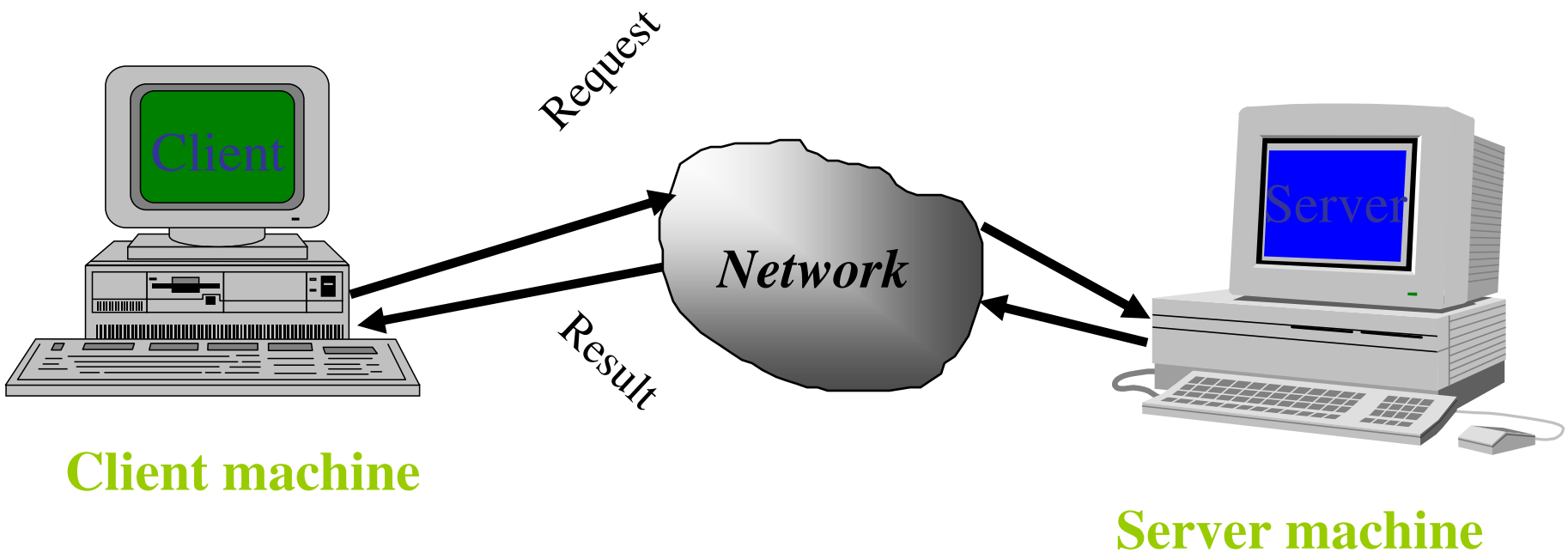
Elements of C-S Computing

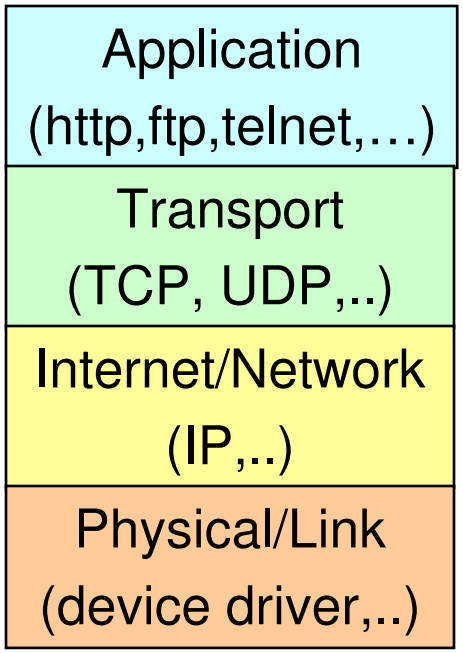
a client, a server, and network



Networking Basics

- Couche phsysique/liaison
  - Fonctionnalité de transmisssion du signal representant un sreal de données d'un ordinateur à un autre.
- couche Internet/Réseaux
  - Un packet de données est envoyé vesr un ordinateur distant (IP protocole)
- Couche de Transport
  - Fonctionnalités de livaraison des paquets a un processus spécifique sur un ordinateur spécifique.
  - Interface de progarmmation
    - Sockets
- Couche application
  - Protocole HTTP, FTP

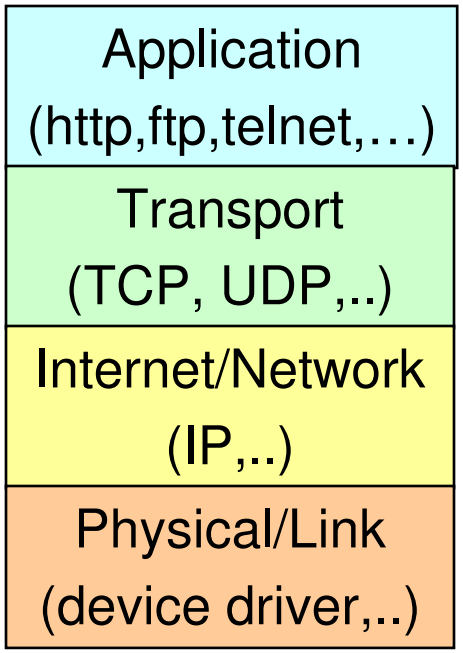
- TCP/IP Stack



Networking Basics

- TCP (Transmission Control Protocol) est un protocole de communication orienté conection qui offre une transfert fiable des paquets entre deux ordinateurs
- Exemple d'applications:
  - HTTP
  - FTP
  - Telnet

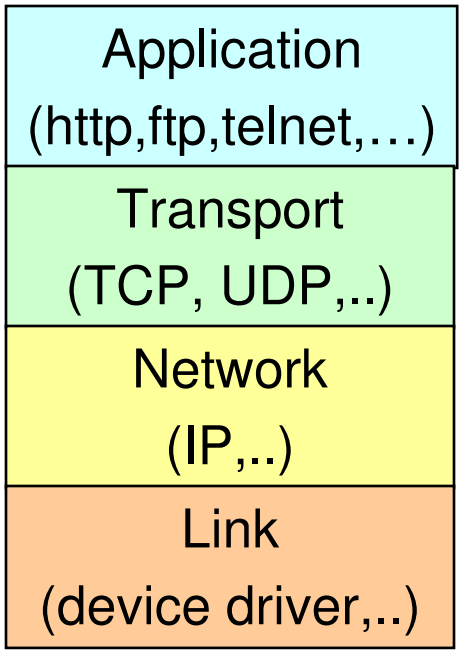
- TCP/IP Stack



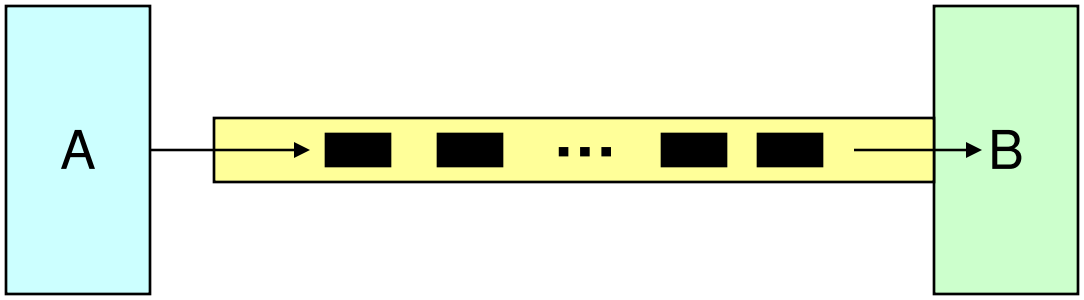
Networking Basics

- UDP (User Datagram Protocol) est un protocole non connecté qui envoie les paquets d'une facon independante sans garantie de livaraison.
- Similar to sending multiple emails/letters to a friends, each containing part of a message.
- Example applications:
  - Clock server
  - Ping

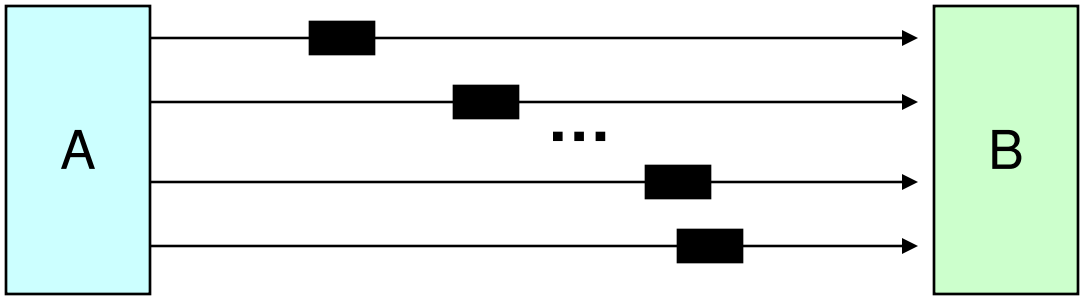
- TCP/IP Stack



TCP Vs UDP Communication



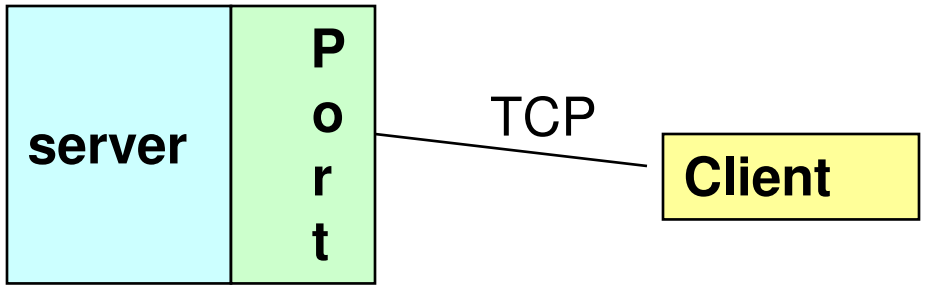
- Connection-Oriented Communication

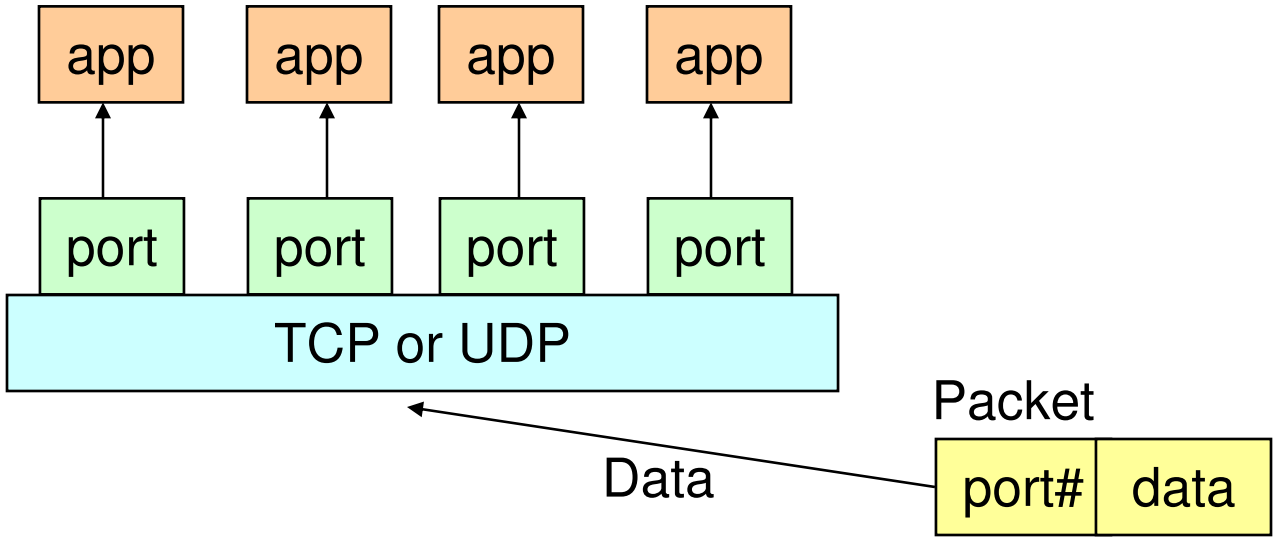


- Connectionless Communication

Ports

- TCP et UDP utilisent les ports pour envoyer les données récues vers un processus particulier.



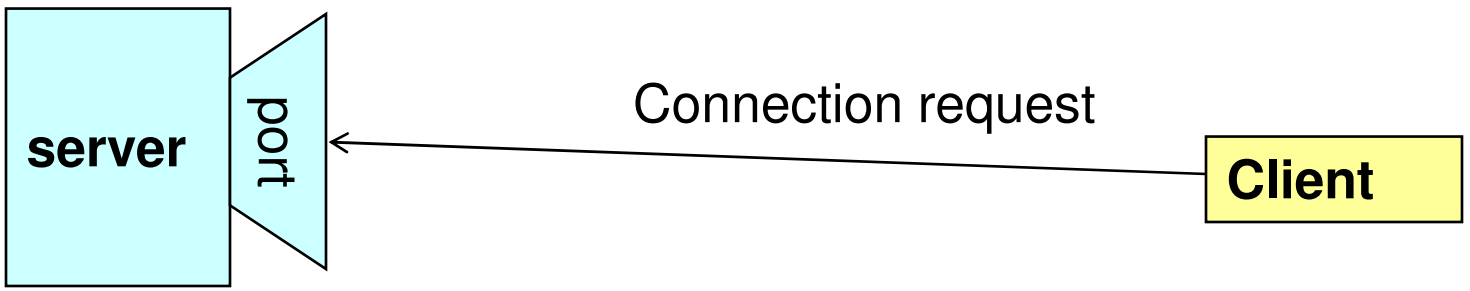


Understanding Ports

- Le port est representé par un nombre positif de 16 bits
- Ports bas utilisés par la SE 1 1024
- Ports de haut niveau utilisés par les utilisateur 1024 65536
- Quelques ports sont réservés pour des procotoles connus:
  - ftp 21/tcp
  - telnet 23/tcp
  - smtp 25/tcp
  - login 513/tcp

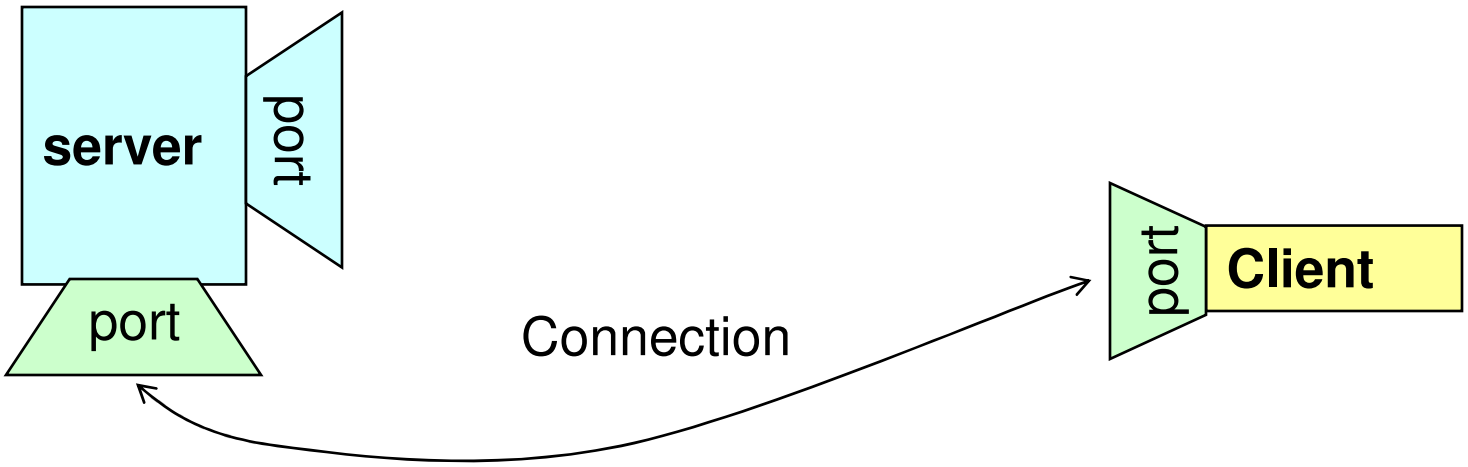
Socket Communication

- Un serveur (programme) est lancé sur un ordinateur spécifique et possède un socket associé à un port particulier. Le socket permet au client d'attendre et d'écouter les clients qui veulent établir des connection.



Socket Communication

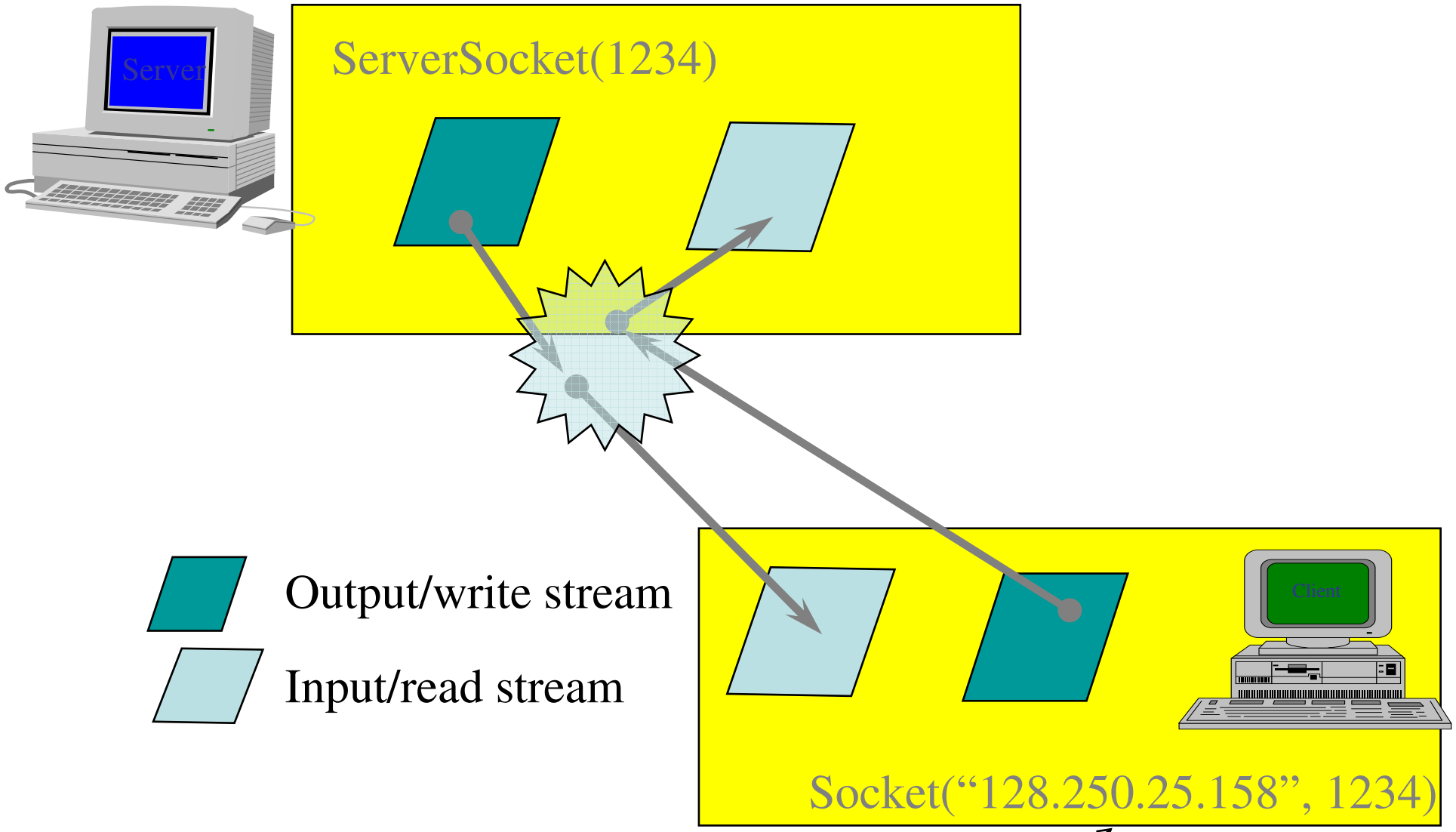
- Si tout va bien le serveur accepte la connection, et crée un nouveau socket associé à un autre port. Le premier socket reste à l'écoute des connections des clients voulant établir des connections.



Sockets and Java Socket Classes

- Un socket est l'extrémité d'un canal de communication entre deux processus.
- Un socket est associé à un port, la couche TCP permet alors d'identifier l'application à laquelle sont envoyés les données.
- Java's .net package provides two classes:
  - Socket – for implementing a client
  - ServerSocket – for implementing a server

Java Sockets

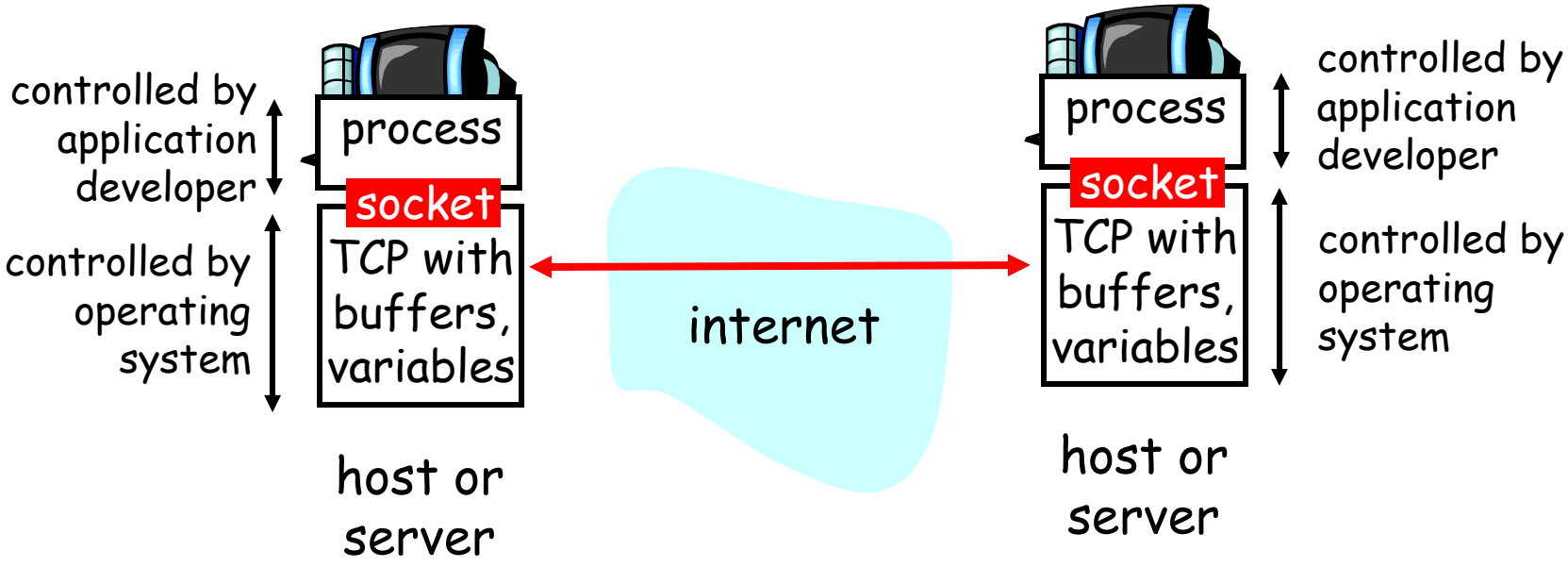


It can be host_name like "mandroo.cs.mu.oz.au"

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

TCP service: reliable transfer of **bytes** from one process to another



Socket programming *with TCP*

Serveur

- Le processus serveur doit etre lancé tout d'abord
- Le serveur doit créer un socket pour chaque demande connection.
  - Permettre de parler avec plusieurs client en méme temps.
  - Distinguer les clients

Client:

- Crée un socket TCP local.
- Spécifier l'adresse IP et le numéro de port du processus serveur.
- Quand le crée résussit à créer un socket une connection TCP est établit entre les clietn et les serveur.

application viewpoint

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

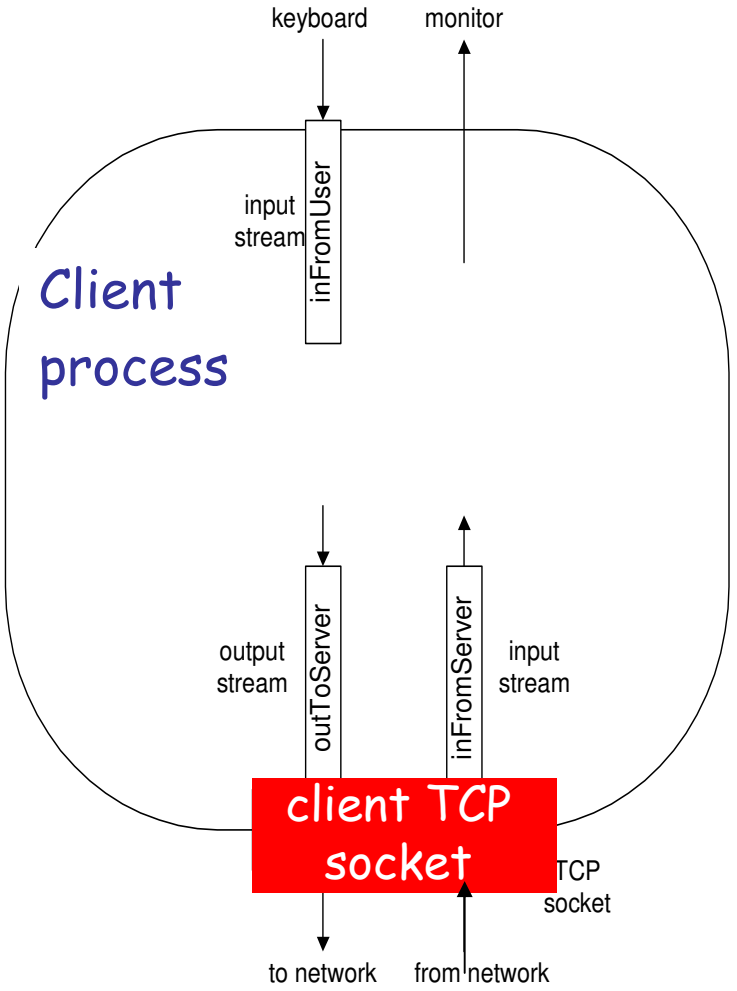
Stream jargon

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or socket.
- An output stream is attached to an output source, e.g., monitor or socket.

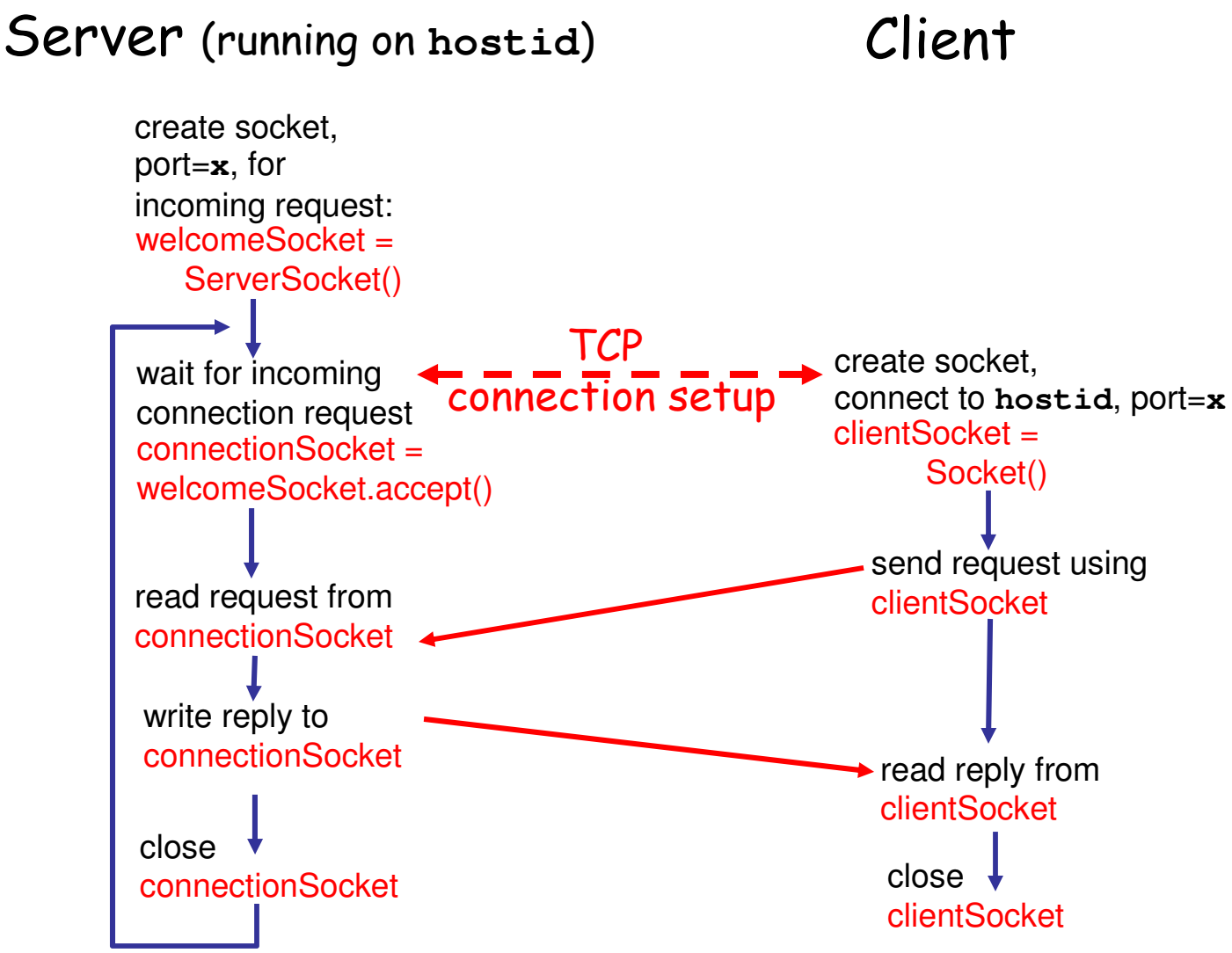
Socket programming with TCP

Example client-server app:

1) Le client lit le texte à partir de l'entrée standard (**`inFromUser`** stream) , et envoie au serveur à travers le socket (**`outToServer`** stream)
2) Le serveur lit les données à partir du socket
3) Convertit en majuscules et renvoie les données au client.
4) Le client lit les données modifiées à partir du socket. (**`inFromServer`** stream)



Client/server socket interaction: TCP



Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Example: Java client (TCP), cont.

Create input stream attached to socket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();
```

Send line to server

```
outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

Example: Java server (TCP)

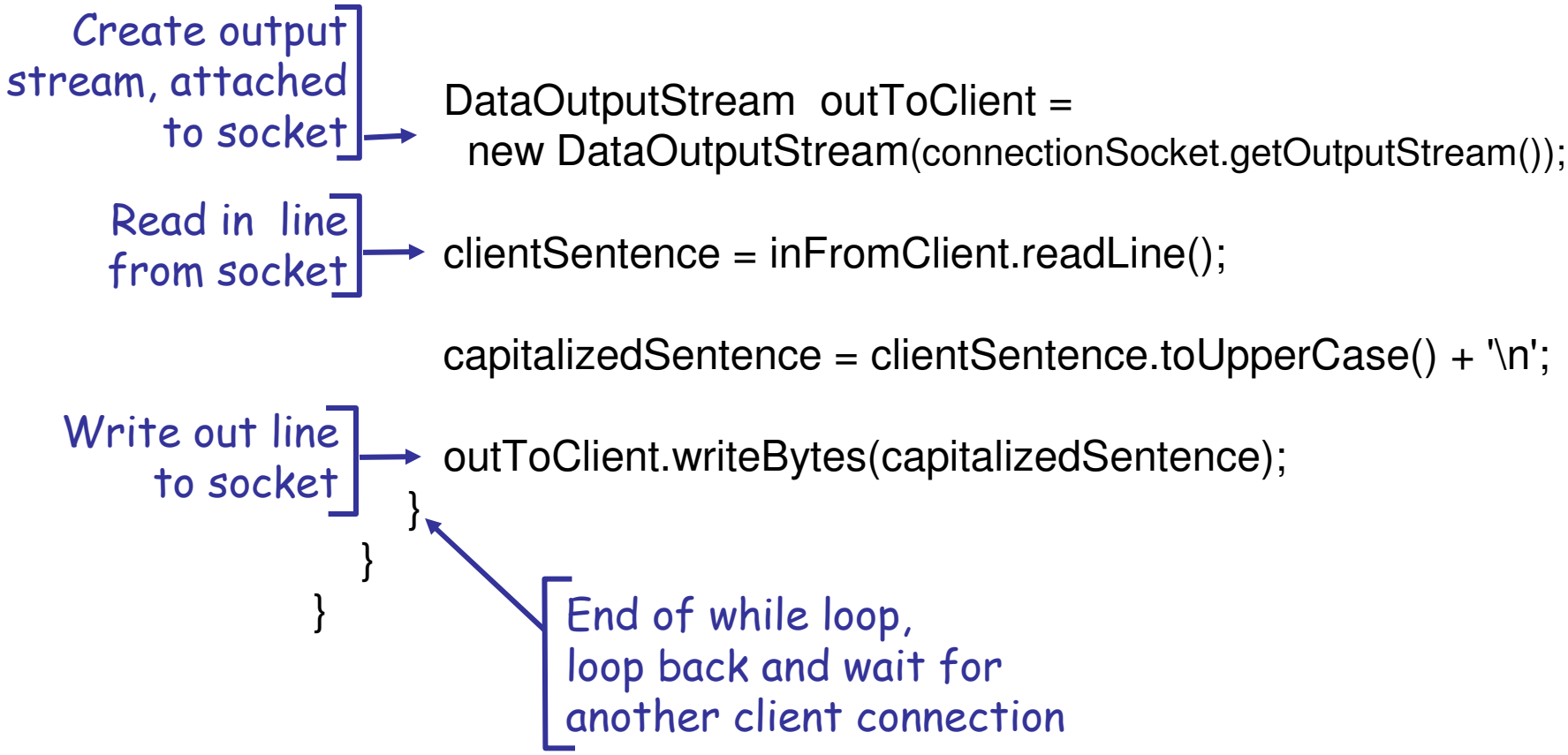
```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

        Socket connectionSocket = welcomeSocket.accept();

       BufferedReader inFromClient =
         new BufferedReader(new
         InputStreamReader(connectionSocket.getInputStream()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Example: Java server (TCP), cont



TCP/IP Sockets in Java: Practical Guide for Programmers

- Kenneth L. Calvert
- Michael J. Donahoo

TCP Client/Server Interaction

Server starts by getting ready to receive client connections…

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

ServerSocket servSock = new ServerSocket(servPort);

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

```
for (;;) {
    Socket clntSock =   servSock.accept();
```

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

Server is now blocked waiting for connection from a client

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

Later, a client decides to talk to the server...

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

Socket socket = new Socket(server, servPort);

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

```
OutputStream out = socket.getOutputStream();
out.write(byteBuffer);
```

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

Socket clntSock = servSock.accept();

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

```
InputStream in = clntSock.getInputStream();
recvMsgSize = in.read(byteBuffer);
```

Client

1. Create a TCP socket
2. Communicate
3. Close the connection

Server

1. Create a TCP socket
2. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

TCP Client/Server Interaction

close(sock);

Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

close(clntSocket)

Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
   a. Accept new connection
   b. Communicate
   c. Close the connection

Most important classes/methods

- java.net.Socket
  - Socket(InetAddress *addr*, int *port*);
    - *Créer un socket avec l'adresse addr et le numero de port*
  - InputStream getInputStream();
    - returns an instance of InputStream for getting info from the implicit Socket object
  - OutputStream getOutputStream();
    - returns an instance of OutputStream for sending info to implicit Socket object.
  - close();
    - close connection to implicit socket object, cleaning up resources.

Important classes, cont.

- java.net.ServerSocket
  - ServerSocket(int *port*);
    - Permet à un programme d'attendre des connexion sur un port particulier.
  - Socket accept();
    - Bloque jusqu'à la demande d'une connection d'un socket distant. Quand la connection est établit une nouvelle instance de socket est créée.

Important class, cont.

- java.net.InetAddress
  - static InetAddress getByName(String *name*)
    - Etant donné un nom d'adresse *name*, cette fonction renvoie un objet InetAddress, cet objet représente le nom (essentiellement représente le nom d'une machine et l'adresse IP);
  - static InetAddress getLocalHost()
    - renvoie un objet InetAddress associé à l'adresse local.

Java API for UDP Programming

- Java API provides datagram communication by means of two classes.
  - DatagramPacket
    - | Msg | length | Host | serverPort |
  - DatagramSocket

Socket programming *with UDP*

UDP: pas de connection entre le client et le serveur

- La source attache explicitement l'adresse IP et le numéro de port à chaque paquet.
- Le serveur doit extraire l'addresse IP et port de chaque paquet.

UDP: les packet peut etre perdues ou récues sans l'ordre.

application viewpoint

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

Pourquoi utiliser un protocole non fiable.

- La fiabilité a un prix, la vitesse. L'etablissement et la fermeture d'une connection peut prendre du temps.

- Par exemple dans une emission audio ou vidéo en temps réel, les paquets perdus en UDP n'ont pas un grand effet, alors qu'on TCP chaque paquet perdu doit etre retransmis => delai

implementation en java :UDP

- L'implementation en java du protocole UDP est basée sur deux principales classes:
  - `DatagramPacket` et `DatagramSocket`.
  - La classe `DatagramPacket permet d'organiser les données binaires dans des pacquets UDP`.
  - La classe `DatagramSocket`. Permet d'envoyer et de recevoire des paquets de differentes sources.

Différence entre l'implémentation TCP/UDP en java

- En UDP la notion de socket serveur (SocketServer) n'existe pas. On peut utiliser le meme type de socket pour recevoire et envoyer des paquets.
- Les sockets TCP permettent de traiter les connections comme des Stream. En UDP nous travaillons avec des paquets. Given two packets, there is no way to determine which packet was sent first and which was sent second.
- En UDP un DatagramSocket peut envoyer et recevoire à partir des sites indépendants. Un socket n'est pas dédié à une connection particuliere.

La DatagramPacket Classe

- La capacité théorique d'un dtagramme UDP est de 65,507 bytes, cepe,dant en pratique la limite actuelle est de 8,192 bytes (8K).
- La classe DatagramSocket possède deux constructeurs un pour envoyer et un pour recevoire.

Constructeurs pour la reception de données

- Pour recevoire des données à partir du réseau:
- public DatagramPacket(byte[] buffer, int length)
- public DatagramPacket(byte[] buffer, int offset, int length)

constructeurs pour envoyer des datagram

- Deux constructeurs pour envoyer les données à travers le réseau:
- public DatagramPacket(byte[] data, int length,InetAddress destination, int port)
- public DatagramPacket(byte[] data, int offset, int length, InetAddress destination, int port)

The get methods

- **public InetAddress getAddress( )**
- Cette méthode renvoie l'objet InetAddress qui contient l'adresse du site distant si le datagramme est récue à partir du réseau.
- **public int getPort( )**
- Renvoie le numéro de port du processus distant.
- **public byte[ ] getData( )**
- Cette méthode renvoie un tableau de bytes contenant dans un Datagram.

The DatagramSocket Class

- If you're writing a client, you don't care what the local port is, so you call a constructor that lets the system assign an unused port (an anonymous port). This port number is placed in any outgoing datagrams and will be used by the server to address any response datagrams.
- If you're writing a server, clients need to know on which port the server is listening for incoming datagrams; therefore, when a server constructs a DatagramSocket, it must specify the local port on which it will listen. However, the sockets used by clients and servers are otherwise identical:

constructor

- **La classe DatagramSocket possède deux constructeurs:**
- **public DatagramSocket( ) throws SocketException**
- Ce constructeur crée un socket utilisant un port choisit par le système d'exploitation.

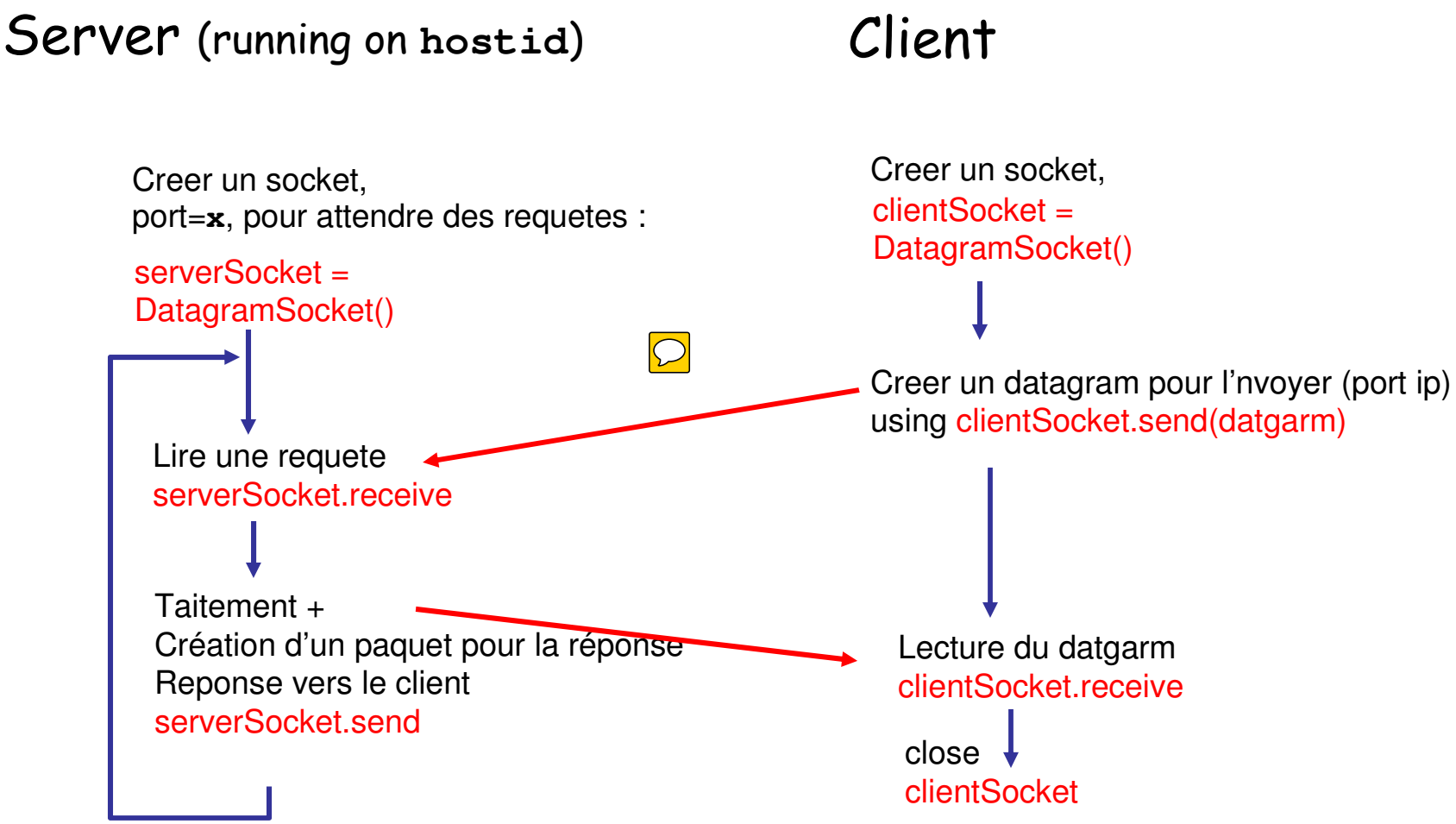
  exemple le cas d'un socket client:

  DatagramSocket client = new DatagramSocket( );

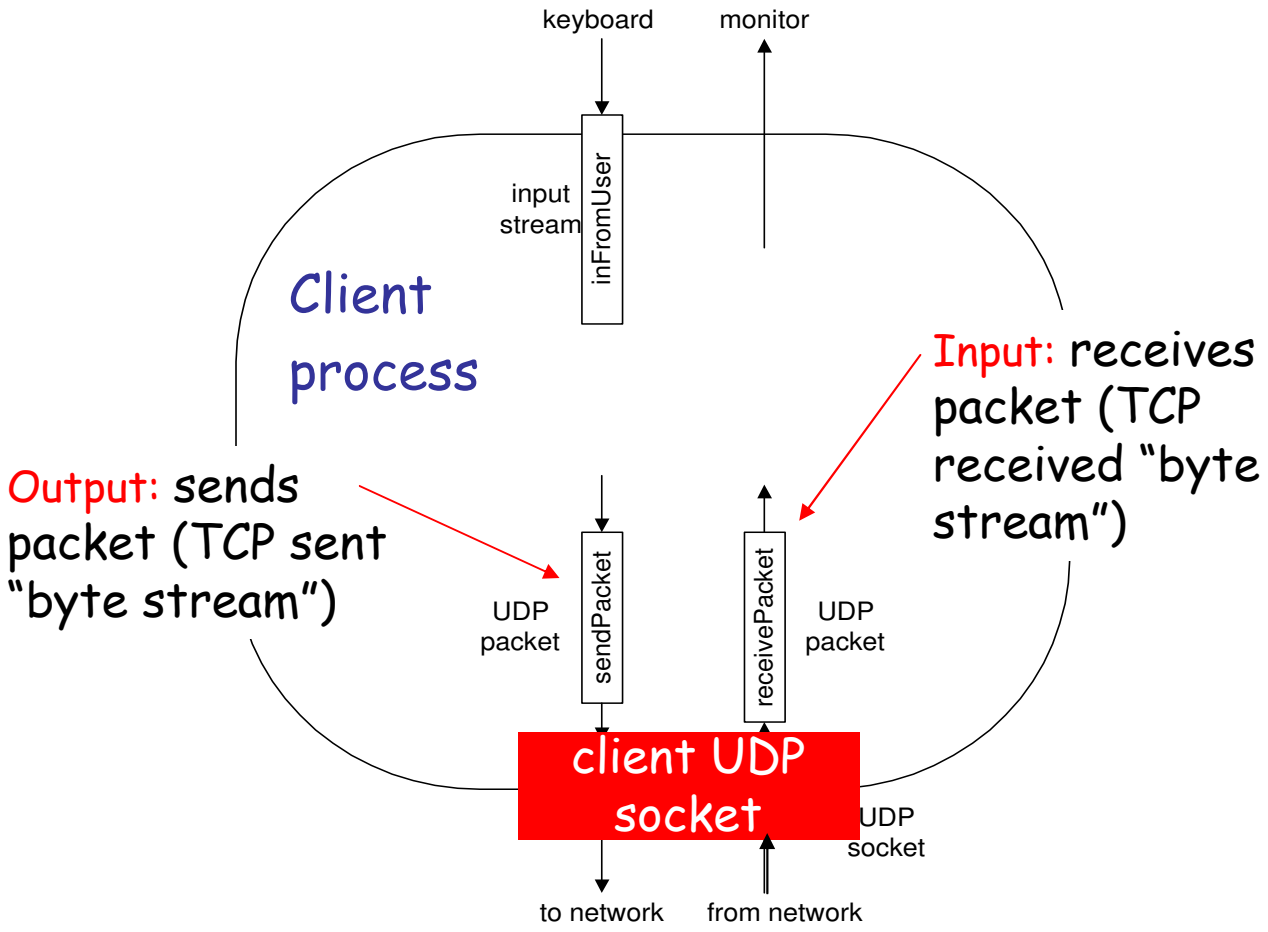
  on pourra trouver le port local avec getLocalPort( )
- Le deuxième constructeur
  - public DatagramSocket(int port) throws SocketException
- Crée un socket qui écoute sur un port particulier et il est utiliser dans le cas d'un serveur.

- Java 1.2 adds our methods that let you choose which host you can send datagrams to and receive
- datagrams from, while rejecting all others' packets.
- **13.3.3.1 public void connect(InetAddress host, int port)** // **Java 1.2**
- The connect( ) method doesn't really establish a connection in the TCP sense.
- However, it does specify that the DatagramSocket will send packets to and receive
- packets from only the specified remote host on the specified remote port. Attempts to
- send packets to a different host or port will throw an IllegalArgumentException.
- Packets received from a different host or a different port will be discarded without an
- exception or other notification.
- A security check is made when the connect( ) method is invoked. If the VM is
- allowed to send data to that host and port, then the check passes silently. Otherwise, a
- SecurityException is thrown. However, once the connection has been made,
- send( ) and receive( ) on that DatagramSocket no longer make the security
- checks they'd normally make.
- **13.3.3.2 public void disconnect( )** // **Java 1.2**
- The disconnect( ) method breaks the "connection" of a connected
- DatagramSocket so that it can once again send packets to and receive packets from
- any host and port.
- **13.3.3.3 public int getPort( )** // **Java 1.2**
- If and only if a DatagramSocket is connected, the getPort( ) method returns the
- remote port to which it is connected. Otherwise, it returns -1.
- **13.3.3.4 public InetAddress getInetAddress( )** // **Java 1.2**
- If and only if a DatagramSocket is connected, the getInetAddress( ) method
- returns the address of the remote host to which it is connected. Otherwise, it returns
- null.

L'interaction client/serveur enUDP



Example: Java client (UDP)



Example: Java client (UDP)

Create input stream

Create client socket

Translate hostname to IP address using DNS

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

class UDPClient {
    public static void main(String args[]) throws Exception
    {

BufferedReader inFromUser =
  new BufferedReader(new InputStreamReader(System.in));

DatagramSocket clientSocket = new DatagramSocket();

InetAddress IPAddress = InetAddress.getByName("hostname");

byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];

String sentence = inFromUser.readLine();

sendData = sentence.getBytes();
```

Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
   new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
    new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}

}
```

Example: Java server (UDP)

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

class UDPServer {
  public static void main(String args[]) throws Exception
    {
      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {
          DatagramPacket receivePacket =
            new DatagramPacket(receiveData, receiveData.length);
          serverSocket.receive(receivePacket);
```

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

Example: Java server (UDP), cont

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

                String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress,
                      port);

serverSocket.send(sendPacket);
    }
  }

}
```

Get IP addr port #, of sender

Create datagram to send to client

Write out datagram to socket

End of while loop, loop back and wait for another datagram

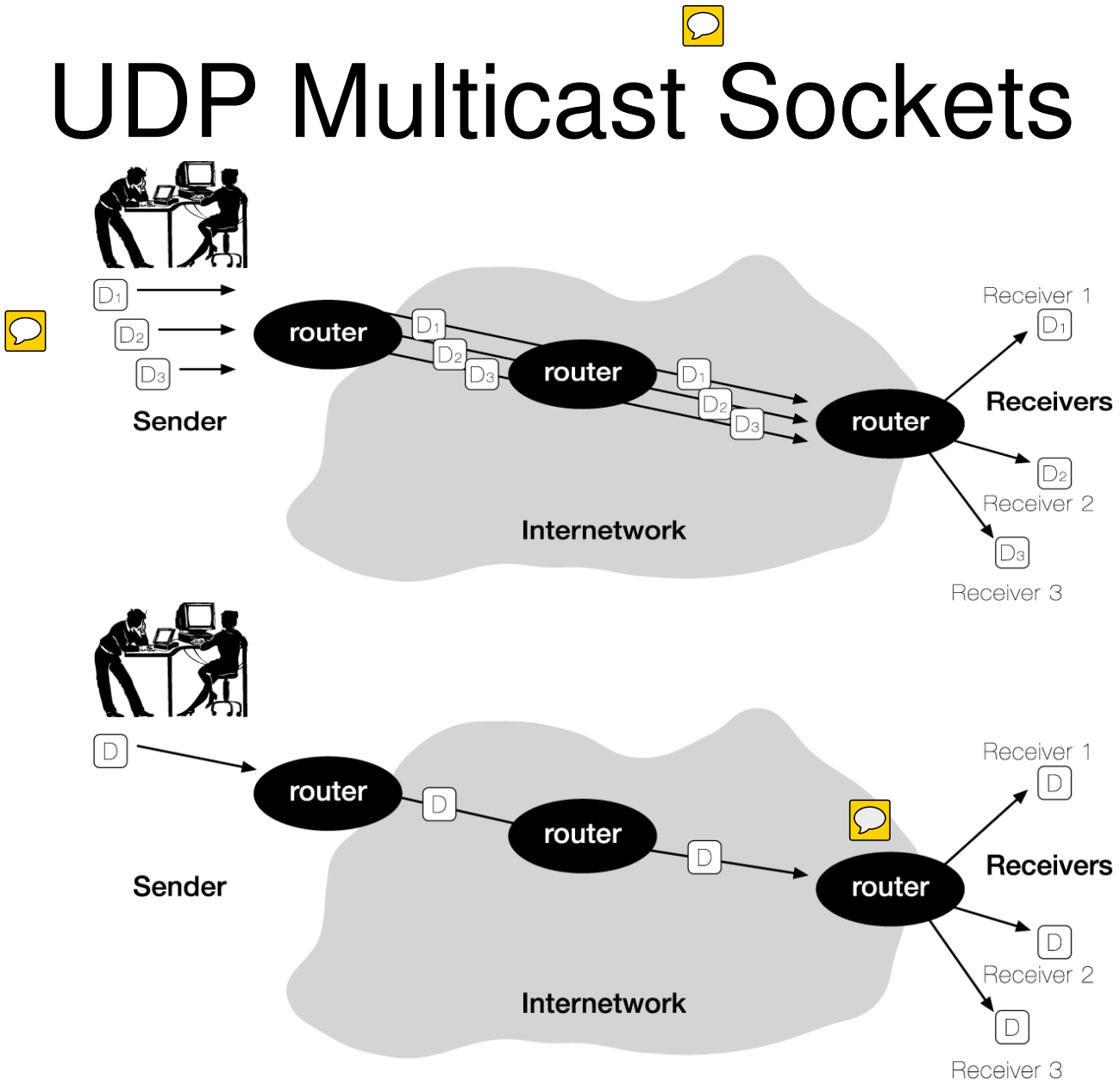
UDP Client: Sends a Message and Gets reply

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

UDP Sever: repeatedly received 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();}
    }
}
```

UDP Multicast Sockets



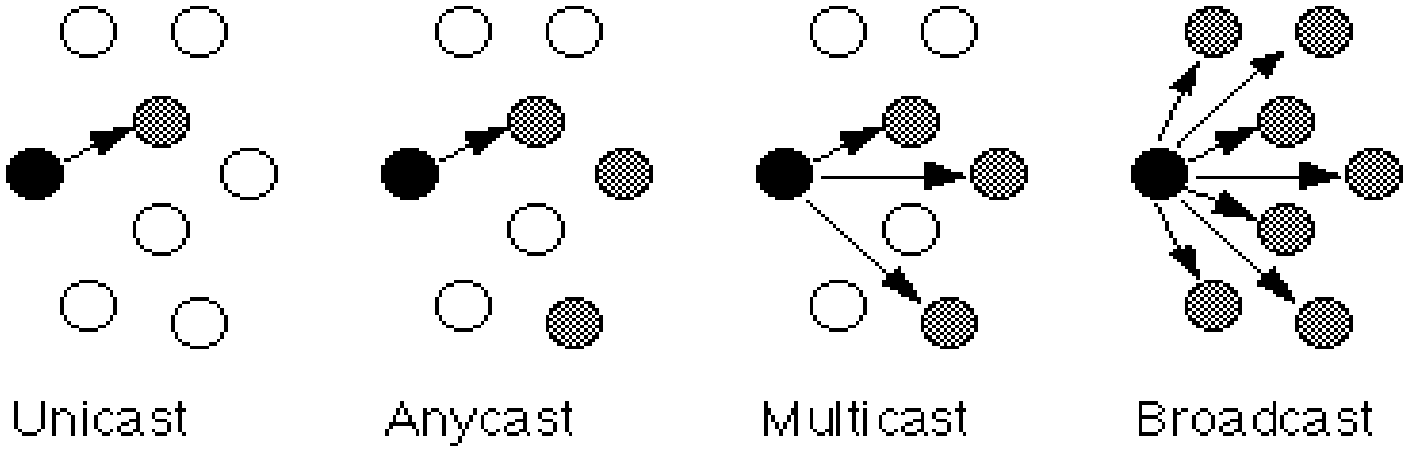
Concepts de bases Multicast

- Les examples précedents TCP/UDP sont tous en mode unicast
- Unicast: communication point à point
- Broadcast: les paquets sont envoyés à tous le monde

  – IP permet le broadcasting mais il est strictement limité.

- Multicast: envoyer des paquets vers plusieurs sites mais pas à tout le monde.

Multicast Basic Concepts



Multicast Basic Concepts

- Besoin du multicast :
  – un seul e-mail envoyé à 6 millions d'addresses
  – une emission temps réel envoyé à 6 millions d'utilisateur
  – Internet crash???
  – Video conferencing: envoyer un stream audio-video à un groupe d'utilisateur.
  – DNS routers
  – News group

Multicast Basic Concepts

- Multicast: principe
  – comme un réunion publique
  – les gens peuvent entrer et sortir selon leurs
  – chacun peut envoyer un message à tout les abonnés du groupe.
  – les gens qui ne sont pas du groupe ne sont pas consérnés

Multicast Examples

- Video conferencing
- DNS routers
- News group
- Multiplayer games
- Distributed file systems
- Massively parallel computing
- Database replication
- Name services
- Directory services

Multicast Example

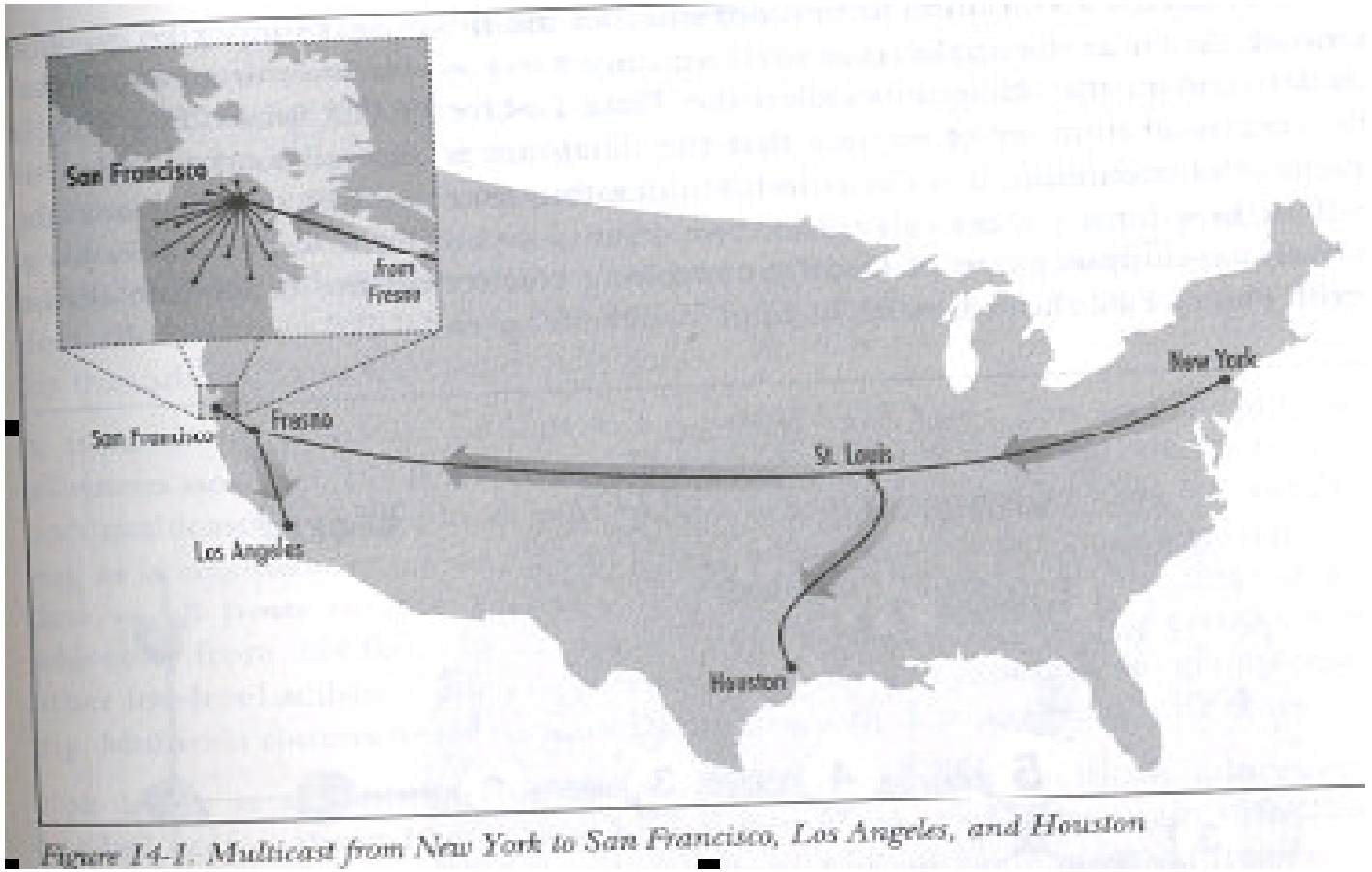


Figure 14-1. Multicast from New York to San Francisco, Los Angeles, and Houston

Multicast Basic Concepts

- La plus grande partie de travail doit etre faite par les routeurs.
- Une application envoie tout simplement un datagram possédant une adresse de multicast. Les routeurs assurent que tous les paquets sont livrés à tous les élements du groupe.

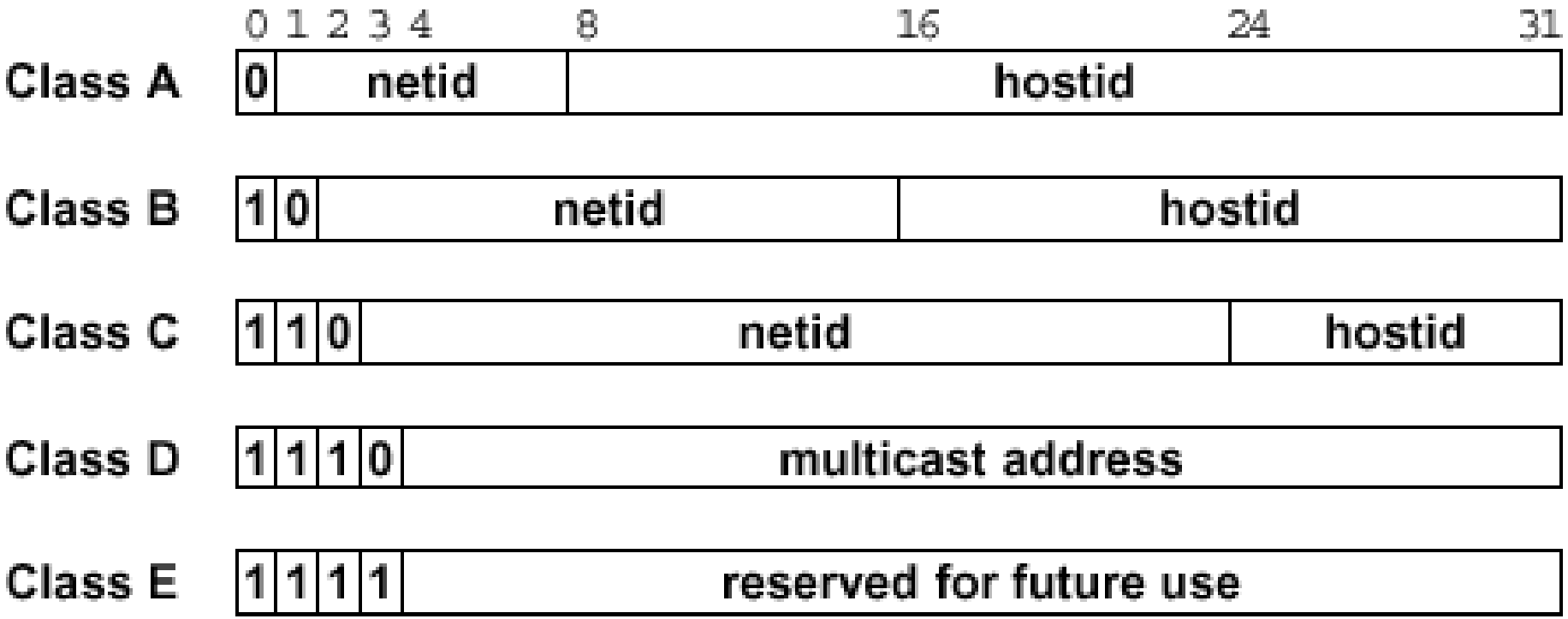
Multicast Basic Concepts

- TTL: time to live in IP header
  - TTL est le maximum nombre de routeurs qu'un paquets peut traverser. Utiliser pour limiter le passage des paquets.
- Multicast: UDP protocol

Multicast Address and Groups

- Une adresse multicast est une adresse d'un groupe de machine appéllés groupe de multicast.
- C'est une adresse IP:
  - Classe D
  - appartenant à 224.0.0.0 to 239.255.255.255
  - Commencant par les 4 bits: 1110
- Comme une autre adresse une multicast adresse peut avoir un nom de domaine.
  - 224.0.1.1 = ntp.mcast.net (network time protocol)

IP addresses



Multicast Address and Groups

- A multicast group is a set of Internet hosts that share a multicast address
- Any data sent to the multicast address is relayed to all the members of the group
- Membership in a multicast group is open; hosts can enter or leave the group at any time
- Groups can be either permanent or transient
  - Permanent groups have assigned address that remain constant
  - Most multicast groups are transient and exist only as long as they have members.

Multicast Address and Groups

- Create a multicast group
  - Pick an random address from 225.0.0.0 to 238.255.255.255
  - http://www.iana.org/assignments/multicast-addresses
- A number of multicast addresses have been assigned for special purposes.
  - all-systems.mcast.net (224.0.0.1) is a multicast group that includes all systems that support multicasting on local subnet
  - This group is commonly used for local testing
  - Also for local testing experiment.mcast.net (224.0.1.20)

Multicast Address and Groups

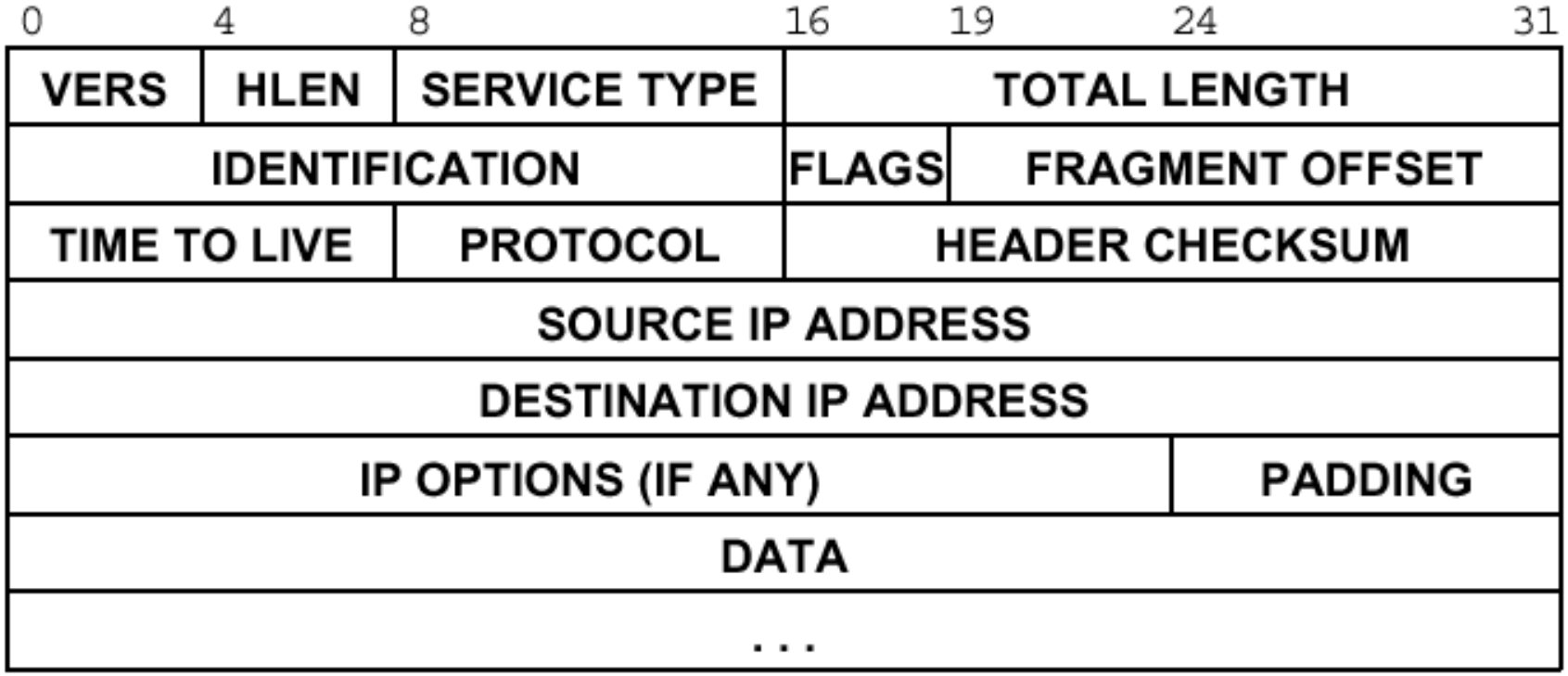
- A number of multicast addresses have been assigned for special purposes. (cont.)
  - (224.0.0.0~ 224.0.0.255) are reserved for routing protocols (gateway discovery …)
  - Multicast routers never forward datagrams with destinations in 224.0.0.0~ 224.0.0.255
  - IANA is responsible for handing out permanent multicast addresses
  - About 10,000 have been assigned
  - See Table 14.1 for permanent multicast addresses
  - Still have 248 Million class D addresses can be used.

Clients and Servers

- When a host wants to send data to a multicast group, it puts that data in multicast datagrams (UDP datagrams with an IP address in class D)
- Most multicast data is either audio or video or both.(Small data lost is fine.)
- Multicast data is sent via UDP
- UDP can be as much as three times faster than TCP

Datagram Format

- TTL: time to live
  - One byte



TTL

- Routers and hosts must decrement the *TIME TO LIVE* field by one and remove the datagram from the internet when its time expires.
- In practice, the TTL acts a “hop limit” rather than an estimate of delays.
- Two uses:
  - It guarantees that datagrams cannot travel around an internet forever.
  - Source might want to intentionally limit the journey of the packet.

TTL

- TTL: the number of hops
- Each time a packet passes through a router, its TTL value is decremented by at least one
  - Some routers may decrement the TTL by two or more.
- When the TTL reaches zero, the packet is discarded.
- All packets would eventually be discarded
- TTL may prevent mis-configured routers from sending packets back and forth to each other indefinitely

TTL

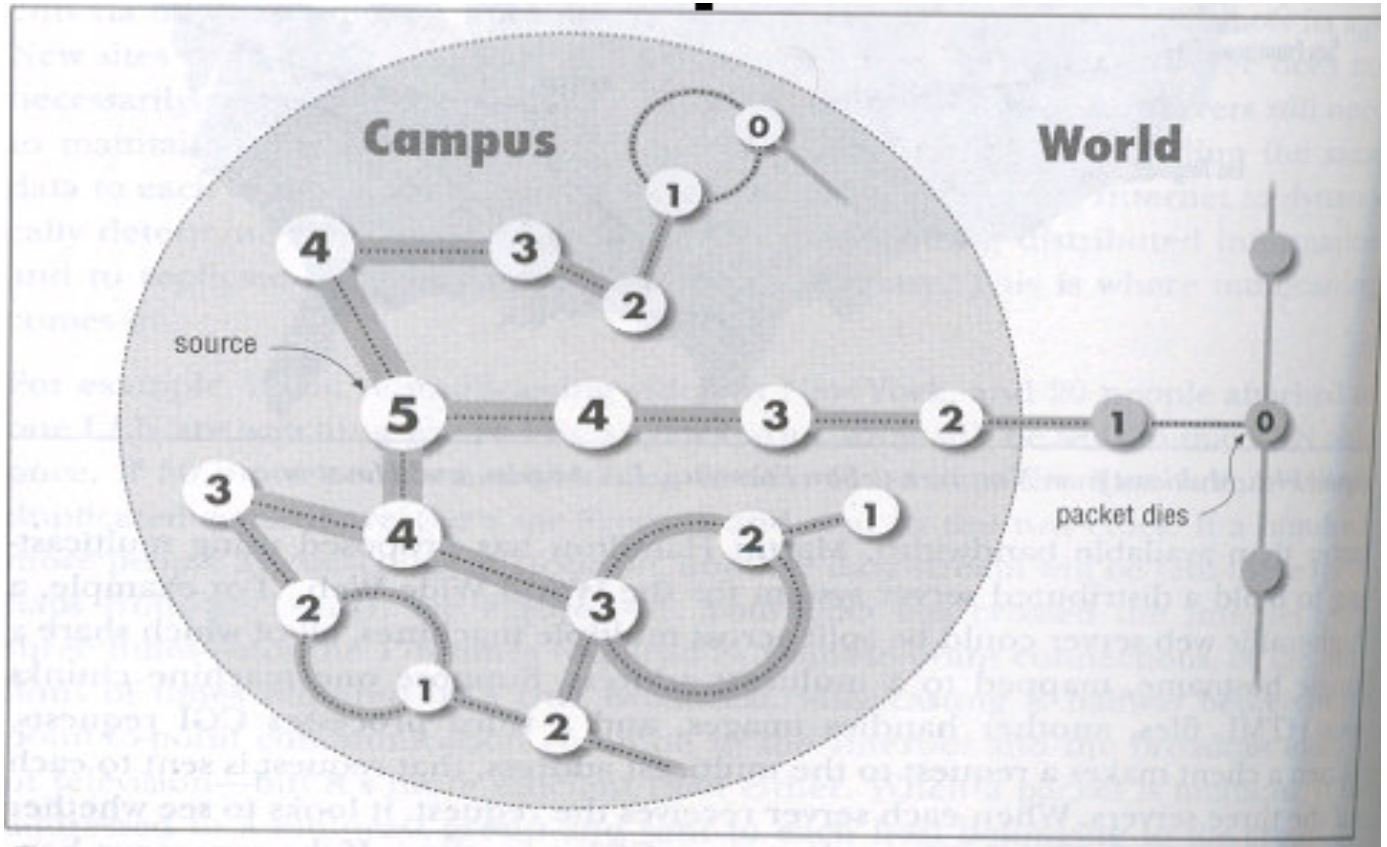


Figure 14-2. Coverage of a packet with a TTL of 5

TTL

- In IP multicasting, TTL is used to limit the multicast geographically.
  - TTL = 0: local host
  - TTL = 1: local subnet
  - TTL = 16: local campus or organization
  - TTL = 32: US backbone
  - TTL = 48: US
  - TTL = 64: North America
  - TTL = 128: high bandwidth sites worldwide
  - TTL = 255: All sites worldwide

Router and Routing

- With multicasting:
  - a multicast socket sends one stream of data over the Internet to the clients' router.
  - The router duplicates the stream and sends it to each of the clients.
- Without multicasting:
  - The server sends four separate but indintical stream of data to the router
  - The router each of the stream to a client.

Router and Routing

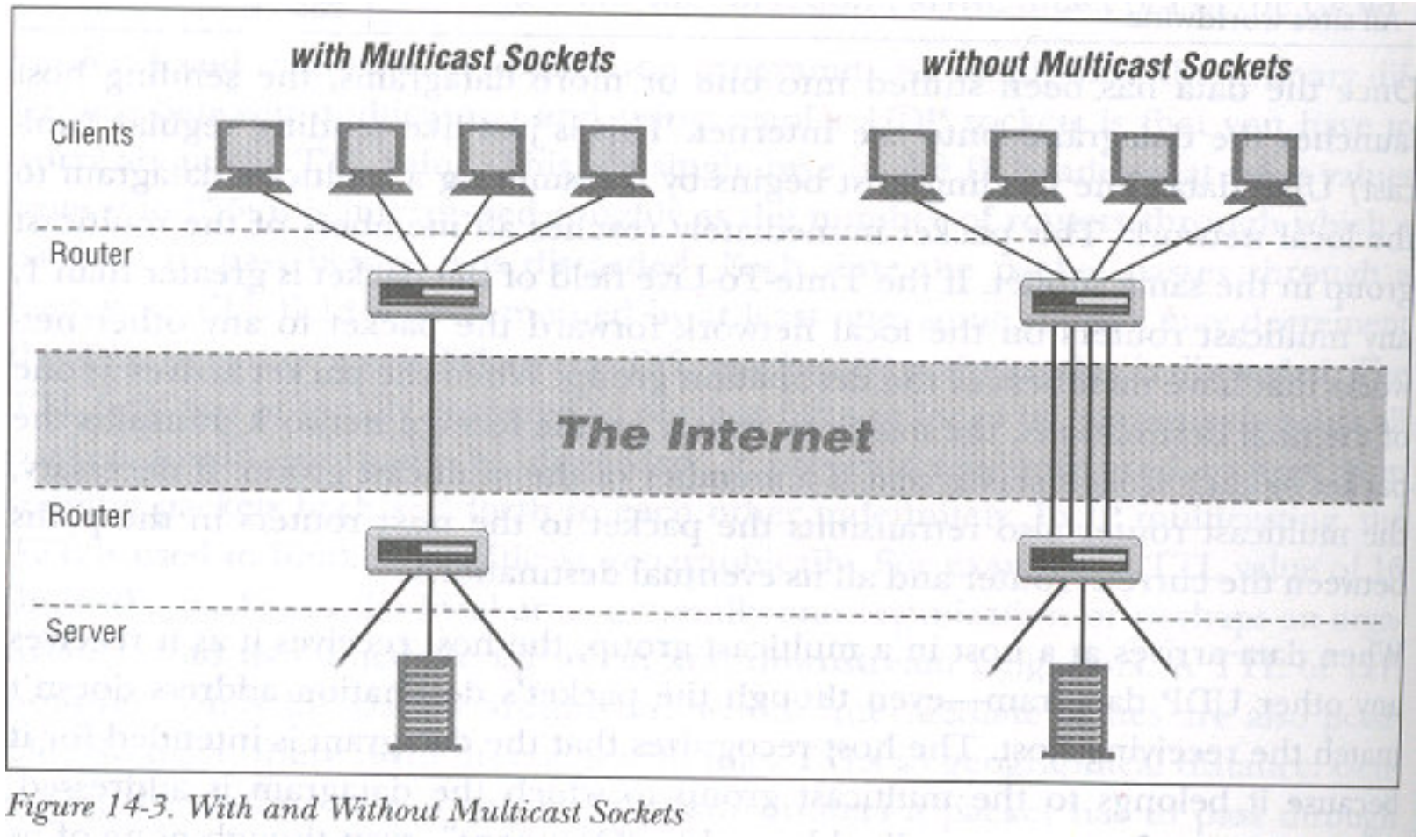


Figure 14-3. With and Without Multicast Sockets

Router and Routing

- Note that real-world routes can be much more complex, involving multiple hierarchies of redundant routers
- Goal of multicast sockets:
  - No matter how complex the network, the same data should never be sent more than once over any given network
  - Programmers don't need to worry about routing issues.
- To send and receive multicast data beyond the local subnet, you need a multicast router
  - Ping all-routers.mcast.net

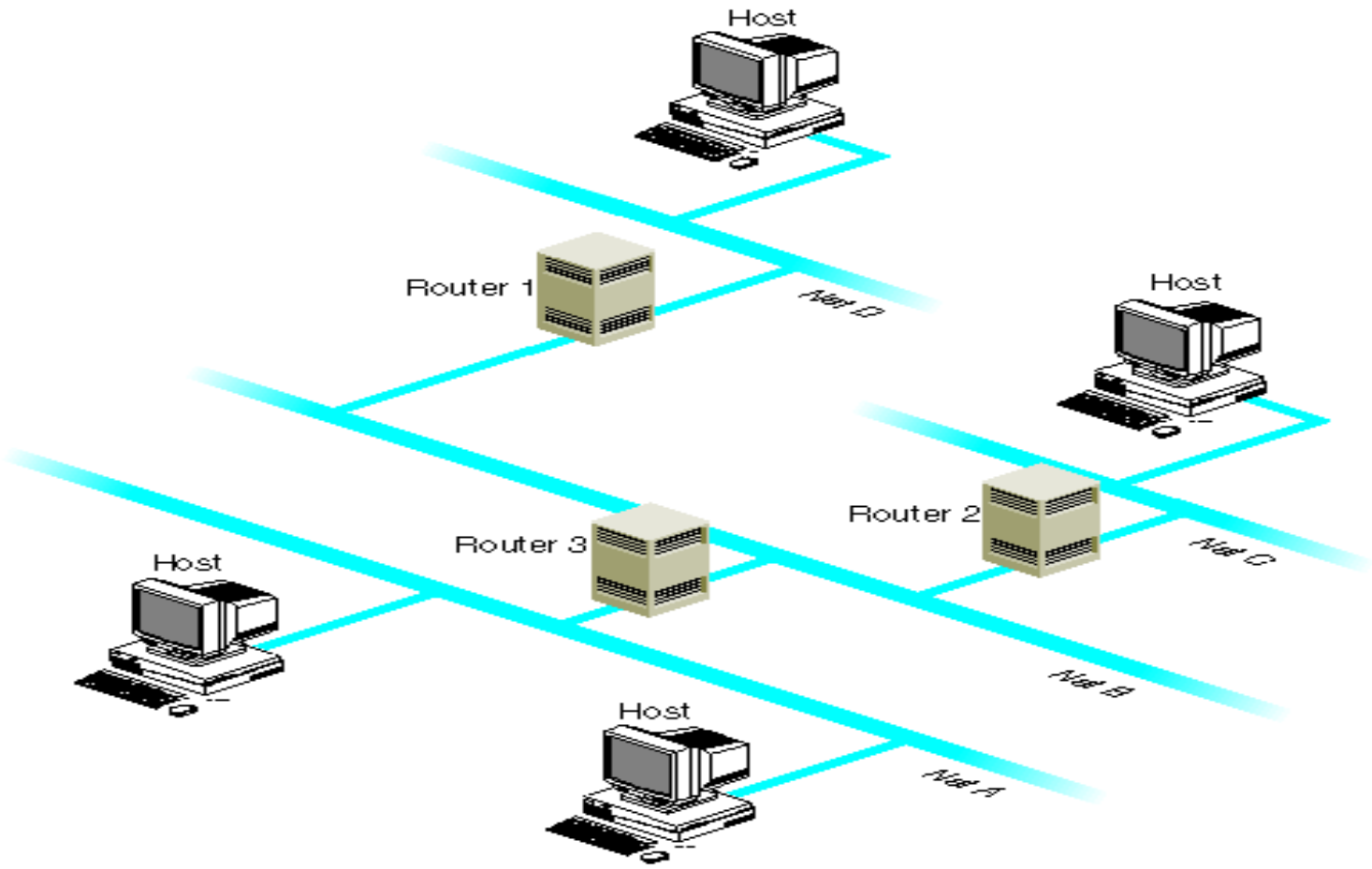
Router and Routing

>Ping all-routers.mcast.net
Pinging all-routers.mcast.net [224.0.0.2] with 32 bytes of data:

Reply from 224.0.0.2: bytes=32 time<10ms TTL=128
Reply from 224.0.0.2: bytes=32 time<10ms TTL=128
Reply from 224.0.0.2: bytes=32 time<10ms TTL=128
Reply from 224.0.0.2: bytes=32 time<10ms TTL=128

Ping statistics for 224.0.0.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

Router and Routing



Multicast Socket

- public class MulticastSocket extends DatagramSocket
  - MulticastSocket inherits from DatagramSocket
- Constructor
  - MulticastSocket(): Create a multicast socket.

    (i.e. use asynymous port)
  - MulticastSocket(int port): Create a multicast socket and bind it to a specific port.

Multicast Socket: communication with a multicast group

- Les principales opérations nécessaires:
  - S'abonner à un groupe de multicast
  - envoyer les données aux élements du groupe
  - Recevoir les donnees des élements du groupe
  - Quiter le groupe.

Multicast Socket: communication with a multicast group

- void joinGroup(InetAddress mcastaddr)
  - Joins a multicast group.
  - Its behavior may be affected by setInterface.
- void send(DatagramPacket p, byte ttl)
  - Sends a datagram packet to the destination, with a TTL (time- to-live) other than the default for the socket.
  - default time to live: 1

Multicast Socket: communication with a multicast group

- Receive method see DatagramSocket
- void leaveGroup(InetAddress mcastaddr)
  - Leave a multicast group.
- void setInterface(InetAddress inf)
  - Set the multicast network interface used by methods whose behavior would be affected by the value of the network interface.
  - Use in multi-homed host

Multicast Socket: communication with a multicast group

- InetAddress getInterface()
  - Retrieve the address of the network interface used for multicast packets.
- void setTimeToLive(int ttl)
  - Set the default time-to-live for multicast packets sent out on this socket.
- int getTimeToLive()
  - Get the default time-to-live for multicast packets sent out on the socket.

MulticastSocket : Methods

MulticastSocket	java.net

Object
└DatagramSocket
└MulticastSocket

public		MulticastSocket( ) throws java.io.IOException
public		MulticastSocket(int port) throws java.io.IOException
public	void	setTTL(byte ttl)
public	void	setTimeToLive(int ttl)
public	byte	getTTL( )
public	int	getTimeToLive( )
public	void	joinGroup(InetAddress mcastaddr)
public	void	leaveGroup(InetAddress mcastaddr)
public	void	setInterface(InetAddress inf)
public	InetAddress	getInterface( )
public	void	send(DatagramPacket p, byte ttl)

Network Programming in Java

- Multicast Networking
  - Multicast is a communication pattern in which a source host sends a message to a group of destination hosts
  - Primary advantage is to decrease network load
    - requires transmission of only a single packet by source
    - interested parties all listen for the same packet(s)

Java Network Programming

- Internet Group Management Protocol (IGMP)
  - the protocol through which hosts tell their local multicast routers that they are interested in receiving multicast packets sent to certain multicast groups
  - Based on information obtained from IGMP, the router can decide whether to forward multicast messages it receives to its subnetwork(s) or not.
    - If there is at least one member of a particular group on a subnetwork, the router will forward the message to that subnetwork. Otherwise, it will discard the multicast packet.

Java Network Programming

- Multicast Networking in Java
  - Class `MulticastSocket`
    - extends `DatagramSocket` with support for IP multicast
    - Constructors
      - `MulticastSocket()`
        `MulticastSocket(int port)`

Java Network Programming

- Methods include:
  - void joinGroup(InetAddress group) throws IOException
  - void leaveGroup(InetAddress group) throws IOException
  - void setTimeToLive(int ttl) throws IOException
    - » corresponding get method
  - void send(DatagramPacket, byte ttl) throws IOException

MulticastSocket

- Client need to open a **MulticastSocket** for joining to the multicast group and the dedicated port no in order to send or receive the multicast traffic

```
mc.joinGroup(224.1.2.3);
```

- After all the operation is done, u need to leave the multicast group

```
mc.leaveGroup(224.1.2.3);
```

- Multicasting uses UDP packets (Not TCP!).

Two Examples

- MulticastSnifer: read data from a multicast group
- MulticastSender: send data to a multicast group

Java Network Programming

- Using Multicast
  - sending a multicast packet

```
MuticastSocket socket = new MulticastSocket();
DatagramPacket packet =
   new DatagramPacket(data, data.length, mGroup, mPort);
socket.send(packet, (byte)64);
socket.close();
```

Network Programming in Java

- Receiving a multicast packet

```
MulticastSocket = new MulticastSocket(mPort);
socket.joinGroup(mGroup);
byte[] buffer = new byte[65508];
DatagramPacket packet =
   new DatagramPacket(buffer, buffer.length);
socket.receive(packet);
socket.leaveGroup(mGroup);
socket.close();
```

Exemple de multicast

```
import java.net.*;
import java.io.*;
public class MulticastSniffer {
public static void main(String[] args) {
InetAddress group = null;
int port = 0;
// read the address from the command line
try {
group = InetAddress.getByName(args[0]);
port = Integer.parseInt(args[1]);
} // end try
catch (Exception e) {
// ArrayIndexOutOfBoundsException, NumberFormatException,
// or UnknownHostException
System.err.println(
"Usage: java MulticastSniffer multicast_address port");
System.exit(1);}
MulticastSocket ms = null;
try {
ms = new MulticastSocket(port);
ms.joinGroup(group);
byte[] buffer = new byte[8192];
while (true) {
DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
ms.receive(dp);
String s = new String(dp.getData( ));
System.out.println(s);}}
catch (IOException e) {
System.err.println(e);}
finally {
if (ms != null) {
try {
ms.leaveGroup(group);
ms.close( );
}
catch (IOException e) {} }}}}
```

Exemple sender

```
import java.net.*;
import java.io.*;
public class MulticastSender {
public static void main(String[] args) {
InetAddress ia = null;
int port = 0;
byte ttl = (byte) 1;
// read the address from the command line
try {
ia = InetAddress.getByName(args[0]);
port = Integer.parseInt(args[1]);
if (args.length > 2) ttl = (byte) Integer.parseInt(args[2]);}
catch (Exception e) {
System.err.println(e);
System.err.println(
"Usage: java MulticastSender multicast_address port ttl");
System.exit(1);}
byte[] data = "Here's some multicast data\r\n".getBytes( );
DatagramPacket dp = new DatagramPacket(data, data.length, ia,
port);
try {
MulticastSocket ms = new MulticastSocket( );
ms.joinGroup(ia);
for (int i = 1; i < 10; i++) {
ms.send(dp, ttl);}
ms.leaveGroup(ia);
ms.close( );}
catch (SocketException se) {
System.err.println(se);}
catch (IOException ie) {
System.err.println(ie);}}}
```