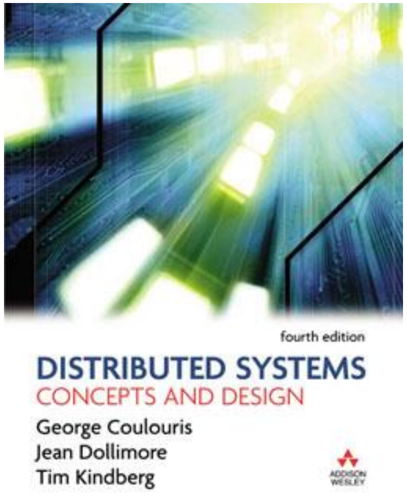
Slides for Chapter 2: Architectural Models



From Coulouris, Dollimore and Kindberg
Distributed Systems:
Concepts and Design

Edition 4, © Pearson Education 2005

Chapter 2 Coulouris et al.

- Prepared by Clark Elliott, DePaul University

- Some slides from Dr. Rajkumar Buyya, University of Melbourne. http://www.gridbus.org/652/LectureSlides.html

Architectural Models

Software Layers

System Architectures

Interfaces and Objects

Design Requirements

Architectural Models – Intro [1]

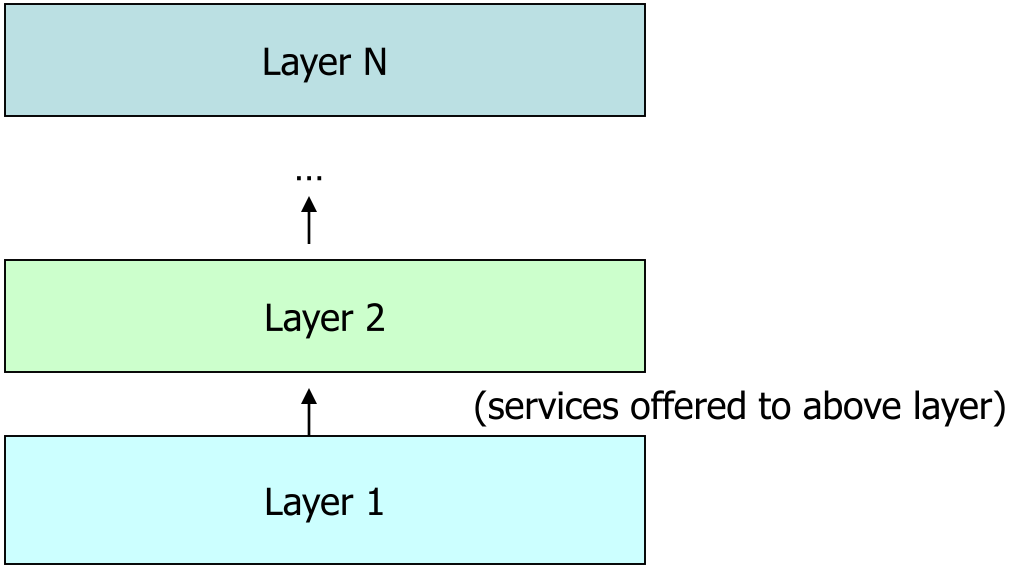
- The architecture of a system is its structure in terms of separately specified components.
  - Its goal is to meet present and likely future demands.
  - Major concerns are make the system reliable, manageable, adaptable, and cost-effective.
- Architectural Model:
  - Simplifies and abstracts the functions of individual components
  - The placement of the components across a network of computers – define patterns for the distribution of data and workloads
  - The interrelationship between the components – ie., functional roles and the patterns of communication between them.

Architectural Models – Intro [2]

- Architectural Model - simplifies and abstracts the functions of individual components:
  - An initial simplification is achieved by classifying processes as:
    - Server processes
    - Client processes
    - Peer processes
      - Cooperate and communicate in a symmetric manner to perform a task.

Software Architecture and Layers

- The term *software architecture* referred:
  - Originally to the structure of software as *layers* or modules in a single computer.
  - More recently in terms of *services* offered and requested between processes in the same or different computers.
- Breaking up the complexity of systems by designing them through layers and services
  - Layer: a group of related functional components
  - Service: functionality provided to the next layer.



Introduction

- E.g. client/server vs. peer-to-peer
- Open systems vs. proprietary systems
- Layered approach vs. high efficiency application coding

Have to address…

- No global clock
- Communication is via messages (without clock!)
- Delays, and high failure
- Vulnerable to attacks on message system
- Remote administration problems
- Massive scale-up problems
- Classic problems like the server-binding problem

Falacies

- Eight fallacies of distributed computing:
    1. The network is reliable.
    2. Latency is zero.
    3. Bandwidth is infinite.
    4. The network is secure.
    5. Topology doesn't change.
    6. There is one administrator.
    7. Transport cost is zero.
    8. The network is homogeneous.

Reason about three dimensions

- A. Study Interaction of processes, including performance [include all…?!]
- B. Study of failures
- C. Study security issues

- But not explicitly efficiency, ease of systems development, openness, expansion, elegance, productivity, robustness, practicality, costs

Reason about three dimensions

- A. performance
- B. reliability
- C. security

… of systems.

- But not explicitly efficiency, ease of systems development, openness, expansion, elegance, productivity, robustness, practicality, costs

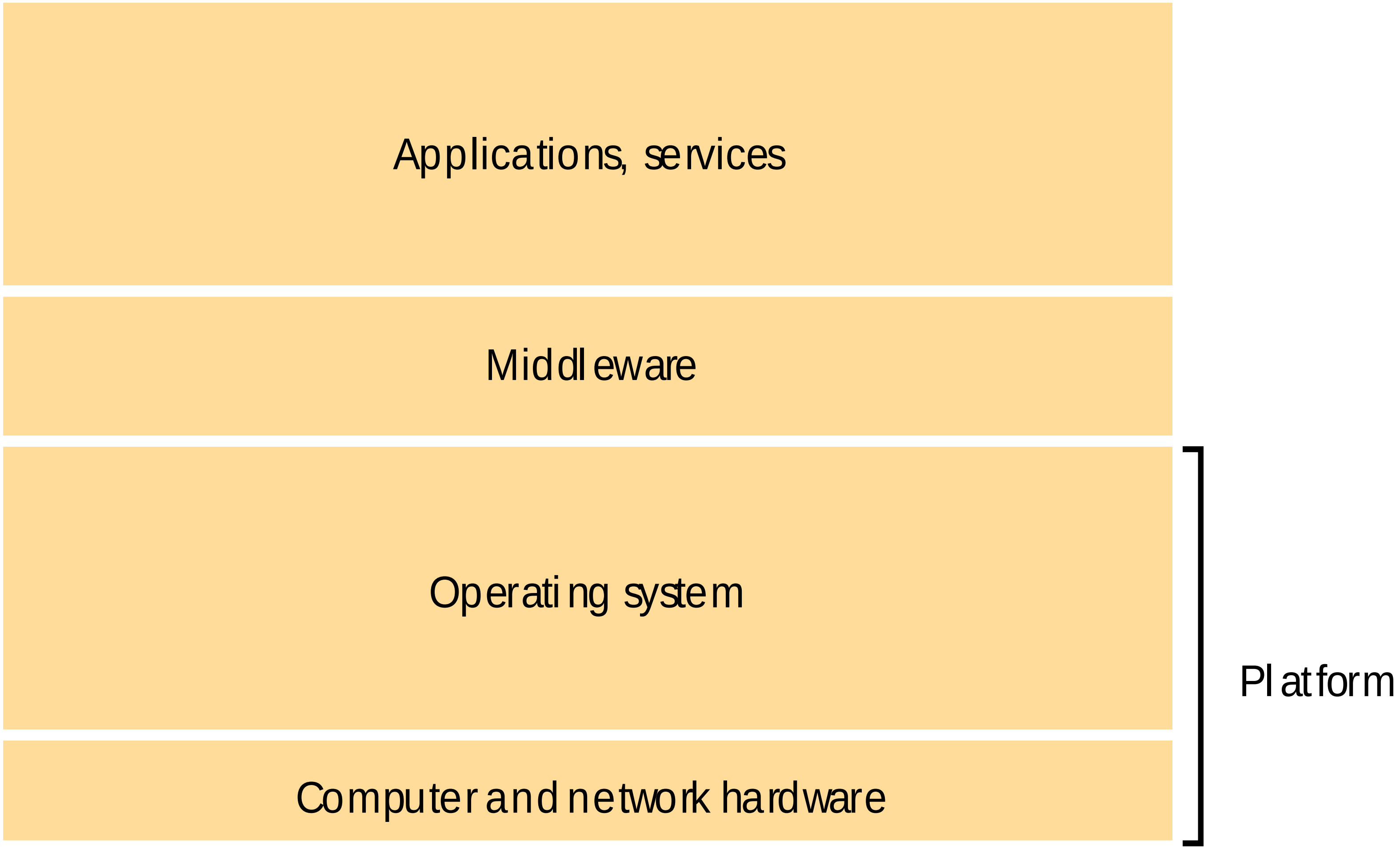
Architectural Models

- Layers
- Cleint/server
- Peer-to-peer
- Services
- Mobile code
- Mobile agents
- Network computers
- Interfaces

Service Layers

- OSI model – abstraction is good. But note, TCP/IP does not use OSI model
- Virtual machines. Java, VM
- Might need some DS rough approximation of global time. Software depends on this. So layer provides it. But under the hood could be NNTP, or hanging a GPS receivers out the window.

Figure 2.1
Software and hardware service layers in distributed systems



Layers

- Hardware, microcode, bit level binary code and assembly. OpSys privlidged and user operations. [Virtual OpSys], shell wrapper, [application VM], Middleware, application
- Middleware provides homogenous interface for application development. May handle, e.g., message passing, buffering, reliability, security.

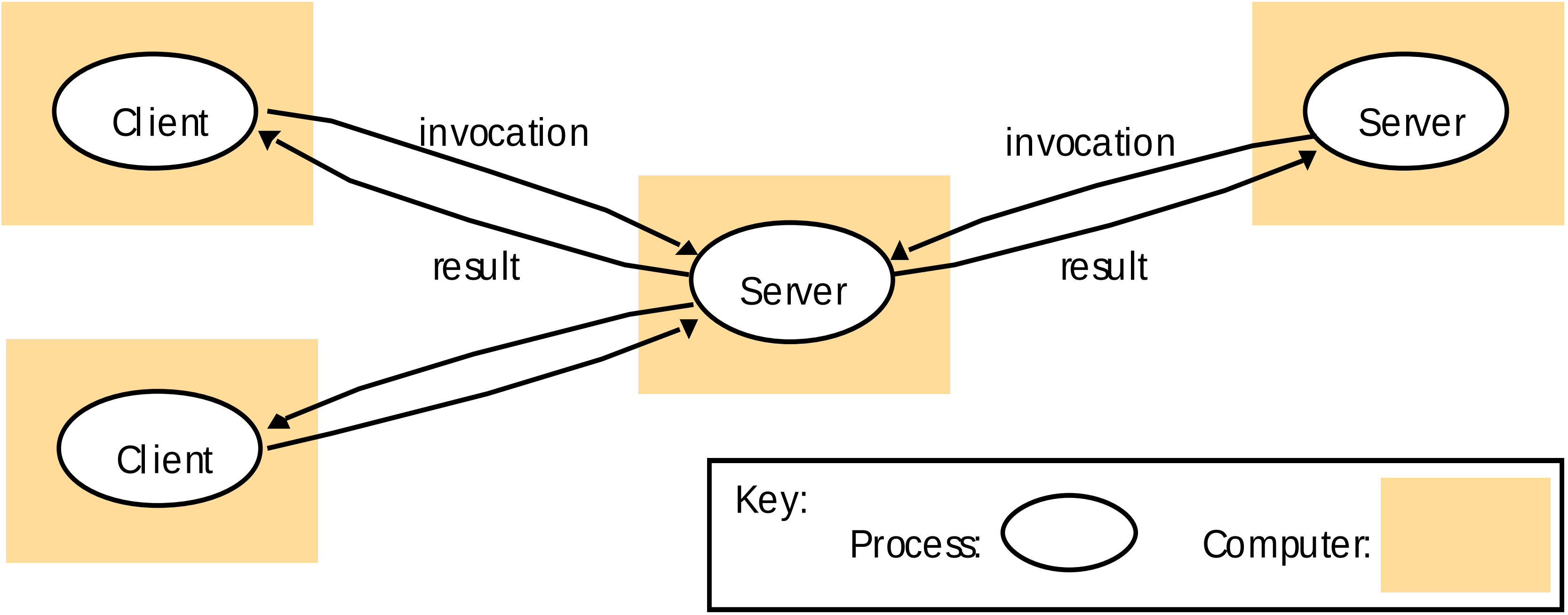
Middleware

- RPC, RMI, DCE, sockets, CORBA, .net
- Always a compromise: ease of use vs. reliability, efficiency, and maintainability of the middleware.
  - Example. Very large mail file. *Application* probably needs to keep a local copy to resend if there are major network problems, and not just rely on TCP.

Where is the knowledge?

- Saltzer, et al., 1984
- May need application level knowledge to perform efficient checks.
- E.g., cannot lose one bit of 1K of critical encrypted backup of central business keys, vs. 15M random bits of non-critical image file. Middleware would have to be (a) inefficient OR (b) allow critical failures.

Figure 2.2
Clients invoke individual servers



Client / Server

- Servers may also be clients of other servers.
- Search engine: clients connect to server, but background process has collected info by repeatedly connecting as a client to servers all over the world.

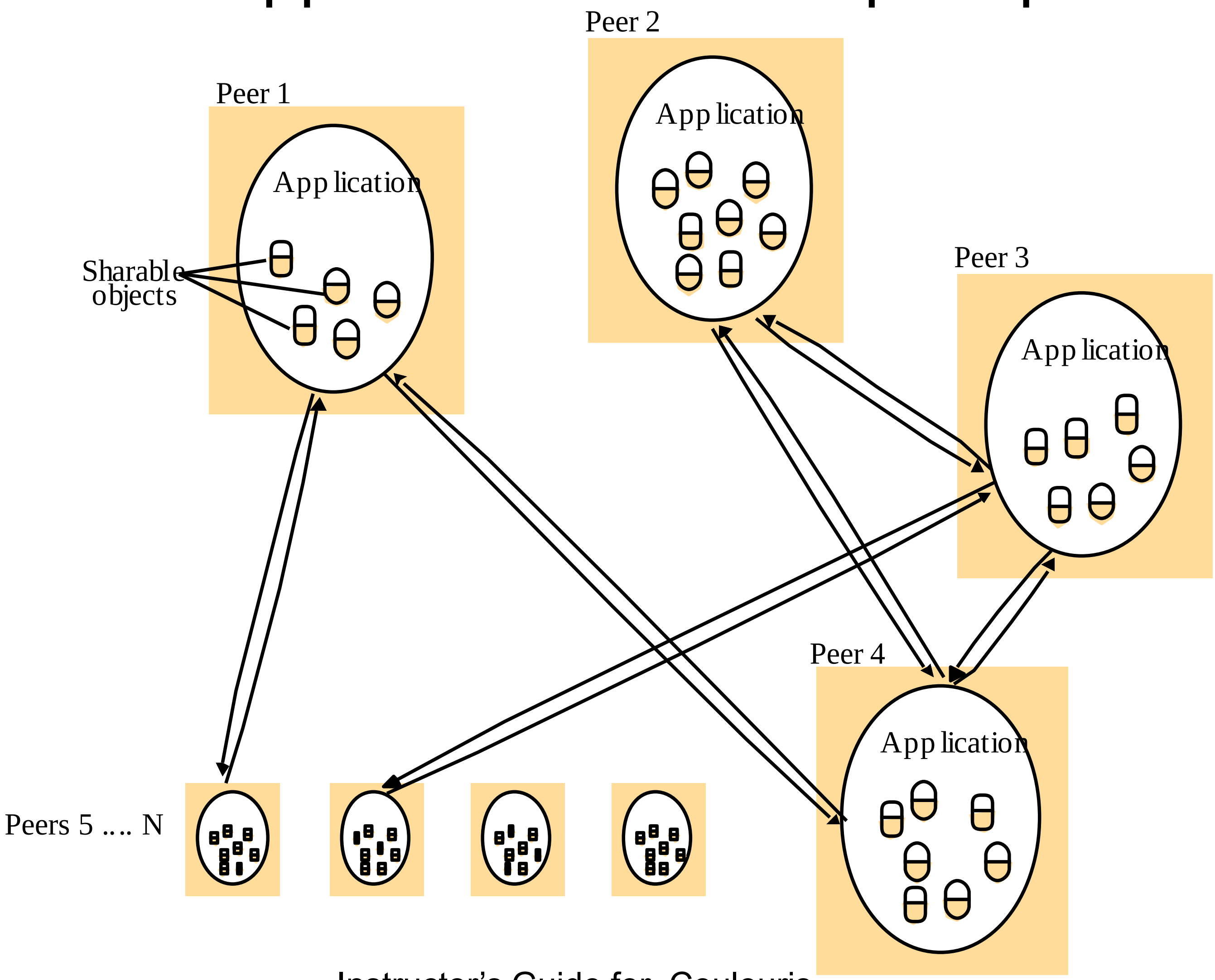
Peer-to-peer

- Claim is made that client/server does not scale well, but, of course, this is nonsense.
- It does not scale *automatically*, but instead requires the expansion of network, server machines, etc.
- Peer to peer e.g., can take advantage of massive wasted computer power sitting on desktops. Napster, Kazza, Gnutella, BitTorrent

Peer-to-peer

- Forces nodes to contribute.
- Same functional characteristics at most, if not all, nodes
- No central authority or failure point.
- Can provide anonymity
- Balances workload across nodes
- Large address space via GUIDs
- Best for static data because secure hash discourages malicious nodes.

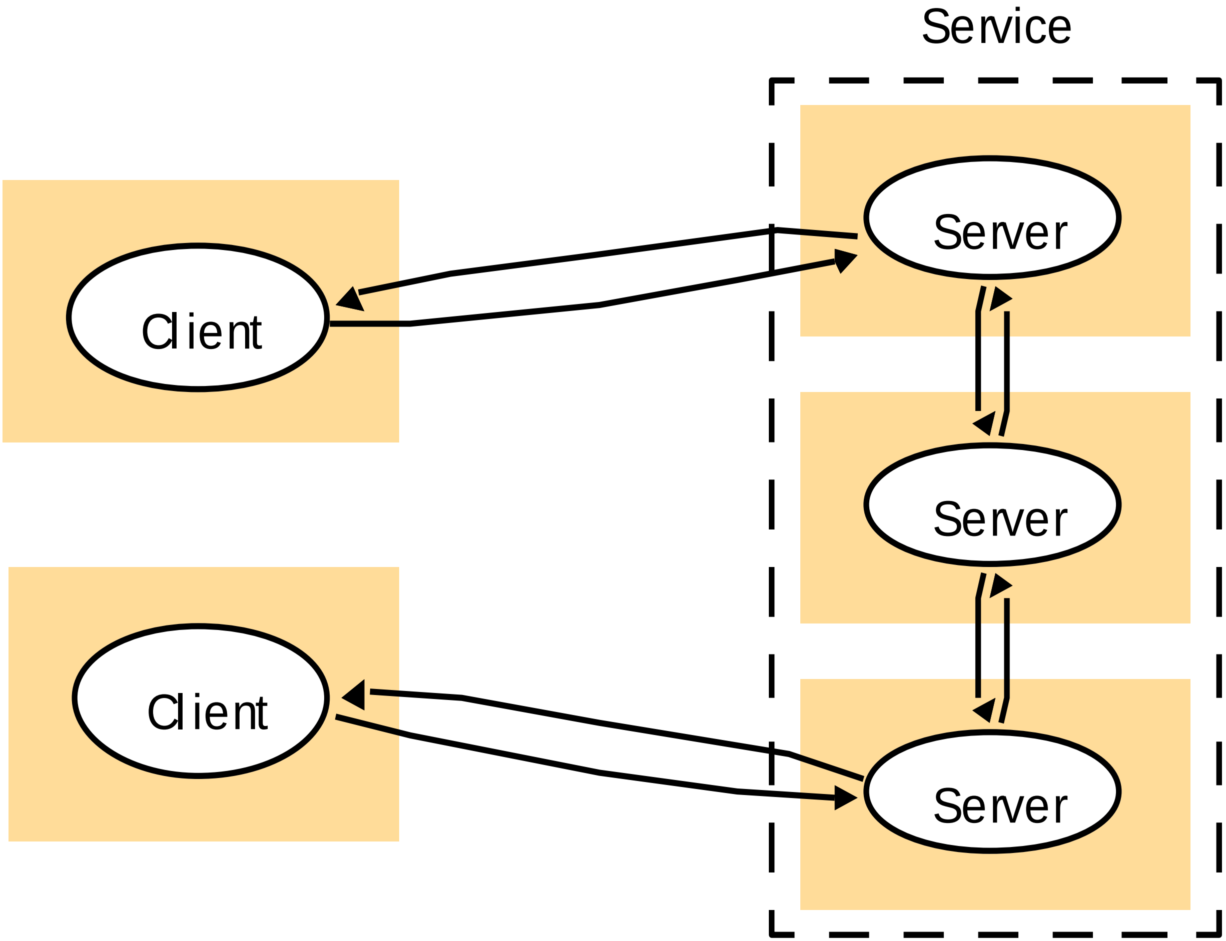
Figure 2.3
A distributed application based on peer processes



Services

- Logically decoupled from the endpoint (ip address / port) of any particular server, but instead clients subscribe to a *service* that might be provided by *many* servers.
- Might have redundancy (replication), or specialization (partitioning) so that multiple servers participated for single client.

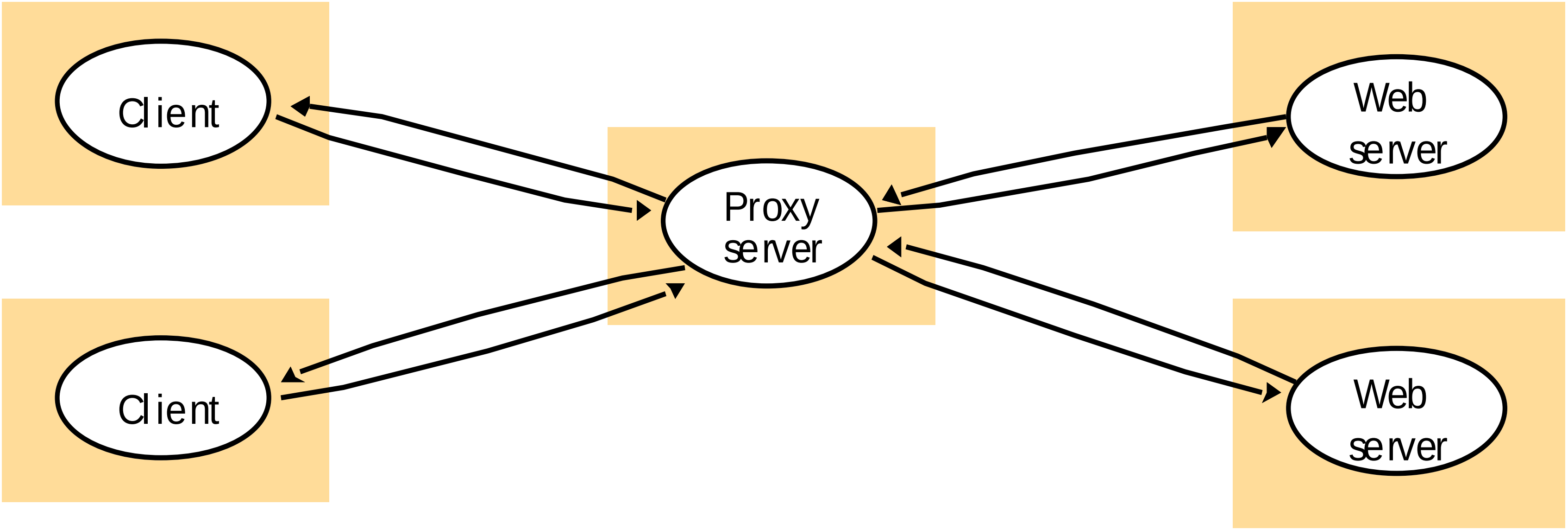
Figure 2.4
A service provided by multiple servers



Proxy servers

- Extra server level that acts as a *filter* for requests
- Caching for efficiency, firewall for protection, route to redundant servers for availability and ease of maintenance, mobile IP routing.
- Typical: caching of popular web pages

Figure 2.5
Web proxy server



Mobile Code

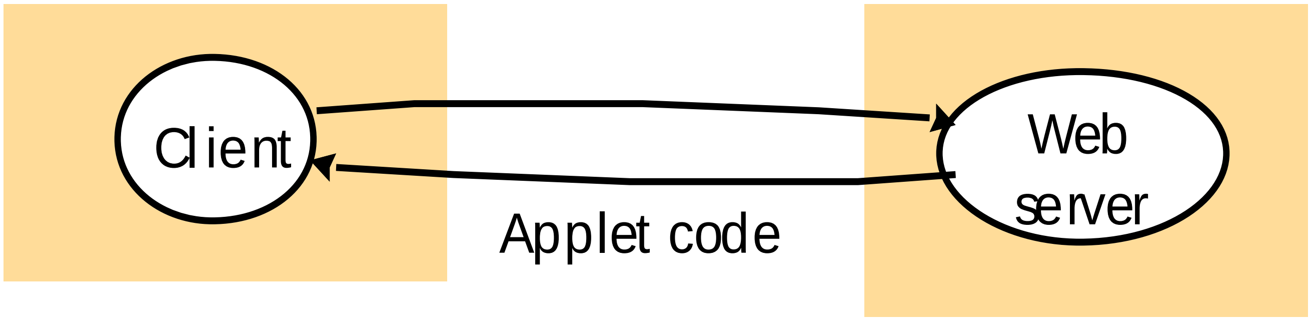
- Code moves from one location to another before or during execution
- Strong mobility – running processes moved while in progress
- Weak mobility – always started from the beginning after moving. Java applets.
- Code may communicate with server

Mobile code

- Push: server initiates updates as needed. This can be tricky: when client connects there is startup delay *OR* Client must stay be connected to get updates.
- Security risk at client so use of “sandbox” that restricts access to local machine. Requires extra layer and may limit, .e.g, use of local, native, multimedia libraries.

Figure 2.6
Web applets

a) client request results in the downloading of applet code

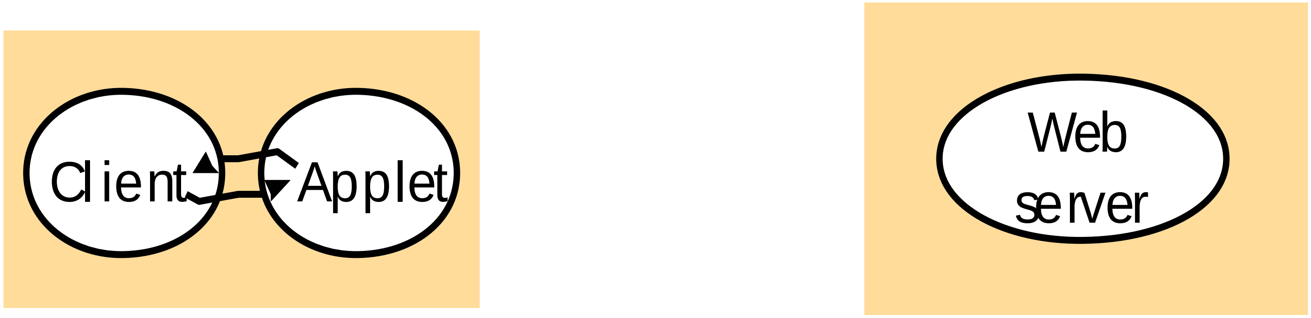


Client

Web server

Applet code

b) client interacts with the applet



Client

Applet

Web server

Mobile agents

- Processes, code libraries, and *state* that move from one location on a network to another to perform work on a remote location’s behalf.
- Interact with local resources, such as databases, other processes, and even users.
- Local invocations are cheaper than remote ones so may lead to efficiency.
- Different programming design
- Security and negotiation are complex because always two different stakeholders, host / agent

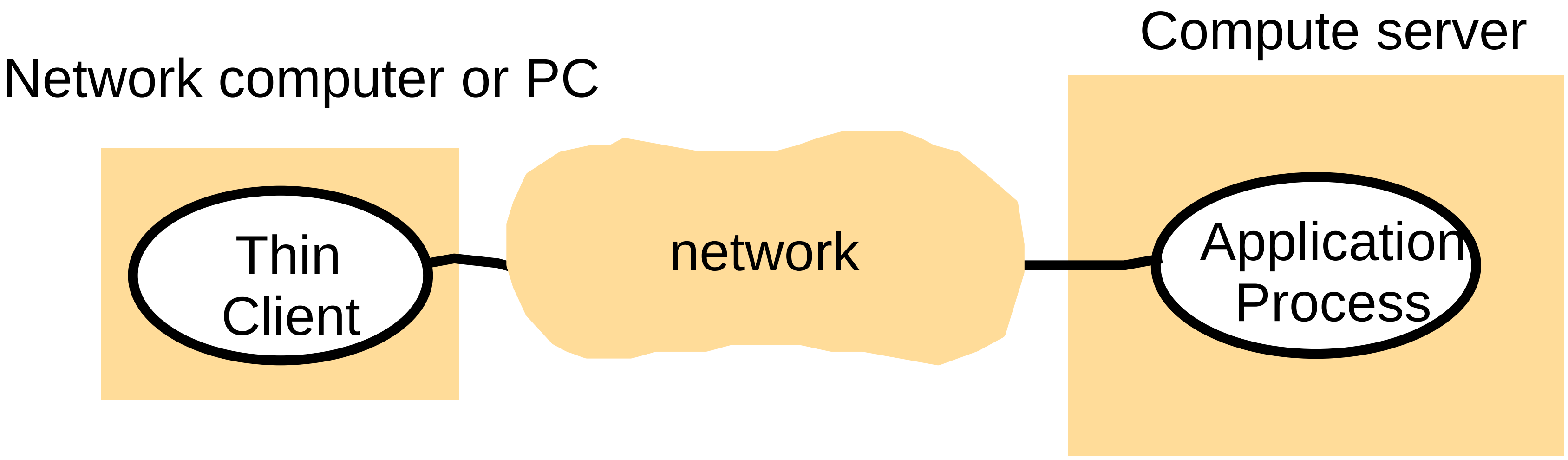
Mobile agents

- One generally unexplored area is using the idle resources of a nework
- Security problems can work both ways: both to the host, but also to the agent.
- Not hugely popular. Note that web crawlers “look” like agents, but are simply a series of local requests to various servers.

Network computers

- Manage files and other critical resources in a central location, correctly, under opsys control. Make transparent to users.
- Very thin client
- Local disk is primarily for caching
- Remote desktops are related, X-11 runs local display *server* on the user machine
- Lucent's plan 9: http://www.ecf.toronto.edu/plan9/plan9faq.html

Figure 2.7
Thin clients and compute servers



Mobile devices

- Wave of the future. Miniaturization of CPUs, lower power requirements, improved battery technology, wireless technology
- WiFi – term does not mean anything, just catchy
- In general DHCP and discovery of local resources is enough as a collection of *clients*

Mobile IP

- If server push is required, or others are using local resources that are mobile, then DHCP will not work
- IP address are subnet-based, for routing purposes, geographically fixed.
- Mobile IP uses home agent (HA) and foreign agent (FA).
- HA acts as a proxy to reroute all packets through a tunnel.
- Mobile-IP-enabled senders can communicate directly thereafter

Mobile devices

- Fundamental problem is second-class citizen / or require continual update and propagation of IP device locations
- Goal: spontaneous reconfiguration to interact with local environment, local service discovery

Interfaces and Objects

- Presentation of interfaces that de-couple implementation (and perhaps location) of services, objects, data, are fundamental to modern DS thinking
- Extension of OO-design interfaces, but also of non-OO client/server.
- IDLs, GUID address-space.

DS-performance -- responsiveness

- Interactive times for users, and interaction times to clients
- More layers is easier to understand and maintain, but degrades turnaround time
- Gives rise to caching
- Web response times, real-time transactions vs. batch processing.

Throughput

- Amount of time it takes to get a given amount of work done. Batch legacy.
- Good throughput does not necessarily mean good responsiveness – may come in bursts.

Quality of Service

- Reliability, security, timeliness, adaptability
- QoS big problem for MultiMedia streams
- Reliability and security are self explanatory but may require yield to some compromises – e.g., sacrifice security for timeliness.
- Adaptability for quick system response to failures or serendipity; new components, components that move.
- May require caching

Multi-media QoS

- Multimedia streams – especially audio – must arrive in a timely manner. Obviously much emphasis on this currently
- Larger buffers. Less security. Dedicated protocols. IPv6 priority – but then why not send everything marked as video?

Caching and replication

- Duplicate data used to be the cardinal sin of computer science.
- Modern web and network use has required relaxation of this constraint
- Semantics between real world and the symbolic representation of it should match.

Tom’s bank account

- Tom has a box in the bank’s vault with money in it.
- When Tom takes money out, it is no longer in the box; when he puts money in to box there is more in the box.
- Symbolic representations of the amount in the box must be consistent. If data resides as only one set of bits this is easy to enforce.

Many copies

- What happens when the virtual Tom’s box lives on several replication servers, in a proxy cache, in a client memory cache, in a client disk cache? This allows Tom to have $204///36 in his box.

Caching requires intelligence

- Network objects (e.g., pages) can be large or small
- Can be used often or seldom
- Might *absolutely require* current copies (e.g., stock prices for real-time trading)
- Might not require current copies (images of faces)
- Might require version updates of a *collection* of resources

Caching intelligence…

- Might be “owned” by server, or by client, each of which is responsible for determining update status. Push/pull.
- Typically little or no semantic intelligence in caching choices
- Many annoying update processes (cached versions of client software) e.g., adobe times fifty, security software, computer grinds to a halt. No intelligence.

Background update process

- When the network lines and local / remote CPU resources are not being used, can update resources more efficiently

Large systemic security problem

- Caching is not only a structural problem for object semantics but also for security
- Many application programmers write for the web.
- Store critical information in Session variables which are transmitted back and forth, and are saved on local disks distributed far beyond control of server sites that wrote the code.

Web (non) security…

- Data is cached either in known, vulnerable, locations all over the world, or…
- Application programmers are trusted to devise their own site-specific security measures. Not their area of expertise.

Discover card goofiness

- Password for discover account was sent to me as clear text in email after a problem, and not at my request.
- Exposed to internet attack, exposed to attack on my unix machine, exposed to mail process attack.
- Was original password I typed in a month earlier, which means what…?!!
- Argued for twenty minutes with the “top security guy” who was an idiot.

AT&T Security Evil / Morons

- To: [...]@yahoo.com
- Subject: AT&T Password Reset

- Dear Valued Customer,
  - Cellular Data Number is 858-xxx-nnnn
  - Your password is [xxxxxxxx]. Please use it when logging into your account via Settings on your iPad.
- Thank You,
- AT&T

Cryptographic Salt

- Random bit-string added to password before producing the encrypted version of the password. Stored on the server.
- Increases the randomness and the length of the password string.

Tricks and heuristics

- Web-caching “protocol”
- Provide expiration times for pages, estimated.
- Expiration time and server time are sent to browser with the page.
- On next request browser determines age of page as value between request and expiration, then makes a choice. No clock.

Fault tolerance

- System keeps running even in presence of hardware, and network, failures.
- Faults in software – this seems a rather provocative statement! Nonsense?
- Achieved through redundancy and matching control software, protocols, design rules.
- High overhead for such systems, and constrains development which is both difficult and expensive.
- Replace the air traffic control system?

Fundamental models

- Difficult to make out what the authors intend, structurally, in this section besides applying modeling logic to architectures
- Model: system features and invariants are made explicit; constraints are studied.
- Allows us to make generalizations about strengths and weaknesses of comparative architectures
- Mathematical proofs of properties

Three dimensions

- A. Interaction / performance (processes)
  - Communication channels
  - Clocks and timing
  - Variants: synchronous, asynchronous
  - Event ordering
- B. Failure / reliability
  - Ommision
  - Arbitrary
  - Timing
  - Masking failures
  - One-to-one communication reliability

- C. Security
  - Protecting objects
  - Process and interaction security
  - The enemy
  - Response to threats
  - Other threats
  - Threat *models*

Basic performance model

- Truly distributed algorithm(s)
- Centralized control in a distributed system

Communication channels

- Latency – time between the start of a message send, and the start of the receipt at the other end. (Time to send an empty packet.) For acknowledgments requires roughly 2x latency value.
- Affected by distance, reliability of network (resends), layers of software, load on network and OpSys, security overhead (e.g., encrypting messages), proxies, superservers and service startup, load-balancing servers.

- Satellites have high throughput, but high latency values – so bad for interactive editing, but good for downloads

Bandwidth

- Maximum number of bits that can be transmitted over the network in a given period of time.
- Affected by overhead bits, errors, sharing loads with all processes.
- May vary greatly. Routing changes and loads are all dynamic.
- Measured as an *average*

Jitter

- Variation in time to deliver individual messages
- Difference between maximum and minimum times.
- Very important to audio streams, and streams that combine video and audio.

Clocks

- Individual computers each have their own clocks.
- Typically billions of cycles different.
- Local processes read local clocks.
- Perfect external reference clock – even if it existed, the time cannot be read in except through millions of instructions.
- So – time stamps are not very meaningful in a distributed system. (Remember the cached data?)

- Can use GPS attached to a computer, but not all boxes have radio access – but sending on local network is subject to message delay. Only good for mSec anyway (when a million instructions might be executed).
- Network time protocol. Other schemes to set clocks approximately.

Two variants of interaction

- Synchronous distributed systems. Hadzilacos and Toueg [1994]
  - Steps in the algorithm a process is executing have upper and lower time bounds.
  - Each process in the distributed system has a local clock with constraints on maximum drift from an externally agreed clock.
- Not very practical, but theoretically useful, and can be used in parts of systems. E.g. might solve parts of server binding problem.

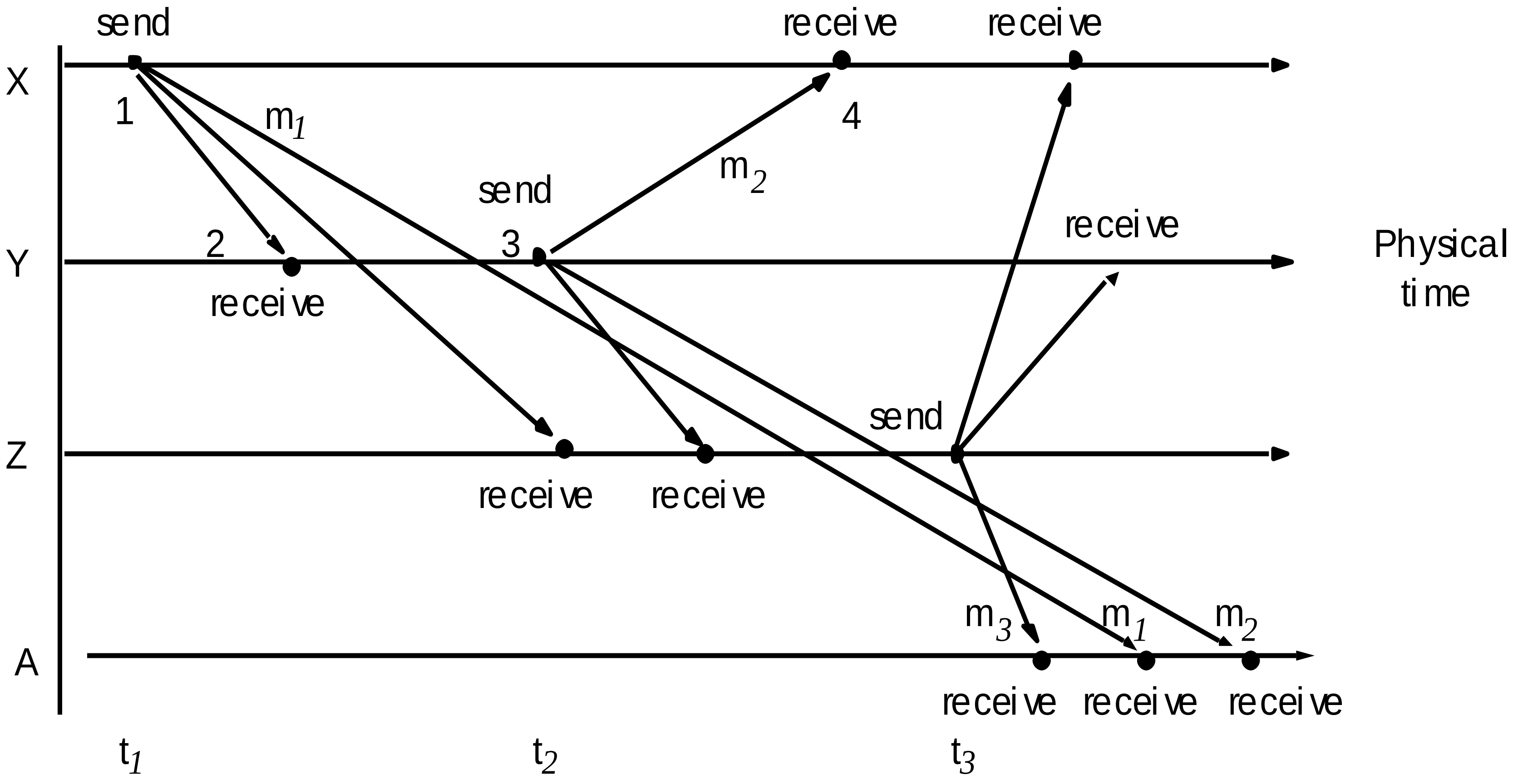
Async distributed systems

- No bounds on:
- Process execution speed. Each step in algorithm might take *any* amount of time.
- Message delay. *Any* amount of time to deliver without theoretical failure.
- Clock drift. No agreement whatsoever on clock drift from external “master” clock.
- Useful theoretically, but practically is always constrained.

Event ordering

- Can obviate the need for global clock, order is enough – *logical* time. Lamport time stamps.
- Even without clocks we can reason that, e.g., a message, if received, is always received *after* it is sent; and also that replies are always sent *after* a message is received. Ordered *sets* of events.
- Thus we have T1, T2, T3, T4 in fig 2.8, and also other ordered sets

Figure 2.8
Real-time ordering of events



Reliability and Failure modeling

- Omissions – missing in action
- Arbitrary – wrong data, wrong algorithm
- Timing – delays beyond assumed limits
- Masking – transparency fails
- ~Process-to-process communication fails

Process Omissions – crash/hang

- Process crashes (stops execution) or hangs (looping, waiting, might be cleared to restart)
  - Timeouts suggest server binding problem – dead or just slow?
  - Can we assume elegant halt (fail-stop)? If so, other procs will be aware and can plan.
  - In general this is a hard problem

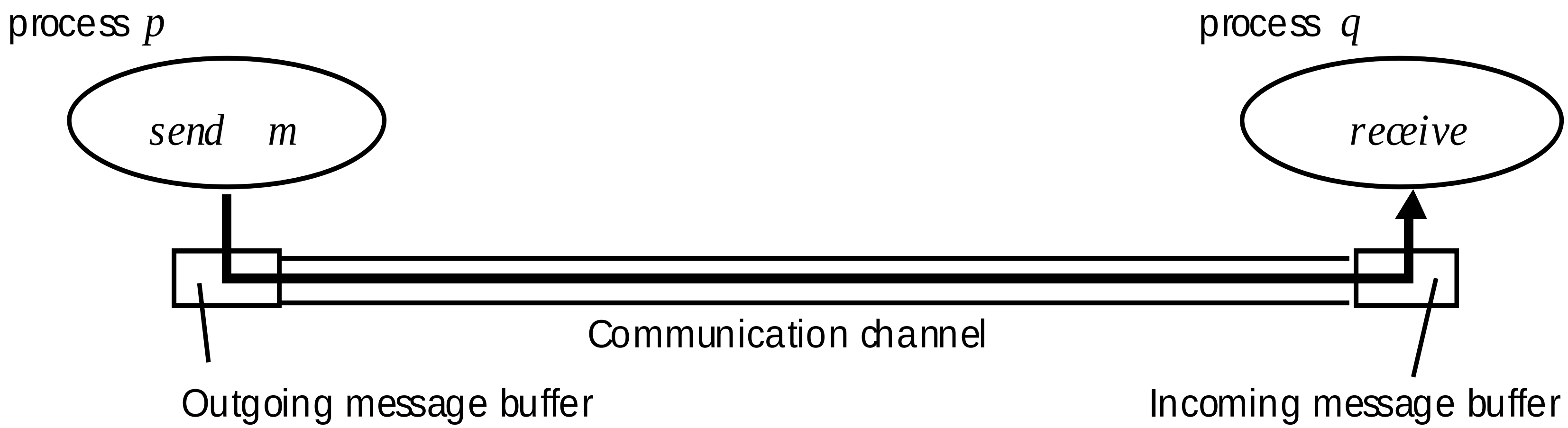
Buffers

- Locations in memory shared between processes.
- Typically circular, with two pointers
- Processes can read the writes of other processes.
- Buffers can get full – so writer blocks
- Buffers can get empty – so reader blocks

Communication omission – lost message

- Send-omission failure between sending process and send buffer
- Channel-omission failure from send buffer to receive buffer
- Receive omission failure between receive buffer and receiving process

Figure 2.9
Processes and channels



Pepperland message omission

- Camps send regular messengers to say they are still O.K.
- If messenger is missing in sync. system then know of failure.
- If messenger arrives then know *were* o.k. at time of message send.
- If messages can fail, no agreement guaranteed. Async. system mimics failures.

Arbitrary failures

- Proc sets wrong values in data, or return values
- Follows wrong algorithm – omits or adds steps
- Corruption of messages, or message headers
- Can be trivial, or impossible, to detect.

Figure 2.10
Omission and arbitrary failures

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer ne arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send*, but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Timing failures

- In synchronous systems we make use of assumptions on upper and lower bounds on message delivery time.
- In practice we make unwarranted assumptions about synchronization in otherwise asynchronous systems
- If a component of a system exceeds the bounds, then system invariants have been violated, and behavior is undetermined.
- Very important for Multimedia, and real-time process control, systems.

Figure 2.11
Timing failures

Class of Failure	*Affects*	*Description*
Clock	Process	Process’s local clock exceeds the bounds on rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message’s transmission takes longer than stated bound.

Masking failures

- Distributed systems have transparency goals, including the masking of failures.
- Some processes can be re-started, messages resent, backup servers used
- A corrupted message might be detected (e.g. CRC) and resent, changing a difficult arbitrary failure into a manageable omission failure.

One-to-one process communication

- Lower levels of a communication protocol may be unreliable, but with reliable process communication built on top of it.
- Validity – any message in outgoing buffer eventually gets delivered to incoming buffer.
- Integrity – identical message arrives exactly once.

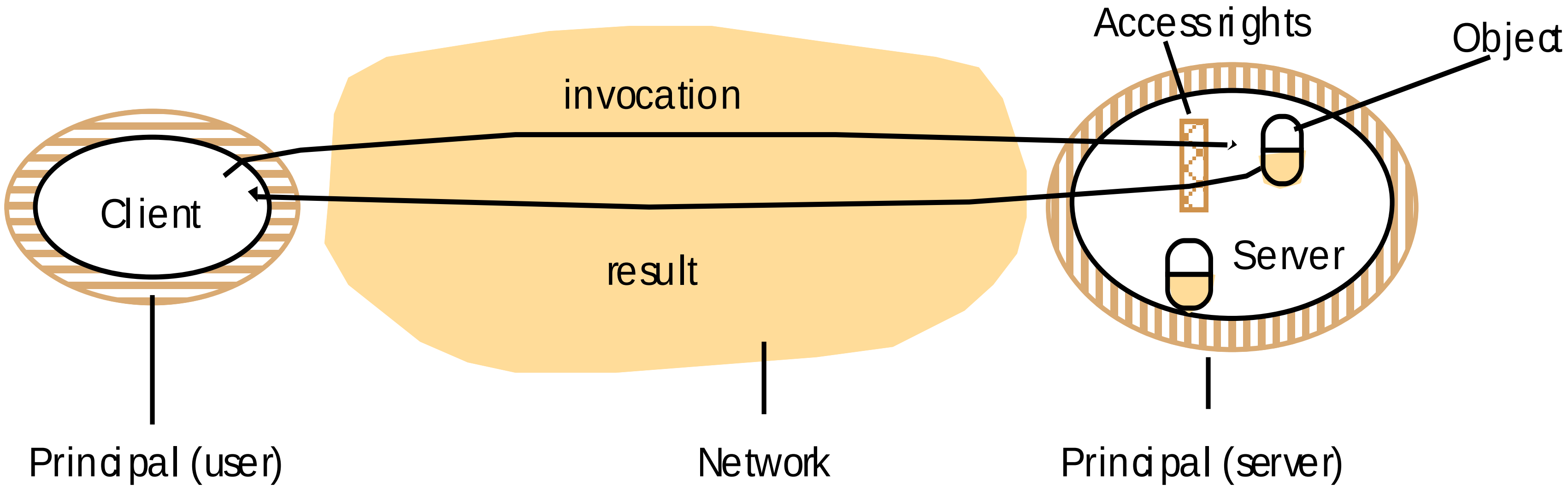
Security

- Protect objects
- Protect processes and communication
- Threats and their detection
- Countermeasures: Cryptography, secrets, authentication, certification, secure channels, security models

Protecting objects

- Access rights can be managed through *object adapters* which set policy for invocations.
- Clients and servers can check the authority of their respective opposite *principal* relative to a transmitted object.
- Linked with the idea of LWPs and threads invoked on the server.

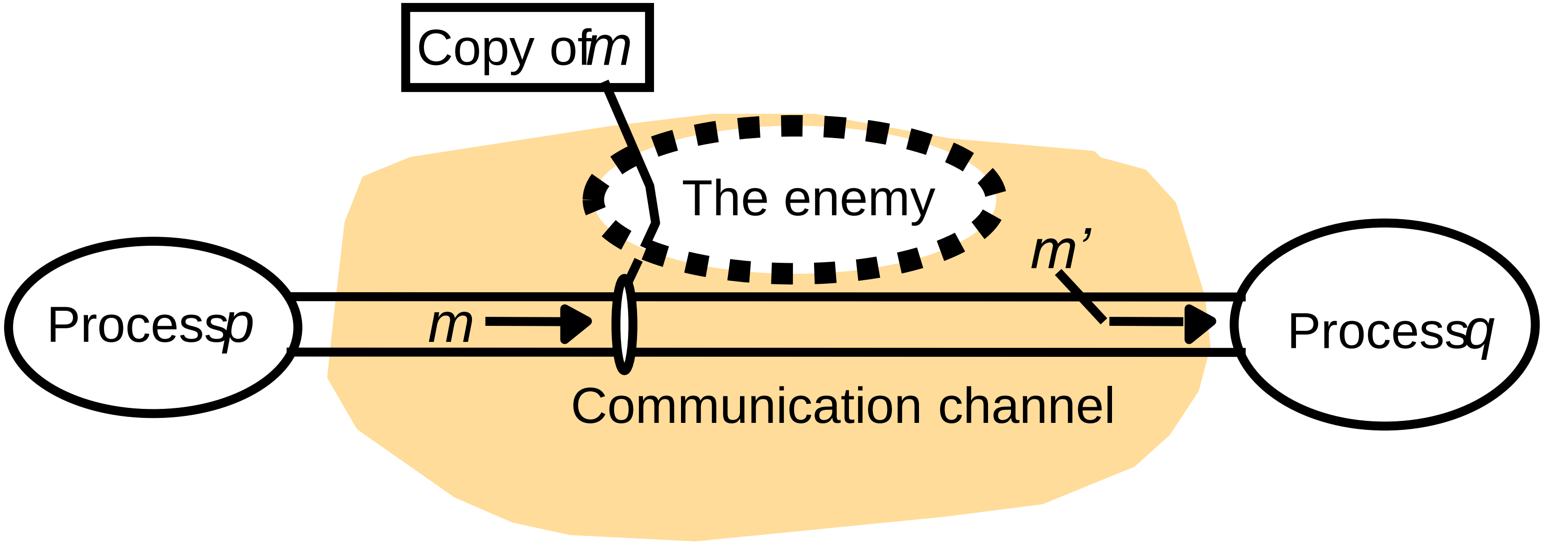
Figure 2.12
Objects and principals



Securing processes

- Services, servers, and peers expose their interfaces, some of which can be automatically detected, enabling anyone to send invocation requests
- Semantic attacks on particular servers can occur.
- Denial-of-service attacks through flooding

Figure 2.13
The enemy



Message attacks

- Confidentiality
- Integrity
- Authentication and attributability
- Non-repudiability
- Guaranteed Delivery
- Guaranteed order and non-replay

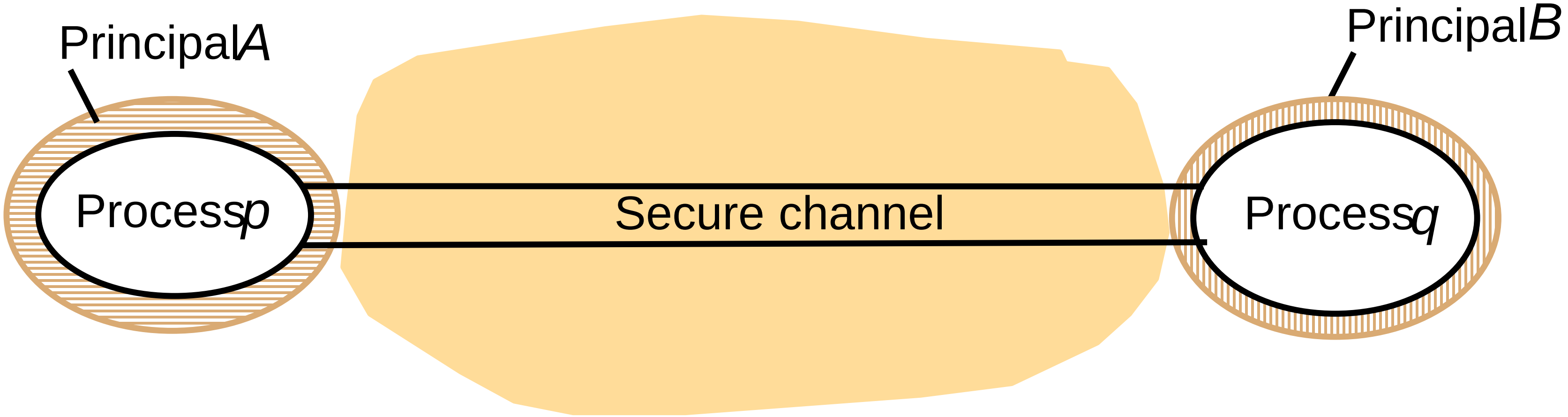
Encryption

- Public key, shared key
- Difficulties with computing power in the future – based on the factoring of large prime numbers. E.g., quantum computing
- Administration can be a nightmare
- Depends on secrets, and these can be compromised.

Authentication

- Encrypt a portion of a message and include sender, time, etc.
- Certificates

Figure 2.14
Secure channels



Secure channels

- Tunneling
- Virtual private networks – logical networks built on top of other protocols.
- TLS, SSL

Two other vulnerabilities

- Denial of service – excessive requests intended to flood server and block valid requests from gaining service
  - Russian mafia DOS attack on Island football betting.
- Mobile code, including trojan horses, spyware, etc.

Security Models

- Encryption and secure channels are only band-aids.
- Threat models and security models are needed.
- The social system of human users interacting with computer systems must also be modeled.
- User *groups* are complex but integrated with any security model.

Summary

- Most distributed systems based on some architecture: client/server, peer-to-peer, centralized thin-client processing, services
- Mobility of code is increasing
- Explosion of mobile devices
- Models of process interaction, failures, security – basic themes in distributed systems

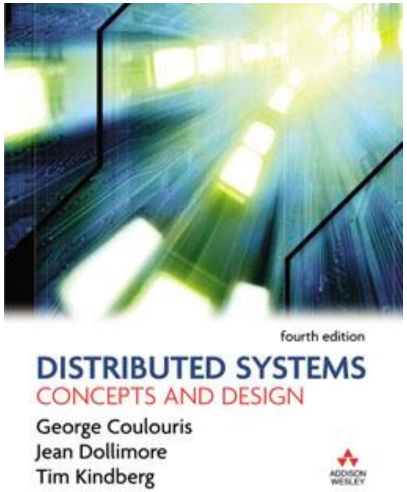
- Synchronous vs. ansychronous models – guarantees of min and max message delivery times, clock drift from external time
- Classification of failures
- Masking of failures
- Omissions
- Threats and remedies

To remember…

- Multiple Authors…architectual model…load balancing…software layers…Client/server…peer-to-peer…open systems…global clock…remote administration…OSI model…virtual machine…middleware…services…

Blank

Figure Slides for Chapter 3: Networking and Internetworking



From Coulouris, Dollimore and Kindberg
Distributed Systems:
Concepts and Design

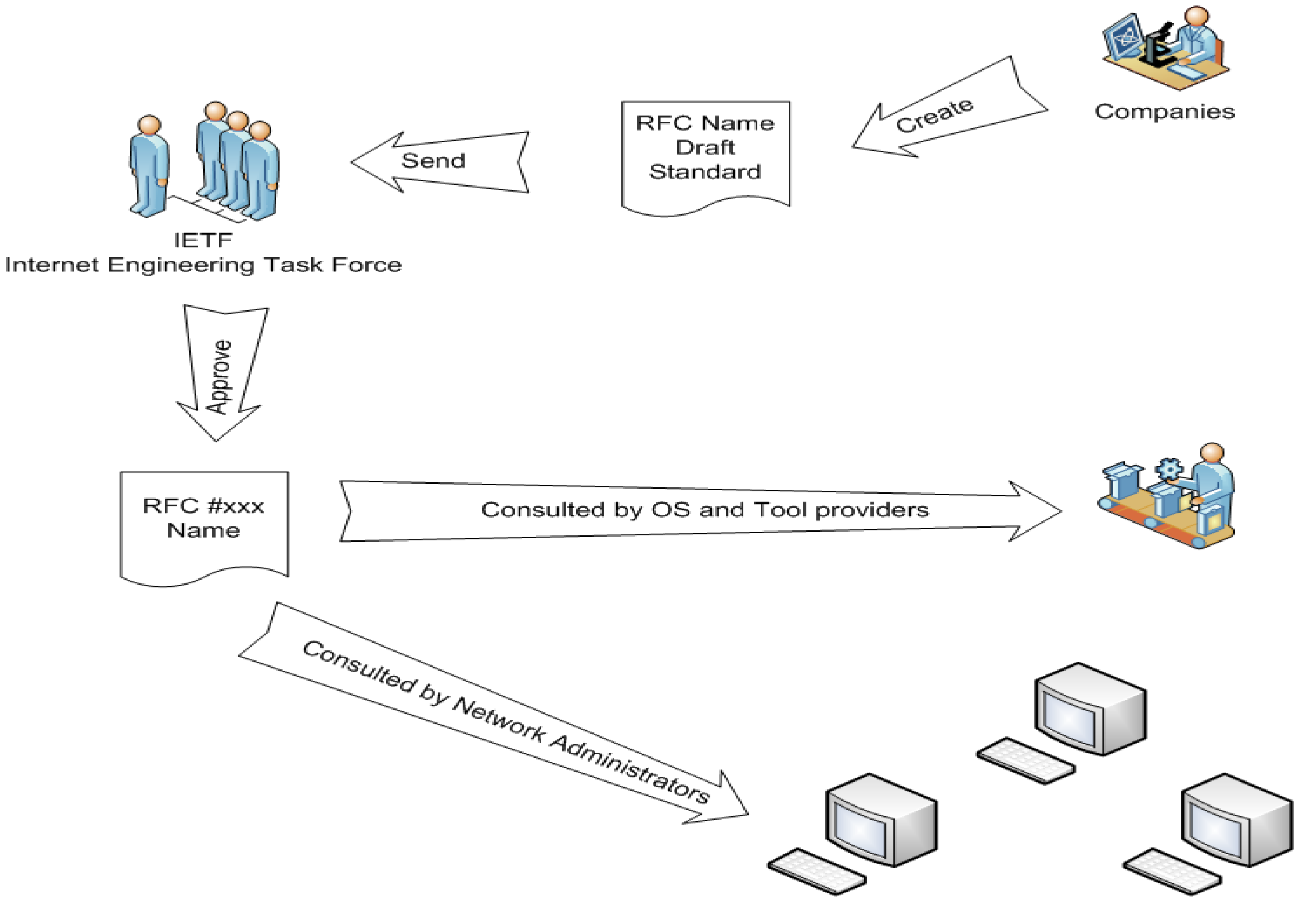
CSC435 DePaul -- Elliott

- Slides prepared by…
  - – Professor Clark Elliott
  - – DePaul University
- Some slides courtesy of Mark Goetsch
  - – DePaul University

Networking

- Networking for Distributed Systems
- Types of networks
- Network principles
- Internet Protocols
- Case studies, Ethernet, WiFi, Bluetooth

RFC Process – request for comments. You can comment too!



Companies
Create
RFC Name
Draft
Standard
Send
IETF
Internet Engineering Task Force
Approve
RFC #xxx
Name
Consulted by OS and Tool providers
Consulted by Network Administrators

Link to the RFC process and list of RFC’s http://www.rfc-editor.org/

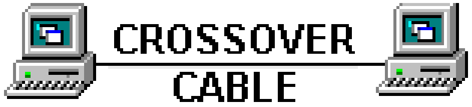
Components

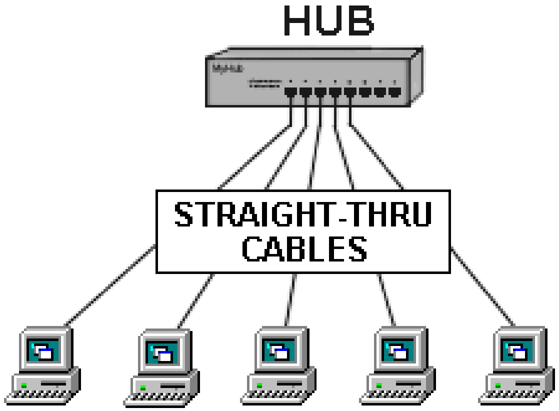
- Trasmission lines: wire, cable, fiber, satellite, wireless
- Harware: routers, switches, hubs, NICs
- Software: protocol stacks, communication handlers, OpSys drivers
- Hosts == computers connected by network
- Nodes == computer or switching device on network.

Physical Layer

Network Interface Card

Cat5e Ethernet Cable

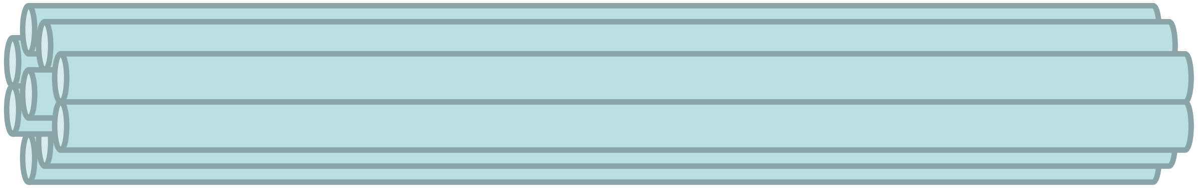




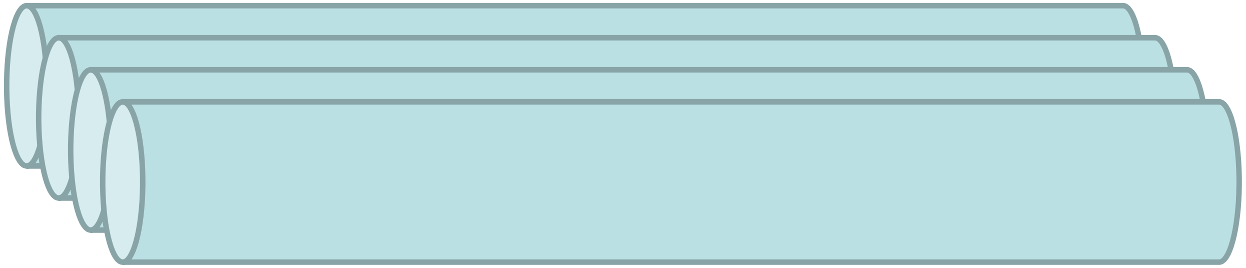
Telephone Line Size

DS-0 64 Kbps 1 Telephone Channel

DS-1/T-1 1.54 Mbps 24 Telephone Channels



DS-3/T-3 44.736 Mbps 28 T-1 Lines



Bits & Bytes

1. Transmission quantity is measured in **10^n** so that 64K is 64 x 1000 or **64,000.**
2. Computer quantity is measured in **2^n** so that 64K is 64 X 1024 or **65,536.**

This is a difference of 2.34%. Close enough for government work??

File size and storage is measured in terms of bytes but transmission is measured in terms of bits (i.e. don't' forget to **multiply/divide by 8**).

Here is a twister -> Remember the calculation for Transmission Time?

Transmission Time = RTT + (Transfer Size / Bandwidth)

We have a 100 ms RTT, a 64K sized file, and a 64K single channel.

TT = .1 s+ (64 * 1024 * 8) bps / (64 * 1000) bps
TT = **8.292** s

Binary Bytes vs. Decimal Bytes

- Network, 1K = 1,000, 1M = 1,000,000
- Computer 1K = 1024, 1M = 1,048,576
  - Except *disk drives* use powers of ten to artificially inflate the size of the drive.
  - 1 Terabyte disk drive = 931 GB
- So how long does it take to send the payload of 1 GB of data over a “1 GB” network line?
- http://www.codinghorror.com/blog/2007/09/gigabyte-decimal-vs-binary.html

It’s not rocket science – or is it?

- The power of ten / powers of two problem causes communication problems between CS people and IT people
- IT managers MUST understand this issue
- *“NASA lost a $125 million Mars orbiter because a Lockheed Martin engineering team used English units of measurement while the agency's team used the more conventional metric system for a key spacecraft operation, according to a review finding released Thursday.”* (*http://www.cnn.com/TECH/space/9909/30/mars.metric.02/*)

Calculating
RTT, Throughput, and Transfer Time

RTT is 2 X Latency and is the round trip time delay

Transfer Time = RTT + (Transfer Size/Bandwidth)

Throughput = Transfer Size/Transfer Time

Example: User wants to fetch a 1MB file across a 1 Gbps network with a 50 ms latency.

RTT = 2 X Latency = 100 ms
Transfer Time = .1 sec + (8 X 1,048,576)/1,000,000,000
= .1 sec + .008 sec
= 108 ms
Throughput = (8 X 1,048,576)/.108 sec = 77,393,816
= 77Mbps and not 1Gbps.

Design

- Development hindered by legacy systems integrated in virtually every connected computer.
- Internet based on decades old technology
- Packet switching discussed in 1962!

Performance

- Latency is delay that occurs after a send before data starts to arrive at dest.
- Can be though of as time to send an empty data packet.
- Other latency beyond that of network – other levels of protocol stack
- Data transfer rate: bits per second rate of data transfer after latency accounted for per channel
- Msg trans time = latency + length / data trns rate

- Transfer rate is constrained by physical characteristics
- Latency determined by software overhead, routing delays, load of network.
- System bandwidth may include many channels at given transfer rate
- For DSes with many small messages, latency may be main constraint

Figure 3.1
Network performance

	Example	*Range*	*Bandwidth (Mbps)*	*Latency (ms)*
Wired:				
LAN	Ethernet	1-2 kms	10-1000	1-10
WAN	IP routing	worldwide	0.010-600	100-500
MAN	ATM	250 kms	1-150	10
Internetwork	Internet	worldwide	0.5-600	100-500
Wireless:				
WPAN	Bluetooth (802.15.1)	10 - 30m	0.5-2	5-20
WLAN	WiFi (IEEE 802.11)	0.15-1.5 km	2-54	5-20
WMAN	WiMAX (802.16)	550 km	1.5-20	5-20
WWAN	GSM, 3G phone nets	worldwide	0.01-02	100-500

Times

- < 1 micro-sec – access local memory
- ½ Msec -- get reply for short message on ethernet from cache memory (~1000x!)
- 4.2 Msec – 7200 RPM hard disk latency
- 200 Msec round trip for internet request

DS issues for networking

- Growth is going to be significant
- Reliability is relatively high already
- Security – firewalls are stupid and restrictive, much need for flexible but effective security
- Increasing use of mobile devices require addresses schemes for finding them
- QoS improvements needed for Multimedia
- Groups are important - require multicast

Transmission rates

- 4 kbit/s – minimum achieved for encoding recognizable speech (using special-purpose speech codecs)
- 8 kbit/s – telephone quality
- 192 kbit/s – Nearly CD quality for a file compressed in the MP3 format
- 1.4 Mbps/s – CD audio (uncompressed, 16 bit samples × 44.1 kHz × 2 channels)
- 2.0 Mbit/s — VHS quality, (compressed)
- 8 Mbit/s — DVD quality, (compressed)
- 27 Mbit/s — HDTV quality, (compressed)
- 480 Mbit/s USB 2.0 (USB 3.0 4.8 Gbit/s proposed)
- 2400 Mbit/s SATA II drives

Types of networks

- WPANs – PDAs, cameras, phones
- LANs – support intranets. Ethernet already up to 1000 Mbps. Cheap for 100 Mbps
- WANs – large distances, routers, internet, latency. Latency is bounded by speed of light (e.g. 0.13sec) , satellite round trip.
- MAN – (metropolitan). DSL, existing wire.
- WiFi – up to 1.5 km, ~54 Mbps
- Internetworks – link all of them together

Transmission types

- Applications transmit *messages*
- Hardware often transmits *packets*
- Break message into homogenous packets
- Media *streams* require high QoS, especially audio. Delayed packets are simply dropped. IPv6 has QoS slots.
- ATM networks are faster, but expensive

Switching schemes

- Broadcast – ethernet, WiFi. No routing
- Circuit switching – throw switches in line to prepare a circuit. POTS, train tracks.
- Packet switching – processing and storage of computers allows free real-time routing
- Frame relay – inspect first few bits, then pass on frame without storing or inspection. ATM uses this. Very fast, parallel, establish virtual circuit first.

Protocols

- Set of rules and formats
- Specification of sequence of messages to exchange
- Specification of the data format

OSI Standard

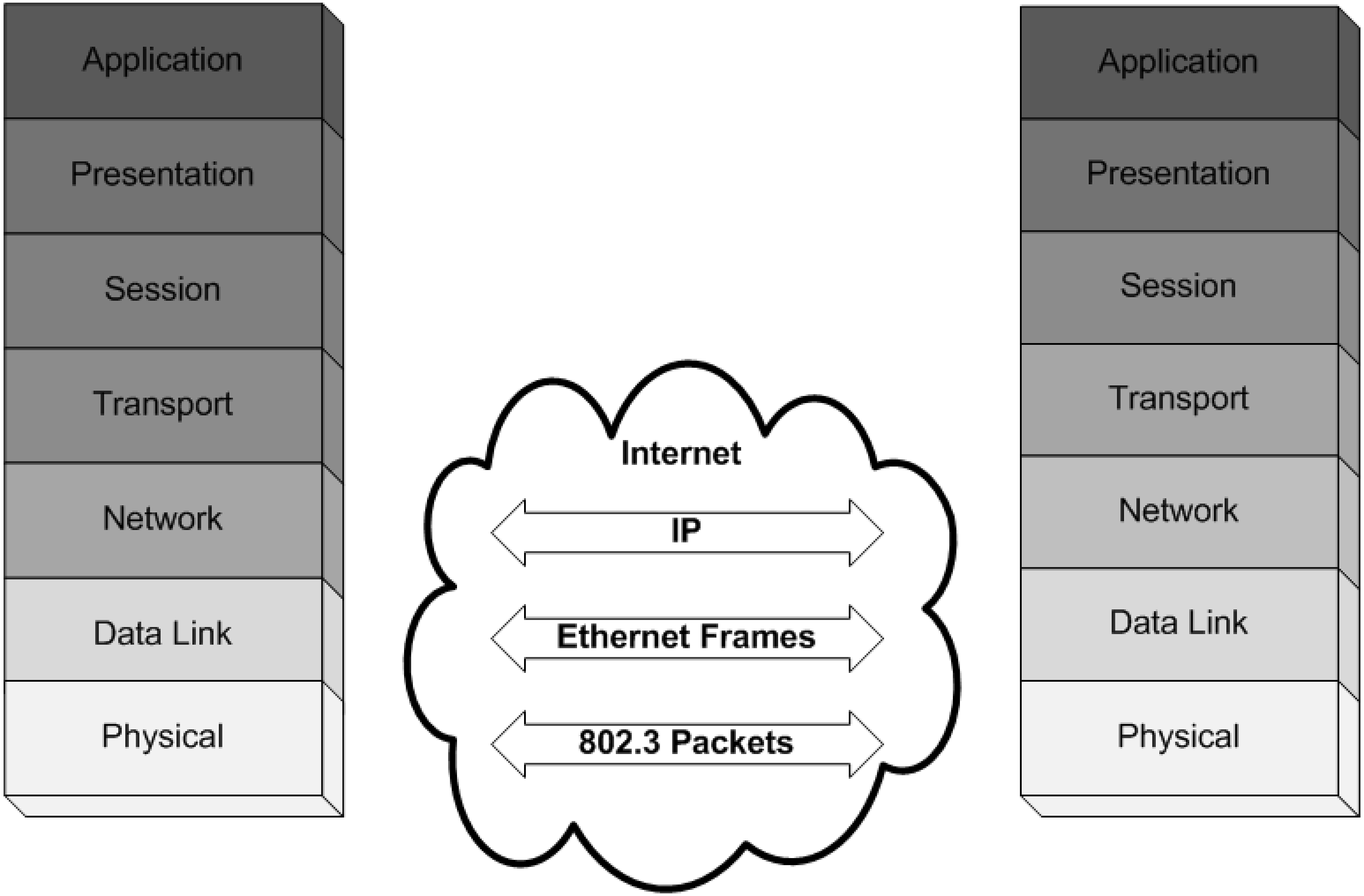
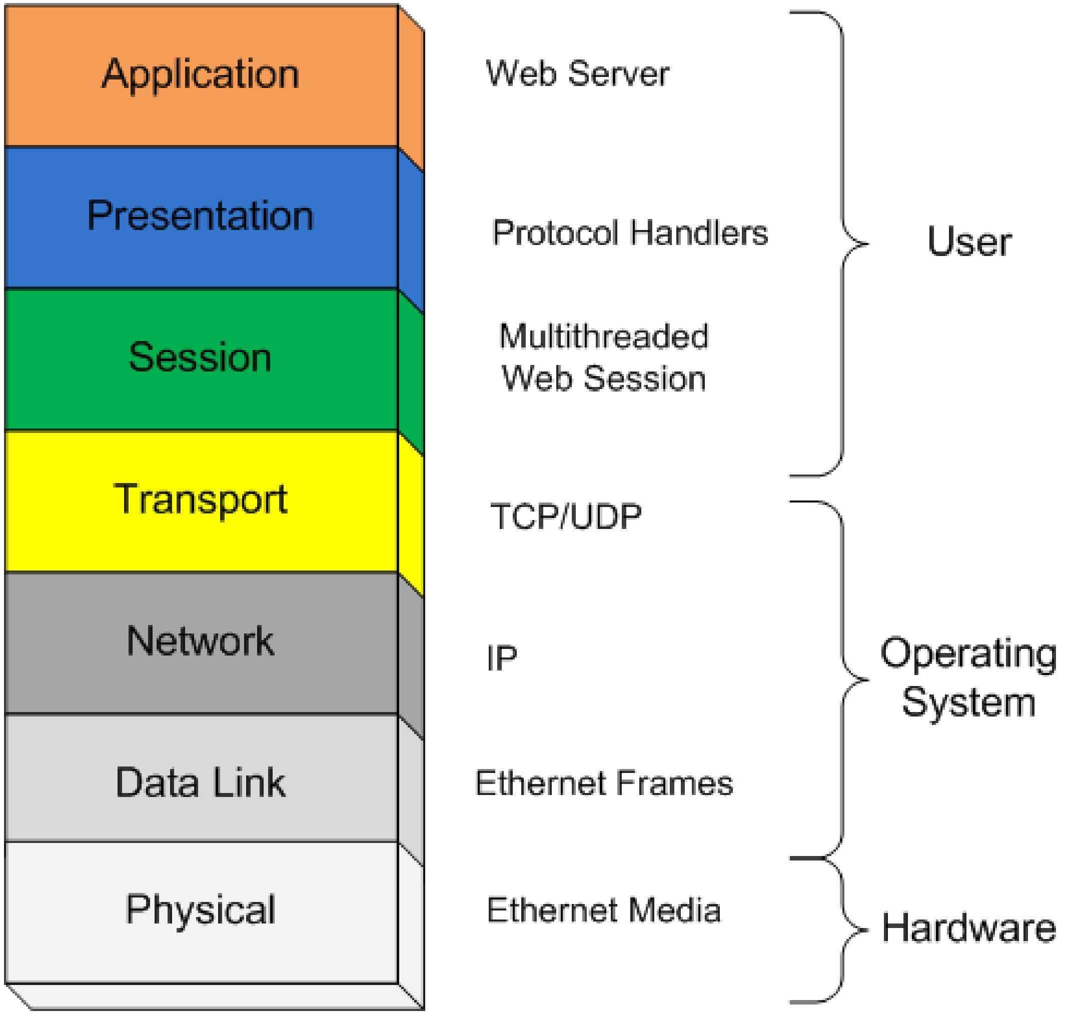


Figure 3.5
OSI protocol summary

Layer	*Description*	*Examples*
Application	Protocols that are designed to meet the communication requirements of specific applications, often defining the interface to a service.	HTTP, FTP, SMTP, CORBA IIOP
Presentation	Protocols at this level transmit data in a network representation that is independent of the representations used in individual computers, which may differ. Encryption is also performed in this layer, if required.	Secure Sockets (SSL),CORBA Data Rep.
Session	At this level reliability and adaptation are performed, such as detection of failures and automatic recovery.	
Transport	This is the lowest level at which messages (rather than packets) are handled. Messages are addressed to communication ports attached to processes, Protocols in this layer may be connection-oriented or connectionless.	TCP, UDP
Network	Transfers data packets between computers in a specific network. In a WAN or an internetwork this involves the generation of a route passing through routers. In a single LAN no routing is required.	IP, ATM virtual circuits
Data link	Responsible for transmission of packets between nodes that are directly connected by a physical link. In a WAN transmission is between pairs of routers or between routers and hosts. In a LAN it is between any pair of hosts.	Ethernet MAC, ATM cell transfer, PPP
Physical	The circuits and hardware that drive the network. It transmits sequences of binary data by analogue signalling, using amplitude or frequency modulation of electrical signals (on cable circuits), light signals (on fibre optic circuits) or other electromagnetic signals (on radio and microwave circuits).	Ethernet base- band signalling, ISDN

OSI Protocol Stack Implemented



Layers → object code

- Each layer makes library calls to the level below which get included in the object code, or as SVC to operating system.
- Fully compiled code might have all layers compiled into executable, with entry point into running OpSys procs or kernal code.
- Each layer “hides” more complex code below.

Layers of detail – one line of app code = ~1500 lines of “real” code.

- Java App: “open a socket”
- .TCP / IP: bind, connect, listen…
- ..Allocate application buffers
- …TCP: handshaking with other host, ports…
- …. Transport: break messages into units, route
- ….. Allocate OpSys buffers
- …….[etc.]
- ………translate to MAC ethernet…

Figure 3.2
Conceptual layering of protocol software

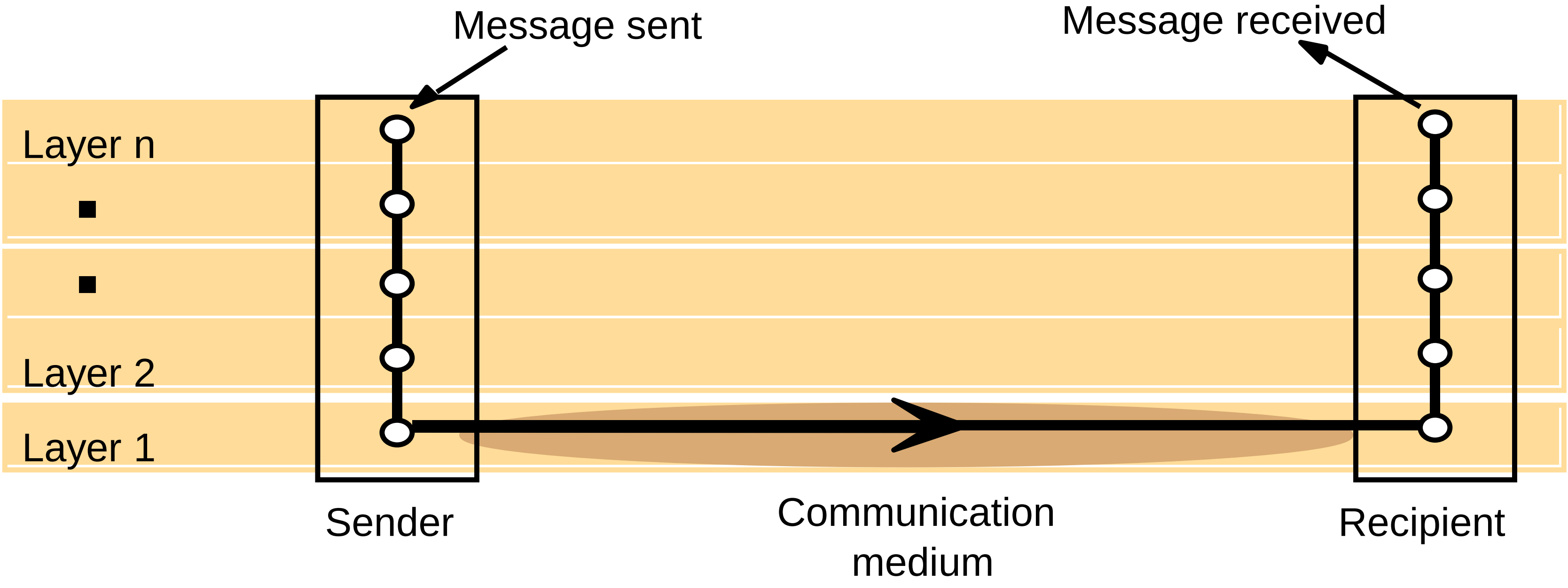


Figure 3.4
Protocol layers in the ISO Open Systems Interconnection (OSI) model

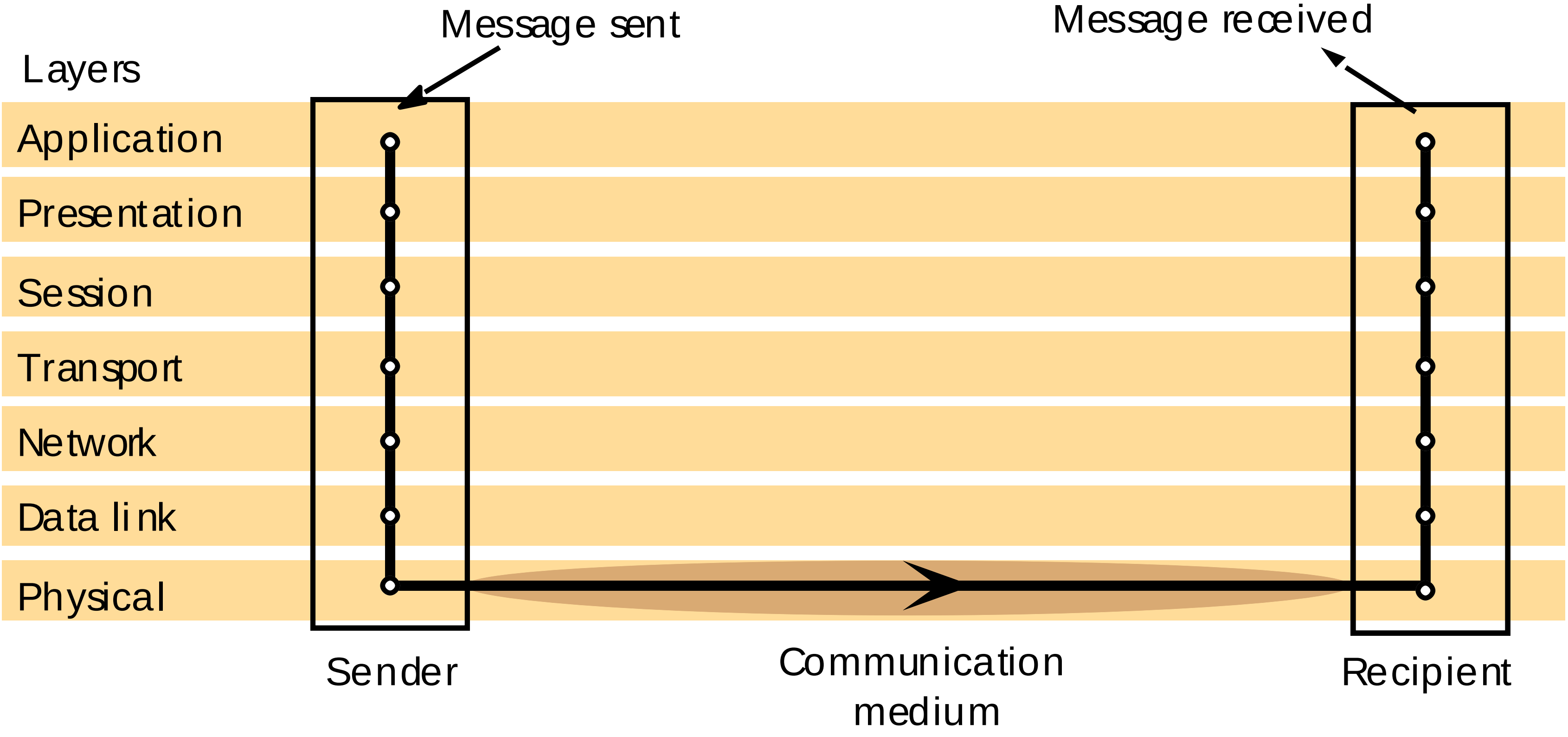


Figure 3.3
Encapsulation as it is applied in layered protocols

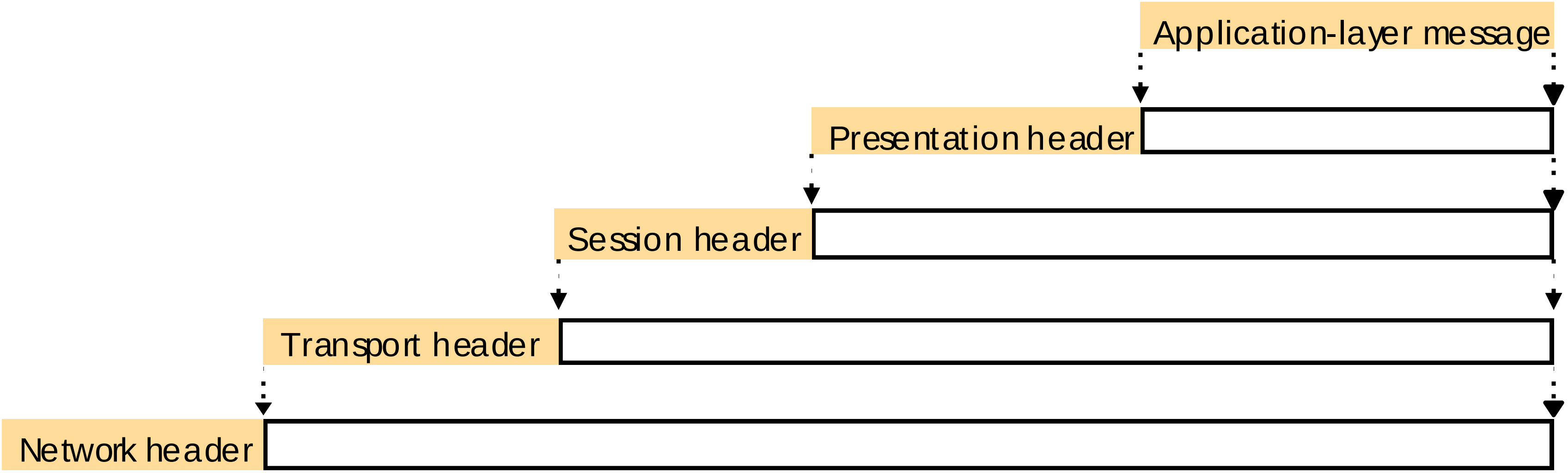


Figure 3.13
Encapsulation in a message transmitted via TCP over an Ethernet

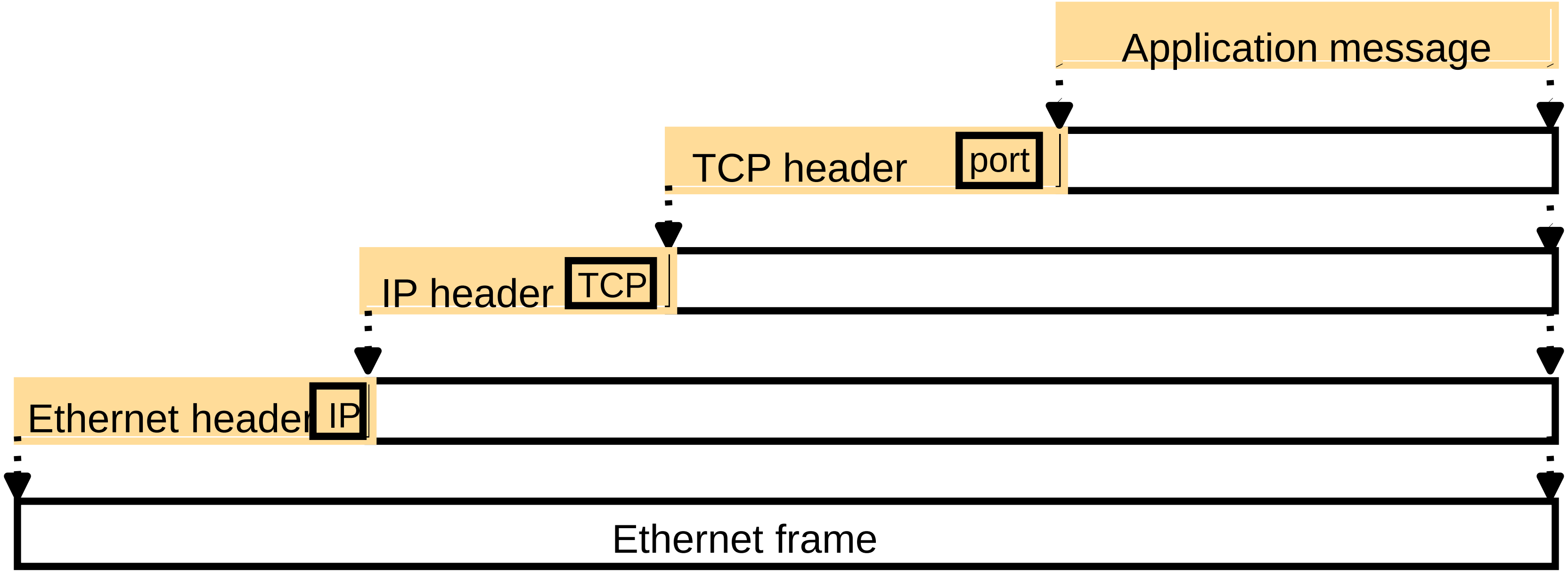
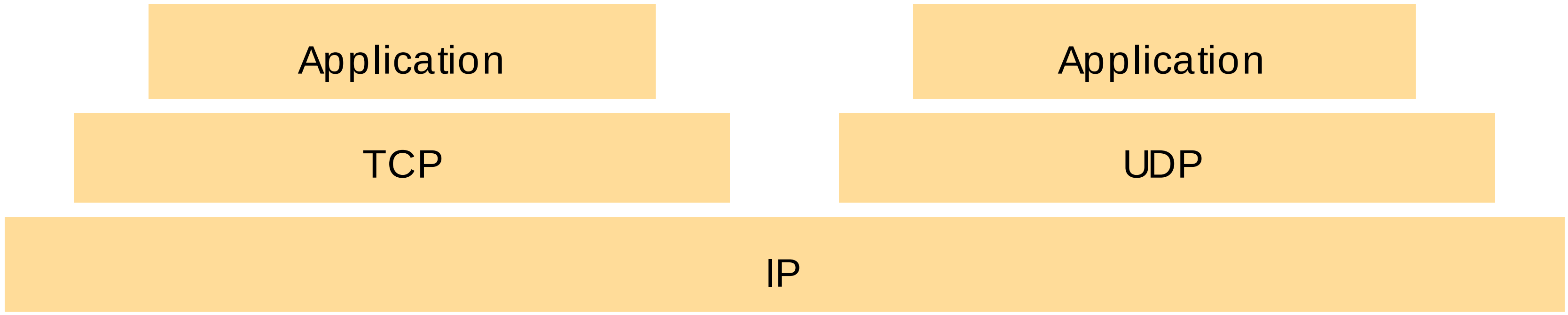
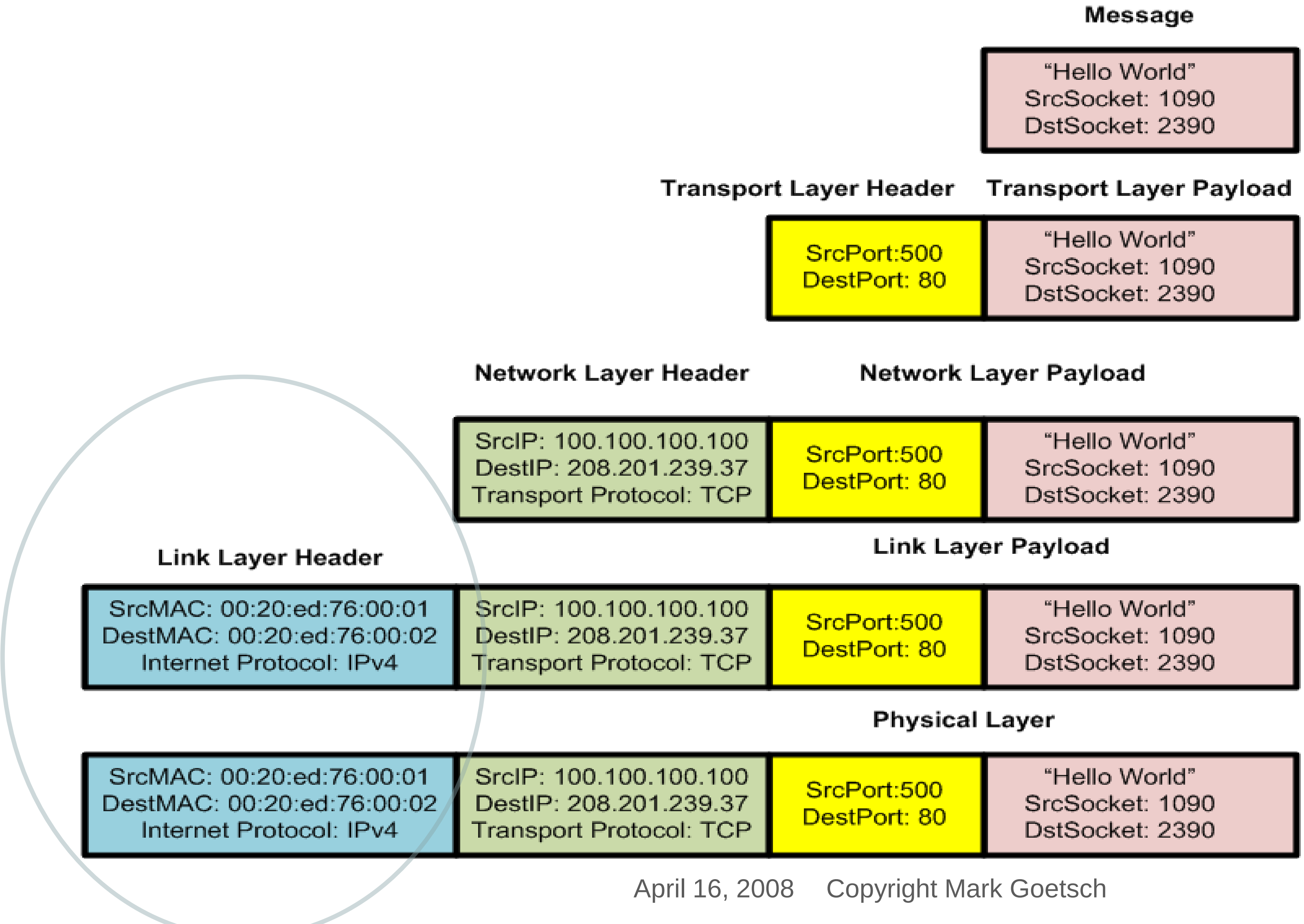


Figure 3.14
The programmer's conceptual view of a TCP/IP Internet



Physical and Link Layer



Message

"Hello World"
SrcSocket: 1090
DstSocket: 2390

Transport Layer Header

SrcPort:500
DestPort: 80

Transport Layer Payload

"Hello World"
SrcSocket: 1090
DstSocket: 2390

Network Layer Header

SrcIP: 100.100.100.100
DestIP: 208.201.239.37
Transport Protocol: TCP

Network Layer Payload

SrcPort:500
DestPort: 80

"Hello World"
SrcSocket: 1090
DstSocket: 2390

Link Layer Header

SrcMAC: 00:20:ed:76:00:01
DestMAC: 00:20:ed:76:00:02
Internet Protocol: IPv4

Link Layer Payload

SrcIP: 100.100.100.100
DestIP: 208.201.239.37
Transport Protocol: TCP

SrcPort:500
DestPort: 80

"Hello World"
SrcSocket: 1090
DstSocket: 2390

Physical Layer

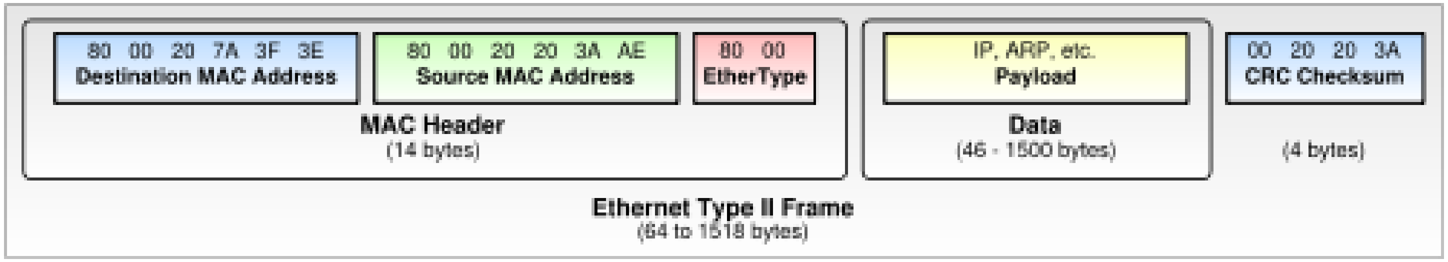
SrcMAC: 00:20:ed:76:00:01
DestMAC: 00:20:ed:76:00:02
Internet Protocol: IPv4

SrcIP: 100.100.100.100
DestIP: 208.201.239.37
Transport Protocol: TCP

SrcPort:500
DestPort: 80

"Hello World"
SrcSocket: 1090
DstSocket: 2390

Data Link Layer



MAC => [Media Access Control] Unique to each Network Interface Card [NIC]. Each frame contains which machine it came from and where it is going.

Ether Type => The Ethernet standard defines this as a sub-type but the 802.3 standard uses this as a length field [64-1,518 bytes]

Payload => In 802.3 standard bytes 43-1,497 contain data. There are three bytes allocated to LLC [where to put the frame in the buffer which is useful when dealing with multiple protocol stacks]

CRC => Validates that there was no corruption in the packet

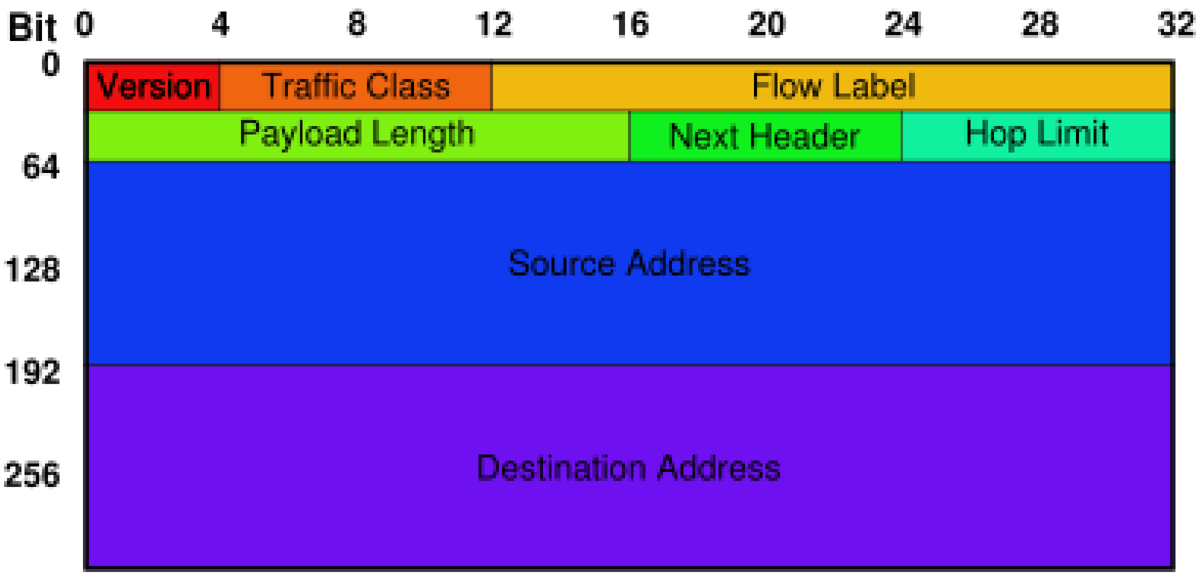
Ethernet was founded from 1973-1975 by Robert Metcalf and David Boggs at Xerox Parc.

IPv4 Packet

+	Bits 0–3	4–7	8–15	16–18	19–31
0	Version	Header length	Type of Service (now DiffServ and ECN)	Total Length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header Checksum	
96	Source Address				
128	Destination Address				
160	Options				
160 or 192+	Data				

Notice the fragment offset. The size of an IP packet can be 65,535 bytes. The maximum size [MTU-Maximum Transmission Unit] for an Ethernet packet is 1,500 bytes. To accommodate this IPv4 can fragment its packet into multiple packets that are sequenced using the fragment offset field. More on this in a couple of slides…

IPv6 Packet



Notice the Traffic Class and Flow Labels that we discuss in relation to QoS. Another thing to notice is the missing fragmentation field. IPv6 uses Path-MTU [PMTU] to query the path for the MTU of every router along the path.

http://www.geocities.com/siliconvalley/vista/8672/network/ipfrag.html for more on fragmentation

IP Addressing

- IP number – identifies which subnet the host (computer) belongs to, used to route IP packets through network.
- Port numbers are assigned, through OpSys utility calls, to applications that read and write to those ports.
- 1-1023 reserved for OpSys utilities: e.g. ftp, http, mail
- 1024 -49151 registered with *Internet Assigned Numbers Authority* (IANA) that has service descriptions
- 49,152 – 65,535 for private use
- In practice 1024 – 65535 are often used for programming.
- Quadrad of send addr / send port / receive addr / receive port is used to identify a conversation and is included in each TCP/IP packet for datagram delivery.

IANA Port Assignments

Port #	Service	Protocol
21	FTP	TCP/UDP/SCTP
23	Telnet	TCP/UDP
25	SMTP	TCP/UDP
80	HTTP	TCP/UDP/SCTP
88	Kerberos	TCP/UDP
118	SQL Server	TCP/UDP
443	HTTPS	TCP/UDP/SCTP

Updated this month http://www.iana.org/assignments/port-numbers

IP Address

What is an IP Address?

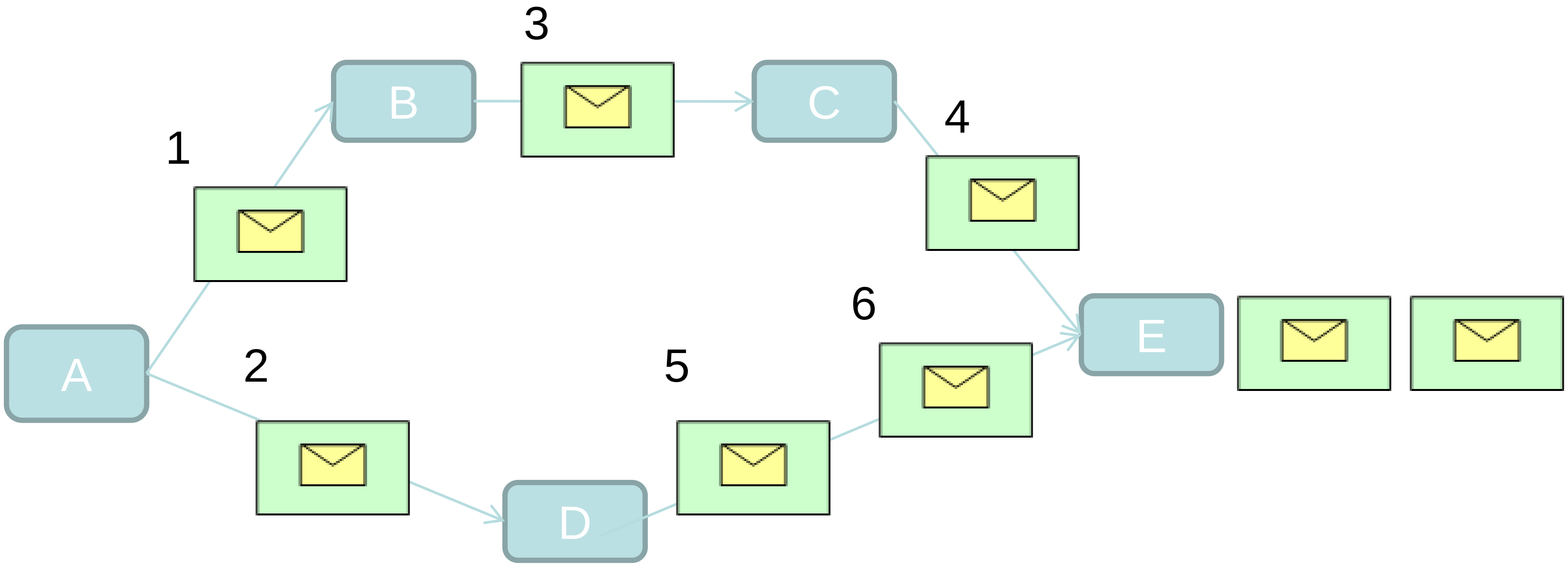
1. An IP Address is a set of 4 numbers that represent the address of any host on in your intranet or Internet.
2. IP Addresses are either static of set dynamically by DHCP every time you log in to a network.
3. Each IP address has two parts: A network ID (prefix) and a Host ID.

Host	IP Address
Yahoo	66.94.234.13 and 216.109.112.135
Microsoft	207.46.232.182 and 207.46.197.32
Whitehouse.gov	63.161.169.137

Routing

- Packets transmitted between local networks (e.g., ethernet) by *hopping* from one router to another.
- Routers keep local tables for various destinations.
- In Virtual Circuits (ATM, X.25) a fixed path is first established, and all packets follow this path. For packet-switching (IP) packets are forwarded dynamically to a router that is closer to the destination.

Packet Switched Network



1. Messages are broken up into packets.
2. Each packet is sent down and individual path [set of nodes]
3. Messages are reassembled on the other side in the same order

Store and forward

- Like the post office. Read the packet into a buffer. Read the header. Select the a router closer to the destination. Forward the packet there on a network line.
- Latency is a problem, and can be thought of as the time it takes to send a packet with no data in it from source to dest.
- Switches read the header “on the fly” and do not store the packet.

Routing algorithm

- Determine where the packet must go next – has to be simple and fast for packet-switching since each tick is charged against latency
- Communicate with neighbor routers about traffic congestion, failures, etc. to know where to forward packets. Slower and more intelligent than the above.

Distance Vector algorithm

- Table of neighbors at each router
- “Next hop” link for all destinations
- Cost of that route by number of hops
- Part one – simply look up next link by destination and send the packet there

- Part two – using RIP (router information protocol) exchange tables with neighbors ~every 30 seconds.
- If destination is in neighbor’s table, and less than existing destination + 1, then the neighbor should be the next link for the destination.
- Update costs relevant to the neighbor.
- If a faulty link is detected, set cost to infinity and send to all neighbors.
- Bandwidths can be factored into costs.

Figure 3.7
Routing in a wide area network

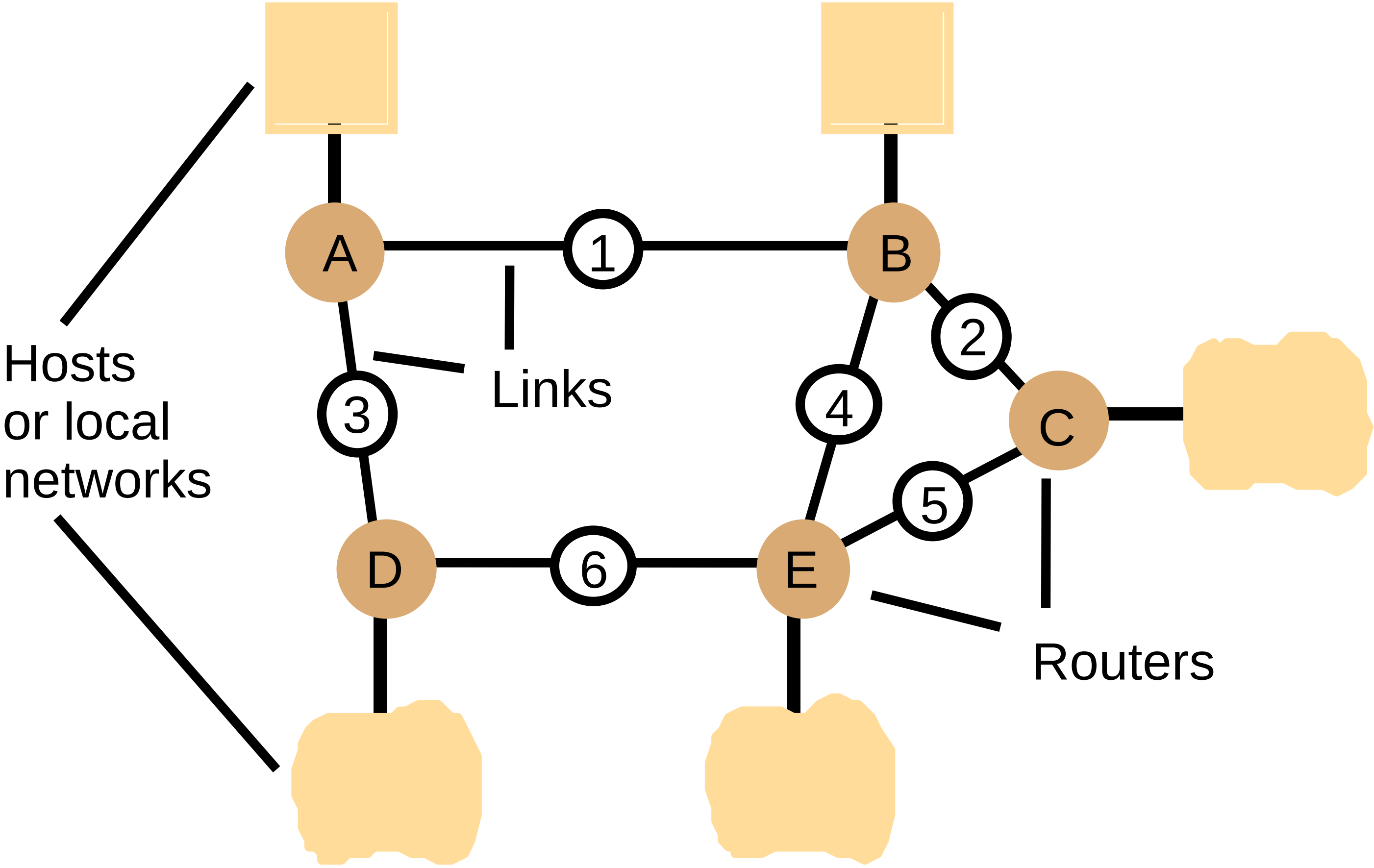


Figure 3.8
Routing tables for the network in Figure 3.7

Routings from A

To	*Link*	*Cost*
A	local	0
B	1	1
C	1	2
D	3	1
E	1	2

Routings from B

To	*Link*	*Cost*
A	1	1
B	local	0
C	2	1
D	1	2
E	4	1

Routings from C

To	*Link*	*Cost*
A	2	2
B	2	1
C	local	0
D	5	2
E	5	1

Routings from D

To	*Link*	*Cost*
A	3	1
B	3	2
C	6	2
D	local	0
E	6	1

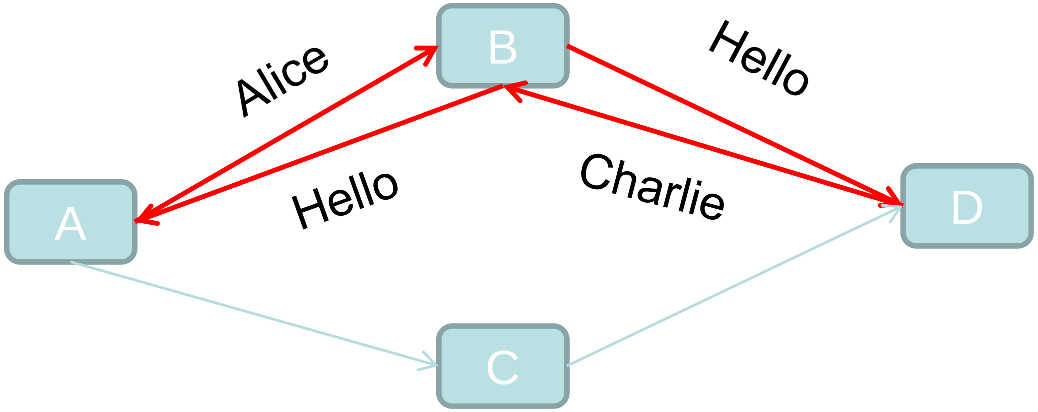
Routings from E

To	*Link*	*Cost*
A	4	2
B	4	1
C	5	1
D	6	1
E	local	0

Congestion

- Rule of thumb: when over 80% use, throughput drops because of packet loss.
- When buffers fill at a network node, then packets are dropped. These must be detected at higher levels, and then resent. Each message dropped uses > 1 unit of network resources, further leading to increased thrashing.
- *Congestion control* gracefully delays packets along the way.

Circuit-Switched Network



1. Circuits are checked to see that they are available
2. Each circuit is switched to create a continuous connection from A to D
3. The connection is maintained till the conversation ends

This was the traditional phone PBX connections [PSTN] that automated the switchboard that we see in movies. VC or virtual circuits were used by frame relay technology to use this with digital and IP.

Subnet masks

- Subnet Mask
- IP Address
- Subnet Address

- 255.255.240.000   11111111.11111111.11110000.00000000
- 150.215.017.009   10010110.11010111.00010001.00001001
- 150.215.016.000   10010110.11010111.00010000.00000000

- So, 150.215.016.000 is the subnet
- From http://www.webopedia.com/TERM/S/subnet_mask.htm
- Borrows from host portion of class B address, to create more local sub networks.

Utilities

Utility	Description
Arp	The MAC addresses that your host knows.
NetStat	The status of your network interfaces and bindings.
Ping	Whether another host is active or not.
TraceRoute	The route to another host.
IPConfig	IP address of host.
NSLookup	Returns the Host name of an address.

Figure 3.6
Internetwork layers

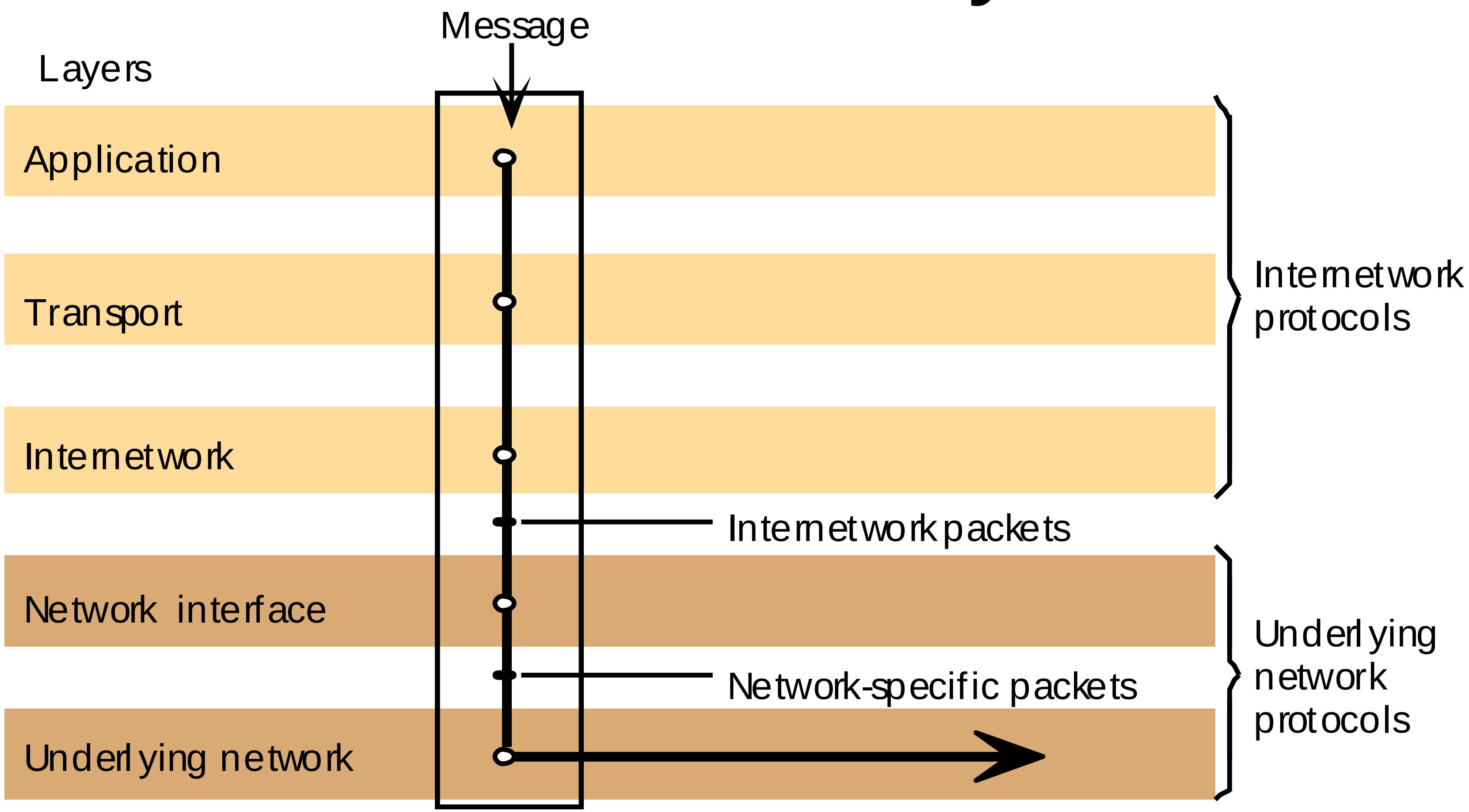
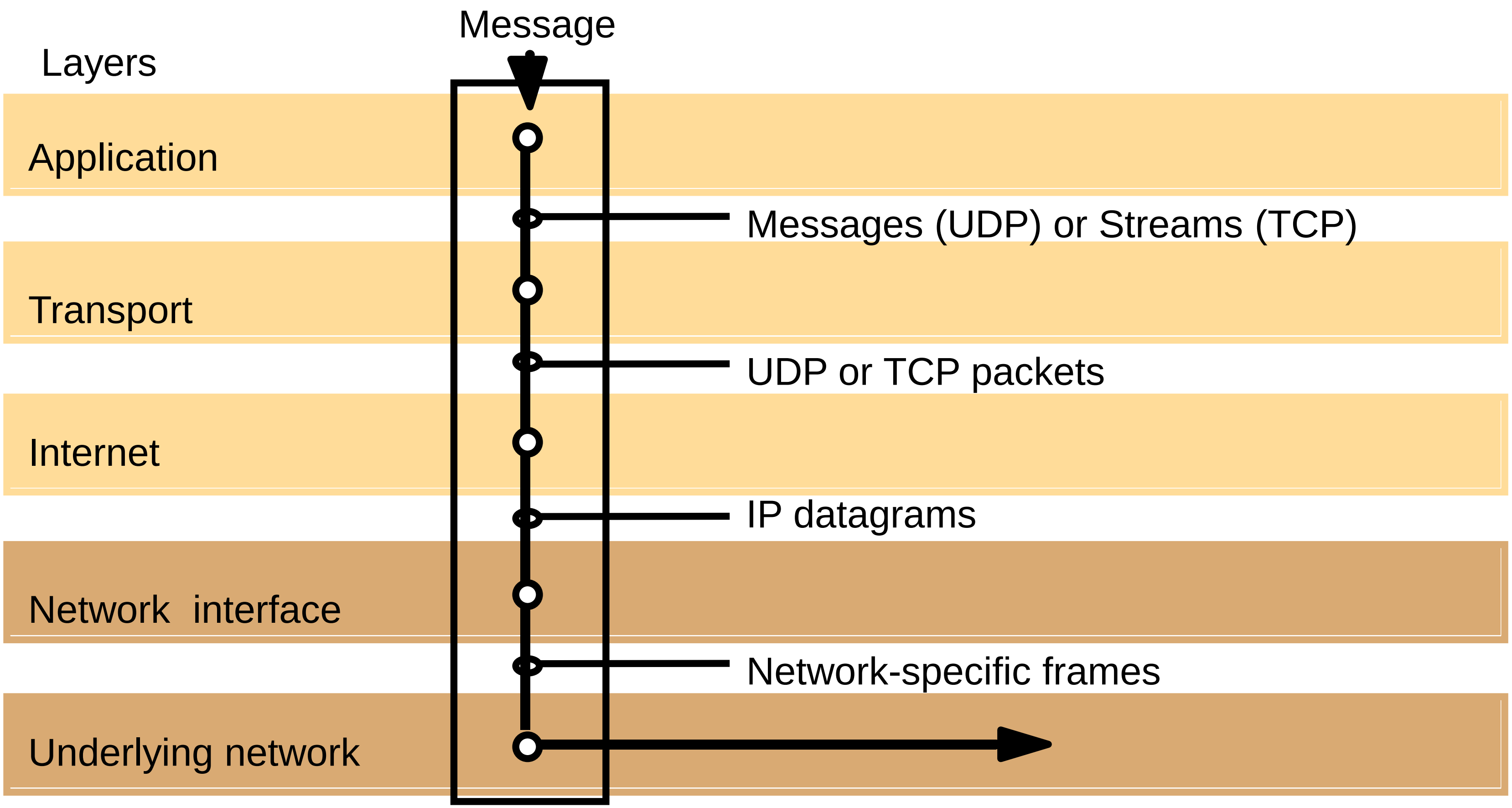


Figure 3.12
TCP/IP layers



Message

Layers

Application

Messages (UDP) or Streams (TCP)

Transport

UDP or TCP packets

Internet

IP datagrams

Network interface

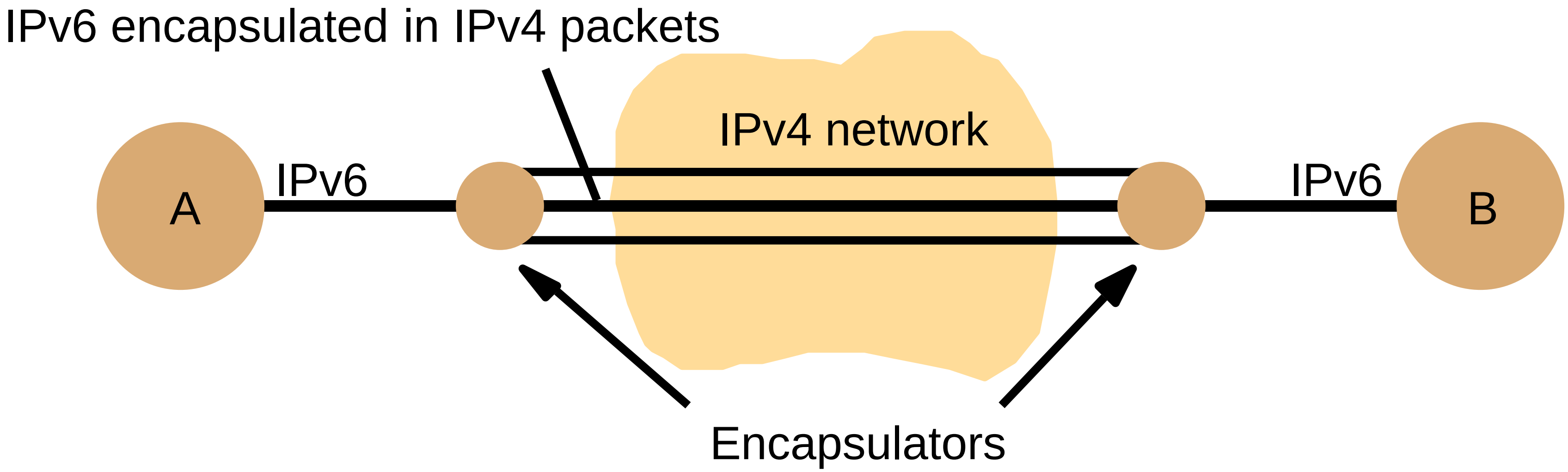
Network-specific frames

Underlying network

Migration to IPv6

- Address space has run out under IPv4
- A new standard IPv6 allows for many more addresses, and also includes improvements for security, multimedia QOS, and routing efficiency.
- Replaces IPv4 and is not compatible.
- Tunneling allows for graceful adoption.

Figure 3.11
Tunnelling for IPv6 migration



Conversion IP / local network address

- Different local networks have different addressing schemes.
- Ethernet is common, and has 48-bit addresses – ARP (address resolution protocol) is used to translate to/from IP.
- Sender: broadcast IP address query.
- Receiver: send back MAC address of NIC.
- Sender: place pair in table, then send packet using MAC address.
- Once all pairs are discovered, no query
- Beyond local net, get MAC of gateway & send

Snooping / spoofing

- Set NIC to promiscuous mode, and listen to everyone’s conversation on ethernet
- Replace your IP address/port in the “from” part of the packet with someone else’s address.
- Note: can include return address in encrypted data for authentication – do not HAVE to use IP header for return address.

IP Routing

- Routing protocols and algorithms have been studied at great length.
- Issues:
  - Not possible for every node to have complete and current knowledge
  - Speed of routing decision is critical
  - Internet is dynamic – nodes come and go, failures happen, bandwidths change, usage is dynamic, usage *type* drifts (media QoS)

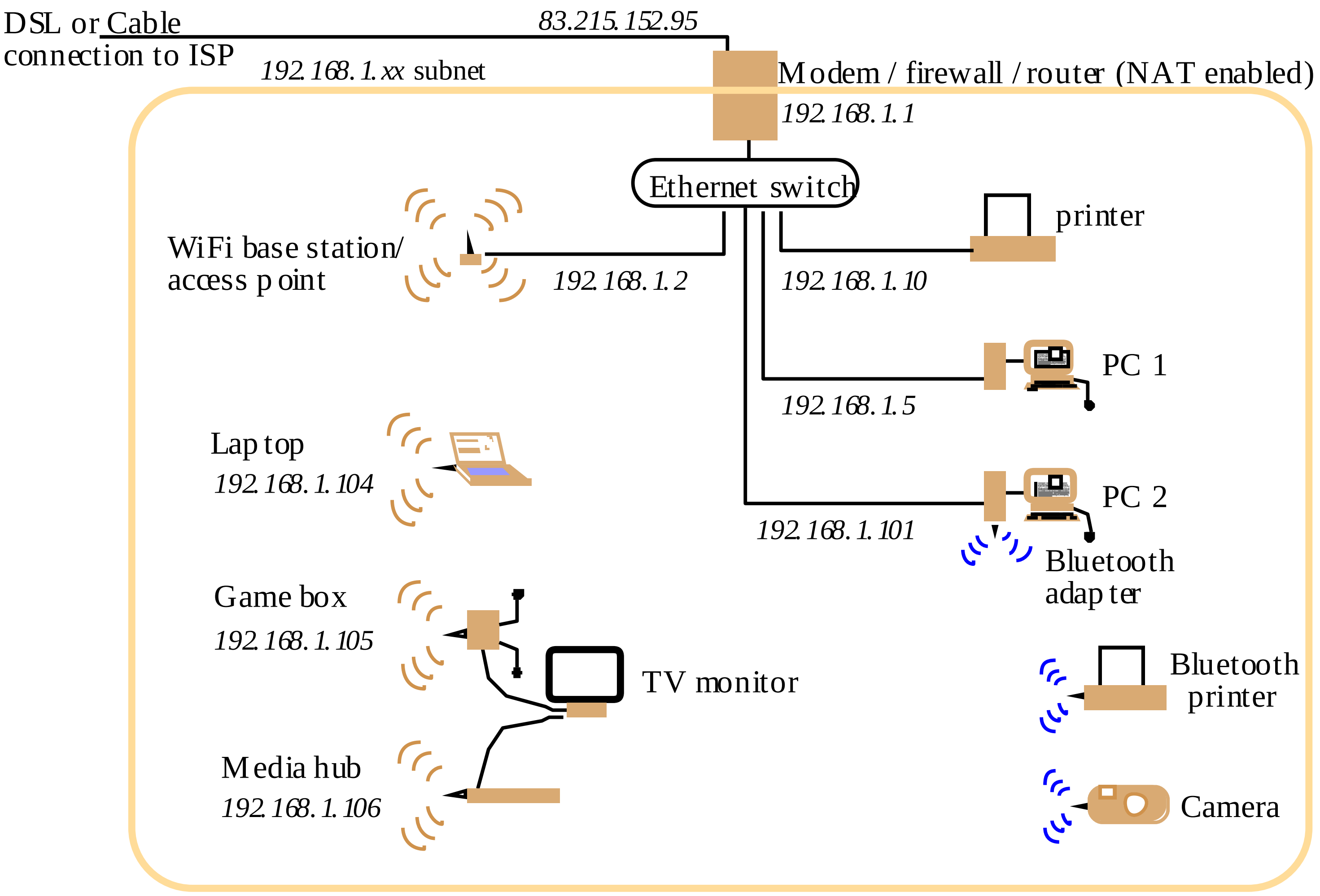
Fixes

- Get table updates at random times
- Default destination allows for incomplete tables – trade routing efficiency for table size.
- IPv6 and retro to IPv4 (after 1993), have geographical and logical meaning to IP adresses.

CIDR

- Classless Interdomain Routing
- Borrows some IP address from class C which did not need them, and donates to class B, which did. Uses masking to for dynamic length of domain / host.
- Does not otherwise concern us, and will be obviated by IPv6

Figure 3.18
A typical NAT-based home network



Network Address Translation

- Used in millions (?) of homes / offices
- 192.168.x.x “home” domain, fake IP address primarily for internal IP hosts that operate as clients to internet.

NAT…

- Local host (lhost) sends packet to internet
- Router saves lhost IP/port in table
- Replaces with router’s IP address and a virtual port that indexes entry in the table.
- Send to dest.
- If reply to IP of router, use port for table lookup, replace address with internal IP and port of lhost. Send to ethernet.
- Keep entries for a while, then discard

Servers on NAT

- With many ISPs the router IP changes, so have to have a forwarding site to translate domain name, and wait for propagation.
- External clients use IP of router (or domain name pointing to it), and a specific port.
- Router forwards all incoming traffic for that port to lhost at some manually assigned port.
- Lhost servers cannot use DHCP

IPv6

- 128 bit addressing, or 1000 IP addresses per square meter of earth’s surface
- Geographic and organizational semantics in addresses
- Flow labels for QoS improvements
- Security headers – but note that destinations are not encrypted (?), authentication for RIP
- Slow migration, but 1 billion Devices in China and India by 2014, plus mobile IP needs.

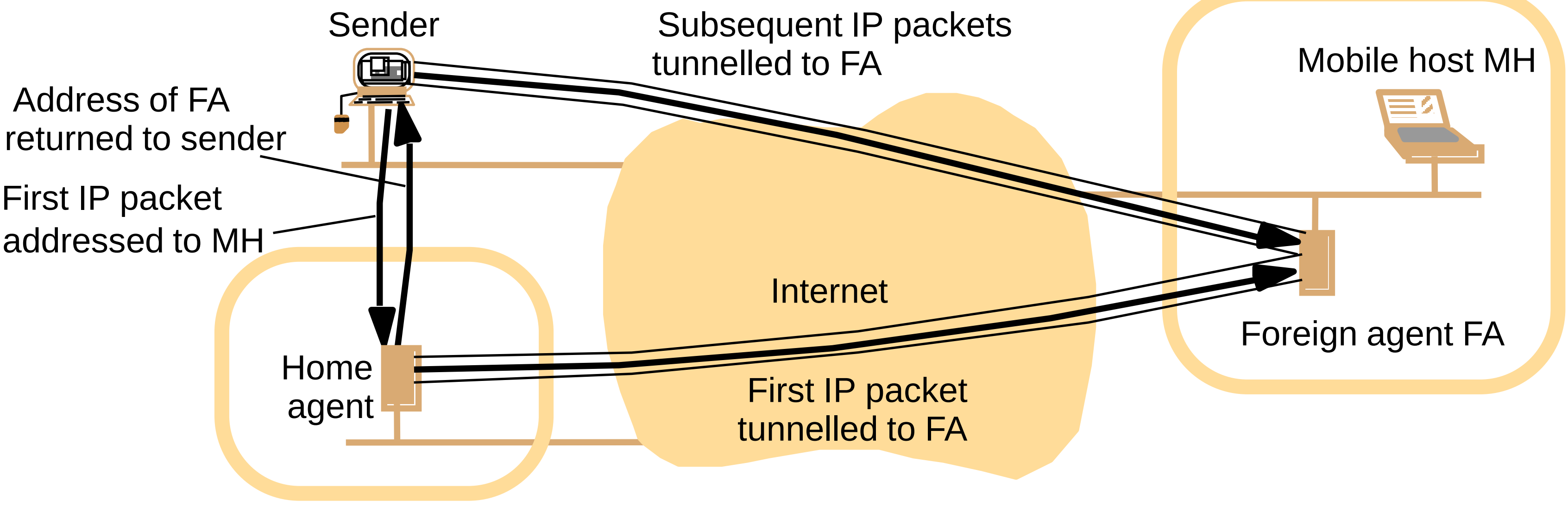
Your laptop

- When you take your laptop to a new location it discovers the local subnet and gets a local IP address from DHCP.
- Your laptop can now act as a client.
- Legacy service conversations with servers on your laptop can no longer find you.
- Address forwarding software can work around this but either low efficiency (every packet is forwarded) or both sides have to know to dynamically update the service address after initial handshaking.

Home agents and Foreign Agents

- Run at home host, and foreign host
- HA must have up-to-date knowledge of the IP location of mobile host (mhost).
- HA tells local routers to get rid of cached records of mhost.
- HA uses ARP (Address Request Protocol) to accept packets as a proxy for mhost.
- Mhost works with FA to inform HA of where it is.
- HA forwards packets wrapped in Mobile-IP packets.
- FA passes to Mhost.
- If sender is Mobile-IP enabled, can now communicate directly, otherwise repeat.
- Servers can now be mobile.

Figure 3.20
The MobileIP routing mechanism



Not best solution

- Requires HA.
- Setup time
- Requires clients to be Mobile-IP or have to route ALL packets twice.
- Mobile phones are first-class citizens as they move from cell to cell.

TCP & UDP in IPv6

- TCP and UDP exploit most of what IPv4 has to offer.
- IPv6 offers much more flexibility so we may finally see new protocols replacing TCP and UDP.

TCP & UDP

- Transport protocols, process-to-process communication msgs, uses *ports* for this.
- Processes acquire ports from the operating system through system calls.

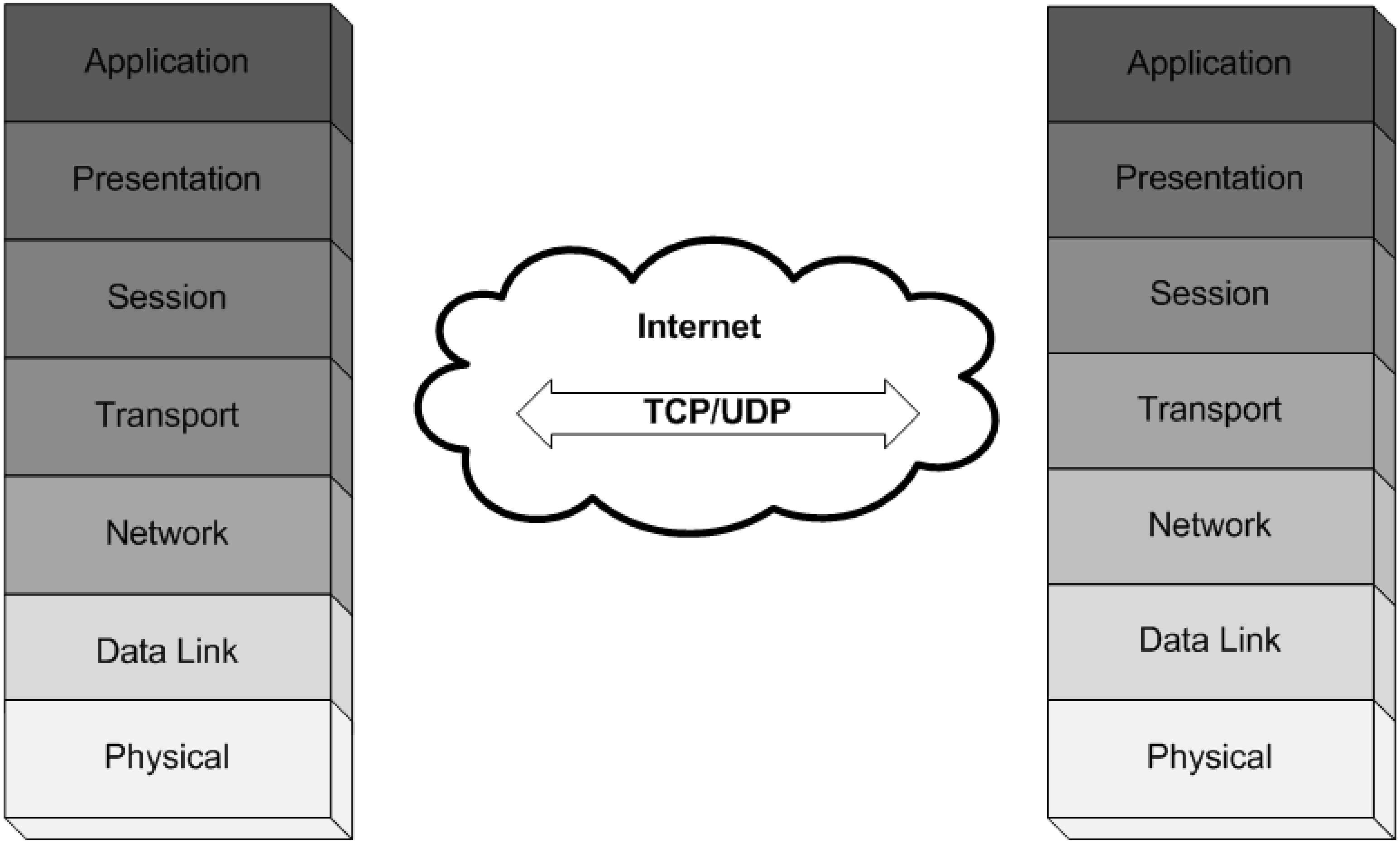
Universal Datagram Protocol - UDP

- “IP up one level”
- Source and dest ports for addressing
- Length. Checksum optionally used by receiving host.
- Cheap, fast.
- No guarantees of delivery, or order of arrival.

Trasmission Control Protocol - TCP

- Reliable transmission of streams
- Connection-oriented.
- Guaranteed delivery in guaranteed order
- IP does not know about it. End-to-end.

Transport Layer – TCP/UDP



TCP features

- Guaranteed sequence – requires “infinite” buffer – wait for first packet to arrive
- Flow control – windows, maximum delay before send, configurable, etc. so that receiver or network is not overwhelmed, and user does not have to wait too long.
- Auto retransmission of packets as needed
- Buffering as needed for flow control and ordering.

TCP Protocol

Connection-Oriented
Data packets are delivered in the correct order.

Reliable
Every piece of data will arrive.

Duplex

You can send information both ways at the same time, through the same connection.

Byte-Oriented
The data is presented as a stream of data rather than a set of packets.

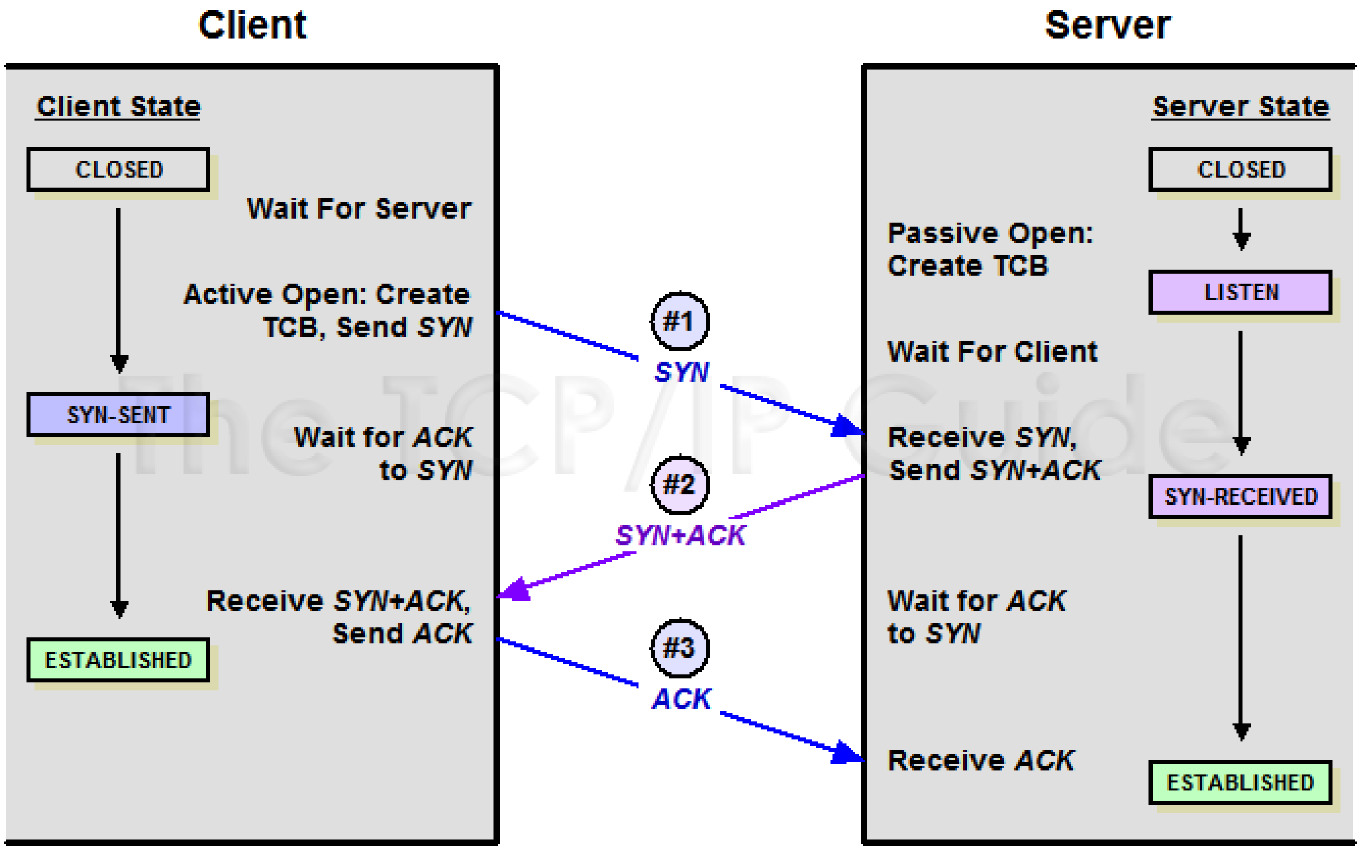
How does this happen?

The Inner Workings of TCP

1. Each TCP connection provides a buffer that will hold the data until it has been verified that it has arrived.
2. The buffer also is used to order the packets in the correct sequence.
3. Every packet that is sent requires an “Ack” from the receiver so that it knows that it has arrived.
4. A CRC field is added and checked by the receiver to see that the data was not corrupted.
5. Each connection is a 4-tuple consisting of the port numbers for both the client and server, and the IP addresses as well.

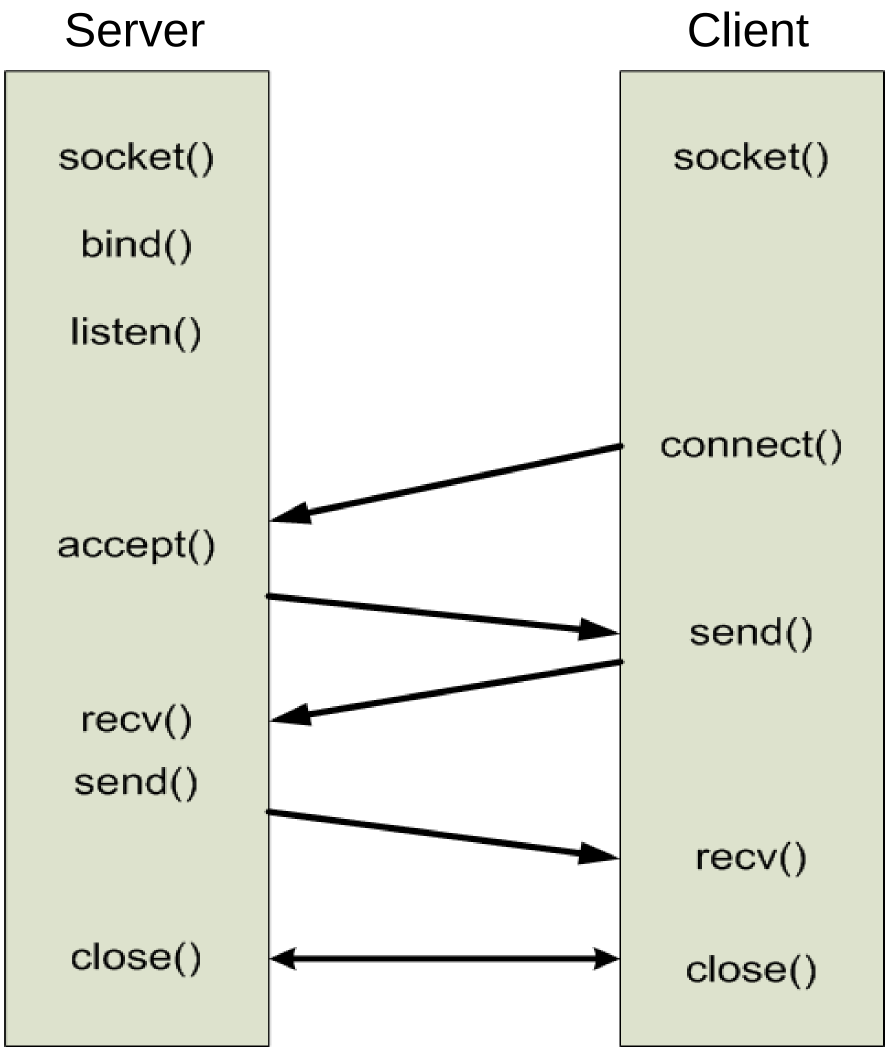
Which has quite a bit of overhead!

3 – Way Handshake

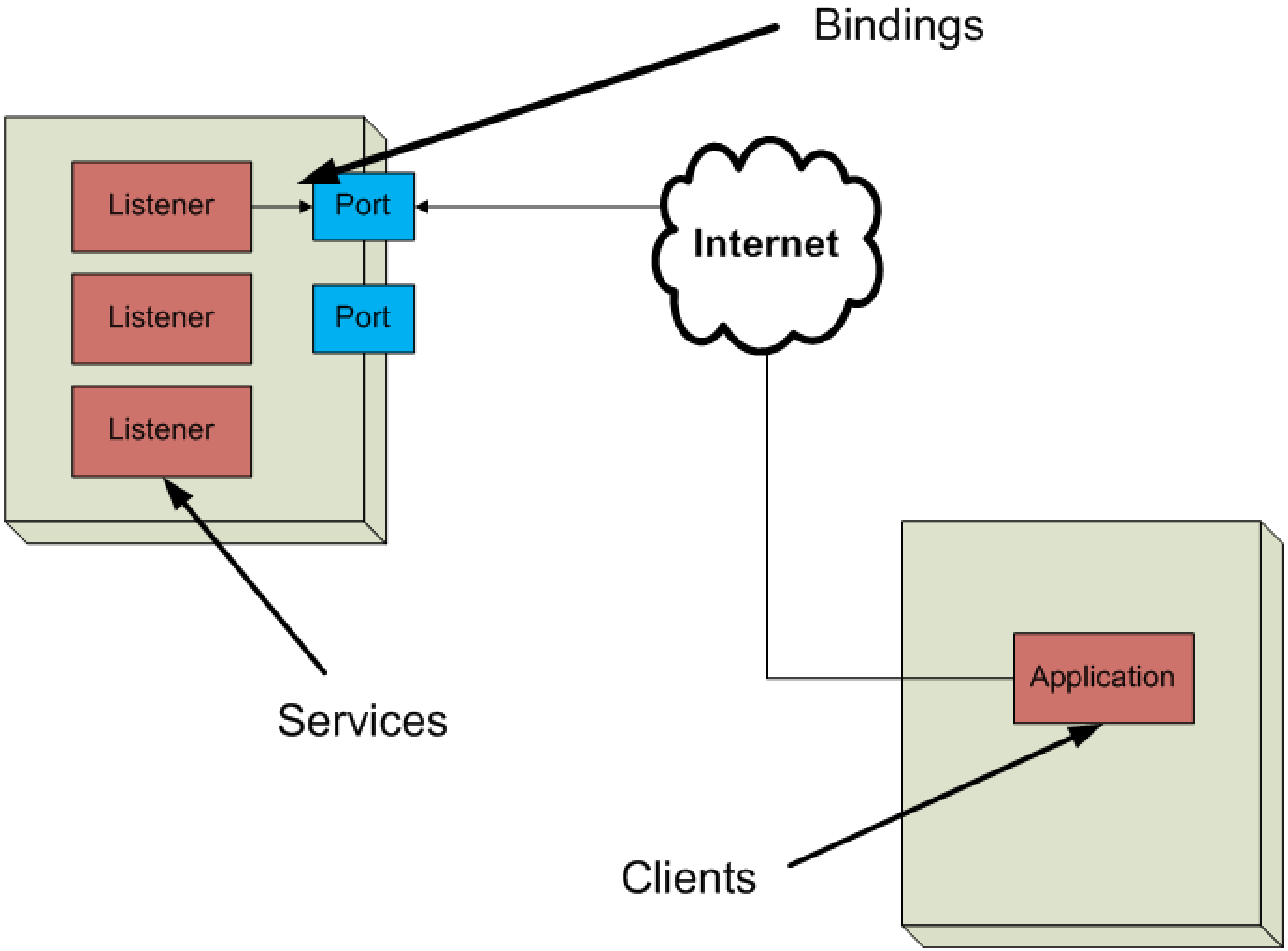


http://www.tcpipguide.com/free/t_TCPConnectionEstablishmentProcessTheThreeWayHandsh-3.htm

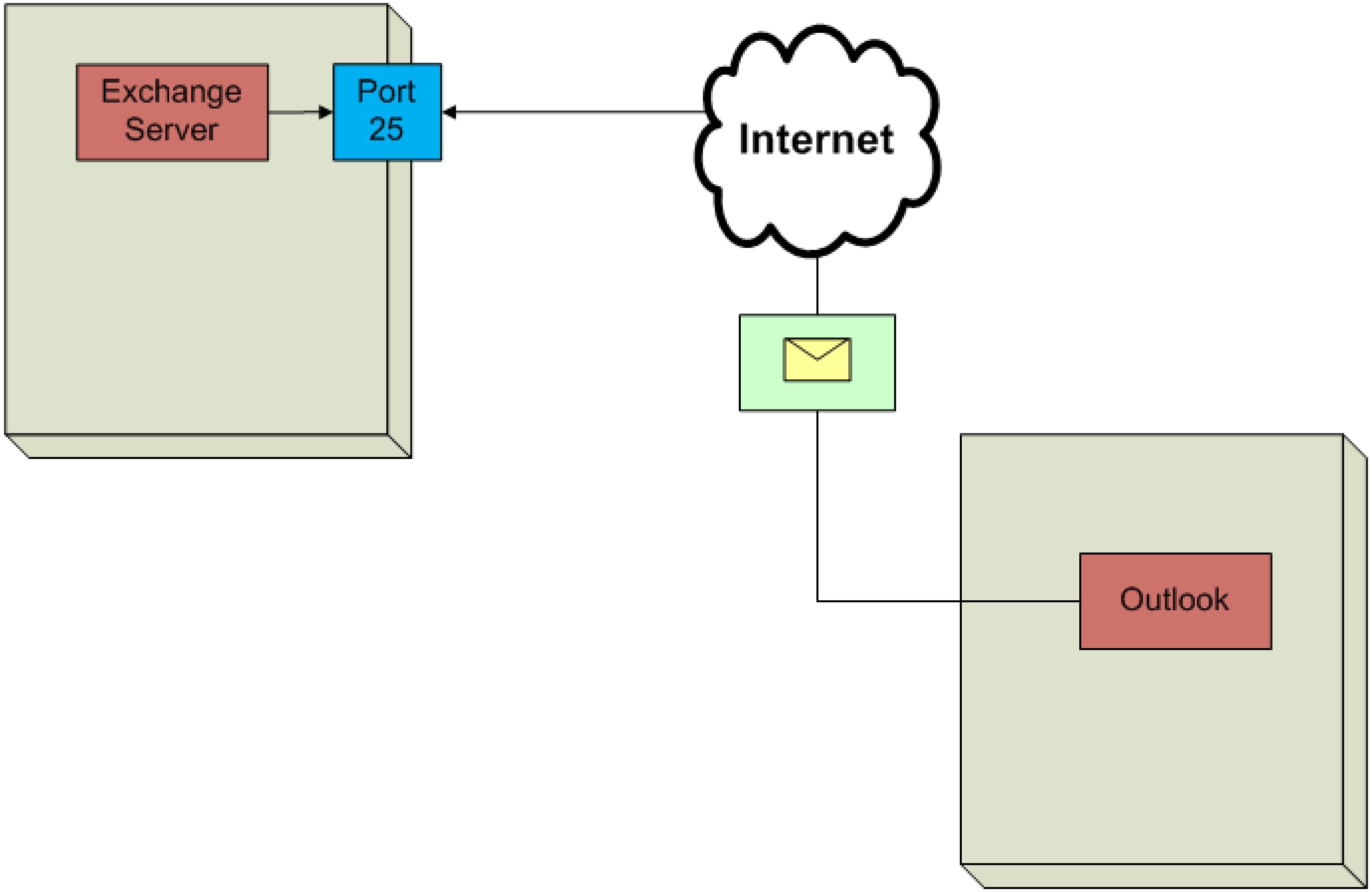
TCP Connection Steps



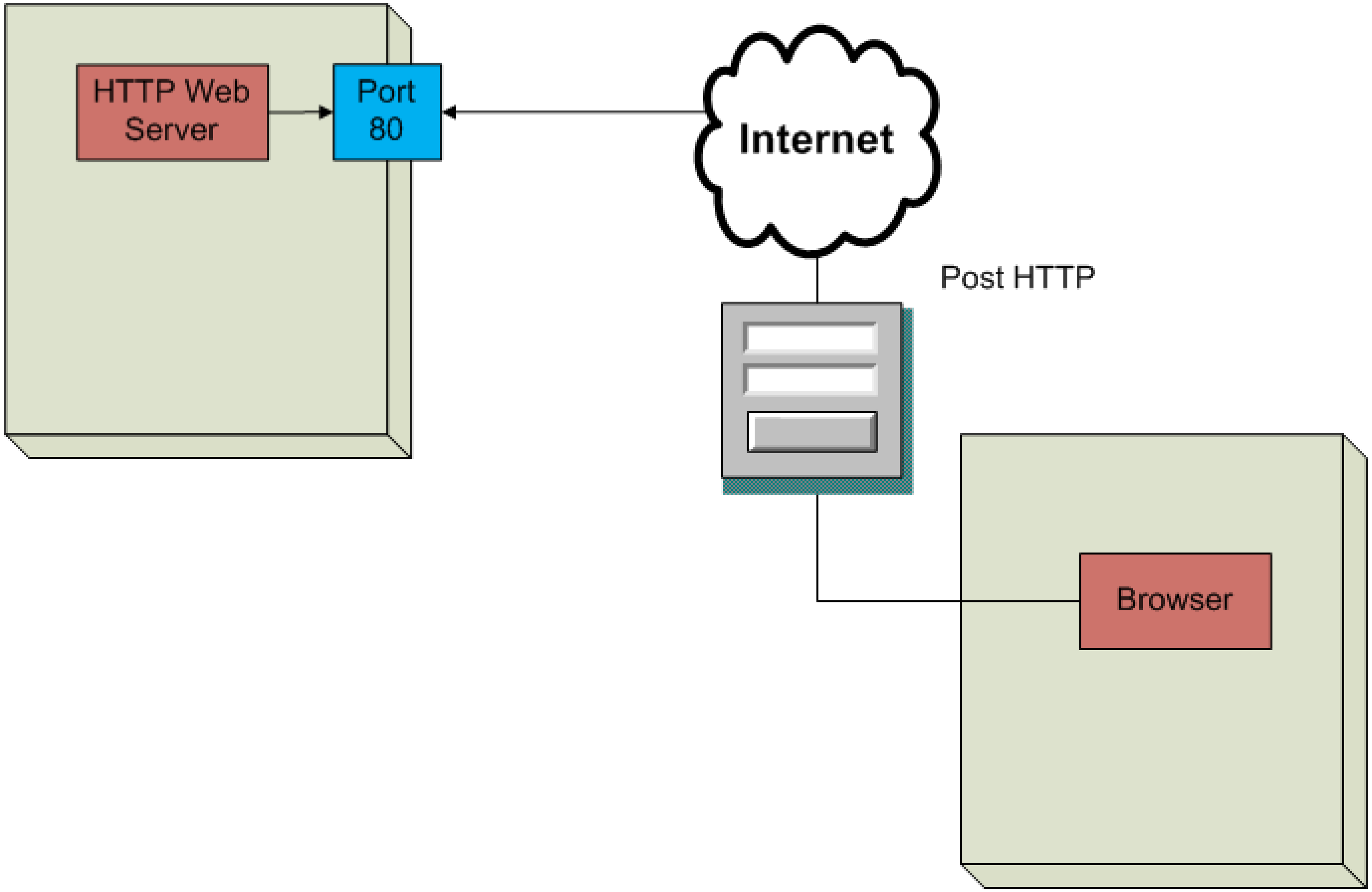
Server TCP/UDP Ports



Example : Email



Example: Web Server



DNS

- Independent of IP
- Tables of symbolic names that get translated to IP addresses.
- Correspondence to local networks.
- Records are cached around the world on DNS servers, and updates are propagated slowly.

Firewalls

- Border router between intranet and larger internet
- Service control – which internal services can be presented, which external service requests are allowed.
  - E.g., workers only visit approved web sites; only official internal web servers allowed, Inspect all IP packets, http requests, tcp/udp headers.

- Behavior control by inspecting content
  - E.g., controlling spam, pornography
- User control – differing levels of authority by user identification.

Filters

- IP packet filtering – discard packets that are not allowed. Inspect addresses, service type, compare service to data
- TCP gateway – monitor the setting up of TCP connections, check TCP segments for correctness, watch for known attacks
- Application gateway - run proxies for applications to control access. E.g. telnet only to certain locations.

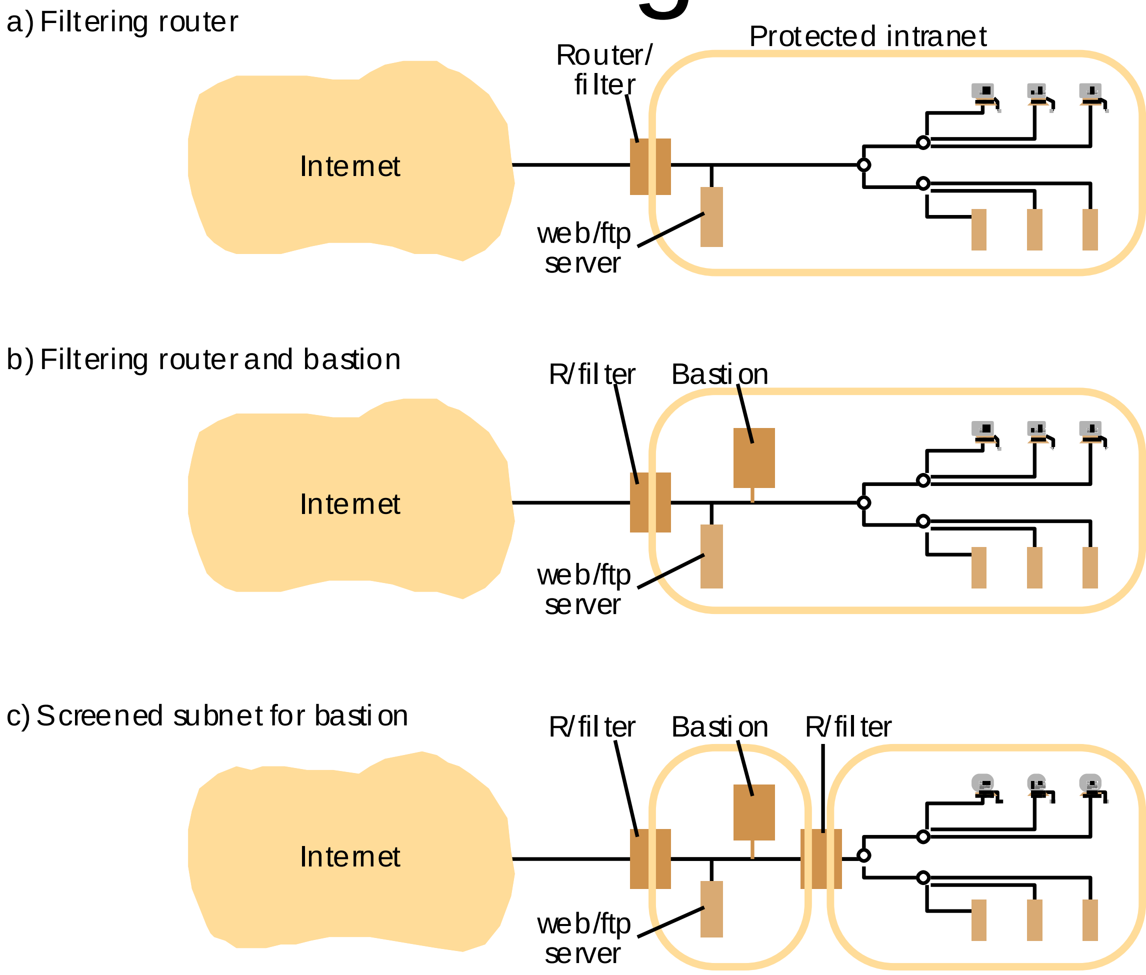
Collection of processes / boxes

- A firewall may be a collection of cooperating processes working at different protocol levels.
- May have more than one dedicated computer for efficiency and reliability
- Firewalls are computers and processes that, themselves, can be attacked.

“Bastion”

- Firewall that runs TCP and application-level software, usually on a separate computer.
- Policy may require that proxies are run for all outside services.
- May have firewalls run in series, to protect internal address from outside inspection.

Figure 3.21
Firewall configurations



a) Filtering router

b) Filtering router and bastion

c) Screened subnet for bastion

IPsec (“IP security”)

- Operates at a lower level than TSL, SSL, so applicable to wide range of applications including basic TCP and UDP without requiring rewrite apps. Provides:
  - Encrypting traffic (so it cannot be read by parties other than those for whom it is intended)
  - Integrity validation (ensuring traffic has not been modified along its path)
  - Authenticating the peers (ensuring that traffic is from a trusted party)
  - Anti-replay (protecting against replay of the secure session).
- Uses IKE internet key exchange
- Mandated in IPv6, but forced onto IPv4 by need.

VPNs

- Virtual Private Networks
- Based on IPsec
- Typical: home ISP user exchanges keys with firewall, and can then access intranet
- Can also connect two intranets

Figure 3.22
IEEE 802 network standards

IEEE No.	*Name*	*Title*	*Reference*
802.3	Ethernet	CSMA/CD Networks (Ethernet)	[IEEE 1985a]
802.4		Token Bus Networks	[IEEE 1985b]
802.5		Token Ring Networks	[IEEE 1985c]
802.6		Metropolitan Area Networks	[IEEE 1994]
802.11	WiFi	Wireless Local Area Networks	[IEEE 1999]
802.15.1	Bluetooth	Wireless Personal Area Networks	[IEEE 2002]
802.15.4	ZigBee	Wireless Sensor Networks	[IEEE 2003]
802.16	WiMAX	Wireless Metropolitan Area Networks	[IEEE 2004a]

Ethernet

- Cheap, reliable, local area network
- Long time defacto standard at 10Mbps
- 100 Mbps is cheap, and 1000 Mbps available.
- Broadcast of all packets
- NICs listen to all traffic and normally ignore all that does match, pass those that do up the protocol stack.
- Separate send and receive – don't start send if other is.
- Collisions can occur resulting in resends, and ultimately thrashing if overloaded. Longer cable / more collisions.
- Use of switches/software, reduces collisions & helps guarantee latency and delivery for QoS. Not obsolete yet

Ethernet

- Uses CSMA/CD:
- Sense if the line is free
- Every computer has full-time access – Multiple Access
- Detect collisions and resend after a random delay.
- Medium Access Control (MAC) is protocol, hence *MAC* addresses are used in hrdwre

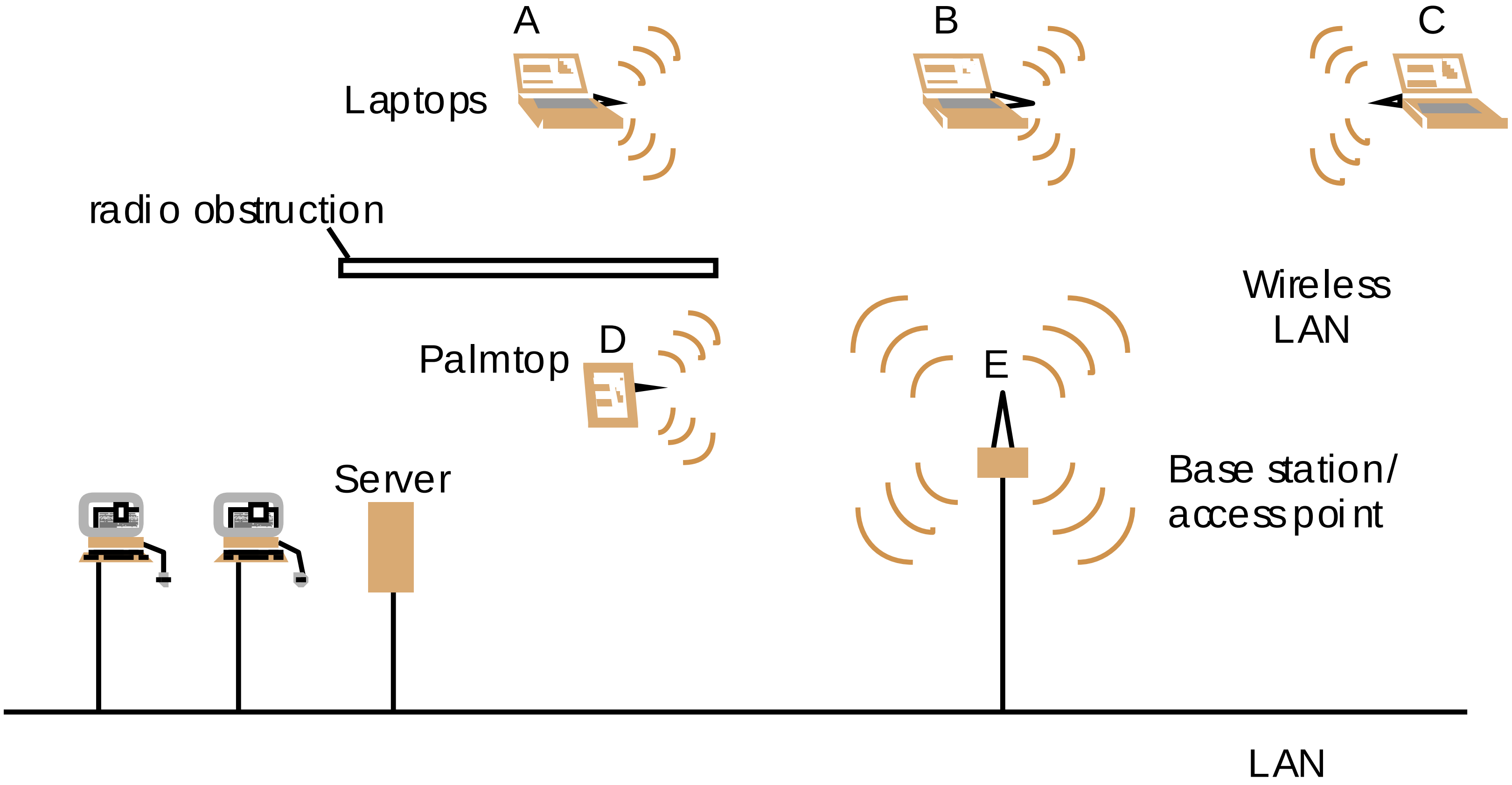
Figure 3.23
Ethernet ranges and speeds

	10Base5	*10BaseT*	*100BaseT*	*1000BaseT*
Data rate	10 Mbps	10 Mbps	100 Mbps	1000 Mbps
Max. segment lengths:				
Twisted wire (UTP)	100 m	100 m	100 m	25 m
Coaxial cable (STP)	500 m	500 m	500 m	25 m
Multi-mode fibre	2000 m	2000 m	500 m	500 m
Mono-mode fibre	25000 m	25000 m	20000 m	2000 m

WiFi

- 802.11g (2.4 GHz) and 802.11a (5 GHz) operate at 54 Mbps
- MAC (media acces control) is broadcast protocol, all can send and receive.
- More lost packets, more collisions:
  - Carrier signal is not consistent.
  - Cannot detect sends of others (e.g., A and D in figure).
  - Nodes may be in range of hub, but not each other.
  - Local radio signal dwarfs remote signal, so collisions not detected.

Figure 3.24
Wireless LAN configuration



Slot reservation extension to Ethernet

- Collision avoidance augmentation via *CSMA/CA;* reserves transmission slots.
- Short negotiation packets (Request to send, RTS; clear to send, CTS) are sent first between sender and receiver, with duration. Others wait until slot has passed.
- Fewer collisions with short packets.
- Improves wireless reliability and QoS.

Bluetooth IEEE 802.15, 2002

- Developed by Ericsson.
- Voice and data
- $5 per device
- Low battery consumption – several hours even with earpiece battery. Wake up and listen for a paging call, so long latency to establish contact.
- Adaptable power transmission from 1 Mwatt (10 meters) to 100Mwatt (100 meters)
- Switches 1600 times / second between 79 sub-bands of 2.4 GHz public frequency for reduced interference

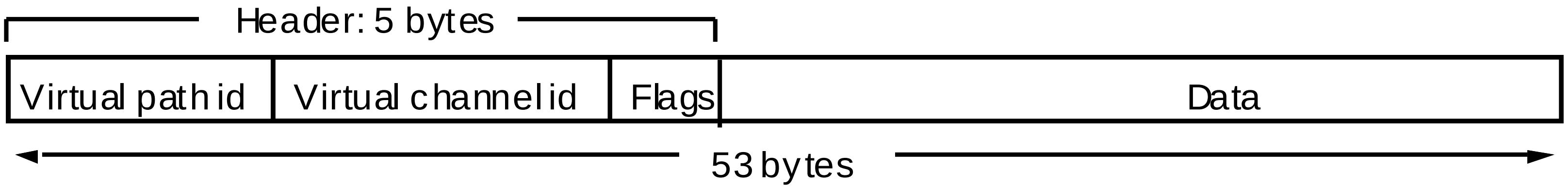
Master / slave

- Pairs operate as master / slave, with master allocating communication slots
- Piconet – up to 7 slaves one master and up to 255 *parked* (dormant) slaves in low power mode waiting for a wakeup signal.
- Scatternet – linked piconets.
- Most devices can be master or slave.
- All devices have 48-bit GUIDs, but slaves assigned 1-7 to save space.

Bluetooth…

- Designed for QoS required of audio.
- Synchronous and async transfer.
- Up to 1 Mbps
- Includes specifications for app-level protocols to encourage interoperation of manufactured devices.
- Association of new / parked devices is slow, up to 10 seconds – so not for e.g., toll road pickup of vehicles.
- Version 2.0: 3 Mbps (cd quailty audio), larger piconets, faster association is in the works.

Figure 3.27
ATM cell layout

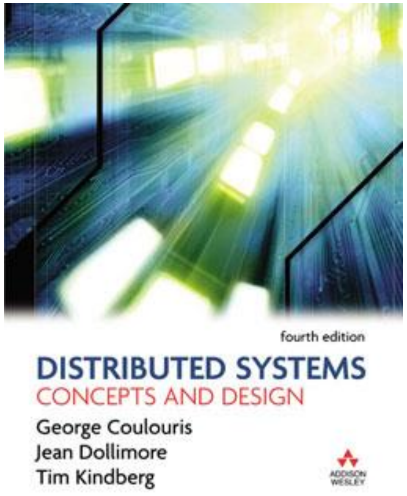


ATM – asynchronous transfer mode

- Fast packet switching w/ suitable QoS for multimedia transmission
- Avoids flow control and error checking
- Fixed-length units of data (cells) 53 bytes
- Establishes a connection first, guarantees bandwidth and latency. Frame relay does not store cell/frame.
- High-speed switching, low latency
- Similar speed to LANs.
- High QoS and speeds of 600Mbps for multimedia will be available for internet.
- Pure ATM up to 1 Gbps.
- Slow to adopt because of expense compared to e.g., 1000Mbps ethernets.

Blank

Figure Slides for Chapter 4: Interprocess Communication



From Coulouris, Dollimore and Kindberg
Distributed Systems:
Concepts and Design

Edition 4, © Pearson Education 2005

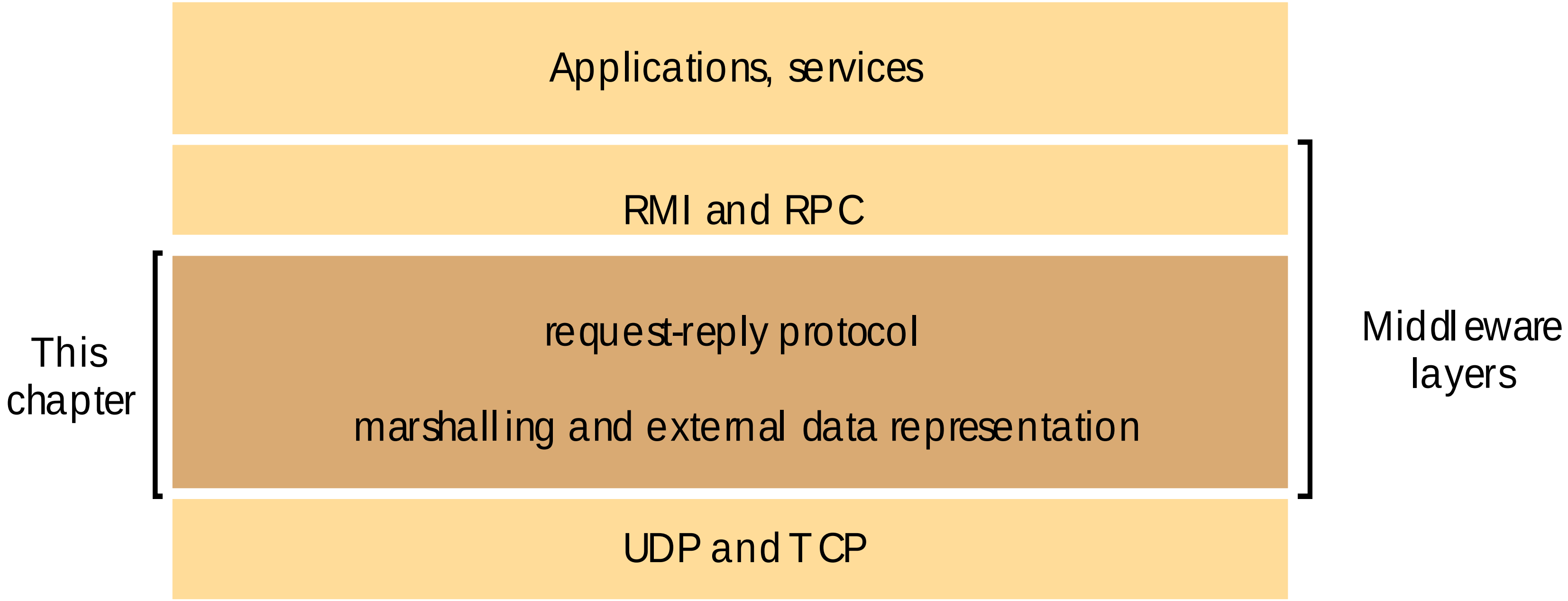
CSC435 DePaul -- Elliott

- Slides prepared by…
  - – Professor Clark Elliott
  - – DePaul University
- Some slides may be courtesy of Mark Goetsch
  - – DePaul University

Overview of Chapter

- Introduction
- API for internet protocols
- External data representation and marshalling !!
- Client/server communication. Request/reply protocols.
- Group communication

Figure 4.1
Middleware layers



API for Internet Protocol

- From the programmer’s point of view:
- UDP – pass a single *message* from from one process to another. *Datagrams*
- TCP – create a two-way stream between processes. Foundation for producer/consumer communication.

Synchronous vs. Ansychronous Calls!

- Fundamental to understanding DSes.
- Synchronous calls are like procedure calls in a program, caller waits for the return.
- Asynchronous (that is, not-synchronous) calls are entirely different and require a whole new programming paradigm. Many variations. All release the caller at some point, so there is less waiting. [Most] require re-establishing contact with the caller.

Synchronous calls

- Caller initiates contact with the receiver, sends request/call/data and *waits* until the receiver returns a response, then continues.
- Caller is *blocked* and can do nothing until the return.
  - Inefficient
  - Caller may *hang* on bad receiver

Asynchronous Calls

- Caller sends request/call/data and then proceeds to useful local work *without waiting.*
- Caller must then be *interruptable*, that is, drop everything and process the return, when the response comes. This is complicated to program.
  - More efficient
  - More complex
  - Caller will not hang

Write your own async. call

- Because of the complexity of responding to an asynchronous response, this is left to the application programmer.
- Multiple threads in one process allow a blocking call -- where one thread waits for the response -- to effect asynchronous calls.
- Threads are synchronized as needed.
- Imagine *INET* client as asynchronous

Sending

- Never seen same port for multiple senders. Usually use *get-next-available-port* call to find a spare port and send with that.

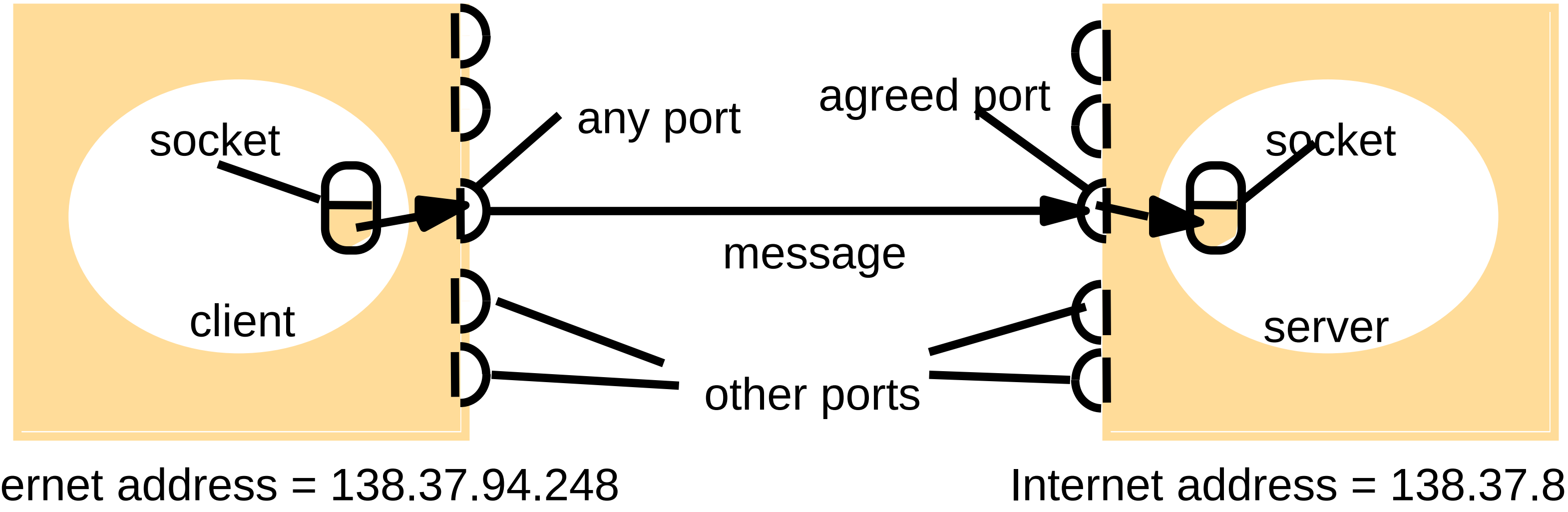
Binding to a server

- Always use a fixed port (e.g. telnet)
- Client refers to service by name and a *name server* or *super server* translates the name into a port number at run-time.
- Some operating systems (Mach) do this translation at a low level. Some opsys (V system) address processes.
- Allows servers to migrate as needed.

Ports

- Can be fixed, e.g. http, ftp, etc.
- Only loosely coupled to processes, so multiple ports in and out can be used.
- Allows use of “throw away” ports as needed – e.g. client send in http.
- Easy to manage in application tables – just a number for a slot in the API.

Figure 4.2
Sockets and ports


socket
client
any port
agreed port
message
other ports
socket
server
Internet address = 138.37.94.248
Internet address = 138.37.88.249

UDP

- Universal Datagram Protocol
- No ack, no resend, no guarantee; cheap!
- Receive method returns address and port of sender, as well as the data.
- Receiver block, sender non-block is usual
- Datagrams arrive on receive port from any sender, in any order, is usual but…
- Can form a direct “connection” between processes which excludes other processes (but still does not guarantee order)

UDP “failures”

- No duplication
- Omissions can occur
- Datagrams can arrive out of order (and also mixed in with those of other conversations)
- Cheap because:
  - No *state* stored at source / destination
  - No extra messages transmitted (ack/syn)
  - Less latency b/c no setup

Hmmm -- Write comments in our code?

- m is message, comes from arg 0
- aSocket is opened to server
- aHost is server IP address
- ServerPort is arbitrary, why not from args?
- reply is datagram response
- If you play, do yourself a favor and put in messages about processing of requests.

Figure 4.3
UDP client sends a message to the server and gets a reply

```
import java.net.*;
import java.io.*;
public class UDPClient{
  public static void main(String args[]){
          // args give message contents and server hostname
          DatagramSocket aSocket = null;
           try {
                    aSocket = new DatagramSocket();
                    byte [] m = args[0].getBytes();
                    InetAddress aHost = InetAddress.getByName(args[1]);
                    int serverPort = 6789;
                    DatagramPacket request = new DatagramPacket(m,  args[0].length(), aHost, serve
                    aSocket.send(request);
                    byte[] buffer = new byte[1000];
                    DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
                    aSocket.receive(reply);
                    System.out.println("Reply: " + new String(reply.getData()));
           }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
           }catch (IOException e){System.out.println("IO: " + e.getMessage());}
          }finally {if(aSocket != null) aSocket.close();}
  }
}
```

Figure 4.4: UDP server repeatedly receives a request and sends it back to the client

```
import java.net.*;
import java.io.*;
public class UDPServer{
            public static void main(String args[]){
            DatagramSocket aSocket = null;
               try{
                        aSocket = new DatagramSocket(6789);
                        byte[] buffer = new byte[1000];
                        while(true){
                           DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                          aSocket.receive(request);
                          DatagramPacket reply = new DatagramPacket(request.getData(),
                                  request.getLength(), request.getAddress(), request.getPort());
                          aSocket.send(reply);
                        }
               }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
              }catch (IOException e) {System.out.println("IO: " + e.getMessage());}
            }finally {if(aSocket != null) aSocket.close();}
   }
}
```

TCP -- ServerSocket

- (Note: TCP is generally discussed elsewhere and also used in class programs)
- ServerSocket (doorbell socket) blocks while listening for requests, result of request is a Socket, and ServerSocket goes back to listening.
- Queue size – drop requests if full

TCP -- Socket

- Used by pair of processes with a connection.
- Construction creates a local socket and port, and also connects to remote
- Throws unknown host, I/O, exceptions
- Provides getInputStream and getOutputStream

Figure 4.5: TCP client makes connection to server, sends request and receives reply

```
import java.net.*;
import java.io.*;
public class TCPClient {
          public static void main (String args[]) {
          // arguments supply message and hostname of destination
          Socket s = null;
            try{
                    int serverPort = 7896;
                    s = new Socket(args[1], serverPort);
                    DataInputStream in = new DataInputStream( s.getInputStream());
                    DataOutputStream out =
                              new DataOutputStream( s.getOutputStream());
                    out.writeUTF(args[0]);           // UTF is a string encoding see Sn 4.3
                    String data = in.readUTF();
                    System.out.println("Received: "+ data) ;
            }catch (UnknownHostException e){
                              System.out.println("Sock:"+e.getMessage());
            }catch (EOFException e){System.out.println("EOF:"+e.getMessage());
            }catch (IOException e){System.out.println("IO:"+e.getMessage());}
          }finally {if(s!=null) try {s.close();}catch (IOException
e){System.out.println("close:"+e.getMessage());}}
          }
}
```

Figure 4.6 TCP server makes a connection for each client and then echoes the client’s request

```
import java.net.*;
import java.io.*;
public class TCPServer {
    public static void main (String args[]) {
            try{
                    int serverPort = 7896;
                    ServerSocket listenSocket = new ServerSocket(serverPort);
                    while(true) {
                            Socket clientSocket = listenSocket.accept();
                            Connection c = new Connection(clientSocket);
                    }
            } catch(IOException e) {System.out.println("Listen :"+e.getMessage());}
    }
}

// this figure continues on the next slide
```

Figure 4.6 continued

```
class Connection extends Thread {
        DataInputStream in;
        DataOutputStream out;
        Socket clientSocket;
        public Connection (Socket aClientSocket) {
          try {
                clientSocket = aClientSocket;
                in = new DataInputStream( clientSocket.getInputStream());
                out =new DataOutputStream( clientSocket.getOutputStream());
                this.start();
           } catch(IOException e)  {System.out.println("Connection:"+e.getMessage());}
        }
        public void run(){
          try {                                         // an echo server
                String data = in.readUTF();
                out.writeUTF(data);
          } catch(EOFException e) {System.out.println("EOF:"+e.getMessage());
          } catch(IOException e) {System.out.println("IO:"+e.getMessage());}
          } finally{ try {clientSocket.close();}catch (IOException e){/*close failed*/}}
        }
}
```

Marshalling and External Data Rep.

- Program data is random access, stored in memory
- Messages are serial, sent bit-after-bit on tramission lines.
- Marshaling (or Marshalling) (and un-) is the process of translating one into the other.
- The problem of automating this is generally unsolvable, & specifically hard.

Marshalling – why hard

- Lots of computer architectures: http://en.wikipedia.org/wiki/Instruction_set#ISAs_implemented_in_hardware
- E.g. big-endian vs. little-endian integers
- Floating point reps. for rapid calculations
- Text: ASCII seven bits, Unicode two bytes, EBCIDIC 8 bits.
- Chosen to go fast, not be compatible

Structure of translation

- Agree on external format and translate at both ends (e.g., CDR, XML)
- Sender translates for receiver (not common?)
- Sender sends agreed meta-information (or is implicit in application) and receiver translates

Why impossible

- Automated marshalling means that without knowing anything about the data, a low level algorithm can marshal it.
- Random access cyclically linked list. What does Null pointer mean? Suppose application programmer uses 237 as the null pointer?
- Union objects: two data types, but one set of bits. – offsets and alignments?
  - Example: boolean bit array set as a number – furnace error codes

Serialization, “flattening”

- To store random access memory objects on disk, or to send them on network lines, the process is the same.
- Serialization is the process of creating a string of bits, a “series,” to be processed one after another, from a collection of bits that can all be immediately accessed in any order.
- Thus all messages, and serialized objects, can also be stored on disk, written to a printer, written to tape.
- This has profound implications for *time* and *canonical sinks* of data.

Odd semantics for objects

- The object can be written to disk, and then restored in 1,000 years.
- The object can be written to disk, and then restored in five new locations every month. Which is the canonical representation?
- An object can *leave* a group, *join* another group, but still be in the first group. It can return in a different state than the object that did not leave.
- An object can be *nowhere* but later restored.
- An object can be half in one location and half in another, during transmission.

Introspection

- Sometimes objects have information about their class available. This has to be marshaled as well.
- Java supports introspection in its serialization.
- CDR sends only the data and assumes knowledge of the objects.

XML namespaces

- Information about contained objects, but may be in an external reference.
- XML namespaces sometimes use URLs as unique strings (UUIDs), and have nicknames within a document.
- Any named item in the document with the nickname thus made globally unique.
- E.g., “ID_num” can appear in many documents, but is made unique by the doc’s namespace

CORBA’s CDR

- Common Object Request Broker Architecture, Common Data Represntation
- External serial form written, and read, by many different, unlike, platforms.
- 15 primitive types, plus constructed types
- No type information. Sender and Receiver must have prior knowledge. Use IDL?

Figure 4.7
CORBA CDR for constructed types

Type	*Representation*
sequence	length (unsigned long) followed by elements in order
string	length (unsigned long) followed by characters in order (can also can have wide characters)
array	array elements in order (no length specified because it is fixed)
struct	in the order of declaration of the components
enumerated	unsigned long (the values are specified by the order declared)
union	type tag followed by the selected member

Figure 4.8
CORBA CDR message

index in sequence of bytes	← 4 bytes →	*notes on representation*
0–3	5	*length of string*
4–7	"Smit"	*‘Smith’*
8–11	"h "	
12–15	6	*length of string*
16–19	"Lond"	*‘London’*
20-23	"on "	
24–27	1934	*unsigned long*

The flattened form represents a *Person* struct with value: {‘Smith’, ‘London’, 1934}

Java serialization

- When objects refer to other necessary objects, they have to be included as well.
- Version checking is performed – often very important in DSes, when services and clients can go off line.
- Class information is written, followed by types, and names of instance variables.
- References are serialized as *handles* referring to an object within the serialized form. The object is written just once. Thereafter the object is not written, just the handle.

RMI – Remote Method Invocation

- Sending objects over networks can be expensive when large class definitions must also be sent.
- But java has a big advantage: the virtual machine is already known to have many of the needed libraries stored locally, so they do not have to be sent!
- Remote references are also first class objects, and so can be sent as args.

Markup languages

- Have both content, and information about the structure (or appearance) of the content.
- Have been around for a long time as part of the printing industry – instructions to typesetters
- SGML very complex, very powerful. 1960s, 2nd ed. of OED, gov. machine-readable IT and law documents
- XML for structured docs for the web, 1998

XML

- Uses tags to describe general structure of contained content
- Associate A/V pairs with logical structure
- Universally readable, so good for archives, cross-platform data, differing applications
- Allows clear configuration files
- Somewhat human readable for debugging
- Can be ‘X’tended with user-defined tags
- All data is represented by characters. Can use base64 encoding for binary data (alphanumeric with +/=)

XML character encoding

- Specified in the prolog
- UTF-8 is default. 1-4 bytes variable, with 1-byte for *us-ascii.*
- ISO-8859-1, us-ascii, other 8-bit encodings

Figure 4.10 XML definition of the Person structure

```
<person id="123456789">
        <name>Smith</name>
        <place>London</place>
        <year>1934</year>
        <!-- a comment -->
</person >
```

Figure 4.11 Illustration of the use of a namespace in the *Person* structure

```
<person pers:id="123456789" xmlns:pers = "http://www.cdk4.net/person">
        <pers:name> Smith </pers:name>
        <pers:place> London </pers:place >
        <pers:year> 1934 </pers:year>
</person>
```

XML Schemas

- Used to formalize the structure of an XML document ahead of time.
- Allows apps to validate for conformity, and can help with parsing.
- Intended that many documents will share a single structure, or *scheme*
- [DTDs Document Type Definitions still widely used for documents. Older technology, less powerful, different syntax]

Figure 4.12 An XML schema for the Person structure

```
<xsd:schema  xmlns:xsd = URL of XML schema definitions   >
    <xsd:element name= "person" type ="personType" />
        <xsd:complexType name="personType">
            <xsd:sequence>
                <xsd:element name = "name"  type="xs:string"/>
                <xsd:element name = "place"  type="xs:string"/>
                <xsd:element name = "year"  type="xs:positiveInteger"/>
            </xsd:sequence>
            <xsd:attribute name= "id"   type = "xs:positiveInteger"/>
        </xsd:complexType>
</xsd:schema>
```

Parsing XML

- Virtually every modern programming language has XML parsers
- See the MIMER assignment for a very powerful construct based on XML parsing libraries for java.

XML vs. CDR efficiency

- XML is Larger than binary counterparts – (but HTTP 1.1 *content-encoding* allows for compression if browser and server are enabled).
- SOAP (for Web services) used with XML
- Section 19.2.4 of CDK: SOAP-XML 14x as large as CDR; SOAP request 882x as long as CORBA invocation.
- Consider a large array of integers: binary in CDR, UTF-8 text in XML.

Comparisons

- CORBA is the heavy-hitter. Allows great efficiency, highly structured, designed to last well into the future. Large start-up costs to study the complex standards and relationships to various languages. Somewhat cumbersome to code. Not so popular b/c of complexity.
- SOAP, XML. Easier to get started. Standards evolving. Less efficient.
- As SOAP develops, starts to look as complex as CORBA to understand. ☺
- RMI limited to java but easy to code.
- RPC is old standard. Lots of legacy code, but low-level programming without a lot of standards. Much built ON TOP of RPC, which is hiding under the hood.

Remote object references

- Should be unique in DSes. If an object is deleted, the reference should return an error, not refer to some other object.
- Including the IP address and port number of an object in the reference helps to locate it, but does not allow the object to relocate.
- May need a more sophisticated locating service. Routing of messages is an area of study.

General Client/Server message exchanges

- UDP might be suitable, since replies can act as acknowledgements (acks).
- Request – cheap, no ack.
- Request / Reply – (*idempotent* op?)
- Request / Reply / Ack Reply

Figure 4.14
Request-reply communication

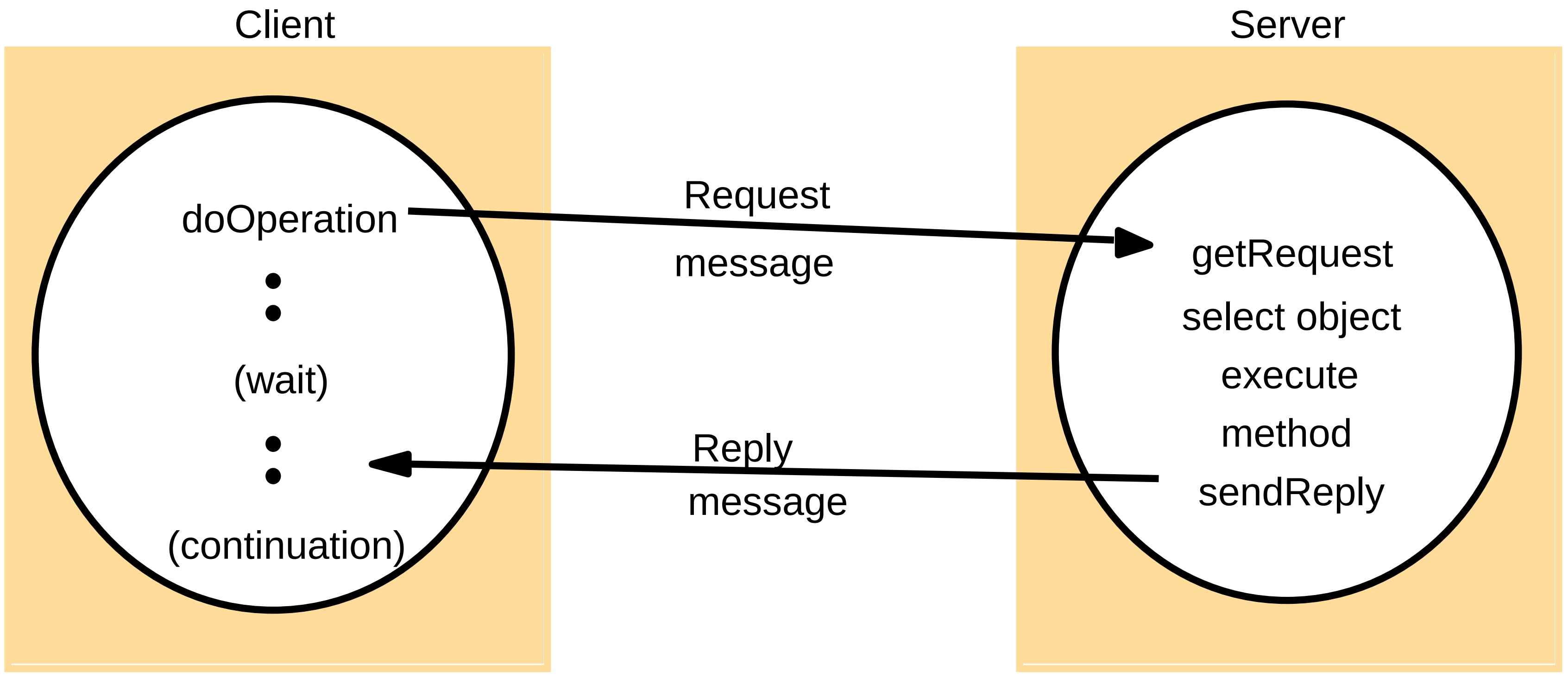


Figure 4.15
Operations of the request-reply protocol

public byte[] doOperation (RemoteObjectRef o, int methodId, byte[] arguments)

sends a request message to the remote object and returns the reply.
The arguments specify the remote object, the method to be invoked and the arguments of that method.

public byte[] getRequest ();

acquires a client request via the server port.

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);

sends the reply message reply to the client at its Internet address and port.

Many Variations

- Requests and replies may be synchronous or asynchronous.
- Asynchronicity may have synchronous components when waiting for acks:
  - Many places to ack: E.g., ack placement in sender queue; ack the receipt of request in the server queue; ack the beginning of processing of the request;
  - Server must be up and running or not? stored and non-stored requests

Message Identifiers

- Messages may get lost, delayed, sent more than once (on a restart), garbled, and arrive out of sequence.
- Make them unique with information:
  - Message ID – increasing sequence number from sender
  - ID of sending process. (IP address, port number, time?) [Already in UDP message?]

Failure decisions to be made

- Do we resend the request? Idempotent operation?
- Has the operation been performed?
- Do we care about multiple requests?
- Do we need to keep histories – for how long? Lost replies? Repeat or store result?
- Rollbacks of larger transactions?

Three RPC protocols

- Request only – e.g. send data from real-time sensors. Server can determine if sender is offline if needed?
- Request / reply – server may have to filter duplicate requests, save history (forever) to reprocess if necessary
- Request/reply/ack-reply only save history until the ack-reply arrives – unless transaction rollback! But do we want *server* to block on RRA?

Server / Client crash

- Client and server crash semantics can get quite complex, unlike on local systems.
- How often do we checkpoint? If we do not checkpoint after *each* client request, how do we tell client what to resend? History?
- Suppose C and S both crash: S crashes, sends note to resend after certain “time” (time stamp of last C request) but C has crashed and misses the message.

TCP instead of UDP?

- Messages may be longer than single datagram. More complex semantics for C/S protocols with multiple messages for each step.
- TCP has greater startup and overhead, but provides guarantees and long messages.
- Supports stream-oriented marshaling techniques such as java's.
- Re-use the connection?
- Combine with UDP?

HTTP -- example

- Client / Server RR architecture
- Uses TCP/IP
- Later version, HTTP 1.1 allows temporary maintenance of connection for multiple requests of objects to build a page.
- Either side can close the connection.
- Considered idempotent at protocol level – but remember that server request might have side effects.

HTTP Methods / Actions

- Mostly self-explanatory
- Keep in mind that GET and POST both send data, and get data back. In practice either can be used; only the syntax is different.
- For that matter PUT probably could too. The server is just a program and the original intent of the gopher (“go – fer”) actions are long obsolete.

Group Communication

- Humans reason well about groups – we understand them.
- Groups are important to security
- Services can be grouped into organizations
- Replications of services and data can be grouped as duplicates all requiring synchronization

Multicast useful for…

- Fault tolerance for replicated services
  - Client requests broadcast to all services which perform identical operation.
- Useful for discovery of services in “spontaneously”-configured groups (16.2)
- Replicated data
  - Value is multicast to all data sinks
- Propagation of events

IP Multicast

- Not used as much as is supported!
- Addressed to computers at IP level, forwarded to procs
- Supported by IPv6, some IPv4 routers, UDP
- (*Mbone* to connect M-cast networks together)
- Sender does not know group, or have to belong
- Can receive as part of multiple groups
- Some support at Ethernet, and Internet levels
- Java: *joinGroup, leaveGroup, setTimeToLive*

Conflicts in Multicast space

- Time To Live (TTL) limits number of router hops allowed
- Requires a free Multicast IP address, but by setting TTL low, avoid conflicts
- Management problems for full internet not well solved (Handley, 1998: session directory [sd] program to browse, start, join multicast sessions)

Reliability and Order of MCast

- UDP: not reliable, not ordered
- Replica servers must be consistent, ouch!
- Discover of spontaneous services not so much a problem
- Data replication: retrieval performance not so bad, but distribution of updates, ouch!
- Event notification: reliability *and* order could be not so bad, or serious ouch.

Figure 4.20
Multicast peer joins a group and sends and receives datagrams

```
import java.net.*;
import java.io.*;
public class MulticastPeer{
        public static void main(String args[]){
         // args give message contents & destination multicast group (e.g. "228.5.6.7")
        MulticastSocket s =null;
         try {
                InetAddress group = InetAddress.getByName(args[1]);
                s = new MulticastSocket(6789);
                s.joinGroup(group);
                byte [] m = args[0].getBytes();
                DatagramPacket messageOut =
                        new DatagramPacket(m, m.length, group, 6789);
                s.send(messageOut);


         // this figure continued on the next slide
```

Figure 4.20 (continued)

```
                    // get messages from others in group
                            byte[] buffer = new byte[1000];
                            for(int i=0; i< 3; i++) {
                              DatagramPacket messageIn =
                                      new DatagramPacket(buffer, buffer.length);
                              s.receive(messageIn);
                              System.out.println("Received:" + new String(messageIn.getData()));
                            }
                            s.leaveGroup(group);
                    }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
                   }catch (IOException e){System.out.println("IO: " + e.getMessage());}
                 }finally {if(s != null) s.close();}
         }
      }
```

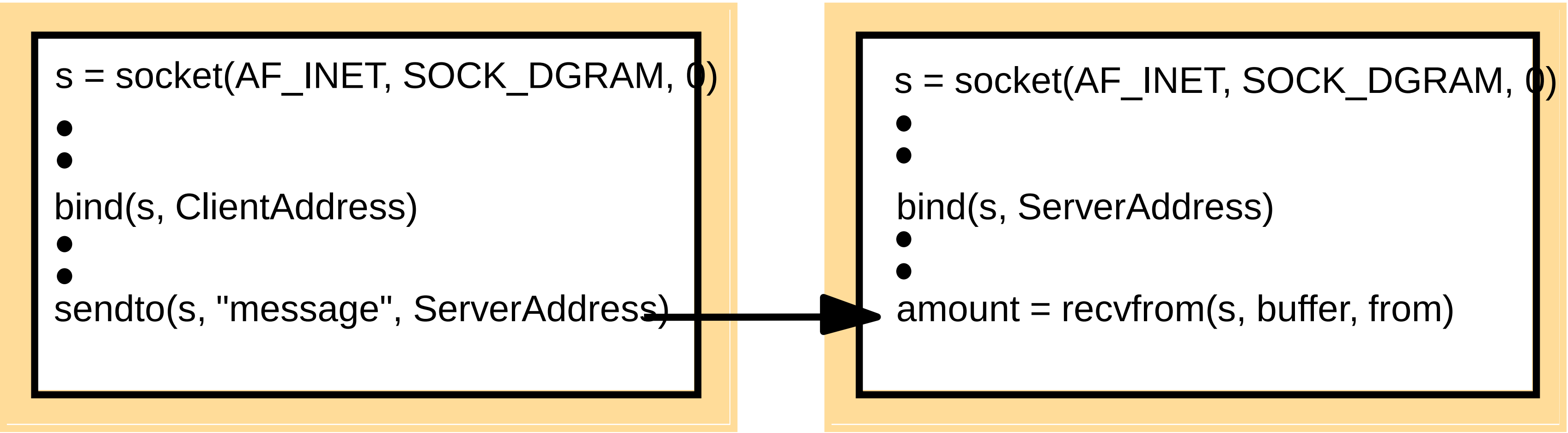
Stream Communication

- Exactly as in JokeServer and InetServer programs
- Server (doorbell) socket accepts connections, spawns a socket to continue conversation, goes back to listening.
- Read and write, both directions, through socket communication using TCP

Figure 4.21
Sockets used for datagrams

Sending a message

Receiving a message

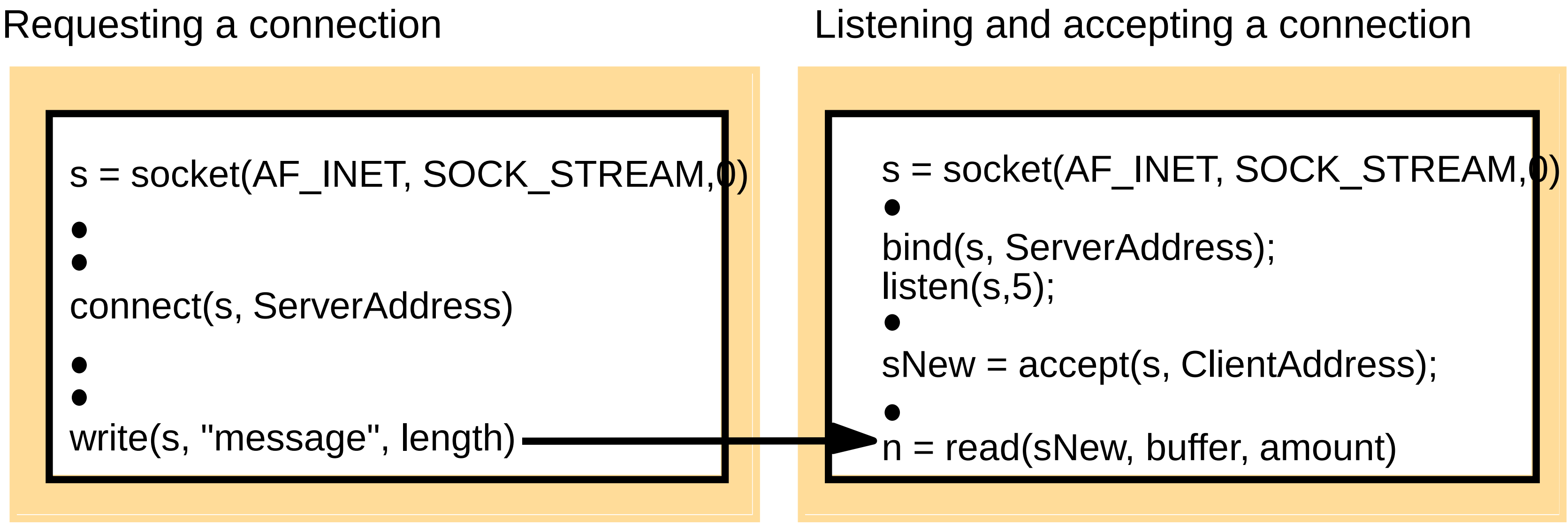


```
s = socket(AF_INET, SOCK_DGRAM, 0)
•
•
bind(s, ClientAddress)
•
•
sendto(s, "message", ServerAddress)
```

```
s = socket(AF_INET, SOCK_DGRAM, 0)
•
•
bind(s, ServerAddress)
•
•
amount = recvfrom(s, buffer, from)
```

ServerAddress and *ClientAddress* are socket addresses

Figure 4.22
Sockets used for streams



Requesting a connection

```
s = socket(AF_INET, SOCK_STREAM,0)
•
•
connect(s, ServerAddress)
•
•
write(s, "message", length)
```

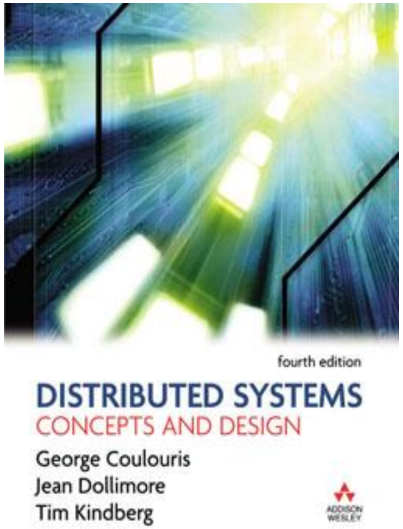
Listening and accepting a connection

```
s = socket(AF_INET, SOCK_STREAM,0)
•
bind(s, ServerAddress);
listen(s,5);
•
sNew = accept(s, ClientAddress);
•
n = read(sNew, buffer, amount)
```

ServerAddress and *ClientAddress* are socket addresses

Blank

Slides for Chapter 5:
Distributed objects and remote invocation



From Coulouris, Dollimore and Kindberg

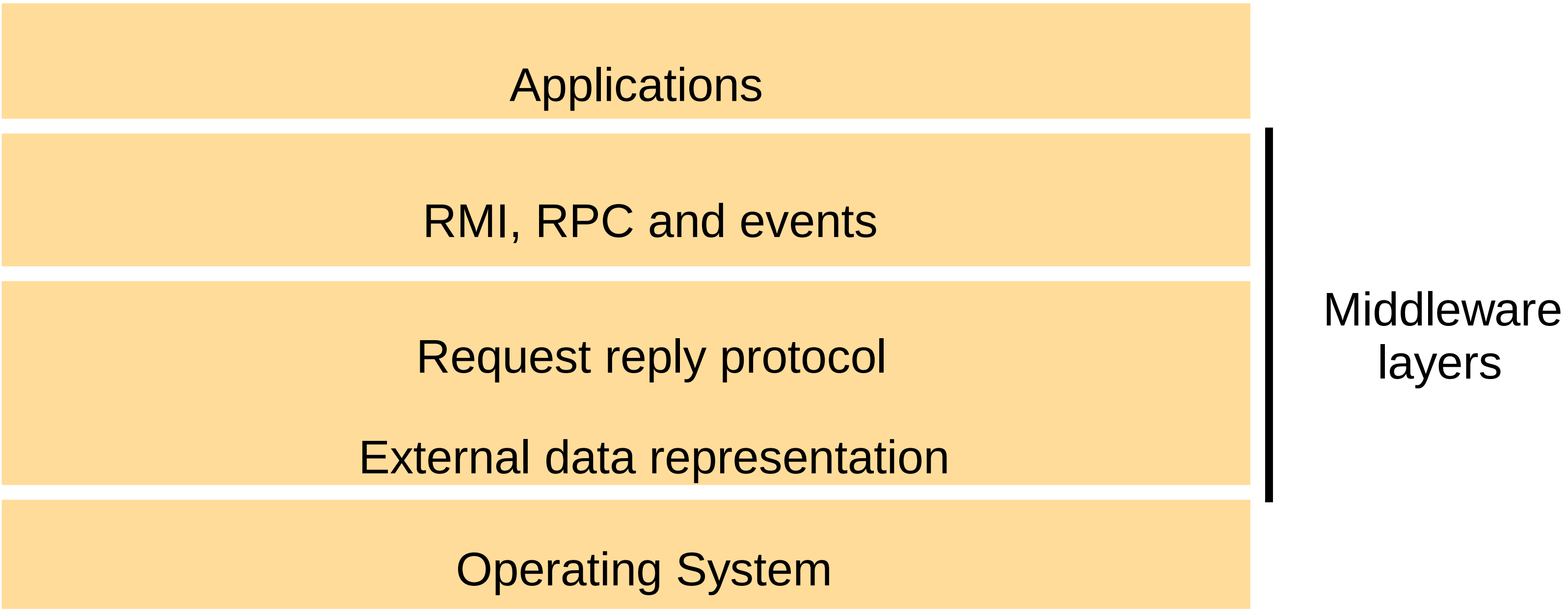
Distributed Systems:
Concepts and Design

Edition 4, © Addison-Wesley 2005

ECT425 DePaul -- Elliott

- Slides prepared by…
  - – Professor Clark Elliott
  - – DePaul University
- Some slides courtesy of Mark Goetsch
  - – DePaul University

Figure 5.1
Middleware layers



Applications

RMI, RPC and events

Request reply protocol

External data representation

Operating System

Middleware layers

The Object Model

- Objects encapsulate data and the methods that access it
- Objects can refer to other objects, and variables can hold these references.
- Methods can be invoked by one object to access data in other objects, usually passing arguments.
- Interfaces describe collections of methods (arguments, return values, exceptions). Objects may provide interfaces if they implement all the methods.
- Invoking a method may affect target object: change its state, instantiate a new object, invoke other methods on the same, or other, objects. Return values can be had.

Exceptions

- Much more common in “real” code
- Graceful change of control for non-customary processing keeps code clean
- Code blocks *throw* control to other code that *catches* it.

DSes are natural extension of OO programming

- Object programs distribute *process state* (roughly, the contents of a program’s variables) naturally in logically separate, encapsulated, locations already
- When clients and servers are in different processes, this mandates *encapsulation*, which makes software engineers happy
- Interfaces fully describe access, so implementation, including on unlike systems, is hidden.

Distributed Object Model

- An object is known as *remote* if it can be accessed remotely. Think *remotely-accessible*
- Objects must first obtain a reference to the remote object, then can invoke methods on it.
- Remote objects must present a *remote interface* – e.g. in Java can accept RMIs
- Remote object references are UUIDs and can be passed around

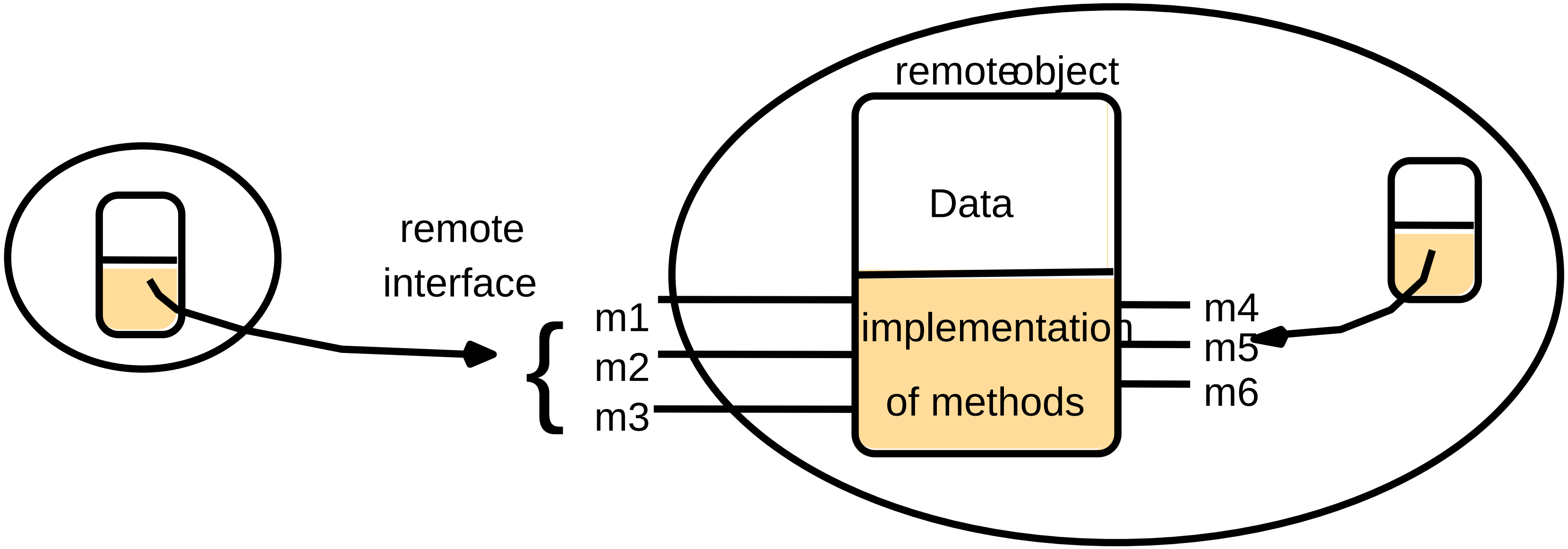
IDLs

- Interface Definition Languages are used for describing remote interfaces.
- All DS middleware systems provide IDLs
- The important part of IDLs is the rules. That IDL utilities (“IDL compilers”) also partially generate stub code is handy but not necessary.

Figure 5.2
CORBA IDL example

```
// In file Person.idl
struct Person {
        string name;
        string place;
        long year;
} ;
interface PersonList {
        readonly attribute string listname;
        void addPerson(in Person p) ;
        void getPerson(in string name, out Person p);
        long number();
};
```

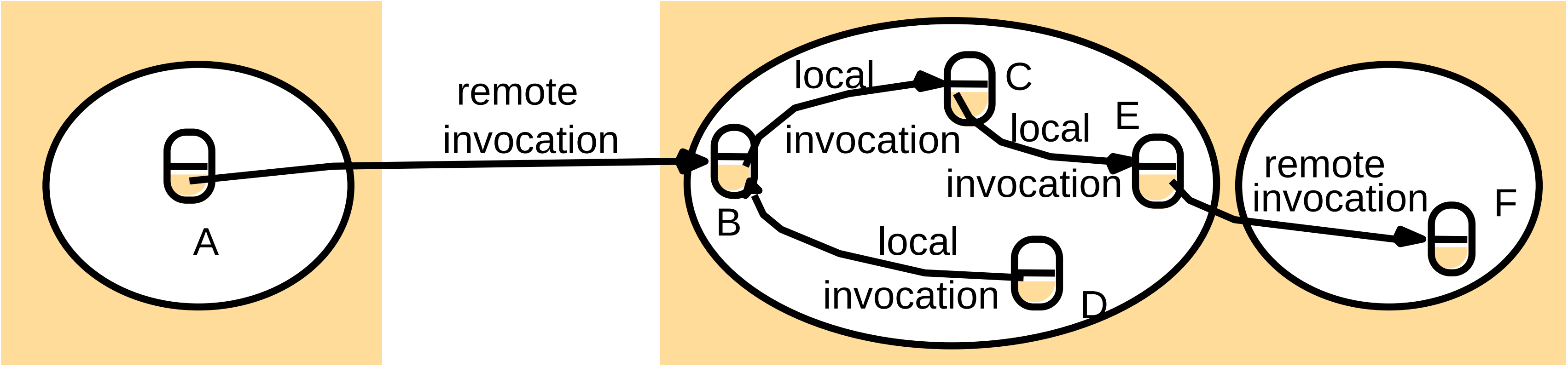
Figure 5.4
A remote object and its remote interface



Obtaining remote references

- In Fig. 5.3 Object A, which already has a reference to B, might also obtain a reference to F.

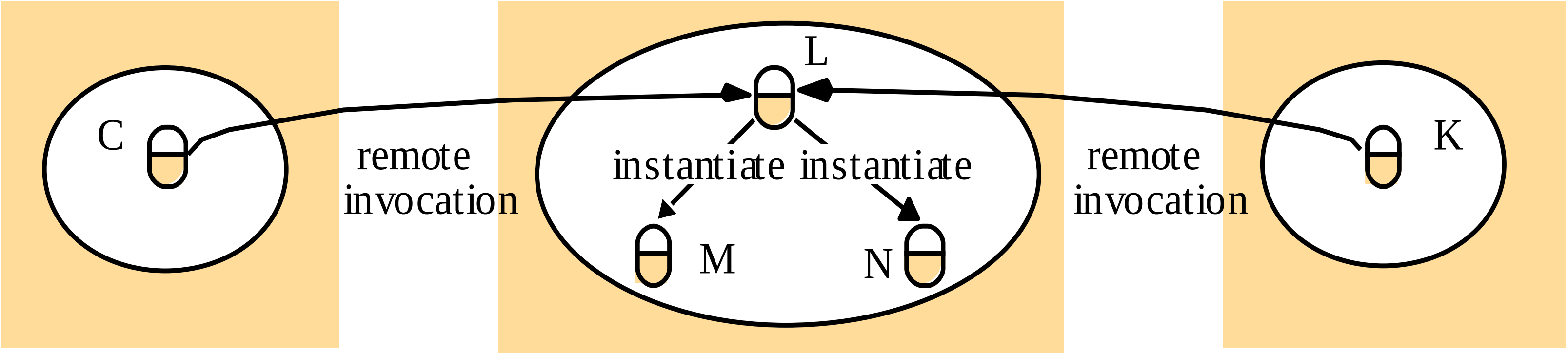
Figure 5.3
Remote and local method invocations



Instantiating remote objects

- Remote objects can be created locally.
- Fig. 5.5: If L provides remote methods to instantiate M and N then C and K can, effectively, also create remote objects.

Figure 5.5 Instantiation of remote objects



Remote garbage collection

- Much harder than the book implies – in fact the book handles this poorly on p 5.5
- The book says that a “reference counting” module added to the local garbage collection system usually will handle this
- So, in a system that can pass remote object references, how does reference counting work? Anyone can copy a UUID.

Request (Call) semantics

- Local method calls are performed exactly once.
- Retry request message
- Duplicate filtering at the server
- Retransmit results – keep history of results at server

Figure 5.6
Invocation semantics

Fault tolerance measures			*Invocation semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

Pick the cheapest semantics

- Saves development and maintenance time
- Simpler to understand

Maybe Invocation Semantics

- Remote method executed once or not at all
- If result message has not been received (e.g. omission/crash failures) it is not certain whether or not execution occurred
- Useful when occasional failures are acceptable.
- Saves money and time

At least once semantics

- Receive result, or message that operation was not performed
- Achieved through retransmission of requests
- Works for *idempotent* operations where duplicate requests are acceptable
- E.g., data collection that can later be filtered.

At most once semantics

- Either result is returned, or exception that no result was returned.
- Method invoked AT MOST one time.
- Resend the request to mask omission failures
- Resend results, but do not re-invoke method, to mask omission failures in the other direction.
- This is most typical

Objects that call remote methods

- If no result is returned, then in general cannot know if it is a network failure, or a server failure.
- Callers must be robust with respect to these kinds of ambiguity that do not occur with local calls. – Detection and recover?
- Latency is much greater, and much more varied.

How transparent?

- Transparency of local/remote methods could be good, or bad.
- Java RMI methods implement *remote* interface, and throw *RemoteExceptions* so the differences can be handled outside the ordinary application code.

Event-notification systems

- Heterogeneous: any kind of system just needs an event-receiving module and/or an event-generating module. This allows interaction of many unlike systems
- Asynchronous: publishers and subscribers do not have to synchronize – just publish and let subscriber worry about it. Persistent storage of events?

Figure 5.7
The role of proxy and skeleton in remote method invocation

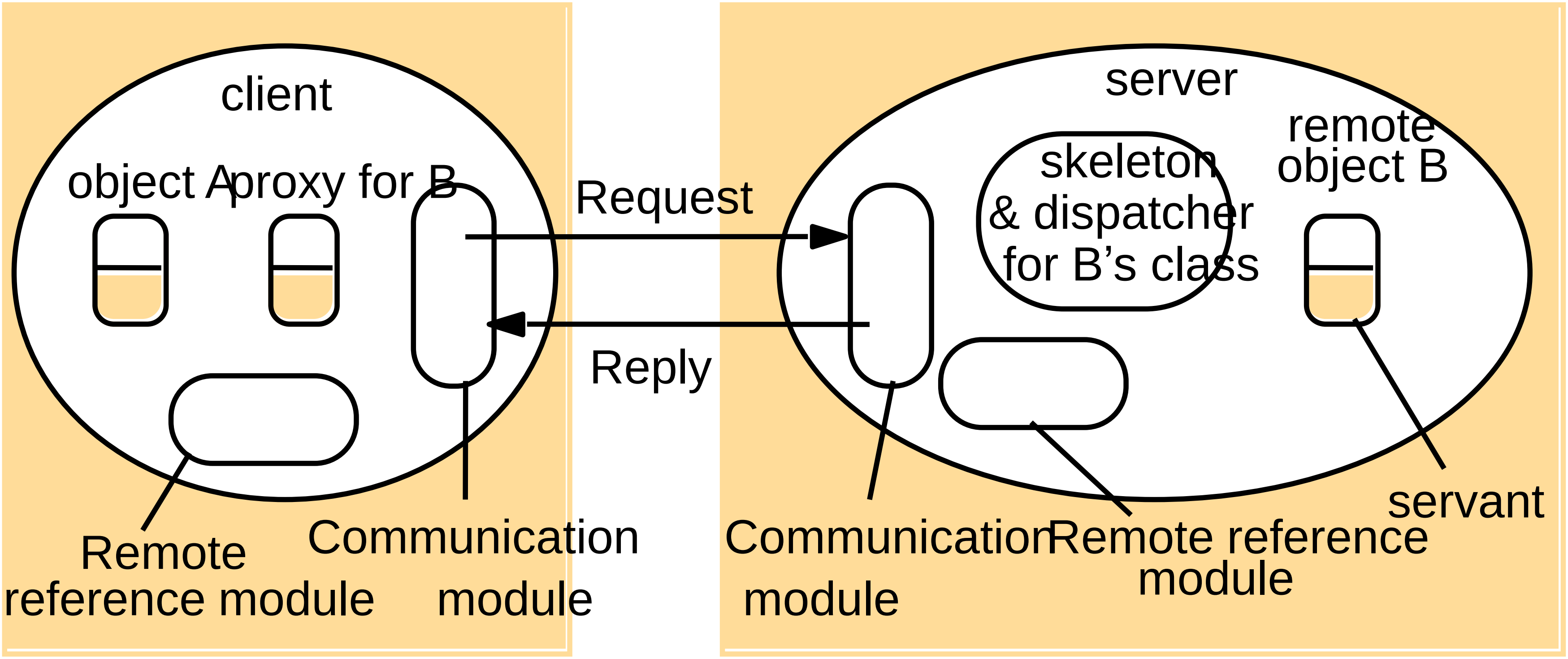


Figure 5.8 Role of client and server stub procedures in RPC in the context of a procedural language

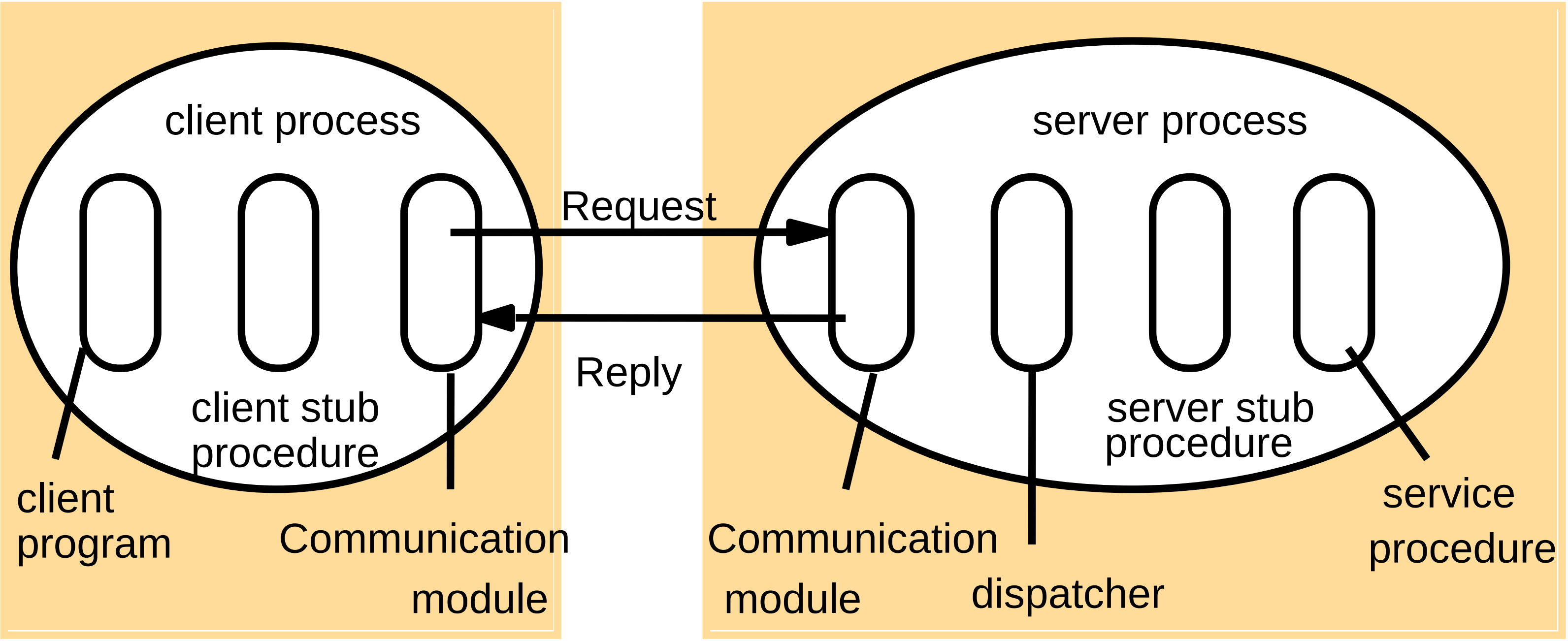


Figure 5.9
Files interface in Sun XDR

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
    int length;
    char buffer[MAX];
};
struct writeargs {
    FileIdentifier f;
    FilePointer position;
    Data data;
};
```

```
struct readargs {
    FileIdentifier f;
    FilePointer position;
    Length length;
};

program FILEREADWRITE {
  version VERSION {
    void WRITE(writeargs)=1;        1
    Data READ(readargs)=2;          2
  }=2;
} = 9999;
```

Figure 5.10
Dealing room system

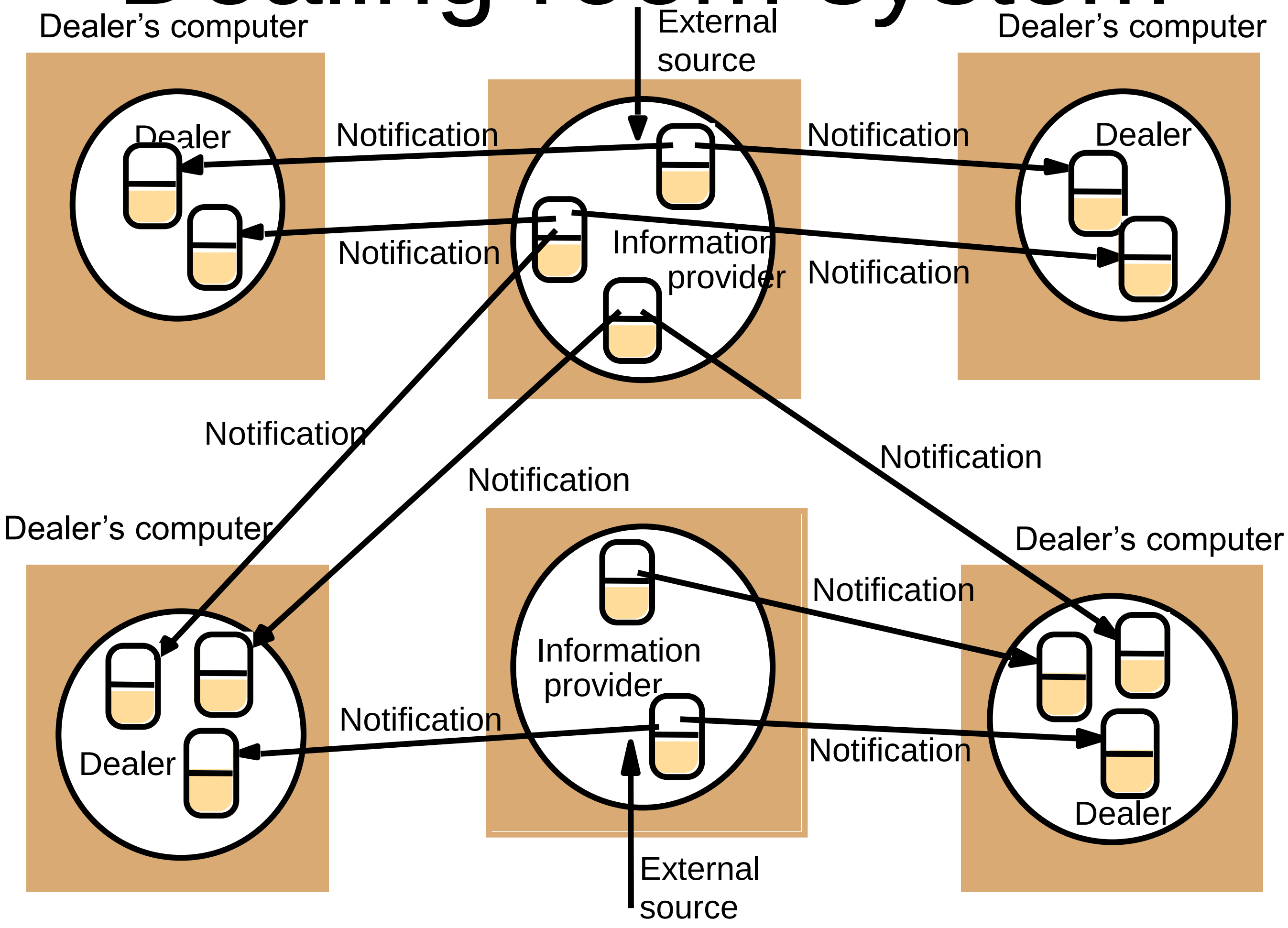


Figure 5.11
Architecture for distributed event notification

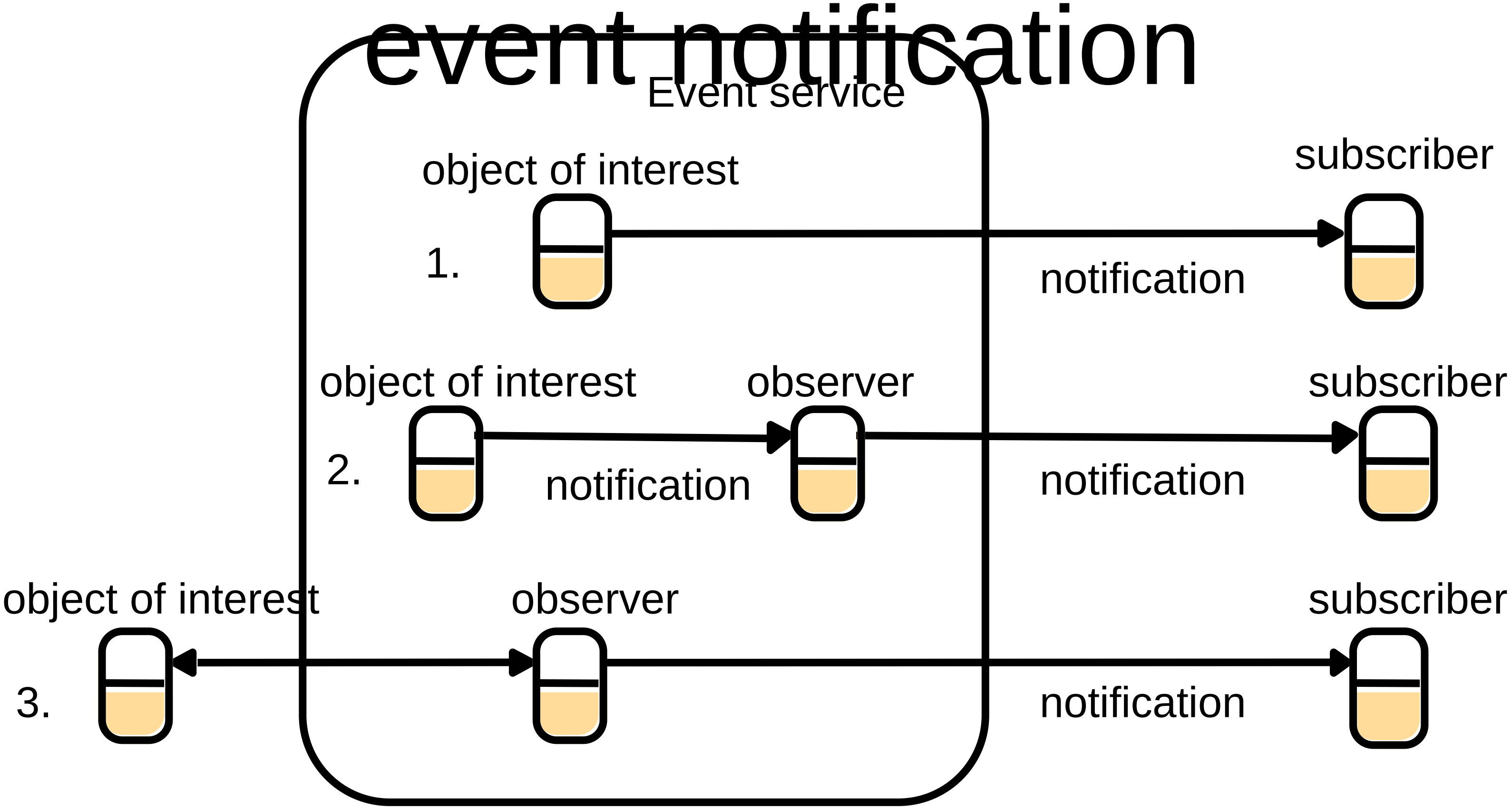


Figure 5.12
Java Remote interfaces *Shape* and *ShapeList*

```
import java.rmi.*;
import java.util.Vector;
public interface Shape extends Remote {
    int getVersion() throws RemoteException;
    GraphicalObject  getAllState() throws RemoteException;          1
}
public interface ShapeList extends Remote {
    Shape newShape(GraphicalObject g) throws RemoteException; 2
    Vector allShapes() throws RemoteException;
    int getVersion() throws RemoteException;
}
```

Figure 5.13 The *Naming* class of Java RMIregistry

void rebind (String name, Remote obj)

This method is used by a server to register the identifier of a remote object by name, as shown in Figure 15.13, line 3.

void bind (String name, Remote obj)

This method can alternatively be used by a server to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.

void unbind (String name, Remote obj)

This method removes a binding.

Remote lookup(String name)

This method is used by clients to look up a remote object by name, as shown in Figure 15.15 line 1. A remote object reference is returned.

String [] list()

This method returns an array of Strings containing the names bound in the registry.

Figure 5.14
Java class *ShapeListServer* with *main* method

```
import java.rmi.*;
public class ShapeListServer{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
         try{
            ShapeList aShapeList = new ShapeListServant();                    1
                Naming.rebind("Shape List", aShapeList );                     2
            System.out.println("ShapeList server ready");
            }catch(Exception e) {
            System.out.println("ShapeList server main " + e.getMessage());}
    }
}
```

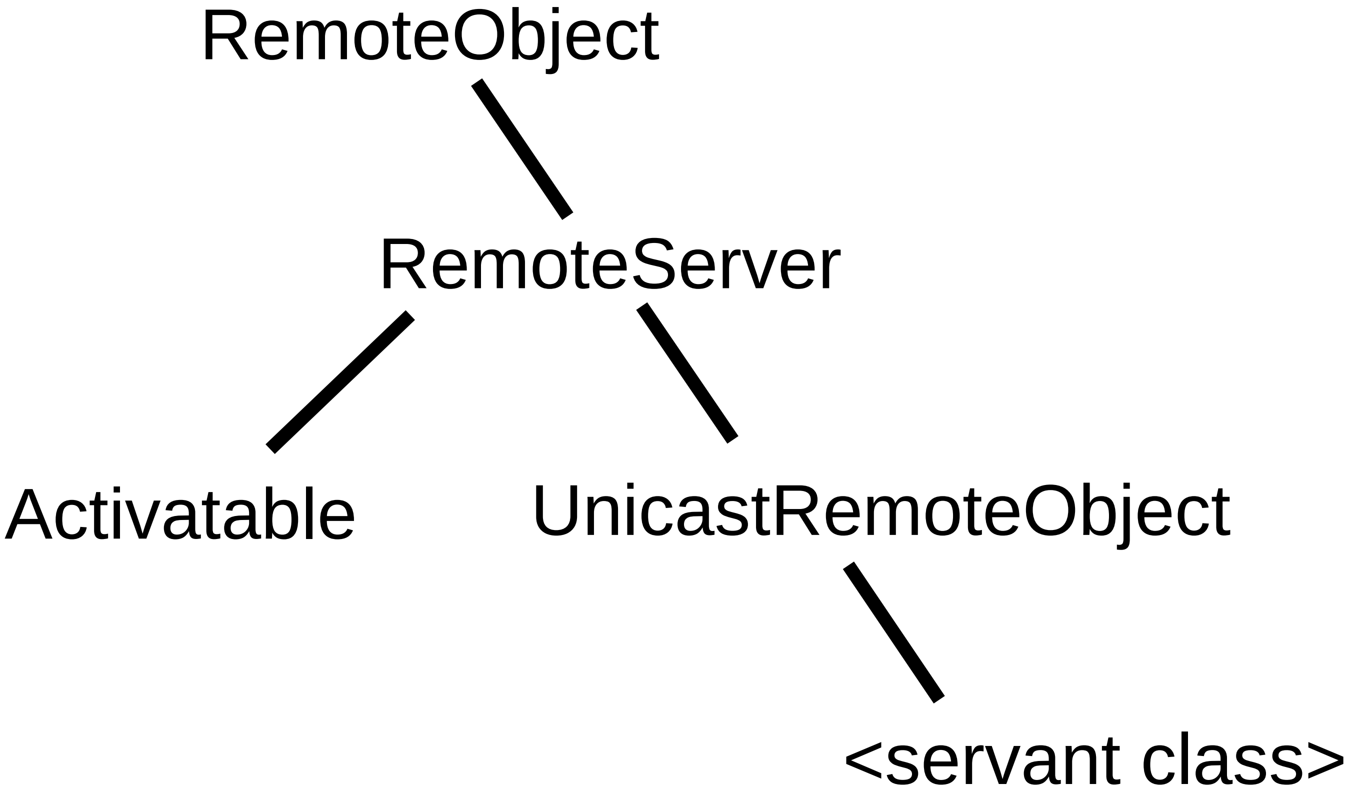
Figure 5.15
Java class *ShapeListServant* implements interface *ShapeList*

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.Vector;
public class ShapeListServant extends UnicastRemoteObject implements ShapeList
        private Vector theList;                  // contains the list of Shapes              1
        private int version;
       public ShapeListServant()throws RemoteException{...}
       public Shape newShape(GraphicalObject g) throws RemoteException {          2
             version++;
                   Shape s = new ShapeServant( g, version);                                   3
                   theList.addElement(s);
                   return s;
       }
       public  Vector allShapes()throws RemoteException{...}
       public int getVersion() throws RemoteException { ... }
}
```

Figure 5.16
Java client of *ShapeList*

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
public class ShapeListClient{
    public static void main(String args[]){
      System.setSecurityManager(new RMISecurityManager());
      ShapeList aShapeList = null;
      try{
            aShapeList  = (ShapeList) Naming.lookup("//bruno.ShapeList") ;
            Vector sList = aShapeList.allShapes();
      } catch(RemoteException e) {System.out.println(e.getMessage());
      }catch(Exception e) {System.out.println("Client: " + e.getMessage());}
    }
}
```

Figure 5.17
Classes supporting Java RMI



RemoteObject

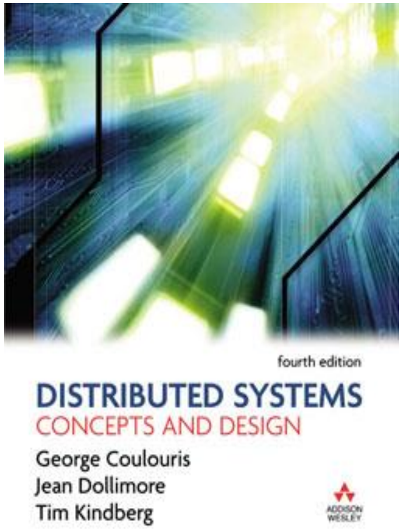
RemoteServer

Activatable

UnicastRemoteObject

<servant class>

Slides for Chapter 7:
Security

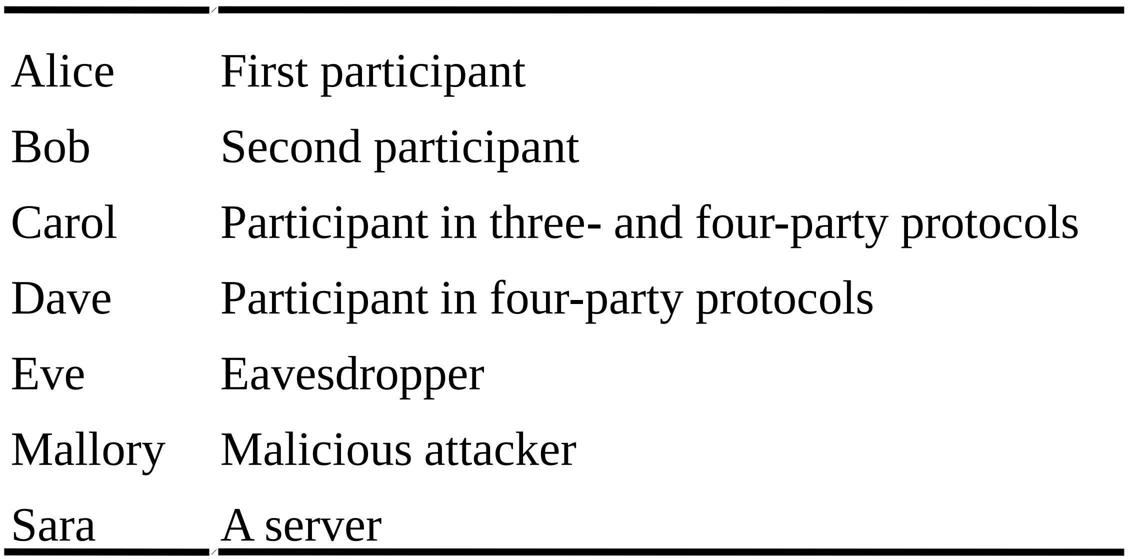


From Coulouris, Dollimore and Kindberg

Distributed Systems:
Concepts and Design

Edition 4, © Addison-Wesley 2005

Figure 7.1
Familiar names for the protagonists in security protocols



Alice	First participant
Bob	Second participant
Carol	Participant in three- and four-party protocols
Dave	Participant in four-party protocols
Eve	Eavesdropper
Mallory	Malicious attacker
Sara	A server

Figure 7.2
Cryptography notations

K_A	Alice’s secret key
K_B	Bob’s secret key
K_{AB}	Secret key shared between Alice and Bob
K_{Apriv}	Alice’s private key (known only to Alice)
K_{Apub}	Alice’s public key (published by Alice for all to read)
$\{M\}_K$	Message M Encrypted with key K
$[M]_{\mathrm{K}}$	Mesage M signed with key K

Symmetric key encryption

- Most common. Typical for passwords when you log into a system.
- Cheaper (less computer time), faster, very good encryption.
- The workhorse of cryptography
- Both sides share the same key.
- Problem is distributing the key

Symmetric key encryption

- Most common. Typical for passwords when you log into a system.
- Cheaper (less computer time), faster, very good encryption.
- The workhorse of cryptography
- Both sides share the same key.
- Problem is distributing the key

Public key encryption

- Non-symmetric keys come in pairs
  - One key used to encrypt.
  - The other is used to decrypt.
  - Either key can be used for either purpose
- RSA (Rivest Shamir Adleman) algorithm is the one commonly used
- NOTE: *Secret Key* and *Private Key* mean the same thing in this discussion.

Public / Private Asymmetric keys

- Asymmetric key pairs are easy to generate and there is free software to do so (e.g. – PGP “Pretty Good Privacy”)
- Both keys are functionally identical:
  - Pick one to be private – keep it secret
  - Pick one to be public – bind it to a stakeholder (person) publically (e.g. web)

Oracles

- Trusted ("talks to God") entities that post their trusted public key, and post the bindings of public keys to stakeholders (people, corporations).
- "Certification" sites where oracles create other trusted certification sites.
- First oracles -- Originally just declared that they were to be trusted, and everyone did -- Verisign

Public key (RSA) example

- S is "secret key," P is "public key," M is "message," C is cyphertext (encrypted version of the message).
- C = P(M)   the ciphertext can be had by applying the public key to the message
- M = S(C) = S(P(M)) the message can be had by applying the secret key to the ciphertext

To work, the system must satisfy (due to Diffie and Hellman, 1976):

(i) S(P(M)) = M for every M

(ii) All (S,P) pairs are distinct

(iii) Deriving S from P is as hard as reading the ciphertext

(iv) Both S and P are easy to compute

Certification sites

- Main job is to be trusted to *link public keys to stakeholders.*
- Everyone must trust that…
  - ***the public key for the certification site is real***
  - ***the secret key for the site has not leaked***.

Publish bindings of stakeholders to public keys

- I am Oracle-55 and I say:
  - “Alices’s public key is AF1077BE”
  - “Bob’s public keys is 3456AAED”
- The encrypted (“signed”) version: <////>
- Use my public key to decrypt it!

Using public-key encryption

- Public keys are published in a “phone book” of public keys, available to all at the trusted certification site
- The matching private key is kept private, and secret by the stakeholder
- If *Alice* wants to ***send a secret message*** to *Bob* she encrypts it using his ***public key***.
- The message cannot be read until after it is decrypted using the secret key that only *Bob* knows --- hence only *Bob* can read the message.

Signing

- If *Alice* wants a ***signed*** copy of a message from *Bob* she can request that he encrypt the message using his ***private key.***
- Anyone can now read the message (including a court of law) using *Bob*'s public key. Assuming a valid publication of his public key, this identifies *Bob* as the author.
- Singing depends on having a trusted source for posting of the bindings of pubic keys to stakeholders.

Third party registration

- *Bob*'s signature is only as good as the site where his public key is posted.
- Third party vendors (the certification/oracle sites) exist to guarantee the authenticity of public keys (to certify them), and to give out public and private key pairs.

Certification

- Once a certification authority (oracle) is established this can be used to certify other sets of public/private keys in various ways:

Certification

- Stakeholders:
  - Stakeholders can be bound to public keys. It is up to the stakeholder to keep the secret key from leaking.
  - When some stakeholder's public key has been certified, you can send them secret messages.

Certification

- First-level Oracles:
  - If an oracle gets busy, it can authorize other sites to authorize stakeholders.

Certification

- Full Oracles:
  - If an oracle gets busy, it can authorize other sites to authorize other oracles.

Certification

- Ad hoc:
- Authority *C* can *sign* (with their private key) a document containing the public keys of party *A* and party *B* and identifying them as belonging to the respective parties. This document can only be decrypted using *C*'s posted, trusted, public key, verifying it as authentic.
- In this way, both *A* and *B* are also known to have attributable public keys for some particular transaction.

Signing weakness

- Once an author *signs* a document, this authenticates them as the author, and the cannot refute it…
- …unless they claim they exposed their secret key.

Signing weakness

- So: for contracts that rely on signing, include the wording that if any party exposes their secret key – no problem, they are just required to pay ten million dollars to the other party (or whatever the maximum deal is worth ☺).

Certification

- Once an authority is established this can be used to certify other sets of public/private keys.
- This can happen in two ways.
- For example, authority C can *sign* (with their private key) a document containing the public keys of party A and party B and identifying them as belonging to the respective parties. This document can only be decrypted using C's posted, trusted, public key, verifying it as authentic.
- In this way, both A and B are also known to have attributable public keys.

Publish bindings of stakeholders to public keys

Stakeholder	Public Key	Oracle status
Alice	AF1077BE	Stakeholder
Bob	3456AAED	Stakeholder
Carol	5544BBCC	Bind Stakeholders
Dave	6611FFDD	Bind Oracles

- The encrypted (“signed”) version: <////>
- Use my public key to decrypt it!

Digital signatures

Requirement:

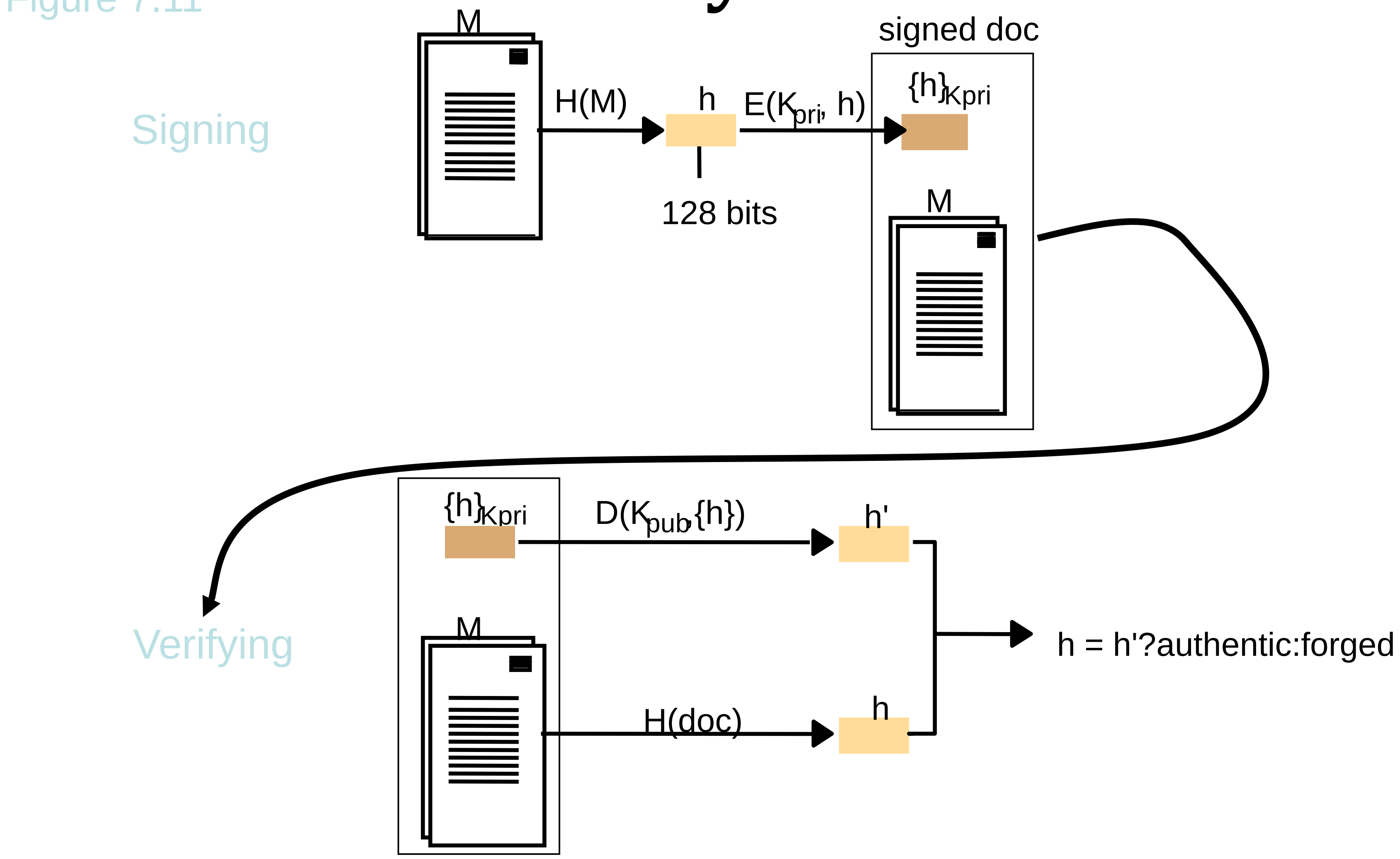
- To authenticate stored document files as well as messages
- To protect against forgery
- To prevent the signer from repudiating a signed document (denying their responsibility)

Encryption of a document in a secret key constitutes a signature

- impossible for others to perform without knowledge of the key
- strong authentication of document
- strong protection against forgery
- weak against repudiation (signer could claim key was compromised – cde: huh? Applies to all above. So, nope!)

Digital signatures with public keys

Figure 7.11



*

Cryptographic Hash Function

- Never necessary, just cheaper
- Reduce long document to short bit string
- Large universe of documents → small universe of possible hashes.
- Easy to compute hash from document
- Brute-force search to produce document from the hash value
- Sign only the hash value
- Verify document by applying public key to hashA
- Compute hashB of purported document
- Compare hashA to hashB

Example

- Hash function is: eight-bit ASCII value of first letter of message (Terrible!!)
- Message is “Hello”;
- ASCII “H” is 72 = 01001000
- H(“Hello”)= 72. H(“Apple”) = 65 H(“Hi”) = ?
- We already have the message, we just need to verify it.

Secure digest functions

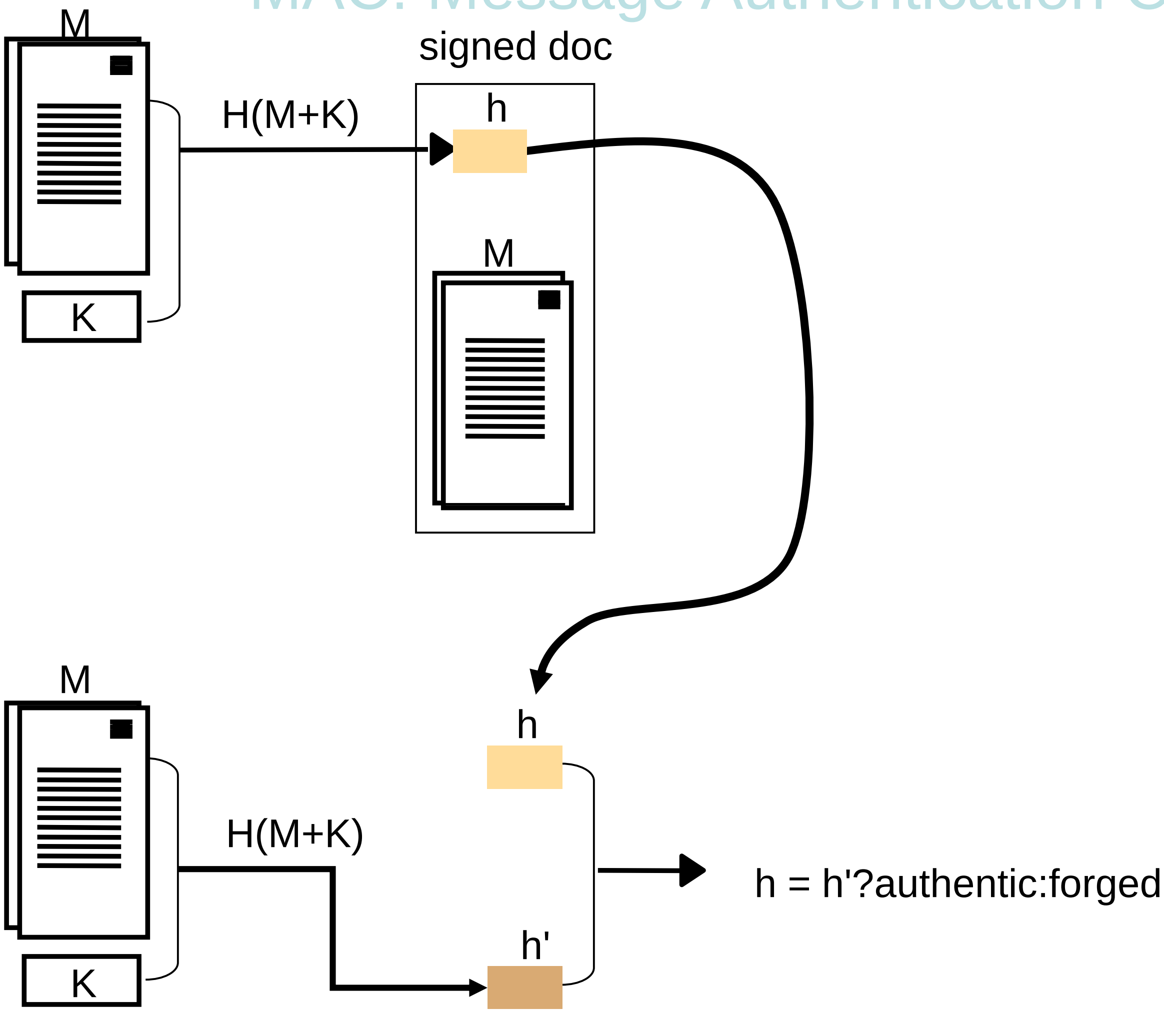
- Encrypted text of document makes an impractically long signature
  - so we encrypt a *secure digest* instead
  - A secure digest function computes a fixed-length hash H(M) that characterizes the document M
  - H(M) should be:
    - fast to compute
    - hard to invert - hard to compute M given H(M)
    - hard to defeat in any variant of the Birthday Attack
- **MD5**: Developed by Rivest (1992). *Computes a 128-bit digest. Speed 1740 kbytes/sec.*
- **SHA**: (1995) based on Rivest's MD4 but made more secure by producing a *160-bit digest, speed 750 kbytes/second*
- **Any symmetric encryption algorithm** can be used in CBC (cipher block chaining) mode. The last block in the chain is H(M)

MACs: Low-cost signatures with a shared secret key

Figure 7.12

MAC: Message Authentication Code

Signing



Verifying

Signer and verifier share a secret key K

Scenario 4:
Digital signatures with a secure digest function

Alice wants to publish a document M in such a way that anyone can verify that it is from her.

1. Alice computes a fixed-length digest of the document Digest(M).
2. Alice encrypts the digest in her private key, appends it to M and makes the resulting signed document (M, $\{\text{Digest(M)}\}_{K_{Apriv}}$) available to the intended users.
3. Bob obtains the signed document, extracts M and computes Digest(M).
4. Bob uses Alice's public key to decrypt $\{\text{Digest(M)}\}_{K_{Apriv}}$ and compares it with his computed digest. If they match, Alice's signature is verified.

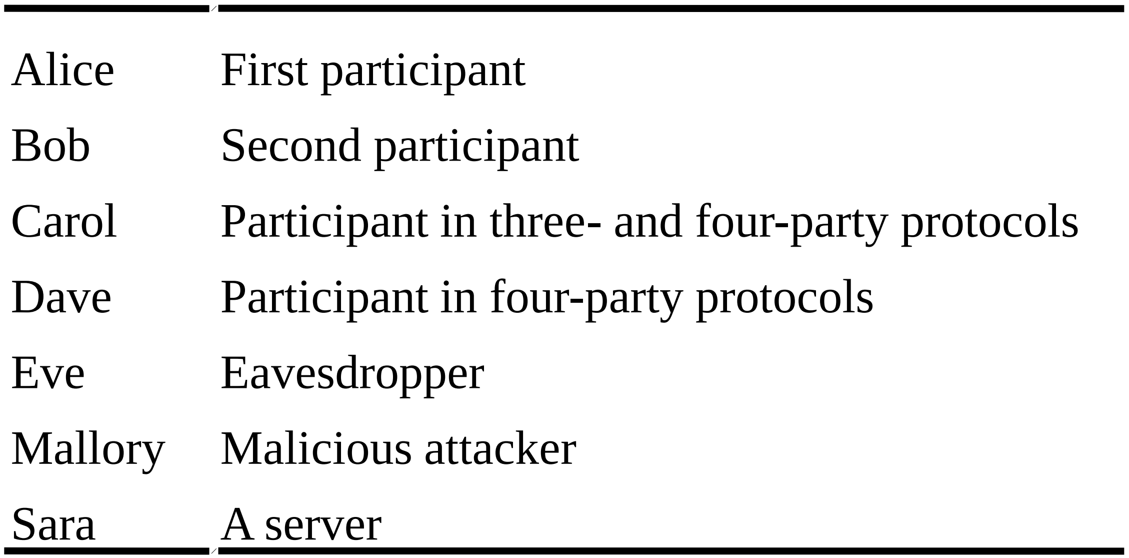
The digest function must be secure against the *birthday attack*

Worst case assumptions and design guidelines

[p. 260]

- Interfaces are exposed
- Networks are insecure
- Algorithms and program code are available to attackers
- Attackers may have access to fast, powerful, resources
- Attackers are current, smart, detailed, highly expert in one area.
- Minimize the trusted base
- Limit the lifetime and scope of each secretxr

Figure 7.1
Familiar names for the protagonists in security protocols



Alice	First participant
Bob	Second participant
Carol	Participant in three- and four-party protocols
Dave	Participant in four-party protocols
Eve	Eavesdropper
Mallory	Malicious attacker
Sara	A server

Figure 7.2 Cryptography notations

K_A	Alice’s secret key
K_B	Bob’s secret key
K_{AB}	Secret key shared between Alice and Bob
K_{Apriv}	Alice’s private key (known only to Alice)
K_{Apub}	Alice’s public key (published by Alice for all to read)
$\{M\}_K$	Message M Encrypted with key K
$[M]_{\mathrm{K}}$	Mesage M signed with key K

Threats and forms of attack

- Eavesdropping
  - obtaining private or secret information
- Masquerading
  - assuming the identity of another user/principal
- Message tampering
  - altering the content of messages in transit
    - man in the middle attack (tampers with the secure channel mechanism)
- Replaying
  - storing secure messages and sending them at a later date
- Denial of service
  - flooding a channel or other resource, denying access to others

Scenario 1: Secret communication with a shared secret key

Alice and Bob share a secret key K_{AB}.

1. Alice uses K_{AB} and an agreed encryption function $E(K_{AB}, M)$ to encrypt and send any number of messages $\{M_i\}_{K_{AB}}$ to Bob.
2. Bob reads the encrypted messages using the corresponding decryption function $D(K_{AB}, M)$.

Alice and Bob can go on using K_{AB} as long as it is safe to assume that K_{AB} has not been *compromised*.

Issues:

Key distribution: How can Alice send a shared key K_{AB} to Bob securely?

Freshness of communication: How does Bob know that any $\{M_i\}$ isn't a copy of an earlier encrypted message from Alice that was captured by Mallory and replayed later?

Scenario 3:
Authenticated communication with public keys

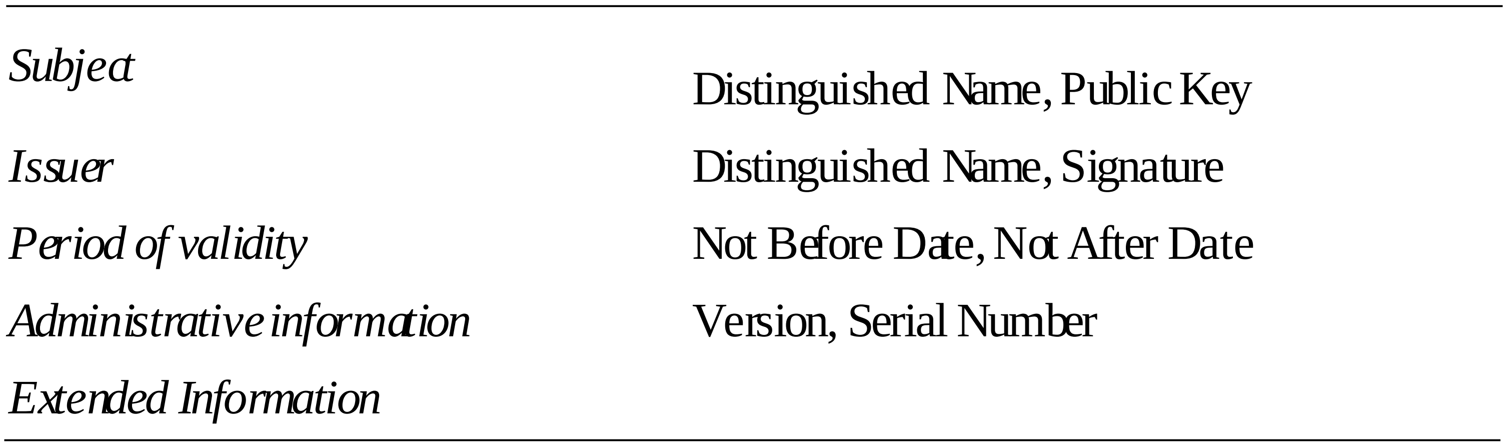
Bob has a public/private key pair <K_{Bpub}, K_{Bpriv}>

1. Alice obtains a certificate that was signed by a trusted authority stating Bob's public key K_{Bpub}
2. Alice creates a new shared key K_{AB} , encrypts it using K_{Bpub} using a public-key algorithm and sends the result to Bob.
3. Bob uses the corresponding private key K_{Bpriv} to decrypt it.

(If they want to be sure that the message hasn't been tampered with, Alice can add an agreed value to it and Bob can check it.)

X509 Certificate format

Figure 7.13



Subject	Distinguished Name, Public Key
Issuer	Distinguished Name, Signature
Period of validity	Not Before Date, Not After Date
Administrative information	Version, Serial Number
Extended Information	

Symmetric encryption algorithms

These are all programs that perform confusion and diffusion operations on blocks of binary data

TEA: a simple but effective algorithm developed at Cambridge U (1994) for teaching and explanation. *128-bit key, 700 kbytes/sec*

DES: The US Data Encryption Standard (1977). No longer strong in its original form. *56-bit key, 350 kbytes/sec*.

Triple-DES: applies DES three times with two different keys. *112-bit key, 120 Kbytes/sec*

IDEA: International Data Encryption Algorithm (1990). Resembles TEA. *128-bit key, 700 kbytes/sec*

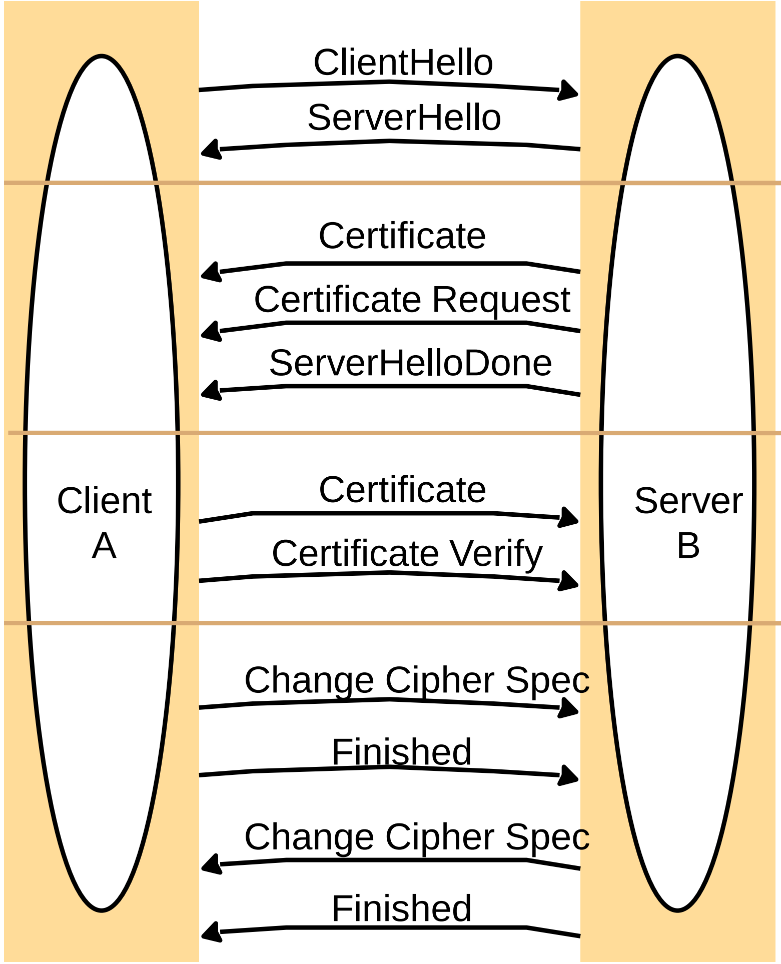
AES: A proposed US Advanced Encryption Standard (1997). *128/256-bit key*.

There are many other effective algorithms. See Schneier [1996].

The above speeds are for a Pentium II processor at 330 MHZ. Today's PC's (January 2002) should achieve a 5 x speedup.

SSL handshake protocol

Figure 7.18



Establish protocol version, session ID, cipher suite, compression method, exchange random start values

Optionally send server certificate and request client certificate

Send client certificate response if requested

Change cipher suite and finish handshake

Includes key master exchange. Key master is used by both A and B to generate:

2 session keys   *2 MAC keys* □

K_{AB} M_{AB}

K_{BA} M_{BA}

*

Scenario 3:
Authenticated communication with public keys

Bob has a public/private key pair <K_{Bpub}, K_{Bpriv}>

1. Alice obtains a certificate that was signed by a trusted authority stating Bob's public key K_{Bpub}
2. Alice creates a new shared key K_{AB} , encrypts it using K_{Bpub} using a public-key algorithm and sends the result to Bob.
3. Bob uses the corresponding private key K_{Bpriv} to decrypt it.

(If they want to be sure that the message hasn't been tampered with, Alice can add an agreed value to it and Bob can check it.)

Scenario 3:
Authenticated communication with public keys

- Mallory might intercept Alice’s initial request to a key distribution service for Bob’s public-key certificate and send a response containing his own public key. He can then intercept all the subsequent messages.

Digital signatures

Requirement:

- To authenticate stored document files as well as messages
- To protect against forgery
- To prevent the signer from repudiating a signed document (denying their responsibility)

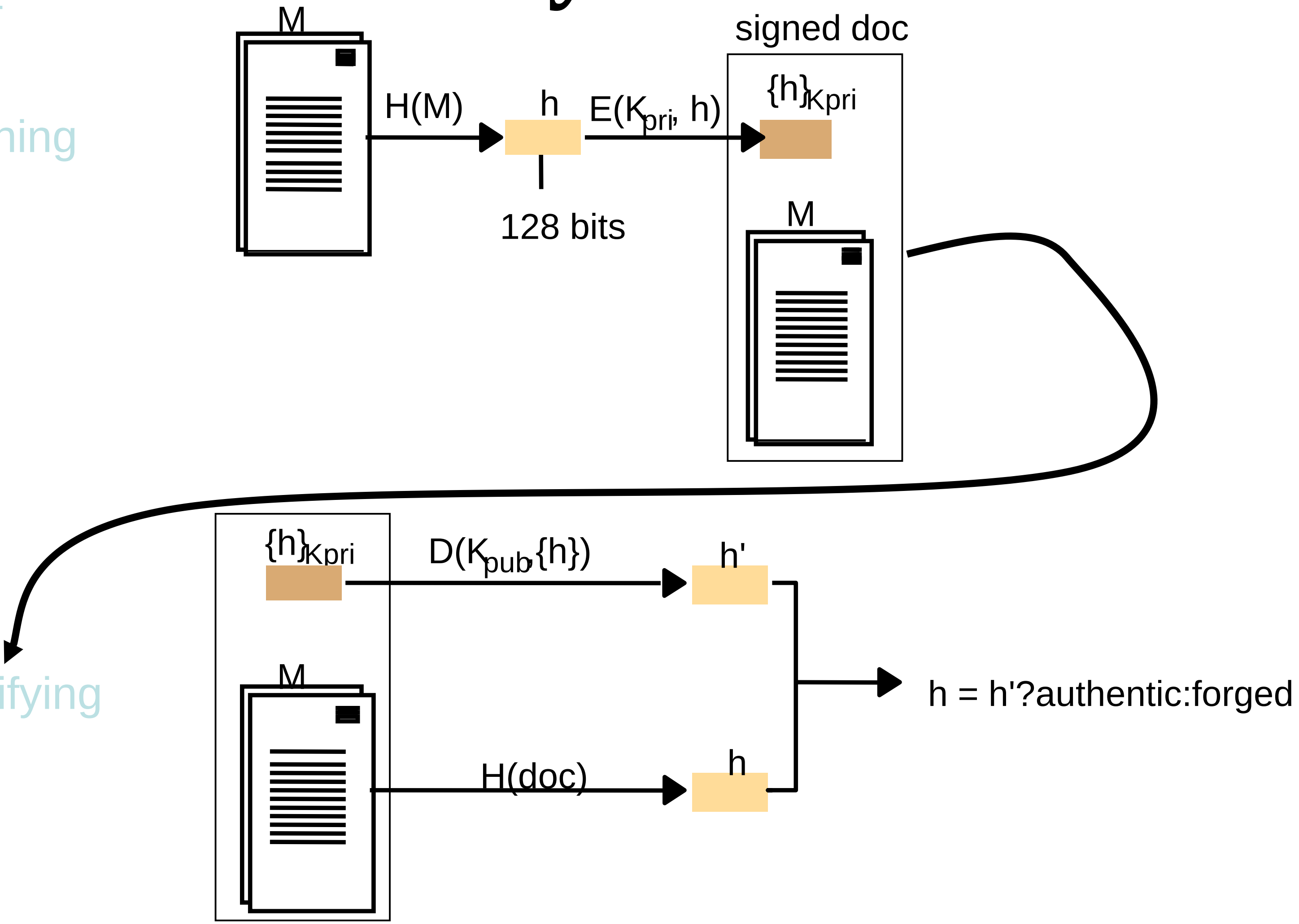
Encryption of a document in a secret key constitutes a signature

- impossible for others to perform without knowledge of the key
- strong authentication of document
- strong protection against forgery
- weak against repudiation (signer could claim key was compromised – cde: huh? Applies to all above. So, nope!)

Digital signatures with public keys

Figure 7.11

Signing



M

H(M)

h

$E(K_{pri}, h)$

128 bits

signed doc

$\{h\}_{Kpri}$

M

$\{h\}_{Kpri}$

$D(K_{pub},\{h\})$

h'

M

H(doc)

h

h = h'?authentic:forged

Verifying

Cryptographic Hash Function

- Never necessary, just cheaper
- Reduce long document to short bit string
- Large universe of documents → small universe of possible hashes.
- Easy to compute hash from document

Cryptographic Hash Function

- Easy to compute hash from document
- Brute-force search to produce document from the hash value
- Sign only the hash value
- Verify document by applying public key to hashA
- Compute hashB of purported document
- Compare hashA to hashB

Example

- Hash function is: eight-bit ASCII value of first letter of message (Terrible!!)

- Message is “Hello”;

- ASCII “H” is 72 = 01001000

- H(“Hello”)= 72. H(“Apple”) = 65 H(“Hi”) = ?

- We already have the message, we just need to verify it.

Secure digest functions

- Encrypted text of document makes an impractically long signature
  - so we encrypt a *secure digest* instead
  - A secure digest function computes a fixed-length hash H(M) that characterizes the document M
  - H(M) should be:
    - fast to compute
    - hard to invert - hard to compute M given H(M)
    - hard to defeat in any variant of the Birthday Attack
- **MD5**: Developed by Rivest (1992). *Computes a 128-bit digest. Speed 1740 kbytes/sec.*
- **SHA**: (1995) based on Rivest's MD4 but made more secure by producing a *160-bit digest, speed 750 kbytes/second*
- **Any symmetric encryption algorithm** can be used in CBC (cipher block chaining) mode. The last block in the chain is H(M)

*

Evil practices of AT&T, Discover, others…

- Discover card, AT&T store plain text PW and they leak.
- Evil hacker never uses the exposed PW to attack Discover or AT&T
- Instead they use *similar* or *the same* PW to attack other accounts belonging to the identified user.
- Discover Card and AT&T are off the hook, but the user’s identity is stolen.
- Evil!

Evil practices of AT&T, Discover, others…

- NEVER EVER save the plain text password in a database on the server. We never need it.

- User registers PW over SSL → Encrypt *some good (long, random)* string plus *salt* using PW → save result in DB

- User logs in with PW over SSL → Encrypt *good string* plus *salt* using PW

- Compared to stored string. Same? Allow entry.

- PLAIN TEXT PW IS NEVER STORED!

Dictionary attack

- Build a dictionary of words and word-ettes, put in column two of array
- http://depaul.edu/~Elliott/301/MySpellchecker.java
- Encrypt each word, and put output in column one
- Sort on column one. Binary search!

- Steal the PWs from corporate DB. (Known string encrypted with PW).

- Binary look up in your hacker table.

- Voila! Log n time to locate original PW.

- *Salt* helps to defeat dictionary attack

- BfaXy string, 9468 salt. B9f4X6y8 to be encrypted with PW

Code table—*perfect* secrecy

You say…	It means…
Orange	Order pizza for the troops
DogCollar	Don’t forget to lock the door
BlueBird	Send the missiles at dawn

But can only send once.

One-time pad—*perfect* encryption

- Bitstring as long as the message itself—the pad
- XOR each bit with the message to produce encrypted text.
- XOR with same bitstring again to reproduce the original text.
- *Only* relationship between Message and Encryption text is the pad itself. Can never be cracked.
- Cheap, but can only be used *one-time* per pad.

1 0 1 1 0 0 ← original message

1 1 1 0 1 0 ← pad

0 1 0 1 1 0 ← XOR with pad=encryption string

Send over network

0 1 0 1 1 0 ← encryption string

1 1 1 0 1 0 ← pad

1 0 1 1 0 0 ← original message from XOR again

Practical and cheap

- Carry a copied Blu-ray disk filled with data to New York
- Use the bits on it in order.
- Now have a ~two-trillion bit pad in both locations.
- Why can this encryption *never* be broken?

Practical and cheap

- “Perfectness” depends on the quality of the random bit-string generator.

- Because there is no perfect generator, we don’t actually have perfect encryption.

- But it is “pretty” perfect if you use a good quality hardware random number device.

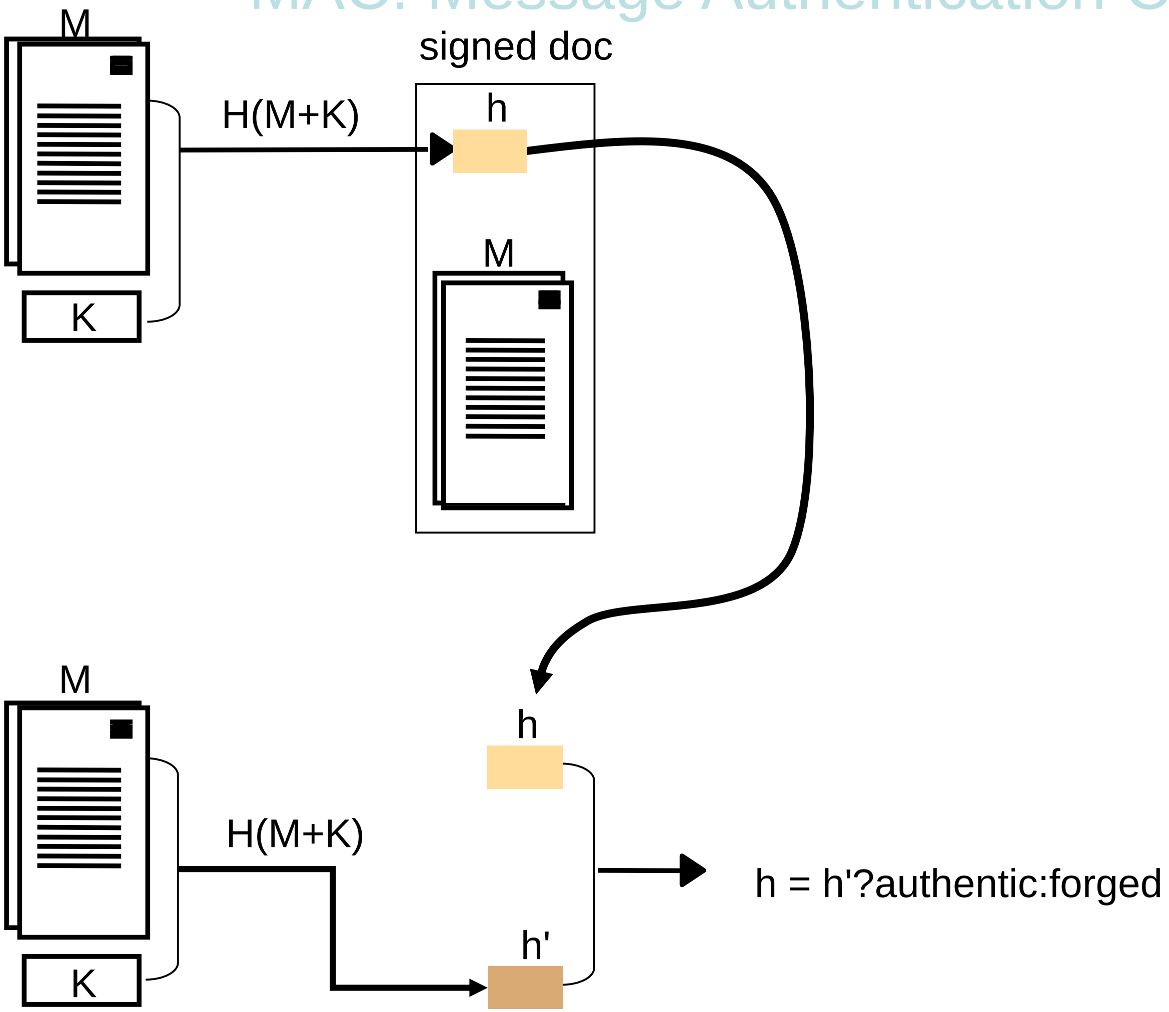
- https://en.wikipedia.org/wiki/Hardware_random_number_generator

MACs: Low-cost signatures with a shared secret key

Figure 7.12

MAC: Message Authentication Code

Signing



Signer and verifier share a secret key K

Verifying

Scenario 4:
Digital signatures with a secure digest function

Alice wants to publish a document M in such a way that anyone can verify that it is from her.

1. Alice computes a fixed-length digest of the document Digest(M).
2. Alice encrypts the digest in her private key, appends it to M and makes the resulting signed document (M, $\{\text{Digest(M)}\}_{K_{Apriv}}$) available to the intended users.
3. Bob obtains the signed document, extracts M and computes Digest(M).
4. Bob uses Alice's public key to decrypt $\{\text{Digest(M)}\}_{K_{Apriv}}$ and compares it with his computed digest. If they match, Alice's signature is verified.

The digest function must be secure against the *birthday attack*

Not Used

Not Used

Not Used

- **Birthday paradox**
- *Statistical result*: if there are 23 people in a room, the chances are even that 2 of them will have the same birthday.
- If our hash values are 64 bits long, we require only 2^{32} versions of M and M' on average.
- This is too small for comfort. We need to make our hash values at least 128 bits long to guard against this attack.

Birthday attack

1. Alice prepares two versions M and M' of a contract for Bob. M is favourable to Bob and M' is not.
2. Alice makes several subtly different versions of both M and M' that are visually indistinguishable from each other by methods such as adding spaces at the ends of lines. She compares the hashes of all the versions of M with all the versions of M'. (She is likely to find a match because of the Birthday Paradox).
3. When she has a pair of documents M and M' that hash to the same value, she gives the favourable document M to Bob for him to sign with a digital signature using his private key. When he returns it, she substitutes the matching unfavourable version M', retaining the signature from M.

Certificates

Certificate: a statement signed by an appropriate authority.

Certificates require:

- An agreed standard format
- Agreement on the construction of chains of trust (see Section 7.4.4).
- Expiry dates, so that certificates can be revoked.

Figure 7.5 Public-key certificate for Bob's Bank

1. *Certificate type*:	Public key
2. *Name*:	Bob's Bank
3. *Public key*:	K_{Bpub}
4. *Certifying authority*:	Fred – The Bankers Federation
5. *Signature*:	$\{Digest(field\ 2 + field\ 3)\}_{K_{Fpriv}}$

X509 Certificate format

Figure 7.13

Subject	Distinguished Name, Public Key
Issuer	Distinguished Name, Signature
Period of validity	Not Before Date, Not After Date
Administrative information	Version, Serial Number
Extended Information	

Certificates as credentials

- Certificates can act as *credentials*
  - Evidence for a principal's right to access a resource
- Th le co te on
  - S cate

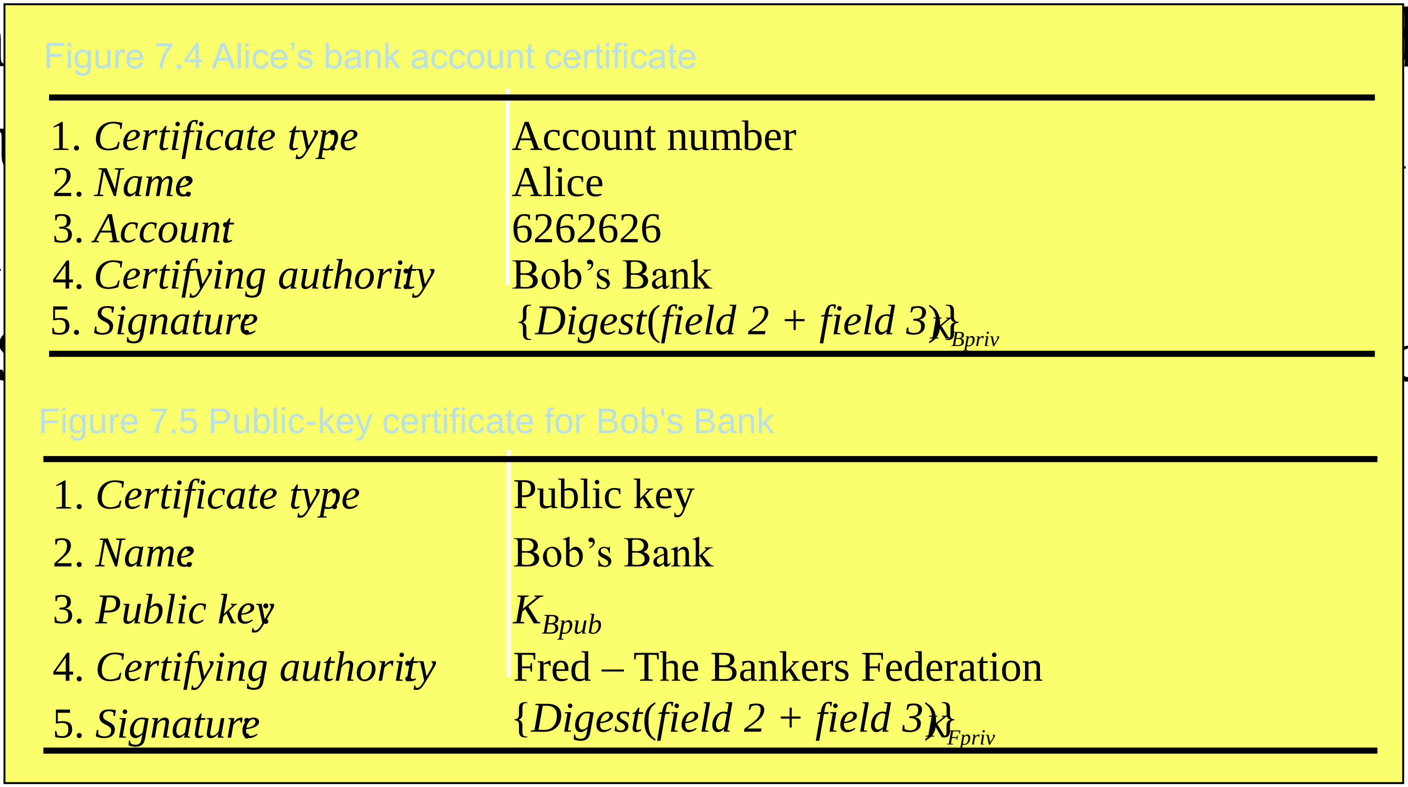


Figure 7.4 Alice’s bank account certificate

1. *Certificate type*	Account number
2. *Name*	Alice
3. *Account*	6262626
4. *Certifying authority*	Bob’s Bank
5. *Signature*	$\{Digest(field\ 2 + field\ 3)\}_{K_{Bpriv}}$

Figure 7.5 Public-key certificate for Bob's Bank

1. *Certificate type*	Public key
2. *Name*	Bob’s Bank
3. *Public key*	K_{Bpub}
4. *Certifying authority*	Fred – The Bankers Federation
5. *Signature*	$\{Digest(field\ 2 + field\ 3)\}_{K_{Fpriv}}$

Access control

- Protection domain
  - A set of <resource, rights> pairs
- Two main approaches to implementation:
  - Access control list (ACL) associated with each object
    - E.g. Unix file access permissions ➲
    - For more complex object types and user communities, ACLs can become very complex
  - Capabilities associated with principals
    - Like a key
    - Format: <resource id, permitted operations,

X.509 weaknesses

- Designed to support the X.500 structure, not the web. Features not needed.
- “Over-functional and underspecified and the normative information is spread across many documents from different standardization bodies.” (Wikipedia)

No delegation control

- Anyone can set up shop as a CA.
- Paid for by the wrong party – the provider, not the consumer, so cheap preferred over quality.
- Warranties denied by Cas
- Wild west, classifying CA s and CA policy is impossible.
- No model for delegation *within* businesses

Certificate Revocation Lists

- Have to check back up the authentication chain
- Blacklisting instead of whitelisting
- Doesn’t really address root CA revocation
- Certificate chain semantics don’t address bilateral trusted relationships.
- Open research area for efficiency
- http://tools.ietf.org/html/rfc5280
- http://users.crhc.illinois.edu/yihchun/pubs/jsac10.pdf

Wikipedia example OSCP

- Online Certificate Status Protocol
- Alice and Bob have public key certificates issued by Ivan, the Certificate Authority (CA).
- Alice wishes to perform a transaction with Bob and sends him her public key certificate.
- Bob, concerned that Alice's private key may have been compromised, creates an 'OCSP request' that contains Alice's certificate serial number and sends it to Ivan.

- Ivan's OCSP responder reads the certificate serial number from Bob's request. The OCSP responder uses the certificate serial number to look up the revocation status of Alice's certificate. The OCSP responder looks in a CA database that Ivan maintains. In this scenario, Ivan's CA database is the only trusted location where a compromise to Alice's certificate would be recorded.

- Ivan's OCSP responder confirms that Alice's certificate is still OK, and returns a signed, successful 'OCSP response' to Bob.
- Bob cryptographically verifies Ivan's signed response. Bob has stored Ivan's public key sometime before this transaction. Bob uses Ivan's public key to verify Ivan's response.
- Bob completes the transaction with Alice.

X.509 weaknesses…

- "Users use an undefined certification request protocol to obtain a certificate which is published in an unclear location in a nonexistent directory with no real means to revoke it."
- Gutmann, Peter. "Everything you Never Wanted to Know about PKI but were Forced to Find Out"

Credentials

- Requests to access resources must be accompanied by *credentials*:
  - Evidence for the requesting principal's right to access the resource
  - Simplest case: an identity certificate for the principal, signed by the principal.
  - Credentials can be used in combination. E.g. to send an authenticated email as a member of Cambridge University, I would need to present a certificate of membership of CU and a certificate of my email address.
- The *speaks for* idea
  - We don't want users to have to give their password every time their PC accesses a server holding protected resources.
  - Instead, the notion that a credential *speaks for* a principal is introduced. E.g. a user's PK certificate speaks for that user.

Delegation

- Consider a server that prints files:
  - wasteful to copy the files, should access users' files *in situ*
  - server must be given restricted and temporary rights to access protected files
- Can use a delegation certificate or a capability
  - a **delegation certificate** is a signed request authorizing another principal to access a named resource in a restricted manner.
  - CORBA Security Service supports delegation certificates.
  - a **capability** is a key allowing the holder to access one or more of the operations supported by a resource.
  - The temporal restriction can be achieved by adding expiry times.

Symmetric encryption algorithms

These are all programs that perform confusion and diffusion operations on blocks of binary data

TEA: a simple but effective algorithm developed at Cambridge U (1994) for teaching and explanation. *128-bit key, 700 kbytes/sec*

DES: The US Data Encryption Standard (1977). No longer strong in its original form. *56-bit key, 350 kbytes/sec*.

Triple-DES: applies DES three times with two different keys. *112-bit key, 120 Kbytes/sec*

IDEA: International Data Encryption Algorithm (1990). Resembles TEA. *128-bit key, 700 kbytes/sec*

AES: A proposed US Advanced Encryption Standard (1997). *128/256-bit key*.

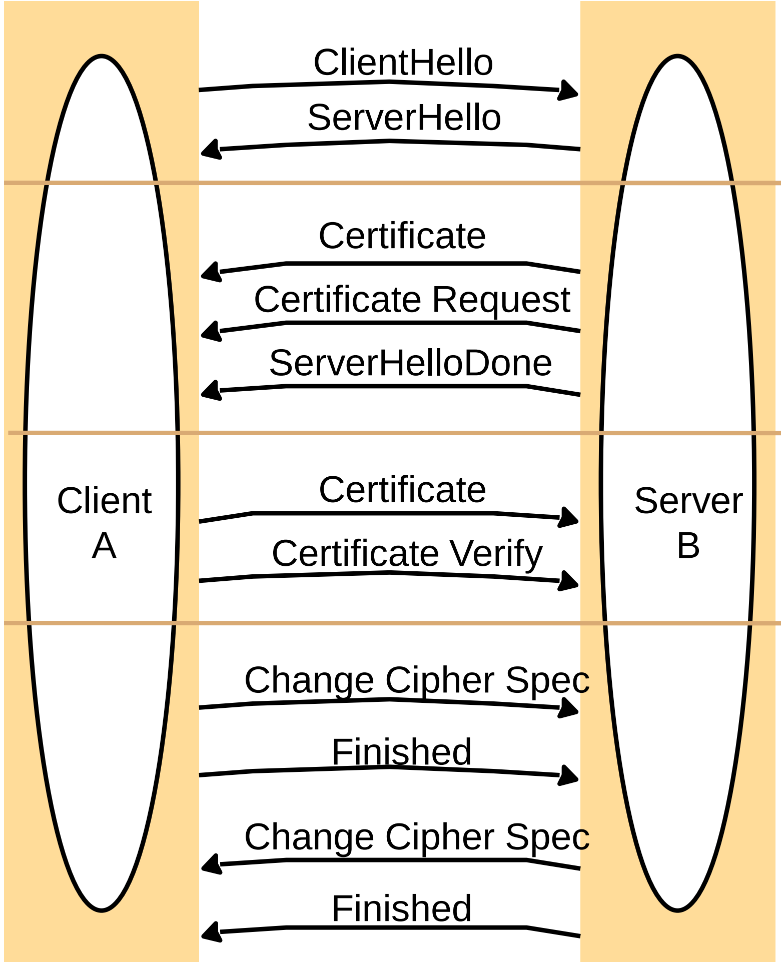
There are many other effective algorithms. See Schneier [1996].

The above speeds are for a Pentium II processor at 330 MHZ. Today's PC's (January 2002) should achieve a 5 x speedup.

*

SSL handshake protocol

Figure 7.18



Establish protocol version, session ID, cipher suite, compression method, exchange random start values

Optionally send server certificate and request client certificate

Send client certificate response if requested

Change cipher suite and finish handshake

Includes key master exchange. Key master is used by both A and B to generate:

2 session keys *2 MAC keys* □

K_{AB} M_{AB}

K_{BA} M_{BA}

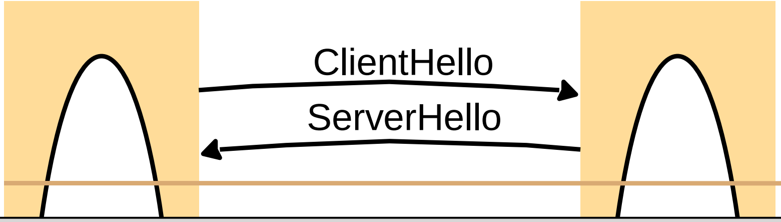
*

Note to Elliott…

- I’ve appended slides to the end that need to be culled, and inserted.

SSL handshake protocol

Figure 7.18



Establish protocol version, session ID, cipher suite, compression method, exchange random start values

ipher suite

Component	*Description*	*Example*
Key exchange method	the method to be used for exchange of a session key	RSA with public-key certificates
Cipher for data transfer	the block or stream cipher to be used for data	IDEA
Message digest function	for creating message authentication codes (MACs)	SHA

to generate:

2 session keys *2 MAC keys* □

K_{AB} M_{AB}

K_{BA} M_{BA}

*

Scenario 3: Authenticated communication with public keys

Bob has a public/private key pair <K_{Bpub}, K_{Bpriv}>

1. Alice obtains a certificate that was signed by a trusted authority stating Bob's public key K_{Bpub}
2. Alice creates a new shared key K_{AB} , encrypts it using K_{Bpub} using a public-key algorithm and sends the result to Bob.
3. Bob uses the corresponding private key K_{Bpriv} to decrypt it.

(If they want to be sure that the message hasn't been tampered with, Alice can add an agreed value to it and Bob can check it.)

Mallory -- attacker

- Intercept Alice’s initial request to a key distribution service for Bob’s public-key certificate and send a response containing his own public key. He can then intercept all the subsequent messages

SSL handshake configuration options

Figure 7.19

Component	*Description*	*Example*
Key exchange method	the method to be used for exchange of a session key	RSA with public-key certificates
Cipher for data transfer	the block or stream cipher to be used for data	IDEA
Message digest function	for creating message authentication codes (MACs)	SHA

Summary

- It is essential to protect the resources, communication channels and interfaces of distributed systems and applications against attacks.
- This is achieved by the use of access control mechanisms and secure channels.
- Public-key and secret-key cryptography provide the basis for authentication and for secure communication.
- Kerberos and SSL are widely-used system components that support secure and authenticated communication.

Threats and forms of attack

- Eavesdropping
  - obtaining private or secret information
- Masquerading
  - assuming the identity of another user/principal
- Message tampering
  - altering the content of messages in transit
    - man in the middle attack (tampers with the secure channel mechanism)
- Replaying
  - storing secure messages and sending them at a later date
- Denial of service
  - flooding a channel or other resource, denying access to others

Threats not defeated by secure channels or other cryptographic techniques

- Denial of service attacks
  - Deliberately excessive use of resources to the extent that they are not available to legitimate users
    - E.g. the Internet 'IP spoofing' attack, February 2000

Crypto-proof threats continued…

- Trojan horses and other viruses
  - Viruses can only enter computers when program code is imported.
  - But users often require new programs, for example:
    - New software installation
    - Mobile code downloaded dynamically by existing software (e.g. Java applets)
    - Accidental execution of programs transmitted surreptitiously
  - Defences: code authentication (signed code), code validation (type checking, proof), sandboxing.

Scenario 1: Secret communication with a shared secret key

Alice and Bob share a secret key K_{AB}.

1. Alice uses K_{AB} and an agreed encryption function $E(K_{AB}, M)$ to encrypt and send any number of messages $\{M_i\}_{K_{AB}}$ to Bob.
2. Bob reads the encrypted messages using the corresponding decryption function $D(K_{AB}, M)$.

Alice and Bob can go on using K_{AB} as long as it is safe to assume that K_{AB} has not been *compromised*.

Issues:

Key distribution: How can Alice send a shared key K_{AB} to Bob securely?

Freshness of communication: How does Bob know that any $\{M_i\}$ isn't a copy of an earlier encrypted message from Alice that was captured by Mallory and replayed later?

Authenticated communication with a server

Bob is a file server; Sara is an authentication service. Sara shares secret key K_A with Alice and secret key K_B with Bob.

1. Alice sends an (unencrypted) message to Sara stating her identity and requesting a *ticket* for access to Bob. ➲
2. Sara sends a response to Alice. $\{\{\text{Ticket}\}_{KB}, K_{AB}\}_{KA}$. It is encrypted

A ticket is an encrypted item containing the identity of the principal to whom it is issued and a shared key for a communication session.

4. Alice sends Bob a request R to access a file: $\{\text{Ticket}\}_{KB}$, Alice, R.
5. The ticket is actually $\{K_{AB}, \text{Alice}\}_{KB}$. Bob uses K_B to decrypt it, checks that Alice's name matches and then uses K_{AB} to encrypt responses to Alice.

- Timing and replay issues – addressed in N-S and Kerberos.
- Not suitable for e-commerce because authentication service doesn't scale…

Scenario 3:
Authenticated communication with public keys

Bob has a public/private key pair <K_{Bpub}, K_{Bpriv}>

1. Alice obtains a certificate that was signed by a trusted authority stating Bob's public key K_{Bpub}
2. Alice creates a new shared key K_{AB} , encrypts it using K_{Bpub} using a public-key algorithm and sends the result to Bob.
3. Bob uses the corresponding private key K_{Bpriv} to decrypt it.

(If they want to be sure that the message hasn't been tampered with, Alice can add an agreed value to it and Bob can check it.)

Scenario 3:
Authenticated communication with public keys

- Mallory might intercept Alice’s initial request to a key distribution service for Bob’s public-key certificate and send a response containing his own public key. He can then intercept all the subsequent messages.

Digital signatures

Requirement:

- – To authenticate stored document files as well as messages
- – To protect against forgery
- – To prevent the signer from repudiating a signed document (denying their responsibility)

Encryption of a document in a secret key constitutes a signature

- impossible for others to perform without knowledge of the key
- strong authentication of document
- strong protection against forgery
- weak against repudiation (signer could claim key was compromised – cde: huh? Applies to all above. So, nope!)

*

Not Used

Not Used

DHCP, NAT, Mobile IP, Bluetooth, Ipsec, ATM

Clark Elliott

Notes on

Dynamic Host Configuration Protocol

Network Address Translation

Etc.

From Coulouris, Dolimore, Kindberg, and CDE

Dynamic Host Configuration Protocol -- DHCP

- Large organizations like DePaul have leases on large numbers of real IP addresses.

- But faculty, administration, and lab machines come and go. One IP address for each student is also a big burden.

- Deciding who is bound to which IP address is a giant administrative nightmare: setting up each user’s machine (or multiple machines) is a giant time sink. Each machine has to be configured *before* it can use the network.

- Much more efficient to allocate IP address only when they are needed, on an *ad hoc* basis, without administration, from a pool of valid, real, IP addresses.

- When a user boots their machine, the DHCP client broadcasts a query to the DHCP server, which returns a valid IP address, a lease length, subnet mask, and default gateway.

- The crucial item: A VALID REAL IP ADDRESS

Three methods of DHCP allocation

- **Dynamic:** just give a client whatever is available from the pool at the time of the request. Typical use. Easy to implement. No administration.

- Automatic: Like dynamic, but keep track of what you've allocated, and always give that IP address to the client.

- Static: keep the MAC addresses of the clients in a table. When that MAC makes a request, assign them the associated IP address in the table. Lose many of the efficiencies, but don't have to configure clients.

Efficiency

- DHCP thus allows an organization to support many more computers than it has IP addresses, depending on who needs an IP address at any given moment.

- The biggest savings is in not having to administer the system. New machines and old machines alike configure themselves.

NAT is not DHCP

- Network Address Translation is something completely different.

- It also allows many more computers to use the internet than there are real IP addresses.

- NAT assigns users “FAKE” (local) IP addresses that cannot be used directly on the internet.

NAT is not DHCP

- NAT addresses work fine on local networks.
- NAT addresses are not unique. Many thousands (millions?) of users will be using the same NAT IP address at any given moment.

- A typical NAT installation is a home or office that has many computers, but little need to support servers. Internet use is primarily as clients.

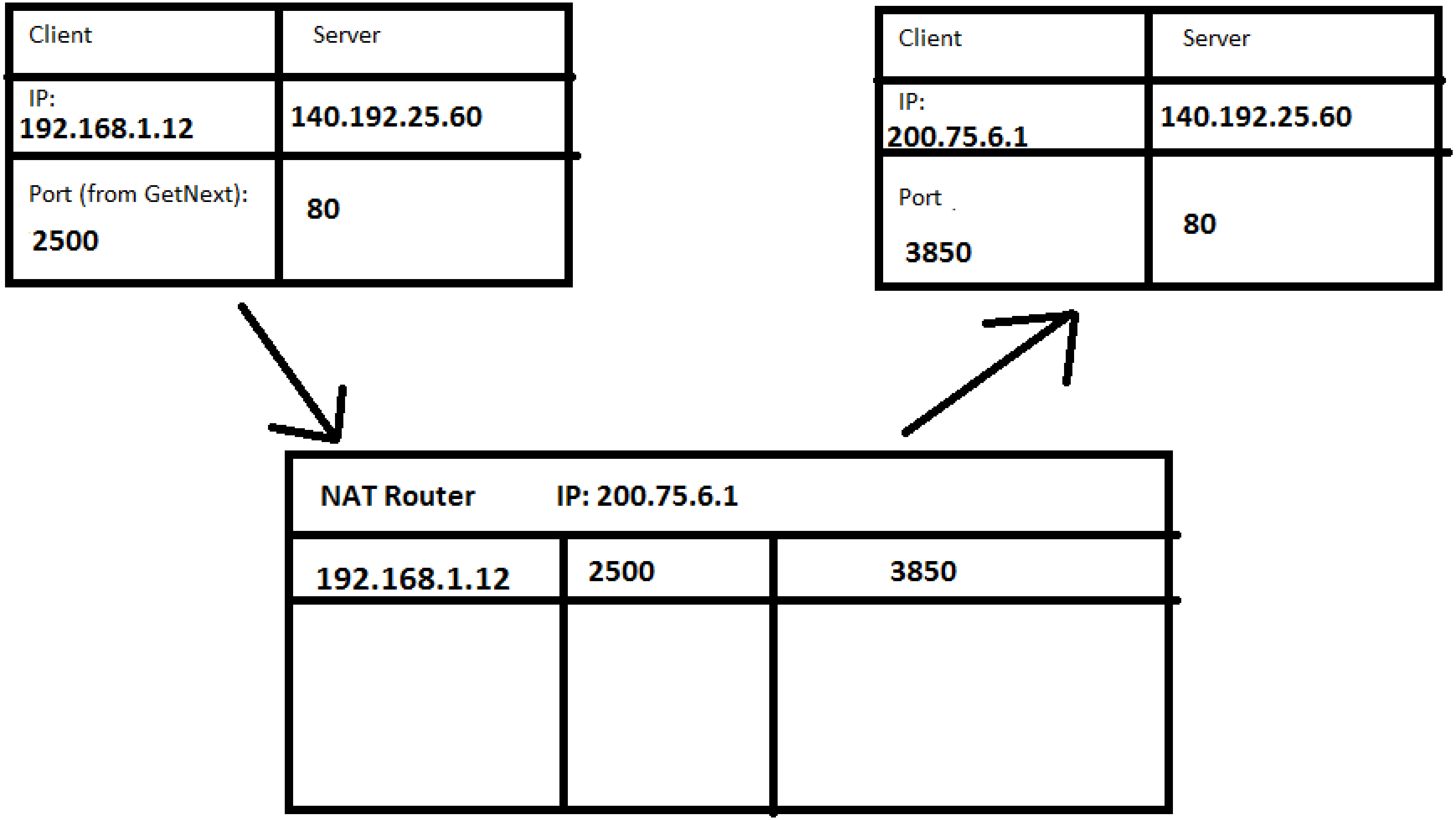
- A NAT-enabled router is required, through which the local network operates.

Network Address Translation

- Used in millions (?) of homes / offices

- 192.168.x.x is reserved for “home” domains, these are fake IP address primarily for internal IP hosts that operate as clients to internet.

Network Address Translation



Client	Server
IP: 192.168.1.12	140.192.25.60
Port (from GetNext): 2500	80

Client	Server
IP: 200.75.6.1	140.192.25.60
Port 3850	80

NAT Router	IP: 200.75.6.1	
192.168.1.12	2500	3850

How NAT works…

- Local host (lhost) sends request packet to internet via the NAT-enabled router
- Router saves the request IP/port in table
- Replaces the IP/Port with *router's IP address* and a *virtual port* that indexes the entry in the table.
- Sends to destination server with router IP/virtual port

- When the server replies, it uses the router IP/virtual port
- The router receives the reply, uses the (virtual) port for table lookup, replaces router IP/virtual port with the local IP and port of lhost.
- Sends to Ethernet
- Original sender retrieves from the local network.
- Keep entries for a while, then discard

Servers on NAT

- With many ISPs the router IP changes, so have to have a forwarding site to translate domain name, and wait for propagation.
- External clients use IP of router (or domain name pointing to it), and a specific port.
- Router forwards all incoming traffic for that port to the lhost at some manually assigned port.
- Lhost servers cannot use relative DHCP

Servers on your laptop

- When you take your laptop to a new location it discovers the local subnet and gets a local IP address from DHCP.
- Your laptop can now act as a client.
- Legacy service conversations with servers on your laptop can no longer find you.
- Address forwarding software can work around this, though with some drawbacks.

Mobile IP

- If server push is required, or others are using local resources that are mobile, then DHCP will not work
- IP address are subnet-based, for routing purposes, geographically fixed.
- Mobile IP uses home agent (HA) and foreign agent (FA).
- HA acts as a proxy to reroute all packets through a tunnel.
- Mobile-IP-enabled senders can communicate directly thereafter

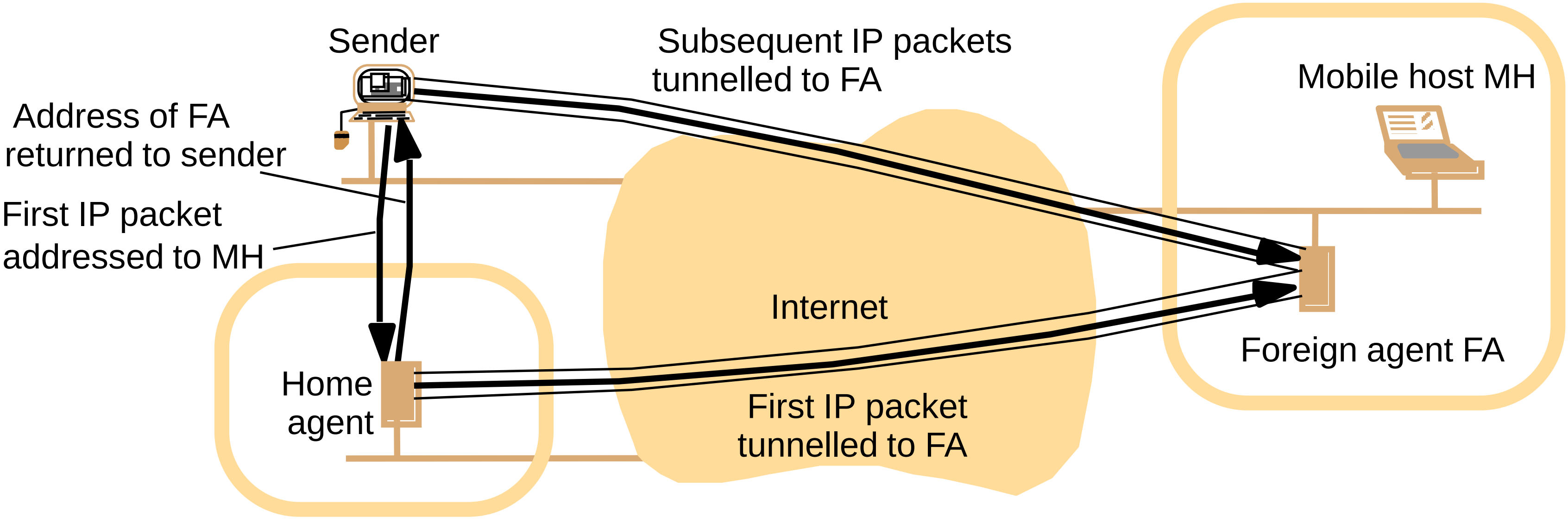
Your laptop

- When you take your laptop to a new location it discovers the local subnet and gets a local IP address from DHCP.
- Your laptop can now act as a client.
- Legacy service conversations with servers on your laptop can no longer find you.
- Address forwarding software can work around this but either low efficiency (every packet is forwarded) or both sides have to know to dynamically update the service address after initial handshaking.

Home agents and Foreign Agents

- Run at home host, and foreign host
- HA must have up-to-date knowledge of the IP location of mobile host (mhost).
- HA tells local routers to get rid of cached records of mhost.
- HA uses ARP (Address Request Protocol) to accept packets as a proxy for mhost.
- Mhost works with FA to inform HA of where it is.
- HA forwards packets wrapped in Mobile-IP packets.
- FA passes to Mhost.
- If sender is Mobile-IP enabled, can now communicate directly, otherwise repeat.
- Servers can now be mobile.

Figure 3.20
The MobileIP routing mechanism


Sender
Subsequent IP packets tunnelled to FA
Mobile host MH
Address of FA returned to sender
First IP packet addressed to MH
Internet
Foreign agent FA
Home agent
First IP packet tunnelled to FA

Not best solution

- Requires HA.
- Setup time
- Requires clients to be Mobile-IP or have to route ALL packets twice.
- Mobile phones are first-class citizens as they move from cell to cell.

Mobile agents

- Processes, code libraries, and *state* that move from one location on a network to another to perform work on a remote location's behalf.
- Interact with local resources, such as databases, other processes, and even users.
- Local invocations are cheaper than remote ones so may lead to efficiency.
- Different programming design
- Security and negotiation are complex because always two different stakeholders, host / agent

Mobile agents

- One generally unexplored area is using the idle resources of a nework
- Security problems can work both ways: both to the host, but also to the agent.
- Not hugely popular. Note that web crawlers “look” like agents, but are simply a series of local requests to various servers.

Mobile devices

- Wave of the future. Miniaturization of CPUs, lower power requirements, improved battery technology, wireless technology
- WiFi – term does not mean anything, just catchy
- In general DHCP and discovery of local resources is enough as a collection of *clients*

IPv6

- 128 bit addressing, or 1000 IP addresses per square meter of earth’s surface
- Geographic and organizational semantics in addresses
- Flow labels for QoS improvements
- Security headers – but note that destinations are not encrypted (?), authentication for RIP
- Slow migration, but 1 billion Devices in China and India by 2014, plus mobile IP needs.

IPsec (“IP security”)

- Operates at a *lower level* than TSL, SSL, SSH, so applicable to wide range of applications including basic TCP and UDP without requiring rewrite apps. Provides:
  - Encrypting traffic (so it cannot be read by parties other than those for whom it is intended)
  - Integrity validation (ensuring traffic has not been modified along its path)
  - Authenticating the peers (ensuring that traffic is from a trusted party)
  - Anti-replay (protecting against replay of the secure session).
- Uses IKE internet key exchange
- Mandated in IPv6, but forced onto IPv4 by need.

VPNs

- Virtual Private Networks
- Based on IPsec
- Typical: home ISP user exchanges keys with firewall, and can then access intranet
- Can also connect two intranets

Bluetooth IEEE 802.15, 2002

- Developed by Ericsson.
- Voice and data
- $5 per device
- Low battery consumption – several hours even with earpiece battery. Wake up and listen for a paging call, so long latency to establish contact.
- Adaptable power transmission from 1 Mwatt (10 meters) to 100Mwatt (100 meters)
- Switches 1600 times / second between 79 sub-bands of 2.4 GHz public frequency for reduced interference

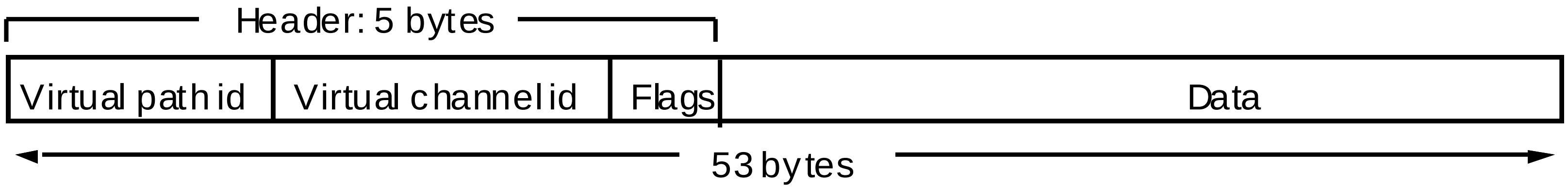
Master / slave

- Pairs operate as master / slave, with master allocating communication slots
- Piconet – up to 7 slaves one master and up to 255 *parked* (dormant) slaves in low power mode waiting for a wakeup signal.
- Scatternet – linked piconets.
- Most devices can be master or slave.
- All devices have 48-bit GUIDs, but slaves assigned 1-7 to save space.

Bluetooth…

- Designed for QoS required of audio.
- Synchronous and async transfer.
- Up to 1 Mbps
- Includes specifications for app-level protocols to encourage interoperation of manufactured devices.
- Association of new / parked devices is slow, up to 10 seconds – so not for e.g., toll road pickup of vehicles.
- Version 2.0: 3 Mbps (cd quailty audio), larger piconets, faster association is in the works.

Figure 3.27
ATM cell layout



ATM – asynchronous transfer mode

- Fast packet switching w/ suitable QoS for multimedia transmission
- Avoids flow control and error checking
- Fixed-length units of data (cells) 53 bytes
- Establishes a connection first, guarantees bandwidth and latency. Frame relay does not store cell/frame.
- High-speed switching, low latency
- Similar speed to LANs.
- High QoS and speeds of 600Mbps for multimedia will be available for internet.
- Pure ATM up to 1 Gbps.
- Slow to adopt because of expense compared to e.g., 1000Mbps ethernets.

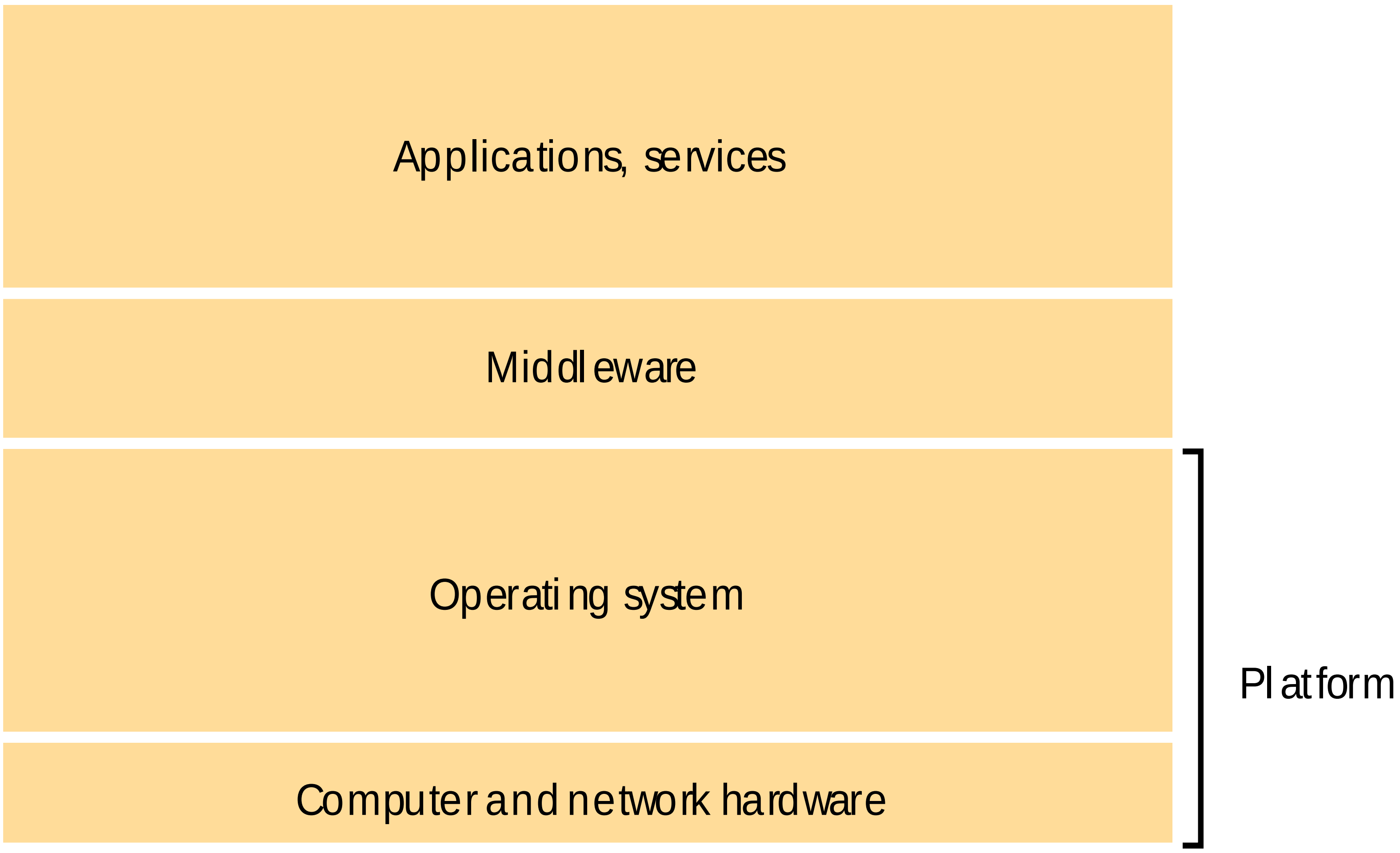
Have to address…

- No global clock
- Communication is via messages (without clock!)
- Delays, and high failure
- Vulnerable to attacks on message system
- Remote administration problems
- Massive scale-up problems
- Classic problems like the server-binding problem

Falacies

- Eight fallacies of distributed computing:
    1. The network is reliable.
    2. Latency is zero.
    3. Bandwidth is infinite.
    4. The network is secure.
    5. Topology doesn't change.
    6. There is one administrator.
    7. Transport cost is zero.
    8. The network is homogeneous.

Figure 2.1
Software and hardware service layers in distributed systems



Layers

- Hardware, microcode, bit level binary code and assembly. OpSys privlidged and user operations. [Virtual OpSys], shell wrapper, [application VM], Middleware, application
- Middleware provides homogenous interface for application development. May handle, e.g., message passing, buffering, reliability, security.

Middleware

- RPC, RMI, DCE, sockets, CORBA, .net
- Always a compromise: ease of use vs. reliability, efficiency, and maintainability of the middleware.
  - Example. Very large mail file. *Application* probably needs to keep a local copy to resend if there are major network problems, and not just rely on TCP.

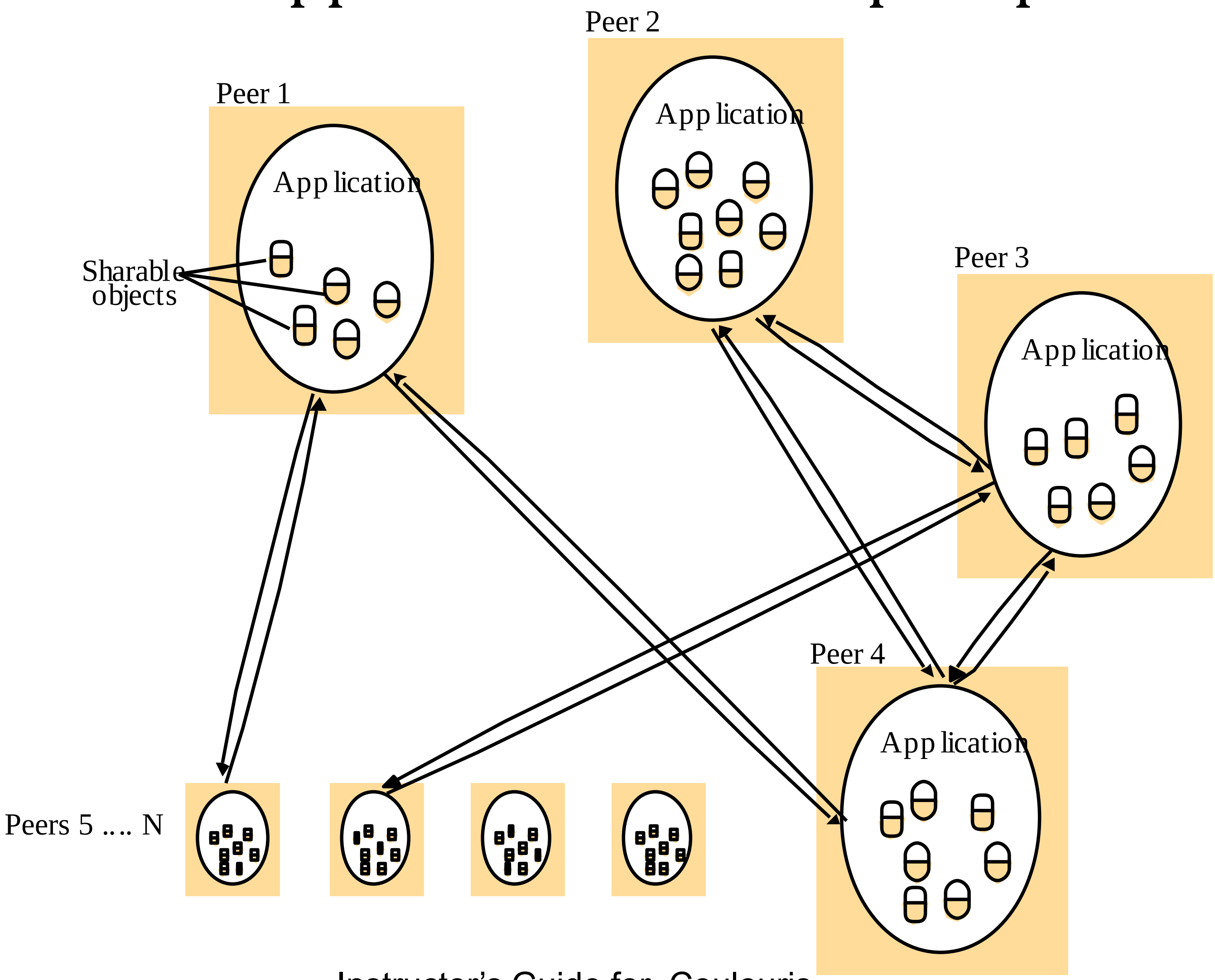
Peer-to-peer

- Claim is made that client/server does not scale well, but, of course, this is nonsense.
- It does not scale *automatically*, but instead requires the expansion of network, server machines, etc.
- Peer to peer e.g., can take advantage of massive wasted computer power sitting on desktops. Napster, Kazza, Gnutella, BitTorrent

Peer-to-peer

- Forces nodes to contribute.
- Same functional characteristics at most, if not all, nodes
- No central authority or failure point.
- Can provide anonymity
- Balances workload across nodes
- Large address space via GUIDs
- Best for static data because secure hash discourages malicious nodes.

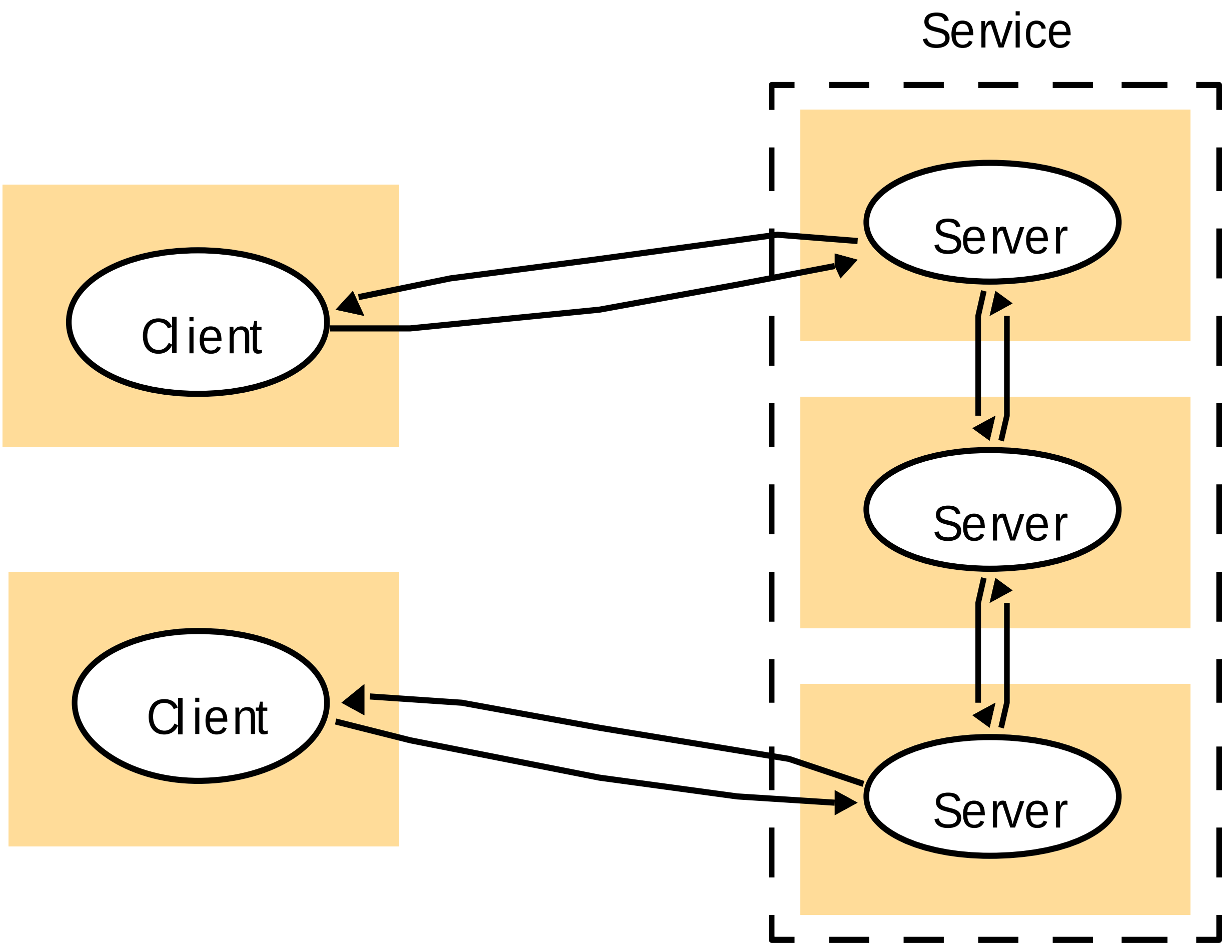
Figure 2.3
A distributed application based on peer processes



Services

- Logically decoupled from the endpoint (ip address / port) of any particular server, but instead clients subscribe to a *service* that might be provided by *many* servers.
- Might have redundancy (replication), or specialization (partitioning) so that multiple servers participated for single client.

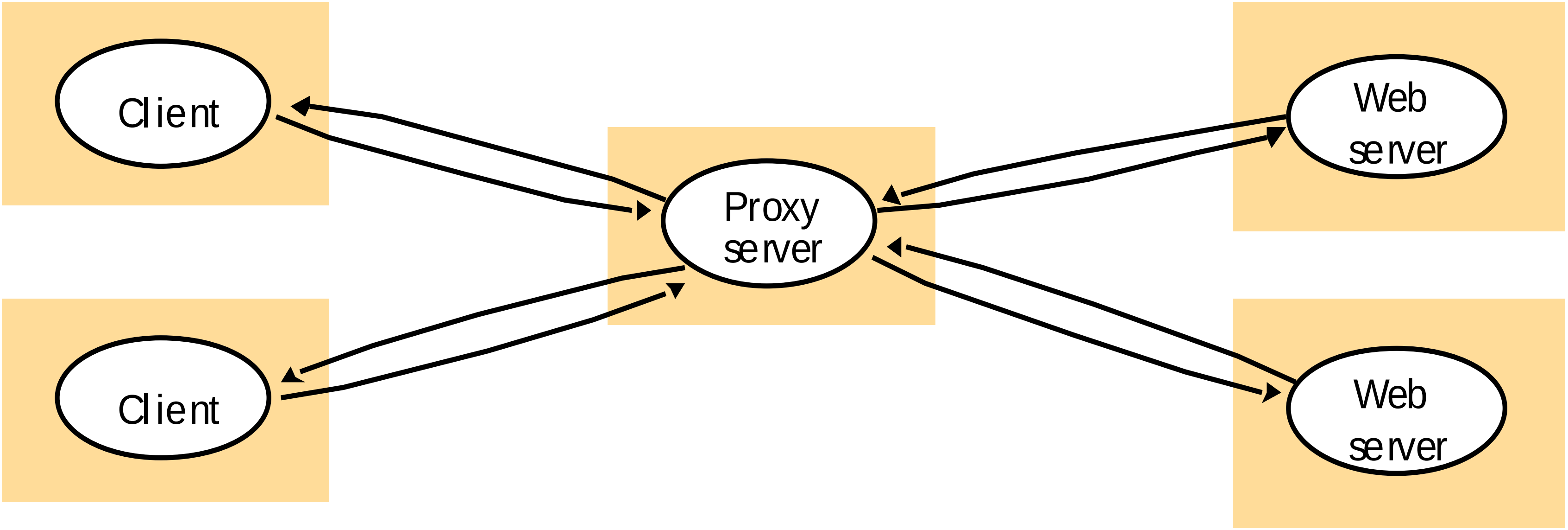
Figure 2.4
A service provided by multiple servers



Proxy servers

- Extra server level that acts as a *filter* for requests
- Caching for efficiency, firewall for protection, route to redundant servers for availability and ease of maintenance, mobile IP routing.
- Typical: caching of popular web pages

Figure 2.5
Web proxy server

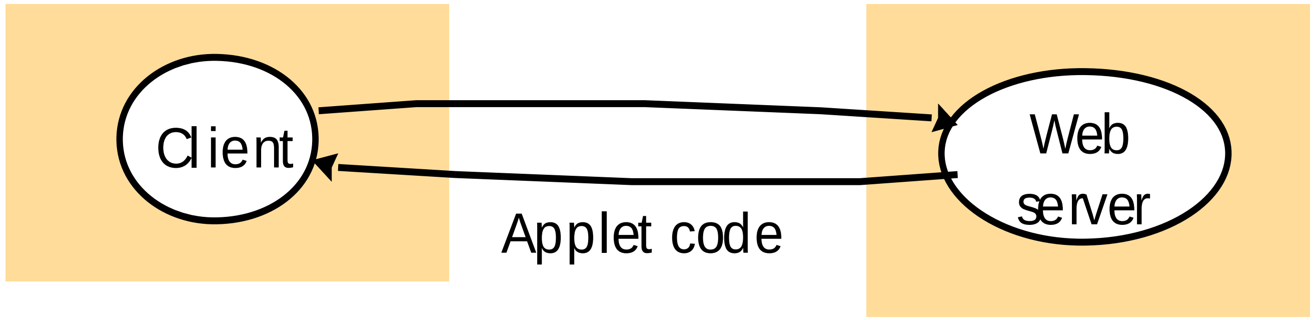


Mobile Code

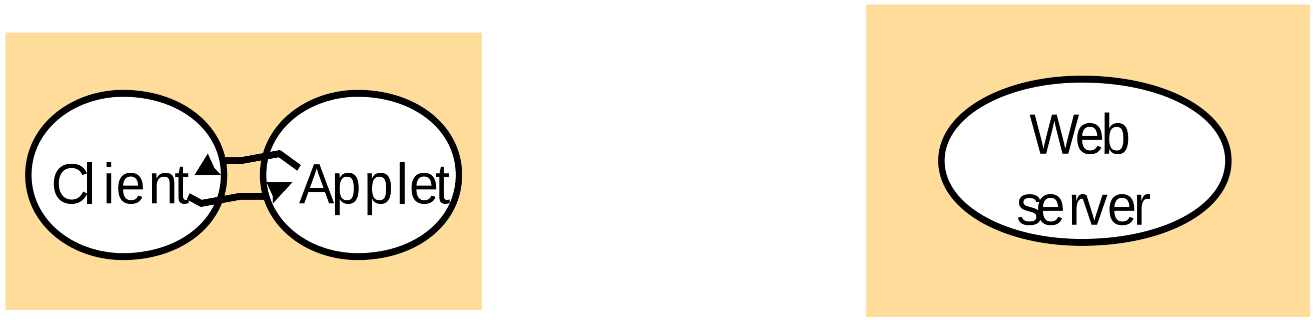
- Code moves from one location to another before or during execution
- Strong mobility – running processes moved while in progress
- Weak mobility – always started from the beginning after moving. Java applets.
- Code may communicate with server

Figure 2.6
Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet



Mobile agents

- Processes, code libraries, and *state* that move from one location on a network to another to perform work on a remote location’s behalf.
- Interact with local resources, such as databases, other processes, and even users.
- Local invocations are cheaper than remote ones so may lead to efficiency.
- Different programming design
- Security and negotiation are complex because always two different stakeholders, host / agent

Mobile agents

- Security problems can work both ways: both to the host, but also to the agent.
- Not hugely popular, but this does not mean they are not the best solution! The old way is not always the best way.
- Note: these are not web crawlers which “look” like agents, but are simply a series of recursive local requests to various servers.

A hypothetical case

- Acme Agents provides optimization software for large proprietary construction company databases.
- Ng Construction Company doesn’t want to leak anything about its customer base, business models, or contracts.
- Ng is leery about running software that has a real-time connection over the internet through which their data passes.

Acme Agent's concerns

- Acme Agents uses a database of its own tweaked for construction company DBs.
- They are reluctant to send their software to Ng and possibly expose their intelligent optimization data, putting them out of business if it leaks.
- It is expensive to send a person to the various construction company sites.

General constraints

- This is a large database, with e.g., many image PDFs of contracts, and construction photos.
- It is computationally expensive to ship the data over network lines.
- Optimization problems occur at times of high load, and DB queries are much better done locally.

A hypothetical agent solution

- Acme ships an optimization agent to Ng, which agent maintains its own internal state.
- Protocols are set up for the two stakeholders to protect their states. Ng and Acme and work through a dual-direction "firewall."

- The database is queried and stressed locally by the agent, using the intelligent software and knowledge contained in the agent, such that information is gained, tests are run, solutions tried, but all working with the local company Ng to not disrupt business use.

- Acme learns more about the problems that can occur in large construction company DBs, along with solutions that (a) work and (b) that don’t work. Acme stores this knowledge in the state of the agent.
- Design/optimization problems (only), stripped of client business information is shipped back to Acme for review during the optimizing process as needed for human intervention

Agent returns home, the off to work again.

- After making/recommending optimizations to Ng's database, the Acme agent is shipped back to Acme, *along with its new internal state.*
- When Lee construction company next hires Acme to optimize its database, the knowledge about optimization of construction databases is used, but none of Ng's business model, or its customer base is exposed.

Benefits

- Lee construction gets better, faster, cheaper optimization.
- So does Ng the next quarter when Acme returns next quarter with the knowledge from Lee's optimization event.
- Acme manages optimization knowledge, and Ng and Lee manage construction knowledge.

- Neither Ng nor Lee has their business data exposed.
- No one else can snoop on business data traversing the internet.
- Realistic control of stressing a database locally.
- Part of the internal state is the *relationships* the agent maintains to other entities—in this case Ng and Lee, which persists.

- This is a variation of shipping the executable code to the site of big data.
- The critical element is that agents maintain their own (often protected) internal state.

Network computers

- Manage files and other critical resources in a central location, correctly, under opsys control. Make transparent to users.
- Very thin client
- Local disk is primarily for caching
- Remote desktops are related, X-11 runs local display *server* on the user machine
- Lucent's plan 9: http://www.ecf.toronto.edu/plan9/plan9faq.html

Mobile devices

- Wave of the future. Miniaturization of CPUs, lower power requirements, improved battery technology, wireless technology
- WiFi – term does not mean anything, just catchy
- In general DHCP and discovery of local resources is enough as a collection of *clients*

Mobile IP

- If server push is required, or others are using local resources that are mobile, then DHCP will not work
- IP address are subnet-based, for routing purposes, geographically fixed.
- Mobile IP uses home agent (HA) and foreign agent (FA).
- HA acts as a proxy to reroute all packets through a tunnel.
- Mobile-IP-enabled senders can communicate directly thereafter

Discover card goofiness

- Password for discover account was sent to me as clear text in email after a problem, and not at my request.
- Exposed to internet attack, exposed to attack on my unix machine, exposed to mail process attack.
- Was original password I typed in a month earlier, which means what…?!!
- Argued for twenty minutes with the “top security guy” who was an idiot.

AT&T Security Evil / Morons

- To: [...]@yahoo.com
- Subject: AT&T Password Reset

- Dear Valued Customer,
  - Cellular Data Number is 858-xxx-nnnn
  - Your password is [xxxxxxxx]. Please use it when logging into your account via Settings on your iPad.
- Thank You,
- AT&T

Cryptographic Salt

- Random bit-string added to password before producing the encrypted version of the password. Stored on the server.
- Increases the randomness and the length of the password string.

Communication channels

- Latency – time between the start of a message send, and the start of the receipt at the other end. (Time to send an empty packet.) For acknowledgments requires roughly 2x latency value.
- Affected by distance, reliability of network (resends), layers of software, load on network and OpSys, security overhead (e.g., encrypting messages), proxies, superservers and service startup, load-balancing servers.

- Satellites have high throughput, but high latency values – so bad for interactive editing, but good for downloads

Bandwidth

- Maximum number of bits that can be transmitted over the network in a given period of time.
- Affected by overhead bits, errors, sharing loads with all processes.
- May vary greatly. Routing changes and loads are all dynamic.
- Measured as an *average*

Jitter

- Variation in time to deliver individual messages
- Difference between maximum and minimum times.
- Very important to audio streams, and streams that combine video and audio.

Clocks

- Individual computers each have their own clocks.
- Typically billions of cycles different.
- Local processes read local clocks.
- Perfect external reference clock – even if it existed, the time cannot be read in except through millions of instructions.
- So – time stamps are not very meaningful in a distributed system. (Remember the cached data?)

- Can use GPS attached to a computer, but not all boxes have radio access – but sending on local network is subject to message delay. Only good for mSec anyway (when a million instructions might be executed).
- Network time protocol. Other schemes to set clocks approximately.

Bits & Bytes

1. Transmission quantity is measured in **10^n** so that 64K is 64 x 1000 or **64,000.**
2. Computer quantity is measured in **2^n** so that 64K is 64 X 1024 or **65,536.**

This is a difference of 2.34%. Close enough for government work??

File size and storage is measured in terms of bytes but transmission is measured in terms of bits (i.e. don’t’ forget to **multiply/divide by 8**).

Here is a twister -> Remember the calculation for Transmission Time?

Transmission Time = RTT + (Transfer Size / Bandwidth)

We have a 100 ms RTT, a 64K sized file, and a 64K single channel.

TT = .1 s+ (64 * 1024 * 8) bps / (64 * 1000) bps
TT = **8.292** s

Binary Bytes vs. Decimal Bytes

- Network, 1K = 1,000, 1M = 1,000,000
- Computer 1K = 1024, 1M = 1,048,576
  - Except *disk drives* use powers of ten to artificially inflate the size of the drive.
  - 1 Terabyte disk drive = 931 GB
- So how long does it take to send the payload of 1 GB of data over a “1 GB” network line?
- http://www.codinghorror.com/blog/2007/09/gigabyte-decimal-vs-binary.html

It’s not rocket science – or is it?

- The power of ten / powers of two problem causes communication problems between CS people and IT people
- IT managers MUST understand this issue
- *“NASA lost a $125 million Mars orbiter because a Lockheed Martin engineering team used English units of measurement while the agency's team used the more conventional metric system for a key spacecraft operation, according to a review finding released Thursday.”* *(http://www.cnn.com/TECH/space/9909/30/mars.metric.02/)*

Calculating
RTT, Throughput, and Transfer Time

RTT is 2 X Latency and is the round trip time delay

Transfer Time = RTT + (Transfer Size/Bandwidth)

Throughput = Transfer Size/Transfer Time

Example: User wants to fetch a 1MB file across a 1 Gbps network with a 50 ms latency.

RTT = 2 X Latency = 100 ms
Transfer Time = .1 sec + (8 X 1,048,576)/1,000,000,000
= .1 sec + .008 sec
= 108 ms
Throughput = (8 X 1,048,576)/.108 sec = 77,393,816
= 77Mbps and not 1Gbps.

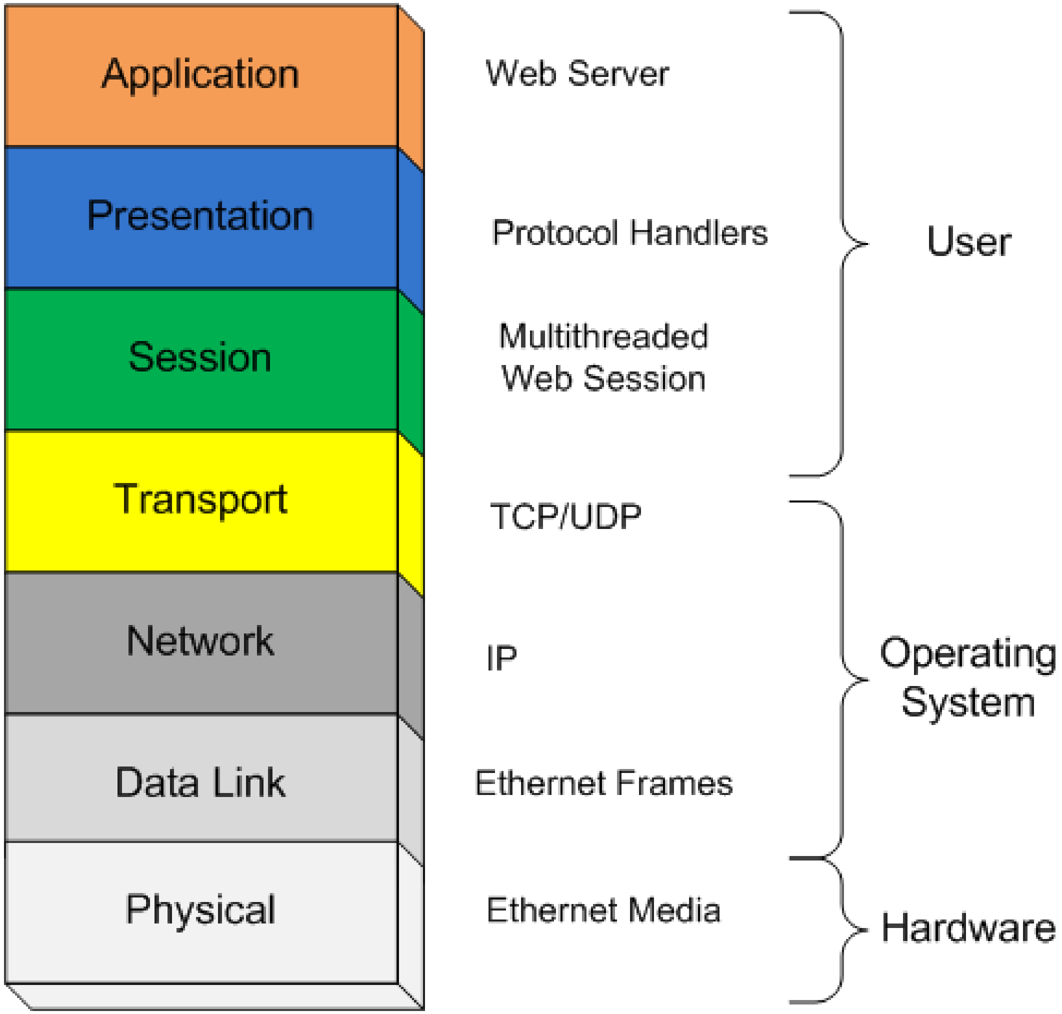
Switching schemes

- Broadcast – ethernet, WiFi. No routing
- Circuit switching – throw switches in line to prepare a circuit. POTS (plain old telephone system), train tracks.
- Packet switching – processing and storage of computers allows free real-time routing
- Frame relay – inspect first few bits, then pass on frame without storing or inspection. ATM uses this. Very fast, parallel, establish virtual circuit first.

Protocols

- Set of rules and formats
- Specification of sequence of messages to exchange
- Specification of the data format

OSI Protocol Stack Implemented



Application — Web Server

Presentation — Protocol Handlers

Session — Multithreaded Web Session

User

Transport — TCP/UDP

Network — IP

Data Link — Ethernet Frames

Operating System

Physical — Ethernet Media

Hardware

Layers → object code

- Each layer makes library calls to the level below which get included in the object code, or as SVC to operating system.
- Fully compiled code might have all layers compiled into executable, with entry point into running OpSys procs or kernal code.
- Each layer “hides” more complex code below.

Layers of detail – one line of app code = ~1500 lines of “real” code.

- Java App: “open a socket”
- .TCP / IP: bind, connect, listen…
- ..Allocate application buffers
- ...TCP: handshaking with other host, ports…
- .... Transport: break messages into units, route
- ..... Allocate OpSys buffers
- ....[etc.]
- ....translate to MAC ethernet…

Figure 3.2
Conceptual layering of protocol software

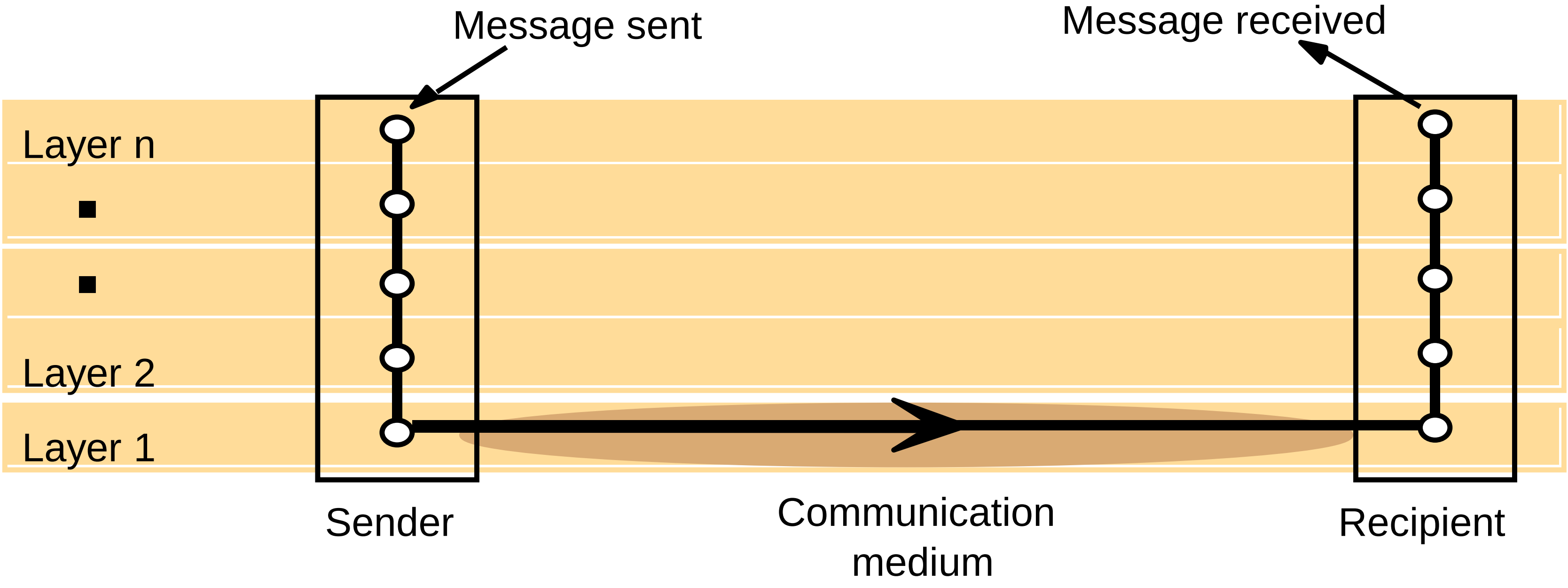


Figure 3.4
Protocol layers in the ISO Open Systems Interconnection (OSI) model

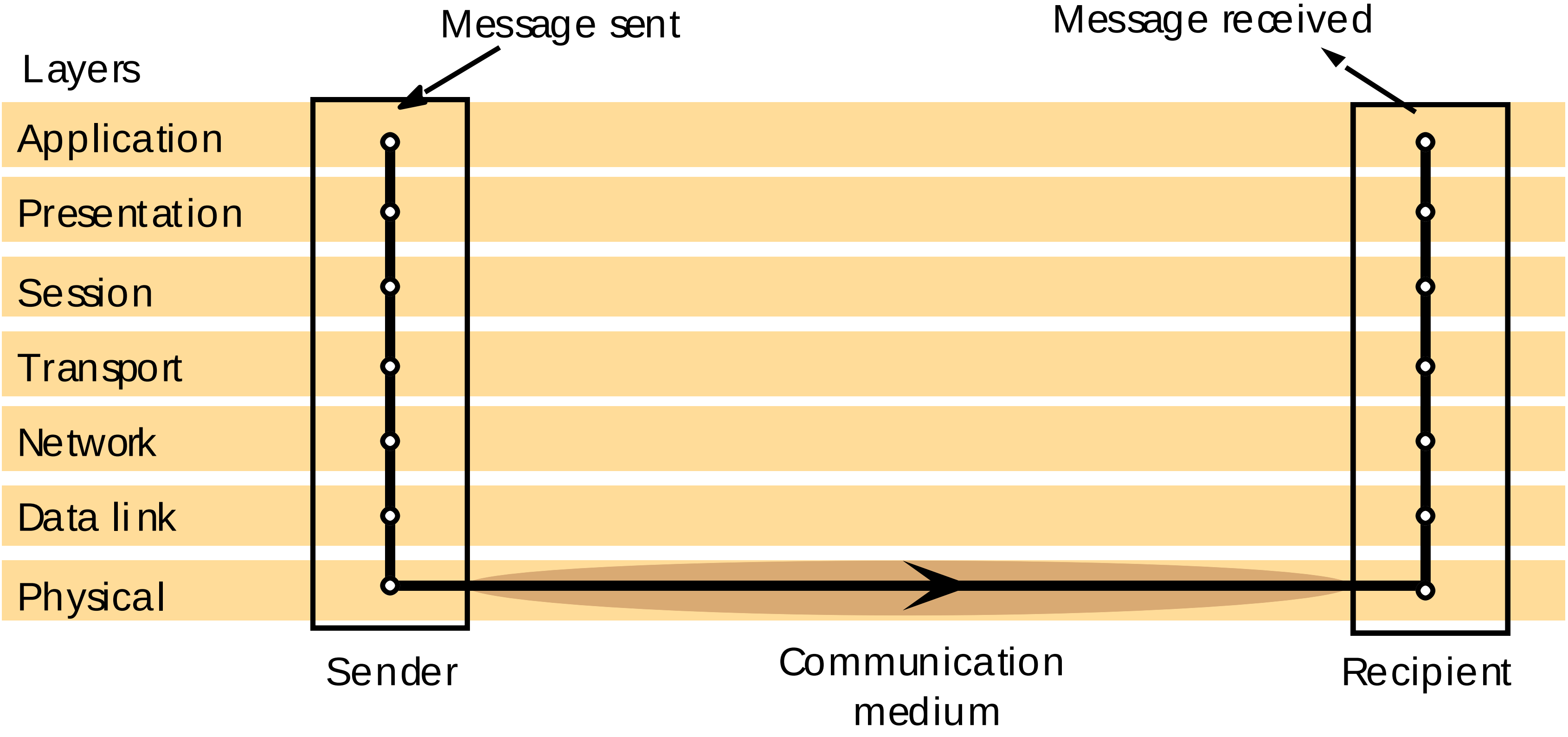


Figure 3.3
Encapsulation as it is applied in layered protocols

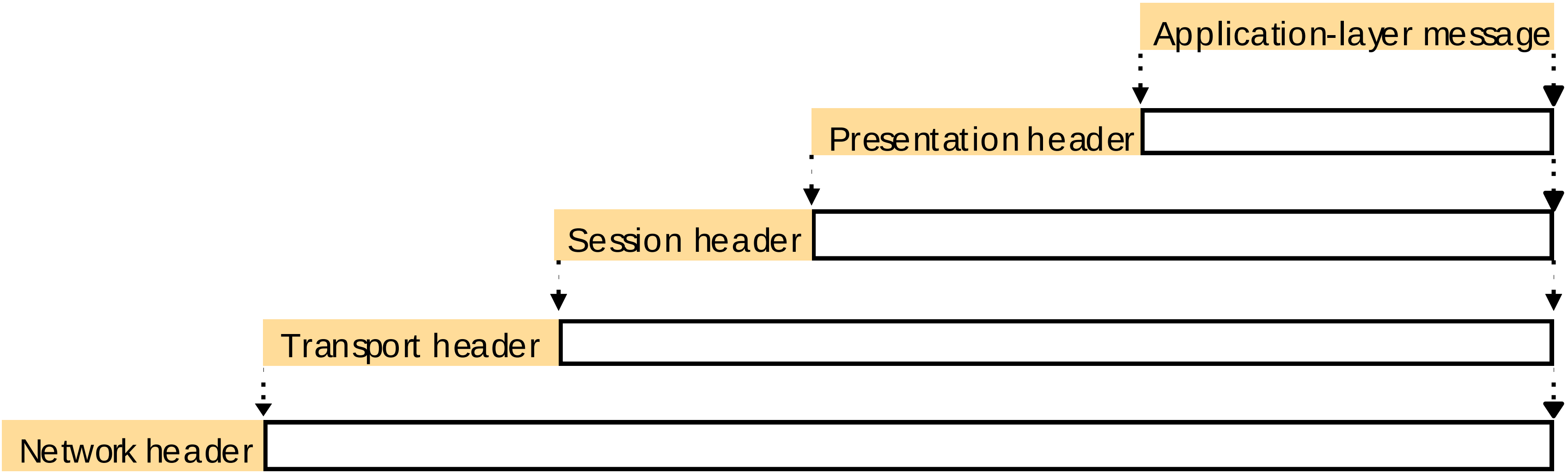


Figure 3.13
Encapsulation in a message transmitted via TCP over an Ethernet

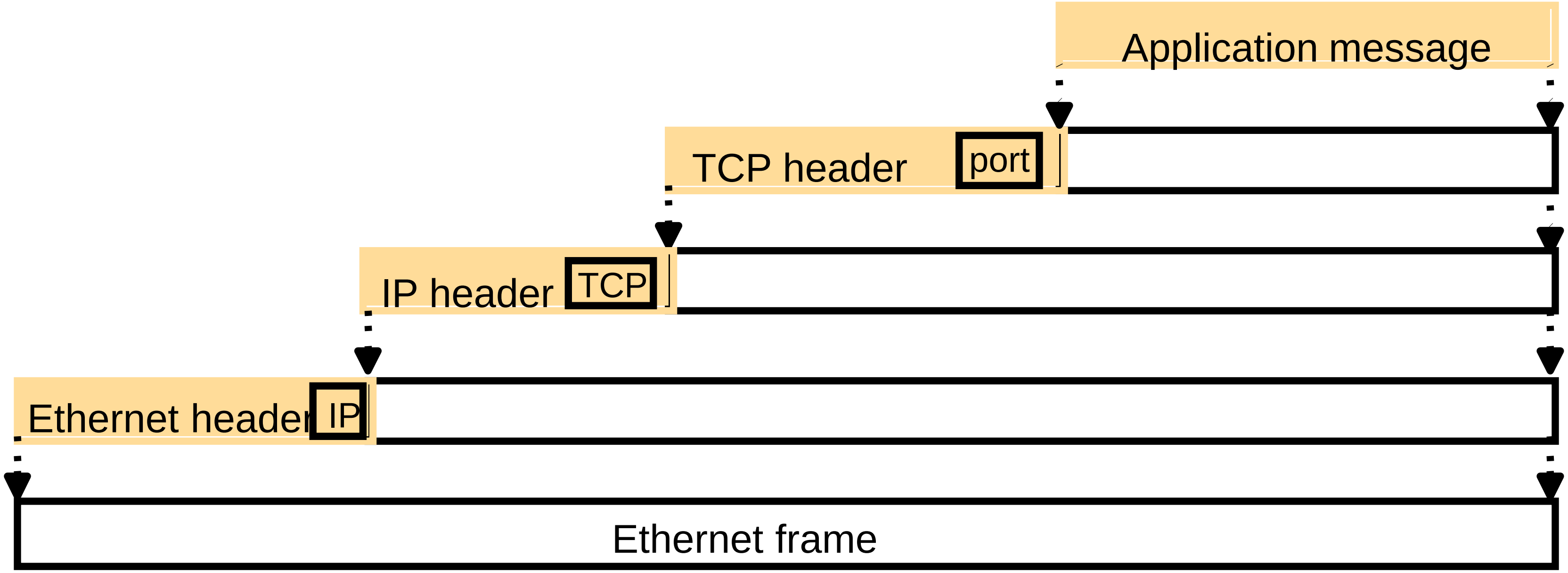
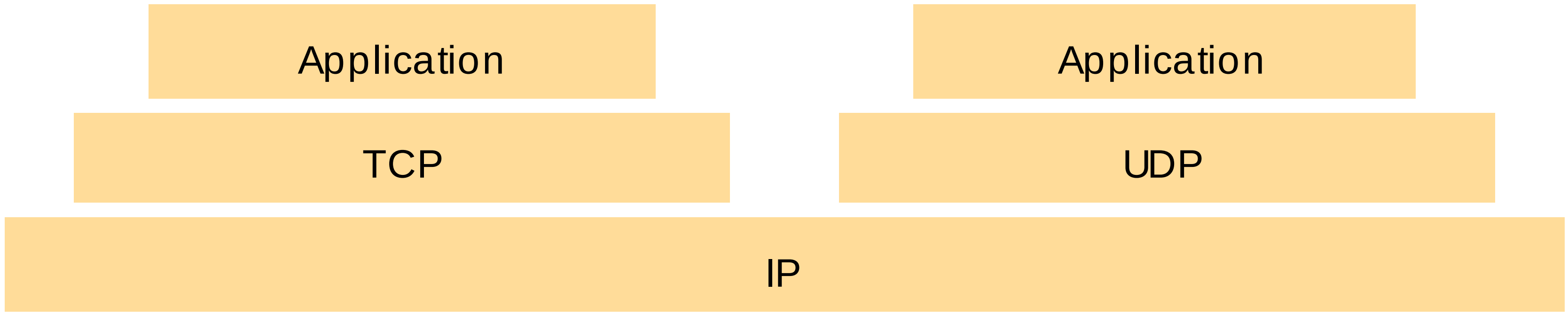


Figure 3.14
The programmer's conceptual view of a TCP/IP Internet



IANA Port Assignments

Port #	Service	Protocol
21	FTP	TCP/UDP/SCTP
23	Telnet	TCP/UDP
25	SMTP	TCP/UDP
80	HTTP	TCP/UDP/SCTP
88	Kerberos	TCP/UDP
118	SQL Server	TCP/UDP
443	HTTPS	TCP/UDP/SCTP

Updated this month http://www.iana.org/assignments/port-numbers

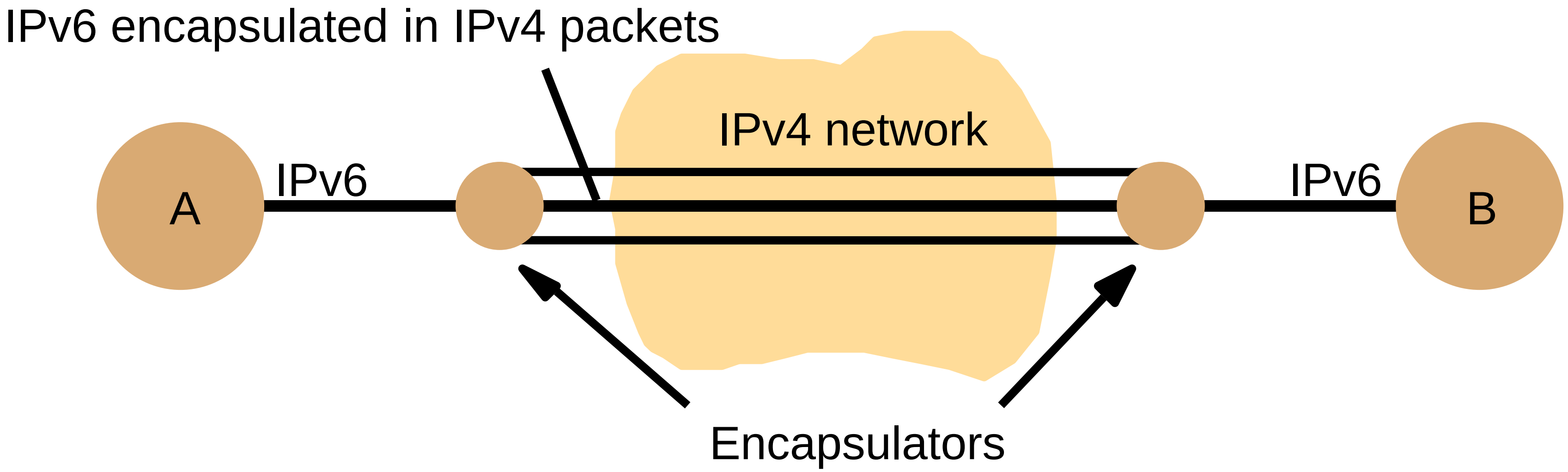
Migration to IPv6

- Address space has run out under IPv4
- A new standard IPv6 allows for many more addresses, and also includes improvements for security, multimedia QOS, and routing efficiency.
- Replaces IPv4 and is not compatible.
- Tunneling allows for graceful adoption.

IPv6

- 128 bit addressing, or 1000 IP addresses per square meter of earth's surface
- Geographic and organizational semantics in addresses
- Flow labels for QoS improvements
- Security headers – but note that destinations are not encrypted (?), authentication for RIP
- Slow migration, but 1 billion Devices in China and India by 2014, plus mobile IP needs.

Figure 3.11
Tunnelling for IPv6 migration



Conversion IP / local network address

- Different local networks have different addressing schemes.
- Ethernet is common, and has 48-bit addresses – ARP (address resolution protocol) is used to translate to/from IP.
- Sender: broadcast IP address query.
- Receiver: send back MAC address of NIC.
- Sender: place pair in table, then send packet using MAC address.
- Once all pairs are discovered, no query
- Beyond local net, get MAC of gateway & send

TCP & UDP

- Transport protocols, process-to-process communication msgs, uses *ports* for this.
- Processes acquire ports from the operating system through system calls.

Universal Datagram Protocol - UDP

- “IP up one level”
- Source and dest ports for addressing
- Length. Checksum optionally used by receiving host.
- Cheap, fast.
- No guarantees of delivery, or order of arrival.

Trasmission Control Protocol - TCP

- Reliable transmission of streams
- Connection-oriented.
- Guaranteed delivery in guaranteed order
- IP does not know about it. End-to-end.

TCP features

- Guaranteed sequence – requires “infinite” buffer – wait for first packet to arrive
- Flow control – windows, maximum delay before send, configurable, etc. so that receiver or network is not overwhelmed, and user does not have to wait too long.
- Auto retransmission of packets as needed
- Buffering as needed for flow control and ordering.

TCP Protocol

Connection-Oriented
Data packets are delivered in the correct order.

Reliable
Every piece of data will arrive.

Duplex

You can send information both ways at the same time, through the same connection.

Byte-Oriented
The data is presented as a stream of data rather than a set of packets.

How does this happen?

The Inner Workings of TCP

1. Each TCP connection provides a buffer that will hold the data until it has been verified that it has arrived.
2. The buffer also is used to order the packets in the correct sequence.
3. Every packet that is sent requires an “Ack” from the receiver so that it knows that it has arrived.
4. A CRC field is added and checked by the receiver to see that the data was not corrupted.
5. Each connection is a 4-tuple consisting of the port numbers for both the client and server, and the IP addresses as well.

Which has quite a bit of overhead!

Ethernet

- Cheap, reliable, local area network
- Long time defacto standard at 10Mbps
- 100 Mbps is cheap, and 1000 Mbps available.
- Broadcast of all packets
- NICs listen to all traffic and normally ignore all that does match, pass those that do up the protocol stack.
- Separate send and receive – don’t start send if other is.
- Collisions can occur resulting in resends, and ultimately thrashing if overloaded. Longer cable / more collisions.
- Use of switches/software, reduces collisions & helps guarantee latency and delivery for QoS. Not obsolete yet

Ethernet

- Uses CSMA/CD:
- Sense if the line is free
- Every computer has full-time access – Multiple Access
- Detect collisions and resend after a random delay.
- Medium Access Control (MAC) is protocol, hence *MAC* addresses are used in hrdwre

Slot reservation extension to Ethernet

- Collision avoidance augmentation via *CSMA/CA;* reserves transmission slots.
- Short negotiation packets (Request to send, RTS; clear to send, CTS) are sent first between sender and receiver, with duration. Others wait until slot has passed.
- Fewer collisions with short packets.
- Improves wireless reliability and QoS.

- Note have not added CDK chapter 4 or 5 yet.