on to B
- B doesn't know who sent the last message (we know it was A)
  - need to add timestamps

To Solve **(Needham-Schroeder)**:

A -> KDC: A,B, Req, N_A

KDC -> A: $E_A\{K_S, \text{Req}, N_A, E_B\{K_S, A\}\}$

Key Management

- Hierarchical
  - master keys used to generate new ones periodically
- Session key distribution server - e.g. Kerberos

Use two Keys - **Asymmetric**:

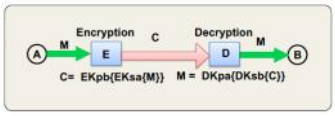
- Public
  - sent out and stored on name servers
- Private
- Both can be used for encryption and decryption

If A wants to send M

- encrypt with A private
- then B public - call it C

If B to receive C

- decrypt with B private
- decrypt with A public - get M



Use public key encryption where:
DKpub(EKpriv(M)) = M and DKpriv(EKpub(M)) = M

Hashing:

- checks integrity of message
- no secrecy
- one way

Certification using Hashing:

- Hash file
- send file and ID - encrypt with your private
- Origin Authority (OA) uses your public to decrypt
- adds a timestamp
  - encrypt with secret key - store the below

$$K_{secOA}\{Alice, H(file), t\}$$

Alice is the ID

  - can re-check later with public key to see contents for verification

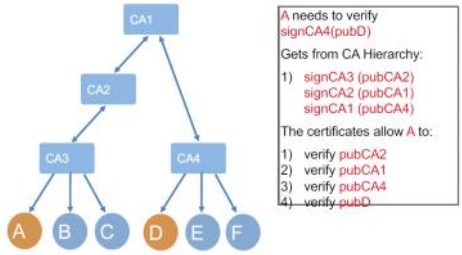
Certification of public keys

- CA - only an authority on issuing a certificate, not content
- Certify an identifier to a public key - signed by certifier

- Certification authority hierarchy
  - follows path up tree to get to required user
  - to get to node authority - need base 'hardcoded' key

X 509 Certificates

- parent-child pair of Certification Authorities (CA)
  - create a certificate of each other



- not encrypted - diff notation

To Solve **(Needham-Schroeder)**:

A -> KDC: A,B, Req, N_A
KDC -> A: $E_A\{Ks, Req, N_A, E_B\{K_S, A\}\}$
A->B: $E_B\{K_S, A\}$
B->A: $E_{Ks}\{N_B\}$
A->B: $E_{Ks}\{N_B-1\}$

N - nonce - used as a timestamp
can be refreshed
Blue parts authenticate the connection

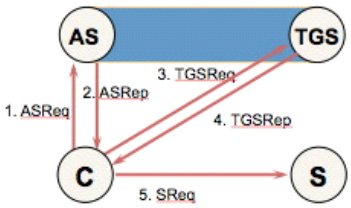
Kerberos Authentication Service

- KDC **shares unique secret key** with each principal (clients and servers)
  - stores passwords for each user - to generate key to encrypt replies to user requests
  - split into Authentication Service and Ticket Granting Service (AS and TGS)
  - **AS provides TGT** - {name, session key, timestamp}K_{KDC} - **once per user login**
  - **TGS provides ticket for each service** - once per service]

- Client A - start session
  - login with username and password - **generate A master key**
  - request a TGT from AS - plain text
  - **AS replies with Session key and a TGT** - encrypted with A's master key
    - **$\{S_A, \{A,S_A,T\}K_{KDC}\}$ K_{AS}**
    - A decrypts reply then removes master key from memory

- Client A - request to connect to a service
  - send message to TGS, contains - **service name, TGT, {timestamp}S_A**
    - TGS decrypts TGT,
      - obtains session key and details of A
      - uses S_A to check encrypted timestamp
  - TGS reply - encrypted by A's session key - S_A
    - **$\{B, K_{AB}, \{A,K_{AB},T\}K_{BS}\}S_A$**
    - generate session key for client and server - **K_{AB}**
    - A passes on highlighted ticket to B
      - encrypted with K_{BS} - so server B knows A was authenticated by KDC

- Client A - talk to Server B
  - send message to server B
    - **$\{A, K_{AB}, T\}K_{BS}$, {timestamp}K_{AB}**
    - B can decrypt first part using it's master key
      - obtain AB session key - use this to decrypt timestamp
  - server B replies with
    - **{timestamp+1}K_{AB}**



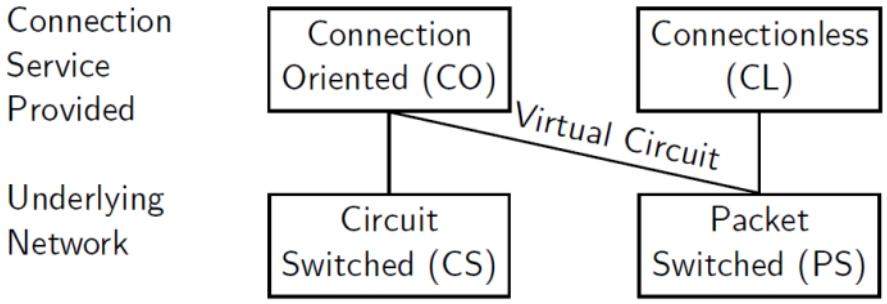
Intro

01 February 2018 00:05

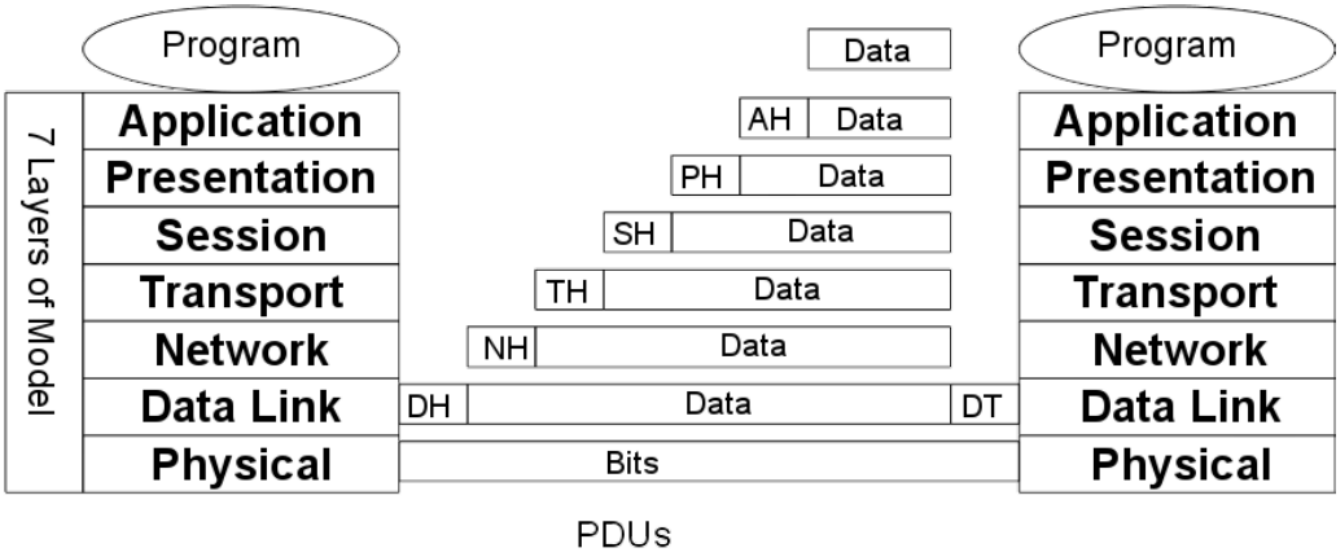
Circuit vs Packet Switching:

- Fixed bandwidth
- Unused bandwidth wasted
- Call set-up required
- Congestion may occur at call
- Overhead on call setup only
- In-order delivery
- Circuit fails if any link or node fails

- Variable bandwidth
- Uses only bandwidth required
- No call set-up
- Congestion may occur at any time (causing delay and reordering)
- Overhead on every packet
- Out-of-order delivery
- New route found if any link or node fails (some data may be lost)



ODI Network Model



Bottom 3 - standards and protocols of the underlying channel

Top 3 - application-orientated standards and protocols to allow end-to-end communication

Networks and storage use 1000 instead of 1024 etc

Physical Layer

15 February 2018 14:03

Manchester Encoding:

- **send clock with data**
- XOR or XNOR clock and data
  - determine value form transition of signal in one clock cycle
  - transition in middle
- requires double bandwidth

Differential Manchester Encoding

- presence/absence of transitions on clk falling edge
- start at HIGH
- 0 - transition at start and middle
- 1 - transition at middle only

Time Division Multiplexing:

- given width for some part of the message
- round-robin
  - share same channel bandwidth
  - decreases digital bandwidth
    - must be less than channel bandwidth / number of channels
- delay slots
- need an allocator and time sync
- bad for bursty data

Frequency Division Multiplexing:

- different carrier frequency
- need guard bands - filters not perfect

Code Division Multiple Access:

- talk same time
- but combine data bits with own code sequence
- separation using coding theory

Digital Phone Lines

- ISDN Basic Rate Interface
  - 2x 64kbit/s data
  - 1x 16kbit/s control
- ISDN Primary Rate Interface
  - 23 (US) or 30 (EU) 64lbit/s data channels
  - 1x 64kbit/s control channel

Ethernet - IEEE 802.3

- 10Base-T: twisted pair cable, allows 10Mb/s (still found in older networks)
- 500m max segment length
- 100Base-TX (fast Ethernet): two twisted pair cables (CAT5); full tuplex
  - Most common cabling in office LANs today (allows 100Mb/s)
  - 100m max segment length
- There is also 1000Base and 10GBase

Wireless - IEEE 802.11

- 2.4Ghz and 5Ghz
- 2.4Ghz divided into 14 channels
  - 1,6,11 are non-overlapping

Data-Link Layer

20 February 2018 14:02

Arrange data into bit-stream to send over physical layer

LLC - Logical Link Control
- single hop
- low level flow and error control

MAC - Media Access Control
- Framing
- Addressing
- Channel Access

Types of service:
- Unacknowledged connectionless
  - no logical connection - each frame is independent
  - no recovery - but fast
- Acknowledged connectionless
  - good for unreliable channels - wireless
  - out of order delivery is possible
- Acknowledged connection-orientated
  - frames are sequenced - in order delivery
  - reliable bit stream
- Error detection and correction (LLC)
  - Detection
    - parity bits
    - Cyclic Redundancy Check
      - Hash-based checksum
    - good for fast / low error rate channels
  - Correction
    - error correcting coding
      - e.g. FEC
    - good for slow / high error rate channels

Data Framing:
- group bits into smaller messages
  - better for retransmission
  - meta data - for LLC and more

Start and end flags:
- either side of Frame data
- if data is same pattern as flag - use and ESCAPE pattern
  - can escape the escape pattern as well

Min Frame size trade-off
- small - lower access delay
  - header is more overhead as % of bits
- large - more time for CD
  - longer access delay
  - padding for small data

MAC **Dynamic Channel Allocation:**

issue of frame's colliding when two devices try to use channel same time
- don't use static allocation as channel wasted when not being used

ALOHA:
- send when data ready
- central station resends same data back
- if reply is garbled - collision occurred
  - wait random time and retry
- theoritcal max of 18% utilisation

Slotted ALOHA:
- can only send data at agreed time intervals
  - central station broadcast signal at start of each time
- 37% chance of no collision

CSMA - Carrier Sense Multiple Access
- listen before sending

IEEE 802.3 / Ethernet:

Frame Format:
-

Bytes:	7	1	6	6	2	0-1500	0-46	4
	Preamble	SFD	DA	SA	Length	Data	Pad	FCS

  - Permeable
    - alternating 0s and 1s - synchronisation of speed
  - SFD - start of frame delimiter
    - 10101011
    - receiver can miss successive preamble as long as it detects SFDs
  - DA - dest. address
    - (16 or) 48 bits
  - SA - source address
    - (16 or) 48 bits
  - Length
    - optional - number of bytes in Data
  - Data
    - MTU - maximum transmission unit of 1500 bytes
  - Pad
    - ensure frame long enough to enable collision detection
  - FCS - frame check sequence
    - CRC - excluding preamble, SFD and FCS

Ethernet
- does not have SFD field
- Type instead of Length Field
  - identifies higher level protocol
    - use values that can't be used for length
  - easy to detect which field

Addresses:
- usually 48 bits
  - written as 6 pairs of hex digits - 00:11:22:33:44:55
  - MAC address is the ethernet address
- send in bytes, MSB to LSB of address
  - for each byte
    - send from LSb to MSb
- first 3 bytes - vendor code
- last 3 bytes - unique code set by vendor

- special addresses - first byte
  - **multicast - first bit is 1**
    - otherwise 0 - unicast
  - **global assigned - second bit is 0**
    - otherwise rest of vendor code is 0
      - apart from this bit = 1 - locally assigned address

Collision Detection:
- IEEE 802.3 uses CSMA/CD
- send frame for twice the propagation delay between nodes
  - slowest rate - 10Mbps - 500 bits
  - max length 2.5km
    - send for 50us min
    - repeaters add delay
  - rounded to **min frame length of 512 bit**s
  - if no data, - need 46 bytes for pad
  - **51.2us (@10Mbps) (slot time)**
- Retry Timing
  - wait between 0 and 2^n-1 slot times on nth retry
    - max 0 to 1023
    - give up on 16th retry

Same host inter-frame gap
- **wait 9.6us before sending next frame**
- give other hosts a chance to use channel

- 37% chance of no collision

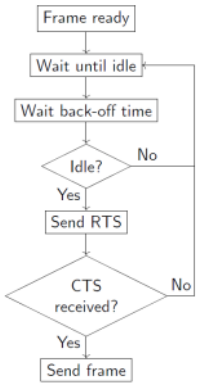
CSMA - Carrier Sense Multiple Access
- listen before sending
- send whole frame when channel not busy - i.e. channel is idle
- if collision occurs - wait random time and retry
- with Collision Detection
  - listen while sending
  - if collision - abort and retry after random time
  - channel time not wasted sending broken frames
- **time to access network is unbounded**

ALOHA - low performance - simple - equal
CSMA/CD - unbounded access time - equal
Token Passing - avoids collisions - bounded access - complex

IEEE 802.5 Token Ring:
- access channel by passing around token
- data and token passed around in a ring
- bounded access time
- complex to implement

MAC in Wireless LANs:
- distributed MAC - no centralisation
- collision cannot be detected while sending data
- no guarantee a node can connect to any other node on the network

Collision Avoidance:
- sender and receiver exchange short frame
  - can be overhead by other stations
- **RTS** - Request To Send - sender
- **CTS** - Clear To Send - receiver
  - both include size of data frame
  - so other stations know how long to wait
- Receiver - send **ACK** on receipt of data frame
- repeated failure results in greater back-off time



CSMA with CA

Inter Frame Spaces - IFS
- back-off time based on IFS
  - **lower IFS can give a frame priority**
- similar to IEEE 802.3 inter frame gaps

IEEE 802.11 - WLAN
- defines physical and MAC layer for WLAN
- Basic Service Set
  - one access point - zero or more stations (devices)
  - stations and AP share same medium and same MAC protocol
  - BSS can overlap with other BSS
    - different WLAN operating on same channel / medium
- Extended Service Set
  - multiple BSS connected
  - appears as single WLAN
- 2.4GHz and 5GHz ISM band
  - regulate power usage

Repeaters - amplify electrical signal
Hubs - join all input ports electronically / physically
- input ports at same speed

Both:
- physical layer
- adds to propagation delay

Bridge / Switch:
- operate on data link layer
- forward frames based on MAC address
- form end of collision domains
  - hosts on a LAN doesn't compete with other hosts on diff. LAN for access
- supports mix of standards / speeds ands protocols
- adds processing delay
  - as well as propagation delay

Switch:
- Table of MAC addresses and ports
- relay frames after reading MAC addresses inside the frame
  - if dest on same port as source - don't forward
  - else if dest had corresponding port - forward to that port
  - else - flood - send to all ports except source
- Table management
  - backward learning - listen to traffic
    - based on source and destination MAC address in frame header
    - TTL on entries
  - Spanning tree
    - no loops between switches
    - ensure it's possible to get source address
    - can be handled in Network layer with TTL on packets
- Delay
  - adds transmission delay
  - adds processing delays
    - store entire frame
    - verify CRC (FCS) before forwarding
  - Cut-Through switch
    - skip verifying CRC
- Virtual LANs
  - 802.1Q - adds VLAN identifier to frame header
  - switch must be VLAN aware - not backwards compatible
  - configured manually via VLAN tables

Network Layer

05 March 2018 22:33

Router:

- forward packets based on destination networks (not hosts)
- Routing table
- updates header fields affected by routing
  - TTL
  - Smaller MTU?
    - Total length, Flags, Frag Offset
  - Checksum
- adds transmission and processing delays
- can queue packets

Routing:

- static - compute route once - load into router
  - fixed table
    - default dest. if no match
  - used in most workstations

- flooding
  - send packet to all neighbours except source
  - remove loops
    - add sequence number
    - doesn't send if seen before

- dynamic - adapt to network topology and load
  - used in packet-switched networks (CL)

  - **Distant Vector Routing**
    - table of distances
    - exchange table data with neighbours
      - update tables after an exchange
    - cons
      - slow to converge
      - node goes down - **count to infinity problem**
        - B link to A gone
        - B goes to C to go to A
        - not realising B is used by C to go to A
      - doesn't consider bandwidth

  - **Link State Routing**
    - each router maintains a map
      - network - more than just neighbours
      - considers cost metrics (delay/bandwidth..)
    - faster convergence, reliable
    - less bandwidth use, but more CPU/mem intensive
    - example of a link state packet
      - routers send these to neighbours (via flooding)
        - are then forwarded
      - used to create a map of network
      - B - has connection to A, C, E

B	
SeqNo	
TTL	
A	4
C	2
E	6

        - source ID
        - SeqNo
          - record this, only forward more recent ones
        - TTL
          - info for current router - decrements
          - delete entry for source ID reaches 0
          - handles corrupt seqNo

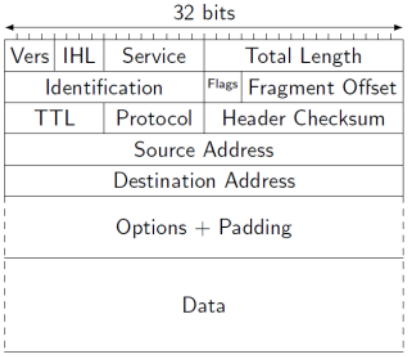
  - Hierarchical Routing
    - infeasible to have map of entire global network
      - router knows local topology in detail
      - and route to global, but not with same details
    - regions
      - can be geographically based
      - separate and organisations network

  - **Internet Routing**
    - AS - autonomous systems - regions on Internet (e.g. ISPs)
    - within an AS - variant of Link State (OSPF / IS-IS)
    - between AS - Border Gateway Protocol (variant of DVR)

Internet Protocol - IP:

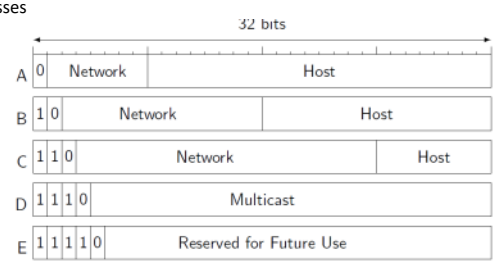
- Datagram orientated
  - data payload size can vary
  - checksum on header, not data
  - unreliable delivery
  - fragmentation - can split data over multiple packets - if Data-link layer requires

- Datagram Format - IPv4



  - **Internet Header Length** - IHL - multiples of 4-bytes - min value of 5
    - (Options + Padding can increase it from 5 to 15)
  - Service - type - (priority)
  - max length - **64KB for IPv4**
  - **fragmentation**
    - each smaller data payload needs its own IP header
    - set Id, Flags and Fragment offset - aids destination to recombine
  - TTL - decrement at each hop in Network
    - drop packet when 0 - (handles routing loops)
  - Proticol - 0 is reserved
    - **TCP = 6, UDP = 17, ICMP = 1**
  - **Header Checksum** - *1s compliment sum of header*
  - Options - padded to multiples of 32 bits

- IP Addressing
  - written as dotted decimal - byte value in decimal - separate bytes with a dot
  - Classes

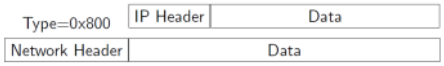


    - look at first 0 for class

  - Subnets
    - split Host field into subnet and host
    - mask - 32 bits - 1s apart from bits used for Host field
      - non-compliant - all 0s or 1s - subnet and/or host

  - Special Addresses
    - loopback - 127.0.0.0/8
    - local host - 0.0.0.0
    - Network ID - Network part + 0s
    - Broadcast - Network part + 1s
  - Private Range
    - never routed publicly
    - /n - fixed value of first n bits
    - 10.0.0.0/8
    - 172.16.0.0/12
    - 192.128.0.0/16

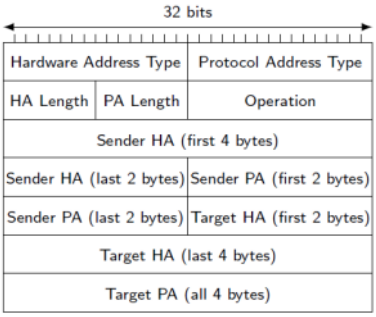
IP and MAC

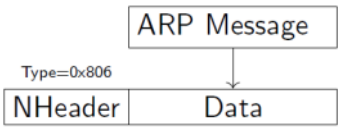


- IP is ALWAYS the final destination
- MAC - physical destination - changed at each hop
  - IP packet removed from frame
  - determine next link
  - create new frame - IP packet is payload of the frame

Dynamic Address Resolution - **ARP:**

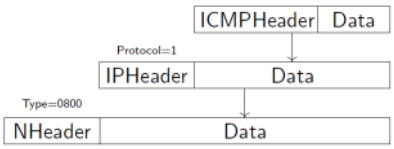
- host maintains lookup tables of IP to data link address
- if no entry for a IP
  - send ARP Request for that IP
  - IP recognised by other host on the LAN
    - sends ARP Response - with its data link address
  - optimisations
    - cache the table
    - when responding, requesters entry as well
  - Option: 1 is Request, 2 is Response
  - Target swaps Sender and Target in response





Internet Control Message Protocol - **ICMP:**

- used to investigate
- behaves as higher level, but part of IP
  - **so addresses IP**. not IP:port



- Header
  - Type and Code - 1 byte each
  - checksum of first 2 bytes
  - rest is dependant
- Popular Clients
  - **Ping**
    - verify connection
    - get round trip time, dropped packets..
  - **Traceroute**
    - details of the hops taken
    - router sends error if TTL is 0
    - send with ++TTL - find next hop

Dynamic Host Configuration Protocol - **DHCP:**

- DHCP Discover packet
  - Host broadcasts a request for IP
- DHCP Sever responds
  - DHCP OFFER packet
    - sends to MAC address of host
- lease IP for fixed period
  - a host must renew
- OFFER also gives
  - IP address of default gateway
  - subnet mask
  - IP address of nameservers and timeservers

Network Address Translation - NAT:

- External: use one routable IP address
- Internal: private IP address range
- Gateway box
  - translates
  - maintain mappings, including ports

IPv6:

- 128-bit address
- 7 field header
  - faster processing
  - more options though extension headers
- can translate between this and IPv4
  - but not compatible directly
  - lose most benefits of IPv6

Transport Layer

10 April 2018 00:14

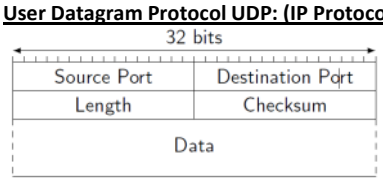
UDP - real-time, small, transaction-less.
TCP - long-lived, reliable

IP Datagram - unreliable connection-less packet service

Ports:

- 16-bit unsigned int, 0 - 65535
- the end-point within a host

User Datagram Protocol UDP: (IP Protocol 17)



- Source is optional (0 default)
- Length
  - in bytes
  - min 8 - no data
  - max depends on network layer protocol
- checksum is optional
  - 1s compliment of sum:
    - IP pseudo header + UDP header + data
  - IP pseudo:
    - Source IP
    - Destination IP
    - Protocol (0x11)
    - packet length
    - can't include as IP don't belong to this level

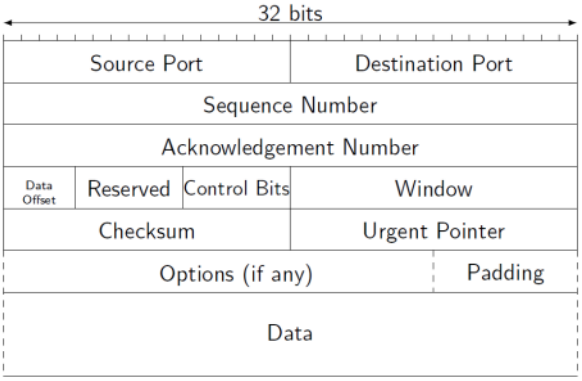
Transmission Control Protocol TCP: (IP Protocol 6)

- connection-orientated service on top of connection-less IP service
- Streams
  - stream of bytes - datagrams concealed
- Reliable
  - detects losses and retransmits
  - maintains order
- Flow
  - control rate of data
- congestion control
  - monitors network to optimise throughput without overloading

Connections:

- endpoint = IP address : TCP port
- Connection: Protocol(TCP) + source IP + source port + dest. IP + dest. port
- send receive same time, but no multicasting or broadcasting
- Open - waits - Server
  - 3 exchanges - SYN
- Active - initiates - client
  - 3 exchanges - FIN

Header



- ACK - value = next expected Sequence number
- Window size - how much space left in buffer
  - sender should not exceed this value
- Checksum - same as UDP
- Urgent P - higher priority data - e.g. errors
- data length obtained from IP header

TCP Procedures

Control Flag

- **SYN**: synchronise sequence numbers
- **ACK**: ACK number valid
- **FIN**: sender has reached end of byte stream
- **RST**: reset connection
- **URG**: urgent pointer valid
- **PSH**: push received segments promptly to application

Connection Control

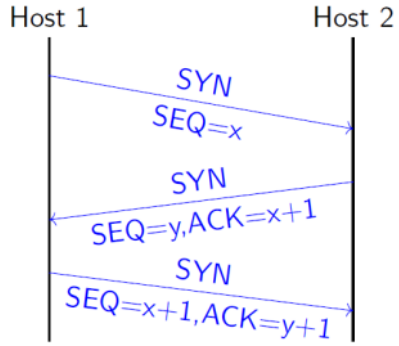
- ensure Sequence number is as expected
- hosts must synchronise Sequence numbers
- ISN - initial sequence number chosen randomly

- SYN - establish ISN

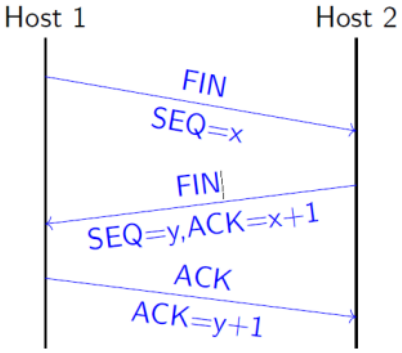
TCP Congestion:

- a network node is dealing with *more traffic than it can handle*
  - send as fast as the slowest link permits
  - packet loss mostly due to congestion
    - so detect congestion when packets start to drop
  - change data rate to adapt

- **TCP sender uses the smaller of**
  - receiver window - normal
  - **Congestion window**
    - grows and shrinks based on packet loss



Treat as two separate connections



Application Layer

10 April 2018 00:15

Names:

- memorable name corresponding to an IP address

Domain Name System - **DNS**

- distributed name look-up system

- must support independent admin of names
- avoid name conflicts - complete name uniqueness only
- naming hierarchies
- uses UDP

- Name resolution
  - local name server usually known by IP
  - local NS knows about you
  - host asks NS to resolve name

- Records
  - **A** - name to IP mapping
  - **MX** - mail server name
  - **NS** - name server name
  - **CNAME** - alias
  - **PTR** - IP to name mapping
  - RRSIG - DNSSEC Signature

- **13 root** name servers (A-M)
  - iterative only

- **iterative vs recursive query**
  - NS doesn't have to support recursive

- Cache answers
  - reply to query with previous obtained answer
  - non-autoreactive answer
  - TTL on entries - based on volatility
    - stable names - like root - have much longer TTL

- Troubleshooting on linux
  - **dig**
  - nslookup

World Wide Web

- client-server based application using the internet

WWW Standards:

- HyperText Transfer Protocol - **HTTP**
  - used by clients to get resources from servers
  - TCP port 80 on server
  - Methods
    - **GET** - client request resource form server
    - **POST** - send data to server resource
    - **HEAD** - request the header of a web page
    - others - PUT, DELETE, LINK, UNLINK
  - **Process**
    - client opens connection, then sends request
    - server responds, then connection is closed
  - **Request format**
    - Request line - method, URI, version
    - Header - additional info
    - Blank line
    - Body (data)
  - **Reply Format**
    - Status line - version, **CODE**, optional message
    - Header
    - Blank line
    - Body - data
  - **Stateless**
    - server doesn't keep track of request
  - HTTP/1.0 - **non-persistent connection**
    - new TCP connection for each request
  - **Cookies**
    - identify users
    - sent by server, stored by client
  - **HTTP/1.1** - persistent connections
    - one TCP connection
    - pipeline HTTP requests

- HyperText Mark-up Language - **HTML**
  - describe structure of web page

- Uniform Resource Identifier - **URI**
  - format - string of characters
  - ID for a resource
- Uniform Resource Locator - **URL**
  - **is a URI**
  - ID a server resource / location
  - **format**
    - scheme://user:password@host:port/path?query#fragment

Email:

- address - <username>@<domain name>
- mail server - **MX record via DNS**
- standard fields:
  - To, Cc, Sender, Reply-to (all are addresses)
  - From, Received, Subject
  - **Message-Id** - unique - helps to detect duplicates and responses

Mail User Agents - **MUA** - client

Mail Transfer Agents - **MTA** - mail server - handles transfers

Mail Delivery Agent - **MDA** - stores mail for user

Usually have MTA and MDA together

Simple Mail transfer Protocol - **SMTP**

- MUA and MTA communicate using TCP, port 25
- Start - HELO, MAIL FROM
- Exchange - RCPT TO, DATA
- End - QUIT
- Usually used to deliver mail to server, not to desktop

MIME - multipurpose internet mail extensions

- extend standard message format, including content types

HTTP Reply Codes

1xx	Informational
2xx	Successful operation: OK (200), Accepted (202), No response (204)
3xx	Redirection: Moved permanently (301), Not modified (304)
4xx	Client error: Bad request (e.g. syntax error) (400), Unauthorised (401), Payment required (402), Not found (404)
5xx	Server error: Service unavailable (503), HTTP version unsupported (505)

    - plain text
    - html text
    - image
    - mixed

Delivering Mail to Users

POP - Post Office Protocol

- client connects to server's port 110
- password requires
- **client can only download Inbox**

IMAP - Internet Message Access Protocol

- sync actions to server
- designed to handle multiple folders
- offline mode for when connection unavailable

POP vs IMAP