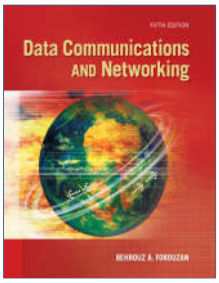


Computer Networks

Lecturer: Dr. A.O. Aldhaibani

Application Layer and HTTP

Application Layer

- The application layer is the 5th layer in TCP/IP.
- The **application** layer provides **services** to the **users**.
- **Application** layer protocols are two types:
  - Standard Protocols.
  - Nonstandard Protocols.
  - Each **standard** or **non-standard** protocol is a **pair** of computer programs that **interact** with the **user** and the **transport layer** to provide a specific **service** to the **user**.

Standard Protocols

- **Standard** Application-Layer Protocols
  - are protocols **standardized** and **documented** by the **Internet authority** such as **Internet Assigned Numbers Authority** (**IANA**).
  - They normally become part of the package that is included in operating systems such as Windows or UNIX.
  - IANA is **owned** by Internet Corporation for Assigned Names and Numbers (**ICANN**).

Nonstandard Protocols

- Nonstandard Application-Layer Protocols are **programs** are written by a **programmer** to provide a service to the user. Skype protocol is a proprietary protocol. Microsoft
- What is needed? is to **write** programs, in one of the **computer languages**, that use the available **services** provided by the transport-layer protocols TCP or UDP.

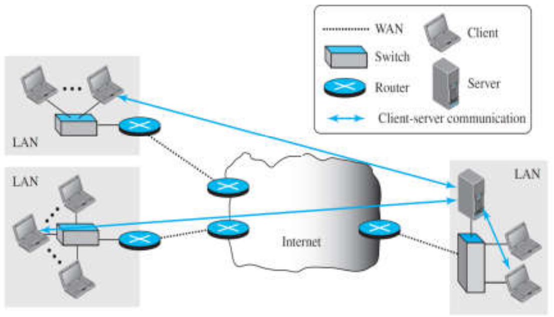
Application Layer Protocols Paradigm

- It should be clear that to use the Internet we need **two** application programs to interact with each other.
- One **running** on a **computer somewhere** in the world, the other **running** on another computer **somewhere else** in the world.
- Two **paradigms** of the application layer protocols:
  - ***client-server*** *paradigm.*
  - ***peer-to-peer*** *paradigm.*

Client-server paradigm

- The **traditional** paradigm is called the **client-server paradigm**.
- In this paradigm:
  - the **service provider** is an application program, called the **server** process;
  - the server runs **continuously** and **waiting** for another application program, called the **client** process,
  - the client make a **connection** through the Internet and **ask** for service.

Figure 25.2 *Example of a client-server paradigm*



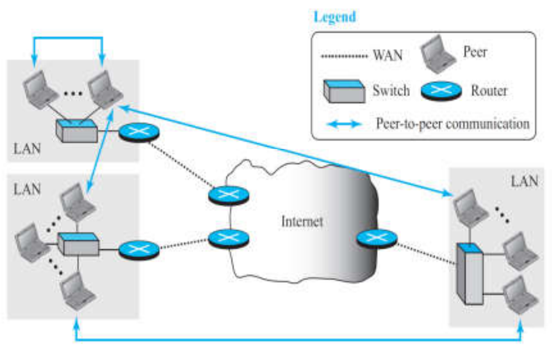
A Problem of Client/Server paradigm

- The **concentration** of the communication load is on the **shoulder of the server**, which means the server should be a **powerful** computer.
- This problem could be **solved** by making a **cluster** of **physical servers** run as **one virtual server**.

Peer-to-peer paradigm

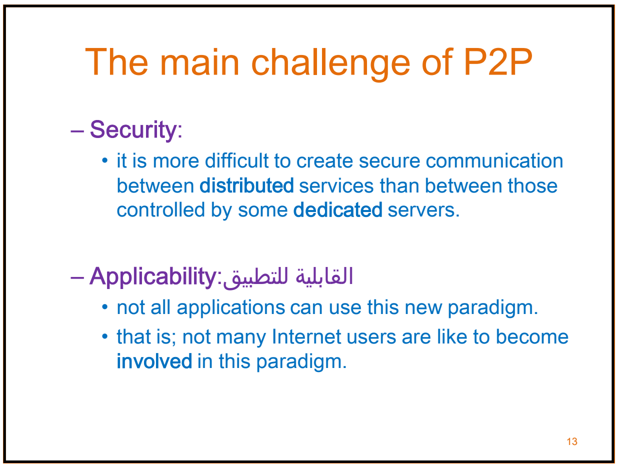
- A new paradigm, called the **peer-to-peer paradigm** (often abbreviated *P2P paradigm*) has emerged to respond to the needs of some new applications.
- In this paradigm, there is **no need** for a server process to be running all the time and waiting for the client processes to connect. The responsibility is **shared** between peers.

Figure 25.3 *Example of a peer-to-peer paradigm*



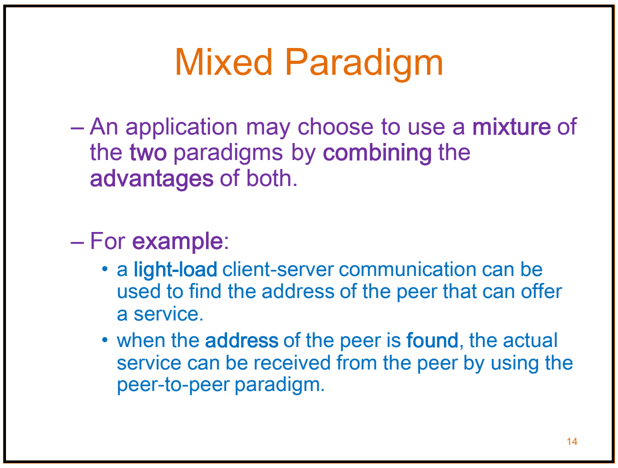
Peer-to-peer paradigm

- The peer-to-peer paradigm can be **used** when some **computers** connected to the Internet have something to **share** with each other.
- **Although**:
  - the peer-to-peer paradigm has been proved to be easily **scalable** and **cost-effective** in **eliminating** the **need** for **expensive servers** to be running and maintained all the time.
  - there are also some **challenges**.
- Ex: BitTorrent, Skype, IPTV, and Internet telephony, that use this paradigm.



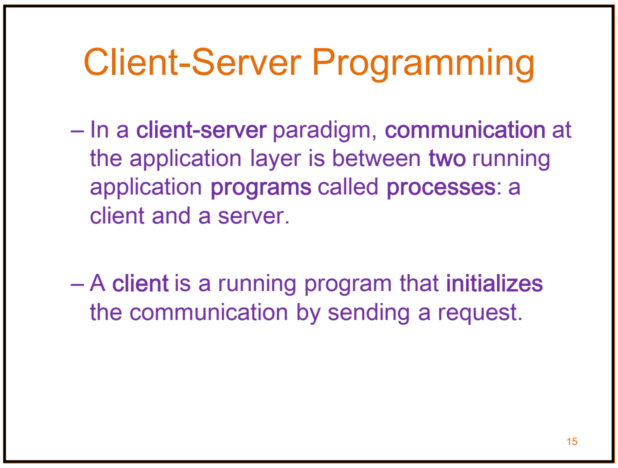
The main challenge of P2P

- Security:
  - it is more difficult to create secure communication between **distributed** services than between those controlled by some **dedicated** servers.
- Applicability:القابلية للتطبيق
  - not all applications can use this new paradigm.
  - that is; not many Internet users are like to become **involved** in this paradigm.



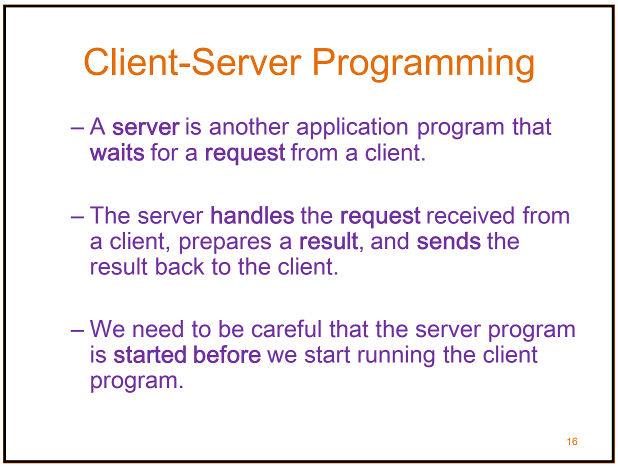
Mixed Paradigm

- An application may choose to use a **mixture** of the **two** paradigms by **combining** the **advantages** of both.
- For **example**:
  - a **light-load** client-server communication can be used to find the address of the peer that can offer a service.
  - when the **address** of the peer is **found**, the actual service can be received from the peer by using the peer-to-peer paradigm.



Client-Server Programming

- In a **client-server** paradigm, **communication** at the application layer is between **two** running application **programs** called **processes**: a client and a server.
- A **client** is a running program that **initializes** the communication by sending a request.



Client-Server Programming

- A **server** is another application program that **waits** for a **request** from a client.
- The server **handles** the **request** received from a client, prepares a **result**, and **sends** the result back to the client.
- We need to be careful that the server program is **started before** we start running the client program.

Application Programming Interface

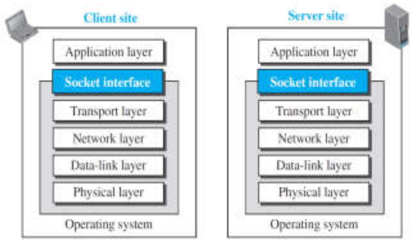
- We need a **set of instructions** to tell the lowest four layers of the TCP/IP suite to:
  - **open** the connection,
  - **send** and **receive** data from the other end, and
  - **close** the connection.
- The set of instructions of this kind is normally referred to as an **application programming interface (API)**.
- An **interface** in programming is a **set** of **instructions** between two **entities**.

Application Programming Interface

- **Several APIs** have been designed for communication. **Three** among them are **common**:
  - **socket interface.**
  - **Transport Layer Interface** (TLI).
  - STREAM.
- we **briefly** discuss only ***socket interface***, the most common one.
- The **socket interface** is a set of instructions that provide communication between the **application layer** and the **operating system**.
- Socket interface started in the **early** 1980s at UC Berkeley as part of a **UNIX** environment.

socket interface

Figure 25.4 *Position of the socket interface*

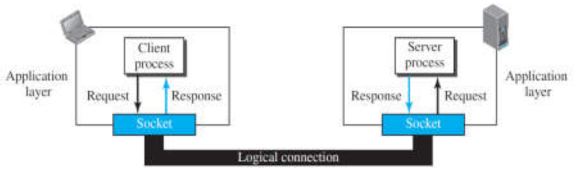


Sockets

- The idea of **sockets** allows us to use the set of all instructions already designed in a programming language for other **sources** and **sinks**.
- The client **thinks** that the **socket** is the **entity** that **receives** the request and **gives** the response;
- The server **thinks** that the **socket** is the **one** that has a **request** and needs the **response**.

Sockets

Figure 25.6 *Use of sockets in process-to-process communication*



Socket Addresses

- A **socket address** should first **define** the computer on which a client or a server is running.

- A **computer** in the Internet is **uniquely defined** by its **IP address**, a 32-bit integer in the current Internet (currently **IP version 4**).

Socket Addresses

- However, **several** client or server processes may be **running** at the same time on a computer.

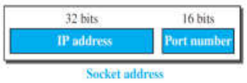
- which means that we **need** another **identifier** to **define** the **specific client** or **server involved** in the communication.

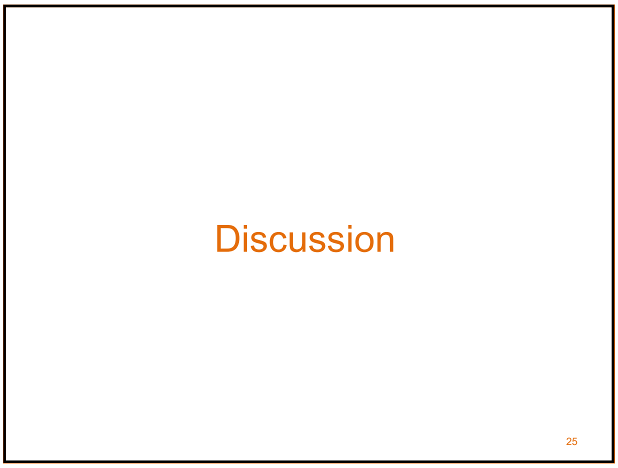
Socket Addresses

- An application **program** can be **defined** by a **port** number, a 16-bit integer.

- This means that a **socket address** should be a **combination** of an **IP address and a port**.

Figure 25.7 *A socket address*





Discussion

25