Chương 2: Nội dung

2.1 Nguyên lý của ứng dụng mạng
- Kiến trúc của ứng dụng
- Các yêu cầu của ứng dụng

2.2 Web và HTTP

2.3 FTP

2.4 Thư điện tử
- SMTP, POP3, IMAP

2.5 DNS

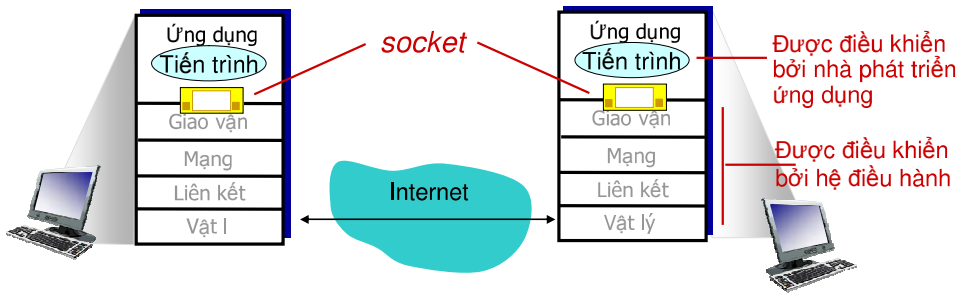
2.6 Ứng dụng P2P

2.7 Lập trình socket với UDP và TCP

Lập trình Socket

Mục đích: hiểu được cách xây dựng ứng dụng truyền thông client/server dùng socket

socket: là cánh cửa giữa các tiến trình ứng dụng và giao thức giao vận end-to-end



Lập trình Socket

Hai loại socket cho hai dịch vụ tầng giao vận:

- *UDP:* truyền các gói tin không tin cậy
- *TCP:* truyền tin cậy, truyền dòng byte có hướng

Ví dụ ứng dụng:

1. Client đọc vào một dòng ký tự (dữ liệu) từ bàn phím và gửi dữ liệu đến server.
2. Server nhận dữ liệu và chuyển các ký tự thành dạng ký tự viết hoa.
3. Server gửi dữ liệu đã được chuyển thành dạng viết hoa về cho client.
4. Client nhận dữ liệu và hiển thị dòng ký tự lên màn hình.

Lập trình mạng trên Java

- Gói java.net
  - InetAddress
  - ServerSocket
  - Socket
  - URL
  - URLConnection
  - DatagramSocket

Lập trình mạng trên Java

- InetAddress class
  - Class mô tả về địa chỉ IP
  - Các phương thức getLocalHost, getByName, hay getAllByName để tạo một InetAddress instance:
    public static InetAddess InetAddress.getByName(String hostname)
    public static InetAddess [] InetAddress.getAllByName(String hostname)
    public static InetAddess InetAddress.getLocalHost()
  - Để lấy địa chỉ IP hay tên dùng các phương thức:
    getHostAddress()
    getHostName()

Lập trình Socket *với UDP*

UDP: không có "kết nối" giữa client & server

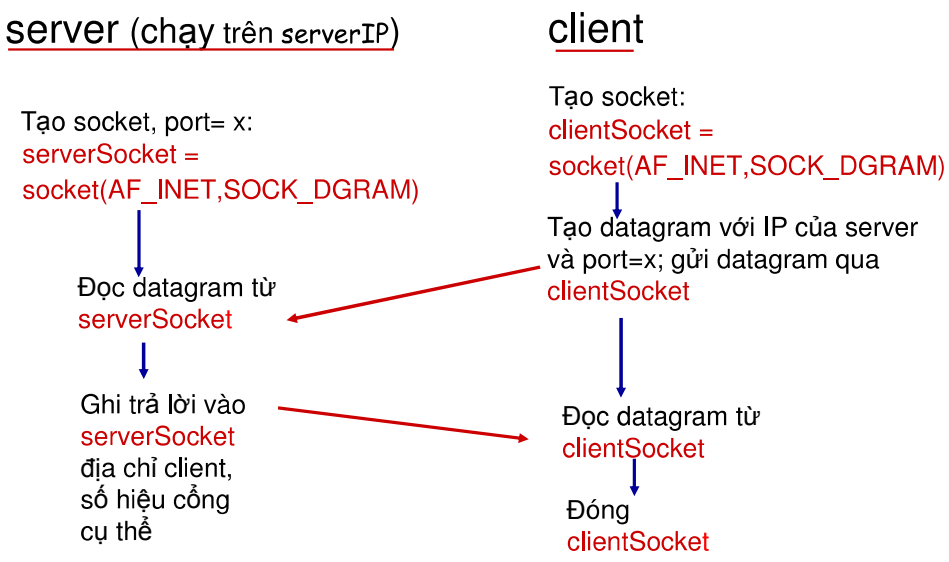
- Không bắt tay trước khi gửi dữ liệu
- Bên gửi gắn địa chỉ IP và số hiệu cổng đích vào trong mỗi gói tin
- Bên nhận sẽ trích địa chỉ IP và số hiệu cổng của bên gửi từ gói tin nhận được

UDP: dữ liệu được truyền có thể bị mất hoặc không đúng trình tự khi nhận

Quan điểm ứng dụng:

- UDP cung cấp truyền không tin cậy theo các nhóm byte ("các gói tin") giữa client và server

Tương tác client/server socket: UDP



Ví dụ: UDP client (Java client)

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

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

tạo input stream

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

tạo client socket

```
      DatagramSocket clientSocket = new DatagramSocket();
```

dịch hostname thành địa chỉ IP dùng DNS

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

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

      String sentence = inFromUser.readLine();
      sendData = sentence.getBytes();
```

Ví dụ: UDP client (Java client)

tạo dữ liệu gửi datagram, độ dài, địa chỉ IP, port

gửi datagram đến server

đọc datagram từ 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();
}
}
```


Ví dụ: UDP server (Java server)

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

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

tạo datagram socket tại port 9876

```
DatagramSocket serverSocket = new DatagramSocket(9876);

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

while(true)
 {
```

tạo không gian để nhận datagram

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

nhận datagram

```
serverSocket.receive(receivePacket);
```

Ví dụ: UDP server (Java server)

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

lấy địa chỉ IP,
số hiệu cổng
của người gửi   InetAddress IPAddress = receivePacket.getAddress();

                int port = receivePacket.getPort();

                        String capitalizedSentence = sentence.toUpperCase();

                sendData = capitalizedSentence.getBytes();

tạo datagram
để gửi tới client   DatagramPacket sendPacket =
                      new DatagramPacket(sendData, sendData.length, IPAddress,
                              port);

ghi datagram
vào socket      serverSocket.send(sendPacket);
              }
            }
          }
```

kết thúc vòng lặp while, quay lại và chờ datagram khác

Lập trình socket *với TCP*

client phải tiếp xúc với server

- Tiến trình server phải chạy trước
- server phải tạo socket (cửa) để đón client tiếp xúc

client tiếp xúc với server bằng cách:

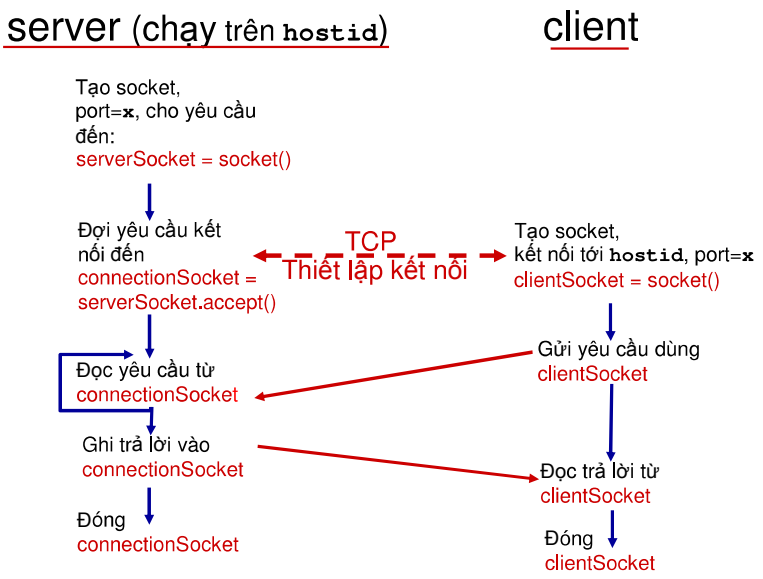
- Tạo TCP socket, xác định địa chỉ IP, số hiệu cổng của tiến trình server
- *Khi client tạo socket:* TCP client sẽ thiết lập kết nối tới TCP server

- Khi được tiếp xúc bởi client, *TCP server sẽ tạo socket mới* cho tiến trình server để truyền thông với client
  - Cho phép server "nói chuyện" với nhiều client
  - Các số hiệu cổng nguồn được dùng để phân biệt các client (xem thêm trong Chương 3)

Quan điểm ứng dụng:

TCP cung cấp truyền tin cậy, truyền dòng byte theo đúng trình tự giữa client và server.

Tương tác client/server socket: TCP



Ví dụ: TCP client (Java client)

```
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());
```

tạo input stream

tạo client socket, kết nối vào server

tạo output stream gắn vào socket

Ví dụ: TCP client (Java client)

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

                    sentence = inFromUser.readLine();

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

                    modifiedSentence = inFromServer.readLine();

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

                    clientSocket.close();

                  }
                }
```

tạo input stream gắn vào socket

gửi dòng đến server

đọc dòng từ server

Ví dụ: TCP server (Java server)

```
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()));
```

tạo socket đón tiếp xúc tại port 6789

chờ client tiếp xúc với server

tạo input stream, gắn vào socket

Ví dụ: TCP server (Java server)

tạo output stream, gắn vào socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

đọc dòng từ socket

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

ghi dòng ra từ socket

```
outToClient.writeBytes(capitalizedSentence);
            }
        }
    }
```

kết thúc vòng lặp while, quay lại và chờ kết nối của client khác

Xây dựng một Web server đơn giản

- Quản lý một yêu cầu HTTP
- Chấp nhận yêu cầu
- Phân tích cú pháp phần tiêu đề (header)
- Lấy file được yêu cầu từ hệ thống file của server
- Tạo thông điệp đáp ứng HTTP:
  - các dòng header + file
- Gửi đáp ứng đến client
- Sau khi tạo server, có thể yêu cầu file dùng trình duyệt (ví dụ: IE)
- Xem giáo trình để biết thêm chi tiết

Chương 2: Tổng kết

Trình bày các vấn đề liên quan đến ứng dụng mạng!

- Kiến trúc của ứng dụng
  - client-server
  - P2P
- Các yêu cầu dịch vụ của ứng dụng:
  - Truyền tin cậy, băng thông, trễ
- Mô hình dịch vụ giao vận của Internet
  - Hướng kết nối, truyền tin cậy: TCP
  - Truyền không tin cậy, truyền gói tin: UDP
- Các giao thức cụ thể:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT
- Lập trình socket: TCP socket, UDP socket

Chương 2: Tổng kết

Quan trọng hơn: được học về các giao thức!

- Trao đổi giữa các thông điệp yêu cầu/đáp ứng:
  - client yêu cầu thông tin hoặc dịch vụ
  - server đáp ứng với dữ liệu, hoặc mã trạng thái
- Định dạng thông điệp:
  - Phần tiêu đề (header): các trường với thông tin về dữ liệu
  - Dữ liệu: thông tin được truyền thông

Các vấn đề quan trọng:

- Thông điệp điều khiển và thông điệp dữ liệu
  - in-band, out-of-band
- Tập trung hóa và không tập trung hóa
- Không trạng thái và có trạng thái
- Truyền thông điệp tin cậy và truyền thông điệp không tin cậy
- "Sự phức tạp tại phần cạnh của mạng"