TRAO ĐỔI THÔNG TIN TRONG HỆ PHÂN TÁN

(Giáo trình này được xây dựng dựa trên giáo trình của PGS. TS. Hà Quốc Trung)

TS. Trần Hải Anh

Nội dung

1. Trao đổi thông tin giữa các tiến trình
2. Lời gọi thủ tục từ xa
3. Trao đổi thông tin hướng thông điệp
4. Trao đổi thông tin dòng

1. Trao đổi thông tin giữa các tiến trình

1.1. Các giao thức phân tầng
1.2. Trao đổi thông tin bằng UDP
1.3. Trao đổi thông tin bằng TCP
1.4. Một số vấn đề cần thảo luận

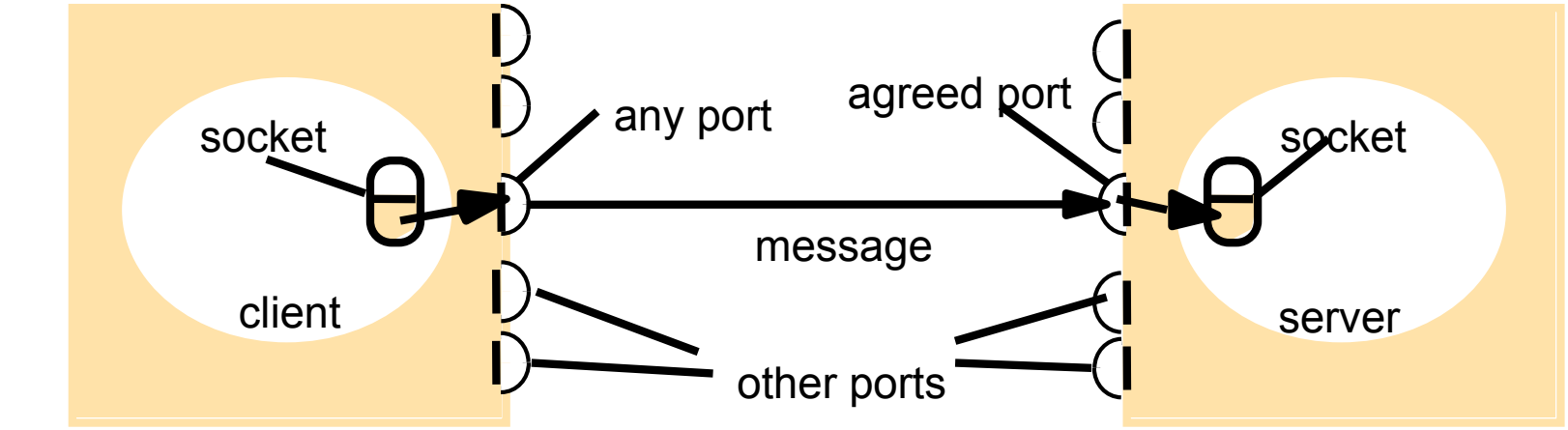
I. Khái niệm

- Giao thức
  - Cấu trúc thông điệp
  - Kích cỡ thông điệp
  - Thứ tự gửi thông điệp
  - Cơ chế phát hiện thông điệp hỏng hay bị mất
  - V.v…
- Phân tầng
- Các loại giao thức
  - Hướng kết nối, không hướng kết nối, tin cậy, không tin cậy
- Các vấn đề của giao thức
  - Send, receive primitives
  - Đồng bộ/không đồng bộ, dừng, không dừng

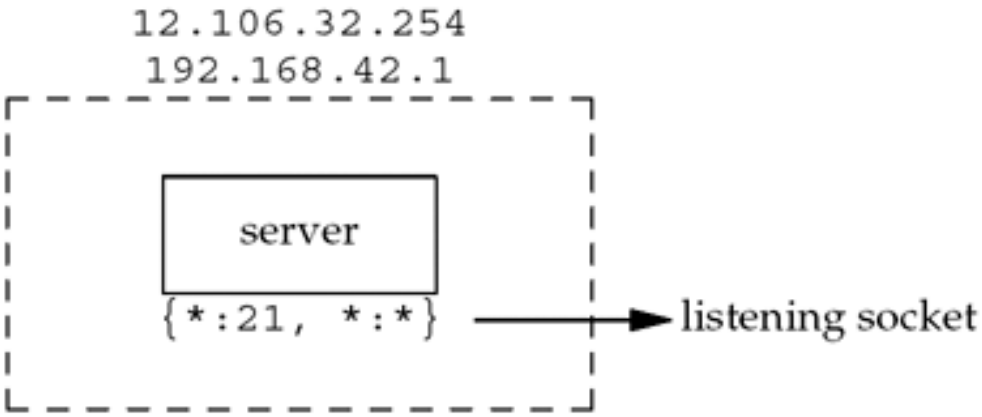
Đặc tính của giao tiếp giữa các tiến trình

- Giao tiếp đồng bộ và bất đồng bộ
- Đích đến của thông điệp
- Độ tin cậy
- Thứ tự

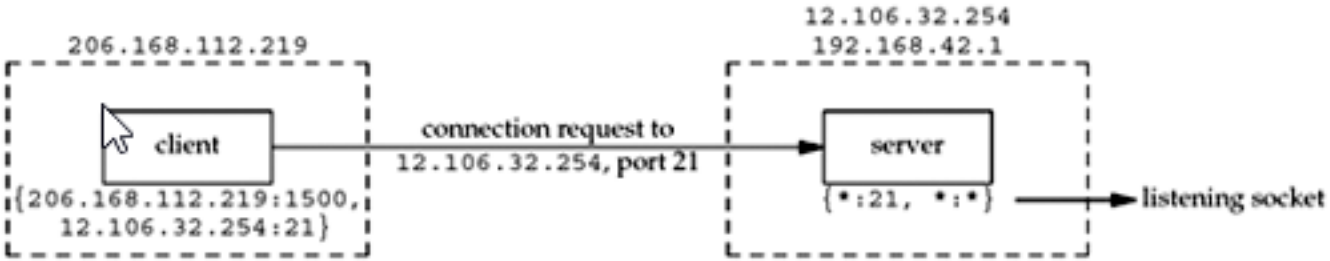
Socket-port


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

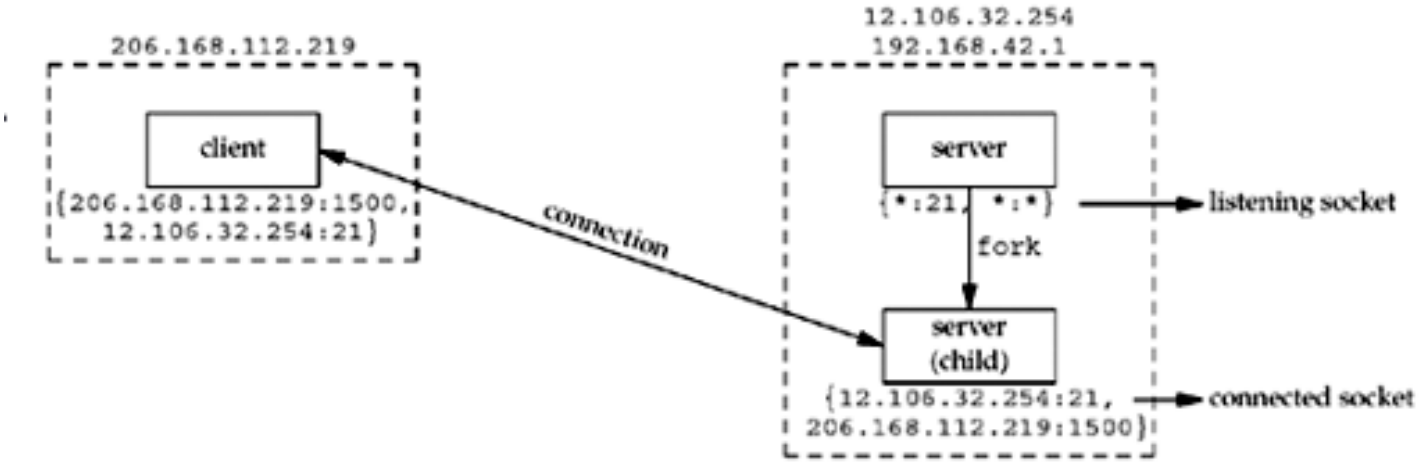
TCP Port Numbers and Concurrent Servers (1)



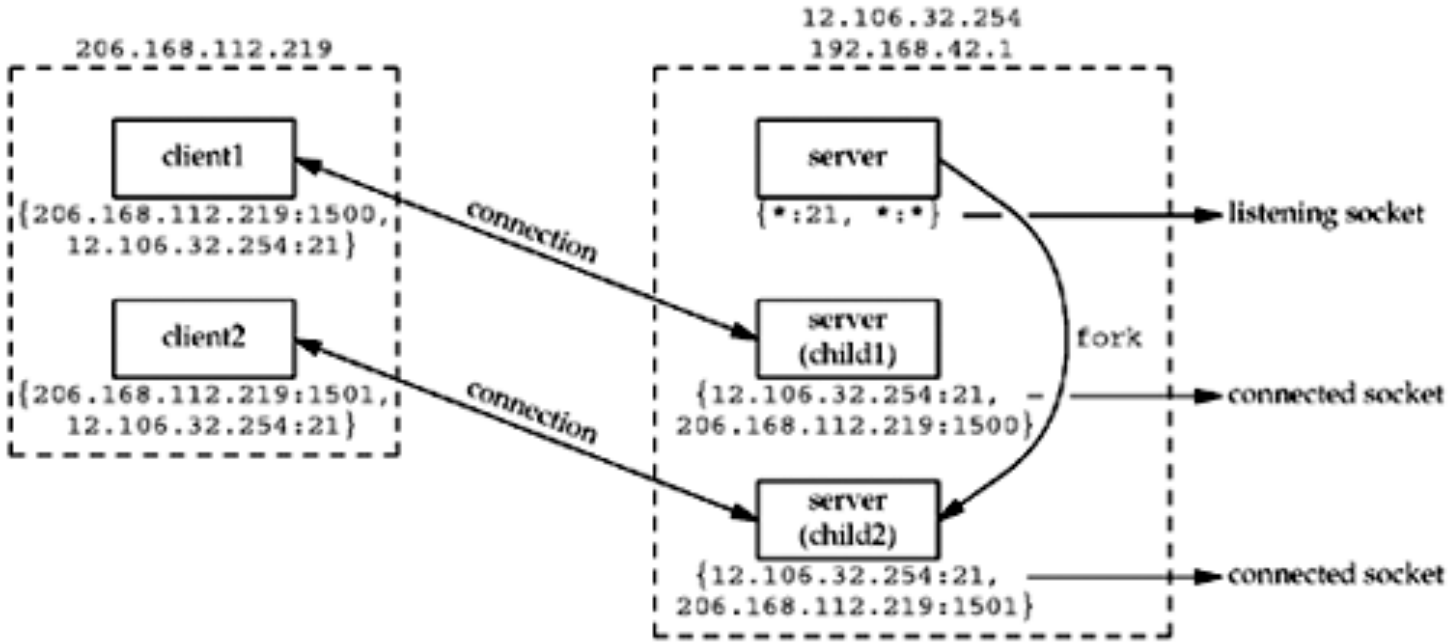
TCP Port Numbers and Concurrent Servers (2)



TCP Port Numbers and Concurrent Servers (3)



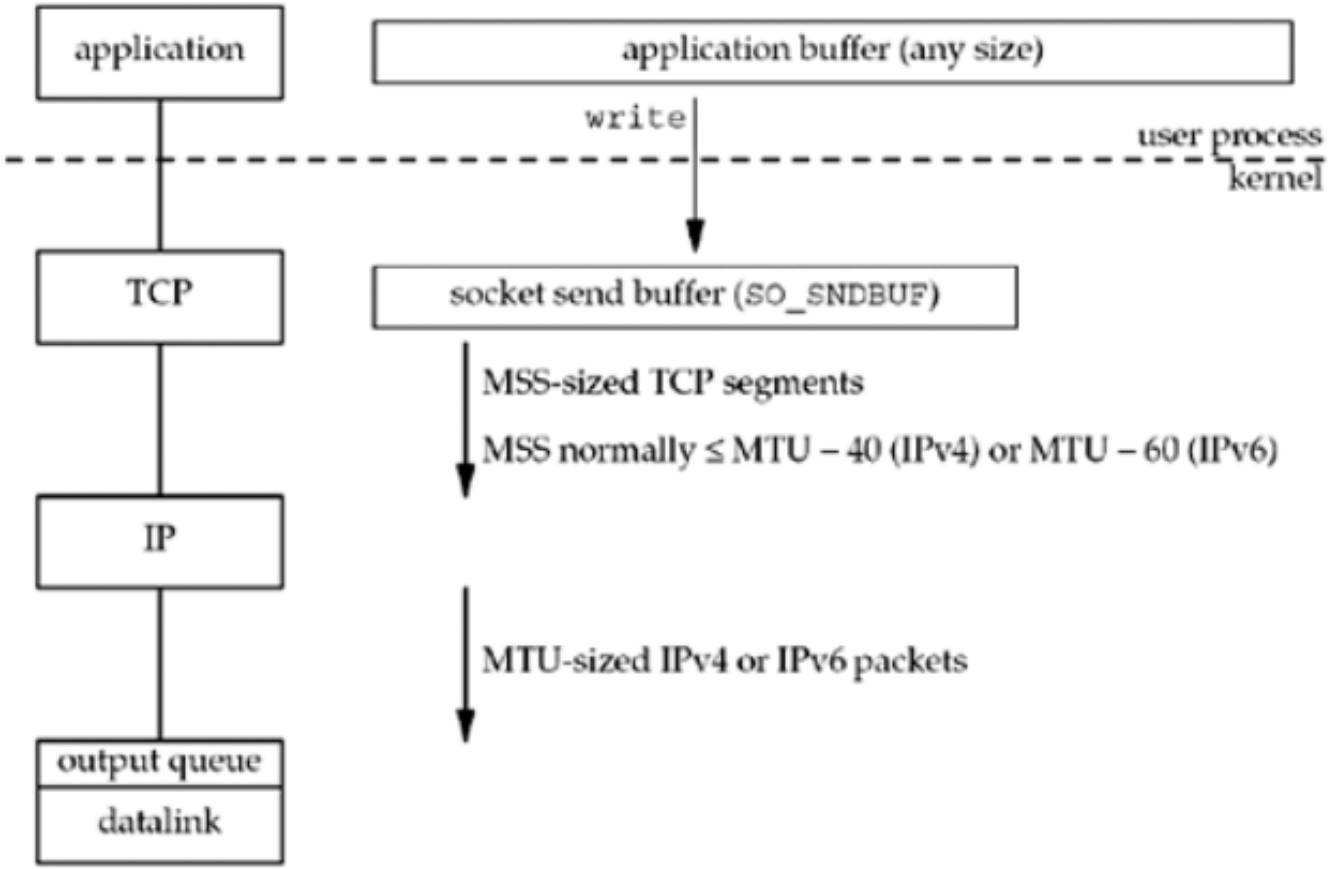
TCP Port Numbers and Concurrent Servers (4)



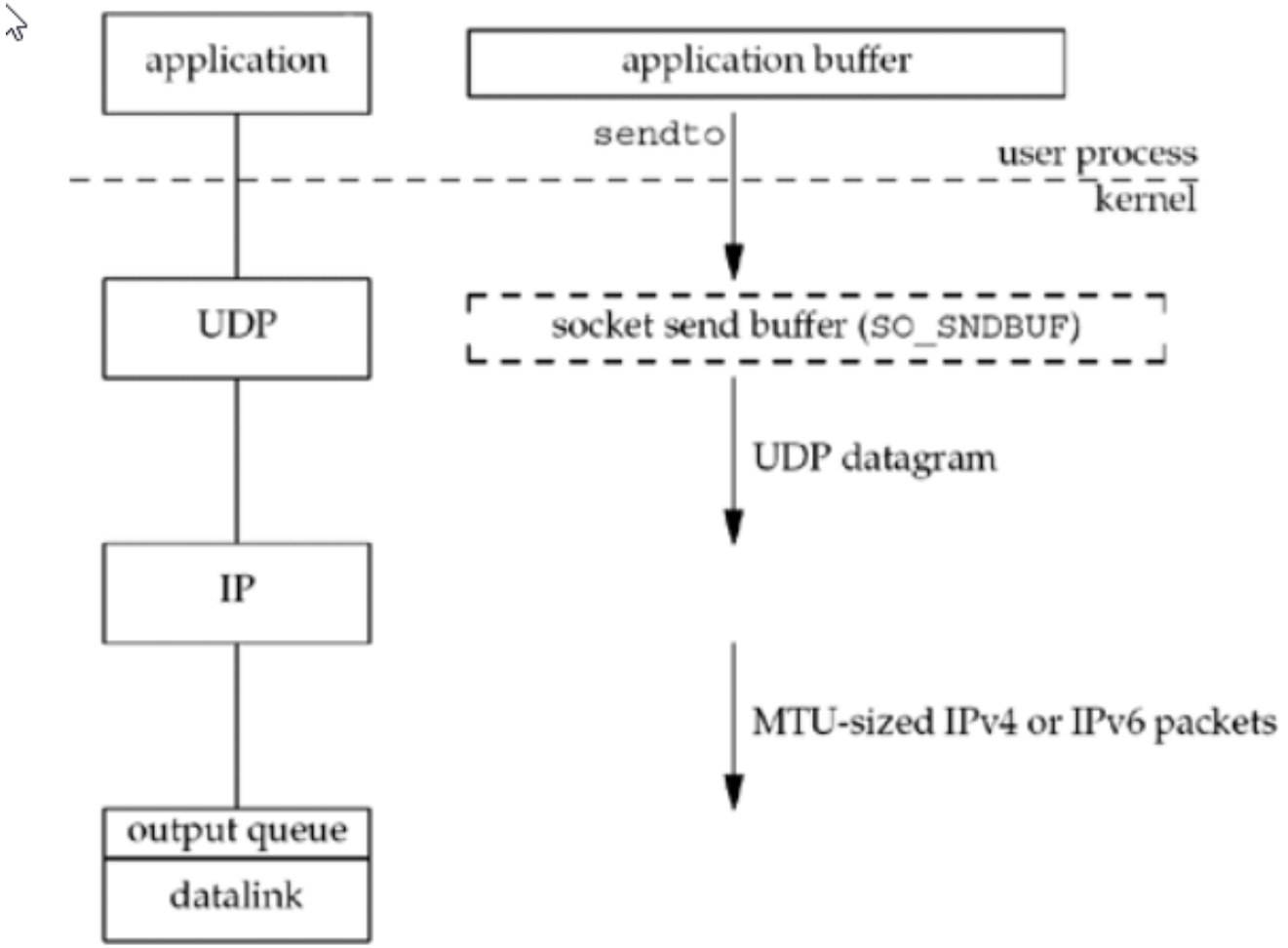
Buffer Sizes and Limitations

- Maximum size of an IPv4 datagram: 65,538 bytes
- MTU (Maximum transmission unit)
- Fragmentation when the size of the datagrram exceeds the link MTU.
  - DF bit (don't fragment)
- MSS (maximum segment size): that announces to the peer TCP the maximum amount of TCP data that the peer can send per segment.
- MSS = MTU – fixed size of headers of IP and TCP

TCP output


application
application buffer (any size)
write
user process
kernel
TCP
socket send buffer (SO_SNDBUF)
MSS-sized TCP segments
MSS normally ≤ MTU – 40 (IPv4) or MTU – 60 (IPv6)
IP
MTU-sized IPv4 or IPv6 packets
output queue
datalink

UDP output


application
application buffer
sendto
user process
kernel
UDP
socket send buffer (SO_SNDBUF)
UDP datagram
IP
MTU-sized IPv4 or IPv6 packets
output queue
datalink

Hỗ trợ của Java

- Class InetAddress:
- Working with IP address and domain name
- ```
  InetAddress aComputer =
  InetAddress.getByName("bruno.dcs.qmul.ac.
  uk");
  ```
  ```

1.2. Trao đổi thông tin bằng UDP

- Đặc điểm:
  - Không hướng kết nối
  - Không tin cậy
  - Không đồng bộ
- Vấn đề:
  - Kích cỡ thông điệp
  - Blocking (*send* không dừng; *receive* bị dừng)
  - Timeouts
  - Receive from any

```
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();}
   }
}
```

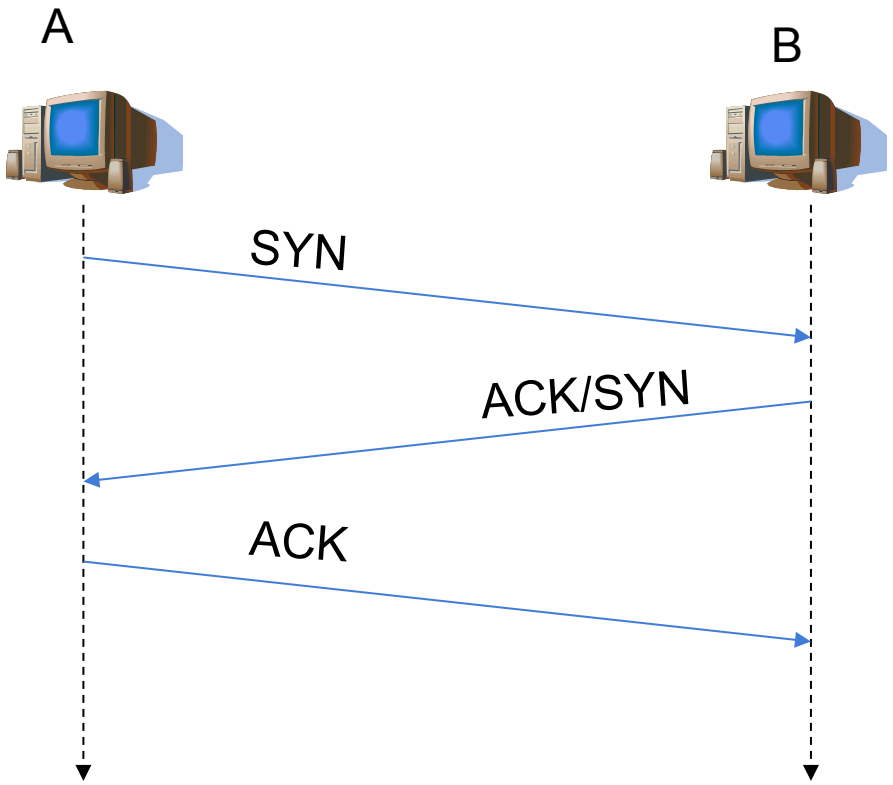
```
import java.net.*;
import java.io.*;
public class UDPClient{
    public static void main(String args[]){
            // args give message contents and server hostname
            DatagramSocket aSocket = null;
              try {
                        aSocket = new DatagramSocket();
                        byte [] m = args[0].getBytes();
                        InetAddress aHost = InetAddress.getByName(args[1]);
                        int serverPort = 6789;
                        DatagramPacket request = new DatagramPacket(m,  m.length, aHost,
serverPort);
                        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();}
   }
}
```

1.3. Trao đổi thông tin bằng TCP-IP

Thiết lập liên kết TCP :

Giao thức bắt tay 3 bước

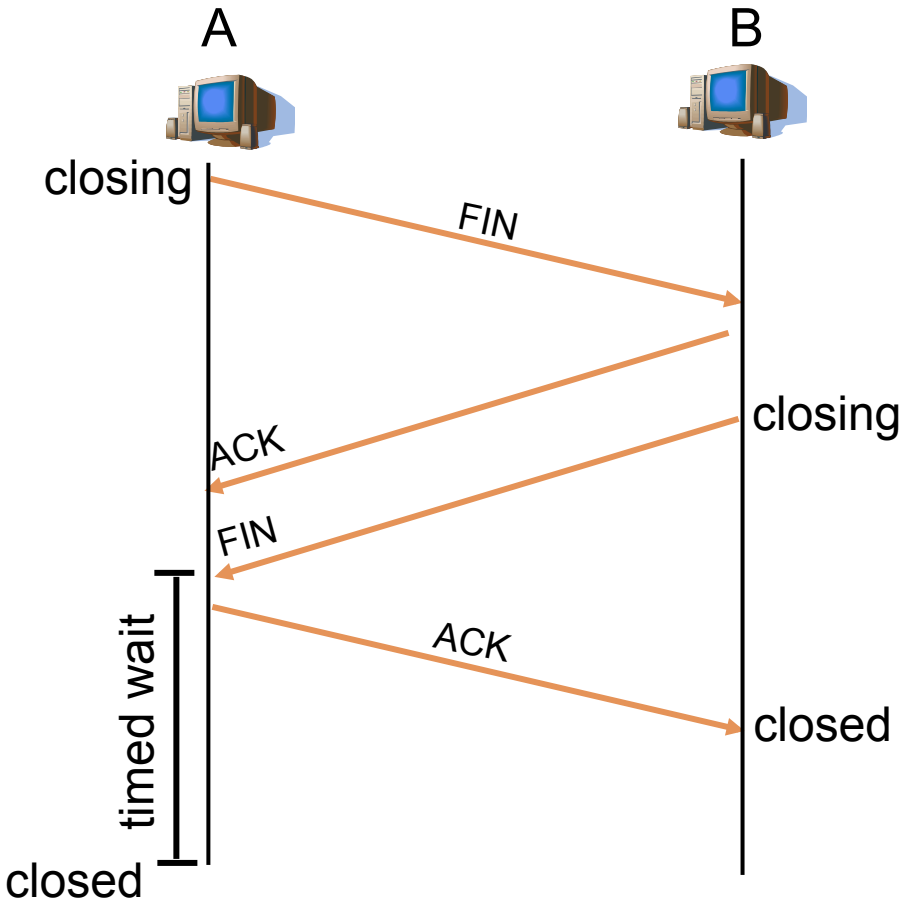


- Bước 1: A gửi SYN cho B
  - chỉ ra giá trị khởi tạo seq # của A
  - không có dữ liệu
- Bước 2: B nhận SYN, trả lời bằng SYNACK
  - B khởi tạo vùng đệm
  - chỉ ra giá trị khởi tạo seq. # của B
- Bước 3: A nhận SYNACK, trả lời ACK, có thể kèm theo dữ liệu

Ví dụ về việc đóng liên kết

- Bước 1: Gửi FIN cho B
- Bước 2: B nhận được FIN, trả lời ACK, đồng thời đóng liên kết và gửi FIN.
- Bước 3: A nhận FIN, trả lời ACK, vào trạng thái "chờ".
- Bước 4: B nhận ACK. đóng liên kết.

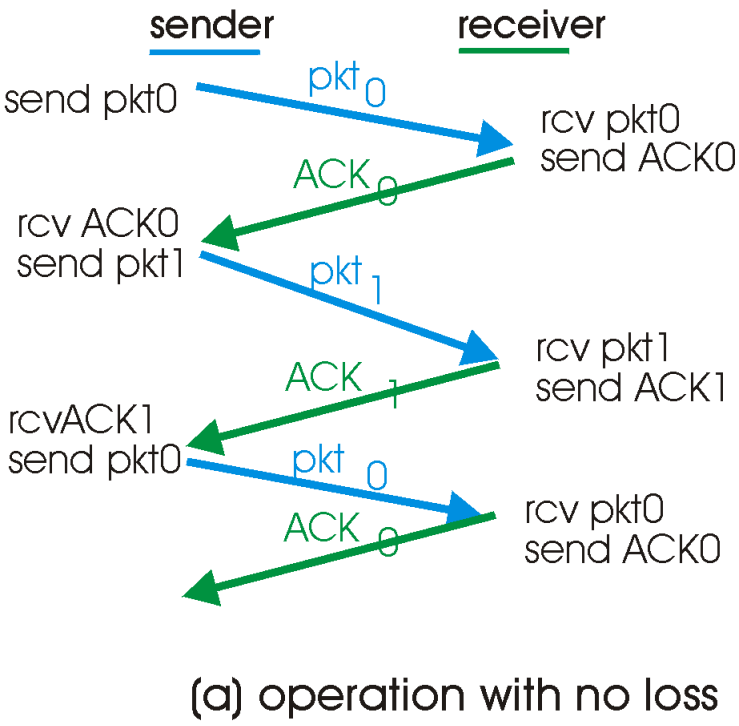
Lưu ý: Cả hai bên đều có thể chủ động đóng liên kết



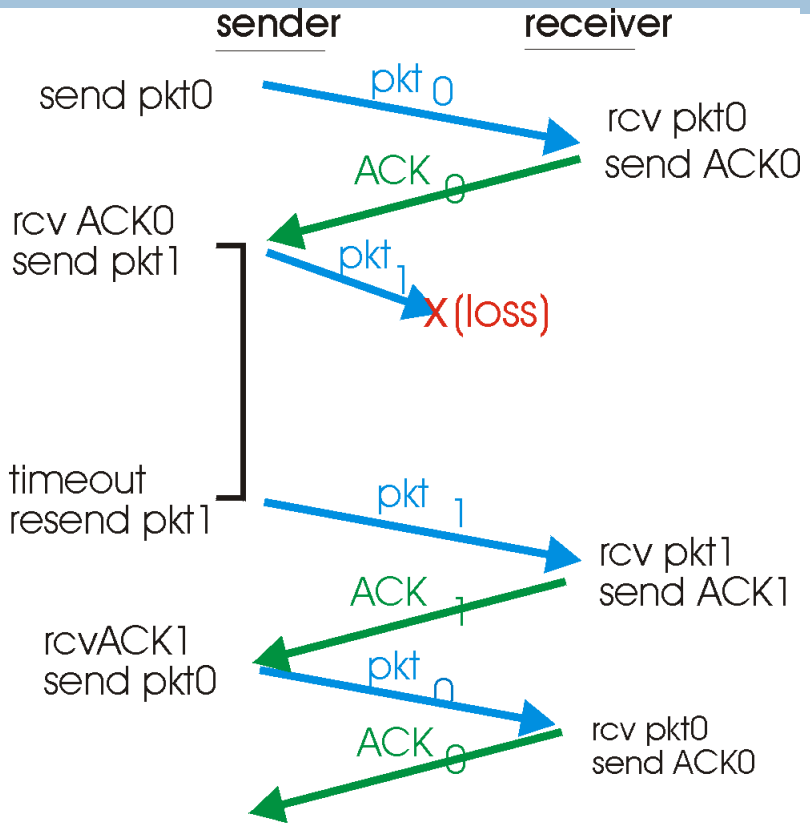
Trao đổi thông tin bằng TCP

- Đồng bộ dữ liệu (data type matching)
- Dừng (cả thao tác gửi và nhận đều là các thao tác dừng)
- Các luồng
- Mức độ tin cậy
  - Thành công=> chắc chắn
  - Chưa báo thành công=> ???
  - Không đảm bảo thông báo đến đích

Một số trường hợp xảy ra

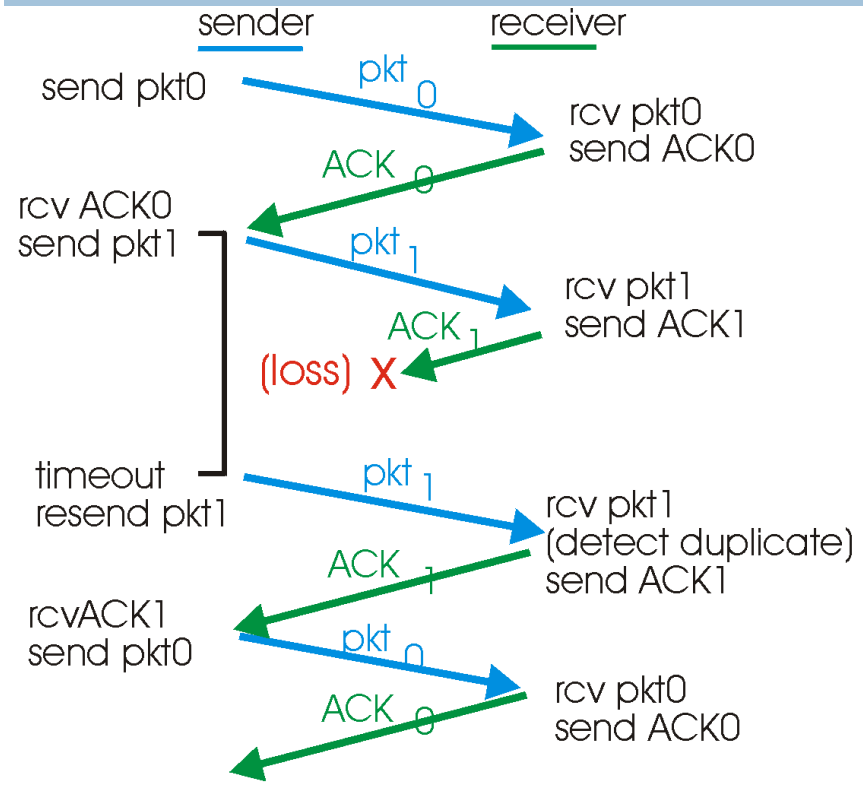


(a) operation with no loss

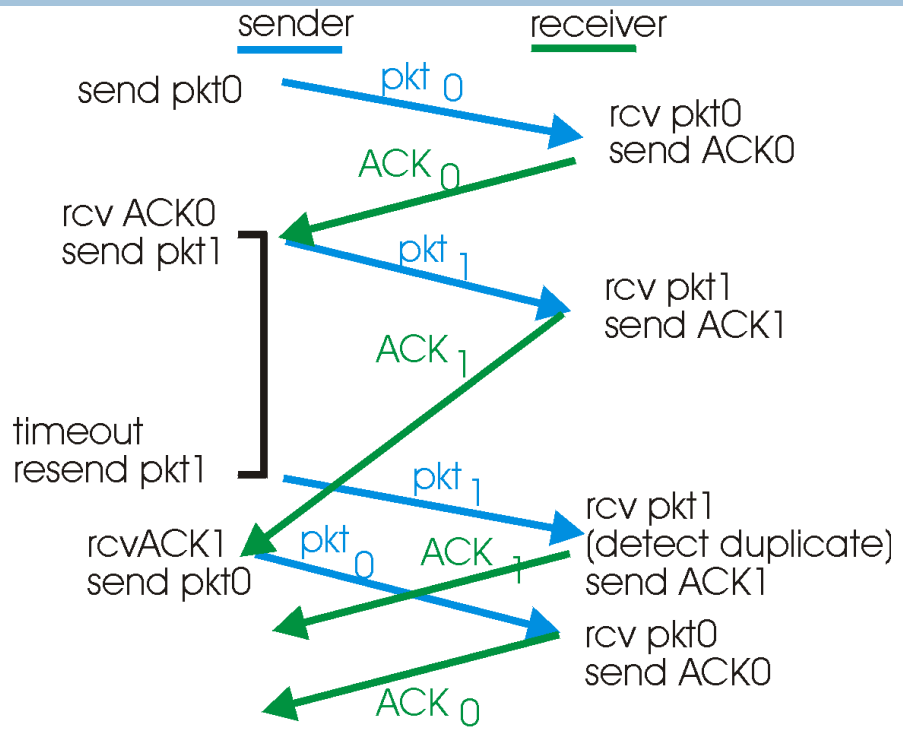


(b) lost packet

Một số trường hợp xảy ra



(c) lost ACK



(d) premature timeout

```
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());}}}
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*/}}}}
```

```
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
                    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());}}
        }
}
```

Các vấn đề của trao đổi thông tin giữa các tiến trình

- Trao đổi bền vững
- Trao đổi tạm thời
- Trao đổi đồng bộ
- Trao đổi không đồng bộ
- Gửi nhận dừng, không dừng
- Tin cậy/không tin cậy
- Trật tự của các thông báo

Nội dung

1. Trao đổi thông tin giữa các tiến trình
2. Lời gọi thủ tục từ xa
3. Trao đổi thông tin hướng thông báo
4. Trao đổi thông tin dòng

2. Lời gọi thủ tục từ xa

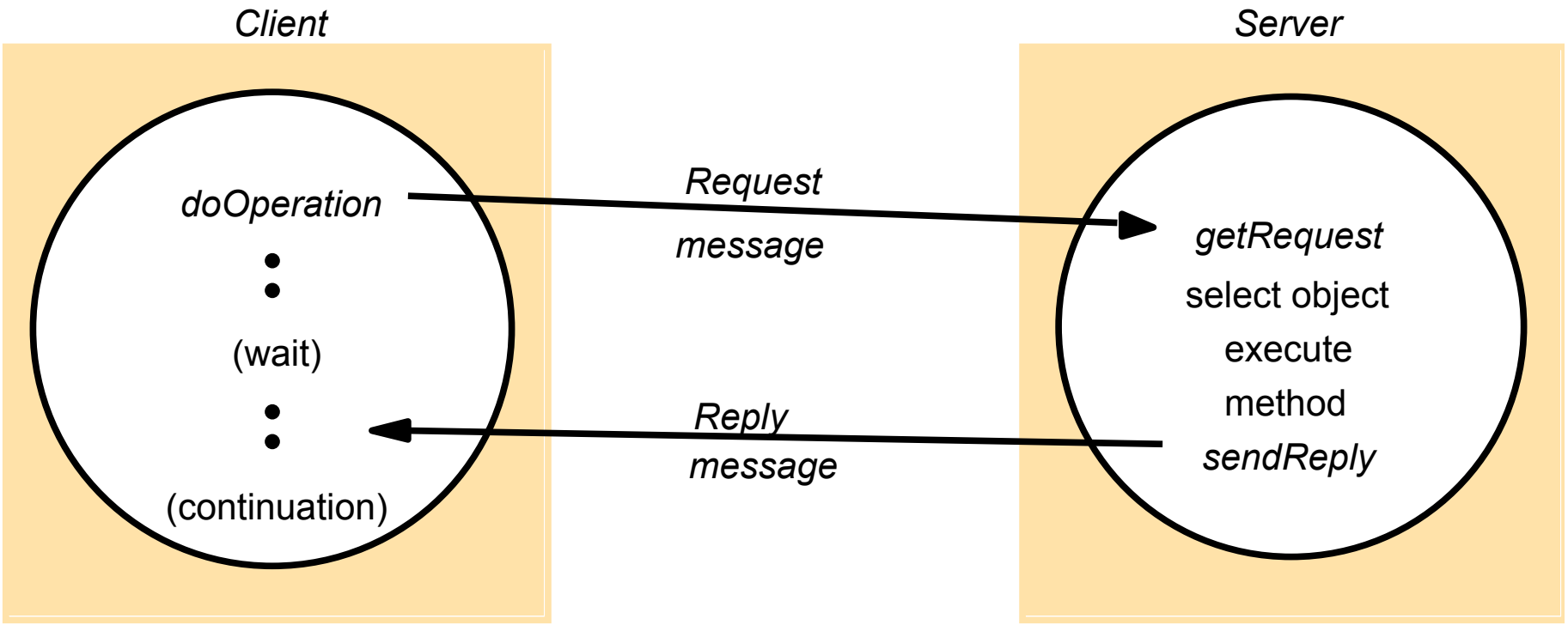
2.1. Giao thức yêu cầu-trả lời
2.2. RPC-Cơ chế lời gọi thủ tục từ xa
2.3. SUN-RPC và DCE-RPC
2.4. RMI

2.1. Giao thức yêu cầu-trả lời

- Là cơ chế bậc cao hơn truyền thông điệp, cho phép trao đổi thông tin giữa 2 tiến trình bằng 2 thông báo gửi nhận liên tiếp
- Hỗ trợ các lời gọi từ xa
- Đồng bộ
- Tin cậy

Yêu cầu-trả lời

- Đặc điểm:
  - Không cần báo nhận
  - Không cần kiểm soát luồng



Thủ tục

- *public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)*
- *public byte[] getRequest ();*
- *public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);*

Đơn vị dữ liệu

messageType	*int (0=Request, 1= Reply)*
requestId	*int*
remoteReference	*RemoteRef*
operationId	*int or Operation*
arguments	*array of bytes*

Các vấn đề thiết kế

- Kích thước thông điệp
- Định danh thông điệp
- Mô hình lỗi
  - Nếu UDP=> bỏ qua, sai thứ tự
  - Nếu TCP=> lỗi dừng
  - Hệ thống phát hiện được các lỗi trên, thông thường bằng timeouts
- Gửi lại=> có khả năng có thông báo lặp=> phụ thuộc vào định danh của thông báo và tính chất của dịch vụ (idempotent, not idempotent)

HTTP: 1 vd của giao thức yêu cầu-trả lời

HTTP *request* message

method	*URL or pathname*	*HTTP version*	*headers*	*message body*
GET	//www.dcs.qmw.ac.uk/index.html	HTTP/ 1.1		

HTTP *reply* message

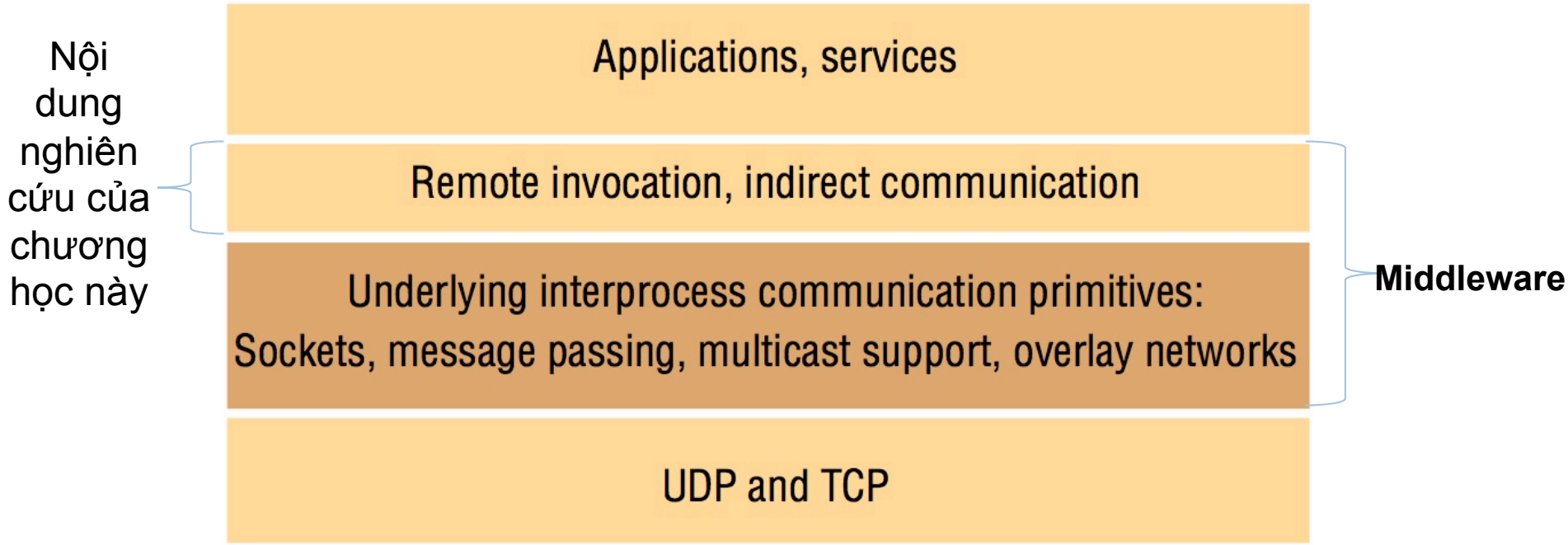
HTTP version	*status code*	*reason*	*headers*	*message body*
HTTP/1.1	200	OK		resource data

3 kiểu giao thức trao đổi

- R only protocol
- RR protocol
- RRA protocol
- Cài đặt trên TCP=> thừa và trùng lặp nhiều chức năng=> sử dụng khi có nhiều dữ liệu

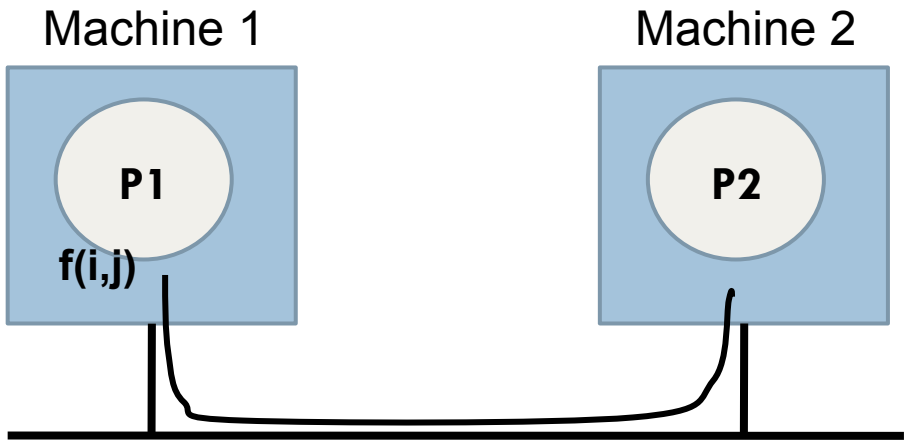
Name	*Messages sent by*		
	Client	*Server*	*Client*
R	*Request*		
RR	*Request*	*Reply*	
RRA	*Request*	*Reply*	*Acknowledge reply*

2.2. Lời gọi thủ tục từ xa
RPC (Remote Procedure Call)



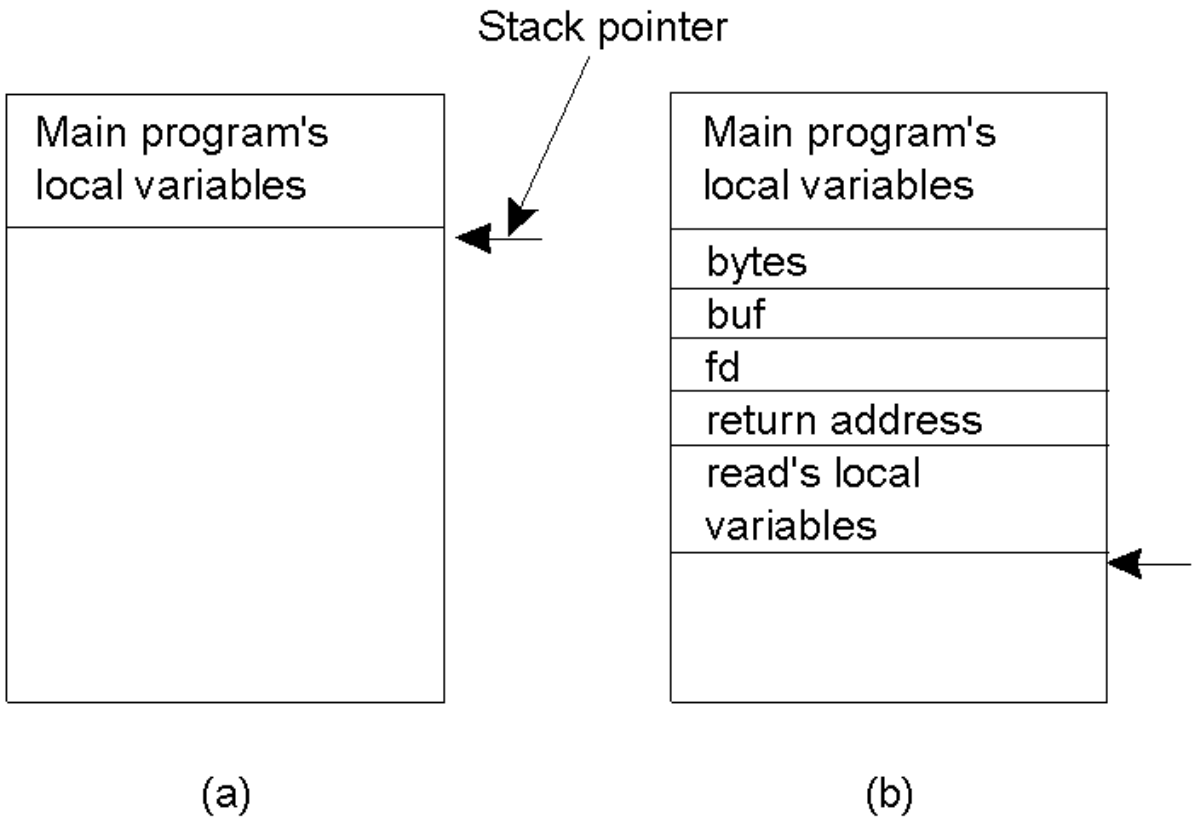
2.2. Khái niệm lời gọi thủ tục từ xa

- Cơ chế truy cập trong suốt với người dùng
- Cơ chế
- Vấn đề:
  - Hệ thống không đồng nhất
    - Không gia nhớ khác nhau
    - Cách biểu diễn thông tin khác nhau
  - Khi một trong hai máy bị hỏng



Lời gọi thủ tục thông thường

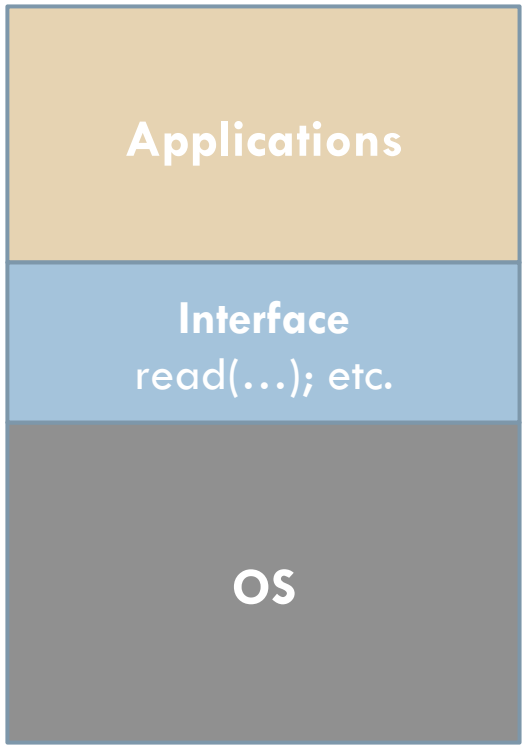
count = read(fd, buf, nbytes)

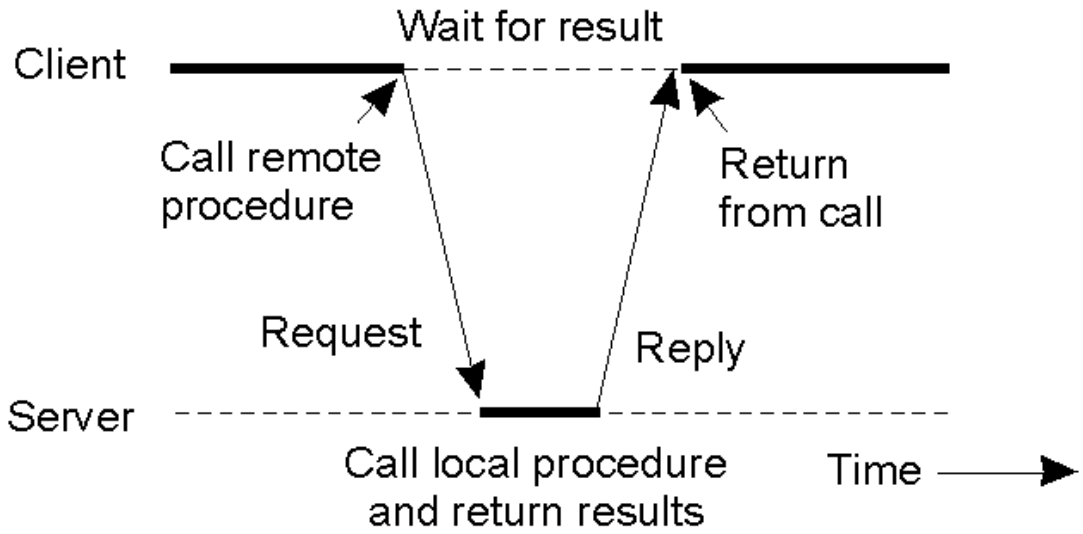


Cơ chế truyền tham số

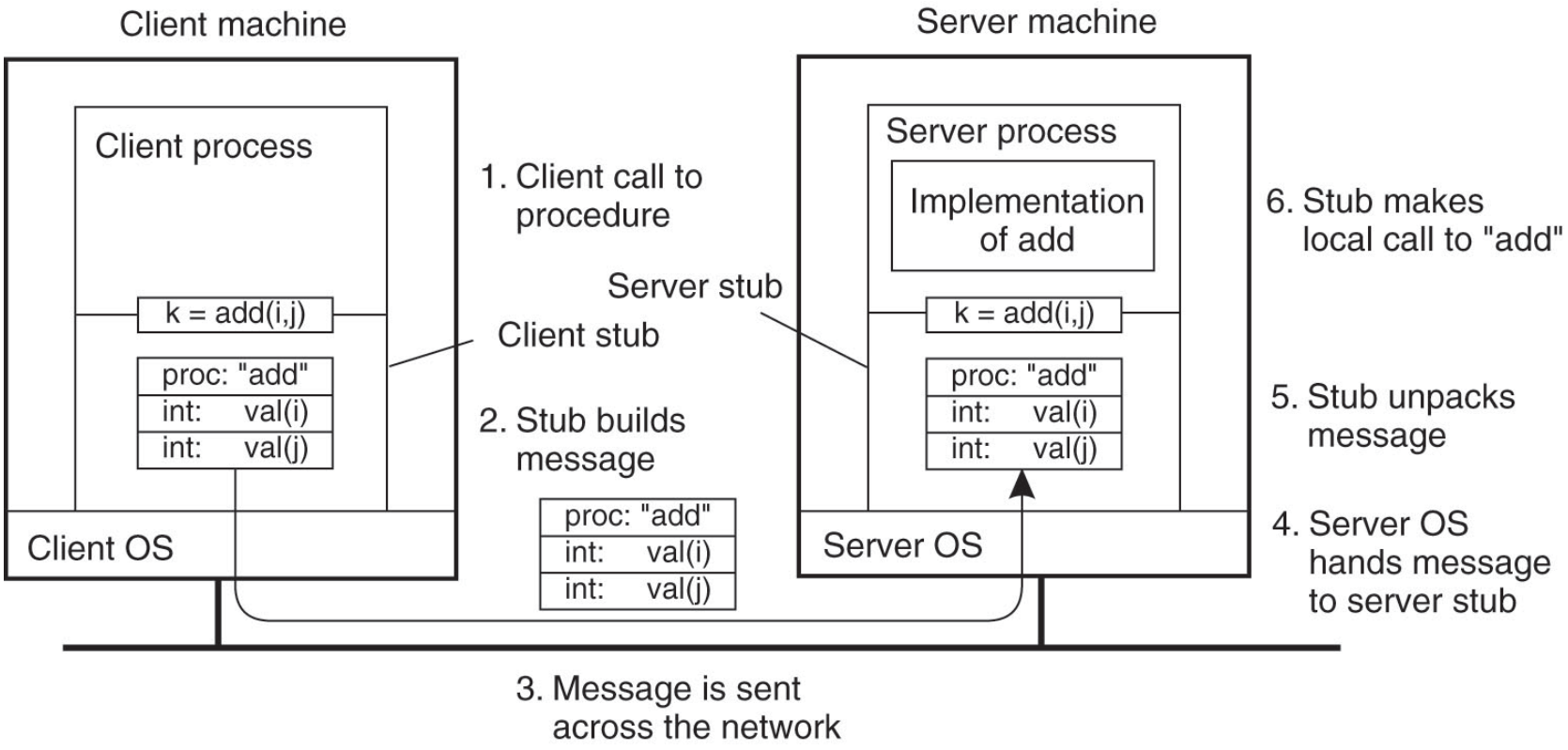
- Tham biến
- Tham chiếu
- Copy/phục hồi
  - CT gọi copy các dữ liệu vào Stack
  - CT gọi phục hồi các dữ liệu từ Stack

Cơ chế RPC


Applications
Interface
read(…); etc.
OS


Wait for result
Client
Call remote procedure
Return from call
Request
Reply
Server
Call local procedure and return results
Time

Cơ chế RPC


Client machine
Server machine
Client process
Server process
Implementation of add
k = add(i,j)
proc: "add"
int: val(i)
int: val(j)
1. Client call to procedure
Server stub
Client stub
2. Stub builds message
6. Stub makes local call to "add"
5. Stub unpacks message
4. Server OS hands message to server stub
Client OS
Server OS
3. Message is sent across the network

Vấn đề với cơ chế truyền tham số

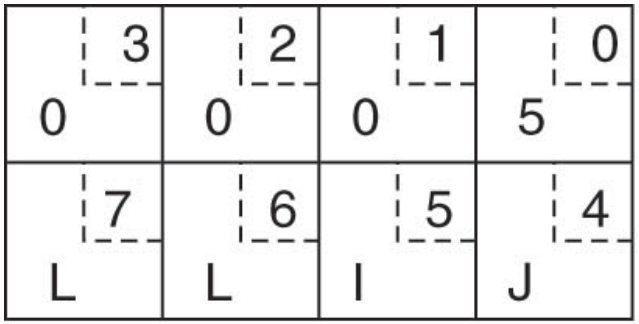
- Không dùng
- Tham biến
  - Vấn đề khi biểu diễn dữ liệu khác nhau
- Tham chiếu
  - Bộ nhớ phân tán
  - Copy dữ liệu???
    - Tham chiếu thay bằng copy/restore
  - Chuyển tham chiếu+code để truy cập vào tham chiếu

Truyền tham số bằng tham biến

- Chỉ hoạt động tốt khi hệ thống đầu cuối là đồng nhất
- Xuất hiện vấn đề khi:
  - Biểu diễn dữ liệu của 2 hệ thống khác nhau
  - Các dữ liệu không thuộc cùng một kiểu, các kiểu dữ liệu khác nhau được biểu diễn khác nhau

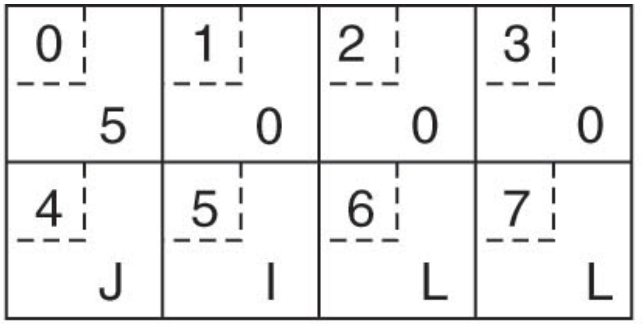
Sai lệch trong truyền bằng tham biến

Intel Pentium (little endian)

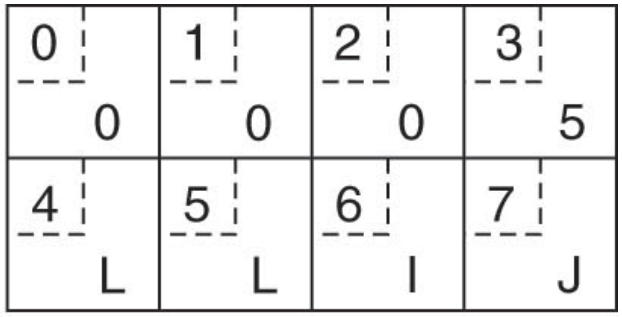


(a)

SPARC (big endian)



(b)



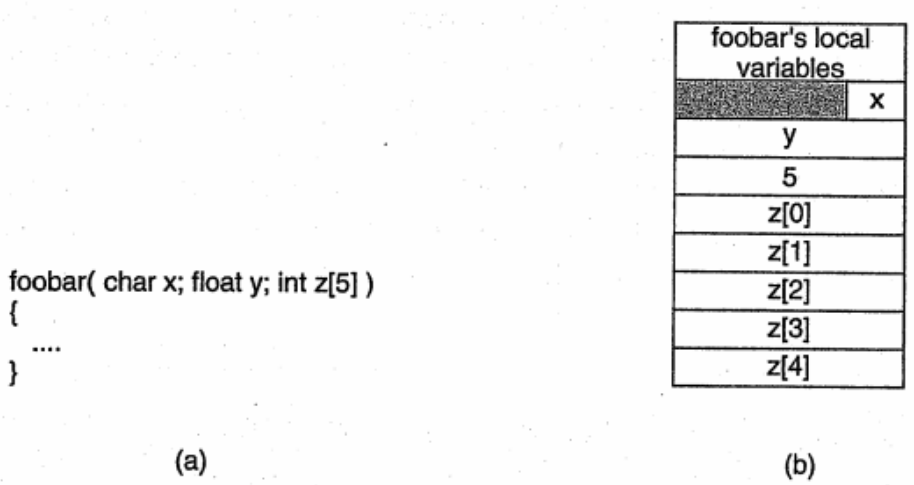
(c)

Truyền tham số bằng tham chiếu

- Vấn đề: tham chiếu chỉ có ý nghĩa cục bộ với 2 máy tính không đồng nhất
- Giải pháp:
  - Cấm sử dụng các tham chiếu
  - Copy/Restore
    - Vấn đề: tốn kém (băng thông, lưu trữ sao chép)
    - Có thể cải thiện bằng cách chỉ copy một lần (cho input hoặc output)
- Không thực hiện được tham chiếu tới các dữ liệu có cấu trúc

Đặc tả tham số

- 2 bên gửi và nhận cùng phải thống nhất về đặc tả tham số (tuân thủ 1 kiểu giao thức)
- Các yếu tố thống nhất:
  - Định dạng thông điệp
  - Cách biểu diễn cấu trúc dữ liệu cơ bản
  - Kiểu trao đổi thông điệp
  - Triển khai client-stub và server-stub



Đặc tả của CORBA

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	1984	*unsigned long*

The flattened form represents a *Person* struct with value: {'Smith', 'London', 1984}

XML


```
<person id="123456789">
        <name>Smith</name>
        <place>London</place>
        <year>1984</year>
        <!-- a comment -->
</person >
```

```
// 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();
};
```

Đặc tả của Sun


```
/*
 * date.x  Specification of the remote date and time server
 */
/*
 * Define two procedures
 *      bin_date_1() returns the binary date and time (no arguments)
 *      str_date_1() takes a binary time and returns a string
 *
 */
program DATE_PROG {
   version DATE_VERS {
      long BIN_DATE(void) = 1;    /* procedure number = 1 */
      string STR_DATE(long) = 2;  /* procedure number = 2 */
   } = 1;                         /* version number = 1 */
} = 0x31234567;                   /* program number = 0x31234567 */
```

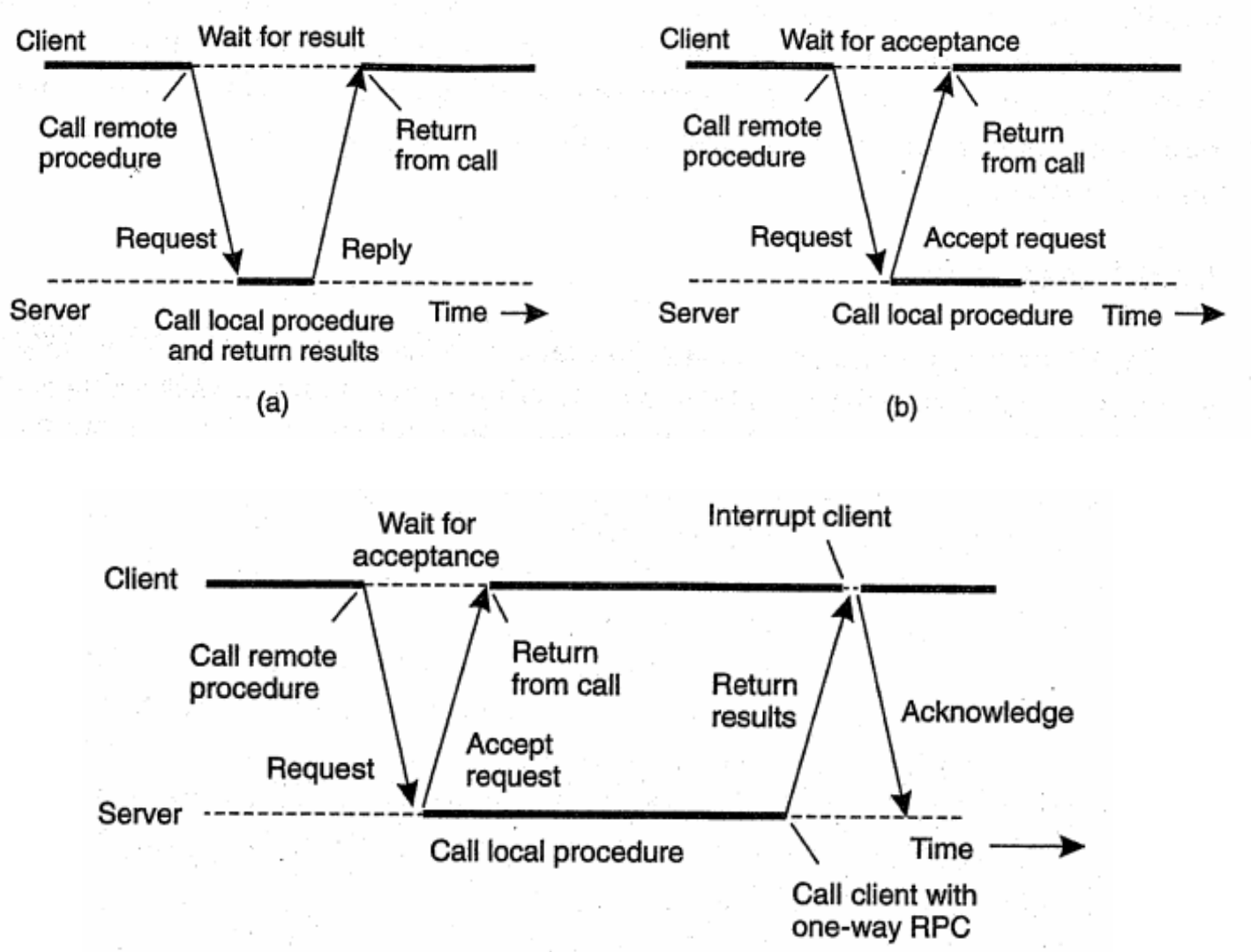
Tính mở của RPC

- Client và Server được cài đặt bởi các NSX khác nhau
- Giao diện thống nhất client và server
  - Không phụ thuộc công cụ và ngôn ngữ lập trình
  - Mô tả đầy đủ và trung lập
  - Thường dùng ngôn ngữ định nghĩa giao diện

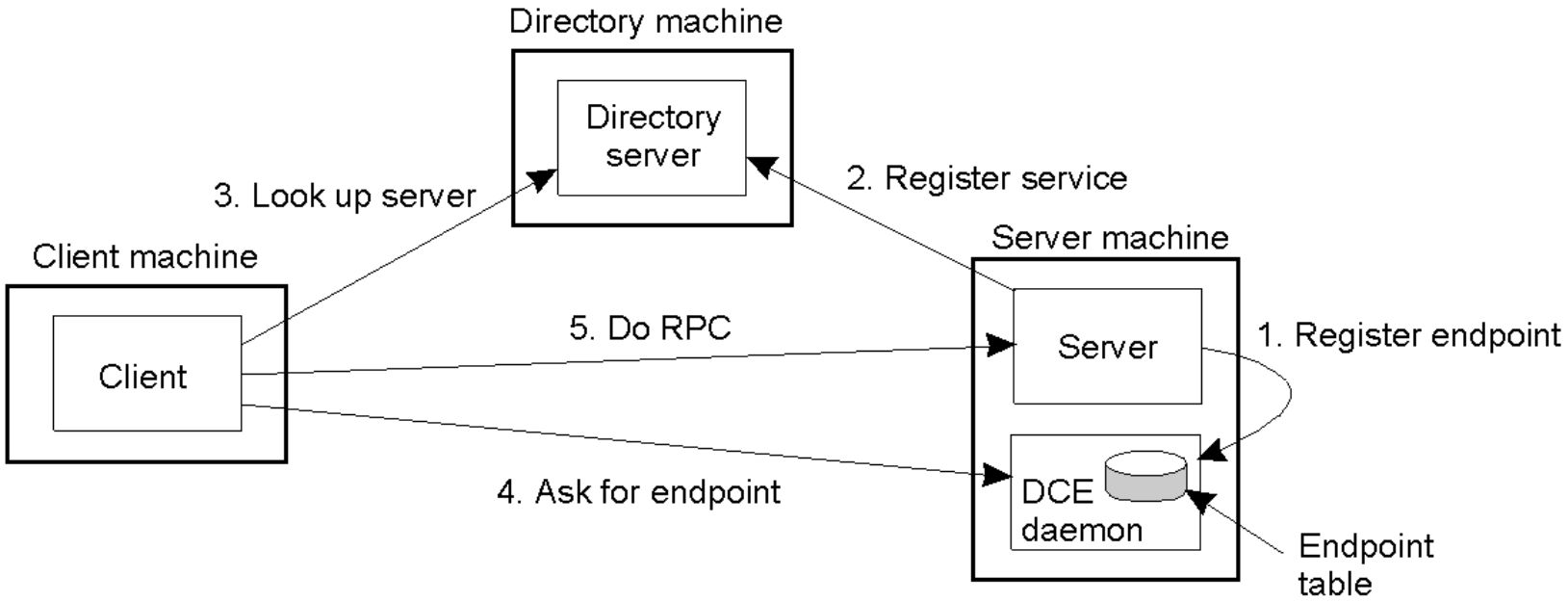
RPC không đồng bộ

- RPC: Client yêu cầu server thực hiện và trả lại kết quả
- Có nhiều trường hợp không cần trả lại kết quả
- Client sau khi gọi RPC tiếp tục thực hiện, không quan tâm đến kết quả trả lại.

RPC không đồng bộ



Liên kết client server



Liên kết client –server

- Liên kết cục bộ
  - Dịch vụ cục bộ trên máy tính server cho biết có những thủ tục nào được cung cấp
- Liên kết toàn cục
  - Dịch vụ thư mục cho biết địa chỉ máy tính và vị trí của dịch vụ.

Vấn đề: tên/địa chỉ (binding)

- Client cần chỉ ra hàm nào gọi từ xa, trên máy nào
  - Hàm nào: tên
  - Máy nào: địa chỉ
- Các thông tin này được lưu trữ tại các bảng
  - Có thể thêm, bớt các dòng trong bảng
  - Tĩnh: dịch vụ tên/thư mục
  - Động: kiểm soát bởi server và client
- Bảng có thể được xây dựng khi dịch, link hoặc thực hiện phụ thuộc vào RPC

RPC

- Người cung cấp thủ tục
  - Xây dựng giao diện
  - Dịch giao diện thành các thư viện mã nguồn
  - Xây dựng mã nguồn của thủ tục
  - Dịch thành chương trình server
  - Khởi động hệ thống RPC
  - Khởi động server
- Người sử dụng thủ tục
  - Cần có các thư viện mã nguồn
  - Xây dựng mã nguồn của client, trong đó
    - Kết nối với hệ thống RPC
    - Tra cứu về RPC cần sử dụng
    - Gọi RPC

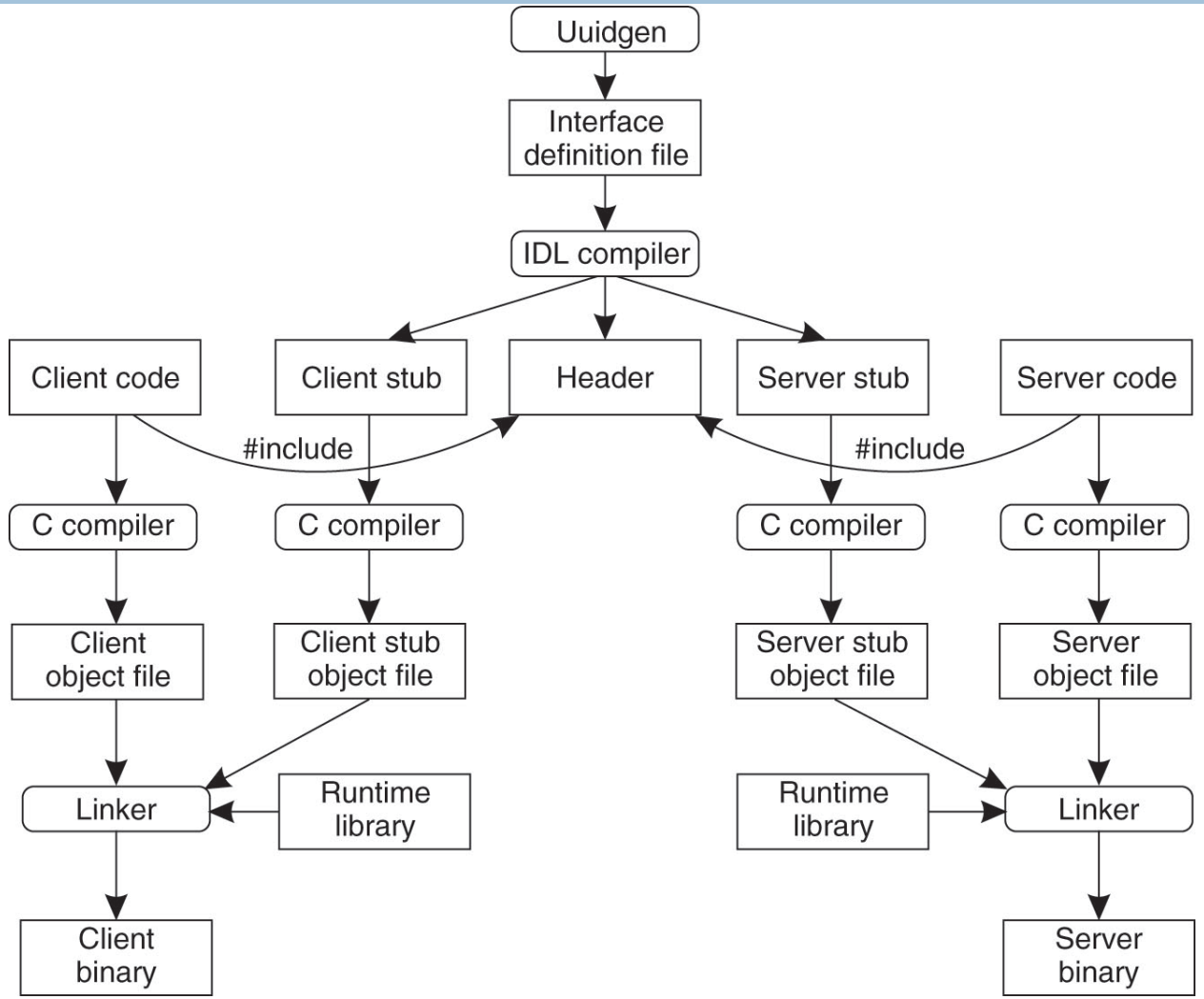
2.3. Hệ thống DCE RPC

- Distributed Computing Environment (DCE) được phát triển bởi Open Group: http://www.opengroup.org/dce/
- Tầng middleware
- Mô hình client-server
- Giao tiếp được thực hiện thông qua RPCs
- Các dịch vụ:
  - Distributed file service
  - Directory service
  - Distributed time service

Mục đích

- Sử dụng RPCs để truy cập các dịch vụ từ xa
- Đơn giản hóa việc lập trình
- Trong suốt
- Client và server hoàn toàn độc lập

Xây dựng chương trình bằng DCE-RPC



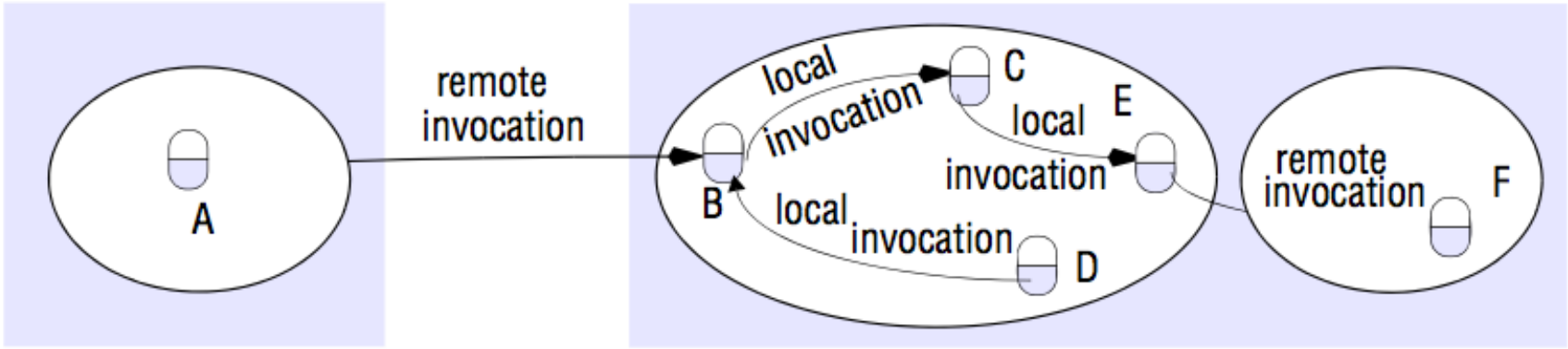
2.4. RMI (Remote Method Invocation)

- So sánh với RPC
  - Giống:
    - Cùng hỗ trợ lập trình với các giao diện
    - Dựa trên giao thức yêu cầu/trả lời
    - Mức độ trong suốt
  - Khác:
    - Lập trình viên có thể sử dụng khai thác hết điểm mạnh của OOP
    - Định danh duy nhất → truyền tham chiếu đối tượng

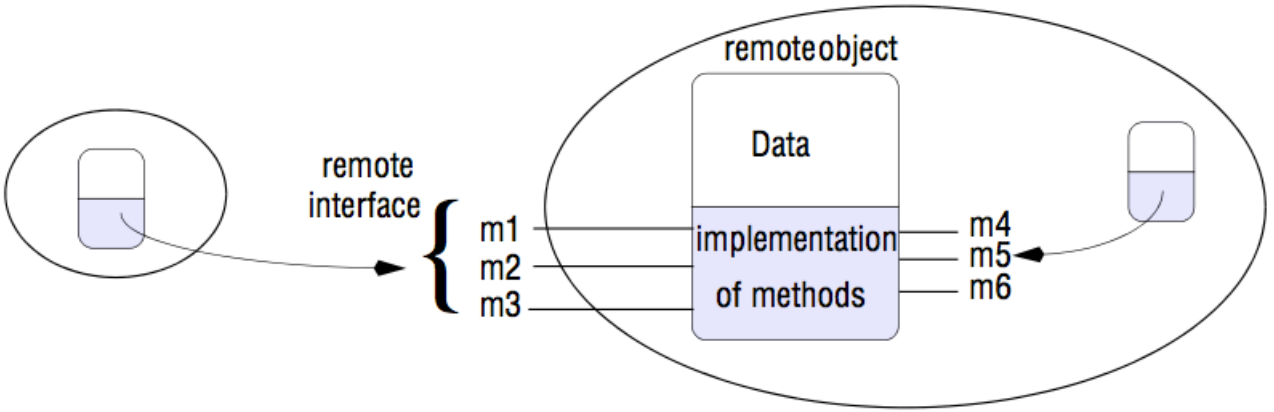
RMI: Lời gọi phương thức từ xa

- Lập trình hướng đối tượng :
  - đối tượng từ xa, ứng dụng phân tán hướng đối tượng
- Các vấn đề cần giải quyết
  - Định vị đối tượng từ xa
  - Trao đổi thông tin với đối tượng
  - Gọi các phương thức của đối tượng
- RMI, T-RMI, DCOM, CORBA

Mô hình đối tượng phân tán



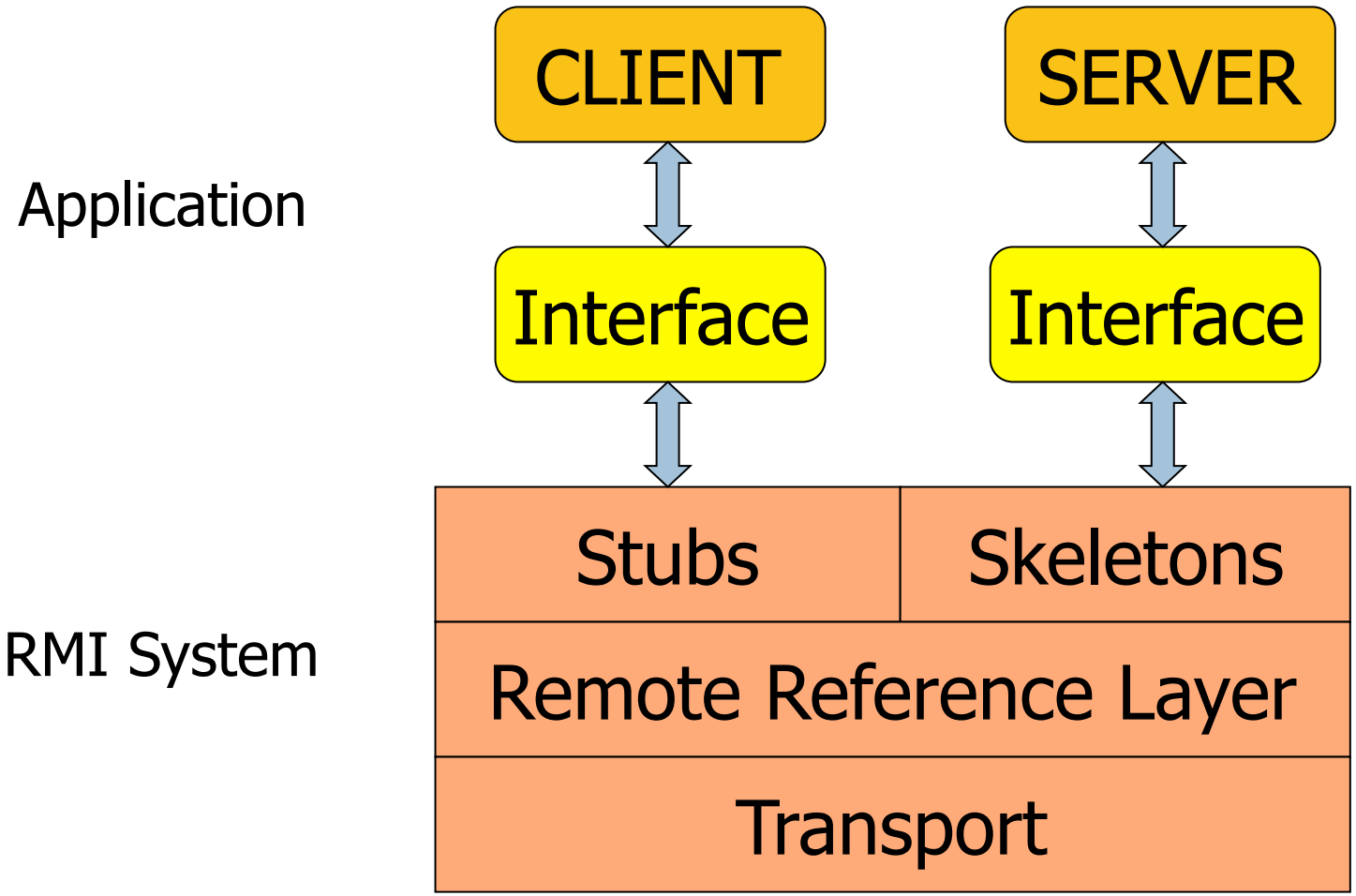
Đối tượng từ xa và giao diện từ xa



Ưu điểm

- Đơn giản, dễ sử dụng
- Trong suốt: lời gọi phương thức từ xa giống lời gọi phương thức cục bộ
- Độ tin cậy cao
- An toàn và bảo mật (do JVM cung cấp)
- Nhược điểm:
  - Chỉ dùng cho java

Kiến trúc


Application
RMI System
CLIENT
SERVER
Interface
Interface
Stubs
Skeletons
Remote Reference Layer
Transport

3. Trao đổi thông tin hướng thông điệp

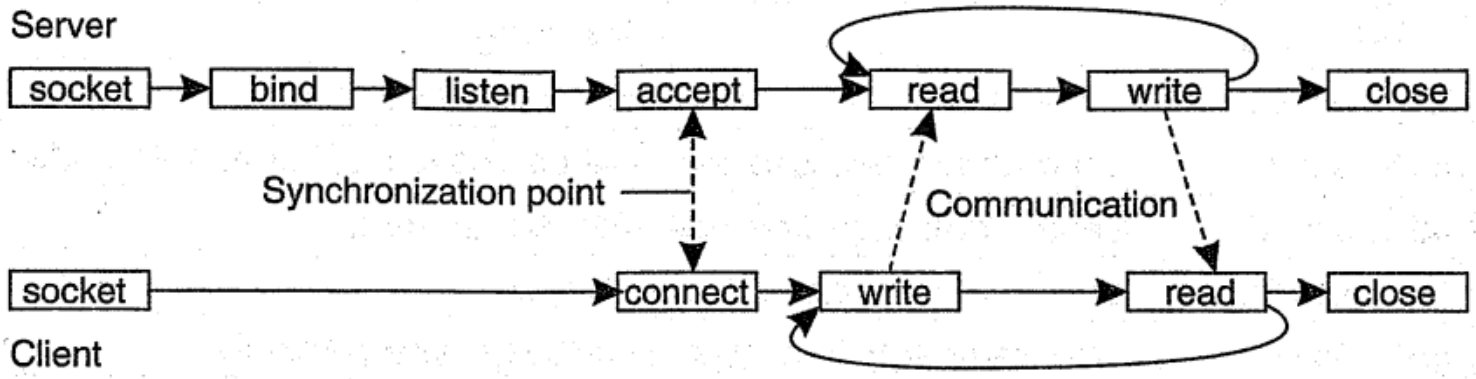
3.1. Trao đổi thông tin hướng thông báo tạm thời

3.2. Trao đổi thông tin hướng thông điệp bền vững

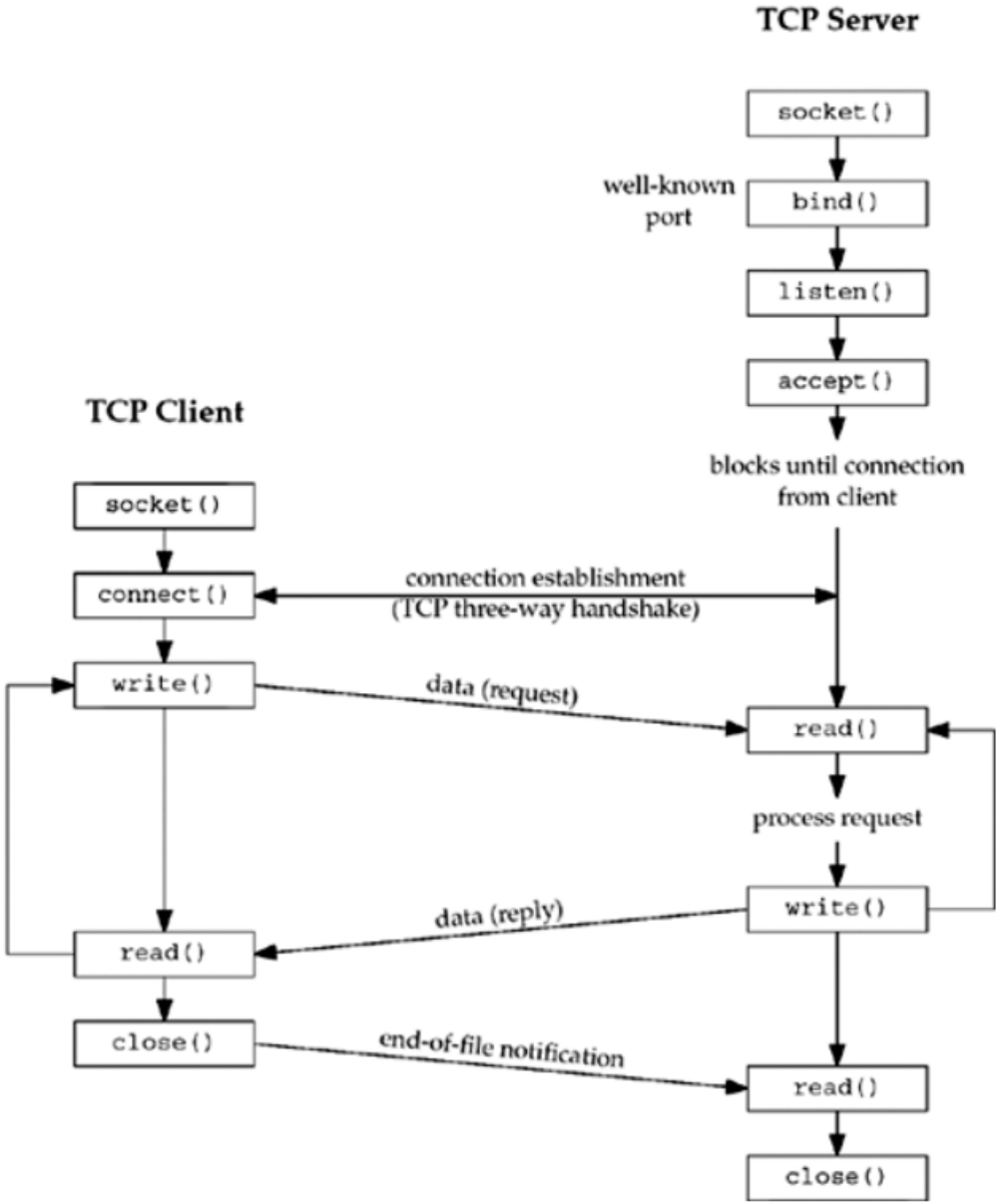
3.1. Trao đổi thông tin hướng thông điệp tạm thời

- Berkeley Sockets

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection



Introduction



socket function

- To perform network I/O, the first thing a process must do is call the *socket* function

```
#include <sys/socket.h>
int socket (int family, int type, int protocol);
```

- Returns: non-negative descriptor if OK, -1 on error

family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_LOCAL	Unix domain protocols (Chapter 15)
AF_ROUTE	Routing sockets (Chapter 18)
AF_KEY	Key socket (Chapter 19)

family

type	Description
SOCK_STREAM	stream socket
SOCK_DGRAM	datagram socket
SOCK_SEQPACKET	sequenced packet socket
SOCK_RAW	raw socket

socket

Protocol	Description
IPPROTO_TCP	TCP transport protocol
IPPROTO_UDP	UDP transport protocol
IPPROTO_SCTP	SCTP transport protocol

protocol

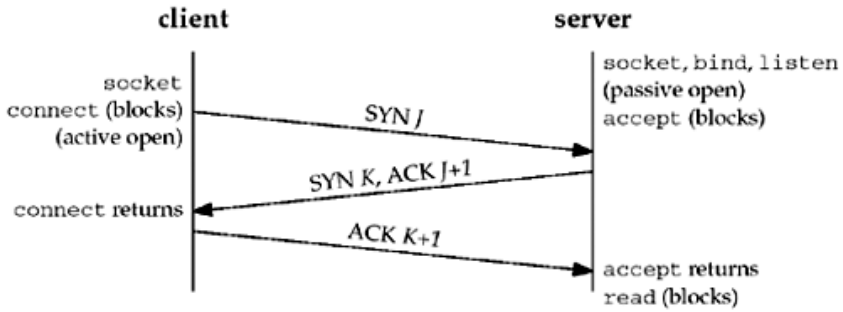
connect Function

- The connect function is used by a TCP client to establish a connection with a TCP server.

```
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *servaddr,
   socklen_t addrlen);
```

- Returns: 0 if OK, -1 on error
- *sockfd* is a socket descriptor returned by the *socket* function
- The second and third arguments are a pointer to a socket address structure and its size.
- The client does not have to call *bind* before calling *connect*: the kernel will choose both an ephemeral port and the source IP address if necessary.

Nhắc lại: Thiết lập liên kết TCP : Giao thức bắt tay 3 bước



- Bước 1: A gửi SYN cho B
  - chỉ ra giá trị khởi tạo seq # của A
  - không có dữ liệu
- Bước 2: B nhận SYN, trả lời bằng SYNACK
  - B khởi tạo vùng đệm
  - chỉ ra giá trị khởi tạo seq. # của B
- Bước 3: A nhận SYNACK, trả lời ACK, có thể kèm theo dữ liệu

connect Function (2)

- Problems with *connect* function:
  1. If the client TCP receives no response to its SYN segment, ETIMEDOUT is returned. (If no response is received after a total of 75 seconds, the error is returned).
  2. If the server's response to the client's SYN is a reset (RST), this indicates that no process is waiting for connections on the server host at the port specified (i.e., the server process is probably not running). Error: ECONNREFSED.
  3. If the client's SYN elicits an ICMP "destination unreachable" from some intermediate router, this is considered a soft error. If no response is received after some fixed amount of time (75 seconds for 4.4BSD), the saved ICMP error is returned to the process as either EHOSTUNREACH or ENETUNREACH.

bind Function

- The bind function assigns a local protocol address to a socket.

```
#include <sys/socket.h>
int bind (int sockfd, const struct sockaddr *myaddr,
   socklen_t addrlen);
```

- Returns: 0 if OK,-1 on error
- Examp

```
struct sockaddr_in address;

/* type of socket created in socket() */
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = INADDR_ANY;
/* 7000 is the port to use for connections */
  address.sin_port = htons(7000);
/* bind the socket to the port specified above */
```

listen Function

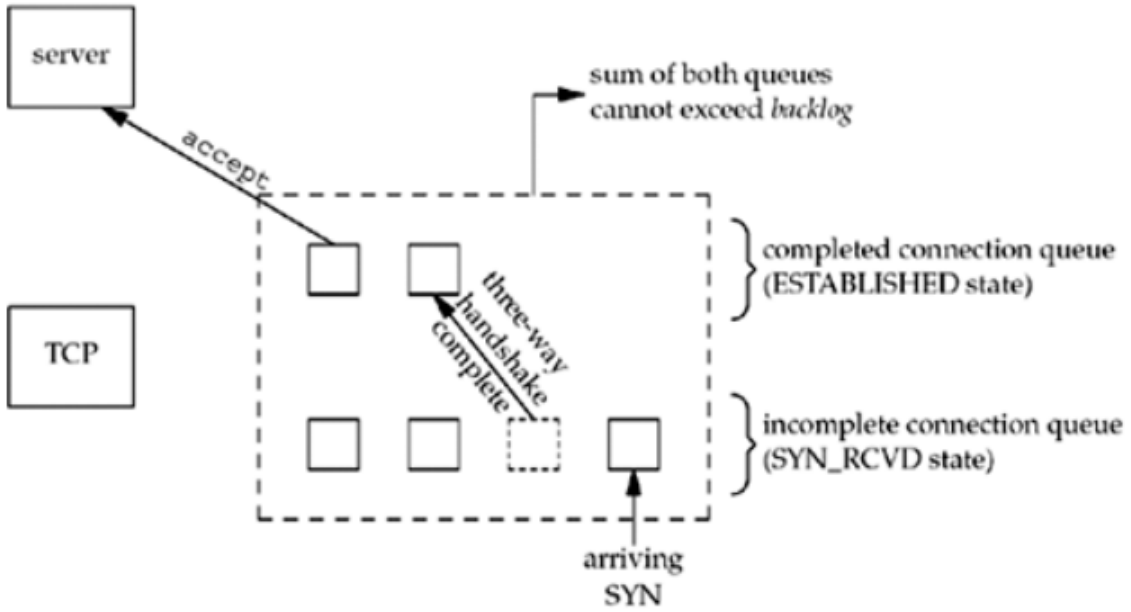
- The listen function is called only by a TCP server.
- When a socket is created by the *socket* function, it is assumed to be an active socket, that is, a client socket that will issue a *connect*.
- The *listen* function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.
- Move the socket from the CLOSED state to the LISTEN state.

```
#include <sys/socket.h>
int listen (int sockfd, int backlog);
```

- Returns: 0 if OK, -1 on error

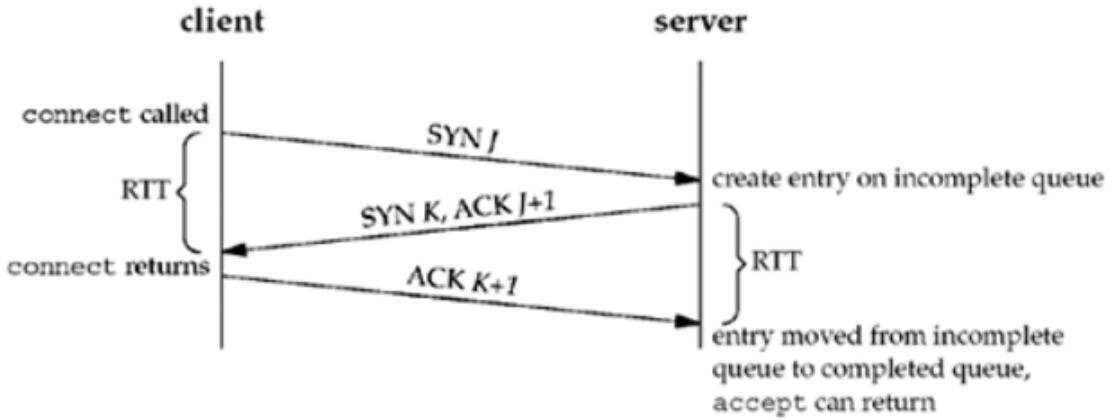
listen Function (2)

- The second argument (*backlog*) to this function specifies the maximum number of connections the kernel should queue for this socket



The two queues maintained by TCP for a listening socket

listen Function (3)



TCP three-way handshake and the two queues for a listening socket.

accept Function

- *accept* is called by a TCP server to return the next completed connection from the front of the completed connection queue.
- If the completed connection queue is empty, the process is put to sleep.

```
#include <sys/socket.h>
int accept (int sockfd, struct sockaddr *cliaddr, socklen_t
   *addrlen);
```

- Returns: non-negative descriptor if OK, -1 on error
- The *cliaddr* and addrlen arguments are used to return the protocol address of the connected peer process (the client).
- *addrlen* is a value-result argument

accept Function

- Example

```
int addrlen;
struct sockaddr_in address;

  addrlen = sizeof(struct sockaddr_in);
  new_socket = accept(socket_desc, (struct sockaddr *)&address, &addrlen);
  if (new_socket<0)
    perror("Accept connection");
```

fork and *exec* Functions

```
#include <unistd.h>
pid_t fork(void);
```

- Returns: 0 in child, process ID of child in parent, -1 on error
- *fork* function (including the variants of it provided by some systems) is the only way in Unix to create a new process.
- It is called once but it returns twice.
- It returns once in the calling process (called the parent) with a return value that is the process ID of the newly created process (the child). It also returns once in the child, with a return value of 0.
- The reason fork returns 0 in the child, instead of the parent's process ID, is because a child has only one parent and it can always obtain the parent's process ID by calling *getppid*.

Example

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    printf("--beginning of program\n");

    int counter = 0;
    pid_t pid = fork();

    if (pid == 0)
    {
        // child process
        int i = 0;
        for (; i < 5; ++i)
        {
            printf("child process: counter=%d\n", ++counter);
        }
    }
    else if (pid > 0)
    {
        // parent process
        int j = 0;
        for (; j < 5; ++j)
        {
            printf("parent process: counter=%d\n", ++counter);
        }
    }
    else
    {
        // fork failed
        printf("fork() failed!\n");
        return 1;
    }

    printf("--end of program--\n");

    return 0;
}
```

- 2 typical uses of fork:
  - A process makes a copy of itself so that one copy can handle one operation while the other copy does another task.
  - A process wants to execute another program.

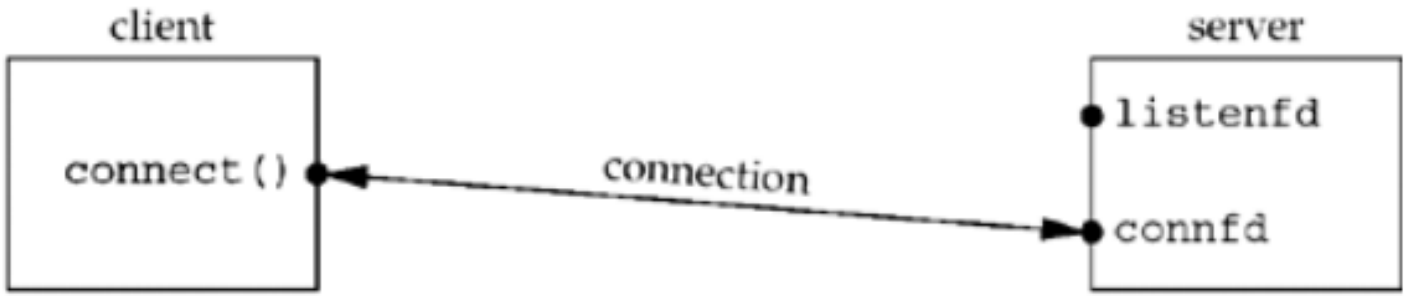
Concurrent Servers

- *fork* a child process to handle each client

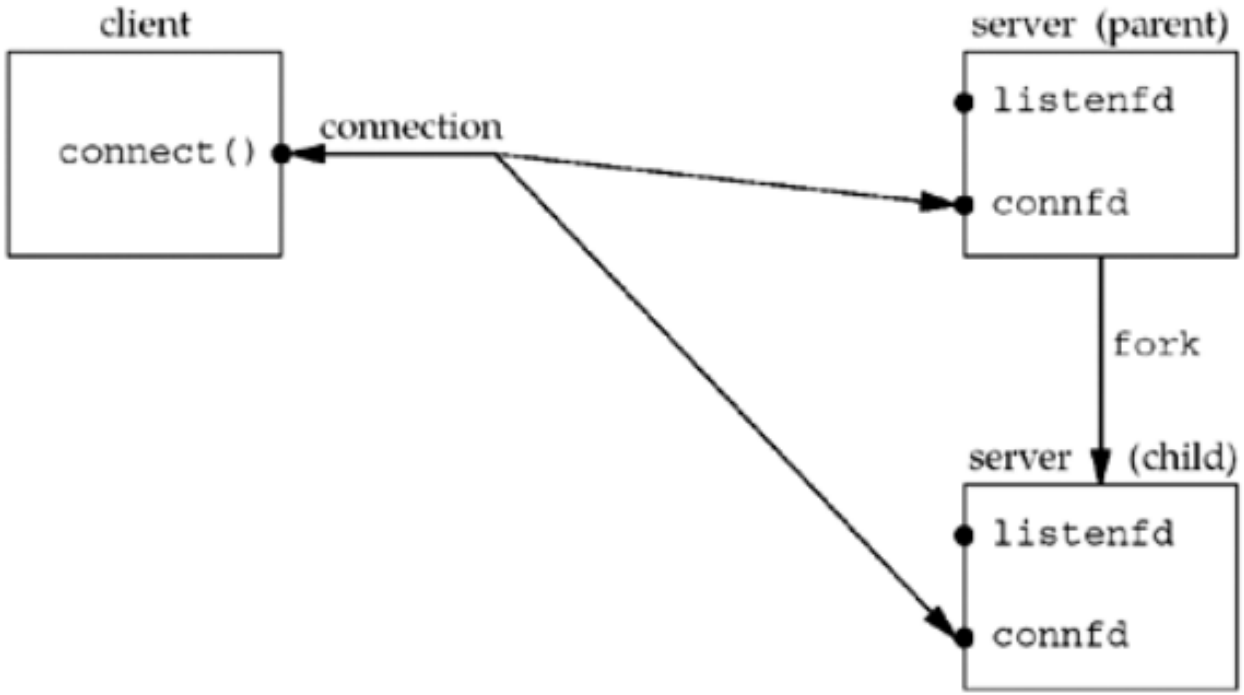
```
pid_t pid;
int   listenfd,  connfd;

listenfd = Socket( ... );

    /* fill in sockaddr_in{} with server's well-known port */
Bind(listenfd, ... );
Listen(listenfd, LISTENQ);

for ( ; ; ) {
    connfd = Accept (listenfd, ... );    /* probably blocks */

    if( (pid = Fork()) == 0) {
       Close(listenfd);    /* child closes listening socket */
       doit(connfd);       /* process the request */
       Close(connfd);      /* done with this client */
       exit(0);            /* child terminates */
    }

    Close(connfd);         /* parent closes connected socket */
}
```

Status of client/server before call to *accept* returns.

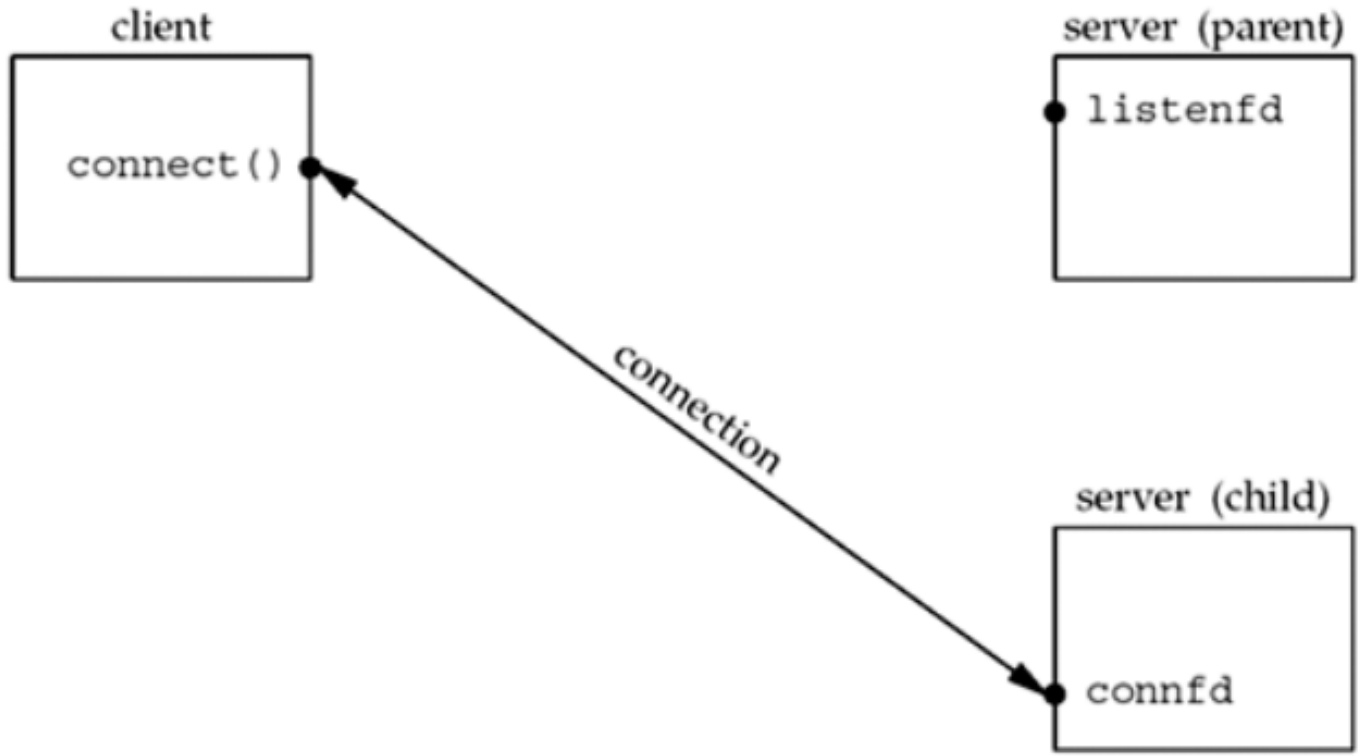
Status of client/server after return from *accept*.



Status of client/server after fork returns.



Status of client/server after parent and child close appropriate sockets.



close Function

- The normal Unix close function is also used to close a socket and terminate a TCP connection.

```
#include <unistd.h>
int close (int sockfd);
```

- Returns: 0 if OK, -1 on error
- If the parent doesn't close the socket, when the child closes the connected socket, its reference count will go from 2 to 1 and it will remain at 1 since the parent never closes the connected socket. This will prevent TCP's connection termination sequence from occurring, and the connection will remain open.

Message-Passing Interface

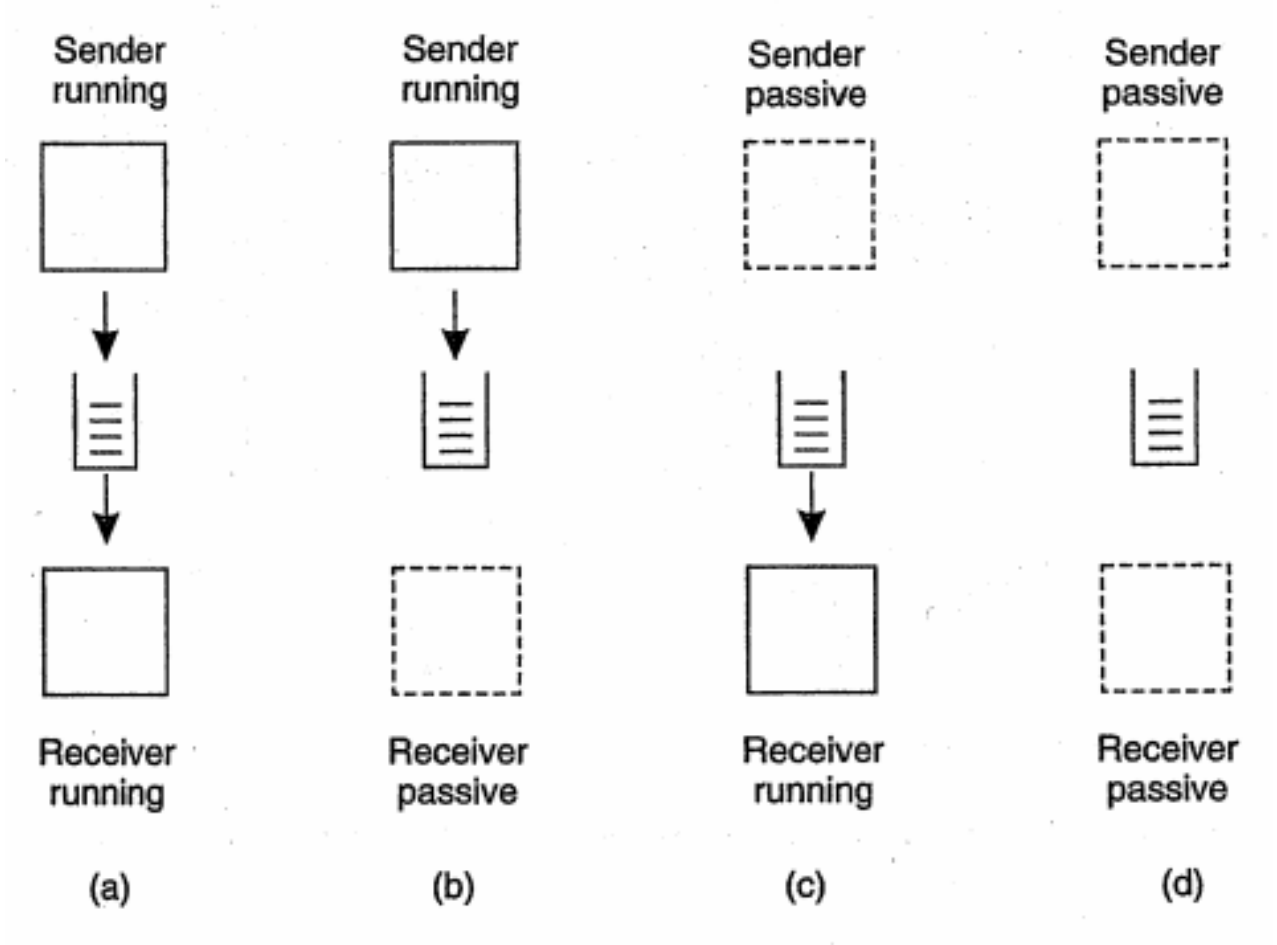
Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

Các hàm truyền thông điệp thường dùng của MPI

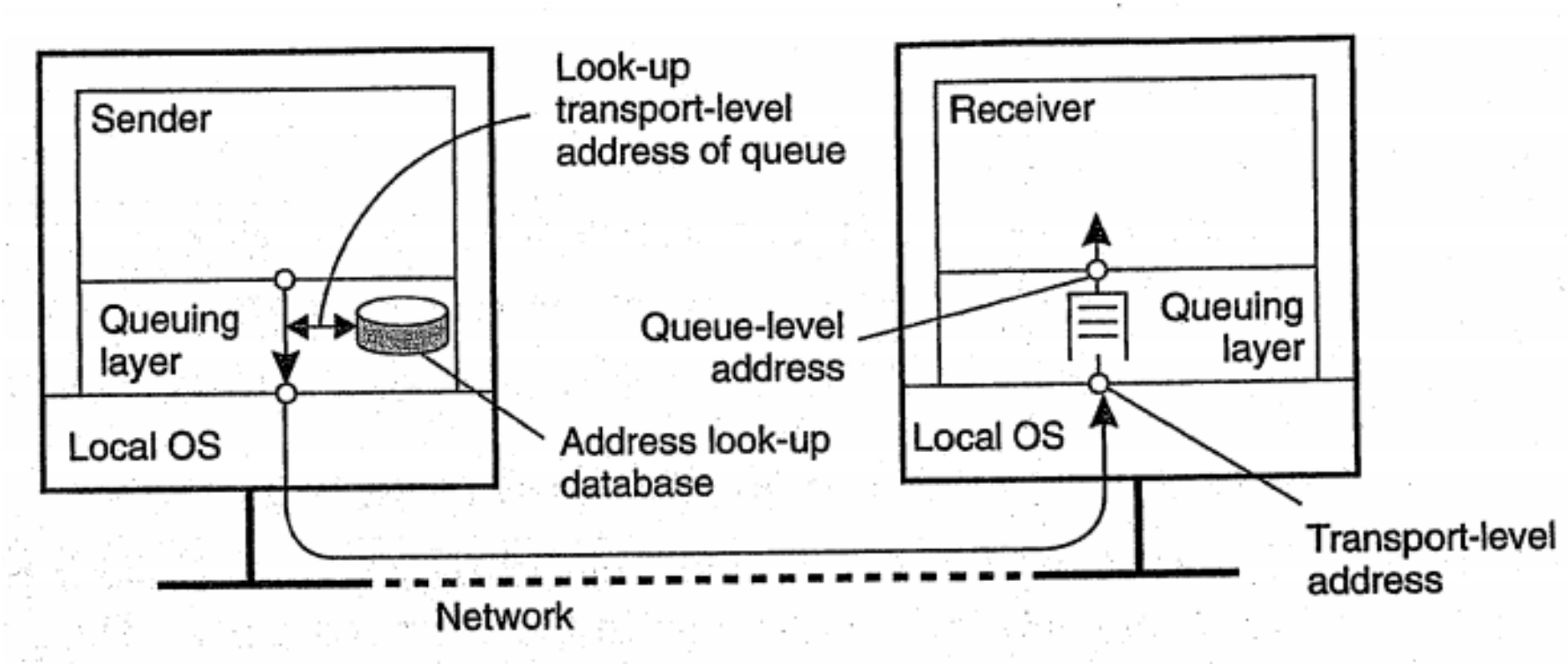
3.2. Trao đổi thông tin hướng thông điệp bền vững

- MOM (Message-Oriented Middleware)
- Hệ thống hàng đợi thông điệp hỗ trợ trao đổi thông tin không đồng bộ bền vững.
- Hỗ trợ khả năng lưu trữ trung gian cho thông điệp
- Chấp nhận độ trễ thời gian cao
- VD: hệ thống email

Mô hình hàng đợi thông điệp

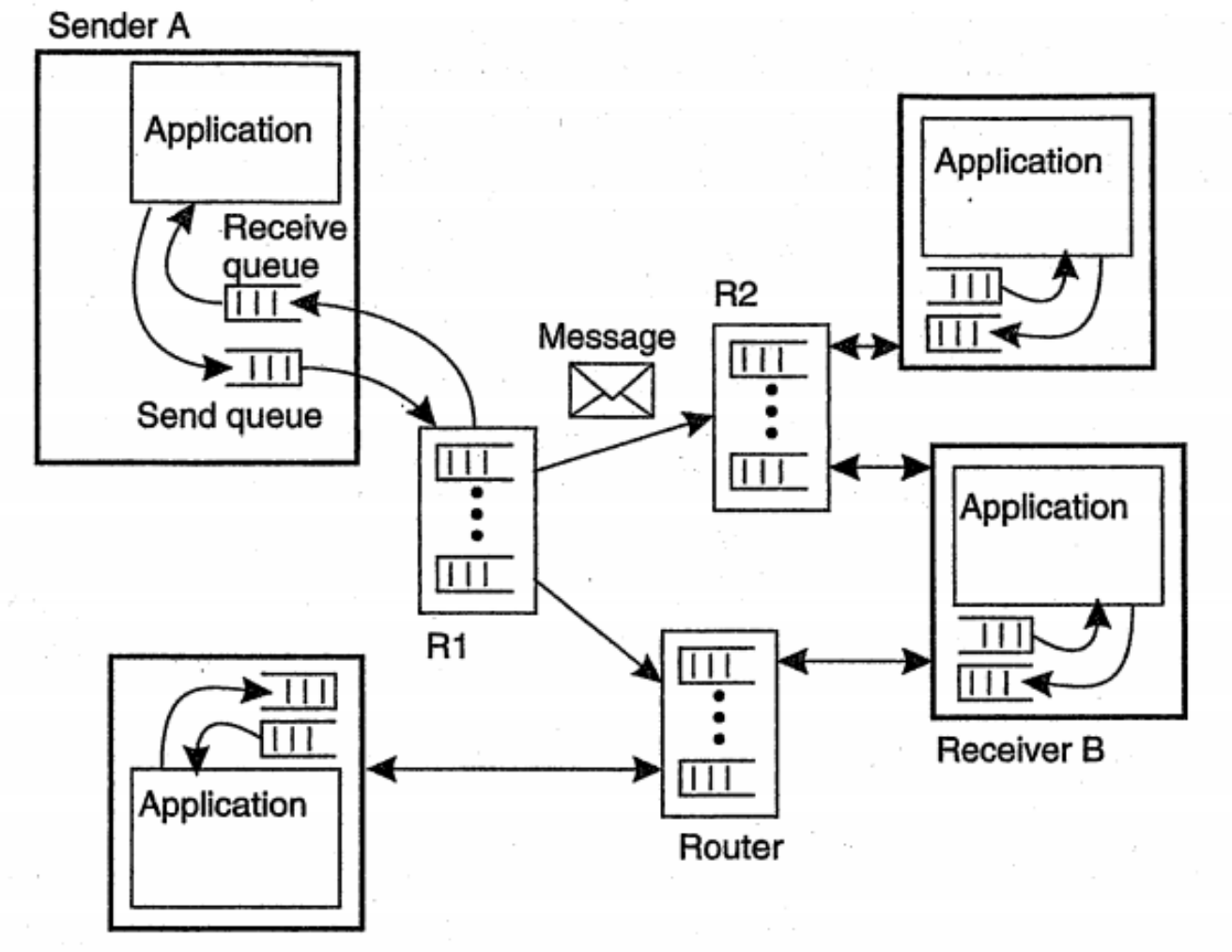


Định địa chỉ

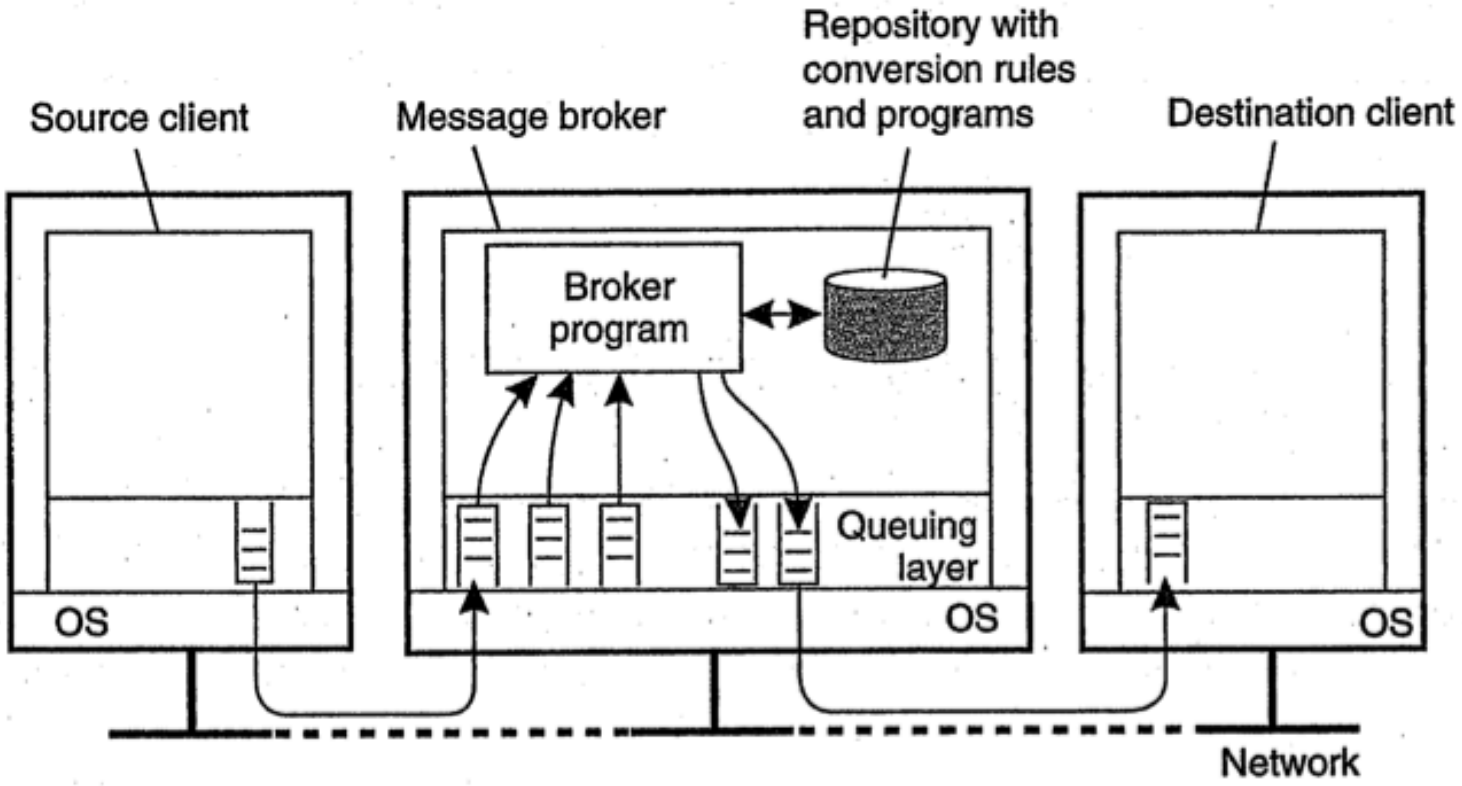


Mối quan hệ giữa định địa chỉ mức hàng đợi và mức mạng

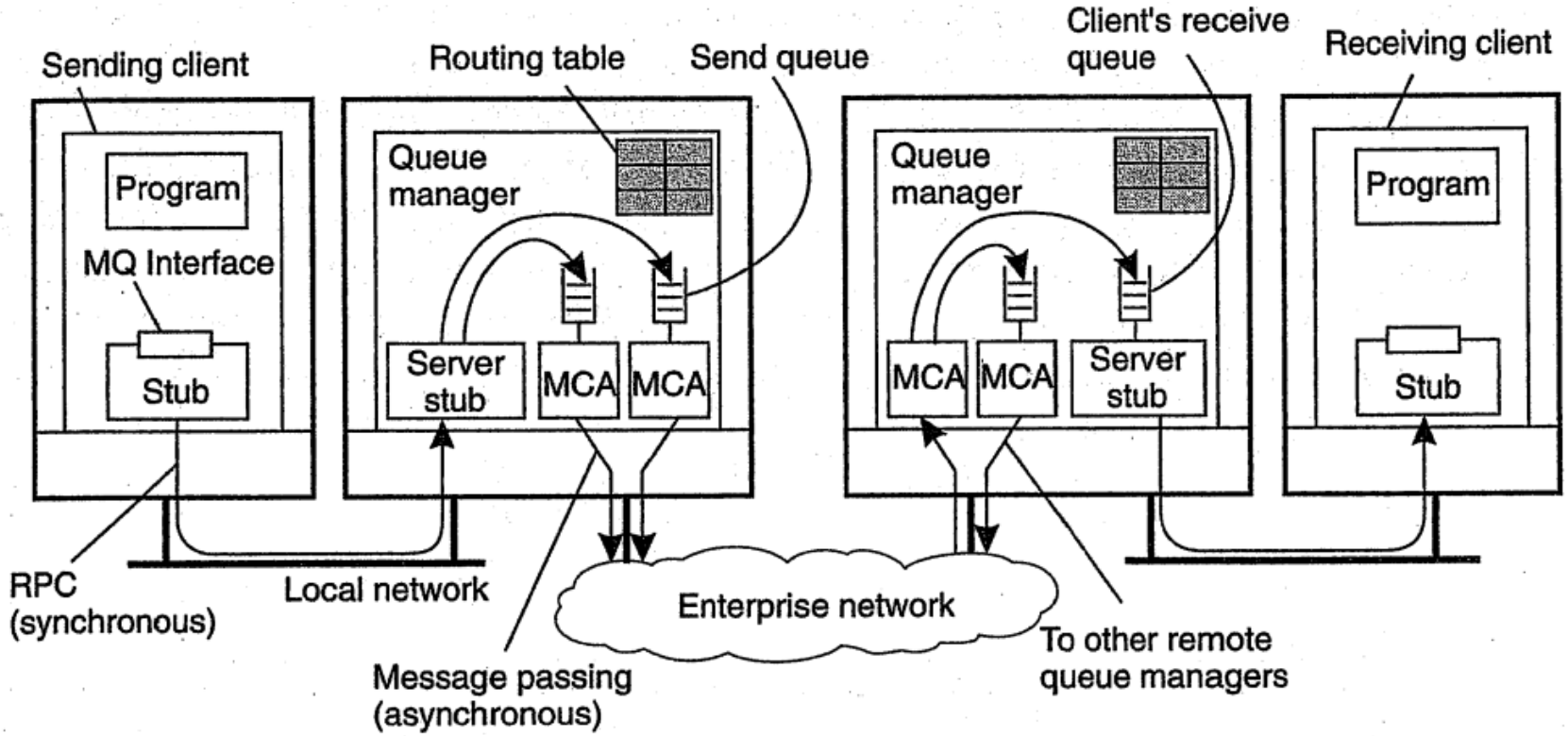
Hệ thống hàng đợi thông điệp với các routers



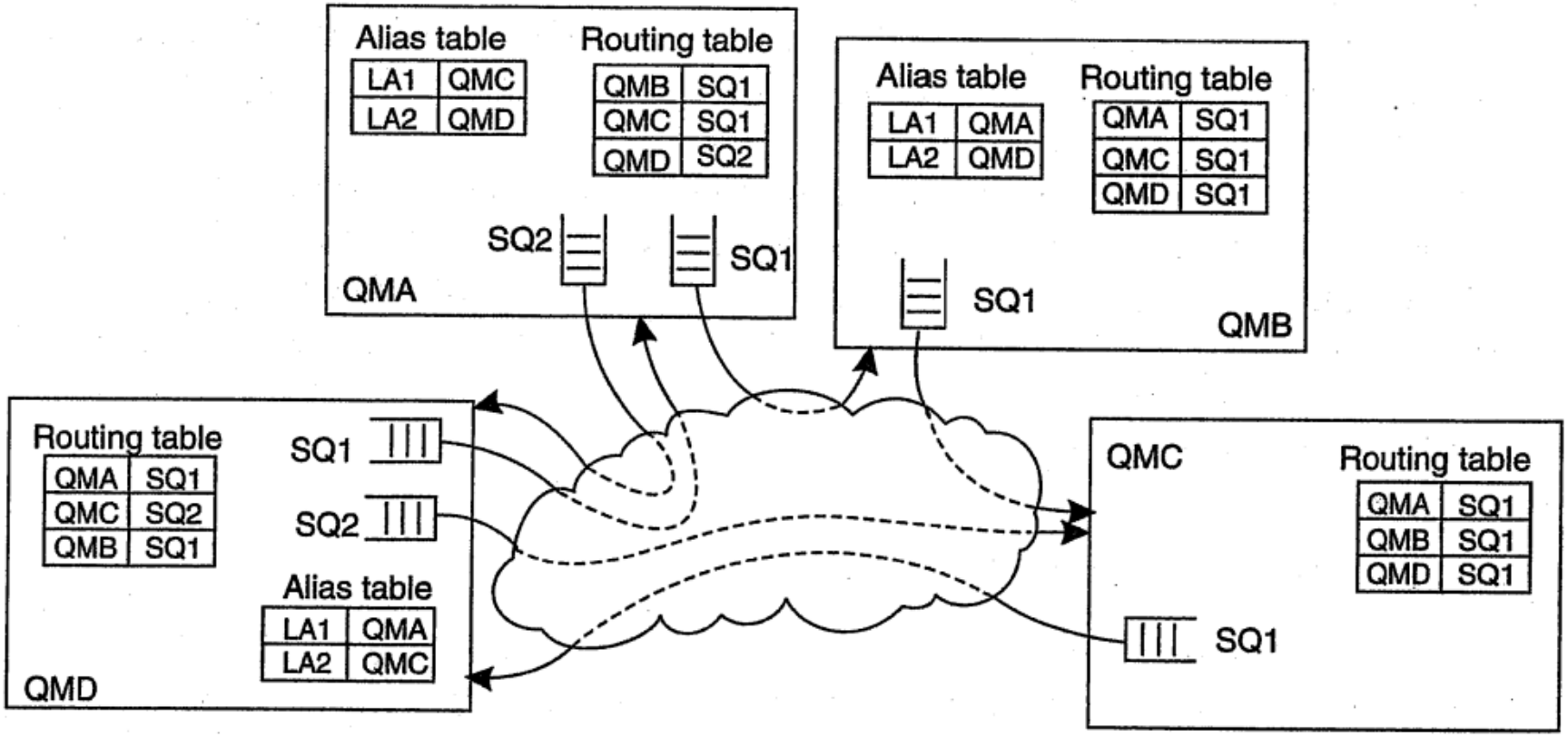
Message Broker



Hệ thống xếp hàng thông điệp WebSphere của IBM



Truyền thông điệp



4. Trao đổi thông tin hướng dòng

4.1. Hỗ trợ cho phương tiện truyền thông liên tục

4.2. Dòng và QoS

4.3. Đồng bộ hoá dòng

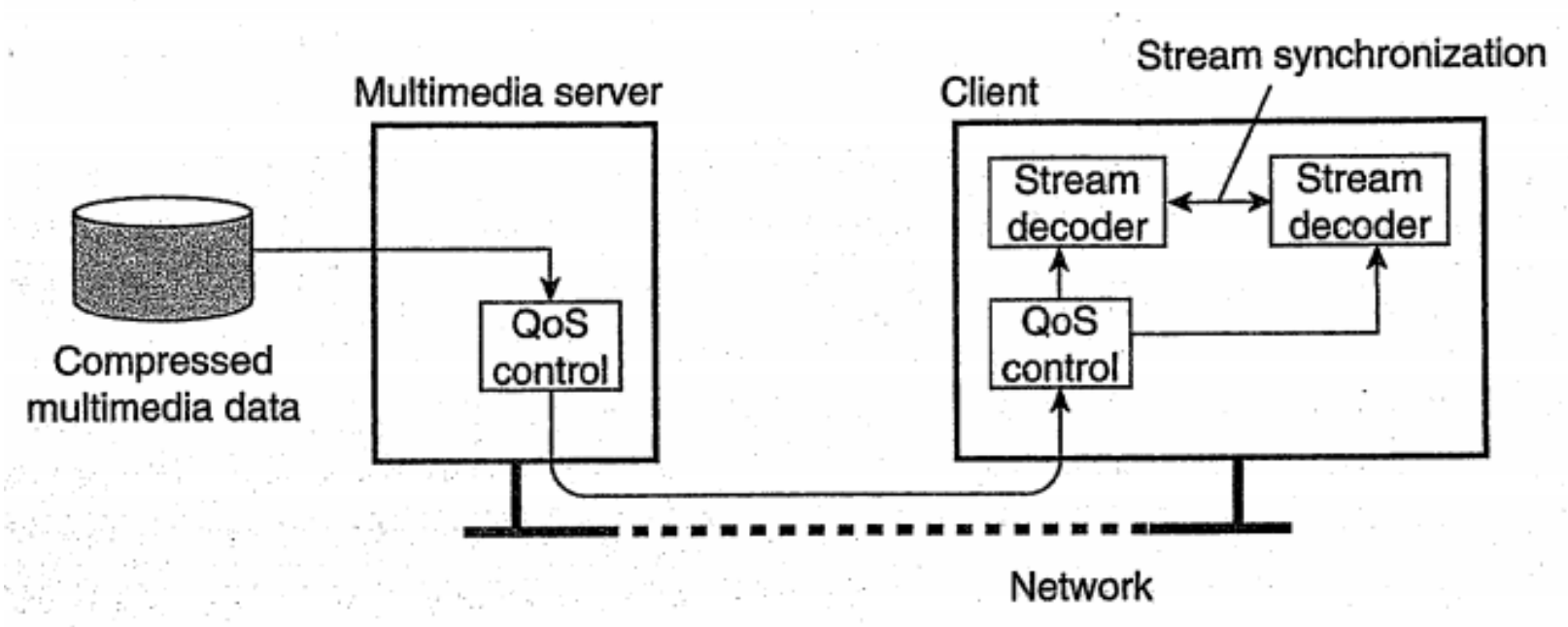
4.1. Hỗ trợ cho phương tiện truyền thông liên tục

- Phương tiện truyền đạt thông tin
  - Lưu trữ
  - Truyền tin
  - Biểu diễn (màn hình, v.v…)
- Phương tiện truyền thông liên tục/rời rạc

Dòng dữ liệu

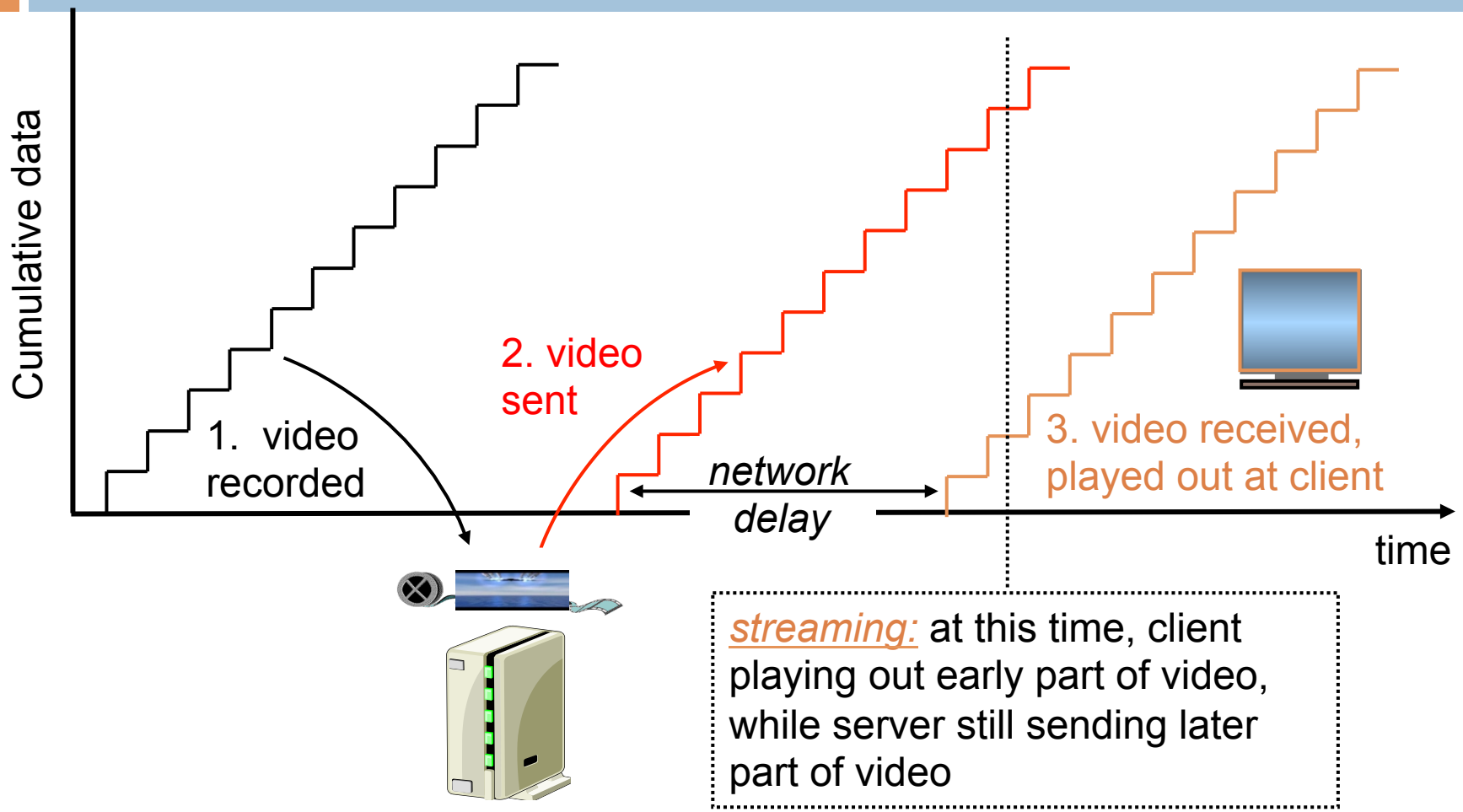
- Chuỗi các đơn vị dữ liệu liên tục
- Truyền tin không đồng bộ
- Truyền tin đồng bộ
- Truyền tin đẳng thời
- Dòng dữ liệu đơn & phức
- Dòng dữ liệu thời gian thực
- Vấn đề
  - Nén dữ liệu
  - Kiểm soát chất lượng đường truyền
  - Đồng bộ hóa

Dòng dữ liệu (cont.)



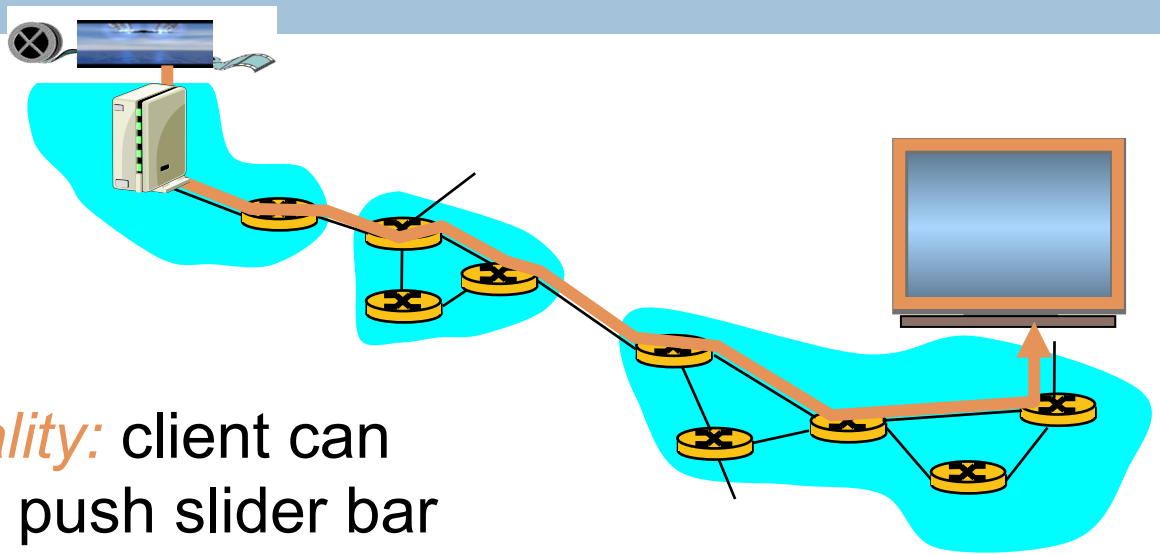
Kiến trúc chung truyền dữ liệu đa phương tiện qua mạng

Truyền dòng dữ liệu đa phương tiện lưu trữ



streaming: at this time, client playing out early part of video, while server still sending later part of video

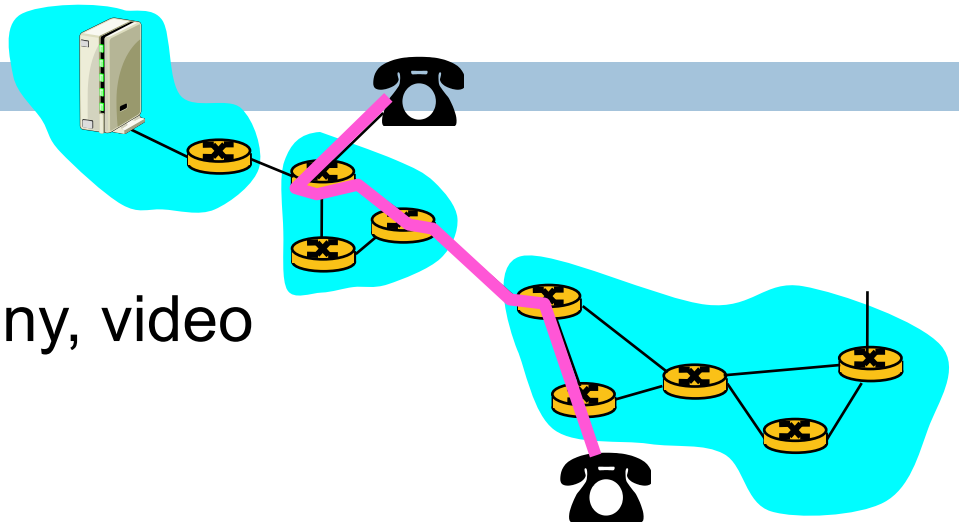
Truyền dòng dữ liệu đa phương tiện: Có tương tác



- *VCR-like functionality:* client can pause, rewind, FF, push slider bar
  - 10 sec initial delay OK
  - 1-2 sec until command effect OK

- timing constraint for still-to-be transmitted data: in time for playout

Real-Time Interactive Multimedia



- applications: IP telephony, video conference, distributed interactive worlds
- end-end delay requirements:
  - audio: < 150 msec good, < 400 msec OK
    - includes application-level (packetization) and network delays
    - higher delays noticeable, impair interactivity
- session initialization
  - how does callee advertise its IP address, port number, encoding algorithms?

Nén dữ liệu audio

- analog signal sampled at constant rate
  - telephone: 8,000 samples/sec
  - CD music: 44,100 samples/sec
- each sample quantized, i.e., rounded
  - e.g., 2^8=256 possible quantized values
- each quantized value represented by bits
  - 8 bits for 256 values
- example: 8,000 samples/sec, 256 quantized values --> 64,000 bps
- receiver converts bits back to analog signal:
  - some quality reduction

Example rates

- CD: 1.411 Mbps
- MP3: 96, 128, 160 kbps
- Internet telephony: 5.3 kbps and up

Nén dữ liệu video

- video: sequence of images displayed at constant rate
  - e.g. 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- redundancy
  - spatial (within image)
  - temporal (from one image to next)

Examples:

- MPEG 1 (CD-ROM) 1.5 Mbps
- MPEG2 (DVD) 3-6 Mbps
- MPEG4 (often used in Internet, < 1 Mbps)

Research:

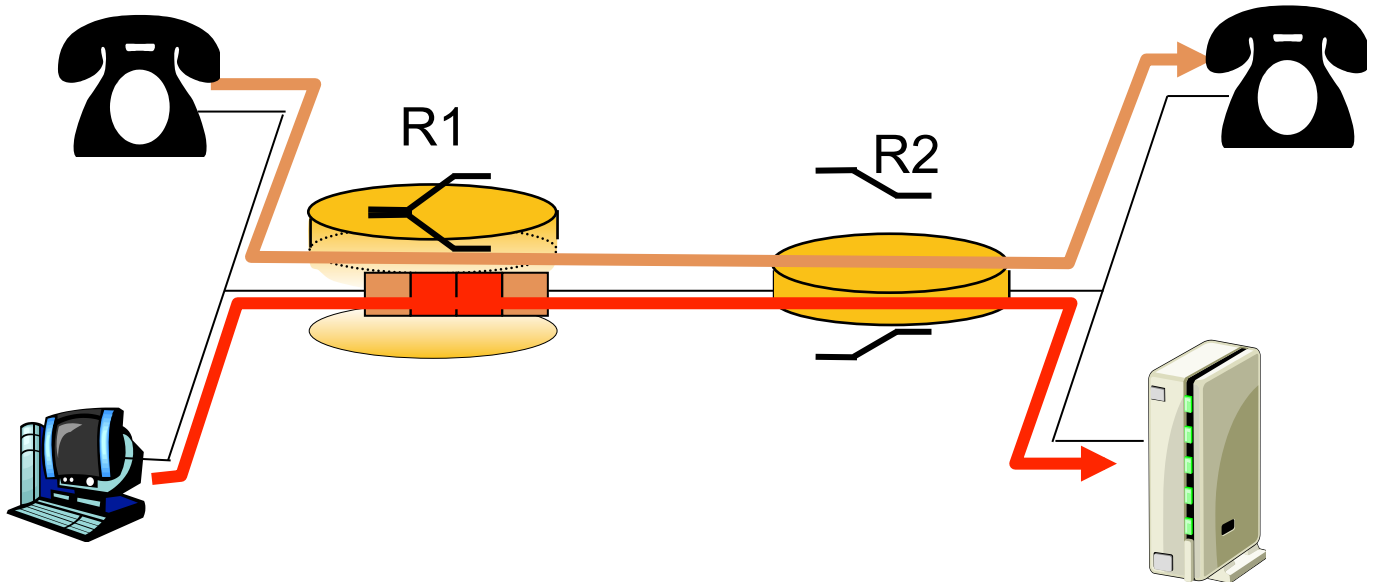
- layered (scalable) video
  - adapt layers to available bandwidth

4.2. Dòng dữ liệu và QoS

- Quality of Service (QoS):
  - bit-rate,
  - delay
  - e2e delay
  - jitter
  - round-trip delay
- Dựa trên tầng IP
  - Đơn giản, best-effort

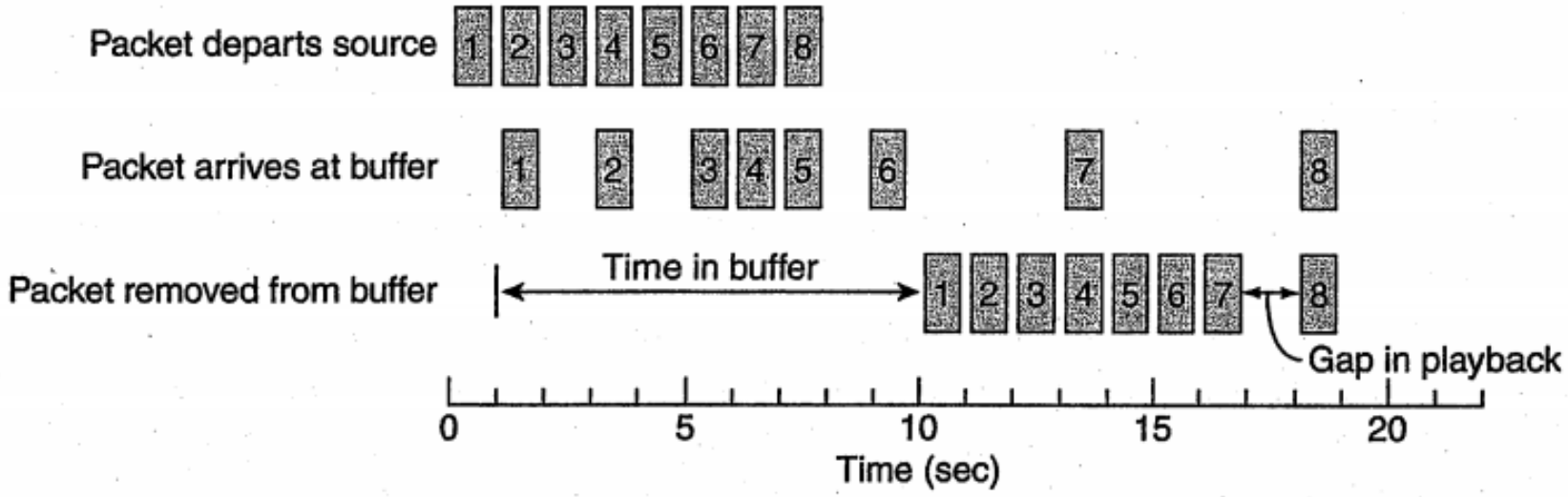
Thực thi QoS

- Differentiated services



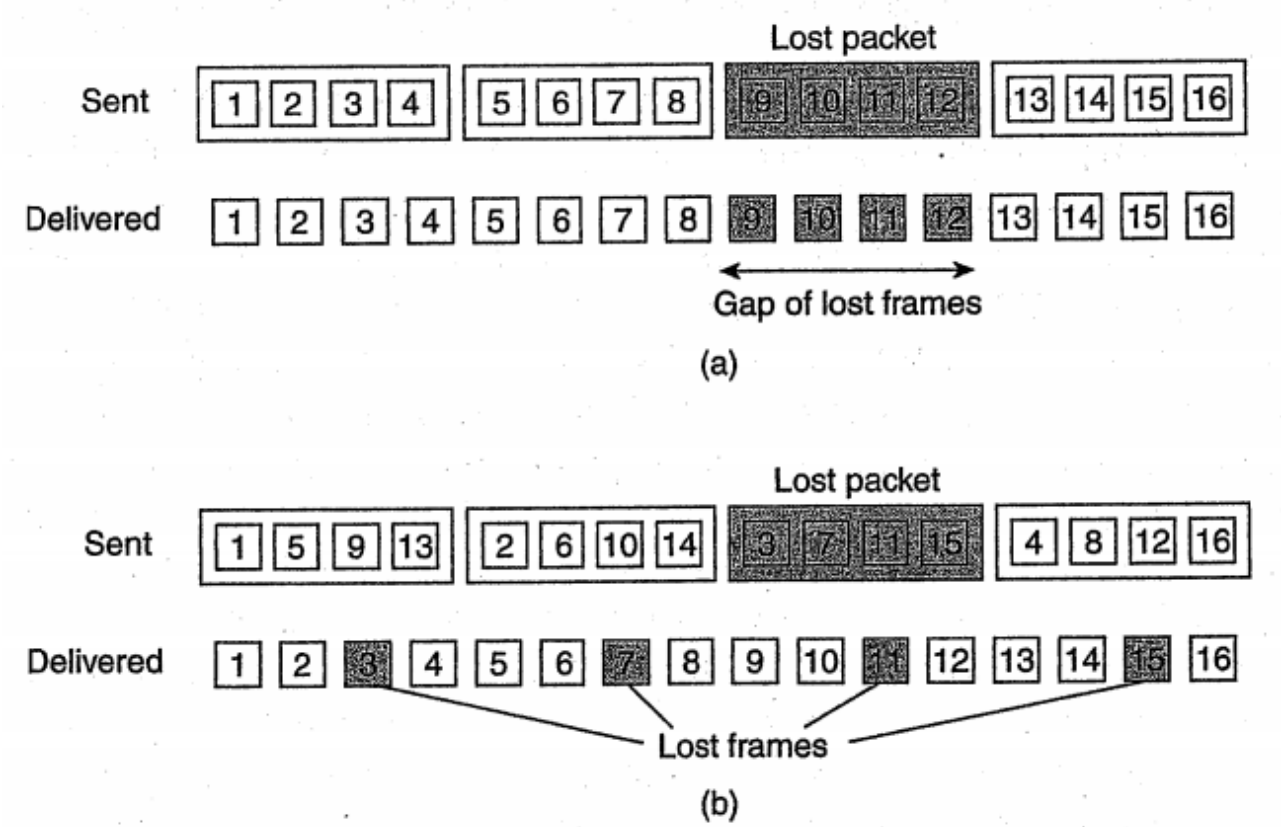
Thực thi QoS (cont.)

- Sử dụng bộ đệm để giảm jitter

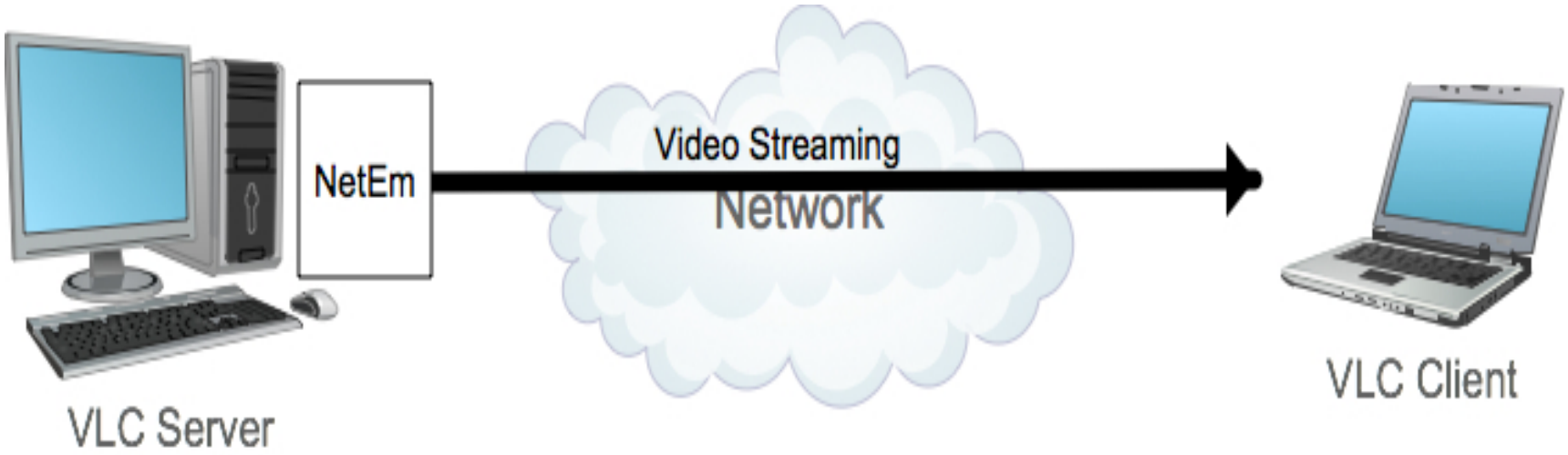


Thực thi QoS (cont.)

- Forward error correction (FEC)
  - Interleaved transmission



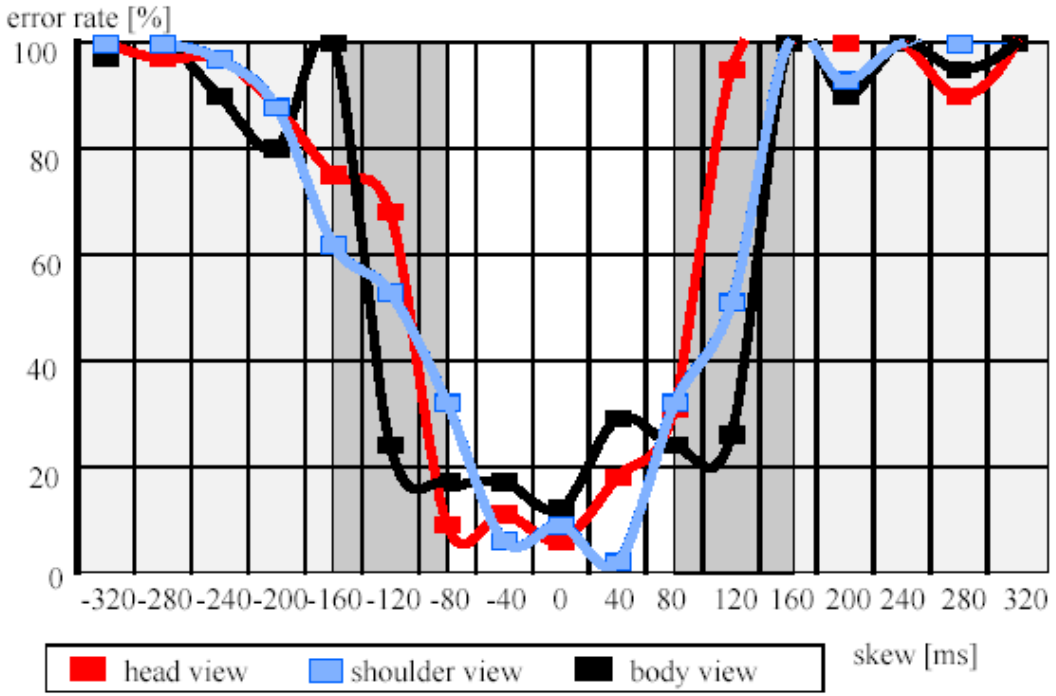
Labwork



4.3. Đồng bộ hóa dòng

- Nhu cầu đồng bộ hóa các dòng dữ liệu
- 2 kiểu đồng bộ:
  - Đồng bộ *dòng dữ liệu rời rạc* và *dòng dữ liệu liên tục.*
  - Đồng bộ 2 *dòng dữ liệu liên tục.*
- Dựa trên đơn vị dữ liệu

Lip Synchronization



Not tolerable

Not detectable

Not tolerable

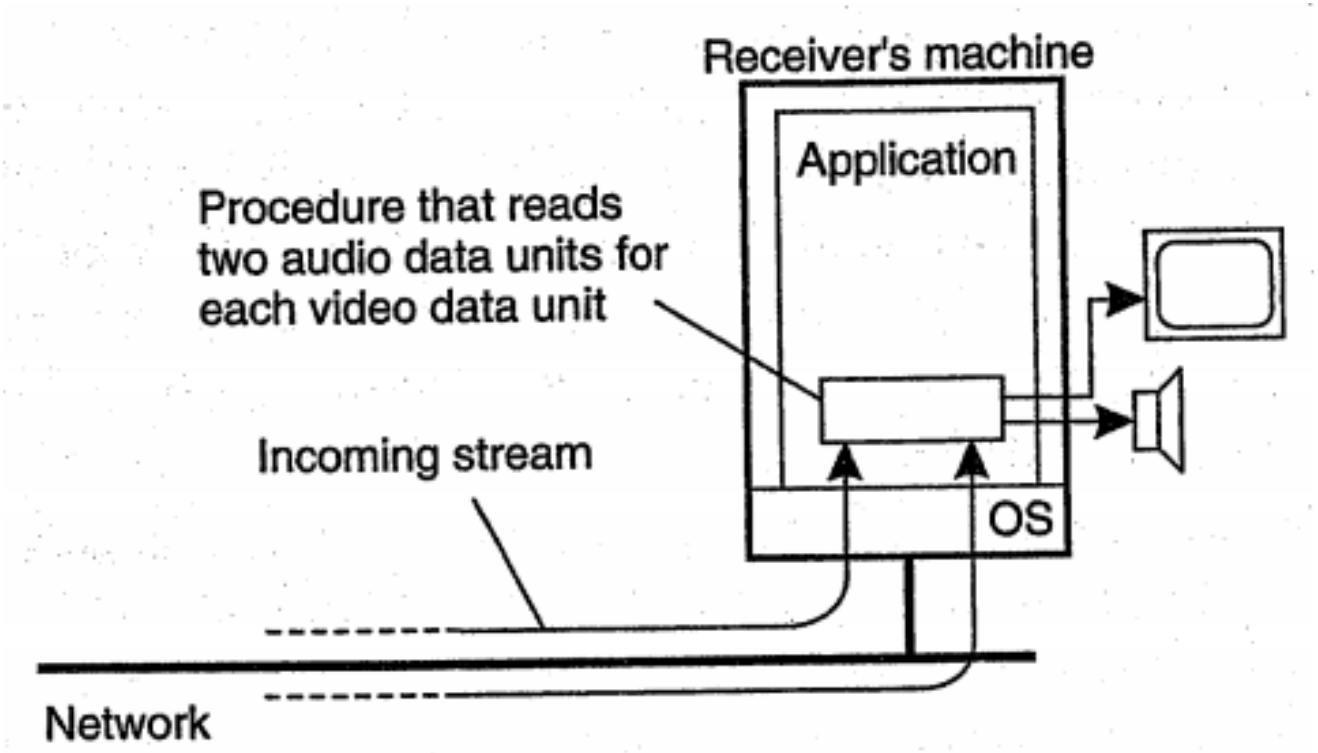
Tolerable

Tolerable

Cơ chế đồng bộ hóa

- Cơ chế cơ bản để đồng bộ 2 dòng
- Sự phân bố các cơ chế đó trong môi trường mạng

Đồng bộ hóa ở mức đơn vị dữ liệu



Đồng bộ hóa có hỗ trợ của những giao diện mức cao