Computer Network I

Reti di Calcolatori I

Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Informatica

Riccardo Caccavale

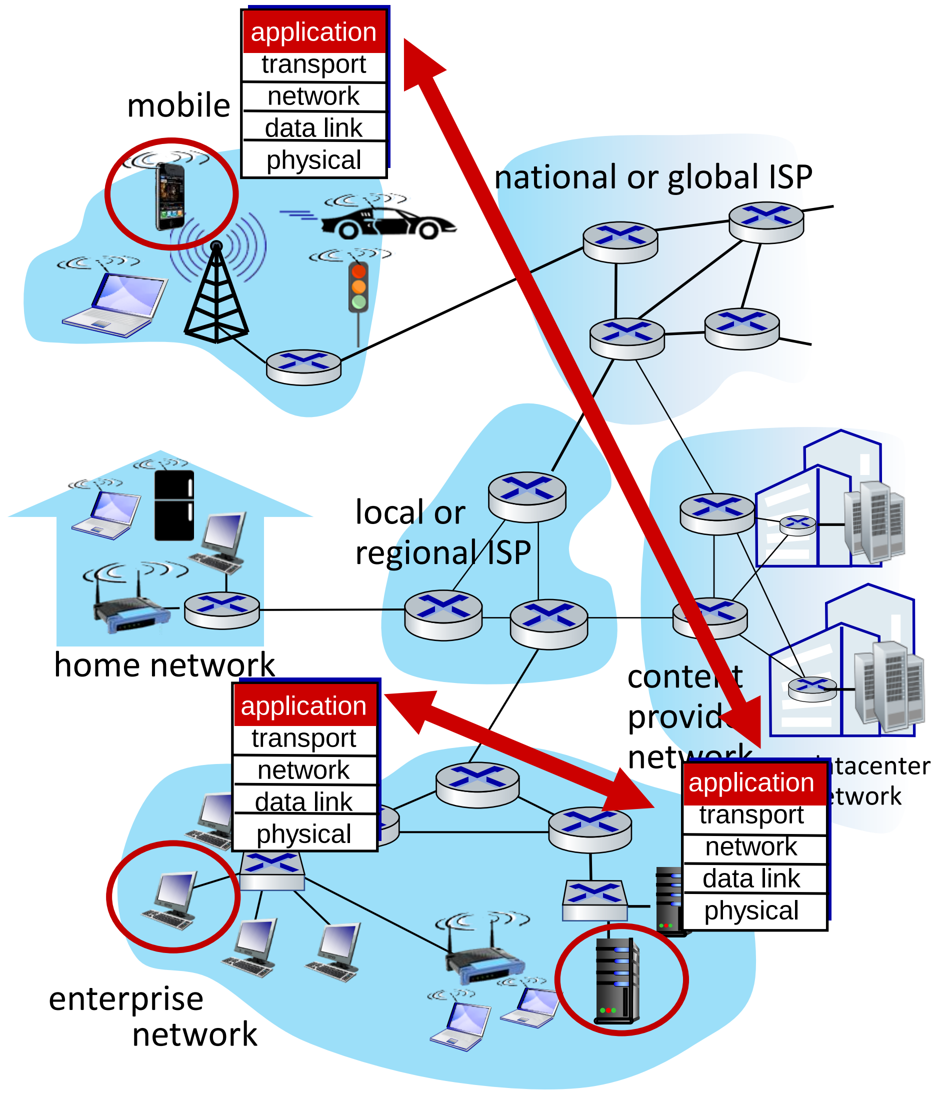
(riccardo.caccavale@unina.it)

Application Layer

Network Applications

- A **network application** is composed by multiple programs that run on different end-systems and communicate with each other over the network.

- *Example*: a Web application includes two distinct programs:
  - the **browser program** running in the user's host (desktop, laptop, tablet, smartphone, and so on).
  - the **Web server program** running in the Web server host.



Application Layer

Some Network Applications

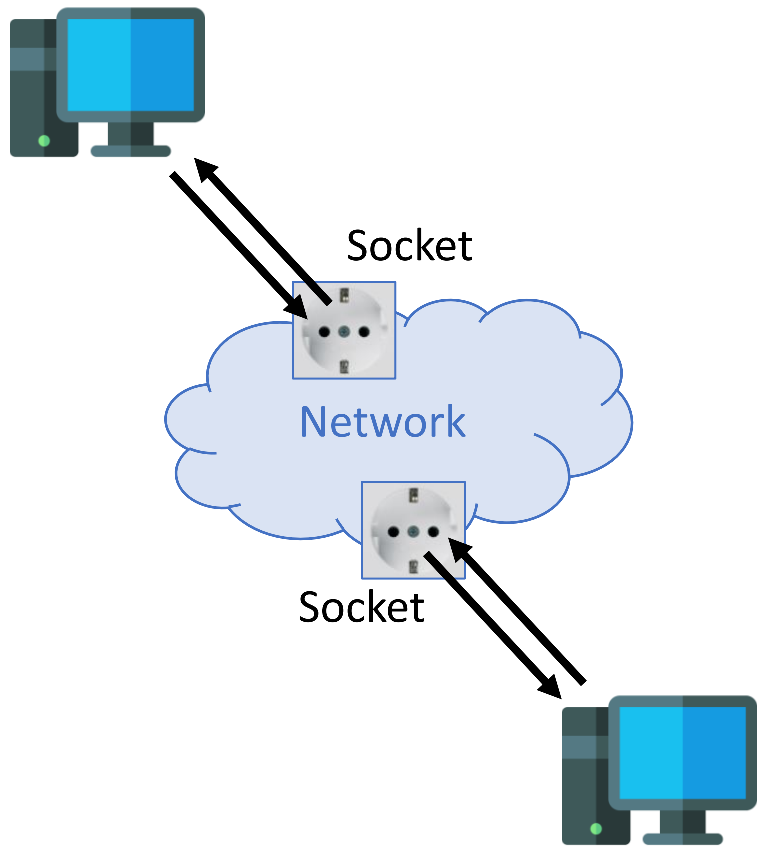
- The **applications** are the programs that runs on the **devices** and allow users to access to **services**.
  - social networking
  - Web
  - text messaging
  - e-mail
  - multi-user network games
  - streaming stored video (YouTube, Netflix)
  - P2P file sharing
  - voice over IP (e.g., Skype)
  - real-time video conferencing
  - Internet search
  - remote login
  - …

Application Layer

Creating Network Applications

- Creating network applications means to write programs that:
  - **Run on different end systems**, perhaps using different languages or OS.
  - **Communicate over network** (e.g., via sockets) by means of a specific protocol.

- No need to write software for network-core devices:
  - Network devices **do not run user applications**.
  - From applications’ perspective, network is ideally **back-box**.
  - There are **specific libraries** (e.g., sockets) implementing network functionalities.

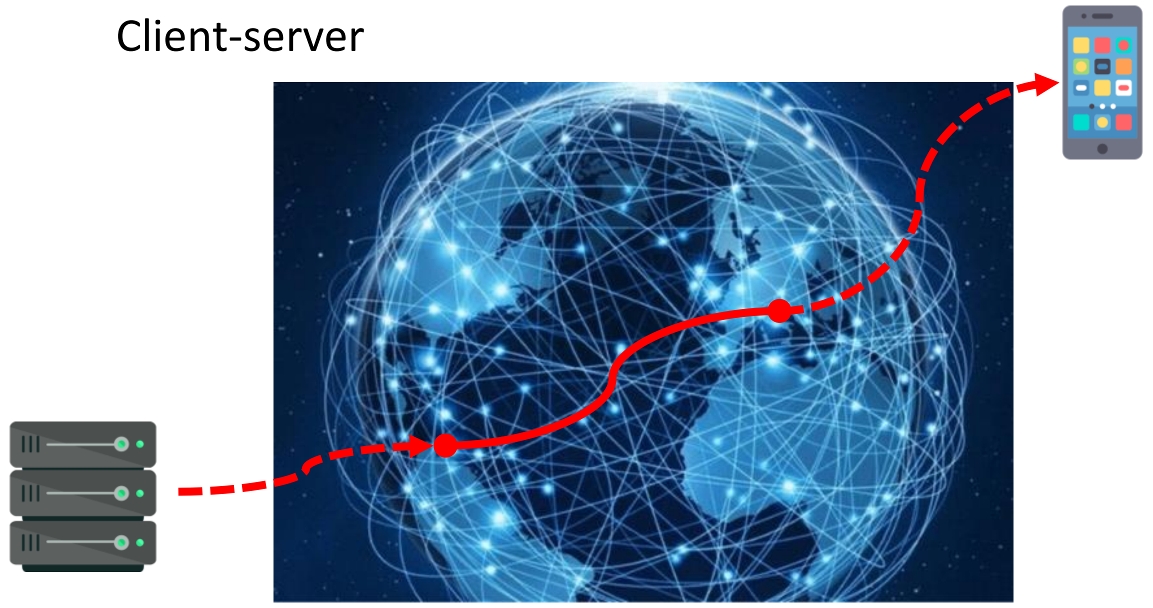


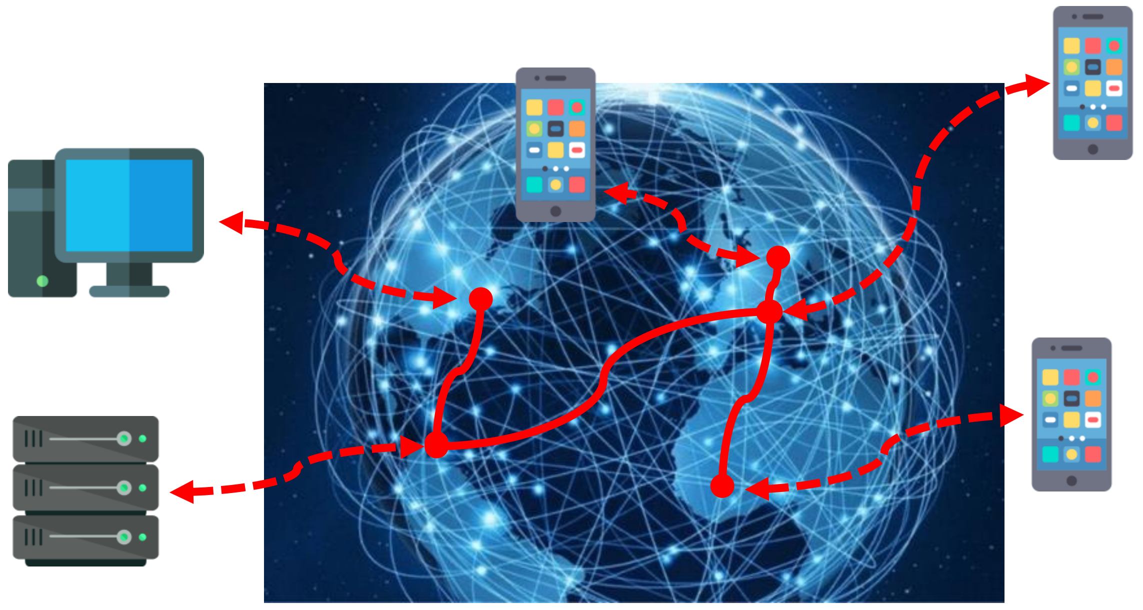
Application Layer

Network Applications

- A **(network) application architecture** is designed by the *application developer* and dictates how the application is structured over the various end systems:
  - **Client-server**: the nodes are heterogeneous, there is an always-on host (server), which provides services to be requests from many other hosts (clients).
  - **Peer-to-peer (P2P)**: the nodes are (ideally) homogeneous, the application exploits direct communication between pairs of intermittently connected hosts (peers).

Client-server





Peer-to-peer

Application Layer
Client-server Applications

- In a client-server architecture, **clients do not directly communicate** with each other.

- The server has a **fixed, well-known address** (IP address) so a client can always contact the server by sending a packet (request) to it.

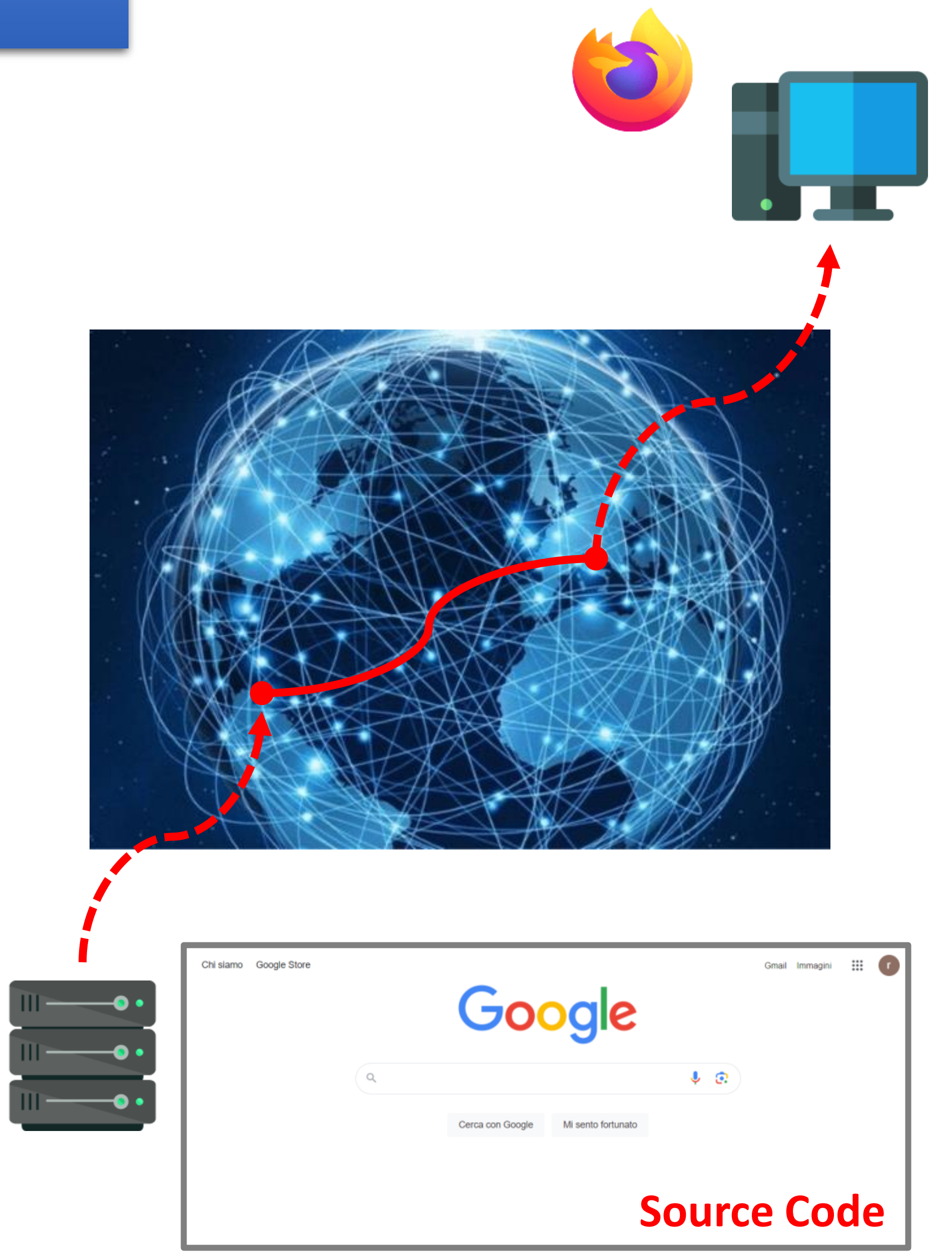
- In a client-server architecture **multiple servers** are often involved to keep up with all the requests from clients. The **client is typically unaware** of that and perceives them as a single server.

- Multiple servers can be:
  - Grouped into **data centers** containing a huge number of servers in a specific location.
    - Servers must be powered, maintained, and well connected.
  - Scattered as **distributed servers** all around the world.
    - Servers must be interconnected and coordinated.
  - Organized in **distributed data centers** (e.g., Google).

Application Layer

Client-server Applications

- A **web application** is a typical client-server example:
  - There is an **always-on Web server** receiving requests from the browsers on the client hosts.
  - When a Web server receives a **request** for an object from a client host, it **responds** by sending the requested object to the client host.
  - The web **server is always reachable** by the hosts.
  - Google has 30 to 50 **data centers distributed around the world**, which collectively handle search, but also YouTube, Gmail, and other services (in Italy, there is one in Milan).



Application Layer

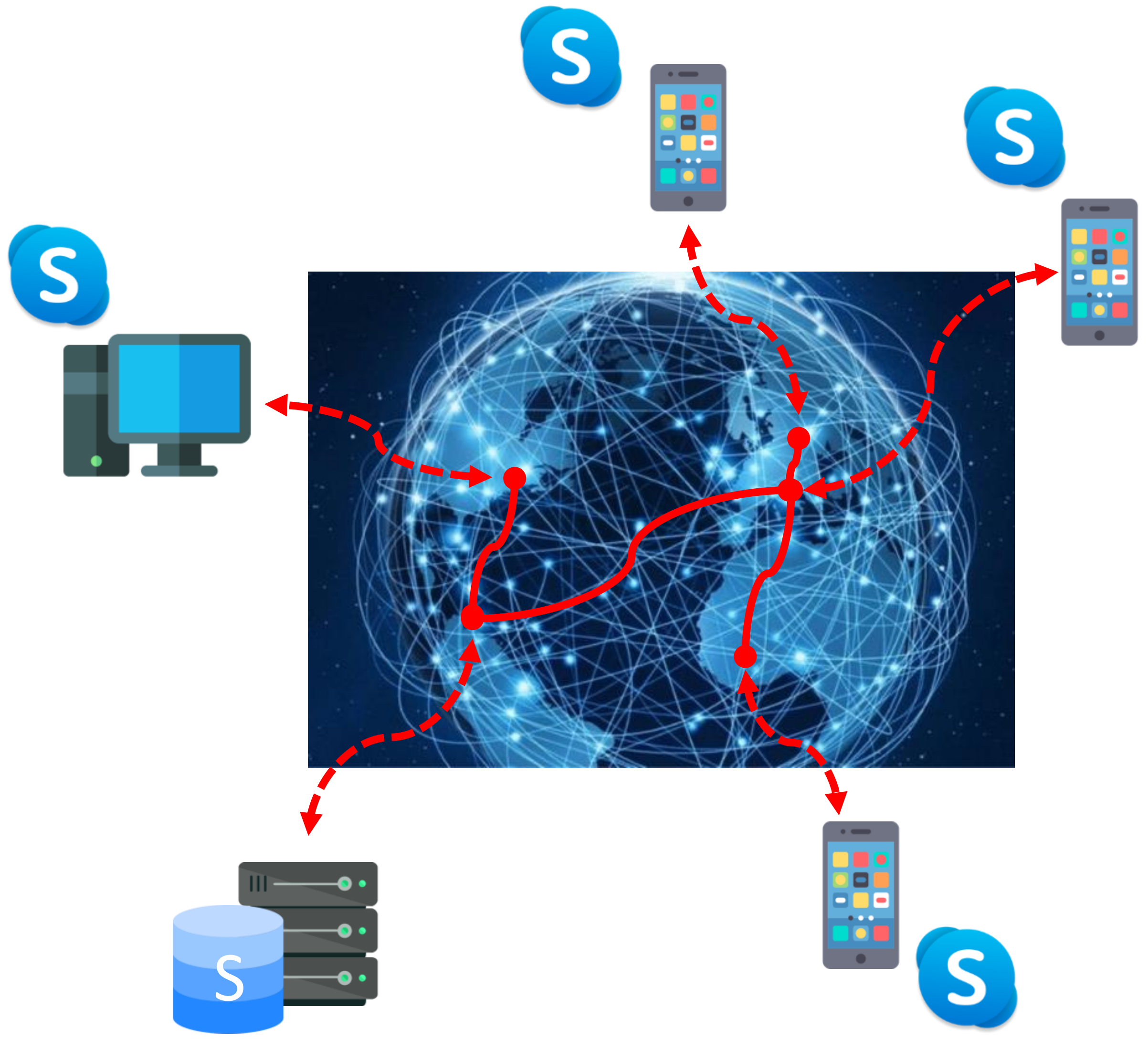
Peer-to-peer Applications

- It is often used for **traffic-intensive applications**.
- There is no single server with fixed address, but **all clients have their own addresses**.
- Pure P2P applications are uncommon: most applications have **hybrid architectures**, combining both client-server and P2P elements.
  - For example, for many instant messaging applications, **servers are used to track the addresses of users**, but user-to-user messages are sent directly between user hosts (without passing through intermediate servers).
- P2P architectures are **scalable** and **distributed**.
  - For example, in a P2P file-sharing application, although each peer generates workload by requesting files, each peer also **adds service capacity** to the system by distributing files to other peers.
- P2P architectures are also **cheap** (no need for infrastructures or significant bandwidth) but there are **security, performance, and reliability issues**.

Application Layer

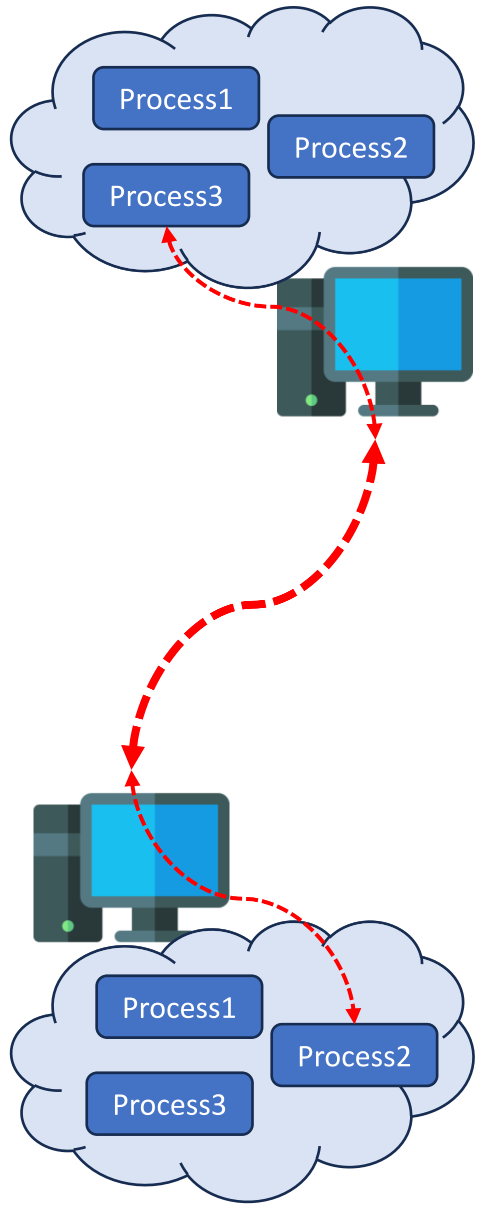
Peer-to-peer Applications

- Typical P2P applications are internet **telephony and video conference** (e.g., Skype) or **file-sharing** (e.g., BitTorent, Napster):
  - Servers are used to **track the IP addresses of users**, but user-to-user messages/requests are sent directly between user hosts (without passing through intermediate servers).
  - In file-sharing applications the server can also **trace all available files** in order to speed up the search.



Application Layer
Communication between Processes

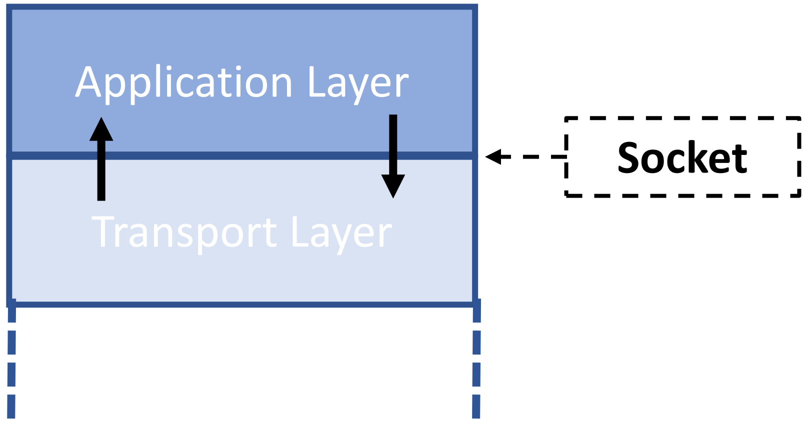
- In a network application there are **processes** running on different machines (potentially running on **different operating systems**) that communicate through the network.
  - In *web applications* the process of a client's browser exchanges messages with the process of the web server.
  - In *P2P file-sharing* a file is transferred from a process in one peer to a process in another peer.
- Between a pair of communicating processes there is typically a **client process** and a **server process**:
  - In *web applications* a browser's process is a client, while the server's process is a server.
  - In *P2P file-sharing* the peer that is downloading can be seen as a client, while the peer that is uploading as a server.
- In a P2P application processes may "change role".



Application Layer

Communication between Processes

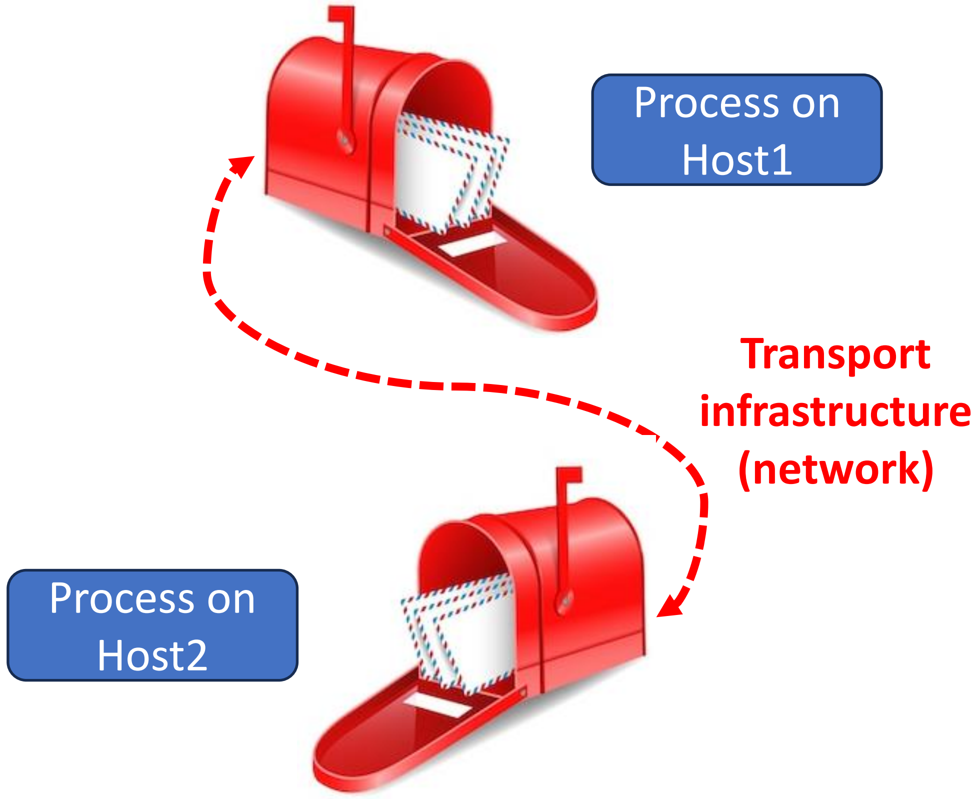
- Most network applications consist of **pairs of communicating processes** sending/receiving messages to/from each other through the underlying network.
- A **socket** is a **software interface** that allows processes to send/receive messages over the network.
- Considering the TCP/IP stack, a **socket is the interface** between the application layer and the transport layer (TCP).



Toward hardware (low-level)

Application Layer
Communication between Processes

- A typical analogy is to consider sockets as **mailboxes**:
  - When a process wants to send a message to another process on another host, it puts the message **into a postal mailbox**.
  - This sending process **assumes** that there is a **transportation infrastructure** on the other side of its door that will transport the message to the mailbox of the destination process.
  - Once the message arrives at the destination host, the message passes through the **receiving process's mailbox** (socket).



Application Layer

Communication between Processes

- **Sockets** are used as **Application Programming Interface** (API) between the application and the network.

- The application developer has control of everything on the application-layer side of the socket but **has little control of the transport-layer** side of the socket.

- The only control that the application developer has on the transport-layer side is:
  1. The **choice** of transport protocol (TCP/UDP).
  2. Perhaps the ability to set a few **transport-layer parameters** such as maximum buffer and maximum segment sizes.

- Once the application developer chooses a transport protocol (if a choice is available), the application is built assuming as given the transport-layer services provided by that protocol.

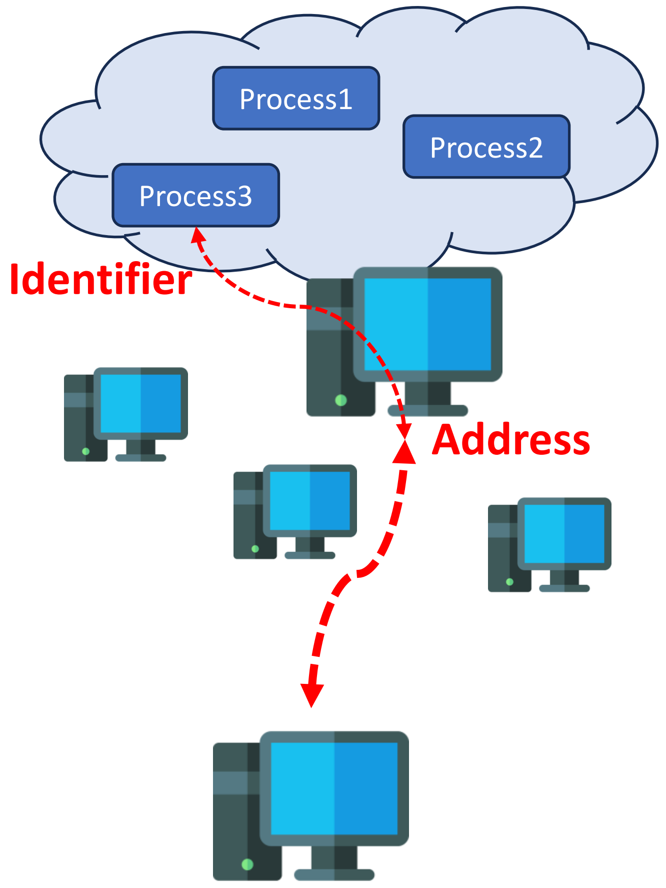
Application Layer

Communication between Processes

- Following the postal mail analogy, **processes need an address** to send messages.

- Since **multiple network applications** can be running on a **single host**, to identify the receiving process, two pieces of information need to be specified:
    1. An **address** of the host (to find the right host on the network).
    2. An **identifier** of the receiving process (to find the right process in the host).

- The **address** works at the network level (routing) while **identifier** works at transport level.



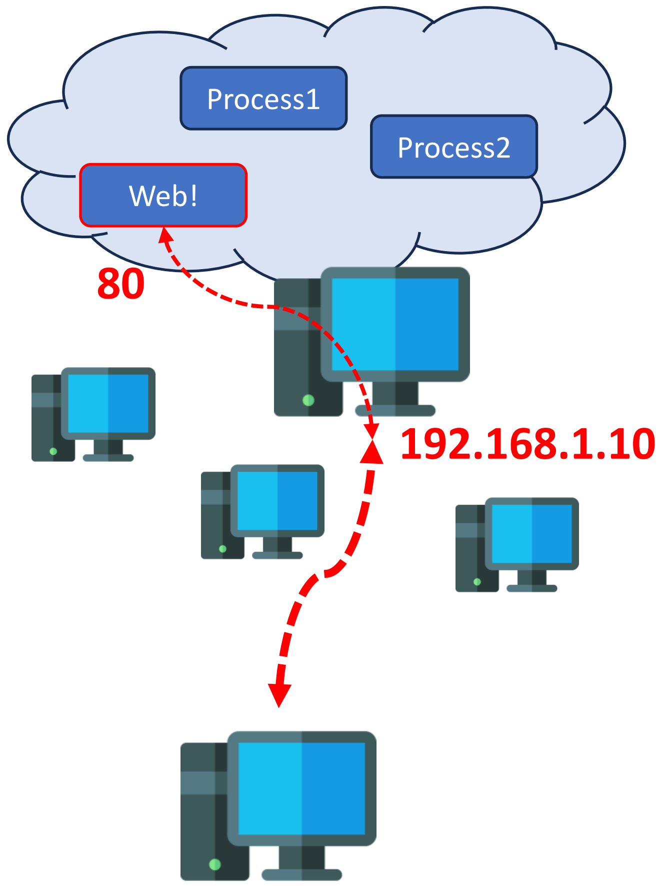
Application Layer

Communication between Processes

- In networks, a host is identified by an **IP address** (32-bit quantity) while the process is identified by a **port number**.

- Popular applications have been conventionally assigned to **specific port numbers**.
  - For example, a Web server is identified by port number 80, a mail server process (using the SMTP protocol) is identified by port number 25.

- A list of well-known port numbers (and protocols) can be found here: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.



Application Layer

Services from transport-layer

- The **transport-layer** (below the application layer) offers protocols to guarantee:
  - **Reliability** of data transfer: data sent by one end of the application is delivered correctly and completely to the other end.
  - **Throughput**: the rate at which the sending process can deliver bits to the receiving process (bit/sec).
  - **Timing**: every bit that the sender pumps into the socket arrives at the receiver’s socket within a time range.
  - **Security**: encryption and decryption of the messages.
- TCP: includes a **handshake** between processes service and a reliable data transfer service (aka **connection-oriented**).
- UDP: is **lightweight** with minimal services, there is no handshake, no guarantee that messages are received (aka **connectionless**).

Application Layer
Protocols

- An **application-layer protocol** defines how network application's processes, running on different end systems, pass messages to each other.
  - how are messages structured? What are the meanings of the various fields in the messages? When do the processes send the messages?

- Specifically, application-layer protocols define:
  1. The **types** of messages exchanged (for example request messages and response messages).
  2. The **syntax** of the various message types, such as the fields in the message and how the fields are delineated.
  3. The **semantics** of the fields, i.e., the meaning of the information in the fields.
  4. The **rules** for determining when and how a process sends messages and responds to messages.

Application Layer

Some Application Protocols

- There are several application-layer protocols that are commonly used on networks (and internet), here some example.

Application	Description
DHCP	Dynamic Host Configuration Protocol, assigns IP addresses
DNS	Domain Name System, translate website names to IP addresses
HTTP/HTTPS	HyperText Transfer Protocol (Secure), transfer web pages
SMTP/SMTPS	Simple Mail Transfer Protocol (Secure), sends email messages
SNMP	Simple Network Management Protocol, manages network devices
Telnet/SSH	Teletype Network (Secure SHell), allows command-line interfacing with remote hosts
FTP/FTPS	File Transfer Protocol (Secure), used to transfer files

Application Layer

Some Application Protocols

- Some application may exchange **sensitive information** (e.g., user's credentials, personal data, etc.).
- On public networks, messages should be **secured**.

Application	Description
DHCP	Dynamic Host Configuration Protocol, assigns IP addresses
DNS	Domain Name System, translate website names to IP addresses
HTTP/HTTPS	HyperText Transfer Protocol (Secure), transfer web pages
SMTP/SMTPS	Simple Mail Transfer Protocol (Secure), sends email messages
SNMP	Simple Network Management Protocol, manages network devices
Telnet/SSH	Teletype Network (Secure SHell), allows command-line interfacing with remote hosts
FTP/FTPS	File Transfer Protocol (Secure), used to transfer files

Application Layer

FTP and FTPS

- FTP (File Transfer Protocol) is one of the **oldest** protocols defined in Internet (first version in 1971) and is used to transfer files between hosts over network.
- In the basic version of FTP data transfer is in **clear-text** (username, password, and files) so it is best used in local or private applications.
- The **secure** version FTPS (FTP Secure) protects username and password and encrypt contents.
- Both FTP and FTPS have two components:
  - The **protocol** that specifies commands (show, get, delete files, etc.).
  - A **software application** implementing the protocol (client-side and a server-side software).

Application Layer

FTP Example

- In Linux we can use ftp and vsftpd (very secure FTP daemon) as client and server applications respectively.

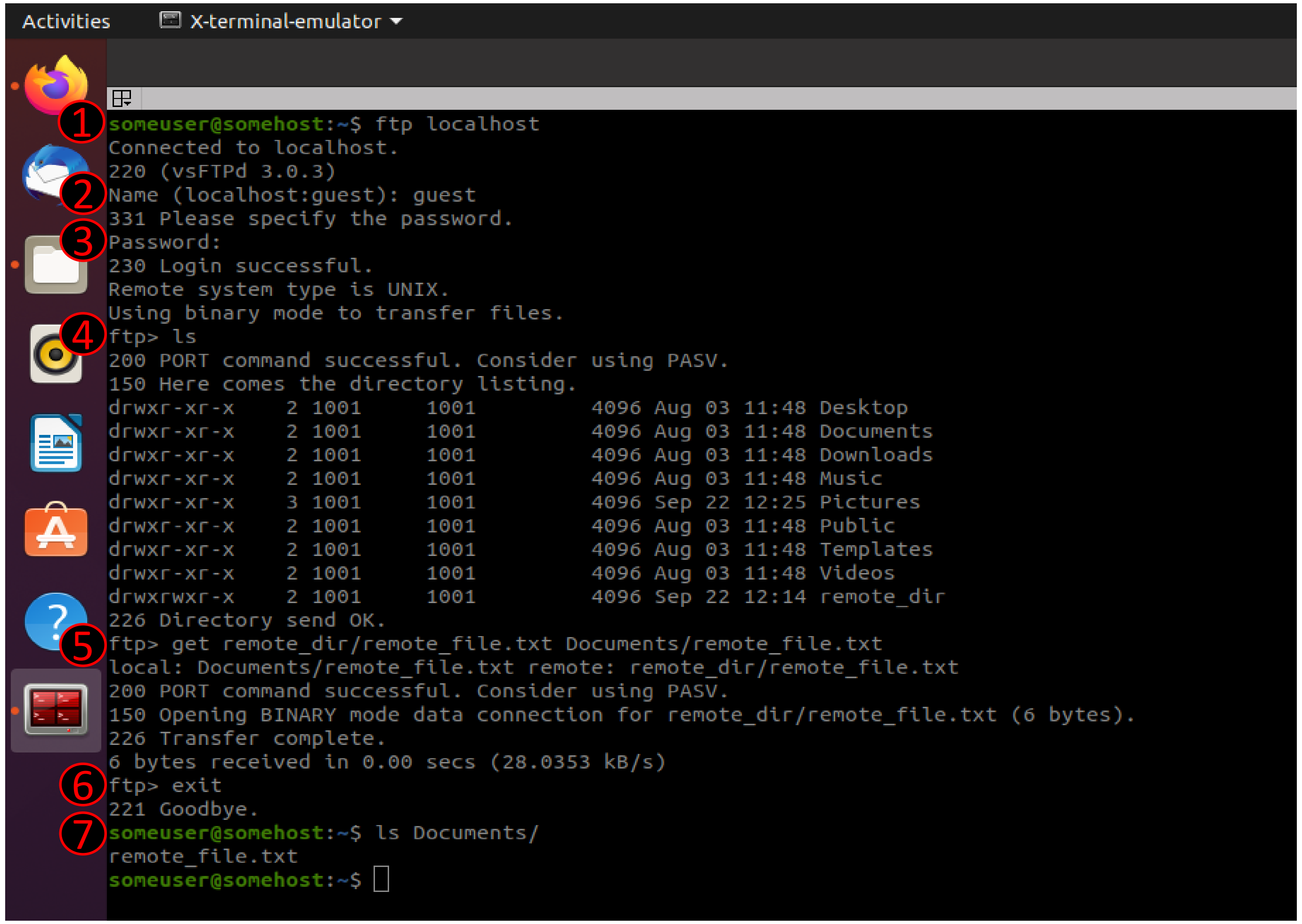
- On the server machine:
  - Install vsftpd:
    - $ sudo apt-get install vsftpd
  - Check ssh server running
    - $ service vsftpd status (check if daemon is running)

- On the client machine:
  - Install ftp:
    - $ sudo apt-get install ftp (it is already available in Ubuntu)
  - Connect to server:
    - $ ftp ADDRESS (usr and pass will be asked)
  - Close connection:
    - $ exit

Note: The server side of this application is implemented as a **daemon**: a program that runs in background and waits for clients to connects. This is a common approach.

Application Layer

FTP Example (get a file)



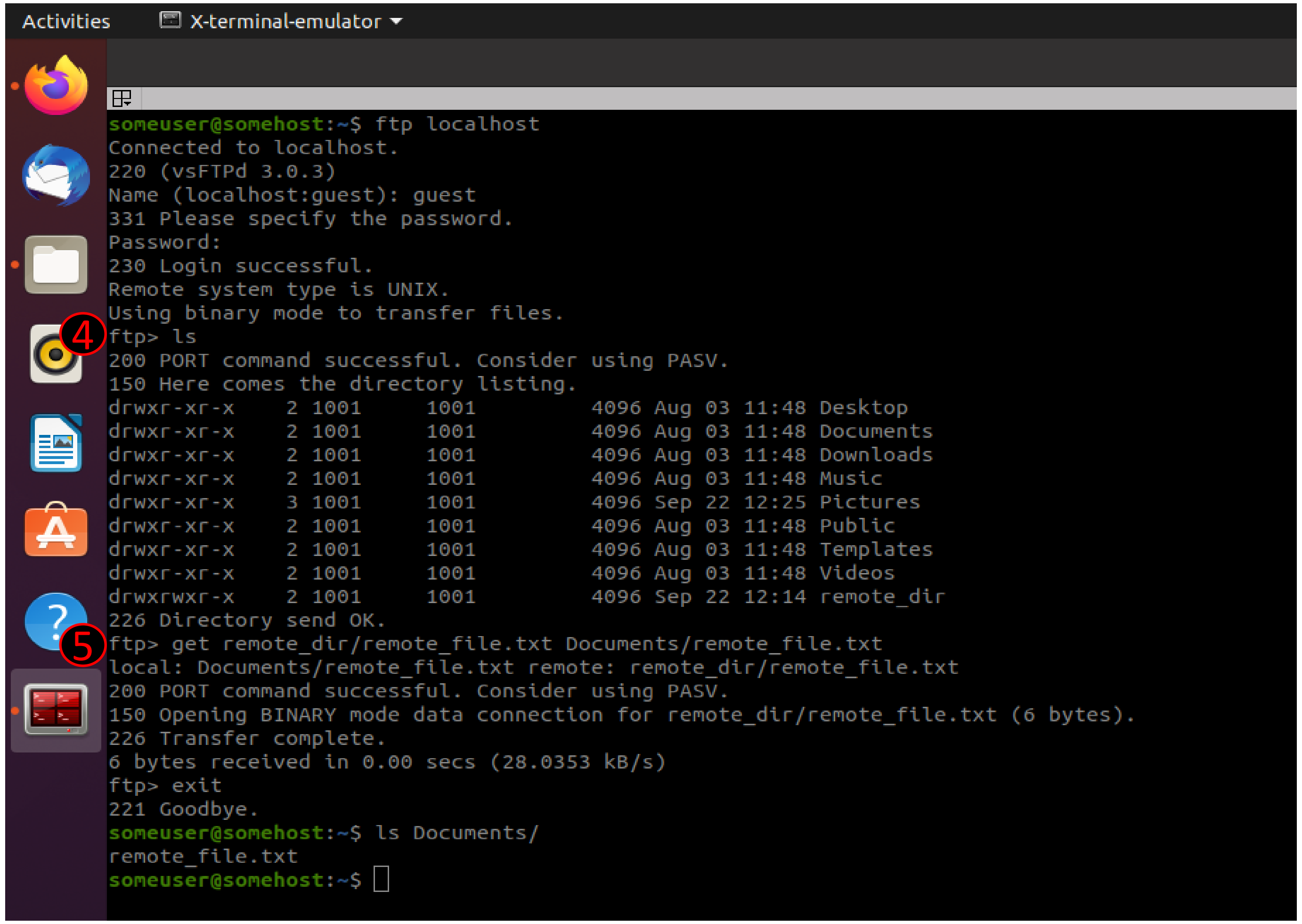
```
someuser@somehost:~$ ftp localhost
Connected to localhost.
220 (vsFTPd 3.0.3)
Name (localhost:guest): guest
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Desktop
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Documents
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Downloads
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Music
drwxr-xr-x    3 1001     1001         4096 Sep 22 12:25 Pictures
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Public
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Templates
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Videos
drwxrwxr-x    2 1001     1001         4096 Sep 22 12:14 remote_dir
226 Directory send OK.
ftp> get remote_dir/remote_file.txt Documents/remote_file.txt
local: Documents/remote_file.txt remote: remote_dir/remote_file.txt
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for remote_dir/remote_file.txt (6 bytes).
226 Transfer complete.
6 bytes received in 0.00 secs (28.0353 kB/s)
ftp> exit
221 Goodbye.
someuser@somehost:~$ ls Documents/
remote_file.txt
someuser@somehost:~$
```

Here we connect to ourselves (localhost), but an IP address can be specified:

1. Run ftp with the address of the remote host.
2. Insert username with which you want to log on remote host.
3. Insert password
4. Run ls to see files and directories
5. Copy the file remote_dir/remote_file.txt into (local) Documents
6. Exit from ftp
7. Check the file

Application Layer

FTP Common Commands



```
someuser@somehost:~$ ftp localhost
Connected to localhost.
220 (vsFTPd 3.0.3)
Name (localhost:guest): guest
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Desktop
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Documents
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Downloads
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Music
drwxr-xr-x    3 1001     1001         4096 Sep 22 12:25 Pictures
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Public
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Templates
drwxr-xr-x    2 1001     1001         4096 Aug 03 11:48 Videos
drwxrwxr-x    2 1001     1001         4096 Sep 22 12:14 remote_dir
226 Directory send OK.
ftp> get remote_dir/remote_file.txt Documents/remote_file.txt
local: Documents/remote_file.txt remote: remote_dir/remote_file.txt
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for remote_dir/remote_file.txt (6 bytes).
226 Transfer complete.
6 bytes received in 0.00 secs (28.0353 kB/s)
ftp> exit
221 Goodbye.
someuser@somehost:~$ ls Documents/
remote_file.txt
someuser@somehost:~$
```

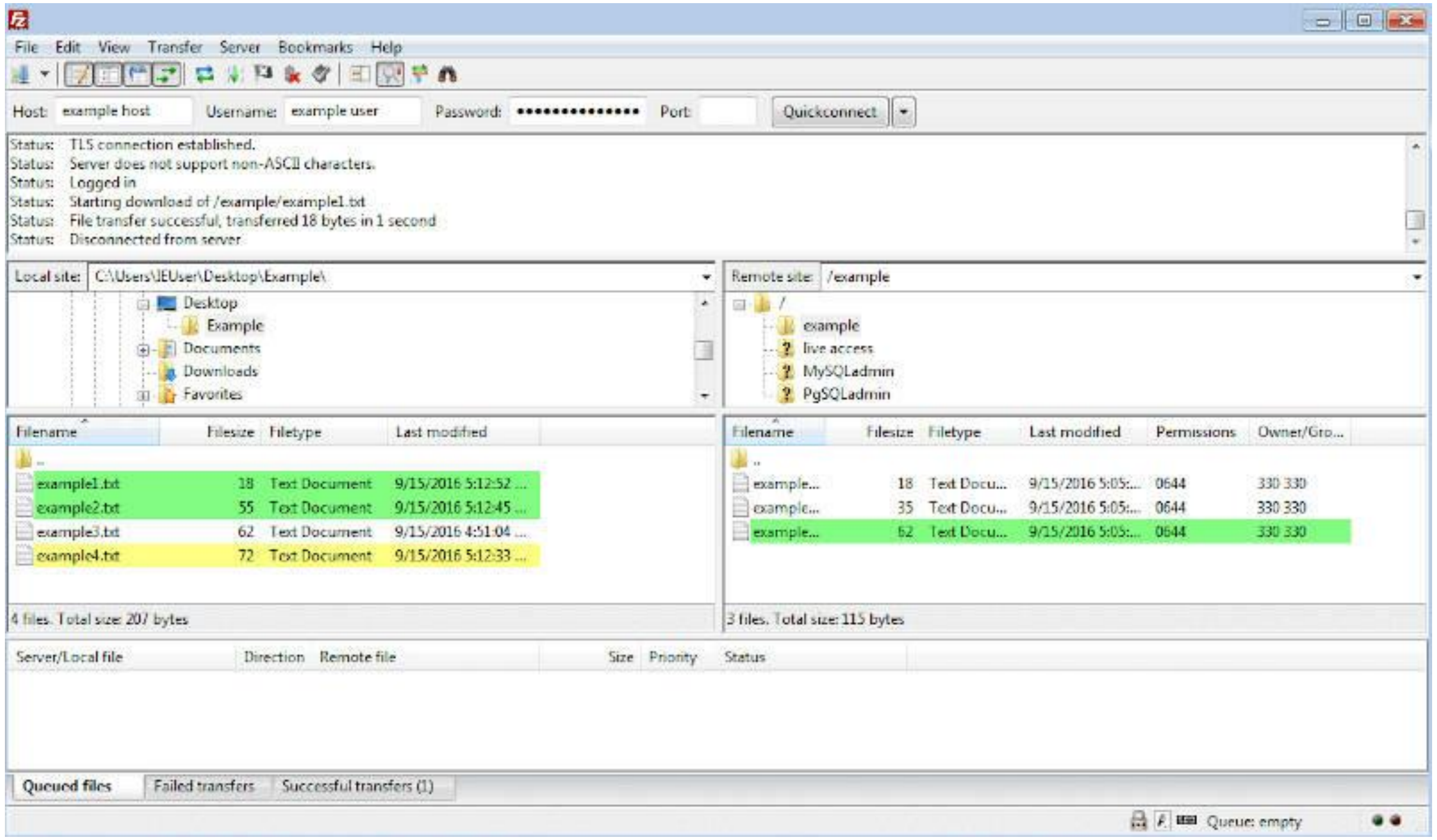
Common FTP commands:

- help - List all available FTP commands.
- cd - Change directory on remote machine.
- lcd - Change directory on local machine.
- ls - View the names of the files and directories in the current remote directory.
- mkdir - Create a new directory within the remote directory.
- pwd - Print the current working directory on the remote machine.
- delete - Delete a file in the current remote directory.
- rmdir- Remove a directory in the current remote directory.
- get - Copies a file from the remote server to the local machine.
- put - Copies a file from the local machine to the remote machine.

Application Layer

FTP Example

- Ubuntu has also GUI-based clients like **nautilus** (standard directory explorer) or **filezilla**.



Host: example host Username: example user Password: ••••••••••••••• Port: Quickconnect

```
Status:   TLS connection established.
Status:   Server does not support non-ASCII characters.
Status:   Logged in
Status:   Starting download of /example/example1.txt
Status:   File transfer successful, transferred 18 bytes in 1 second
Status:   Disconnected from server
```

Local site: C:\Users\IEUser\Desktop\Example\

Desktop
Example
Documents
Downloads
Favorites

Remote site: /example

/
example
live access
MySQLadmin
PgSQLadmin

Filename	Filesize	Filetype	Last modified
..			
example1.txt	18	Text Document	9/15/2016 5:12:52 ...
example2.txt	55	Text Document	9/15/2016 5:12:45 ...
example3.txt	62	Text Document	9/15/2016 4:51:04 ...
example4.txt	72	Text Document	9/15/2016 5:12:33 ...

4 files. Total size: 207 bytes

Filename	Filesize	Filetype	Last modified	Permissions	Owner/Gro...
..					
example...	18	Text Docu...	9/15/2016 5:05:...	0644	330 330
example...	35	Text Docu...	9/15/2016 5:05:...	0644	330 330
example...	62	Text Docu...	9/15/2016 5:05:...	0644	330 330

3 files. Total size: 115 bytes

Server/Local file	Direction	Remote file	Size	Priority	Status

Screenshot of filezilla's GUI: filezilla is a cross-platform FTP client (Linux, Windows, Mac OS).

Application Layer

Telnet and SSH

- Telnet (short for teletype network) is a client/server application protocol that provides access to **virtual terminals** of remote systems on local area networks or the Internet.

- The Secure Shell Protocol (SSH) is a replacement for the unsecured Telnet (SSH is a cryptographic protocol) for operating network services over an unsecured network. It is still based on a client-server architecture.

- Both Telnet and SSH have two components:
  - The **protocol** that specifies how messages are structured and how hosts communicate within a session.
  - A **software application** implementing the protocol (client-side and a server-side software).

Application Layer

SSH Example

- In Linux we can use openssh as a client/server application that allows SSH between two hosts.

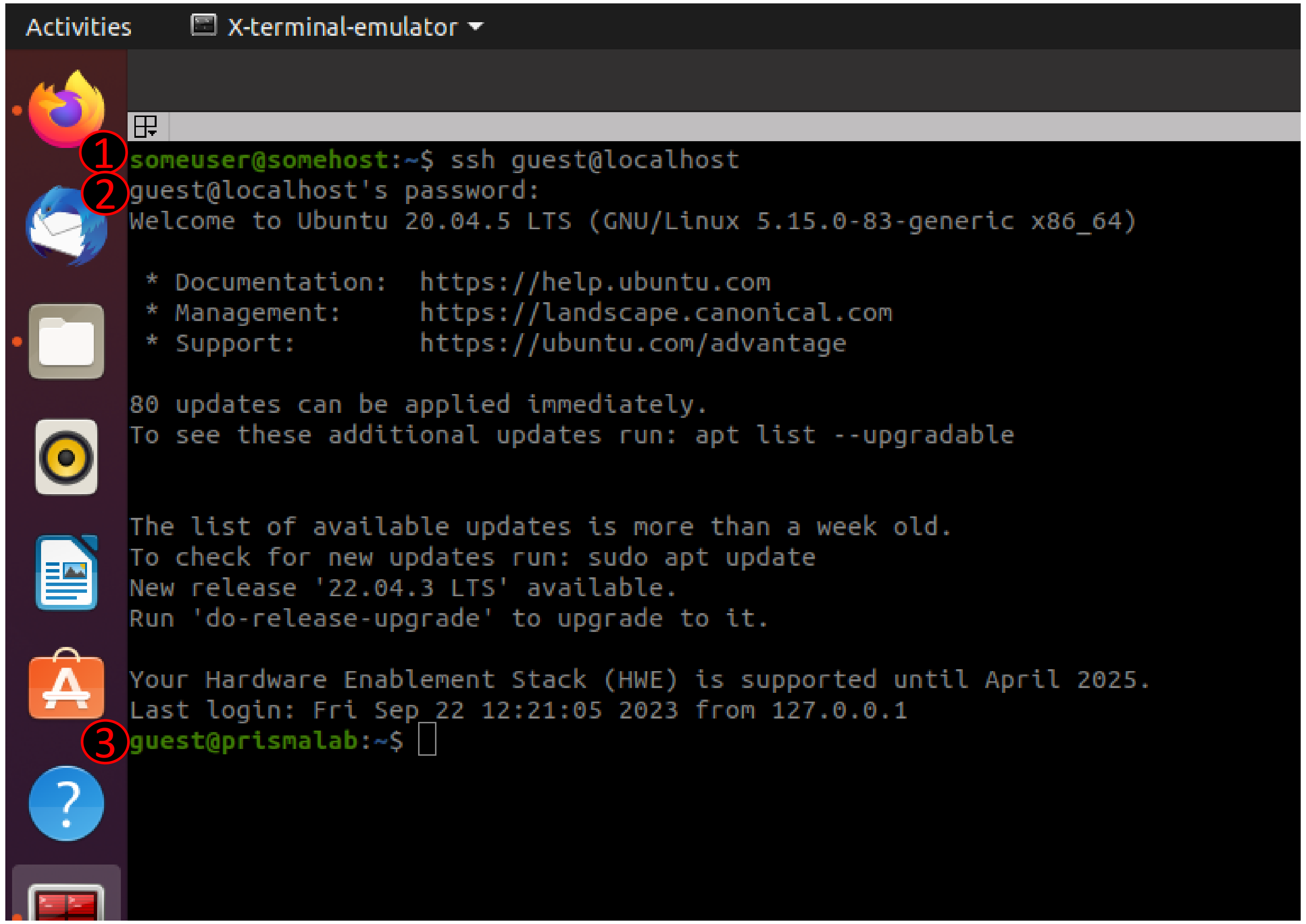
- On the server machine:
  - Install openssh:
    - $ sudo apt-get install openssh-server
  - Check ssh server running
    - $ ssh localhost (try connection with yourself)
      or
      $ sudo service ssh status (check if daemon is running)

- On the client machine:
  - Install openssh:
    - $ sudo apt-get install openssh-client
  - Connect to server:
    - $ ssh USERNAME@ADDRESS (pass will be asked)
  - Close connection:
    - $ exit

Note: Once you are connected to the remote users, you may use all commands the shell can offer.

Application Layer

SSH Example



```
someuser@somehost:~$ ssh guest@localhost
guest@localhost's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-83-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

80 updates can be applied immediately.
To see these additional updates run: apt list --upgradable


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
New release '22.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Fri Sep 22 12:21:05 2023 from 127.0.0.1
guest@prismalab:~$ 
```

Here we connect to ourselves (localhost), but an IP address can be specified:

1. Run ssh with username@address of the remote host.
2. Insert password
3. The terminal will show username@hostname of the remote shell. Now you may enter commands or close the connection (exit command).