Lecture 3
Introduction to Distributed Systems
(06-10,06-11,06-15-2009. Chapter 3, 1, 2)

Lecture Outline

1. Computer networks (Chapter 3)
2. Architectural models of distributed systems (Chapter 2)
3. Major challenges (Chapter 1)
4. System architectures
5. Fundamental models

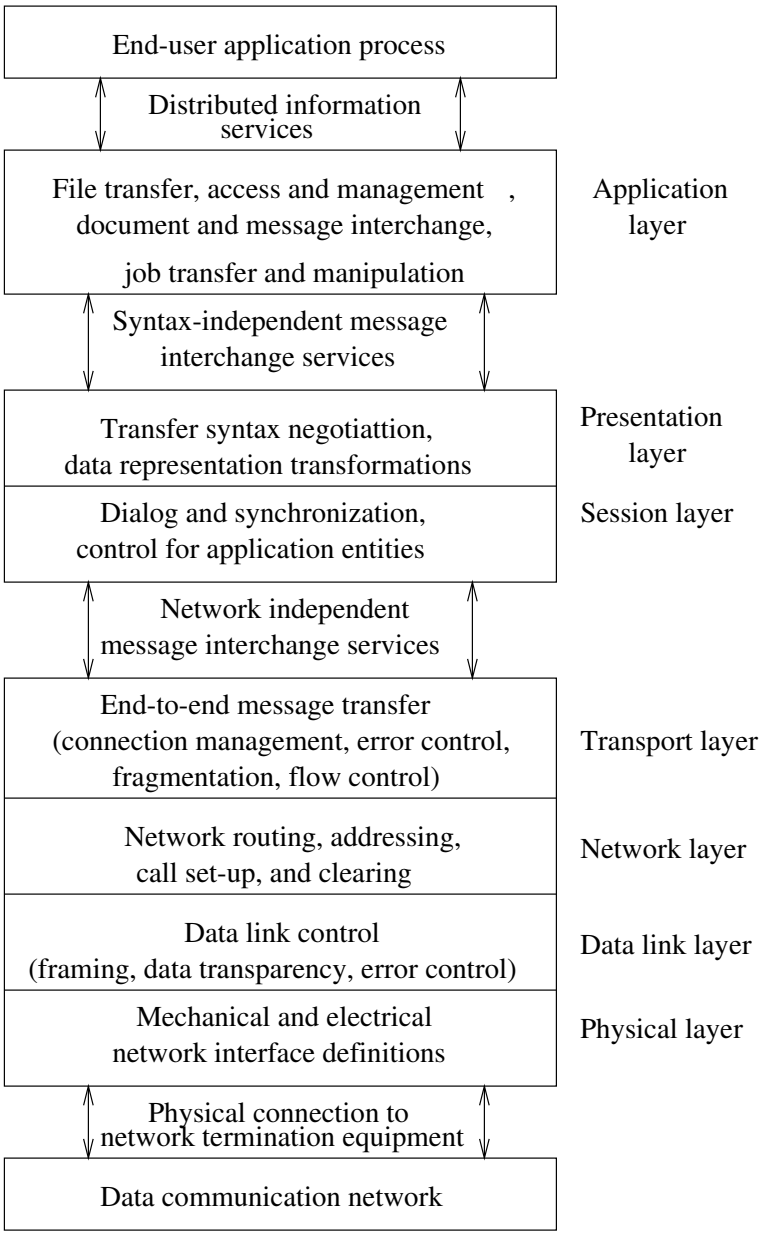


Figure 6: OSI reference model

(1) Communication protocols

- A communication protocol is a set of rules used for a designated session of communication.
- High-level protocols – used for communications occurring at high-level exchange of information.

– Low-level protocols – used for communications occurring at low physical level exchange of information.

(2) Connection and connectionless oriented communications

a. Connection oriented communications (telephone systems, ftp). Advantages: reliable.

b. Connectionless oriented communications (post mails, emails in BITNET). Advantages: flexible, efficient.

c. Datagrams (packets) and packet (data) switching vs circuit switching

(3) The Seven-layer ISO OSI Reference Model (Fig.3.4, p.77)

a. Principles

(a) Layer creation criteria: level of abstraction;
(b) Each layer: perform a well-defined function;
(c) Layer function: easy to be standardized;
(d) Layer boundaries: minimize information flow between layers;
(e) Number of layers: reasonable.

b. The Physical Layer

Services: transmitting bits through physical channels
Design issues: ensure that a bit 1 sent will be received as 1

c. The Data Link Layer

Gen. Services:	providing error-free transmission services to its upper layer
Design issues:	disassemble messages and reassemble messages
	detecting loss of messages, retransmitting, detecting duplicates
	flooding control, piggybacking

d. The Network Layer

Gen. Services:	controlling the network operations
Design issues:	routing: static, semi-dynamic, dynamic
	congestion control
	accounting
	internetworking

e. The Transport Layer

Gen. Services:	Accept and pass data
Design issues:	multiple and multiplex transport
	determine service types to session layer and network

(error-free point-to-point FIFO channel, no guaranteed order)
the first end-to-end layer
connection management

f. The Session Layer

Gen. Services: establish sessions between users
data transport
enhanced services: remote login, file transfer
token management
synchronization

g. The Presentation Layer

Gen. Services: ensure syntax and semantics of the information
coding and encoding
data compression, cryptography

h. The Application Layer

Gen. Services: general and special user-support facilities
Examples: terminal mapping
file system mapping
electronic mails

(4) Local area networks (LANs, Sect.3.5)

a. LANs and WANs

b. Characteristics of LANs

(a) A diameter of no more than a few kilometers;
(b) A total data rate of at least several Mbps;
(c) Complete ownership by a single organization.

c. Special factors in LAN design

(a) High bandwidth; designers are not restricted to use public lines;
(b) LAN cables are more reliable than those in WANs;
(c) Criteria for protocols are different due to the availability of high bandwidth channels;
(d) Network architecture is simpler in LANs than in WANs.

d. IEEE Standard 802: Fig. 3.22, p.112

(a) Three IEEE standards for LANs: CSMA/CD, token bus, token ring, called IEEE 802.

(b) IEEE 802 was also adopted by ANSI and ISO (ISO 8802) as international standard

(c) 802.1 gives introduction to the set of standards, and defines interface primitives. 802.2 describes the upper part of the data link layer using the LLC (Logical Link Control). 802.3 MAC Sub Layer Protocol – a bus utilizing CSMA/CD protocol.

(5) IEEE Standard 802.3 and Ethernet
802.3 is for a 1-persistent CSMA/CD LAN, the general ideas are:

a. When a station has a frame to transmit, it first checks the channel (carrier sense)

b. If the channel is busy, the station waits until it is idle

c. If the channel is detected idle, the station transmits its frame

d. If two or more stations transmit frames simultaneously, a collision occurs

e. The station stops transmission immediately once it detects the collision. It waits for a random amount of time before doing retransmission

(6) IEEE Standard 802.4: Token Bus

a. 802.4 is a collision free protocol.

b. The protocol. Basic ideas:

(a) Physically, the token bus is a linear or tree-shaped cable attached with stations.

(b) The whole network is logically organized as a ring. A station knows its left and right neighbors

(c) A special control frame is circulating around the logical ring. Only the token holder is allowed to transmit frames. When a station receives the token, it may transmit its frames for a certain amount of time, and then it must pass the token to one of its neighbor stations. If a station does not have any data to send when it receives the token, it passes the token immediately.

(d) Frames are divided into four priority classes: 0, 2, 4, and 6, with 0 being the lowest, and 6 the highest.

c. Logical ring maintenance

(7) IEEE Standard 802.5: Token Ring

a. Why rings?

(a) Ring structures can be used for both LANs and WANs. Ring technique are well developed.

(b) IEEE 802.5 was based on the IBM's ring networks. IEEE 802.5 is called *token ring*.

b. Ring structure.

(a) A ring consists of a collection of ring interfaces connected by point-to-point lines.

(b) Each bit arrives at an interface is copied into a 1-bit buffer and recopied out onto the ring again. The bit in the buffer can be inspected and possibly modified before being copied out. This copying introduces 1-bit delay at each interface.

c. **Token.** Like in token bus, a special bit pattern, called **token** is circulating around the ring.

(a) When a station wants to transmit its frames, it first must grab the token, and remove it from the ring. It then starts its transmission.

(b) After transmission, the station must return the token to the ring.

(c) Token rings are collision free

d. Two modes of the interface: listen and transmit. To be able to switch from listen to transmit mode in one bit time, the interface needs to buffer frames.

e. Ring Maintenance. Token bus handles ring maintenance in a distributed fashion, in the sense no central controller is present. Token ring handles the ring maintenance in a centralized fashion. Each token ring has a **monitor station** which oversees the ring. If the monitor station goes down, the contention algorithm will elect another monitor station. Each station has the ability to become the monitor station.

(a) Monitor's responsibilities:

· Seeing if the token is lost;

· Taking actions if the ring breaks;

· Cleaning up the ring when garbled frames appear and watching out for orphan frames.

(b) To check for if the token is lost, the monitor has a timer that is set to sum of the longest token holding time of each station. If the timer goes off, the monitor drains the ring and issues a new token.

(c) If a frame is damaged, the monitor detects it by its invalid format or checksum. The monitor then drains the ring and issues a new token when the ring is clean.

(d) The monitor detects orphan frame by using the monitor bit in the access control byte.

(e) Monitor introduce extra delay if necessary.

(8) TCP/IP Protocol Suite: Sect.3.4

a. Relationship to OSI reference model (Fig.3.12, p.90, Fig.3.13, p.91)

b. IPv4 vs. IPv6

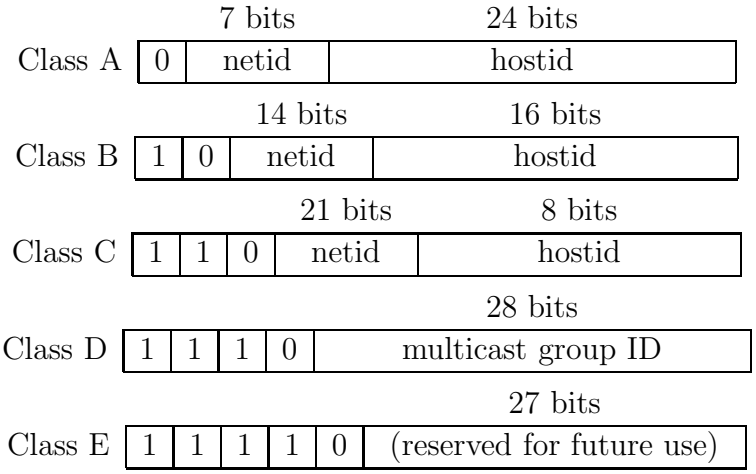
(a) IPv4 number: 32 bit integer, divided into 4 decimal integers.

(b) IPv6 number: 128 bit integer, divided into 32 hexadecimal numbers. Example:

5f1b:df00:ce3e:e200:0020:0800:2078:e3e3

c. IPv4 addresses (Fig.3.15, p.92, Fig.3.16, p.93)

(a) An IPv4 Internet address occupies 32 bits, and encode both the network ID and host ID. The host ID is relative to the network ID. Every host on a TCP/IP Internet must have a unique 32 bit address.



(b) Address ranges:

Class	Range
A	**0**.0.0.0 to **127**.255.255.255
B	**128**.0.0.0 to **191**.255.255.255
C	**192**.0.0.0 to **223**.255.255.255
D	**224**.0.0.0 to **239**.255.255.255
E	**240**.0.0.0 to **247**.255.255.255

d. IP, UDP, and TCP

e. IP routing

2. Architectural models of distributed systems (Chapter 2)

(1) The whole system can be viewed as consisting of four layers (Fig.7).

a. Applications and services

(a) Server: a process that accept service requests from other processes

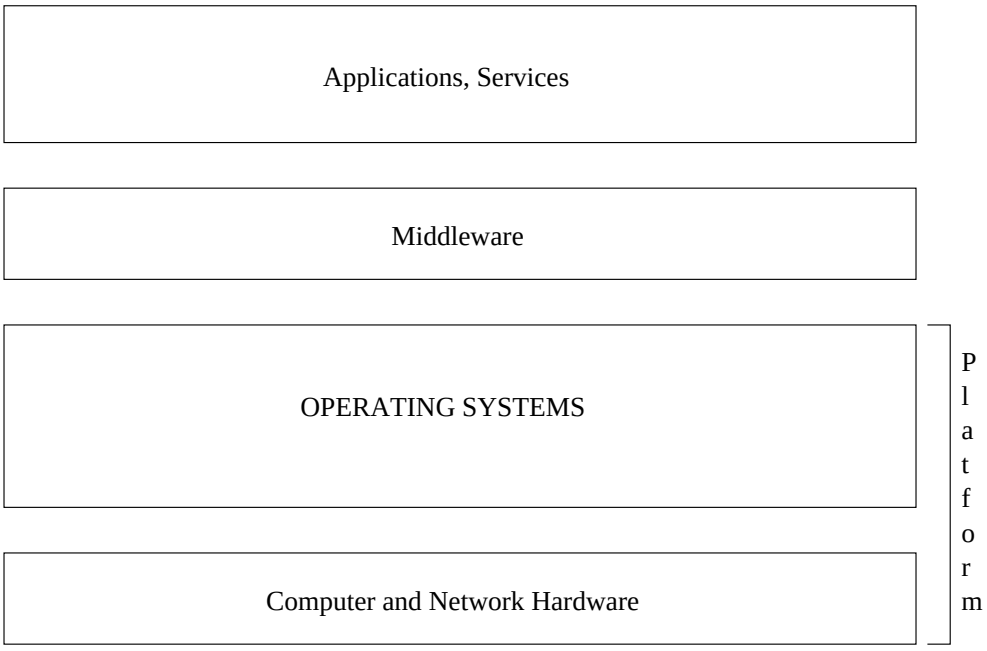


Figure 7: Software and hardware service layers in distributed systems (Fig.2.1,p.32)

(b) Service: a service can be provided by more than one server process

(c) Example:

- Web service
- NTP (network time protocol) service

b. Middleware

(a) Middleware is a software layer that provides programming abstraction and masking of heterogeneity of the underlying networks, hardware components, and programming languages. Example middleware:

- RPC facility
- Java RMI (remote method invocation)
- CORBA (Common Object Request Broker Architecture)

(b) Forms of components of middleware: they can be processes (services) or objects in a number of computers in a DS

(c) Main functionalities of middleware:

- providing high level building blocks that facilitate construction of software components. High level communication facility (that hide low level comm. details) in particular.
- providing services for use by application programs. Examples: naming services, transaction processing services, event reporting services.

(d) Limitations of middleware

- Example: Email transfer. Typically TCP based. TCP is reliable. In case TCP cannot establish connections, or email transfers of large emails are interrupted, a middleware component can be added to do retransmission (therefore providing more transparently reliable email services to users). However, if communications cannot go through for a prolonged period of time, the added middleware still cannot abstract the nature of comm. from users.
- The nature of distributedness of DSs requires communications for many essential functions provided by a DS.
- The problems that can occur in communications cannot be easily controlled or cannot be controlled at all by communication systems.
- Thus, in many cases middleware will only be able provide incomplete abstraction of services to users.

c. Operating systems

(a) Distributed operating systems (DOS) running on individual computers in a DS provide essential services to the middleware. Examples:
- File locking and data replication
- Process scheduling and management

(b) DOSs also interact with hardware components to hide hardware dependent features from middleware.

(c) Some DOSs services are part of middleware too

d. Computer and network hardware

(a) They are vendor specific. Different computers may have different characteristics. Different network hardware may have different comm. capabilities.

(b) Examples:
- FDDI (a high speed LAN) is faster than a traditional Ethernet. FDDI is ring based while Ethernet is bus based.
- The architecture of a Digital UNIX Alpha workstation is 64-bit. A Pentium III is 32-bit.

(2) The ISO OSI reference model is not suitable for distributed systems.

a. The model is a very general model for computer networks. DSs is based on but is on top of computer networks. A DS expects the underlying computer networks to provide all functions (such as connection-oriented comm.) that are supported by these networks. A DS is not concerned with how these functions are implemented.

b. The nature of DSs demands higher level support of abstraction and transparency. Distributed applications in a DS utilizes all functions by computers and network

hardware to implement *functional* services. The structure and interrelations of these services have little to do with the layered structure of the underlying computer networks.

3. Major challenges (Chapter 1)

(1) Heterogeneity

a. Questions:

(a) Should different types of computers be allowed in a DS?

(b) Should the users be allowed to know the existence of such machines?

(c) If yes, how to define and provide transparent services?

b. Types of heterogeneity:

(a) Computer hardware heterogeneity: different instruction sets, incompatible data representations (high-low, low-high orders). Can be handled relatively easily, except the last one. **Example**: different UNIX systems.

(b) Network heterogeneity: transmission media, signaling techniques, computer-media interfaces, lower-level protocols (eg. data link layer protocols).

(c) Operating system heterogeneity: most troublesome. UNIX OSs can hardly talk with VAX/VMS.

(d) Programming language heterogeneity.

c. Causes of heterogeneity

(a) Different machines were purchased at different times.

(b) Special needs or services.

(c) Personal preference.

d. Possible solutions: heterogeneity is a major problem that prevents efficient utilization of resources and management in DSs. A significant portion of middleware is dedicated to handling heterogeneity.

(2) Autonomy and interdependence

a. Two types of systems:

(a) The first type: each system controls its own resources and provides resource services to other systems (through RPC, RMI, and etc.).

(b) The second type: computers depend on networked services, such as filing and authentication (NFS in UNIX).

b. Advantages and problems of the 1st type DSs.

(a). Advantages: (i) Relative independence (autonomy). The crash of a single machine will cause a relative minor inconvenience. **Example**: Text editors; (ii) Higher security; (iii) Easier to manage.

(b) Problems: (i) Redundancy of functions and resources; (ii) Heterogeneity problem; (iii) Higher cost of resource maintenance.

c. Advantages and problems of the 2nd type DSs.

(a) Advantages: (i) Higher ratio of resource sharing; (ii) Lower cost of hardware and software maintenance.

(b) Problems: interdependence. The crash of a single machine (the file server) will cause the unavailability of many resources; **Example**: Text editors again; (ii) Complexity in software. **Example**: Yellow page services in UNIX. Must ensure security; (iii) Much harder to manage.

d. Summary:

(a) Autonomy and interdependence are both desired (as justified by the advantages listed above).

(b) Unnecessary interdependence should be avoided (or we lose the very purpose of having DSs).

(3) Openness

a. Openness: the degree by which a DS can be extended, re-implemented, and communicated with.

b. Openness can be achieved by using standard interfaces that are based on well-known standards

b. Summary

(a) Open systems are characterized by the fact that the interfaces are published, discussed and preferably well adopted.

(b) The communication interfaces and interfaces for accessing shared resources are essential.

(c) Open systems can be constructed from heterogeneous software and hardware through well designed middleware.

(4) Security

a. Due to distributedness, a DS is subject to security attacks that computer networks face. Examples:

(a) Financial information transfer

(b) Confidential personal information

b. Two major security issues

(a) Correctly identifying a remote user or other agent;

(b) Protecting the contents of sensitive information.

c. Security issues still under research:

(a) Denial of service attack: an attack may disrupt a service by making it not available

· by exploring bugs in server software

· by other means (eg. flood the computer hosting the service with requests)

(b) Security of mobile code. Eg: email virus. Difficult to prevent.

(5) Scalability

a. Explosion of computers and web servers in Internet

a) Computers in Internet with registered IP addresses (Fig.1.5, p.20)

Date	Computers	Web servers
1979, Dec.	188	0
1989, July	130,000	0
1999, July	56,218,000	5,560,866
2003, Jan.	171,638,297	35,424,956

Figure 1.5 Computers in the Internet

b) Computers vs. web servers in the Internet (Fig.1.6, p.20)

Date	Computers	Web servers	Percentage
1993, July	1,776,000	130	0.008
1995, July	6,642,000	23,500	0.4
1997, July	19,540,000	1,203,096	6
1999, July	56,218,000	6,598,697	12
2001, July	125,888,197	31,299,592	25
2003, July		42,298,371	

Figure 1.6 Computers vs. web servers in the Internet

b. Major scalability challenges:

(a) Controlling the cost of physical resources:

· Grow the DS at reasonably cost as demand for a resource grows.

· Grow the system linearly in response to demand. Eg.: a file server can server 20 users. A newly added same file server should server another 20 users.

(b) Controlling the performance loss:
   - As the size of a resource (such as a database file) grows, the time needed to access the resource will grow.
   - A well designed DS should utilize good hierarchical algorithms so that the time taken to access a resource of size n will grow as a function of $\lg n$ (i.e. $O(\lg n)$)

(c) Preventing exhaustion of software resources:
   - A resource not well structured may become unavailable after a period of growth.
   - Example: (i) IPv4 addresses. (ii) a database designed with a fixed up limit size.

(d) Avoiding performance bottleneck:
   - A hot resource in a DS may cause comm. congestion and slow response. Such resources then become performance bottlenecks that hinder performance.
   - Example: DNS database (distributed across Internet. Not easily becomes a bottleneck).

(6) Failure handling and recovery

a. Many types of failures are possible in A DS (Ch.2 has more details on types of failures)

b. Most failures in a DS are partial failures: the failure will not disrupt the whole system.

c. Major failure handling issues:

(a) Failure detection.
   - Eg.: using checksum to detect corrupted packets.
   - Not all failures can be easily detected. Eg.: non-response of a remote server may be caused by many possible reasons.
   - Challenge: detect failures earlier and accurately

(b) Failure masking: failure transparency.
   - Some failures can be masked from users. A file may be replicated.
   - Effects of some other failures can be made less severe. Eg.: a corrupted message can be retransmitted.

(c) Failure tolerance and redundancy. Services can be made redundant to allow failure tolerance.
   - A disk can be duplicated.
   - A raid array can be used to tolerate single disk failure.

- Between any two computers in a DS there should be at least two routes.
- For a service, more than one server running on different computers can be created.

(d) Failure recovery:

- In case a failure cannot be tolerated, a DS should undo (roll back) to non-permanent changes at the point of failure and return to its most recent consistent state before failure occurs.
- Check points can be used to log incremental consistent state.

(7) Concurrent control: access to shared resources must be controlled to ensure that such resources are always in consistent states. A issue of distributed coordination:

a. Conventional concurrency control techniques s.a. semaphores can be used within a single computers. But they are not adequate in a DS environment.

b. Common techniques:

(a) Locks and tokens

(b) Timestamps

(8) Transparency: the ultimate challenge. Cf Lecture 1

4. System architectures

(1) **Client-server**

a. General ideas: control to individual resources is distributed among various processes (servers) across the DS. Trend: reduce the OS kernel, and move part of the kernel up to higher levels (middleware) as system services. Processes are classified as **servers** and **clients**. Each server process is in charge of a particular resource and client processes request the use of the resources through corresponding server processes. (Fig.2.2,p.34)

b. Clients compete the use of resources

c. Problems with the client-server architecture:

i. Control of individual resources is centralized in a single server. If the computer supporting the server fails, that resource becomes inaccessible. Compromise reliability of DSs.

ii. Each server is a potential bottleneck, particularly when more clients demanding the same few resources are added. Static.

iii. Multiple implementation of similar controls may be added to improve reliability and availability. Consistency then becomes an issue.

d. Intuitive and most popular

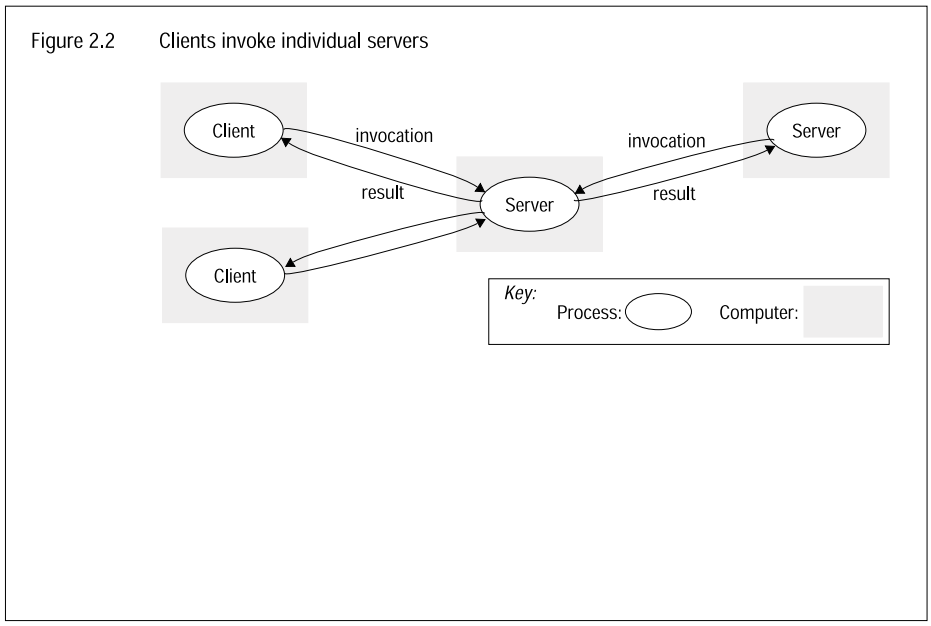


Figure 8: Clients invoke individual servers (Fig.2.2,p.34)

(2) **Peer-to-peer** (Fig.2.3, p.36)

a. Basic architecture

(a) All processes in this model play similar roles – they are peers to each other. These processes play similar roles and interact with each other. They cooperate as *peers* to perform a distributed activity/computation. There is no clear distinction of clients and servers of these peer processes.

(b) The peer processes are coded in such a way that consistency of application-level resources (such as a replicated file) is maintained and actions on these resources are synchronized if necessary.

b. Examples: *whiteboard* application: a group of programmers on several different computers view and modify an image. A peer process runs on each computer. Modification of the image by an individual programmer is communicated through middleware. Advantages: better response.

(3) Variations

a. Multiple servers (Fig.2.4, p.37)

(a) Also called *process pools*. A service is implemented by multiple server processes (possible running in separate host computers).

(b) The resources (objects) to be managed by the service could be partitioned among the servers. Alternatively, resources could be replicated to improve

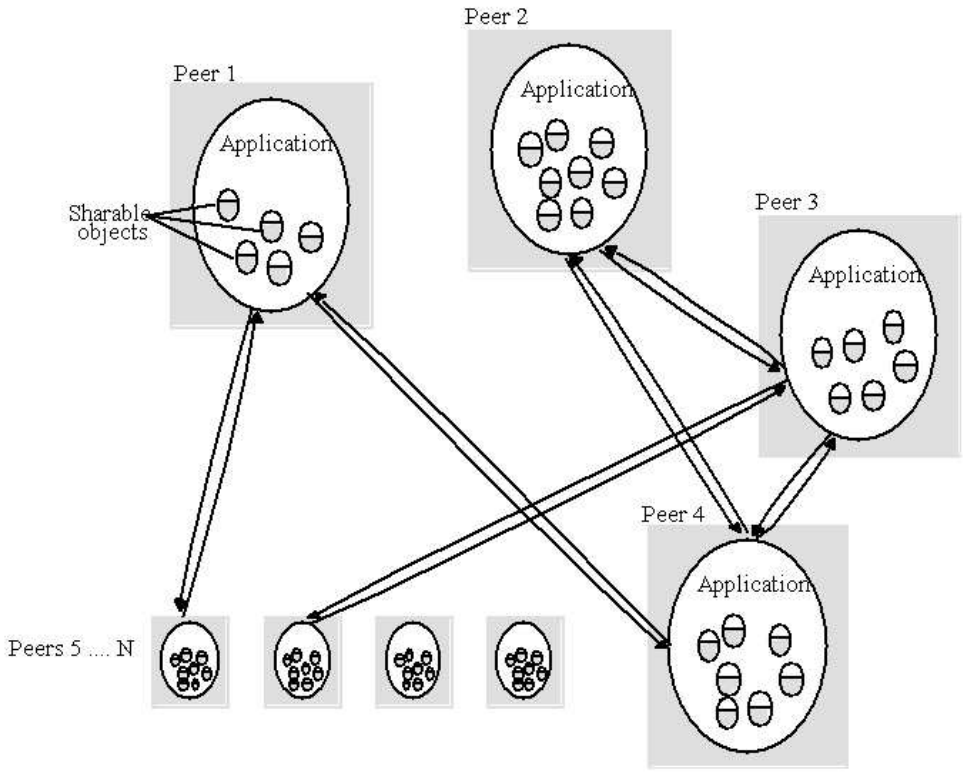


Figure 9: A distributed application based on peer-to-peer architecture (Fig.2.3, p.36)

availability and performance

b. **Proxy servers and caches** (Fig.2.5, p.38)
   (a) Cache: a store of recently referenced objects that are closer than the objects themselves.
   (b) Web proxy servers: a server that keep a shared cache to be referenced by client sites.
   (c) Proxy servers: allow access of shared cache through firewall.
   (d) Characteristics: cached data is read only to client sites.

c. Mobile code (p.17, Chapter 1, for def.)
   (a) Definition: code that can be sent from one computer to another and run at the destination computer.
   (b) Applet: a special mobile code that originally resides on a Web server. A client can select through a Web link the mobile code and download it from server and run it (Fig.2.6,p.39).
   (c) Advantage: running of the downloaded code locally by a client is much faster and smoother than running it in realtime on a server.
   (d) Potential security problems

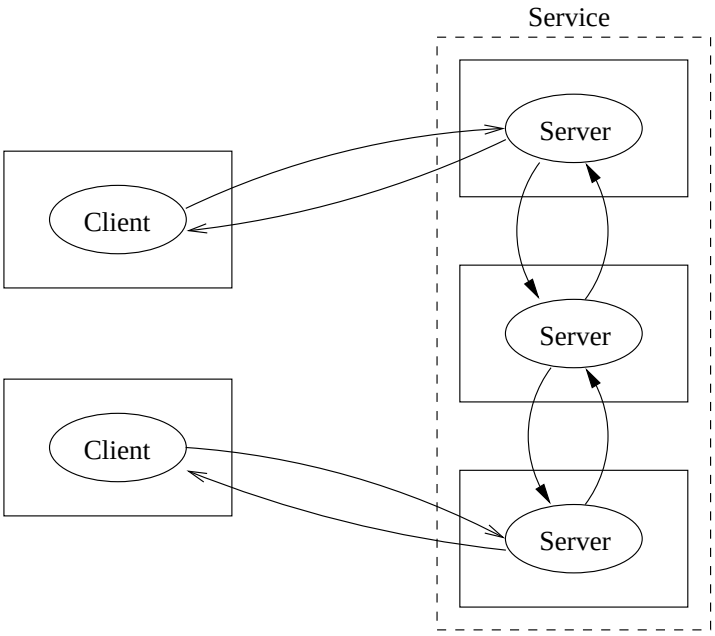


Figure 10: A service provided by multiple servers (Fig.2.4, p.37)

d. Mobile agent (p.39)

   (a) Definition: a mobile agent is a running program that can move (migrate) from one computer to another in a network while carrying task on behalf of some applications.

   (b) Examples

      · Software system maintenance

      · Information collection (user statistics in a DS)

   (c) Potential security problems (similar as mobile agent)

e. Network computers

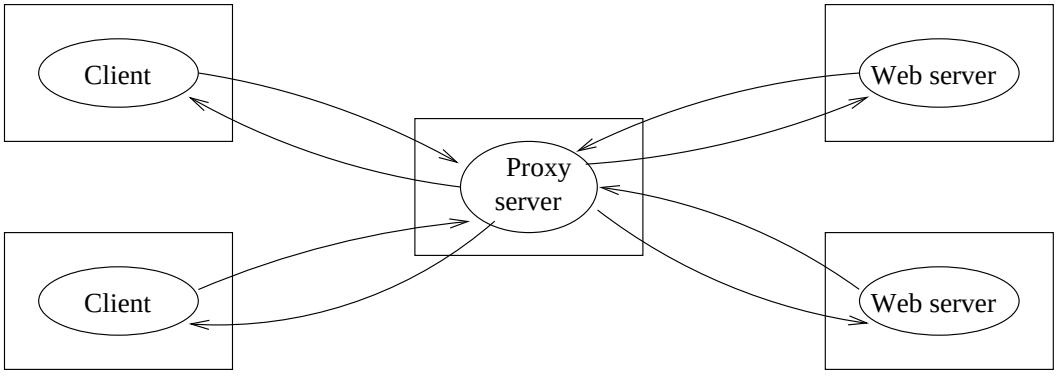
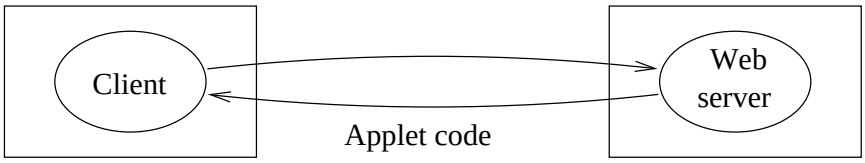


Figure 11: Web proxy server (Fig.2.5, p.38)



a) Client request results in downloading of applet code

b) Client interacts wtih applet

Figure 12: Web applets (Fig.2.6,p.39)

(a) Motivation: OS and application software require large amount of code. More importantly management and maintenance of the software require sophisticated skills.

(b) Definition: a network computer is one that does not have its own OS and any other application software. All of them have to be downloaded from remote servers. Applications are run locally while resources such as files are managed by remote servers. Users of NC only have to operate a remote controller similar to the one of a typical DVD player.

(c) Cache storage is used to reduce need of data transfer.

f. Thin clients (p.40)

(a) Definition: a thin client is a software layer that supports a window-based user interface on a remote computer (Fig.2.7, p.40)

(b) An X-Window software that runs on a window based PC. The software allows a user to interact with a remote X-window based UNIX computer on a local window based PC. The user can launch X-window based applications locally.

g. Discussions:

(a) The selection of an architecture and the distribution of responsibilities between processes and between computers is an important design issue in DS.

(b) Often a DS may adopt more than architectures to provide maximum flexibility and performance.

(4) Interfaces and objects

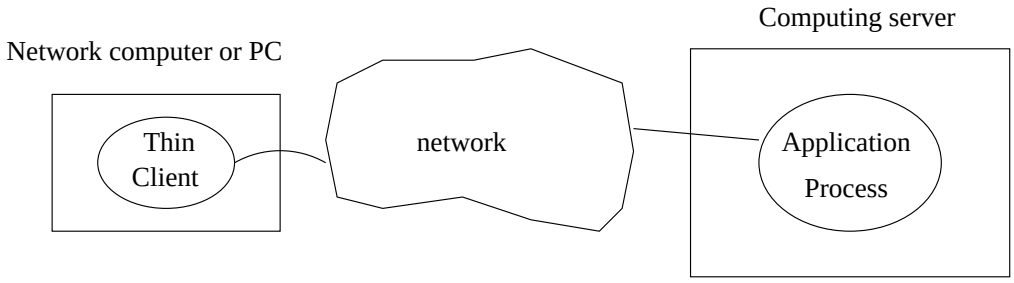


Figure 13: Thin clients and computing servers (Fig.2.7,p.40)

a. The set of functions available for invocation in a process (servers or peers) are defined in interface specification.

b. Interfaces in basic client-server architecture: fixed and static.

c. Interfaces supported by object-oriented model: dynamic. The number and types of objects hosted by each process may vary. The location of objects may also migrate (location transparency).

(5) Design requirements of distributed architectures

a. Performance

(a) Distribution of resources in DSs creates challenges of managing concurrent updates of distributed resources.

(b) Performance of a DS is affected by communication capacities, processing capabilities, concurrency, and failures:

· Responsiveness. Hindered by concurrency, communications, and failures. Example: Web clients browsing a Web site.

· Throughput. The amount of computation that can be completed in certain units of time. In addition to other facts listed above, it can also be hindered by processing speed of individual computers.

· Load balancing. To eliminate bottleneck. May involve migration of computation activities.

b. Quality of service (QoS)

(a) Main quality properties: performance, reliability, security, and adaptability.

(b) Adaptability: change the system configuration and resource allocation to meet the changing system environment. Eg. changing the location of a database used by a Web client in response to a change of access pattern to that database.

c. Use of caching and replication

(a) Motivations: to improve performance and reliability

(b) Web-cache protocol.
- Web browser and proxy servers cache most referenced contents.
- A browser and proxy server can validate a cached response by checking *freshness* of cached data.
- An expiry time may be associated to each cached piece of data.

(c) Caching and replication may hinder performance if not properly managed or mis-used (cf. section for Web proxy server).

d. Dependability issue

(a) Dependability: correctness, security, and fault tolerance.

(b) Fault tolerance. A DS application should continue to function correctly in the presence of faults in hardware, software, and networks (cf. section for failure handling and recovery).
- Redundancy is the primary technique. Redundancy can be at different levels.
- Architecture level: Multiple hardware components (e.g. computers and disk drives)
- Communication protocol level.

(c) Security: cf. Section 2 of this lecture

5. Fundamental models

(1) Fundamental properties of DSs: distributed processes that communicate with one another by sending messages over a network.

a. Models based on these fundamental properties allow closer examination of characteristics of issues.

b. A system model is a paradigm regarding the way a certain aspect of the system interacts and behaves.

c. Questions to be addressed in a system model:

(a) Main entities in the system

(b) Patterns of interactions of the entities

(c) Characteristics that affect their behaviors

(2) Interaction model

a. *Distributed algorithm* and *distributed processes*

(a) A distributed process is a member of a set of processes running in a DS that cooperate/interact to complete a specific task/computation. A distributed process only has partial info about the overall state and progress of the whole computation.

(b) Distributed algorithms determine the steps taken by a distributed process, including transmission of messages and access/update of remote resources.

b. Interaction model defines/captures the methods in which distributed processes interact/cooperate with one another.

c. Two major factors that affect process interactions:
   (a) Performance of communication channels
   (b) Lack of accurate global time (and state)

d. Two variants of interaction models:
   (a) Synchronous DS.
      i. A DS is a *synchronous* DS if (p.50):
         - The time to execute each step of a process has known lower and upper bound.
         - Each message transmitted over a comm. channel is received within a known bounded time.
         - Each process has a local clock whose drift rate from real time has known bound.
      ii. Obtaining realistically accurate values for these three bounds is practically very difficult. DSs designed based on inaccurate bounds are not reliable.
      iii. Modeling a distributed algorithm as a synchronous system can provide useful info as how the algorithm may behave under real DSs.
   (b) Asynchronous DS
      i. A DS is an *asynchronous* DS if (p.51):
         - The time to execute each step of a process can vary and may take arbitrarily long. There is no known upper bound.
         - Each message transmitted over a comm. channel may experience arbitrarily long delays. There is no known upper bound on a transmission.
         - Each process has a local clock whose drift rate from real time can be arbitrary.
      ii. The asynchronous model makes no assumptions about upper bounds of execution speeds, message transmission delays, and clock drift amount. Well suitable for Internet.
      iii. Any distributed algo that is valid under an asynchronous model is also valid under any synchronous model. The reverse does not hold.
      iv. Asynchronous model is easier to implement than the synchronous model. A DS whose interaction model is asynchronous can/have to use timeout to limit the effects of arbitrary delays.

e. Event ordering in DSs

(a) Knowing the execution order of two events from two cooperating processes is useful in many cases. For example, a process P_1 has requested updating a file managed by a process P_2. We want to know transactions.

T_1	T_2
...	...
$x := x + 1;$	$y := x + z$
...	...
commit;	commit;

The copy of value modified by T_1 is termed *dirty* before T_1 commits. We are interested in knowing whether T_2 performed the operation $y := x + z$ before or after T_1 commits.

(b) Again, due to nature of distributedness, events that occur physically in sequence may not be observed by individual users/processes in the same sequence as they occur.

i. Example (p.52,53) An email message and two of responses. They may arrive in a physical mailbox out of ordering.

ii. If physical clocks can be accurately synchronized, then physical timestamp can be used to order the email messages.

(c) Logical clocks (to be covered in next chapter) are an alternative.

i. Maintaining accurate physical time is difficult and expensive (the more accurate, the more expensive). No algorithm can guarantee 100% accuracy.

ii. Some times accurate physical occurrence time of events is not very important. What are important are their relative ordering: which one *happens before* before which one.

(3) Failure model

a. A failure model defines methods by which failures may occur in a DS. It provides guidance as how to detect, solve, and prevent failures

b. *Omission failures*: a process or comm. channel fails to perform actions that are supposed to be done

(a) *Process omission failures*:

i. Main type of process omission failures: *process crash*

ii. Process crash that can be detected certainly by other processes is called *fail-stop*.

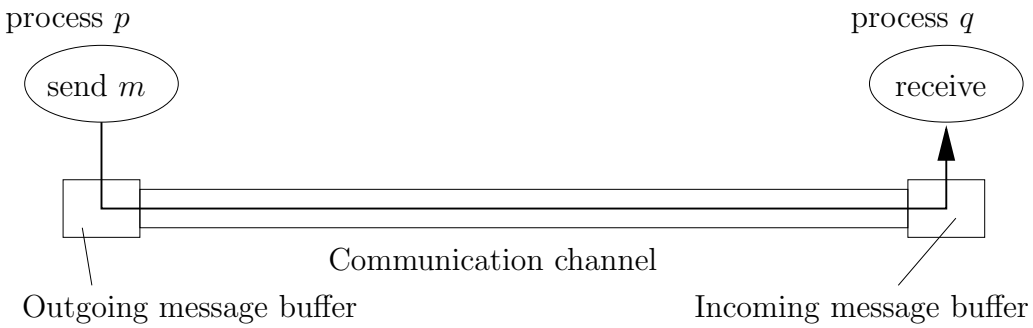


Figure 14: Processes and channels (Fig.2.9, p.55)

(b) *Communication omission failures*:

i. A COF occurs if the comm. channel does not transport a message from a sender's outgoing buffer to the receiver's incoming buffer (Fig.2.9,p.55).

ii. *send-omission failure*: loss of message between the sending process and the outgoing message buffer.

iii. *receive-omission failure*: loss of message between the incoming message buffer and the receiving process.

iv. *channel-omission failure*: loss of message between the two buffers.

c. Arbitrary failures: the worst possible failures. It can be any type and can happen any where at any time.

(a) Both processes and comm. channels can have arbitrary failures.

(b) A process may omit intended processing steps or take unwarranted steps any time and any where.

(c) A comm. channel may corrupt a message or generate a duplicate copy of a message.

d. Summary of omission and arbitrary failures: Fig.2.10 (p.56)

Class of failures	*Affects*	*Description*
Fail-stop	Process	Process halts and remains so. Other processes may detect this state.
Crash	Process	Process halts and remains so. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message
Send-omission	Process	A process completes a *send*, but the message is not put its out going 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 behavior: it may send/transmit arbitrary messages at arbitrary times; commit omissions; a process may stop or take an incorrect step.

Figure 2.10 Omission and arbitrary failures

e. *Timing failures*. They only apply to synchronous DSs. A timing failure occurs when a time limit set on process execution time, message delivery, or clock drift rate is exceeded. A timing failure may also cause a performance failure. Fig.2.11 (p.57) summarizes timing failures.

Class of Failures	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between steps.
Performance	Channel	A message's transmission takes longer than the set timeout period

Figure 2.11 Timing failures

Benign failures: omission failures, timing failures, and performance failures.

f. *Masking failures*.

(a) It is possible to provide reliable services even in the presence of failures.

(b) A failure is *masked* by a service if the service can hide it altogether by converting it into a more acceptable type of failures. Examples:

- Checksums can be used to convert an arbitrary failure into an omission failure.
- Omission failures could be masked through use of retransmission (plus timeout).

g. Reliability of one-to-one comm. A *reliable communication* is defined in terms of *validity* and *integrity*:

(a) Validity: any message (in an outgoing buffer) is eventually delivered to the corresponding incoming buffer.

(b) Integrity: the message received is identical to the one sent and no duplicate messages delivered.

The communication in a DS is termed reliable if validity and integrity can be ensured under *all possible* communication problems.

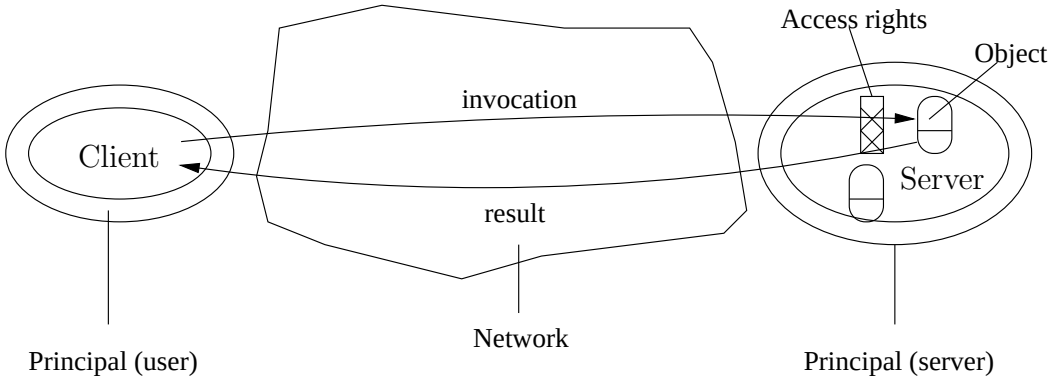


Figure 15: Objects and principals (Fig.2.12, p.58)

(4) Security model

a. Security model: methods to

(a) secure distributed processes and communication channels; and

(b) protect objects accessed by these processes.

b. Protecting objects (Fig.2.12,p.58)

(a) *Access rights*: each object is associated with a list of access rights – who is allowed and what is allowed for each operation (method).

(b) *Principal*: a principal is an *authority* who issues an access request/invocation (from a client) or result (from a server). A principal can be a user or a process (service).

(c) A server is responsible for verifying the identity of the principal behind each invocation and checking that each such invocation has sufficient access rights.

c. Securing processes and their interactions against threats. Interactive processes have to communicate messages with one another.

(a) Traveling messages are subject to attacks.

(b) Processes exported interfaces for supported services, hence make themselves subject to external threats too.

Therefore both messages contents and processes must be secure.

d. Security threats

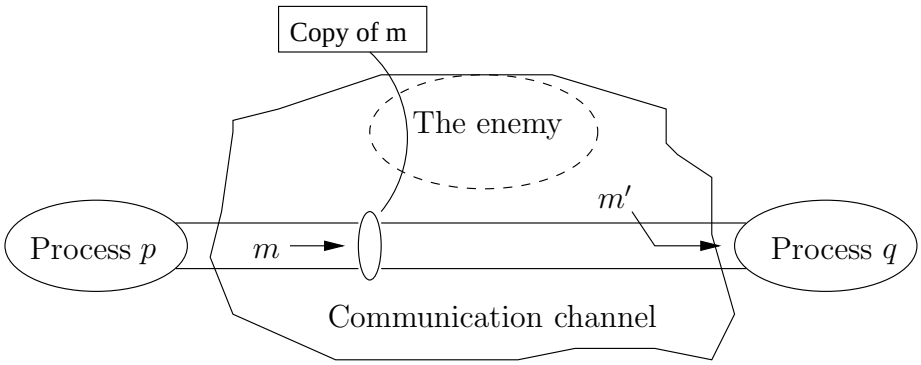


Figure 16: The enemy (Fig.2.13, p.59)

(a) *The enemy*. Under the proposed security model, an enemy is an object that is capable of sending any messages to any process and reading or copying any messages between a pair of processes (Fig.2.13, p.59)

(b) *threats to processes*.

i. A process may receive messages from another process whose identity may not be determined by the former. Eg.: a process P may intercept the IP address of a process P_1 which is communicating with a third process P_2. P then may use P_1's IP address to communicate with P_2.

ii. The described problem is caused by lack of knowledge of the source of a message by both client and server processes:

- For a server, it may serve many clients. It's difficult for it to check the identity of the principal behind a particular request message. As explained in above example, Even if a principal's identity is required, an enemy could still fake such an identity.
- Similarly, a client may have difficulty to determine the identity of principal of a reply message.

(c) *threats to communication channels*.

i. An enemy may copy, alter, or inject false messages onto a comm. channel.

ii. An enemy can also copy and save of a message, then send that message at a later time (e.g. making multiple deposits into a banking account).

e. Defeating security threats – security channels

(a) Processes can be protected and communication channel threats can be eliminated if comm. channels are made *secure*, i.e. not subject to all threats.

(b) *cryptography* and *encryption*:

· *encryption*: the process of scrambling a message in a way to hide its contents.

· *cryptography*: the science of keeping messages secure. Cryptography is based on encryption algorithms that use secret keys, which are difficult to guess and usually a large number obtained by using mathematical theory.

(c) *Authentication* of messages: verify the the authenticity of messages by including in a message an encrypted portion that contains enough info to guarantee its authenticity.

Example: in a message requesting read permission of a file, the identity of the file, the date and time of request, the principal's identity, may be encrypted with a secret key shared between the file server and the requesting process.

(d) Security channels. A security channel is a comm. channel connecting a pair of processes, each of which acts on behalf of a principal (Fig.2.14,p.60). A security channel has the following properties:

i. Each process knows reliably the identity of the principal on whose behalf the other process is executing (client knows it is requesting/invoking the the identity of the principal of the server that will server its request; the server knows the identity of the principal behind the invocations and can check their access rights before operations are granted).

ii. A security channel ensures privacy and integrity (protecting against tampering) of the data transmitted across it (encryption is used).

iii. Each message includes a time stamp (physical or logical) to prevent messages from being replayed.

f. Other possible threats - *denial of service attacks*.

(a) *Deniel of service*.

· Making a service out of service by exploring software bugs in a server's code.

· Congesting comm. links to a server site or exhausting a server site's computation power by making excessive requests.

(b) *Mobile code*. Emails that carry attachment are a good example of potential threats of mobile code. There are many other methods that are used to

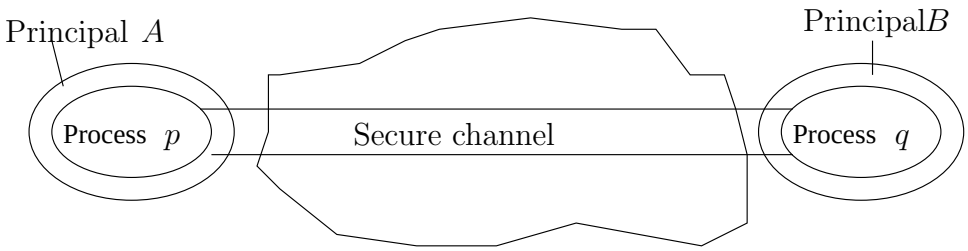


Figure 17: Secure channels (Fig.2.14, p.60)

explore mobile code. Example: in a UNIX based computer, explore software bug in the code of a process with root privilege.

(5) Use of security models

a. The discussed security model can be used as the basis in dealing with the security aspect of a secure DS.

b. Secure channels have to be used against security threats.

c. Employment and management of security channels may incur substantial operation overhead.

d. Due to their enormous complexity, security threats in DSs may emerge from many sources in systems network environment, physical environment, and human environment.

e. Many different security models may be available in design of a DS.

   (a) During the design phase, *threat models* are constructed to list all possible threats the their consequences.

   (b) Strategies and techniques to prevent each such threats will be evaluated in terms of their effectiveness and cost.

   (c) Example: in a UNIX system, do we use classical password protection scheme for user login authentication, or do we use *kerberos*?