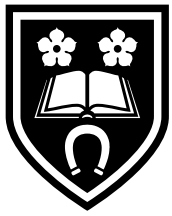


UNIVERSITY OF
LEICESTER

DEPARTMENT OF INFORMATICS

CO4219/CO7219 Internet and Cloud Computing

Lecture Notes

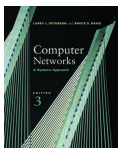
Internet and Cloud Computing

Part I – Networking

1. Foundations of Networking
2. Sliding Window, Ethernet and WiFi
3. Internetworking (IP, Routing)
4. End-to-end Protocols: TCP and UDP

Textbooks for Part I

- Main textbook:



Larry L. Peterson, Bruce S. Davie: Computer Networks – A Systems Approach, 3rd Edition, Morgan Kaufmann, 2003. ISBN 1-55860-833-8.

- Further reading:
  - Andrew S. Tanenbaum: Computer Networks, 4th Edition, Pearson, 2003, ISBN 0-13-038488-7.
  - William Stallings: Data and Computer Communications, 7th Edition, Pearson, 2004, ISBN 0-13-183311-1.

1. Foundations of Networking

Networks

- **Computer Networks:**
  - Built from general-purpose hardware
  - Not optimized for one specific application
  - Able to carry many different types of data and support a wide range of applications: e-mail, WWW, FTP, video-on-demand, electronic commerce, distributed computing, digital libraries, p2p file sharing, ...
- As opposed to special-purpose networks:
  - Voice telephone network
  - Cable network
  - Network connecting terminals to mainframe

1.1 Applications

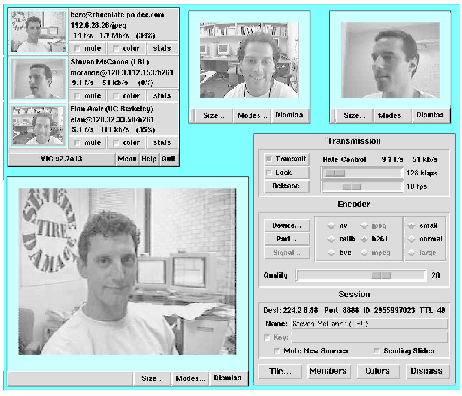
Application 1: World Wide Web

- When you enter "http://www.mkp.com/pd3e" into your browser, as many as 17 messages may be exchanged over the Internet:
  - Up to 6 messages to translate www.mkp.com into its IP address (129.35.69.7).
  - 3 messages to set up a TCP connection.
  - 4 messages to request and download the page.
  - 4 messages to tear down the TCP conncection.

Application 2: Audio/Video Streaming

- Instead of downloading the whole audio or video file, "stream" it over the Internet and play it as it arrives.
- Video-on-demand, video conferencing etc.
- For interactive applications such as video conferencing, it is important to have small delays.

Video Application: vic



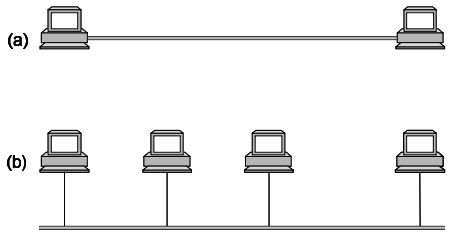
1.2 Requirements

- Application programmer: wants the network to provide the services needed by his or her application, e.g. reliable delivery of messages
- Network designer: wants a cost-effective design, e.g. efficient resource usage and fair bandwidth allocation to users
- Network provider: wants a network that is easy to administer and manage, e.g. easy isolation of faults and easy accounting for usage

1.2.1 Connectivity

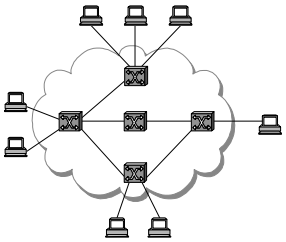
- Each computer (node) on the network should be able to communicate with every other computer (node).
- If the network is to support a large number of computers (e.g. the Internet), the network design must **scale** or **be scalable**.
- At the lowest level, we have physical **links** connecting a pair of nodes (point-to-point) or a larger set of nodes (multiple-access).

Types of Physical Links



Direct links: (a) point-to-point; (b) multiple-access.

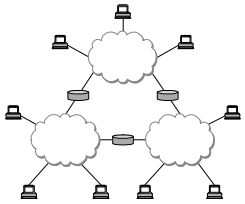
Switched Networks



- Packet switched: most computer networks
- Circuit switched: telephone network

Nodes inside "cloud" are **switches**, outside are **hosts**.

Internetworks



- Independent networks ("clouds") are interconnected to form an **internetwork**.
- The main example of an internetwork is the Internet.
- A node connected to several networks is called **router** or **gateway**.

Network Characterization by Size

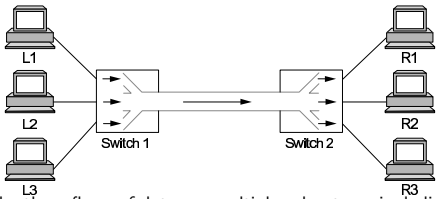
- LAN (local area network): less than 1km
- MAN (metropolitan area network): tens of kilometers
- WAN (wide area network): national or international

Addresses and Routing

- Each node on the network must have an address (a byte string that distinguishes it from all other nodes).
- When a source node wants to send a message to a destination node, it specifies the address of the destination node.
- The switches and routers of the network use the destination address to decide how to forward the message toward the destination. This process is called routing.
- Apart from unicast (sending to one node), there is also multicast (sending to a set of nodes) and broadcast (sending to all nodes of a network).

1.2.2 Cost-Effective Resource Sharing

All nodes must share the network resources. Example:



- The three flows of data are multiplexed onto a single link by Switch 1.
- They are demultiplexed into separate flows by Switch 2.

Types of Multiplexing

- STDM (synchronous time-division multiplexing): divide time into equal-sized quanta, assign quanta to flows in round-robin fashion
- FDM (frequency-division multiplexing): transmit different flows at different frequencies

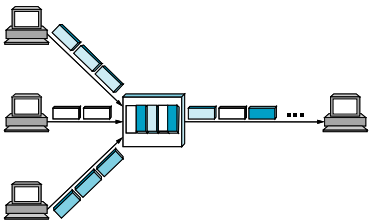
Both are not ideal if the number of flows changes or if flows are idle. Computer networks mostly use:

- Statistical Multiplexing
  - Share link over time as in STDM, but transmit data from each flow "on demand" rather than during a predetermined time slot.

Packets

- In statistical multiplexing, the size of data blocks that can be transmitted at one time is usually limited.
- This limited-size data block is referred to as packet.
- A message can be of arbitrary size and is broken into packets for transmission (fragmentation) and reassembled at the receiver.

Packet Multiplexing Example

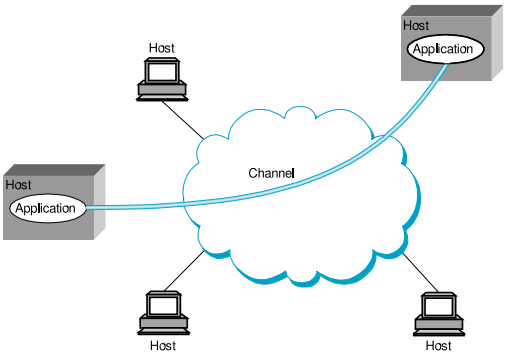


- Switch must decide in which order to service packets, e.g. FIFO (first-in first-out) or round-robin.
- If more packets arrive than can be forwarded by the switch, they are stored in a buffer or, if the switch runs out of buffer space, discarded (congestion).

1.2.3 Support for Common Services

- A network does not only deliver packets; it must provide means for application programs running on the hosts connected to the network to communicate in a meaningful way.
- Many application programs need similar services; the network designer must identify the right set of common services so that they can be implemented once, rather than in each application program separately.
- Intuitively, we want to have **channels** over which application-level processes can communicate with each other.

Abstract Channel between Processes



Channel Functionality

What functionality should channels provide?

- Do all messages need to be delivered, or is it acceptable if some fail to arrive?
- Must all messages arrive in the same order in which they are sent?
- Does the network need to ensure that no third parties are able to eavesdrop on the channel?

Examples of basic types of channels:

- Request/reply channel: client sends request, server answers with reply; e.g. FTP, digital library, WWW
- Message stream channel: continuous stream of data, e.g. videoconferencing or video-on demand

Where to Implement It

- Reoccurring issue in network design: **Where** should functionality be implemented?
  - Network provides bit pipe, all high-level functionality is implemented at end hosts.

    versus
  - Push functionality onto the switches, allow end hosts to be "dumb" devices (e.g., telephone handsets).

Reliability

- Reliable message delivery is highly desirable.
- Possible causes for failures:
  - Machine crashes, fibre cuts
  - Transmission errors due to interference
  - Buffer overflow at switches
  - Software bugs
- Main classes of failures:
  - Bit errors (or, if consecutive bits are affected, burst errors): affect one out of 10^6 to 10^7 bits on copper-based cable, and one out of 10^{12} to 10^{13} bits on optical fibre.
  - Packet loss
  - Node or link failure

1.3 Network Architecture

- A network architecture is a general blueprint that guides the design and implementation of networks.
- Two widely referenced architectures:
  - OSI architecture
  - Internet architecture
- Identify abstractions that provide a useful service and can be efficiently implemented in the underlying system.
- Abstraction naturally leads to **layering**.

1.3.1 Layering and Protocols

- Layering: Start with the services provided by the underlying hardware, and then add a sequence of layers, each providing a higher level of service.
- Services provided at higher layer are implemented in terms of services provided by lower layers.
- Simple example:

Application programs
Process-to-process channels
Host-to-host connectivity
Hardware

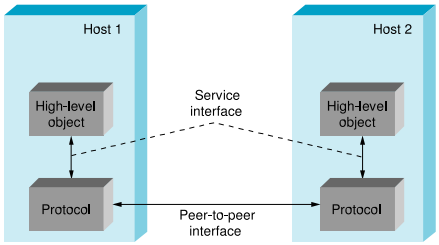
Alternative Abstractions

- Slightly more general example:

Application programs	
Request/reply channel	Message stream channel
Host-to-host connectivity	
Hardware	

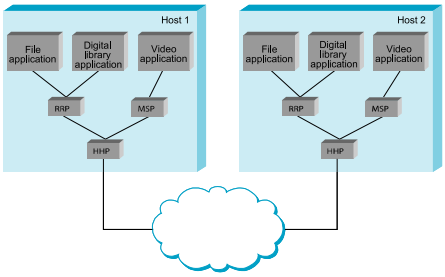
- The abstract objects that make up the layers are called **protocols**.
- Each protocol defines two interfaces:
  - **Service interface** to objects on same node
  - **Peer interface** (or peer-to-peer interface) to its counterpart (peer) on another node

Protocol Interfaces



- At the hardware level, the peers communicate directly.
- Higher-level protocols communicate with their peers by passing messages to some lower-level protocol.

Protocol Graph

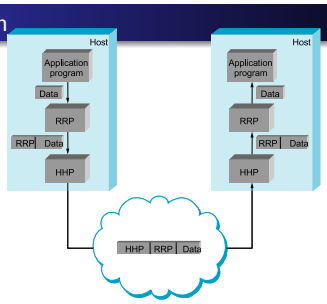


- In this example: HHP = host-to-host protocol, RRP = request/reply protocol, MSP = message stream protocol
- We say, e.g., that the file application uses the services of the **protocol stack** RRP/HHP.

Specification and Implementation

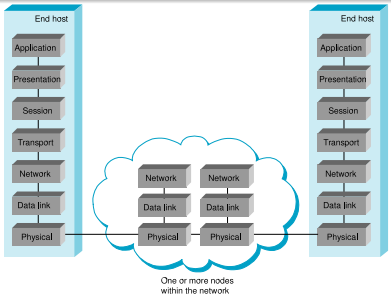
- The term "protocol" is used in two different ways:
  - It can refer to the abstract interfaces (service interface and peer interface), the **protocol specification**.
  - It can refer to the module that implements these two interfaces.
- Two different implementations that accurately implement a protocol specification are said to **interoperate**.

Encapsulation



- Each layer encapsulates message it gets from higher layer by adding its **header** to the **body** (**payload**).
- At destination, message is processed in opposite order (using a **demux key** to select the right higher-level protocol or application, if necessary).

Open Systems Interconnection (OSI)



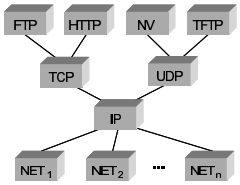
- OSI architecture: reference model defined by ISO (International Standards Organization).
- Seven layers, one or more protocols at each layer.

OSI Layers

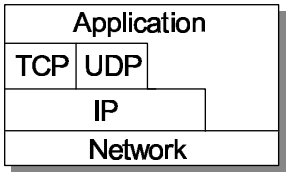
- Layer 1 (**physical**): transmission of raw bits over a link
- Layer 2 (**data link**): collects streams of bits into a larger aggregate called **frame**; typically implemented by network adaptors and device drivers
- Layer 3 (**network**): handles routing of **packets** among the nodes
- Layer 4 (**transport**): implements "process-to-process" channel, unit of data exchanged is called **message**
- Layer 5 (**session**): provides name space to tie together different transport streams of a single application
- Layer 6 (**presentation**): format of data to be exchanged (bit order, video encoding, etc.)
- Layer 7 (**application**): FTP, HTTP, etc.

Internet Architecture

Protocol graph:



Alternative view:



- Internet architecture (or: TCP/IP architecture) has evolved from ARPANET.
- Can be fitted into 7-layer OSI model, but is usually viewed as 4-layer model.
- IETF (Internet Engineering Task Force)

Internet Architecture Layers

- Lowest layer (Network): variety of network protocols, implemented by combination of hardware and software (e.g. Ethernet, FDDI, ATM, etc.)
- Second layer (IP): Internet Protocol, a single protocol that supports the interconnection of multiple networking technologies into a single, logical internetwork
- Third layer: Two main protocols (end-to-end protocols) that provide alternative logical channels:
  - TCP (Transmission Control Protocol): reliable byte-stream
  - UDP (User Datagram Protocol): unreliable datagram delivery
- Top layer: Range of application protocols such as HTTP, FTP, Telnet, SMTP (Simple Mail Transfer Protocol), etc.

Internet Architecture Features

- **No strict layering**: Application is free to bypass layers and to directly use IP or even the underlying networks
- Central philosophy: **IP is the focal point** of the architecture (cf. hourglass shape of protocol graph); issue of message delivery is completely separated from process-to-process communication service.
- IETF culture: In order for someone to propose a new protocol to be included in the architecture, they must produce both a protocol specification and **at least one representative implementation**.

"We reject kings, presidents, and voting. We believe in rough consensus and running code." (Dave Clark)

1.4 Implementing Network Software

- Phenomenal success of the Internet: number of computers connected to the Internet has been doubling every year since 1981; total Internet traffic surpassed voice phone system in 2001.
- Contributing factor: much of the Internet functionality is provided by software running on general-purpose hardware.
- New functionality can be added with "just a small matter of programming."
- New applications and services (e-commerce, videoconferencing, IP telephony, etc) have been showing up at a phenomenal pace.

1.4.1 API (Sockets)

- The interface exported by the network is called **application programming interface (API)** and is typically provided by the operating system (OS) on a computer.
- The API provides the syntax by which the services of the protocols can be invoked.
- The **socket interface**, originally provided by Berkeley Unix, is now supported by virtually all OSs.
- Main abstraction: the socket. Think of it as the point where a local application process attaches to the network.

Socket Operations

- create a socket
- attach a socket to the network
- send messages through a socket
- receive messages through a socket
- close socket

Socket interface in C/C++:

- int socket(int domain, int type, int protocol)

  domain: PF_INET, PF_UNIX, PF_PACKET
  type: SOCK_STREAM, SOCK_DGRAM
  protocol: UNSPEC for TCP and UDP
  ➟ returns a handle for the socket created

Passive Open (Server)

- int bind(int socket, struct sockaddr *address, int addr_len)
  binds socket to the specified address (IP address and port number) on local machine
- int listen(int socket, int backlog)
  specifies how many connections can be pending on that socket
- int accept(int socket, struct sockaddr *address, int *addr_len)
  blocks until a remote participant has established a connection, then returns a **new** socket for that connection (address of remote participant is stored in address)

Active Open (Client)

- int connect(int socket, struct sockaddr *address, int addr_len)
  establishes connection to the remote participant specified by address

Once the connection is established:

- int send(int socket, char *message, int msg_len, int flags)
  sends message over specified socket
- int recv(int socket, char *buffer, int buf_len, int flags)
  receives message from specified socket into buffer

Sockets in Java

- Networking-related classes are available in the package java.net.
- Classes Socket and ServerSocket implement TCP sockets (client socket and server socket, respectively).
- Class DatagramSocket implements UDP socket.
- Classes InetAddress and InetSockAddress implement IP addresses and socket addresses (IP address + port number), respectively.

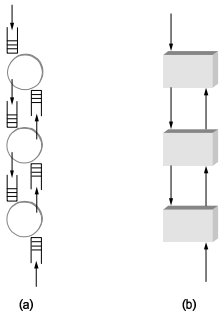
1.4.2 Example Application

- Simple talk application
- Server accepts connections on port 5432 and displays all strings received.
- Client connects to server and sends strings typed in by the user.
- Only one connection between server and client at any time.
- C source files: client.c and server.c
- Java source files: Client.java and Server.java

1.4.3 Protocol Implementation Issues

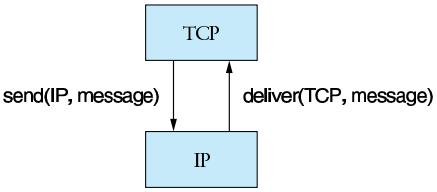
- Applications access TCP in a similar way as TCP accesses IP.
- However, different layers of the protocol stack do not use the socket interface to call each other.
- Protocol-to-protocol interfaces are optimised for efficiency.

Process Model



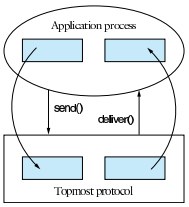
- Messages are passed through several layers of the protocol stack.
- Two choices for assigning message processing to processes (threads):
  - (a) process-per-protocol: requires many context switches
  - (b) process-per-message: uses procedure calls, usually more efficient

Receive vs. Deliver



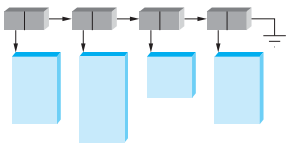
- Incoming messages are handed from lower layers to higher layers.
- Inefficient to use receive operation initiated by higher layer.
- Typically a protocol uses an **upcall** to deliver a received message to a high-level protocol.

Message Buffers



- Second inefficiency of socket interface: message must be copied from application's buffer to network buffer, and vice versa.
- Copying of messages is very time-consuming. (Note: Processor speed is growing much faster than memory speed.)

Copy-Free Message Abstraction



- Most network subsystems define an abstract data type for messages that is shared by all protocols.
- This allows to pass messages up and down the protocol graph without copying.
- Copy-free manipulation of messages (adding and stripping header, etc.).
- Implementation usually involves list of pointers to message buffers.

1.5 Performance

- Computer networks are expected to perform well.
- In networking, "design for performance" is often better than "first get it right and then make it fast."
- **Bandwidth and throughput**: confusing terms
  - Literally, bandwidth is the width of a frequency band, e.g. a telephone line supporting 300 Hz to 3,300 Hz has bandwidth 3,000 Hz.
  - Commonly, bandwidth refers to the number of **bits per second** that can be transmitted over a link (e.g. 10 Mbps Ethernet).
  - **Throughput** refers to the **measured performance**, e.g. an application may only be able to transmit 2 Mbps over a 10 Mbps link.

1.5.1 Bandwidth and Latency

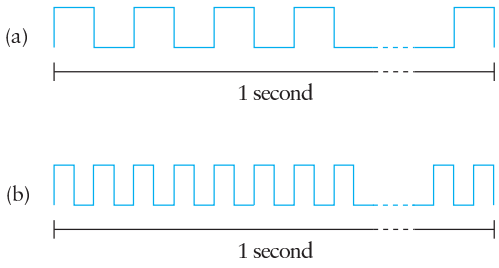
Bandwidth (throughput):

- Number of bits that can be transmitted over the network or link in a certain time.
- Example: 10 Mbps = 10 million bits per second
- It takes 0.1 μs to transmit one bit over a 10 Mbps link.

Latency (delay):

- Corresponds to how long it takes a message to travel from one end of the network or link to the other
- **Round-trip time (RTT)**: time it takes to send a bit from one node to another and back.
- Typical RTT: 1 ms in LAN, 100 ms cross-country (USA)

The "Width" of a Bit



- (a) 1 Mbps link: each bit is 1 μs wide
- (b) 2 Mbps link: each bit is 0.5 μs wide

Latency Calculation

- Latency = Propagation + Transmit + Queue
- Propagation = Distance / SpeedOfLight
- Transmit = MessageSize / Bandwidth

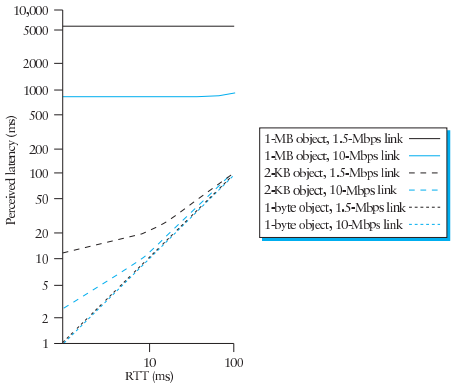
Speed of Light:

- In vacuum: 3.0×10^8 m/s
- In cable: 2.3×10^8 m/s
- In fibre: 2.0×10^8 m/s

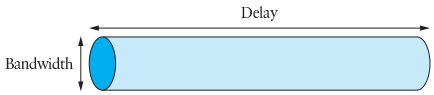
Examples

- Transmitting 1 byte to a server and receiving a 1 byte response:
  - Transmit time: 8 μs on 1 Mbps link
    0.08 μs on 100 Mbps link
  - ➠ RTT is important, bandwidth does not matter
- Downloading a 25 MB image file from a digital library
  - Transmit time: 20 seconds on 10 Mbps link
  - ➠ RTT is irrelevant, bandwidth matters
- **Remark**: Kbps and Mbps usually refer to 10^3 bps and 10^6 bps, while KB and MB usually mean $2^{10} = 1{,}024$ bytes and $2^{20} = 1{,}048{,}576$ bytes, respectively.

Illustration: Bandwidth and Latency



1.5.2 Delay × Bandwidth Product



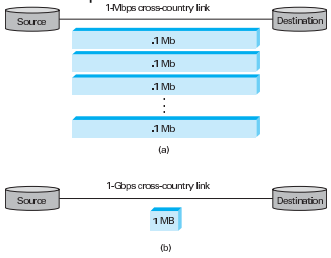
- Delay × bandwidth represents the number of bits that are "in flight" on a link (volume of the "pipe").
- For example, a 45 Mbps link with 50 ms one-way delay holds

$$50 \times 10^{-3} \text{ seconds} \times 45 \times 10^6 \text{ bits/second}$$

$= 2.25 \times 10^6$ bits, or about 275 KB of data.
If the sender transmits at full rate, 45 Mbps × 100 ms = 550 KB of data have been transmitted by the time the first response from the receiver arrives at the sender!

1.5.3 High-Speed Networks

- Bandwidth is ever increasing, but latency remains essentially the same.
- Transmitting a 1 MB file corresponds to (a) 80 pipes full of data over a 1 Mbps cross-country link, but (b) only 1/12 of one pipe on a 1 Gbps link.



Effective Throughput

- Effective end-to-end throughput:

$$\text{Throughput} = \frac{\text{TransferSize}}{\text{TransferTime}}$$

where: $\text{TransferTime} = \frac{1}{2}\text{RTT} + \frac{\text{TransferSize}}{\text{Bandwidth}}$

Here, $\frac{1}{2}$RTT represents the propagation delay of the link.

- Sending a 1 MB file over a 1 Gbps network with an RTT of 200 ms requires a transfer time of

100 ms + 1 MB / 1 Gbps = 108 ms.

Effective throughput: 1 MB / 108 ms = 74.1 Mbps.

1.5.4 Application Performance Needs

- The needs of applications are not always as simple as wanting "as much bandwidth as the network can provide" or "as little delay as possible".
- For example, a video stream with 352×240 pixels, 24 bits per pixel, and 30 frames per second would require

$$352 \times 240 \times 24 \times 30 \text{ bps} \approx 61 \text{ Mbps}$$

  throughput. Additional bandwidth is of no interest.
- In practice, video streams are compressed and have a lower average transmission rate, but occasional bursts.

Jitter

- **Jitter** is the variation in latency from packet to packet.
- If video stream source transmits one packet every 33 ms (i.e. 30 packets per second), then it is desirable that the packets arrive at the same rate at the receiver.
- If packets incur different delays, the interpacket gaps are variable at the receiver:

- Jitter is usually caused by queueing delays at intermediate nodes.
- Limited jitter can be handled by buffering (e.g. delaying the playback of a video stream).

Client.java

```
import java.io.*;
import java.net.*;

public class Client {
        public static void main(String[] args) throws IOException {
                Socket csocket = null;
                PrintWriter out = null;

                if (args.length!=1) {
                        System.err.println("usage: java Client hostname");
                        System.exit(1);
                }

                String HostName = args[0];

                try {
                        csocket = new Socket(HostName,5432);
                        out = new PrintWriter(csocket.getOutputStream(), true);
                } catch (UnknownHostException e) {
                        System.err.println("Don't know about host: "+HostName+".");
                        System.exit(1);
                } catch (IOException e) {
                        System.err.println("Couldn't get I/O for the "+
                                        "connection to server "+HostName+".");
                        System.exit(1);
                }

                BufferedReader stdIn =
                        new BufferedReader(new InputStreamReader(System.in));
                String nextline;
                while ((nextline = stdIn.readLine()) != null) {
                        out.println(nextline);
                }
                out.close();
                csocket.close();
                stdIn.close();
        }
}
```

Server.java

```
import java.net.*;
import java.io.*;

public class Server {
        public static void main(String[] args) throws IOException {
                ServerSocket serverSocket = null;

                try {
                        serverSocket = new ServerSocket(5432);
                } catch (IOException e) {
                        System.err.println("Could not listen on port: 5432.");
                        System.exit(1);
                }


                for (;;) {
                Socket clientSocket = null;
                try {
                        clientSocket = serverSocket.accept();
                } catch (IOException e) {
                        System.err.println("Accept failed.");
                        System.exit(1);
                }
                BufferedReader in = new BufferedReader(new InputStreamReader(cli
entSocket.getInputStream()));
                String nextline;
                while ((nextline = in.readLine())!=null) {
                        System.out.println(nextline);
                }
                in.close();
                clientSocket.close();
                }
        }
}
```

client.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 5432
#define MAX_LINE 256

int
main(int argc, char * argv[])
{
  struct hostent *hp;
  struct sockaddr_in sin;
  char *host;
  char buf[MAX_LINE];
  int s;
  int len;

  if (argc==2) {
    host = argv[1];
  }
  else {
    fprintf(stderr, "usage: simplex-talk host\n");
    exit(1);
  }

  /* translate host name into peer's IP address */
  hp = gethostbyname(host);
  if (!hp) {
    fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
    exit(1);
  }

  /* build address data structure */
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
  bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
  sin.sin_port = htons(SERVER_PORT);

  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
  }
  if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    perror("simplex-talk: connect");
    close(s);
    exit(1);
  }
  /* main loop: get and send lines of text */
  while (fgets(buf, sizeof(buf), stdin)) {
    buf[MAX_LINE-1] = '\0';
    len = strlen(buf) + 1;
    send(s, buf, len, 0);
  }

  return 0;
}
```

server.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT  5432
#define MAX_PENDING  5
#define MAX_LINE     256

int
main()
{
  struct sockaddr_in sin;
  char buf[MAX_LINE];
  int len;
  int s, new_s;

  /* build address data structure */
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
  sin.sin_addr.s_addr = INADDR_ANY;
  sin.sin_port = htons(SERVER_PORT);

  /* setup passive open */
  if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    perror("simplex-talk: socket");
    exit(1);
  }
  if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0) {
    perror("simplex-talk: bind");
    exit(1);
  }
  listen(s, MAX_PENDING);

  /* wait for connection, then receive and print text */
  while(1) {
    if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0) {
      perror("simplex-talk: accept");
      exit(1);
    }
    while ((len = recv(new_s, buf, sizeof(buf), 0)))
      fputs(buf, stdout);
    close(new_s);
  }
}
```

CO7219:
Internet and Cloud Computing

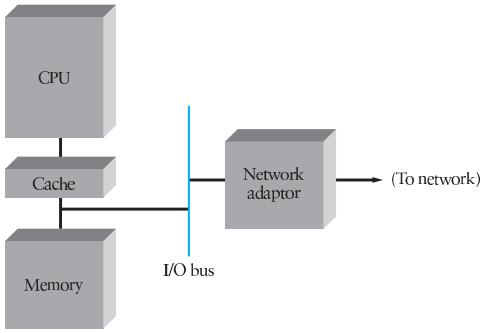
2. Sliding Window, Ethernet and WiFi

Direct Link Issues

- Simplest network: Two or more hosts directly connected by some physical medium.
- Five main issues:
  - **Encoding** bits so that they can be understood by a receiver.
  - **Framing**: delineating the sequence of bits into complete messages that can be delivered to the end node.
  - **Error detection**
  - Make the link appear **reliable** in spite of occasional corruption of frames.
  - **Media access control**: coordinate access to the link among multiple hosts

Hardware Building Blocks

- Typical node: general-purpose computer



Reliable Transmission

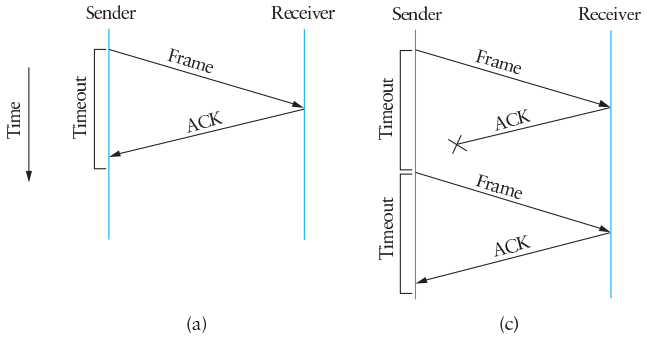
- Frames are sometimes corrupted (and this is detected using CRC or other error-detecting codes).
- Even when error-correcting codes are used, some errors are too severe to be corrected.
- A link-level protocol that wants to deliver frames reliably must somehow recover from lost frames.
- Two fundamental mechanisms:
  - **Acknowledgment** (ACK): Receiver confirms that a frame has arrived. The ACK is either a control frame (only header) or piggybacked on a data frame.
  - **Timeout**: If no ACK is received within a certain time, the original frame is retransmitted.

Approach is called **automatic repeat request** (ARQ).

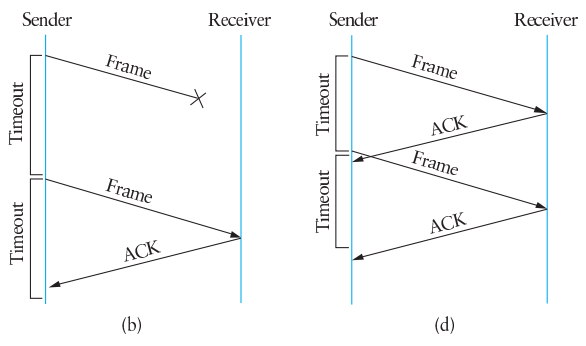
Stop-and-Wait

- Stop-and-wait is the simplest ARQ scheme.
- After transmitting one frame, the sender waits for an acknowledgment before transmitting the next frame.
- If the acknowledgment does not arrive after a certain time, the sender times out and retransmits the original frame.

Stop-and-Wait Scenarios

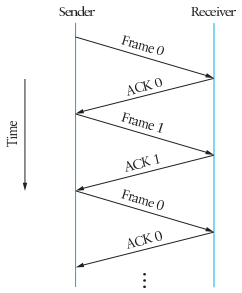


Stop-and-Wait Scenarios II



Sequence Numbers

- In scenarios (c) and (d), the receiver gets two copies of the same frame.
- In order to recognize such duplicates, 1-bit sequence numbers are used:



Shortcoming of Stop-and-Wait

- Stop-and-wait allows the sender to have only one outstanding frame on the link at any time.
- For example, on a 1.5-Mbps link with RTT 45 ms, a frame size of 1 KB implies a maximum sending rate of

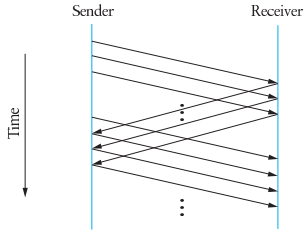
$$\frac{\text{BitsPerFrame}}{\text{TimePerFrame}} = \frac{1024 \times 8 \text{ bits}}{0.045\text{s}} \approx 182\text{Kbps}$$

This is only one-eighth of the link capacity!

Note that the delay $\times$ bandwidth product is 45 ms $\times$ 1.5 Mbps $=$ 67500 bits $\approx$ 8 KB for this link.

Sliding Window

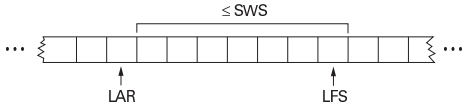
- Consider again the link with delay $\times$ bandwidth product of 8 KB.
- We would like to "keep the pipe full" and send the ninth frame roughly when the first acknowledgment arrives.



Sliding Window Algorithm

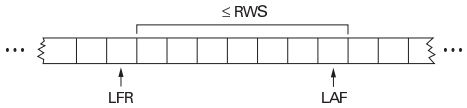
- Assign sequence number SeqNum to each frame.
- Sender maintains:
  - SWS (send window size): upper bound on number of outstanding (unacknowledged) frames that the sender can transmit
  - LAR (last acknowledgment received): sequence number of the last frame for which an ACK was received
  - LFS (last frame sent): largest sequence number of a frame that has been sent.
- Invariant: LFS – LAR $\leq$ SWS

Sliding Window on Sender



- When an ACK arrives, LAR moves to the right and the sender can send another frame.
- Sender associates a timer with each frame it transmits, and a frame is retransmitted if the timer expires before an ACK is received.
- Sender must be willing to buffer up to SWS frames, as they may need to be retransmitted.

Sliding Window on Receiver



- Receiver maintains:
  - RWS (receive window size): upper bound on number of out-of-order frames the receiver is willing to accept.
  - LAF (largest acceptable frame)
  - LFR (last frame received)
- Invariant: LAF – LFR $\leq$ RWS

Receiver Behaviour

- When a frame with sequence number SeqNum arrives:
  - If SeqNum $\leq$ LFR or SeqNum $>$ LAF, discard frame.
  - If LFR $<$ SeqNum $\leq$ LAF, the frame is accepted.

  Let SeqNumToAck be the largest sequence number that has the property that all frames with sequence number up to SeqNumToAck have been received.

  If frame SeqNumToAck has not yet been acknowledged, the receiver sends an ACK for sequence number SeqNumToAck (even if higher-numbered frames have been received) and sets LFR $=$ SeqNumToAck and LAF $=$ LFR $+$ RWS.
- The ACKs are called **cumulative** here, as an ACK for frame SeqNum acknowledges also all earlier frames.

Sliding Window Example

- Assume LFR $=$ 5 (i.e. last ACK was sent for SeqNum 5) and RWS $=$ 4, thus LAF $=$ 9.
- The receiver accepts frames with SeqNum 6 to 9.
- If frames 7 and 8 arrive, the receiver stores them, but does not need to send an ACK. (Technically, it may also resend an ACK for frame 5.)
- If, after that, frame 6 arrives, the receiver stores it and sends an ACK for SeqNumToAck $=$ 8. LAF increases to 12.
- In this scenario, frame 6 was probably lost during the first transmission and resent after the timeout.

Remarks on Sliding Window

- When a frame is lost and a timeout occurs, the amount of data in transit decreases, as the sender can advance its window only when the lost frame is acknowledged.
- Variations of sliding window:
  - Can use **duplicate acknowledgments** to detect frame losses.
  - **Negative Acknowledgment** (NACK): When frame 5 and then frame 7 arrives, receiver sends a NACK for frame 6.
  - **Selective Acknowledgment**: Receiver acknowledges exactly those frames it has received, not just the highest-numbered frame received in order.

Setting the Window Size

- The send window size determines the maximum number of outstanding frames on the link.
- SWS can be computed from the delay $\times$ bandwidth product of the link (if known!).
- RWS can be anything between 1 and SWS.
  - RWS = 1: Receiver does not buffer any out-of-order frames.
  - RWS = SWS: Receiver can buffer **any** frame that sender transmits.

Finite Sequence Numbers

- In practice, sequence numbers cannot grow infinitely large.
- Instead, they are stored in a header field of some finite size.
- For example, a 3-bit header field for sequence numbers allows only numbers 0,1,...,7.
- Sequence numbers must be reused: they "wrap around."
- Receiver must be able to distinguish between different incarnations of the same sequence number.
- In stop-and-wait, only one frame could be outstanding at any time, and two different sequence numbers (0, 1) were enough.

How Many Sequence Numbers?

- Assume that sequence numbers range from 0 to MaxSeqNum $-$ 1.
- If RWS $=$ 1, then MaxSeqNum $\geq$ SWS+1 is sufficient.
- If RWS $=$ SWS, we need MaxSeqNum $\geq 2 \cdot$ SWS.
  - Consider SWS $=$ RWS $=$ 7 and sequence numbers 0,...,7. This does **not** work:
    - Sender transmits 0,...,6
    - Receiver acknowledges, but ACKs are lost
    - Sender retransmits 0,...,6, but receiver expects frames 7,0,1,2,3,4,5, so the retransmitted frames will be mistaken for new frames.
- All this assumes that frames are not reordered in transit!

Roles of Sliding Window Protocol

- It delivers frames reliably across an unreliable link.
- It can preserve the order in which frames are transmitted, so that they are delivered to the higher layer protocols at the receiving end in the same order as they were sent.
- It can support **flow control**: a feedback mechanism by which the receiver (or the network) is able to throttle the sender (by adapting the send window size).

Some of these roles are more relevant for end-to-end connections than for individual links.

Therefore, the sliding window algorithm is also used in some end-to-end protocols.

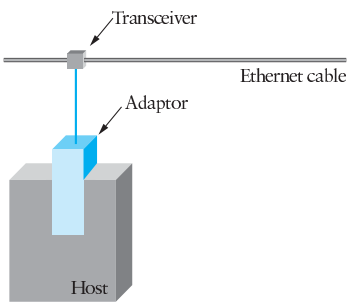
Concurrent Logical Channels

- Simple alternative to sliding window algorithm used in ARPANET.
- Keep the pipe full while still using the simple stop-and-wait algorithm.
- Idea: Multiplex several logical channels onto a single point-to-point link, and run stop-and-wait on each of these channels.
- When a node has a frame to send, it uses the lowest idle channel.
- Note: Reordering of frames can happen.
- In ARPANET: 8 logical channels over ground links, 16 logical channels over satellite links.

Ethernet (802.3)

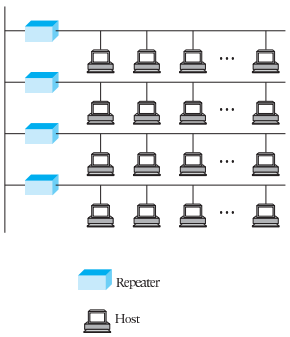
- Most successful local area networking technology of the last 20 years.
- Developed in the mid-1970s at Xerox Palo Alto Research Center (PARC).
- Based on **Carrier Sense Multiple Access with Collision Detect (CSMA/CD)** approach.
  - Carrier Sense: Nodes can distinguish between idle and busy link.
  - Collision Detect: Node listens as it transmits and can detect when a frame it is transmitting has interfered with another transmission.
- 10 Mbps Ethernet standard (Xerox, DEC, Intel) in 1978, basis of IEEE 802.3 (also: 100 Mbps – Fast Ethernet, 1000 Mbps – Gigabit Ethernet).

Transceiver and Adaptor



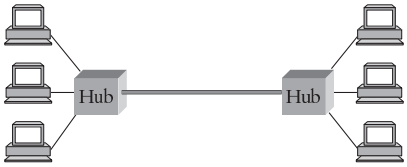
- Ethernet segment: up to 500 m in length (coaxial cable), taps at least 2.5 m apart.

Ethernet Repeater



- Repeaters forward the signal on all outgoing segments.

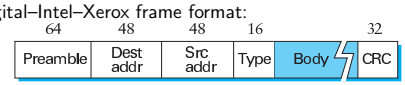
Ethernet Hub



- With 10BaseT cable (twisted pair, typically Category 5), it is common to have several point-to-point segments coming out of a multiway repeater (called **hub**).
- Even if an Ethernet consists of several segments connected by repeaters or hubs, data transmitted by any one host reaches **all other hosts**.

Ethernet Frame Format

- Digital–Intel–Xerox frame format:



  - 48-bit addresses (MAC addresses, MAC = media access control), usually burnt into ROM of adaptor.
  - **Type** field serves as demultiplexing key.
  - **Body**: 46 to 1500 bytes of data.
- Minor difference in IEEE 802.3: Length instead of Type, Type at beginning of Body.
- An adaptor receives frames addressed to it (**unicast**), to the **broadcast address** (all 1s), or to a **multicast address** that it has been instructed to listen to.
- In **promiscuous mode**, it receives all frames.

Transmitter Algorithm

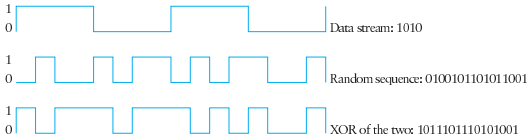
- If the adaptor has a frame to send:
  - If the line is idle, send immediately.
  - Otherwise, wait until current transmission finishes, and then transmit immediately.
- When the sender detects a **collision** (someone else was sending at the same time), it sends a **jamming sequence** (32 bits) and stops transmitting.
- In Ethernet, the maximum latency is at most 51.2μs and the minimum frame length is 512 bits, so every collision is detected. (It takes 51.2μs to transmit 512 bits over a 10 Mbps link.)

Exponential Backoff

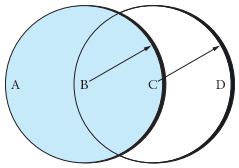
- After a collision, the sender repeatedly waits a certain amount of time and tries again:
  - After n collisions, it waits for $k \times 51.2\mu s$, where $k \in \{0, 1, \ldots, 2^n - 1\}$ is chosen **randomly**.
- This means:
  - After 1st collision, wait $k \times 51.2\mu s$ for $k \in \{0, 1\}$.
  - After 2nd collision, wait $k \times 51.2\mu s$ for $k \in \{0, 1, 2, 3\}$.
  - After 3rd collision, wait $k \times 51.2\mu s$ for $k \in \{0, \ldots, 7\}$.
  - After 4th collision, wait $k \times 51.2\mu s$ for $k \in \{0, \ldots, 15\}$.
  - . . . and so on
- Adaptors typically retry up to 16 times (but cap n at 10).
- Ethernet works best under lightly loaded conditions (utilization below 30%).

Wireless (802.11)

- IEEE 802.11 Wireless LAN
- Designed for limited geographical area (homes, office buildings, campuses).
- Radio-based versions run at 11 Mbps or 54 Mbps.
- Spread spectrum radio: signals are spread over a wider frequency band than normal, so as to minimise the impact of interference from other devices:



Collisions

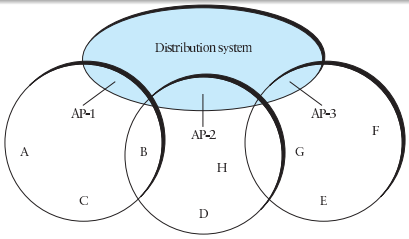


- Collision detection as in Ethernet is not always possible.
- If C and A send to B at the same time, the collision cannot be detected by C or A (**hidden node** problem).
- Simultaneous transmission from B to A and from C to D is possible, but when C hears B's transmission, it may conclude that it should not send (**exposed node** problem).

Collision Avoidance

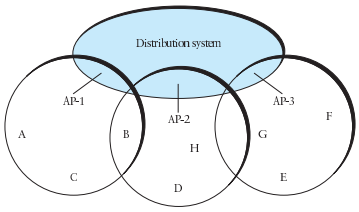
- **MACA**: Multiple Access with Collision Avoidance
  - When A wants to send a data packet to B, A sends an RTS (request to send) control frame that includes the length of the data packet.
  - When B receives RTS, it responds with a CTS (clear to send) control frame (with same length field).
  - When A receives the CTS, it sends the data packet.
  - Nodes that hear the CTS do not send during the packet transmission.
  - Nodes that hear the RTS but not the CTS are free to transmit.
  - If two RTS frames collide, the sender does not receive a CTS and waits a random time before trying again (exponential backoff).

Distribution System



- In practice, most IEEE 802.11 networks use **access points** (AP) that are connected by a **distribution system** (layer 2, e.g. Ethernet).
- Instead of nodes talking directly to each other, each node associates itself with an access point and sends/receives frames only to/from that AP.

Example

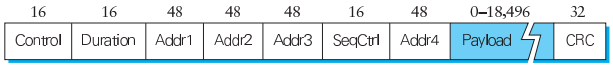


- If A wants to communicate with E:
  - A sends a frame to its access point AP-1.
  - AP-1 forwards the frame to AP-3 via the distribution system.
  - AP-3 sends the frame to E.

Selecting an AP: Scanning

- When a node joins the network or is unhappy with its current AP, it uses **active scanning**:
  1. The node sends a **Probe** frame.
  2. All APs within reach respond with a **Probe Response** frame.
  3. The node selects an AP and sends it an **Association Request** frame.
  4. The AP replies with an **Association Response** frame.
- If a node moves and changes its AP, the new AP notifies the old AP of the change.
- APs also periodically broadcast a **Beacon** frame that advertises its capabilities, and a node can change to this AP by sending it an **Association Request** frame (**passive scanning**).

802.11 Frame Format



- **Control**:
  - Type: data, RTS, CTS, or scanning-related
  - Two 1-bit fields: ToDS and FromDS
- If one node sends directly to another, ToDS/FromDS are both 0, Addr2 is the source and Addr1 is the destination.
- If the frame was sent by a node to its AP, forwarded to another AP, and is now being sent to the destination:
  - ToDS and FromDS are 1
  - Addr4 is source node, Addr3 is first AP
  - Addr2 is second AP, Addr1 is destination node

CO7219: Internet and Cloud Computing

3. Internetworking

3.1 Internetworking

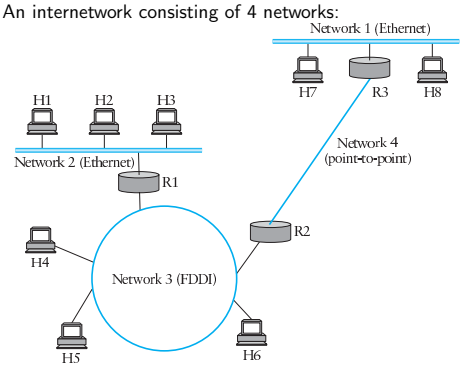
- A single network can be built using point-to-point links, shared media (Ethernet, WLAN), and switches.
- We want to interconnect different networks and build an **internetwork**.
- Two important problems must be addressed:
  - **Heterogeneity**: The individual networks may be based on very different technologies.
  - **Scale**: The Internet has roughly doubled its size each year for 20 years. How can one do **routing** and **addressing** efficiently for millions of nodes?

3.1.1 What is an Internetwork?

- An **internetwork** or **internet** is an arbitrary collection of networks (also called **physical networks**) interconnected to provide some sort of host-to-host packet delivery service.
  - An internet is a "network of networks".
- The **Internet** (with capital "I") is the widely used, global internetwork to which most networks are connected.
- The nodes that interconnect the networks are called **routers** (or sometimes **gateways**).

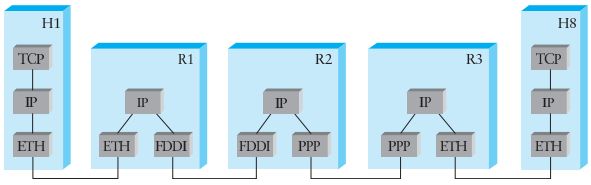
Example

- An internetwork consisting of 4 networks:



Internet Protocol (IP)

- The Internet Protocol is the key tool used today to build scalable, heterogeneous internetworks.
- IP runs on all nodes (hosts and routers) in a collection of networks and defines the infrastructure that allows them to function as a single logical network.



3.1.2 IP Service Model

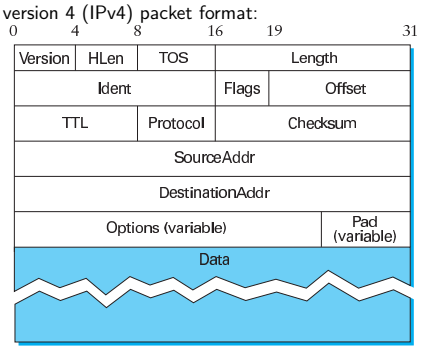
- **Datagram Delivery**
  - A datagram is a type of packet that is sent in a connectionless manner over a network. It carries enough information to let the network forward it to its correct destination.
  - "Connectionless" means that there is no advance setup mechanism to tell the network what to do when the packet arrives.
  - The network makes its **best effort** to get the packet to its destination. If something goes wrong (packet gets lost, corrupted, misdelivered), the network does nothing (**unreliable** service).
  - Packets can get delivered out of order, or more than once.

Connectionless Best-Effort Service

- **Advantages:**
  - Simplest service one could ask from a network.
  - Asking for a reliable packet delivery service would mean that a lot of extra functionality has to be put into the routers.
  - Best-effort service allows to keep the routers as simple as possible (one of the design goals of IP).
  - Enables IP to "run over anything". Today IP runs over many network technologies that did not even exist when IP was invented.
- Higher-level protocols and applications that run over IP need to be aware of all the possible failure modes.

IP Packet Format

- IP version 4 (IPv4) packet format:



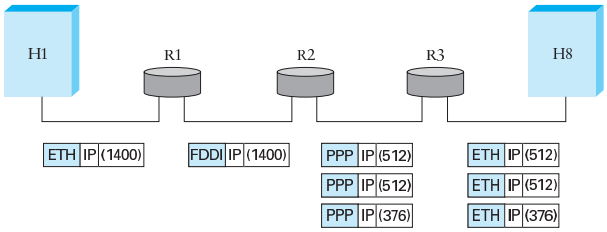
IP Header Fields

- Version: IP version number (4 in IPv4)
- HLen: Length of Header (in 32-bit words), normally 5
- TOS: Type of Service
- Length: Length in bytes (data + header), max. 65536
- Ident, Flags, Offset: used for fragmentation and reassembly
- TTL: Time to Live (number of hops before packet is discarded, default 64)
- Protocol: Demultiplexing key (e.g. TCP = 6, UDP = 17)
- Checksum: Internet checksum of header
- SourceAddr, Destination Addr: IP addresses of src/dest
- Options: rarely used (present only if HLen > 5)

Fragmentation and Reassembly

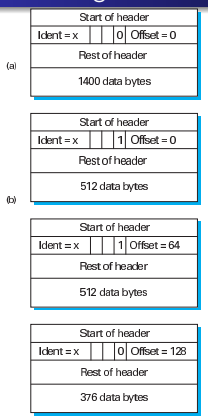
- **Problem:** IP allows packets of size up to 64 KB, but each network technology has its own limits on the size of the data in a packet: 1500 bytes for Ethernet, 4500 bytes for FDDI, etc.
- When an IP packet is to be sent over a network whose MTU (maximum transmission unit) is smaller than the size of that packet, the packet gets **fragmented** (split into smaller packets).
- The packet fragments are put together (**reassembly**) at the receiving host. All fragments originating from the same packet have the same identifier in the Ident field.
- The Ident field is chosen by the sending host and is intended to be unique among all datagrams from this source to the destination over a reasonable time period.

Fragmentation Example



- Remarks:
  - Each fragment is itself a self-contained IP datagram, transmitted independent of the other fragments.
  - Each IP datagram is reencapsulated (in a link-layer frame) for each physical network over which it travels.

Headers of Fragments



(a) unfragmented packet

(b) 3 resulting fragments

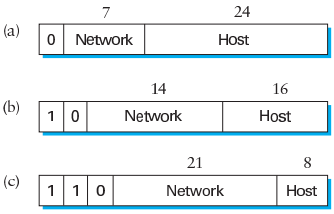
Remarks:

- In all fragments except the last, the M bit of the Flags field is set to 1.
- The Offset field gives the offset in the original packet where the data of this fragment begins (in multiples of 8 bytes).
- Note that a fragment can be split into smaller fragments if necessary.

3.1.3 Global Addresses

- In the IP service model, we need a global addressing scheme in which no two hosts have the same address (**global uniqueness**).
- Ethernet addresses are globally unique, but they are **flat** and provide no clues to routing protocols.
- IP addresses are **hierarchical**: They are made up of several parts that correspond to some sort of hierarchy.
  - Think of part of the IP address as identifying a network in the internetwork, and the rest of the IP address as identifying a host inside that network.
- Routers are attached to two or more networks: They have an address on each of these networks. Think of IP addresses as belonging to network interfaces rather than to hosts.

"Classful" IP Addresses



(a) class A WANs

(b) class B site/campus

(c) class C LANs

- IP addresses are 32 bits long, written in the form 143.210.72.129 (i.e. each byte is given in decimal).

3.1.4 Datagram Forwarding in IP

- **Forwarding** is the process where an IP router takes a packet from an input and sends it out on the appropriate output.
- Every IP datagram contains the IP address of the destination, the "network part" of the address identifying a physical network that is part of the Internet.
- All hosts and routers that share the same network part of their address are connected to the same physical network, and can thus communicate with each other by sending frames over that network.
- Each physical network that is part of the Internet contains at least one router that is connected to at least one other physical network.

Forwarding an IP Datagram

- Any node – host or router – first compares the network part of the destination address with the network parts of the addresses of all its interfaces.
- If a match is found, the destination lies on the same physical network as the interface, and the packet can be delivered to the destination over that network.
- Otherwise, the node will send the packet to some (other) router, the **next hop** router:
  - The router finds the next hop by consulting its forwarding table (conceptually, this table contains a list of ⟨NetworkNum,NextHop⟩ pairs).
  - Normally, there is a **default router** that is used if no entry in the forwarding table matches the destination network. Hosts may have only a default router.

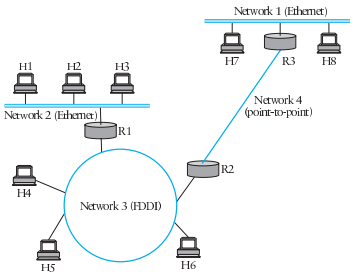
Forwarding on a Router

if (destination NetworkNum = NetworkNum of one of my interfaces)
then
    deliver packet to destination over that interface
else
    if (destination NetworkNum is in my forwarding table)
    then
        deliver packet to corresponding NextHop router
    else
        deliver packet to default router
    fi
fi

Forwarding on a Host

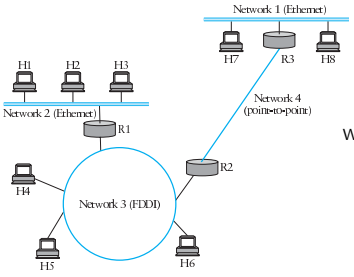
if (destination NetworkNum = my NetworkNum)
then
    deliver packet to destination directly
else
    deliver packet to default router
fi

Example 1



- If H1 sends to H2, it can send the packet directly since H1 and H2 are on the same physical network (so H2 has the same network part in the IP address).

Example 2



R2's forwarding table:

NetworkNum	NextHop
1	R3
2	R1
With directly connected networks:	
1	R3
2	R1
3	Interface 1
4	Interface 0

- If H1 sends to H8, it sends the packet to its default router R1.
- If R2 is R1's default router, R1 forwards the packet to R2.
- R2 sends the packet to next hop R3. R3 delivers packet to H8.

Remarks

- In the example, the forwarding table of R2, with only four entries, gives R2 enough information to reach all 8 hosts in the internetwork.
- The same table would work if each physical network had hundreds of nodes.

➠ Hierarchical addressing (splitting the address into network part and host part) has improved the **scalability** of the network: Forwarding tables only need entries for each network, not for each host.

3.1.5 Address Translation (ARP)

- If a router or host wants to deliver a packet to an IP address in the same physical network, it needs to use the addressing scheme of that particular network.
  - For example, if the network is an Ethernet, it needs to know the 48-bit MAC address of the destination.
- The **Address Resolution Protocol (ARP)** enables each host on a network to build a mapping between IP addresses and link-layer addresses:
  - To determine the MAC address for a particular IP address, a node broadcasts an **ARP query** (containing that IP address) in the network.
  - The node with that IP address responds to the sender of the ARP query with an **ARP response** containing its link-layer address.
  - Nodes maintain an **ARP cache** or **ARP table**.

Classless Routing (CIDR)

- "Classful" IP addresses with network part and host part make routing in the Internet somewhat scalable, but a router still needs to know about **all** networks connected to the Internet, and there are many of them.
- **Solution:** Classless interdomain routing (CIDR, pronounced "cider").
- CIDR uses **aggregation** to minimize the number of routes that a router needs to know.
  - For example, class C network numbers from 192.4.16 to 192.4.31 have the same top 20 bits:
    11000000 00000100 0001
  - If these 16 networks are close to each other, the top 20 bits can be used like a 20-bit network number.

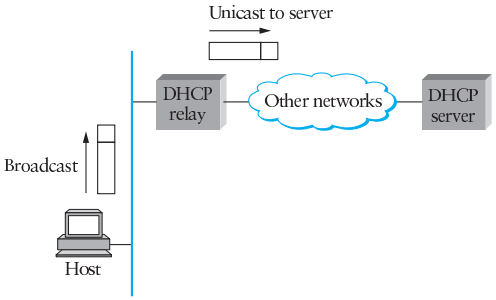
Route Aggregation

- Effectively, CIDR means that network addresses can be of arbitrary length ("classless" addressing).
- Route aggregation can happen repeatedly.
  - For example, if a provider network has assigned 20-bit network addresses
    11000000000001000000
    and 11000000000001000001
    to two customers, it can advertise a route to the common 19-bit prefix to the rest of the network:
    1100000000000100000
  - All routers in the rest of the network treat the 19-bit prefix as a single network (➠ scalability).
  - With CIDR, routers forward packets according to the "longest match" in their forwarding table.

3.1.6 Host Configuration (DHCP)

- Manual configuration of IP addresses in a local area network is tedious and error-prone.
- Primary automated configuration method: **Dynamic Host Configuration Protocol (DHCP)**
  - A newly booted or attached hosts sends a DHCPDISCOVER message to 255.255.255.255 (IP broadcast in local network).
  - DHCP server responds with IP address assigned to host.
  - Assignment of IP addresses to hosts can be **static** (IP address is always the same for each MAC address) or **dynamic** (server maintains pool of available IP addresses and assigns them dynamically).

DHCP Relay Agents



- To avoid having a DHCP server on every network, **relay agents** can forward DHCPDISCOVER messages to the DHCP server (and replies back to the host).

3.1.7 Error Reporting (ICMP)

- **Internet Control Message Protocol (ICMP)**, a companion protocol of IP, defines error messages sent back to the source host when an IP datagram cannot be processed successfully.
  - Destination host unreachable (e.g. link failure).
  - Reassembly process failed.
  - TTL (time to live) has reached 0. (➠ **traceroute**)
  - IP header checksum failed.
  - ...
- ICMP also defines control messages that a router can send back to a source host, e.g. an ICMP-Redirect that tells the host that there is a better route to the destination.

3.2 Routing

- **Forwarding**: Taking a packet, looking at its destination address, consulting the forwarding table, and sending the packet to the next hop router specified in the table.
- **Routing**: The distributed process by which the forwarding tables at the nodes are built.
- We can distinguish **routing tables**, i.e., tables containing a mapping from network numbers (or prefixes) to next-hop routers, and **forwarding tables**, i.e., tables giving for each network number (or prefix) the outgoing interface and MAC information.
  - Routing tables are produced by the routing algorithm.
  - Forwarding tables are built from the routing tables.

Routing and Forwarding Tables

- **Routing table**:

Network Number	NextHop
10	171.69.245.10

- **Forwarding table**:

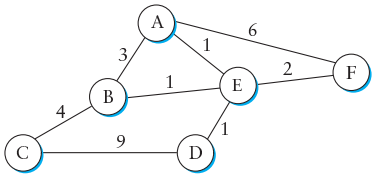
Network Number	Interface	MAC Address
10	if0	8:0:2b:e4:b:1:2

- Routing table and forwarding table contain essentially the same information, but the forwarding table is stored in such a way that the lookup problem can be solved quickly, and it also contains all forwarding information in the form that is needed to do the actual forwarding (interface, and MAC address of next-hop router).

Routing Algorithms

- First, we consider **intradomain** routing, or **interior gateway protocols (IGPs)**.
  - A routing **domain** is an internetwork in which all routers are under the same administrative control (e.g. a university campus or an ISP).
  - Intradomain routing algorithms are suitable in the context of small to midsized networks only (fewer than 100 nodes, in practice).
  - Intradomain routing is complemented by **interdomain routing** between the different routing domains.

3.2.1 Network as a Graph



- Nodes: hosts, switches, routers, networks
- Edges: network links
- Edge costs: indicate desirability of sending traffic over that link
- Routing should find the lowest-cost path between any two nodes.

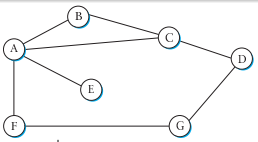
Need for Distributed Algorithms

- The network changes dynamically (node or link failures, addition of new nodes or links, heavily loaded links receive higher costs).
- Therefore, it is not sufficient to compute all shortest paths once and then store the respective information in all nodes.
- In practice, routing protocols provide **distributed, dynamic** ways to maintain shortest paths in a changing network.
- Distributed computation faces many challenges, e.g., two routers may have different ideas about the shortest path to a destination and pass a packet back and forth between them.

3.2.2 Distance Vector Routing (RIP)

- Distance-vector routing is based on the Bellman-Ford algorithm for shortest paths.
- **Basic ideas:**
  - Each node constructs an array (vector) containing the distances to all other nodes, and sends it to its neighbours.
  - If a node learns a shorter path to a destination from a neighbour, it updates its own distance information for that destination.
  - Initially, each node knows the costs of the links to its directly connected neighbours. Links that are down are assigned infinite cost.

Example: Initial Distances



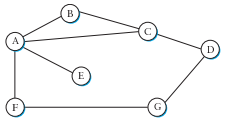
	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

- All edge costs in this example are 1.
- Each node only knows the information in one row of the table.

Initial Routing Table

- Node A's initial routing table:

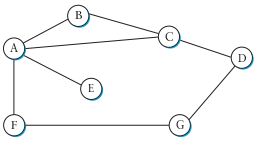
Destination	Cost	NextHop
B	1	B
C	1	C
D	∞	–
E	1	E
F	1	F
G	∞	–



- After receiving distance vectors of its neighbours:

Destination	Cost	NextHop	
B	1	B	
C	1	C	
D	2	C	(updated as C has distance 1 to D)
E	1	E	
F	1	F	
G	2	F	(updated as F has distance 1 to G)

Final Distances



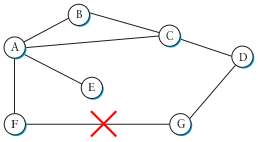
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

- After a few exchanges of information between neighbours, all nodes have consistent routing tables and correct distance information (**convergence**).
- Still, each node knows only one row of the matrix.

Remarks

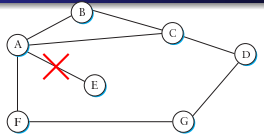
- All nodes achieve a consistent view of the network in the absence of any centralised authority.
- When does a node send distance information to its neighbours?
  - **Periodic update**: Typically, every few seconds or minutes.
  - **Triggered update**: Whenever a node's routing table changes, it sends its updated distance information to its neighbours.
- When a link or node fails, the neighbouring nodes notice it, update their distances, and send updated distances to their neighbours. Normally the system settles down fairly quickly to a new consistent state.

Failure Example



- When the link F–G fails, F detects it, sets its distance to G to ∞, and passes this on to A.
- A knows that its 2-hop path to G is through F, so A also sets its distance to G to ∞.
- With the next routing update from C (which has a 2-hop path to G), A learns that it has a 3-hop path to G (through C).
- When A advertises this to F, node F learns that it has a path to G of cost 4 (through A).

Count-to-Infinity Problem



- A–E fails.
- A advertises distance ∞ to E, but B and C may still advertise distance 2 to E.
- B concludes it has a path of distance 3 to E (via C) and advertises it.
- A thinks it has a path of distance 4 to E (via B) and advertises it.
- C thinks it has a path of distance 5 to E (via A) and advertises it.
- Cycle continues until distance values reach ∞. Until then, the network does not stabilise.

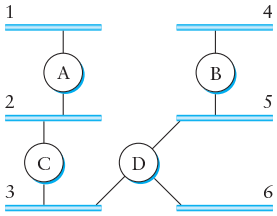
Partial Solutions

- Use a relatively small number as an approximation of infinity, e.g. 16.
- **Split horizon**: When a node sends a routing update to its neighbour, it does not send those routes that it learned from that neighbour.
- **Split horizon with poison**: When a node sends a routing update to its neighbour, it send those routes that it learned from that neighbour as well, but it puts negative information in the route to ensure that the neighbour will not use it (e.g. path cost ∞).
- However, the split horizon techniques only work for routing loops involving two nodes.

Routing Information Protocol (RIP)

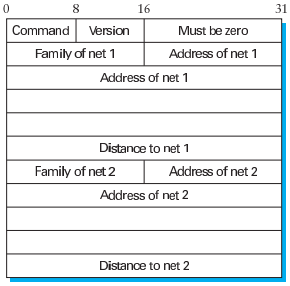
- One of the most widely used routing protocols in IP networks.
- Was distributed with the BSD version of Unix.
- RIP is the canonical example of a routing protocol built on the distance-vector algorithm.
- Routers running RIP actually advertise distances to **networks**. They send periodic updates every 30 seconds (and triggered updates when the routing table changes).
- RIP uses link costs equal to 1, and it uses the number 16 to represent infinity (so a network running RIP must not have a shortest path of more than 15 hops).

Example Network Running RIP



- For example, router C advertises to router A that it can reach:
  - networks 2 and 3 at cost 0
  - networks 5 and 6 at cost 1
  - network 4 at cost 2

RIP Packet Format

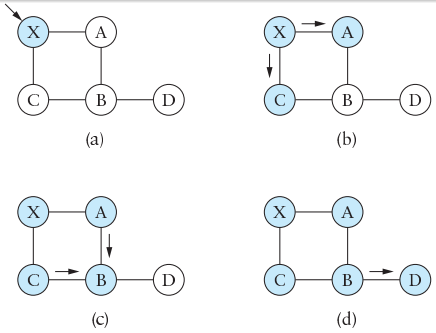


- Majority of the packet consists of ⟨network-address,distance⟩ pairs.
- RIP supports multiple address families, not just IP addresses.

3.2.3 Link-State Routing (OSPF)

- Second major class of intradomain routing protocols.
- Idea: Disseminate information about the whole network to every node.
- Two basic mechanisms:
  - Reliable dissemination of link-state information.
  - Calculation of routes from the sum of all the accumulated link-state knowledge.
- Initially, each node knows only the state of the link to each neighbour (up or down) and its cost.
- The information of each node (states and costs of the links to its neighbours) is put into a **link-state packet** (LSP) and **flooded** to all other nodes.

Flooding a LSP



- Each node forwards the LSP to all neighbours except the one from which the LSP was received.

Remarks

- LSPs are generated periodically and in response to topology changes.
- LSPs contain sequence numbers that make it possible to distinguish a new LSP from an older one.
- Once a node has a copy of the LSP from every other node, it can compute a complete map of the network topology.
- From the complete network map, a node can then compute the shortest paths to all other nodes (e.g. using the **Dijkstra** algorithm) and build its routing table.

The OSPF Protocol

- OSPF = Open Shortest Path First
- Most widely used link-state routing protocol.
- Uses **authentication** of routing messages.
  - Protection against misconfigured routers etc.
- Introduces additional hierarchy by allowing a routing domain to be partitioned into **areas**.
  - Reduces amount of information that must be transmitted to and stored in each node.
  - A router does not necessarily need to know how to reach each network in its domain; it may be sufficient to know how to get to the right area.
- **Load balancing**: OSPF allows to distribute traffic evenly among multiple routes of the same cost.

3.2.4 Metrics

- Both distance-vector and link-state routing use link costs (metrics) for shortest path calculations.
- It is **not easy** to determine appropriate link costs.
  - Simplest approach: Assign cost 1 to all links.
    - Used by RIP.
    - Not good because it completely ignores latency, capacity, and load.
  - Original ARPANET routing metric: Number of packets in the queue for the link.
    - Did not work well as it moves packets towards the shortest queue rather than towards the destination.
    - Does not take bandwidth or latency into account.

Second ARPANET Metric

- Second version of ARPANET routing metric ("new routing mechanism"): Compute packet delay as

  $$PacketDelay = (DepartTime - ArrivalTime) + TransmissionTime + Latency$$

  and use average packet delay as link metric.
  - Works reasonably well under light load (where the static values TransmissionTime and Latency dominate the cost).
  - However, bad behaviour under heavy load:
    - Cost of congested link goes up, all traffic moves away from it, link becomes idle and cost goes down, traffic moves back to the link ➠ many links are idle a lot of the time under heavy load.
  - Other problem: Range of link values was too large.

Third ARPANET Metric

- Third version: "revised ARPANET routing mechanism."
- Major changes:
  - Dynamic range of the metric reduced.
  - Link type taken into account.
  - Variation of metric smoothed over time.
- Properties:
  - Highly loaded link at most 3 times as expensive as when it is idle.
  - Most expensive link at most 7 times as expensive as cheapest link.
  - High-speed satellite link more attractive than low-speed terrestrial link.
  - Link utilisation affects cost only at moderate to high loads.

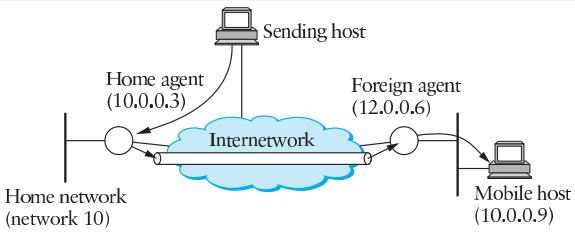
3.2.5 Routing for Mobile Hosts

- A host's IP address consists of network part and host part.
- But: A mobile host may move from network to network.
- A host normally gets a new IP address when it moves to a new network (e.g. using DHCP), but we would like this to be **transparent** to the user and to the applications running on the host and their remote counterparts.
- Procedures to address this problem are referred to as **Mobile IP**.
- Design decision for Mobile IP: Should work without any changes to software of nonmobile hosts or the majority of routers in the Internet.

Mobile IP

- Every mobile node has a **home network** with a router that is the **home agent** of the mobile node.
- The mobile node has a permanent IP address, its **home address**, whose network part is that of the home network.
- When the mobile node is in a foreign network, it registers with a **foreign agent** in that network. The foreign agent contacts the home agent and provides a **care-of address**, usually its own IP address.
- Traffic to the mobile node is
  - sent to its home address,
  - intercepted by the home agent and forwarded (IP tunnel) to the foreign agent, and
  - delivered to the mobile node by the foreign agent.

Illustration of Mobile IP



- The home agent uses ARP messages to associate the home address of the mobile node with its own hardware address ("proxy ARP" technique).
- Thus, all traffic sent to the home address reaches the home agent and can be "tunneled" to the foreign agent.

Remarks

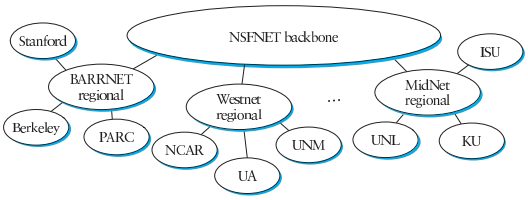
- If the mobile node has acquired a new IP address in the foreign network, it can act as the foreign agent itself.
- Traffic sent from the mobile node to a fixed node can be sent directly: The mobile node uses its home address as source address, and the fixed node's address as destination address.
- If traffic is sent between two mobile hosts, the same mechanism is used in both directions.
- Mobile routing provides **security challenges**: A malicious node wishing to intercept packets for a mobile node could contact the home agent and pretend to be the foreign agent for that node. Therefore, **authentication mechanisms** are required.

Route Optimisation in Mobile IP

- Basic routing mechanism of Mobile IP incurs the "triangle routing problem": Instead of sending traffic directly to the mobile node, it must make a detour through the home agent.
- **Proposed solution**: Inform the sending node about the current care-of address (IP address of foreign agent)!
  - Nodes maintain a **binding cache** of mappings from mobile node addresses to care-of addresses.
  - If a packet for the mobile node arrives at the home agent, the home agent informs the sender about the current care-of address (**binding update** message).
  - Sender then sends packets to foreign agent directly.
  - When the foreign agent receives a packet for a mobile node that is no longer registered with it, it sends a **binding warning** back to the sender.

3.3 Global Internet

- Tree structure of the Internet in 1990:

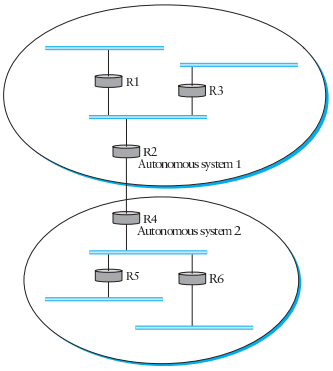


- End user sites (typically consisting of multiple Ethernets and other networks, connected by routers and bridges).
- Regional "service provider" networks.
- National backbone network.

Autonomous Systems

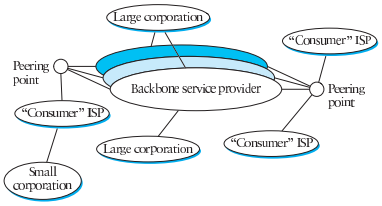
- Each provider and end user is likely to be an administratively independent entity.
- An **Autonomous System (AS)** is a network that is administered independently of other ASs.
- Different providers may have different ideas about the best routing protocol to use within their network, and on how metrics should be assigned to the links.
- Divide the routing problem into two parts:
  - **intradomain** routing (inside a routing domain or AS)
  - **interdomain** routing (between ASs)
- Each AS can run whatever intradomain routing protocol it chooses.

Example with Two ASs



Interdomain Routing Protocols

- First interdomain routing protocol in the recent history of the Internet: **Exterior Gateway Protocol (EGP)**. Had severe limitations, was designed for treelike topology.
- EGP was replaced by the **Border Gateway Protocol (BGP)**, currently in version 4 (BGP-4). Works for arbitrarily interconnected ASs, such as today's multibackbone Internet:



IP Version 6 (IPv6)

- Motivation for a new version of IP: scaling problems caused by the Internet's massive growth; address space being consumed too fast.
- IETF began looking into an expansion of the IP address space in 1991.
- Since longer addresses dictate a change in the IP header, a new version of IP is necessary.
- A new version of IP requires new software in all routers and hosts, so it was felt that one might as well use the new version to fix as many other things in IP as possible.
- New version of IP initially known as **IP Next Generation (IPng)**, but now called **IPv6**.

Wish List Items for IPv6

- Larger address space.
- Support for real-time services.
- Security support.
- Autoconfiguration (i.e., ability of hosts to automatically configure themselves with information such as IP address and domain name).
- Enhanced routing functionality, including support for mobile hosts.

Besides, a transition period during which IPv4 and IPv6 can be used concurrently must be supported.

Note: Many of these features were absent from IPv4 at the time IPv6 was being designed, but support for them has been added to IPv4 in recent years.

IPv6 Addresses

- IPv6 uses 128-bit addresses ($\Rightarrow 3 \times 10^{38}$ unique addresses).
- Even based on pessimistic estimates of address assignment efficiency, IPv6 addresses provide over 1500 addresses per square foot of the earth's surface.
- IPv6 address space is subdivided in various ways based on the leading bits.
- Entire functionality of IPv4's main address classes (A, B, and C) is contained inside the 001 prefix (**Aggregatable Global Unicast Addresses**).
- Other prefixes reserved for multicast addresses, link local use addresses, site local use addresses, ...

IPv6 Addresses (2)

- Part of IPv6 addresses starting 00000000 are reserved for:
  - **IPv4-compatible IPv6 addresses** (32-bit IPv4 address zero-extended to 128 bits)
  - **IPv4-mapped IPv6 addresses** (32-bit IPv4 address prefixed with 2 bytes all 1s and then zero-extended to 128 bits)
- IPv6 addresses are written in the form:
  47CD:1234:4422:AC02:0022:1234:A456:0124
- :: used as short-hand for blocks of 0000 words.
- Special notation for IPv4-mapped IPv6 addresses:
  ::FFFF:128.96.33.81

Aggregatable Global Unicast Addresses

- Address allocation similar to that being deployed with CIDR in IPv4 (so that aggregation can be used effectively).
- Hierarchical assignment of addresses. For example, an IPv6 address **might** look like this:

3	m	n	o	p	125−m−n−o−p
001	RegistryID	ProviderID	SubscriberID	SubnetID	InterfaceID

  - RegistryID: identifier assigned to e.g. a European address registry
  - ProviderID: identifier of service provider
  - SubscriberID: identifier of end user network
  - SubnetID: identifier of physical network

Other Features of IPv6

- **Autoconfiguration**: A host can combine its 48-bit MAC address with the correct prefix (advertised by a router) or with the standard link local use prefix (if it does not need a globally unique address) to obtain an IP address. No need for a DHCP server.
- **Advanced Routing Capabilities**: IPv6 packets can have a routing header that specifies nodes or areas that the packet should visit on its way to the destination.
  - This feature could be used for provider selection on a packet-by-packet basis, for example.

Deployment of IPv6

- IPv6-capable host operating systems are widely available, and router vendors offer varying degrees of IPv6 support.
- However, IPv6 deployment has not begun in any meaningful way.
- **Network address translation (NAT)** has addressed the problem of IPv4 addresses becoming sparse, thus reducing the need for an upgrade to IPv6.
  - NAT allows all nodes in a network to share one (or few) globally unique IPv4 addresses.
- It is not clear when IPv6 deployment will begin in earnest, and what will cause it.
  - Applications that do not work well with NAT could be driving factor (multiplayer gaming, IP telephony, etc.).

CO7219: Internet and Cloud Computing

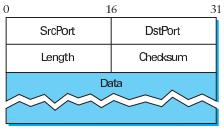
4. End-to-End Protocols

End-to-End Protocols

- The **transport** level of the network architecture turns the best-effort host-to-host packet delivery service into a process-to-process communication channel.
- Examples of desirable properties:
  - guarantees message delivery.
  - delivers messages in the same order they are sent.
  - delivers at most one copy of each message.
  - supports arbitrary-length messages.
  - supports synchronisation between sender and receiver.
  - allows receiver to apply flow control to the sender.
  - supports multiple application processes on each host.

4.1 Simple Demultiplexer (UDP)

- Simplest possible transport protocol: extends host-to-host delivery service into process-to-process communication service, adds no further functionality.
- The Internet's **User Datagram Protocol (UDP)** is an example of such a transport protocol.
- UDP uses the concept of **ports** to indirectly identify an application process on a host (cf. socket interface):
  A ⟨port,host⟩ pair is used as demultiplexing key.
- UDP header:

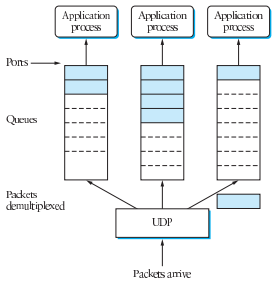


Port Numbers

- How does a process learn the port for the process to which it wants to send a message?
  - **Well-known port** approach: Specific services always use the same port (e.g. DNS: 53, mail: 25, talk: 517), cf. /etc/services.
    Sometimes the port is just the starting point for communication, to agree on a port to be used for subsequent communication.
  - **Port Mapper**: Client sends message to Port Mapper's well-known port asking for the port it should use to to talk to the service it requires.
- As a UDP packet contains the port number of the sending application, the recipient simply sends replies to that port.

Port Implementation

- Operating systems typically implement ports as message queues:



Remark: Multimedia applications often employ the real-time transport protocol (RTP) that runs on top of UDP.

4.2 Reliable Byte Stream (TCP)

- The Internet's **Transmission Control Protocol (TCP)** provides a reliable, connection-oriented, byte-stream service.
- TCP is the most widely used protocol of its type, and has been very carefully tuned.
- TCP guarantees reliable, in-order delivery of a stream of bytes.
- TCP is a full-duplex protocol, i.e., each TCP connection supports a pair of byte streams, one in each direction.
- TCP includes mechanisms for **flow control** (preventing the sender from overrunning the receiver) and **congestion control** (preventing the sender from overloading the network).

4.2.1 End-to-End Issues

TCP uses the sliding window algorithm, but over the Internet rather than a point-to-point link, so the following difficulties arise:

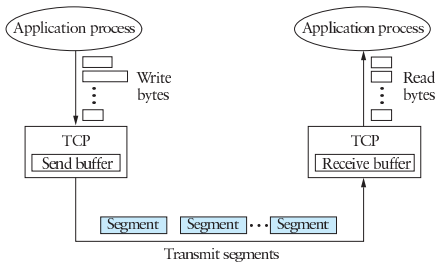
- TCP needs an explicit connection establishment phase during which the two sides agree to exchange data and establish some shared state for the sliding window algorithm to begin.
- The RTT can differ widely across different TCP connections and can vary dynamically ➠ TCP needs an adaptive timeout mechanism.
- Packet reordering is possible ➠ TCP assumes a maximum segment lifetime (MSL), typically 120 seconds.

End-to-End Issues (cont.)

- Any kind of computer can connect to the Internet, so the amount of resources dedicated to a TCP connection are highly variable ➠ TCP must include a flow control mechanism.
- The sender has no idea what links will be traversed to reach the destination, so it can potentially create network congestion ➠ TCP needs a congestion control mechanism.

➠ Running the sliding window algorithm over the Internet is much more complex than running it over a dedicated point-to-point link.

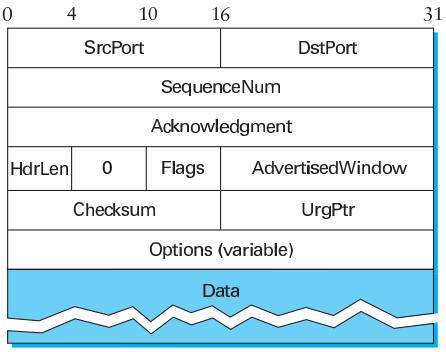
4.2.2 Segment Format



- TCP is byte-oriented, but transmits packets called **segments**.
- TCP's demux key is the 4-tuple:

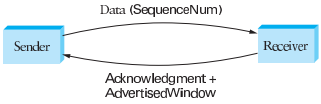
  $\langle SrcPort, SrcIPAddr, DstPort, DstIPAddr \rangle$

TCP Header Format



Explanation of TCP Header Fields

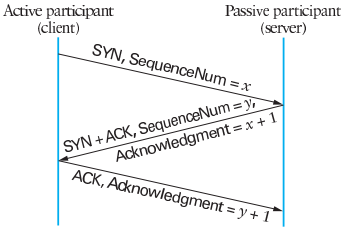
- For sliding window algorithm:



  - SequenceNum: seq. number of first **byte** in segment
  - Acknowledgment, AdvertisedWindow: information about the flow going in the other direction
- HdrLen: Length of header in 32-bit words
- Flags: SYN, FIN, RESET, PUSH, URG, ACK
- UrgPtr: indicates where the nonurgent data in the segment begins (urgent data present if URG flag is set).

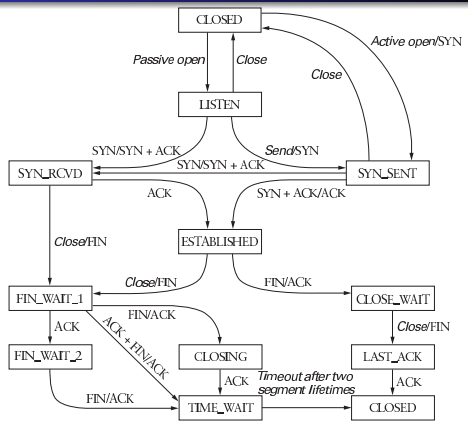
4.2.3 Connection Establishment and Termination

- Three-way handshake to establish connection:



  - Acknowledgment field represents "next sequence number expected," so value is $x + 1$ ($y + 1$).
  - x and y are chosen randomly (to avoid confusion with sequence numbers of a previous incarnation of the same TCP connection).

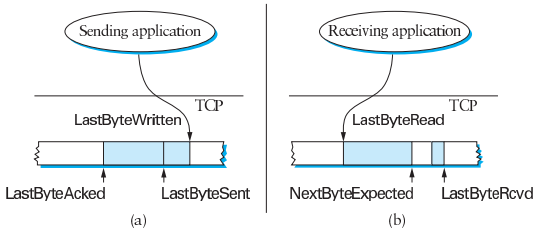
State Transition Diagram



4.2.4 Sliding Window Revisited

- TCP differs from the sliding window algorithm discussed earlier by adding a **flow control** mechanism.
- Rather than having a fixed-size sliding window, the receiver **advertises** a window size to the sender.
- The sender is limited to having at most AdvertisedWindow many bytes of unacknowledged data at any time.
- The receiver selects a suitable value of AdvertisedWindow based on the amount of memory allocated to the connection for buffering data.
- The value of AdvertisedWindow can change dynamically.
- The idea is to keep the sender from overrunning the receiver's buffer.

TCP Send Buffer and Receive Buffer



- Receive buffer is of size MaxRcvBuffer
- Receiver advertises AdvertisedWindow as:
  MaxRcvBuffer – ((NextByteExpected – 1) – LastByteRead)
- Sender ensures:
  LastByteSent – LastByteAcked $\leq$ AdvertisedWindow

Further Remarks

- The receiver can reduce the AdvertisedWindow all the way down to 0 and effectively stop the sender from sending data (but the sender will keep trying to send a 1-byte segment periodically if the AdvertisedWindow is 0).
- TCP uses **adaptive retransmission**: Timeouts are selected based on estimated RTT and RTT variance.
- TCP uses certain rules to trigger the transmission of segments (e.g. if both the window and the available data are larger than MSS, a segment of size MSS is sent, where MSS = maximum segment size).
- TCP extensions allows larger advertised windows by using a **scaling factor**.
  - STS-12 (622 Mbps) link with RTT 100 ms has delay × bandwidth product 7.4 MB, but the 16-bit field for AdvertisedWindow would limit window size to 64 KB.

UNIVERSITY OF LEICESTER

CO4219/CO7219
Internet and Cloud Computing
Part II: Cloud Computing

- Introduction to cloud computing
- Scalable distributed computation with MapReduce
- Hadoop implementation of MapReduce algorithms
- Privacy and security
- Fault-tolerance

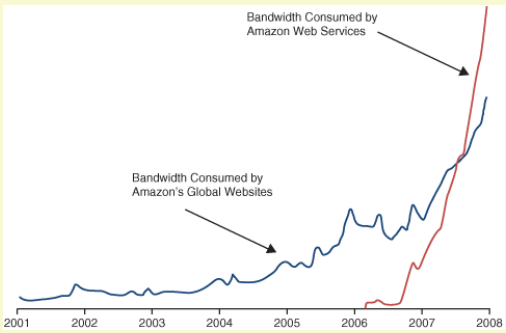
www.le.ac.uk

Textbooks

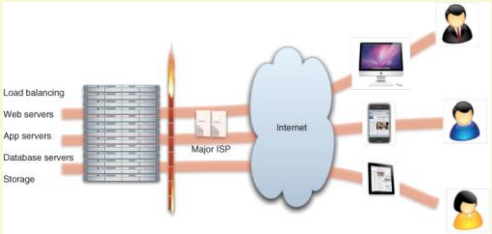
- J. Rosenberg, A. Mateos: **The Cloud at Your Service**: The when, how, and why of enterprise cloud computing. Manning Publications, 2010. (available via the Library in Safari Books Online)
- J. Lin, C. Dyer: **Data-Intensive Text Processing with MapReduce**. Morgan & Claypool Publishers, 2010. http://lintool.github.io/MapReduceAlgorithms/ed1n/MapReduce-algorithms.pdf
- T. White: **Hadoop – The Definitive Guide**. 3rd Ed, O'Reilly, 2012.

What is cloud computing? (CAYS Ch. 1)

- Hottest buzzword in the IT world in the last few years
- Computing services offered by a third party, available for use when needed, that can be scaled dynamically, in response to changing needs
- First public cloud offering from Amazon attracted 500,000 customers in the first 18 months.



Origin of the "cloud" metaphor



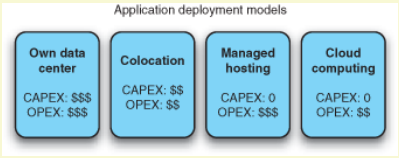
Five main principles of cloud computing

- **Pooled computing resources**, available to any subscribing user
- **Virtualized computing resources** to maximize hardware utilization
- **Elastic scaling** up or down according to need (dynamic scale without capital expenditure)
- **Automation** – build, deploy, configure, provision and move, without manual intervention
- **Metered billing** – per-usage business model, pay only for what you use

UNIVERSITY OF LEICESTER 7

Pooled computing resources

- Hosting choices for IT organizations:



- CAPEX=capital expenditure, OPEX = operational exp.
- Cloud computing exploits economies of scale

UNIVERSITY OF LEICESTER 8

Virtualization of compute resources

- Each physical server is partitioned into many virtual servers
- Each virtual server can run an operating system and applications
- Shift to multicore servers strengthens the impact of virtualization
- Virtualization increases utilization of physical servers dramatically

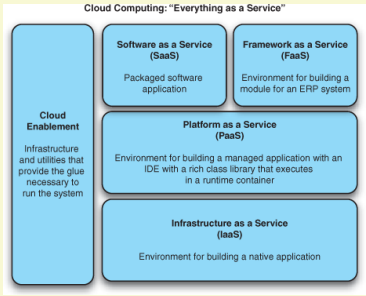
UNIVERSITY OF LEICESTER 9

Benefits from moving to the cloud

- Economic benefits, incl. change from CAPEX to OPEX (for example, a medium-sized company reduced the monthly cost for server hosting from $2,200 to $250 by moving to the cloud)
- Agility benefits from not having to procure and provision servers
- Efficiency benefits that may lead to competitive advantages
- Security (can be) stronger and better in the cloud

UNIVERSITY OF LEICESTER 10

X-as-a-Service taxonomy



UNIVERSITY OF LEICESTER 11

Infrastructure as a Service (IaaS)

- Also Hardware as a Service (HaaS)
- IaaS provider supplies virtual machine images of different OS flavours
- User can bring online and use instances of virtual machines as needed
- Charged for use of VMs, storage and bandwidth
- Example: Amazon Elastic Compute Cloud (EC2), Microsoft Azure

UNIVERSITY OF LEICESTER 12

Platform as a Service (PaaS)

- Platform abstracts away interaction with the virtual operating system.
- Simpler to use than IaaS, but less flexible and requires coding in specific languages supported by the PaaS provider
- Examples: Google AppEngine, Microsoft Azure

UNIVERSITY OF LEICESTER

13

Software as a Service (SaaS)

- Software and data hosted in the cloud, available on demand
- Examples: Gmail, Google Docs, Salesforce CRM
- Framework as a Service (FaaS) allows customers to augment and enhance the capabilities of the base SaaS system, creating custom, specialised applications

UNIVERSITY OF LEICESTER

14

Technological underpinnings (CAYS Ch.2.1)

- Data center with servers on a network
- Virtualized servers
- Access API (provision and decommission servers etc)
- Storage (machine images, applications, persistent data)
- Database (needed for many applications)
- Elasticity (ability to expand and contract applications)

UNIVERSITY OF LEICESTER

15

Data Centers

- Servers often mounted in 19-inch rack cabinets (42 slots), in single rows with corridors between them
- Need unwavering power – backup batteries and diesel generators to handle brownouts and outages
- Cooling via air-conditioning or water cooling
- Ample network bandwidth
- Physical and logical security
- Disaster recovery contingencies

UNIVERSITY OF LEICESTER

16

Data Center Scale

- Traditional data centers cost $100-200 million
- Cloud data centers cost $500 million or more
- Geographic proximity to heavy usage areas and access to cheap power.
- Electricity costs of $30 million per year (data centers consume 0.5% of the world's electricity)
- Volume discounts: Amazon spent $90 million for 50,000 servers from Rackable/SGI in 2008, which would cost $215 million without discount

UNIVERSITY OF LEICESTER

17

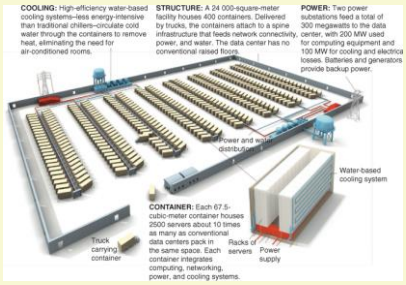
Google Data Centers

- Cheap servers built from components
- High-efficiency power supplies, power management features (voltage/frequency scaling): Power usage efficiency 1.125 (compared to a typical 2.5)
- Dalles, Oregon location near Dalles Dam for cheap power and cooling
- Google have data centers in about 25 locations with 450,000 servers (as of 2007)

UNIVERSITY OF LEICESTER

18

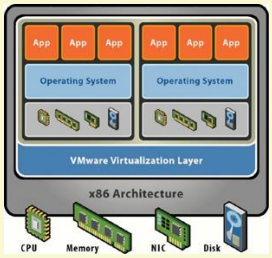
Expandable, modular cloud data center



UNIVERSITY OF LEICESTER

19

Virtualization



- Guest operating systems access virtual hardware resources
- Virtualization leads to higher utilization and increases flexibility and agility

UNIVERSITY OF LEICESTER

20

Databases in the cloud

- Conventional databases: Relational database management systems (RDBMS), also called SQL databases
- RDMBS do not easily scale to cloud-scale data sets and workloads (e.g. because of referential integrity)
- Since 1998, rapidly growing movement towards non-SQL type databases, e.g. key-value databases (Amazon SimpleDB, Google BigTable)
- NoSQL systems employ a distributed architecture

UNIVERSITY OF LEICESTER

21

Computing in the Cloud (DITPM, Ch. 1)

- "Big data": vast repositories of public and private data
- Gathering, analyzing, monitoring, filtering, searching, organizing web content must tackle big-data problems
- Business intelligence and analysis of user behaviour data: data warehousing, data mining, analytics.
- High-energy physics, astronomy, bioinformatics, …

UNIVERSITY OF LEICESTER

22

Big ideas in MapReduce cloud computing

- Scale "out", not "up": use large numbers of low-end servers, not expensive high-end servers
- Assume failures are common: If mean-time between failures (MTBF) is 1,000 days, a 10,000-server cluster has 10 server failures per day on average
- Move processing to the data – exploit data locality
- Process data sequentially, avoid random access
- Hide system-level details from application developer
- Seamless scalability

UNIVERSITY OF LEICESTER

23

MapReduce basics (DITPM CH. 2)

- Divide and conquer: partition a large problem into many subproblems, process the subproblems in parallel, then combine the solutions
- MapReduce provides a simple abstraction that hides many system-level details from the programmer
- Storage is typically handled by a distributed file system that sits underneath MapReduce
- MapReduce is a programming model, an execution framework, and its implementation (e.g Hadoop)

UNIVERSITY OF LEICESTER

24

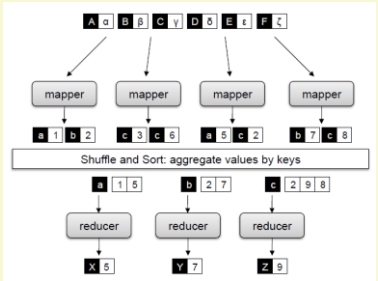
Mappers and reducers

- Input processing in two stages
  1. User-specified computation (in parallel) over all input records in the data set (Map)
  2. Aggregation of results by another user-specified computation (Reduce)
- More complex algorithms can be decomposed into a sequence of MapReduce jobs
- Map and Reduce operate on key-value pairs

Signatures of mapper and reducer

- Convention: [T] denotes a list of elements of type T
- map: (k1,v1) → [(k2,v2)]
  Map key-value pair (k1,v1) to a list of key-value pairs
- reduce: (k2, [v2]) → [(k3,v3)]
  Reduce one key with a list of values to a list of key-value pairs
- All mapper outputs with the same key are passed to the same reducer
- Each reducer's output is written to the file system

Simplified view of mappers and reducers



DITPM p. 22

Simple WordCount in MapReduce

- **class** Mapper
  method MAP(docid a, doc d)
  for all term t in doc d **do**
  EMIT(term t, count 1)
- **class** Reducer
  method REDUCE(term t, counts [c1,c2,…])
  sum ←0
  for all count c in counts [c1, c2, …] **do**
  sum ← sum + c
  EMIT (term t, count sum)

Comments on implementation details

- Mapper and Reducer are classes that implement MAP and REDUCE methods
- Hadoop initializes a Mapper object for each map task, and calls the MAP method on each key-value pair (and similarly for Reducer)
- Programmer provides hint on number of Mappers, and can specify precisely the number of Reducers
- Mappers and Reducers are allowed to have side effects

More on implementation details

- Input key-value pairs, intermediate key-value pairs (mapper output and reducer input), and output key-value pairs can all be different
- MapReduce jobs with no reducers are possible – output is then one file per mapper
- In the most common case, input to a MapReduce job comes from data stored on the distributed file system, but databases such as BigTable can also be used for input and output

Execution Framework

- MapReduce program consists of code for mappers and reducers (and combiners and partitioners), plus configuration parameters (e.g. Input/output)
- Developer submits MapReduce job to the submission node (in Hadoop: job tracker)
- Execution framework ("runtime") takes care of everything else (distributed code execution on clusters ranging from one node to thousands)

Scheduling

- MapReduce jobs are broken down into smaller units called tasks (e.g. map tasks and reduce tasks)
- Scheduler maintains task queue and tracks progress of running tasks
- Speed of a MapReduce job is sensitive to unusually slow tasks ("stragglers").
- Speculative execution: Execute identical copy of a task on a different machine, use the first result.

Data/code co-location

- Input and output data is stored in a distributed file system
- To achieve data locality, the scheduler starts a task on the node that holds a particular block of data needed by the task.
- If this is not possible, necessary data is streamed over the network (preferably to a node in the same rack)

Synchronization

- In MapReduce, synchronization is accomplished by a "barrier" between the map and reduce phases.
- Barrier groups intermediate key-value pairs by key and routes them to the right reducer: "shuffle and sort"
- For m mappers and r reducers, up to m * r distinct copy operations may be needed.
- Reduce phase cannot start until map phase is complete.

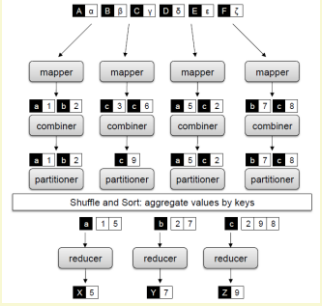
Error and fault handling

- MapReduce runtime must be resilient to hardware and software errors:
  - hardware faults
  - disk failures
  - RAM errors
  - Planned data center outages (maintenance, upgrade)
  - Unplanned data center outages (e.g. power failure)

Partitioners and Combiners

- In addition to mappers and reducers, we have:
  - Partitioners: divide intermediate key space and assign intermediate key-value pairs to reducers
  - Combiners: "mini-reducers" that allow local aggregation on the output of mappers, prior to the "shuffle and sort"
  - Hadoop runtime uses combiners at its discretion: combiner may be invoked zero, one, or multiple times

Mapper, combiner, partitioner, reducer



Note: In Hadoop, partitioners are actually executed before combiners.

DITPM p. 28

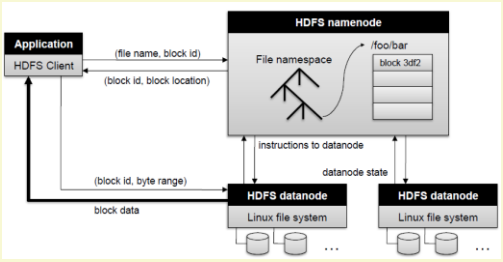
Storage in HPC

- Conventional High-Performance Computing (HPC):
  - Storage is viewed as a distinct and separate component from computation.
  - Use e.g. network-attached storage (NAS) or storage area networks (SAN)
  - Compute node fetches input from storage, processes the data, writes back results
  - Link between compute nodes and storage nodes becomes bottleneck as data size increases

Distributed File System (DFS)

- Abandon separation of computation and storage
- HDFS (Hadoop Distributed File System) is an open-source implementation of the Google File System (GFS) that supports Hadoop
- Divide user data into blocks of 64 MB.
- Master-slave architecture: master maintains file namespace (metadata, directory structure, file to block mapping, block location, access permissions), slaves manage actual data blocks

HDFS Architecture



HDFS Reliability

- By default, HDFS stores three separate copies of each data block on servers in different physical racks
- This makes HDFS resilient to datanode crashes and network failures that bring a rack offline
- Replicated blocks also make it easier to co-locate data and processing in MapReduce
- Namenode directs creation of additional copies of a block as needed.

HDFS design choices

- Support modest number of **large** files:
  - Many small files would exceed the namenode's memory.
  - Mappers in MapReduce process (part of) a single file: Mapping over many small files would be inefficient (recall the m * r copy operations!)
- Support batch oriented workloads (long streaming reads and large sequential writes). No data caching.

HDFS Design Choices (cont.)

- Applications must be aware of the characteristics of the distributed file system (only a subset of possible file operations is supported)
- Deployed in an environment of cooperative users. Assume data center environment where only authorized users have access. No strict enforcement of file permissions.
- Self-monitoring and self-healing mechanisms to robustly cope with common failure modes.

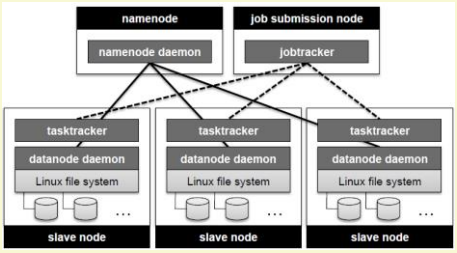
UNIVERSITY OF LEICESTER 43

Single-master design of HDFS

- Single point of failure: If the namenode goes down, the entire file system becomes unavailable.
- Why the single-master design?
  - Simple implementation, file system consistency
  - No data is ever moved through the namenode, so it does not become a bottleneck.
  - MTBF can be on the order of months, and a warm standby node can be used to quickly switch over in case of a failure

UNIVERSITY OF LEICESTER 44

Hadoop Cluster Architecture



UNIVERSITY OF LEICESTER 45

Nodes in a Hadoop cluster

- namenode: runs namenode daemon
- job submission node: runs job tracker (receives user submitted jobs, coordinates execution of mappers and reducers)
- slave node:
  - task tracker (execute tasks of a MapReduce job)
  - datanode daemon (serves HDFS data)

UNIVERSITY OF LEICESTER 46

MapReduce Algorithm Design (DITPM,Ch.3)

- Programming techniques to control execution and flow of data:
  - Use complex data structures as keys and values
  - Execute user-specified initialization and termination code in mapper and reducer
  - Preserve state in mappers and reducers
  - Control sort order of intermediate keys (to determine order of processing by reducer)
  - Control the partitioning of the key space

UNIVERSITY OF LEICESTER 47

Local Aggregation

- Reduce the number and size of key-value pairs that need to be shuffled from mappers to reducers
- Can be done using separate combiners or with "in-mapper combining"
- Recall:
  - Combiners are "mini-reducers" that process the output of mappers locally.
  - Hadoop does not guarantee that combiners are actually executed.

UNIVERSITY OF LEICESTER 48

Original mapper for Wordcount example

- **class** Mapper
  method MAP(docid a, doc d)
  for all term t in doc d **do**
  EMIT(term t, count 1)
- Outputs (word,1) for every word in the document.
- Can we change this to output only one key-value pair for each word in the document?

Aggregation on per-document basis

- **class** Mapper
  method MAP(docid a, doc d)
  H ← new ASSOCIATIVEARRAY // default val 0
  for all term t in doc d **do**
  H{t} ← H{t} + 1
  for all term t in H **do**
  EMIT(term t, count H{t})
- Outputs only one key-value pair per term.

Aggregation across multiple documents

- **class** Mapper
  method INITIALIZE
  H ← new ASSOCIATIVEARRAY // default val 0
  method MAP(docid a, doc d)
  for all term t in doc d **do**
  H{t} ← H{t} + 1
  method CLOSE
  for all term t in H **do**
  EMIT(term t, count H{t})
- This is the "in-mapper combining" design pattern

Advantages of "in-mapper combining"

- Advantages:
  - Provides control over when local aggregation occurs and how it takes place
  - Typically more efficient than using separate combiners (key-value pairs need to be generated)
- Disadvantages:
  - Potential for order-dependent bugs
  - Need sufficient main memory for intermediate results (can use "flush when full" to address this)

Mean Value Computation

- Task:
  - Given: Input of key-value pairs, keys are strings and values are numbers
  - For each key k, compute the mean of all values v such that (k,v) is an input key-value pair
- Need to take care when writing the Combiner as MEAN(1,2,3,4,5) ≠ MEAN(MEAN (1,2), MEAN (3,4,5))
- Idea: Use (sum,count) pairs as intermediate values

Mean Value Computation: Mapper

- **class** Mapper
  method MAP(string t, integer r)
  EMIT(string t, pair(r, 1))
- Mapper outputs one key-value pair for each input key-value pair

Mean Value Computation: Combiner

- **class** Combiner
  method COMBINE(string t, pairs [(s1,c1),(s2,c2)...])
  sum ← 0
  cnt ← 0
  for all pair(s,c) in pairs[(s1,c1),...] **do**
  sum ← sum + s
  cnt ← cnt + c
  EMIT(string t, pair (sum,cnt))

Mean Value Computation: Reducer

```
class Reducer
    method REDUCE(string t, pairs [(s1,c1),(s2,c2)...])
        sum ← 0
        cnt ← 0
        for all pair(s,c) in pairs[(s1,c1),...] do
            sum ← sum + s
            cnt ← cnt + c
        r_avg ← sum/cnt
        EMIT(string t, double r_avg)
```


Mapper with in-mapper combining

```
class Mapper
    method INITIALIZE
        S ← new ASSOCIATIVEARRAY  // default val 0
        C ← new ASSOCIATIVEARRAY  // default val 0
    method MAP(string t, integer r)
        S{t} ← S{t} + r
        C{t} ← C{t} + 1
    method CLOSE
        for all string t in S do
            EMIT(string t, pair(S{t},C{t}))
```


Building word co-occurrence matrices

- Application problem: For each pair of unique words in a text corpus, calculate the number of times that the pair occurs together in a sentence/paragraph/ document.
- Many applications in text mining, information retrieval, natural language processing, data mining, etc.
- Idea: Use complex key or value types

"pairs" pattern: pairs as keys

```
class Mapper
    method MAP(docid a, doc d)
        for all term w in doc d do
            for all term u in NEIGHBORS(w) do
                EMIT(pair(w,u), count 1)
class Reducer
    method REDUCE(pair p, counts [c1,c2,…])
        s ←0
        for all count c in counts [c1, c2, …] do
            s ← s + c
        EMIT (pair p, count s)
```


'stripes' pattern (stripes as values): Mapper

```
class Mapper
    method MAP(docid a, doc d)
        for all term w in doc d do
            H ← new ASSOCIATIVEARRAY
            for all term u in NEIGHBORS(w) do
                H{u} ← H{u} + 1
            EMIT(term w, stripe H)
```

- Mapper emits pairs of a term and an associative array with co-occurring terms

'stripes' pattern (stripes as values): Reducer

```
class Reducer
    method REDUCE(term w, stripes [H1,H2,…])
        Hf ← new ASSOCIATIVEARRAY
        for all stripe H in stripes [H1, H2, …] do
            SUM(Hf,H)
        EMIT (term w, stripe Hf)
```

- Here, SUM(Hf,H) adds H{t} to Hf{t} for all t.
- Each final key-value pair corresponds to a row of the word co-occurrence matrix

UNIVERSITY OF LEICESTER

61

Comparison between 'pairs' and 'stripes'

- Pairs algorithm generates more key-value pairs
- Stripes approach is more compact, but the value type is more complex, causing more serialization and deserialization overhead
- Stripes approach assumes that associative array fits into the main memory (scalability bottleneck)
- Both approaches can benefit from the use of combiners (or in-mapper combining)
  - More opportunities for local aggregation with stripes approach, due to smaller key space

UNIVERSITY OF LEICESTER

62

Speed Comparison: pairs vs stripes

- Corpus of 2.27 million documents, 5.7 GB in total
- Hadoop cluster with 19 slave nodes, each with 2 single-core processors
- Stripe approach much faster: 11 minutes compared to 62 minutes for pairs approach
- In pairs approach, 2.6 billion intermediate key-value pairs (reduced to 1.1 billion by combiners)
- In stripes aproach, 653 million intermediate key-value pairs (reduced to 28.8 million by combiners)

UNIVERSITY OF LEICESTER

63

Computing relative frequencies

- In the co-occurence matrix, m_{ij} is the absolute frequency of word w_j co-occurring with word w_i
- What if we want to know the relative frequency of word w_j co-occurring with word w_i?

$$f(w_j \mid w_i) = \frac{N(w_i, w_j)}{\sum_{w'} N(w_i, w')} = \frac{N(w_i, w_j)}{N(w_i, *)}$$

Here, $N(w_i,w_j)$ is the absolute frequency of word w_j co-occurring with word w_i

UNIVERSITY OF LEICESTER

64

Relative frequencies with stripes approach

- Using the "stripes" approach, computing relative frequencies is straightforward:
  - Reducer gets w_i as key and absolute frequencies $N(w_i,w_j)$ for all w_j as values
  - Add up all $N(w_i,w_j)$ to get total frequency $N(w_i,*)$
  - Then calculate relative frequency for each w_j by dividing $N(w_i,w_j)$ by total frequency $N(w_i,*)$
- Requires that associative array fits into memory

UNIVERSITY OF LEICESTER

65

Relative frequencies with pairs approach

- Reducer receives (w_i,w_j) as key and count as value
- To compute $f(w_j|w_i)$ we need $N(w_i,*)$.
- To compute $N(w_i,*)$ on the reducer, we need to:
  - Ensure all pairs (w_i,w_j) for any w_j are routed to the same reducer (using a partitioner)
  - Ensure the intermediate keys (w_i,w_j) are sorted first by w_i, then by w_j.
  - Build associative array in memory on reducer.

UNIVERSITY OF LEICESTER

66

Avoiding the memory bottleneck

- Is it possible to compute relative frequencies using the "pairs" approach without building the associative array on the reducer?
- Yes, by coordinating several mechanisms in MapReduce:
  - Mapper outputs extra key (word,*) with value 1 for every pair of co-occuring words
  - Use sort order to ensure that (word,*) values arrive before (word,word2) values at reducer

Example processing in reducer

key	values	Reducer action
(dog,*)	[12,14,27]	Calculate N(dog,*)=12+14+27=53
(dog,ant)	[1,2,3]	f(ant|dog)= (1+2+3)/53 = 6/53
…		
(dog,zebra)	[1,2,1,1]	f(zebra|dog)=(1+2+1+1)/53 = 5/53
(door,*)	[24,7]	Calculate N(door,*)=24+7=31
…		

- Partitioners ensures that all pairs (dog,...) are sent to the same reducer
- Reducer maintains state between calls

Summary: "Order inversion" design pattern

- By proper coordination, we can access the result of a computation (e.g. an aggregate statistic) before processing the data.
- Convert problem of sequencing computations into a sorting problem
- By controlling how keys are sorted, we can present data to the reducer in the required order
- This greatly reduces memory requirements of the reducer.

Order inversion for relative frequencies

- Summary of techniques we used:
  - Emit special key-value pair for each co-occurring word
  - Control sort order of intermediate key
  - Define custom partitioner to ensure that all pairs with the same left word are shuffled to the same reducer
  - Preserve state across multiple keys in the reducer

Secondary sorting

- MapReduce sorts intermediate results by key
- What if we want to sort by value?
- Example:
  - Input is dump of sensor records (ti,mi,ri) where ti is time stamp, mi is the sensor, and ri is the sensor reading.
  - We want to output for each sensor mi its sequence of sensor readings

Secondary sorting – first attempt

- The mappers could output key-value pairs with key mi and value (ti,ri).
- The sensor readings of the same sensor all arrive at the same reducer, but in arbitrary order.
- To sort the readings by time, the reducer needs to keep them all in memory until the last key-value pair is processed => memory bottleneck.

Secondary sorting – "value-to-key conversion"

- Mapper makes timestamp part of the key and emits pairs with key (mi,ti) and value ri.
- Partitioner ensures that all pairs (mi,...) are sent to same reducer
- Sort order ensures that pairs (mi,ti) are sorted first by mi and then by ti
- Reducer receives all sensor readings for a sensor in sorted order (but must preserve state between calls)

UNIVERSITY OF LEICESTER

Relational joins

- In relational databases we often want to join two tables on a column value, for example:
  - Table S: (student, module)
  - Table R: (module, convenor)
  - Join (written in SQL):
    Select * from S,R where S.module = R.module
- How can we do this in MapReduce (without a database)?

UNIVERSITY OF LEICESTER

Scenario with two tables

- Table S contains tuples (ki,si,Si), where ki is the column we wish to join on, si is the unique id in S, and Si is remaining data.
- Table T contains tuples (ki,ti,Ti)
- Example scenario:
  - S stores user profiles (ki is username)
  - T stores logs of online activity
  - Joining S and T allows an analyst to break down online activity by demographics

UNIVERSITY OF LEICESTER

Reduce-side join

- Map over both data sets
- Emit join column as intermediate key, and the tuple itself as intermediate value
- Since key-value pairs with the same key are routed to the same reducer, the reducer can join the tuples as required
- This works fine for a one-to-one join where each tuple from S is joined with at most one from T

UNIVERSITY OF LEICESTER

Example for reduce-side one-to-one join

Key	Values	Reducer action
k1	[(s2,S9),(t8,T7)]	Emit (k1,(s2,S9,t8,T7))
k2	[(t3,T12)]	-
k3	[(t4,T19),(s3,S4)]	Emit(k3,(s3,S4,t4,T19))

- Values from S and T can arrive in either order
- Keys with only one value do not produce a join result

UNIVERSITY OF LEICESTER

Reduce-side one-to-many join

- If one tuple from S is joined with many from T, they will arrive in arbitrary order at the reducer, but we would like the tuple from S to come first.
- We can again use value-to-key conversion to ensure that this happens:
  - Mappers emit ((ki,si),Si) and ((kj,tj),Tj)
  - Sort order is such that (ki,si) comes before (ki,tj)
  - Partitioner must ensure that all pairs (ki,...) arrive at the same reducer

UNIVERSITY OF LEICESTER

Reduce-side many-to-many join

- With approach from one-to-many join, tuples with key k1 arrive at the reducer in this order:
  - ((k1,s1), S1)
  - ((k1,s2),S2)
  - ....
  - ((k1,s10),S10)
  - ((k1,t1),T1)
  - ...
  - ((k1,t17),T17)
- Reducer keeps all (si,Si) for key k1 in memory,
- And then outputs joined tuple pairs for each (ti,Ti) that is processed

UNIVERSITY OF LEICESTER 79

Map-side join

- Assume that both S and T are sorted by the same key (the join key) and partitioned into corresponding blocks (e.g. as result of a previous MapReduce step)
- Each mapper reads the i-th block of S (local) and the i-th block of T (remote read), performs the join, and emits the joined tuples
- Reducers only pass the data on

UNIVERSITY OF LEICESTER 80

Memory-backed join

- In some cases, one of the two relations, say T, fits completely into main memory.
- Each mapper reads T into main memory (e.g. into an associative array) in the initialization method.
- When the mapper processes a tuple from S, it performs the join with all tuples from T and emits the joined tuples
- For large T, distributed storage in the main memory of several servers can be considered.

UNIVERSITY OF LEICESTER 81

Summary of design patterns seen so far

- In-mapper combining: perform local aggregation in the mapper
- "pairs" and "stripes": keep track of joint events or of all events that co-occur with the same event
- Order inversion: convert sequencing of computations into a sorting problem
- Value-to-key conversion: move part of the value into the key and exploit MapReduce framework for sorting

UNIVERSITY OF LEICESTER 82

Useful MapReduce programming techniques

- Construct complex keys or values that bring together the data necessary for a computation
- Execute user-specified initialization and termination code in mapper or reducer
- Preserve state across multiple inputs in mapper and reducer
- Control sort order of intermediate keys
- Control partitioning of intermediate key space

UNIVERSITY OF LEICESTER 83

Security and the private cloud (CAYS, Ch. 4)

- Cloud computing enables startups and small and medium-size enterprises to launch services without huge prior investment
- Large enterprises tend to be more reluctant to move to the cloud due to security concerns
- Alternatives to public clouds: private clouds, and virtual private clouds

UNIVERSITY OF LEICESTER 84

Security concerns slowing cloud adoption

- Users spanning different corporations and trust levels interact with the same set of compute resources
- Public cloud offerings are exposed to more attacks
- Many nations have laws requiring SaaS providers to keep customer data and copyrighted material within national boundaries
- Some businesses may not like the ability of a country to get access to their data (e.g. via USA PATRIOT Act)

UNIVERSITY OF LEICESTER 85

Major cloud data center security

- Physical security
  - House data centers in nondescript facilities (security by obscurity) with extensive setback and military-grade perimeter control berms
  - Physical access is strictly controlled (security staff, video surveillance, intrusion detection systems)
  - Authorized staff use two-factor authentication at least three times to access data-center floors
  - Log and audit all access by employees

UNIVERSITY OF LEICESTER 86

Physical security

- Perimeter security (razor wire)
- Biometric authentication (palm reader)
- Access control (man trap)
- Most public cloud providers have SAS 70 Type II certification

UNIVERSITY OF LEICESTER 87

Access control measures

- Typical access control for initial sign-up:
  - Billing validation: check billing address of credit card used for payment
  - Identity verification via phone (out of band): generate PIN through browser, enter via phone
  - Create sign-in credentials (strong passwords, ideally multi-factor such as RSA SecurID)
- Use secret authentication key for each API call (digital signature)

UNIVERSITY OF LEICESTER 88

Cloud data security

- In many ways, the cloud is more secure than most data centers within an organisation:
  - Centralizing data in the cloud means less leakage than distributing it all over the organization
  - For centralized data it is easier to monitor access and usage
- But: If there is a breach in the cloud, centralized data can mean a more comprehensive and damaging data theft

UNIVERSITY OF LEICESTER 89

Cloud versus local data centers

- If an incident occurs, the cloud provides a faster and more comprehensive means of response (state-of-the-art intrusion detection, fast acquisition of forensic data, short downtime)
- Cloud providers are providing more and better built-in verification (e.g. MD5 hash on all stored S3 objects in Amazon's S3)
- Local data centers will become more expensive and less reliable compared to the biggest cloud providers
- Cloud becomes cheaper, more secure, more reliable

UNIVERSITY OF LEICESTER 90

Network security in the cloud

- All public clouds provide a firewall (which allows restricting traffic by protocol, service port, or source IP address (block))
- Many prominent public cloud providers have strong capabilities of defending against DDoS (distributed denial of service) attacks
- User applications run on virtual servers, and the host OS can prevent address spoofing and packet sniffing

UNIVERSITY OF LEICESTER 91

Data storage security

- Cloud provider's disk virtualization layer can guarantee that data is never exposed to another user
- Still, it is good practice to store data in an encrypted file system on top of the virtualized disk device
- Access control lists can limit access to storage containers
- Data and control messages can be transferred securely via SSL-encrypted access to storage API

UNIVERSITY OF LEICESTER 92

Private cloud

- A computing architecture that provides hosted services to a specific group of people behind a firewall.
- A private cloud uses virtualization, automation, and distributed computing to provide on-demand elastic computing capacity to internal users.

UNIVERSITY OF LEICESTER 93

Principles of cloud computing revisited

- Principles that remain the same as for public cloud:
  - Virtualization: high utilization of assets
  - Elasticity: dynamic scale without CAPEX
  - Automation: build, deploy, configure, provision, move without manual intervention
- Principles that do not necessarily hold:
  - Pooled resources (available to any subscribing users)
  - Metered billing (pay for what you use)

UNIVERSITY OF LEICESTER 94

Primary considerations for private cloud

- **Security:** applications that require direct control and custody over data for security or privacy reasons
- **Availability:** applications that require access to a defined set of computing resources that cannot be guaranteed in a shared resource pool environment
- **User community:** Organization with a large number of users who need access to utility computing resources
- **Economies of scale:** Existing data center and hardware resources that can be used, and ability to purchase equipment at favourable pricing levels

UNIVERSITY OF LEICESTER 95

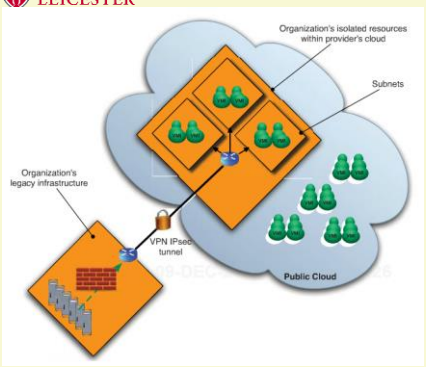
Concerns about deploying a private cloud

- Private clouds are small: Few corporate data centers see anything close to the type of volume seen by cloud-computing providers
- Legacy applications don't "cloudify" easily
- On-premises doesn't necessarily mean more secure
- Do what you do best – private clouds will always be many steps behind the public clouds (whose providers constantly optimize how they operate)

UNIVERSITY OF LEICESTER 96

Virtual private cloud (VPC)

- A VPC is a secure and seamless bridge between an organization's existing IT infrastructure and a provider's public cloud
- Idea:
  - Organisation that has its own IT infrastructure adds additional web-facing servers to an application when the traffic exceeds the on-premise capacity
  - Back-end, database servers, authentication servers etc. remain in organisation's own data center



CAYS p. 93

Virtual private cloud security

- An organisation's existing infrastructure is connected to a set of isolated cloud-computing resources via a virtual private network connection
- The organisation's existing management capabilities and security services extend to the virtual private cloud and protect information there in the same way as in their own data center (including firewalls, intrusion-detection systems, etc)

Usage scenarios of virtual private clouds

- **Expanding corporate applications into the cloud** to reduce total cost of ownership (email systems, financial systems, CRM applications, etc.)
- **Elastically scaling websites in the cloud**: Temporarily add web servers when traffic load exceeds on-premises capacity ("cloudbursting")
- **Disaster recovery**: Periodically back up mission-critical data to the VPC, and in the event of a disaster, quickly launch replacement compute capacity in the cloud to ensure business continuity

Cloud Security - Summary

- Security remains the biggest fear factor for larger organisations considering a move to the cloud, but there are many features and mechanisms that can (at least partially) address these concerns in the public cloud
- For organisations with enough scale, buying power and expertise, private clouds offer advantages of increased control, predictability and security
- Virtual private clouds represent a hybrid model that can be considered as an alternative