Unix/Linux provides *sockets* to communicate between processes that may be on different systems (i.e., across a network)

- e.g. web servers, networked databases, networked message queues, ...
Sockets

*Socket* = an end-point of a channel

- commonly used to construct client-server systems
- either locally (Unix domain) or network-wide (Internet domain)
- server creates a socket, then ...
  - binds to an address (local or network)
  - listens for connections from clients
- client creates a socket, then ...
  - connects to the server using known address
  - writes to server via socket (i.e. sends *requests*)
  - reads from server via socket (i.e. receives *responses*)
Sockets

```c
int socket(int Domain, int Type, int Protocol)
```

- requires `#include <sys/socket.h>`
- creates a socket, using ...
  - `Domain` ... communications domain
    - AF_LOCAL ... on the local host (Unix domain)
    - AF_INET ... over the network (Internet domain)
  - `Type` ... semantics of communication
    - SOCK_STREAM ... sequenced, reliable stream
    - SOCK_DGRAM ... connectionless, unreliable packet transfer
  - `Protocol` ... communication protocol
    - many exist (see `/etc/protocols`), e.g. IP, TCP, UDP, ...
- returns a socket descriptor (small `int`) or -1 on error
Sockets

```c
int bind(int sockfd, struct sockaddr *Addr, socklen_t addrLen)
```

- associates an open socket with an address
- for Unix Domain, address is a pathname in the file system
- for Internet Domain, address is IP address + port number

```c
int listen(int sockfd, int backlog)
```

- wait for connections on socket `sockfd`
- allow at most `backlog` connections to be queued up
Sockets

SockAddr = struct sockaddr_in

- sin_family ... domain: AF_UNIX or AF_INET
- sin_port ... port number: 80, 443, etc.
- sin_addr ... structure containing host address
- sin_zero[8] ... padding

Example:

```c
struct sockaddr_in web_server;
server = gethostbyname("www.cse.unsw.edu.au");
web_server.sin_family = AF_INET;
web_server.sin_addr.s_addr = server;
web_server.sin_port = htons(80);
```
Sockets

```c
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)
```

- `sockfd` has been created, bound and is listening
- blocks until a connection request is received
- sets up a connection between client/server after `connect()`
- places information about the requestor in `addr`
- returns a new socket descriptor, or -1 on error

```c
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
```

- connects the socket `sockfd` to address `addr`
- assumes that `addr` contains a process listening appropriately
- returns 0 on success, or -1 on error
Pseudo-code showing structure of a simple client program:

```c
s = socket(Domain, Type, Protocol)
serverAddr = {Family,HostName,Port}
connect(s, &serverAddr, Size)
write(s, Message, MsgLength)
read(s, Response, MaxLength)
close(s)
```

(See http://www.linuxhowtos.org/C_C++/socket.htm)
Pseudo-code showing structure of a server program:

```c
s = socket(Domain, Type, Protocol)
serverAddr = {Family, HostName, Port}
bind(s, serverAddr, Size)
listen(s, QueueLen)
while (1) {
    int ss = accept(s, &clientAddr, &Size)
    handleConnection(ss)
}
```
Popular software architecture:
Life of an IP Packet

Video available at
https://www.youtube.com/watch?v=9BGWrLiT9qs
Networks

\textit{Network} = interconnected collection of computers

Flavours of networks:

- \textit{local area networks} ... within an organisation/physical location
- \textit{wide area networks} ... geographically dispersed (WAN)
- \textit{Internet} ... global set of interconnected WANs

Why do we need networks?

- previously ... transfer data, send text-based emails
- nowadays ... communication, communication, communication
- sharing resources e.g. printers, large storage devices, ...
What are the basic requirements for a network?

- get data from machine A to machine B
- A and B may be separated by 100’s of networks and devices

How to achieve this?  

(using postal service analogy)
- need a unique address for destination
- identify a route  (first post office)
- process at intermediate nodes  (other post offices)
- follow certain protocols  (envelopes, stamps fees)
Overview of Network Communication

How a file is sent over the network:

- File data divided into *packets* by source device
  - packets are small fixed-size chunks of data, with headers
- Passed across *physical link* (wire, radio, optic fibre)
- Passing through multiple *nodes* (routers, switches)
  - each node decides where to send it next (for best route)
- Packets reach destination device
- Re-ordering, error-checking, buffering
- File received by receiving process/user
Components of the Internet ...

- millions of *connected devices*
  - e.g. PC, server, laptop, smartphone
  - *host* = end system, running network apps

- *communication links*
  - e.g. fibre, copper, radio, satellite
  - *bandwidth* = transmission rate

- *packet switches*
  - e.g. routers, network switches
  - compute next hop, forward packets
The Internet

Internet communications are based on a 5-layer ‘stack’:

- **Physical layer**: bits on (wires or fibre optics or radio)
- **Link layer**: ethernet, MAC addressing, CSMA etc.
- **Network layer**: routing protocols, IP
- **Transport layer**: process-process data transfer, TCP/UDP
- **Application layer**: DNS, HTTP, email, Skype, torrents, FTP etc.

Typical packet encapsulates data from all lower layers

Why so many layers (of abstraction)?

- each layer encapsulates one aspect of network transport
- provides layered *reference model* for discussion
- modularization eases maintenance/updating
  - e.g. changing implementation of one layer doesn’t affect other layers
The Internet

Path of data through network layers
Network protocols govern all communication activity on the network. Protocols provide communication rules:

- format and order of messages sent/received
- actions taken on message transmission/receipt

Protocols are defined in all of the layers, e.g.

- link layer: PPP (point-to-point protocol), ...
- network layer: IP (internet protocol), ...
- transport layer: TCP (transmission control), UDP (user datagram)
- application layer: HTTP, FTP, SSH, POP, SMTP, ...

Higher-level layers typically have a wider variety of protocols.
Network Application Layer

The application layer directly supports the apps we interact with, e.g.

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
Client-Server Architecture

*Client-server* = common way of structuring network communication

*Server* is a data provider

- process that waits for requests
- always-on host, with permanent IP address
- possibly using data centres / multiple CPUs for scaling

*Client* is a data consumer

- sends requests to server; collects response
- may be intermittently connected, may have dynamic IP address
- does not communicate directly with other clients

Peer-to-peer (P2P) systems run both client and server processes on each host
Client-Server Architecture

Client-server systems are frequently implemented with sockets ...
Server processes must have a unique Internet-wide address
- part of address is IP address of host machine
- other part of address is port number where server listens

Example: 128.119.245.12:80
- address of web server on gaia.cs.umass.edu

Some standard port numbers
- 22 ... ssh  (Secure Shell)
- 25 ... smtp  (Simple Mail Transfer Protocol)
- 53 ... dns   (Domain Name System)
- 80 ... http  (Web server)
- 389 ... ldap  (Lightweight Directory Access Protocol)
- 443 ... https (Web server (encrypted))
- 5432 ... PostgreSQL database server
IP Addresses

*IP Address* = unique identifier for host on network

- given as a 32-bit identifier (dotted quad), e.g. 129.94.242.20
- special case: 127.0.0.1 (loopback address referring to local host)
- IP addresses are assigned by
  - sys admin entering into local registry (for ‘permanent’ addresses)
  - dynamically, by getting a temporary address from DHCP server

Note: the world is running out of 32-bit IP addresses

- why? Internet of Things ... *every* networked device needs an IP
- IPv6 uses 128-bit addresses e.g.
- distinct addresses: IPv4 $4 \times 10^9$, IPv6 $3 \times 10^{38}$
Application-layer Protocols

Each application-layer protocol defines
- *types* of messages
  - different types of requests and responses
- message *syntax*
  - what fields are in messages; how fields are delineated
- message *semantics*
  - meaning of information in fields
- processing *rules*
  - when and how processes respond to messages

Protocols can be *open* (e.g. HTTP) or *proprietary* (e.g. Skype)
The HTTP Protocol

HTTP = HyperText Transfer Protocol
- an extremely important protocol (drives the Web)
- message types: URLs (requests) and Web pages (responses)
- message syntax: headers + data (see details later)

URLs are the primary type of request

Web pages are the primary type of response
- contain HTML; may contain references to other types of objects
- all web objects are addressable via a URL
Client-server model:

- **client** = *Web browser* (e.g. Chrome)
  - sends HTTP requests
  - receives HTTP responses
  - shows response as web page
- **server** = *Web server* (e.g. Apache)
  - receives HTTP requests
  - sends HTTP responses
Transport layer view of HTTP application layer

- using TCP
- client initiates TCP connection (socket) to server, port 80
- server accepts TCP connection from client
- client sends HTTP request messages (e.g. GET)
- server responds with HTTP messages (e.g. HTML)
- interaction completes, connection (socket) closed
HTTP request message

URL can also include a *query string*, e.g.

```
http://cse.unsw/course/view.php?c=COMP3231&s=verbose
```
HTTP request

First line of HTTP request contains (method, path, protocol), e.g.

```
GET /cs1521/17s2/index.html HTTP/1.1
```

- no need to mention host, since connection already established
- GET requests data from resource specified by path
  - query string is included in the path
- POST submits data to be processed by specified resource
  - query string is included in the body
- HEAD same as GET, but returns only header (no data)
HTTP response message

HTTP/1.1 200 OK
Date: Sun, 26 Sep 2010 20:09:20 GMT
Server: Apache/2.0.52 (CentOS)
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT
ETag: "17dc6-a5c-bf716880"
Accept-Ranges: bytes
Content-Length: 2652
Keep-Alive: timeout=10, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1

data data data data data data data ...

data, e.g., requested HTML file

status line (protocol, status code, status phrase)
header lines

status line (protocol, status code, status phrase)
The HTTP Protocol

Response status codes appear in first line of HTTP response

- 200 OK ... successful request
- 301 Moved Permanently ... requested object moved
  - returns new URL for client to use in future requests
- 400 Bad Request ... request cannot be processed
  - possible reasons: bad request syntax, request size too large, ...
- 403 Forbidden ... valid request cannot be processed
  - possible reasons: user does not have permission for operation
- 404 Not Found ... path does not exist on server
- 500 Internal Server Error ... server cannot complete request
  - possible reasons: server side script fails, database not accessible, ...
Network requests typically use server names, e.g.

- http://www.cse.unsw.edu.au/cs1521/17s2/

Setting up a TCP connection needs an IP address, not a name. The **Domain Name System** provides name→IP address mapping. Can access this on Unix/Linux via the `host` command, e.g.

```
$ host www.cse.unsw.edu.au
www.cse.unsw.edu.au has address 129.94.242.51
$ host a.b.c.com
Host a.b.c.com not found: 3(NXDOMAIN)
```

(assumes that you have a network connection)
In real life, often have one object referenced by many names, e.g.
  • a person: name, SSN, TFN, passport #, ...

On the Internet, each host has ...
  • one or more symbolic names, unique IP address
  • symbolic: www.cse.unsw.edu.au, IP: 129.94.242.51

Note:
  • a given IP address may be reachable via several names
  • a given name may map to several IPs (e.g. for load distribution)
Server Addresses (DNS)

*Domain Name System* (DNS)

- effectively a distributed database of name→IP mappings
- implemented across a hierarchy of *name servers*
- name servers cooperate to *resolve* names to IP addresses

This is an extremely important core function on the Internet

Why not centralise DNS?

- central point of failure, high traffic volume
- distant database (lag), maintenance of very large DB
Server Addresses (DNS)

Resolving www.amazon.com in this system

- contact a root DNS server to find .com DNS server
- contact .com DNS server to get amazon.com DNS server
- contact amazon.com DNS server to get IP of their web server
Two styles of name resolution

- *iterated query* ... work done by client
  - client contacts name server X
  - gets response “I don’t know, but ask name server Y”
  - OR gets response “Here is the IP address”
  - client repeats above steps until name resolved

- *recursive query* ... work done by name servers
  - client contacts name server X
  - X contacts name server Y, Y contacts Z, ...
  - query propagates until name resolved
How are the various DNS servers structured/managed?

- **top-level domain (TLD) name servers**
  - .com, .org, .edu and all country-level domains (e.g. .uk)
  - Network Solutions maintains servers for .com
  - AusRegistry maintains servers for .au

- **authoritative name servers**
  - maintains mappings from names to IP within an organisation
  - all hosts within the organisation are registered here

- **local (default) name servers**
  - maintains cache of name→IP mappings
  - starting point for DNS queries, forward to TLD server (if !cached)