

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

MCS 260 Lecture 38
Introduction to Computer Science
Jan Verschelde, 26 November 2007

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Transmission Media

types of connections

The unit for transmission speed is **bps**, bits per second.

Four categories of media used for data transmission:

1. **twisted pair wire:** pair of copper wires used for telephone communications. Transmission speeds vary from 2,400 to 33,600 bps.
2. **coaxial cable:** for many telephone lines or television signal. Data transmission over short distances guarantees speeds up to 10^7 bps.
3. **optical fibre:** signals transmitted via light-emitting diodes encode bits as presence or absence of light. Transmission speeds reach up to 10^9 bps.
4. **electromagnetic waves:** wireless over short distances or via satellites across long distances.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module `socket` in Python

implementing a simple client/server

Transmission Media

types of connections

The unit for transmission speed is bps , bits per second.

Four categories of media used for data transmission:

1. **twisted pair wire:** pair of copper wires used for telephone communications. Transmission speeds vary from 2,400 to 33,600 bps .
2. **coaxial cable:** for many telephone lines or television signal. Data transmission over short distances guarantees speeds up to 10^7 bps .
3. **optical fibre:** signals transmitted via light-emitting diodes encode bits as presence or absence of light. Transmission speeds reach up to 10^9 bps .
4. **electromagnetic waves:** wireless over short distances or via satellites across long distances.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module `socket` in Python

implementing a simple client/server

Transmission Media

types of connections

The unit for transmission speed is bps , bits per second.

Four categories of media used for data transmission:

1. **twisted pair wire:** pair of copper wires used for telephone communications. Transmission speeds vary from 2,400 to 33,600 bps .
2. **coaxial cable:** for many telephone lines or television signal. Data transmission over short distances guarantees speeds up to 10^7 bps .
3. **optical fibre:** signals transmitted via light-emitting diodes encode bits as presence or absence of light. Transmission speeds reach up to 10^9 bps .
4. **electromagnetic waves:** wireless over short distances or via satellites across long distances.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module `socket` in Python

implementing a simple client/server

Transmission Media

types of connections

The unit for transmission speed is bps , bits per second.

Four categories of media used for data transmission:

1. **twisted pair wire:** pair of copper wires used for telephone communications. Transmission speeds vary from 2,400 to 33,600 bps .
2. **coaxial cable:** for many telephone lines or television signal. Data transmission over short distances guarantees speeds up to 10^7 bps .
3. **optical fibre:** signals transmitted via light-emitting diodes encode bits as presence or absence of light. Transmission speeds reach up to 10^9 bps .
4. **electromagnetic waves:** wireless over short distances or via satellites across long distances.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module `socket` in Python

implementing a simple client/server

Transmission Media

types of connections

The unit for transmission speed is bps , bits per second.

Four categories of media used for data transmission:

1. **twisted pair wire:** pair of copper wires used for telephone communications. Transmission speeds vary from 2,400 to 33,600 bps .
2. **coaxial cable:** for many telephone lines or television signal. Data transmission over short distances guarantees speeds up to 10^7 bps .
3. **optical fibre:** signals transmitted via light-emitting diodes encode bits as presence or absence of light. Transmission speeds reach up to 10^9 bps .
4. **electromagnetic waves:** wireless over short distances or via satellites across long distances.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module `socket` in Python

implementing a simple client/server

Network Topologies

structure of computer networks

MCS 260 L-38

26 November 2007

Computer
Networks

transmission media and
network topologies

client/server architecture

layers, protocols, and
sockets

Network
Programming

a simple client/server
interaction

the module socket in Python
implementing a simple
client/server

Three most common regular topologies:



a *star network*: one central node
is connected to all other nodes



a *ring network*: nodes form a closed circuit,
messages circle around the ring of nodes



a *bus network*: all nodes on a single bus,
used in the Von Neumann architecture

Network Topologies

structure of computer networks

MCS 260 L-38

26 November 2007

Computer
Networks

transmission media and
network topologies

client/server architecture

layers, protocols, and
sockets

Network
Programming

a simple client/server
interaction

the module socket in Python

implementing a simple
client/server

Three most common regular topologies:



a *star network*: one central node
is connected to all other nodes



a *ring network*: nodes form a closed circuit,
messages circle around the ring of nodes



a *bus network*: all nodes on a single bus,
used in the Von Neumann architecture

Network Topologies

structure of computer networks

MCS 260 L-38

26 November 2007

Computer
Networks

transmission media and
network topologies

client/server architecture

layers, protocols, and
sockets

Network
Programming

a simple client/server
interaction

the module socket in Python
implementing a simple
client/server

Three most common regular topologies:



a *star network*: one central node
is connected to all other nodes



a *ring network*: nodes form a closed circuit,
messages circle around the ring of nodes



a *bus network*: all nodes on a single bus,
used in the Von Neumann architecture

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Client/Server Architecture

for software and hardware

Client/server architecture defines the communication between two computers:
one is the server and the other acts as the client.

A client places a request or order to a server.
The server processes the request.

The client does not need to know how the server processes the requests.

We distinguish between software and hardware client/server architectures:

- ▶ web and database servers offer software services;
- ▶ file and print servers offer hardware services.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Client/Server Architecture

for software and hardware

Client/server architecture defines the communication between two computers:
one is the server and the other acts as the client.

A client places a request or order to a server.
The server processes the request.

The client does not need to know how the server processes the requests.

We distinguish between software and hardware client/server architectures:

- ▶ web and database servers offer software services;
- ▶ file and print servers offer hardware services.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Client/Server Architecture

for software and hardware

Client/server architecture defines the communication between two computers:
one is the server and the other acts as the client.

A client places a request or order to a server.
The server processes the request.

The client does not need to know how the server processes the requests.

We distinguish between software and hardware client/server architectures:

- ▶ web and database servers offer software services;
- ▶ file and print servers offer hardware services.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Client/Server Architecture

for software and hardware

Client/server architecture defines the communication between two computers:
one is the server and the other acts as the client.

A client places a request or order to a server.
The server processes the request.

The client does not need to know how the server processes the requests.

We distinguish between software and hardware client/server architectures:

- ▶ web and database servers offer software services;
- ▶ file and print servers offer hardware services.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Client/Server Architecture

for software and hardware

Client/server architecture defines the communication between two computers:

one is the server and the other acts as the client.

A client places a request or order to a server.

The server processes the request.

The client does not need to know how the server processes the requests.

We distinguish between software and hardware client/server architectures:

- ▶ web and database servers offer software services;
- ▶ file and print servers offer hardware services.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Four Layers in Internet Communication

application, transport, network, and link

1. The **application layer** consists of client and server software. The application prepares the message and defines the destination address.
2. The **transport layer** formats the message by chopping it into packets attaching a sequence number and destination address to each packet. When receiving, packets are collected and reassembled into a message.
3. The **network layer** determines the (intermediate) address for each packet. This layer detects when a packet has reached its final destination.
4. The **link layer** is responsible for the actual transmission of packets.

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

Four Layers in Internet Communication

application, transport, network, and link

1. The **application layer** consists of client and server software. The application prepares the message and defines the destination address.
2. The **transport layer** formats the message by chopping it into packets attaching a sequence number and destination address to each packet. When receiving, packets are collected and reassembled into a message.
3. The **network layer** determines the (intermediate) address for each packet. This layer detects when a packet has reached its final destination.
4. The **link layer** is responsible for the actual transmission of packets.

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

Four Layers in Internet Communication

application, transport, network, and link

1. The **application layer** consists of client and server software. The application prepares the message and defines the destination address.
2. The **transport layer** formats the message by chopping it into packets attaching a sequence number and destination address to each packet. When receiving, packets are collected and reassembled into a message.
3. The **network layer** determines the (intermediate) address for each packet. This layer detects when a packet has reached its final destination.
4. The **link layer** is responsible for the actual transmission of packets.

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

Four Layers in Internet Communication

application, transport, network, and link

1. The **application layer** consists of client and server software. The application prepares the message and defines the destination address.
2. The **transport layer** formats the message by chopping it into packets attaching a sequence number and destination address to each packet. When receiving, packets are collected and reassembled into a message.
3. The **network layer** determines the (intermediate) address for each packet. This layer detects when a packet has reached its final destination.
4. The **link layer** is responsible for the actual transmission of packets.

Computer Networks

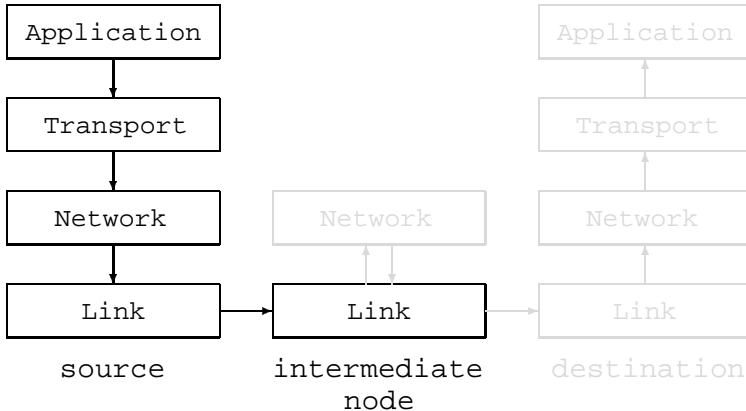
transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

Following a Message

through all four layers



Computer Networks

transmission media and network topologies
client/server architecture

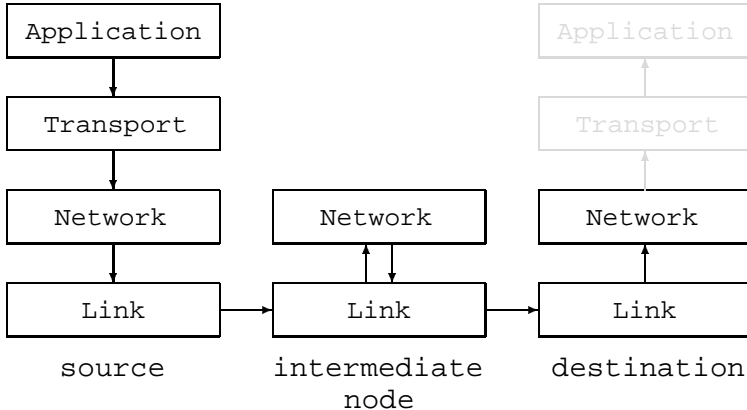
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

Following a Message

through all four layers



Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

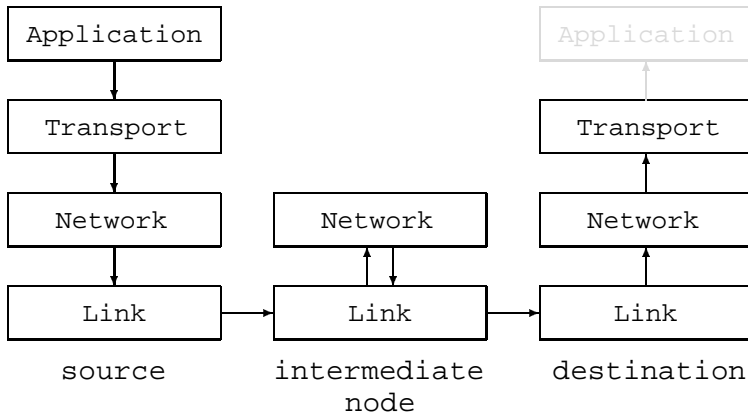
Network Programming

a simple client/server interaction

the module `socket` in Python
implementing a simple client/server

Following a Message

through all four layers



Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module `socket` in Python

implementing a simple client/server

Network Protocols

TCP and UDP

Network protocols are rules for network communication.

We consider two types of protocols:

TCP Transmission Control Protocol

First a message is sent that data is coming.

Only after the receiver acknowledges this message will the sender send the data.

All successful transmissions are confirmed, and retransmissions are acknowledged.

UDP User Datagram Protocol

Unlike TCP, no connection is established prior to sending data. The sender just carries on after sending a message.

TCP is connection oriented, UDP is connectionless.

TCP is more reliable whereas UDP is more streamlined.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Network Protocols

TCP and UDP

Network protocols are rules for network communication.

We consider two types of protocols:

TCP Transmission Control Protocol

First a message is sent that data is coming.

Only after the receiver acknowledges this message will the sender send the data.

All successful transmissions are confirmed, and retransmissions are acknowledged.

UDP User Datagram Protocol

Unlike TCP, no connection is established prior to sending data. The sender just carries on after sending a message.

TCP is connection oriented, UDP is connectionless.

TCP is more reliable whereas UDP is more streamlined.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Network Protocols

TCP and UDP

Network protocols are rules for network communication.

We consider two types of protocols:

TCP Transmission Control Protocol

First a message is sent that data is coming.

Only after the receiver acknowledges this message will the sender send the data.

All successful transmissions are confirmed, and retransmissions are acknowledged.

UDP User Datagram Protocol

Unlike TCP, no connection is established prior to sending data. The sender just carries on after sending a message.

TCP is connection oriented, UDP is connectionless.

TCP is more reliable whereas UDP is more streamlined.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Network Protocols

TCP and UDP

Network protocols are rules for network communication.

We consider two types of protocols:

TCP Transmission Control Protocol

First a message is sent that data is coming.

Only after the receiver acknowledges this message will the sender send the data.

All successful transmissions are confirmed, and retransmissions are acknowledged.

UDP User Datagram Protocol

Unlike TCP, no connection is established prior to sending data. The sender just carries on after sending a message.

TCP is connection oriented, UDP is connectionless.

TCP is more reliable whereas UDP is more streamlined.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Network Protocols

TCP and UDP

Network protocols are rules for network communication.

We consider two types of protocols:

TCP Transmission Control Protocol

First a message is sent that data is coming.

Only after the receiver acknowledges this message will the sender send the data.

All successful transmissions are confirmed, and retransmissions are acknowledged.

UDP User Datagram Protocol

Unlike TCP, no connection is established prior to sending data. The sender just carries on after sending a message.

TCP is connection oriented, UDP is connectionless.

TCP is more reliable whereas UDP is more streamlined.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Network Protocols

TCP and UDP

Network protocols are rules for network communication.

We consider two types of protocols:

TCP Transmission Control Protocol

First a message is sent that data is coming.

Only after the receiver acknowledges this message will the sender send the data.

All successful transmissions are confirmed, and retransmissions are acknowledged.

UDP User Datagram Protocol

Unlike TCP, no connection is established prior to sending data. The sender just carries on after sending a message.

TCP is connection oriented, UDP is connectionless.

TCP is more reliable whereas UDP is more streamlined.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Network Protocols

TCP and UDP

Network protocols are rules for network communication.

We consider two types of protocols:

TCP Transmission Control Protocol

First a message is sent that data is coming.

Only after the receiver acknowledges this message will the sender send the data.

All successful transmissions are confirmed, and retransmissions are acknowledged.

UDP User Datagram Protocol

Unlike TCP, no connection is established prior to sending data. The sender just carries on after sending a message.

TCP is connection oriented, UDP is connectionless.

TCP is more reliable whereas UDP is more streamlined.

Computer Networks

transmission media and network topologies

client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Sockets are objects programs use to connect.

Sockets were introduced in 1981 in BSD Unix. Originally used for communication between processes, i.e.: between two programs on same computer.

Sockets support communication across platforms, independent of the operating system.

In addition to the IP address of the computer, both server and client must use the same *port*. A port is a 16-bit integer, some are reserved for particular protocols. Any port between 1,024 and 65,535 is free.

Sockets support both TCP and UDP.

Computer Networks

transmission media and network topologies
client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python
implementing a simple client/server

Sockets are objects programs use to connect.

Sockets were introduced in 1981 in BSD Unix. Originally used for communication between processes, i.e.: between two programs on same computer.

Sockets support communication across platforms, independent of the operating system.

In addition to the IP address of the computer, both server and client must use the same *port*. A port is a 16-bit integer, some are reserved for particular protocols. Any port between 1,024 and 65,535 is free.

Sockets support both TCP and UDP.

Computer Networks

transmission media and network topologies
client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python
implementing a simple client/server

Sockets are objects programs use to connect.

Sockets were introduced in 1981 in BSD Unix. Originally used for communication between processes, i.e.: between two programs on same computer.

Sockets support communication across platforms, independent of the operating system.

In addition to the IP address of the computer, both server and client must use the same *port*. A port is a 16-bit integer, some are reserved for particular protocols. Any port between 1,024 and 65,535 is free.

Sockets support both TCP and UDP.

Computer Networks

transmission media and network topologies
client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python
implementing a simple client/server

Sockets are objects programs use to connect.

Sockets were introduced in 1981 in BSD Unix. Originally used for communication between processes, i.e.: between two programs on same computer.

Sockets support communication across platforms, independent of the operating system.

In addition to the IP address of the computer, both server and client must use the same *port*. A port is a 16-bit integer, some are reserved for particular protocols. Any port between 1,024 and 65,535 is free.

Sockets support both TCP and UDP.

Computer Networks

transmission media and network topologies
client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python
implementing a simple client/server

Sockets are objects programs use to connect.

Sockets were introduced in 1981 in BSD Unix. Originally used for communication between processes, i.e.: between two programs on same computer.

Sockets support communication across platforms, independent of the operating system.

In addition to the IP address of the computer, both server and client must use the same *port*. A port is a 16-bit integer, some are reserved for particular protocols. Any port between 1,024 and 65,535 is free.

Sockets support both TCP and UDP.

Computer Networks

transmission media and network topologies
client/server architecture

layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python
implementing a simple client/server

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

Computer Networks

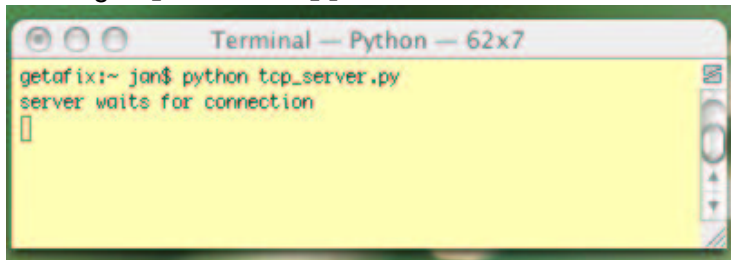
transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

A simple Client/Server Interaction

Running `tcp_server.py`:

A screenshot of a terminal window titled "Terminal — Python — 62x7". The terminal shows the command `getafix:~ jan$ python tcp_server.py` being executed. The output is `server waits for connection` followed by a cursor on a new line. The terminal has a yellow background and a dark border.

```
Terminal — Python — 62x7
getafix:~ jan$ python tcp_server.py
server waits for connection
█
```

The client program `tcp_client.py`
will run on another terminal.

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

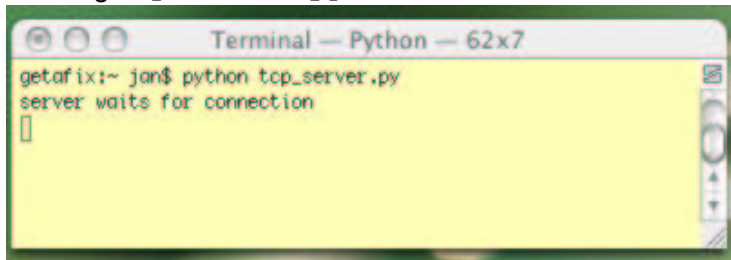
Network Programming

a simple client/server interaction

the module `socket` in Python
implementing a simple client/server

A simple Client/Server Interaction

Running `tcp_server.py`:

A screenshot of a terminal window titled "Terminal — Python — 62x7". The terminal shows the command `getafix:~ jan$ python tcp_server.py` being executed, followed by the output `server waits for connection` and a cursor on a new line.

```
Terminal — Python — 62x7
getafix:~ jan$ python tcp_server.py
server waits for connection
█
```

The client program `tcp_client.py`
will run on another terminal.

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

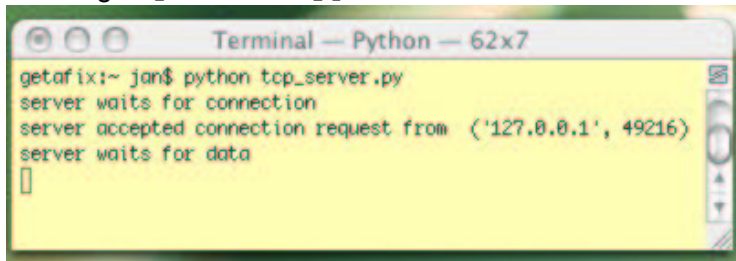
Network Programming

a simple client/server interaction

the module `socket` in Python
implementing a simple client/server

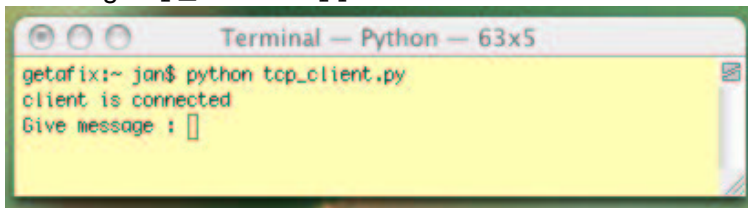
Connecting Client and Server

Running `tcp_server.py` on one terminal:

A terminal window titled "Terminal — Python — 62x7" with a yellow background. The text inside shows the execution of a Python script. The prompt is "getafix:~ jan\$". The user enters "python tcp_server.py". The output is "server waits for connection", "server accepted connection request from ('127.0.0.1', 49216)", and "server waits for data". There is a cursor on a new line.

```
getafix:~ jan$ python tcp_server.py
server waits for connection
server accepted connection request from ('127.0.0.1', 49216)
server waits for data
█
```

Running `tcp_client.py` on another terminal:

A terminal window titled "Terminal — Python — 63x5" with a yellow background. The text inside shows the execution of a Python script. The prompt is "getafix:~ jan\$". The user enters "python tcp_client.py". The output is "client is connected" and "Give message :". There is a cursor on a new line.

```
getafix:~ jan$ python tcp_client.py
client is connected
Give message : █
```

Computer

Networks

transmission media and
network topologies
client/server architecture
layers, protocols, and
sockets

Network

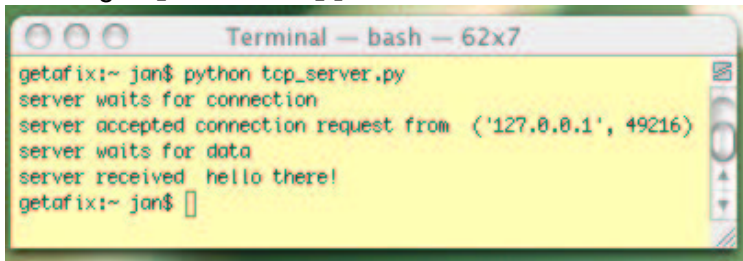
Programming

a simple client/server
interaction

the module `socket` in Python
implementing a simple
client/server

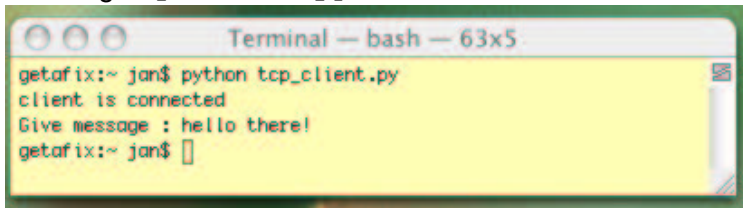
Passing Data from Client to Server

Running `tcp_server.py` on one terminal:

A terminal window titled "Terminal — bash — 62x7" with a yellow background. The text inside shows the execution of a Python script. The prompt is "getafix:~ jan\$". The user enters "python tcp_server.py". The output is: "server waits for connection", "server accepted connection request from ('127.0.0.1', 49216)", "server waits for data", "server received hello there!". The prompt returns to "getafix:~ jan\$".

```
getafix:~ jan$ python tcp_server.py
server waits for connection
server accepted connection request from ('127.0.0.1', 49216)
server waits for data
server received hello there!
getafix:~ jan$
```

Running `tcp_client.py` on another terminal:

A terminal window titled "Terminal — bash — 63x5" with a yellow background. The text inside shows the execution of a Python script. The prompt is "getafix:~ jan\$". The user enters "python tcp_client.py". The output is: "client is connected", "Give message : hello there!". The prompt returns to "getafix:~ jan\$".

```
getafix:~ jan$ python tcp_client.py
client is connected
Give message : hello there!
getafix:~ jan$
```

Computer

Networks

transmission media and
network topologies
client/server architecture
layers, protocols, and
sockets

Network

Programming

a simple client/server
interaction

the module `socket` in Python
implementing a simple
client/server

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

Sockets in Python

the socket module

```
from socket import *
```

The socket module exports the method `socket()` which returns an object representing a socket.

The first argument of `socket` is the Address Family (AF):

- ▶ `AF_UNIX` : for UNIX sockets;
- ▶ `AF_INET` : most commonly used for internet

`AF_INET` supports both TCP and UDP, given respectively by `SOCK_STREAM` and `SOCK_DGRAM` as second argument of `socket()`. For example:

```
sock = socket(AF_INET, SOCK_STREAM)
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module `socket` in Python

implementing a simple client/server

Sockets in Python

the socket module

```
from socket import *
```

The `socket` module exports the method `socket()` which returns an object representing a socket.

The first argument of `socket` is the Address Family (AF):

- ▶ `AF_UNIX` : for UNIX sockets;
- ▶ `AF_INET` : most commonly used for internet

`AF_INET` supports both TCP and UDP, given respectively by `SOCK_STREAM` and `SOCK_DGRAM` as second argument of `socket()`. For example:

```
sock = socket(AF_INET, SOCK_STREAM)
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module `socket` in Python

implementing a simple client/server

Sockets in Python

the socket module

```
from socket import *
```

The `socket` module exports the method `socket()` which returns an object representing a socket.

The first argument of `socket` is the Address Family (AF):

- ▶ `AF_UNIX` : for UNIX sockets;
- ▶ `AF_INET` : most commonly used for internet

`AF_INET` supports both TCP and UDP, given respectively by `SOCK_STREAM` and `SOCK_DGRAM` as second argument of `socket()`. For example:

```
sock = socket(AF_INET, SOCK_STREAM)
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module `socket` in Python

implementing a simple client/server

Sockets in Python

the socket module

```
from socket import *
```

The `socket` module exports the method `socket()` which returns an object representing a socket.

The first argument of `socket` is the Address Family (AF):

- ▶ `AF_UNIX` : for UNIX sockets;
- ▶ `AF_INET` : most commonly used for internet

`AF_INET` supports both TCP and UDP, given respectively by `SOCK_STREAM` and `SOCK_DGRAM` as second argument of `socket()`. For example:

```
sock = socket(AF_INET, SOCK_STREAM)
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module `socket` in Python

implementing a simple client/server

Sockets in Python

the socket module

```
from socket import *
```

The `socket` module exports the method `socket()` which returns an object representing a socket.

The first argument of `socket` is the Address Family (AF):

- ▶ `AF_UNIX`: for UNIX sockets;
- ▶ `AF_INET`: most commonly used for internet

`AF_INET` supports both TCP and UDP, given respectively by `SOCK_STREAM` and `SOCK_DGRAM` as second argument of `socket()`. For example:

```
sock = socket(AF_INET, SOCK_STREAM)
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module `socket` in Python

implementing a simple client/server

Sockets in Python

the socket module

```
from socket import *
```

The `socket` module exports the method `socket()` which returns an object representing a socket.

The first argument of `socket` is the Address Family (AF):

- ▶ `AF_UNIX`: for UNIX sockets;
- ▶ `AF_INET`: most commonly used for internet

`AF_INET` supports both TCP and UDP, given respectively by `SOCK_STREAM` and `SOCK_DGRAM` as second argument of `socket()`. For example:

```
sock = socket(AF_INET, SOCK_STREAM)
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module `socket` in Python

implementing a simple client/server

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

Defining Connections

the methods `bind` and `connect`

After a socket is created, **both server and client** define

```
server_address = (hostname, number)
```

To bind the socket to the address, **the server** does

```
sock.bind(server_address)
```

and **the client** contacts the server then via

```
sock.connect(server_address)
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module `socket` in Python
implementing a simple client/server

Defining Connections

the methods `bind` and `connect`

After a socket is created, **both server and client** define

```
server_address = (hostname, number)
```

To bind the socket to the address, **the server** does

```
sock.bind(server_address)
```

and **the client** contacts the server then via

```
sock.connect(server_address)
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module `socket` in Python
implementing a simple client/server

Defining Connections

the methods `bind` and `connect`

After a socket is created, **both server and client** define

```
server_address = (hostname, number)
```

To bind the socket to the address, **the server** does

```
sock.bind(server_address)
```

and **the client** contacts the server then via

```
sock.connect(server_address)
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module `socket` in Python
implementing a simple client/server

Taking Requests

the methods `listen` and `accept`

With `listen()` the server indicates how many incoming connections will be accepted:

```
sock.listen(2) # accept at most 2 connections
```

The server takes requests via the `accept()` method:

```
client, client_address = sock.accept()
```

The `accept()` method returns

1. a socket `client` for receiving data;
2. the address of the client.

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module `socket` in Python
implementing a simple client/server

Taking Requests

the methods `listen` and `accept`

With `listen()` the server indicates how many incoming connections will be accepted:

```
sock.listen(2) # accept at most 2 connections
```

The server takes requests via the `accept()` method:

```
client, client_address = sock.accept()
```

The `accept()` method returns

1. a socket `client` for receiving data;
2. the address of the client.

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module `socket` in Python
implementing a simple client/server

Taking Requests

the methods `listen` and `accept`

With `listen()` the server indicates how many incoming connections will be accepted:

```
sock.listen(2) # accept at most 2 connections
```

The server takes requests via the `accept()` method:

```
client, client_address = sock.accept()
```

The `accept()` method returns

1. a socket `client` for receiving data;
2. the address of the client.

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module `socket` in Python
implementing a simple client/server

Taking Requests

the methods `listen` and `accept`

With `listen()` the server indicates how many incoming connections will be accepted:

```
sock.listen(2) # accept at most 2 connections
```

The server takes requests via the `accept()` method:

```
client, client_address = sock.accept()
```

The `accept()` method returns

1. a socket `client` for receiving data;
2. the address of the client.

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module `socket` in Python
implementing a simple client/server

Sending and Receiving Data

the methods `send` and `recv`

The **client** sends data with `send()`

```
sock.send(data)
```

The **server** receives data applying `recv()`

```
data = client.recv(buffer)
```

to the socket `client` obtained with `accept()`.

When all is over, **both client and server** do

```
sock.close()
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module `socket` in Python
implementing a simple client/server

Sending and Receiving Data

the methods `send` and `recv`

The **client** sends data with `send()`

```
sock.send(data)
```

The **server** receives data applying `recv()`

```
data = client.recv(buffer)
```

to the socket `client` obtained with `accept()`.

When all is over, **both client and server** do

```
sock.close()
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module `socket` in Python
implementing a simple client/server

Sending and Receiving Data

the methods `send` and `recv`

The **client** sends data with `send()`

```
sock.send(data)
```

The **server** receives data applying `recv()`

```
data = client.recv(buffer)
```

to the socket `client` obtained with `accept()`.

When all is over, **both client and server** do

```
sock.close()
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module `socket` in Python
implementing a simple client/server

the program tcp_server.py

```
from socket import *

hostname = ''      # blank so any address can be used
number = 11267    # number for the port
buffer = 80       # size of the buffer

server_address = (hostname, number)
server = socket(AF_INET, SOCK_STREAM)
server.bind(server_address)
server.listen(2)

print 'server waits for connection'
client, client_address = server.accept()
print 'server accepted connection request from ', \
      client_address

print 'server waits for data'
data = client.recv(buffer)
print 'server received ', data

server.close()
```

Computer Networks

transmission media and
network topologies
client/server architecture
layers, protocols, and
sockets

Network Programming

a simple client/server
interaction
the module socket in Python
**implementing a simple
client/server**

the program tcp_server.py

```

from socket import *

hostname = ''      # blank so any address can be used
number = 11267    # number for the port
buffer = 80       # size of the buffer

server_address = (hostname, number)
server = socket(AF_INET, SOCK_STREAM)
server.bind(server_address)
server.listen(2)

print 'server waits for connection'
client, client_address = server.accept()
print 'server accepted connection request from ', \
      client_address

print 'server waits for data'
data = client.recv(buffer)
print 'server received ', data

server.close()

```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

the program tcp_server.py

```

from socket import *

hostname = ''      # blank so any address can be used
number = 11267    # number for the port
buffer = 80       # size of the buffer

server_address = (hostname, number)
server = socket(AF_INET, SOCK_STREAM)
server.bind(server_address)
server.listen(2)

print 'server waits for connection'
client, client_address = server.accept()
print 'server accepted connection request from ', \
      client_address
print 'server waits for data'
data = client.recv(buffer)
print 'server received ', data

server.close()

```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

the program tcp_server.py

```

from socket import *

hostname = ''      # blank so any address can be used
number = 11267    # number for the port
buffer = 80       # size of the buffer

server_address = (hostname, number)
server = socket(AF_INET, SOCK_STREAM)
server.bind(server_address)
server.listen(2)

print 'server waits for connection'
client, client_address = server.accept()
print 'server accepted connection request from ', \
      client_address

print 'server waits for data'
data = client.recv(buffer)
print 'server received ', data

server.close()

```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

the program tcp_server.py

```

from socket import *

hostname = ''      # blank so any address can be used
number = 11267    # number for the port
buffer = 80       # size of the buffer

server_address = (hostname, number)
server = socket(AF_INET, SOCK_STREAM)
server.bind(server_address)
server.listen(2)

print 'server waits for connection'
client, client_address = server.accept()
print 'server accepted connection request from ', \
      client_address

print 'server waits for data'
data = client.recv(buffer)
print 'server received ', data

server.close()

```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

the program tcp_server.py

```

from socket import *

hostname = ''      # blank so any address can be used
number = 11267    # number for the port
buffer = 80       # size of the buffer

server_address = (hostname, number)
server = socket(AF_INET, SOCK_STREAM)
server.bind(server_address)
server.listen(2)

print 'server waits for connection'
client, client_address = server.accept()
print 'server accepted connection request from ', \
      client_address
print 'server waits for data'
data = client.recv(buffer)
print 'server received ', data

server.close()

```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module socket in Python
implementing a simple client/server

the program tcp_client.py

```
from socket import *

hostname = 'localhost' # on same host
number = 11267         # same port number
buffer = 80           # size of the buffer

server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)

print 'client is connected'
data = raw_input('Give message : ')
client.send(data)

client.close()
```

Computer Networks

transmission media and
network topologies
client/server architecture
layers, protocols, and
sockets

Network Programming

a simple client/server
interaction

the module socket in Python

**implementing a simple
client/server**

the program tcp_client.py

```
from socket import *

hostname = 'localhost' # on same host
number = 11267         # same port number
buffer = 80           # size of the buffer

server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)

print 'client is connected'
data = raw_input('Give message : ')
client.send(data)

client.close()
```

Computer Networks

transmission media and
network topologies
client/server architecture
layers, protocols, and
sockets

Network Programming

a simple client/server
interaction

the module socket in Python

implementing a simple
client/server

the program tcp_client.py

```
from socket import *

hostname = 'localhost' # on same host
number = 11267         # same port number
buffer = 80           # size of the buffer

server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)

print 'client is connected'
data = raw_input('Give message : ')
client.send(data)

client.close()
```

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction

the module socket in Python

implementing a simple client/server

the program tcp_client.py

Computer Networks

transmission media and
network topologies
client/server architecture
layers, protocols, and
sockets

Network Programming

a simple client/server
interaction

the module socket in Python

implementing a simple
client/server

```
from socket import *

hostname = 'localhost' # on same host
number = 11267         # same port number
buffer = 80           # size of the buffer

server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)

print 'client is connected'
data = raw_input('Give message : ')
client.send(data)

client.close()
```

the program tcp_client.py

```
from socket import *

hostname = 'localhost' # on same host
number = 11267         # same port number
buffer = 80           # size of the buffer

server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)

print 'client is connected'
data = raw_input('Give message : ')
client.send(data)

client.close()
```

Computer Networks

transmission media and
network topologies
client/server architecture
layers, protocols, and
sockets

Network Programming

a simple client/server
interaction

the module socket in Python

implementing a simple
client/server

Summary + Assignments

We started chapter 12 in *Making Use of Python*; see also §17.1-5 in *The Art & Craft of Computing*.

Assignments:

1. Extend the client/server interaction to simulate a password dialogue. After receiving data from a client, the server returns `access granted` or `access denied` depending on whether the received data matches the password.
2. Consider the simplified blackjack game we implemented as our third project. How would you implement the game using a client/server protocol?
3. Describe how you would implement the game of rock, paper, and scissors in a client/server manner.

Computer Networks

transmission media and network topologies
client/server architecture
layers, protocols, and sockets

Network Programming

a simple client/server interaction
the module `socket` in Python
implementing a simple client/server