

practical guide to networks

Nikola Bebić / rač / petnica

quick intro

1. host a
2. host b
3. communication between them
4. ???
5. profit

tcp

The Transmission Control Protocol (TCP) is one of the main protocols of the internet protocol suite. [...]

TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network.

summary (tcp)

- everything* sent is received, unchanged and in exactly the same order
- connection based – *client* initiates a connection to a *server*
 - client / server dichotomy

tcp client

1. create a `socket`
2. `connect` to a server (usually separate from step 1.)
 - `server = host + port`
3. communicate with server (`send` / `recv`)
4. `close` the connection

tcp client example

```
from socket import *

# create a socket object
sock = socket(AF_INET, SOCK_STREAM)

# connect to a server = (host, port)
sock.connect(("www.example.com", 1234)) # ( address )

# communicate with server
sock.send(b'data here')                # ( data )
data = sock.recv(1024)                  # ( length )

# close the connection
sock.close()
```

for other languages check out <https://rosettacode.org/>

what is 'communication'

- depends on protocol
 - segmentation, ...
- usually synchronous
 - client sends something, server responds, ...
 - easy to implement

demo client

- client sends a message of variable length, terminated with `'\r\n'`
- server responds with a message, also terminated
- repeat

task: read a message from console, send it, print response

tcp server

1. create a `socket` (same as client)
2. `bind` to an address (ip, port)
3. `listen` on the socket
4. while running (can be in parallel)
 - i. `accept` a client
 - ii. communicate with client (`send` / `recv`)
 - iii. `close` the connection

tcp server example

```
from socket import *

sock = socket(AF_INET, SOCK_STREAM)

sock.bind(('0.0.0.0', 1234))      # ( address )

sock.listen()                    # ( [backlog] )

while true:
    client, addr = sock.accept()

    # communicate with client
    client.send(b'data here')     # ( data )
    data = client.recv(42)        # ( length )

    client.close()
```

python threading

```
from threading import Thread

def do_something(a, b, c):
    # handle things
    print(a + b - c)

# somewhere else
thr = Thread(target=do_something, args=(x, y, z))

thr.start()
# spins a new thread and calls `do_something(x, y, z)`
# this one continues execution
```

demo server

- client connects, sends name as LONGSTRING
 - two bytes unsigned integer, length of the string
 - data as astring of ascii characters
 - hint python `struct` module
- after that, client sends messages as LONGSTRING

task: print each message as it is received

note: there are multiple clients

udp

User Datagram Protocol (UDP) is one of the core members of the Internet Protocol suite.

With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network.

It has no handshaking dialogues, [...] there is no guarantee of delivery, ordering, or duplicate protection.

summary (udp)

- possible loss of data, reordering or (rarely) duplication
- "fire and forget" networking – no need to connect, just send data
- receiving: just listen for whatever comes through
 - bind an address first

udp send/recv example

```
from socket import *

# create a datagram socket (udp)
sock = socket(AF_INET, SOCK_DGRAM)

# bind to address, (needed only if receiving)
sock.bind( ('0.0.0.0', 7373) ) # ( address )

# send some data
sock.sendto(b'data here', ('www.example.com', 5678)) # ( data, address )

# receive some data
data, addr = sock.recvfrom(1500) # ( buffer )

# close the connection
sock.close()
```

udp demo pt1

- `username = LONGSTRING`
- `message = LONGSTRING`
- `example:`

```
00 04 70 72 6f 66 00 0b 68 65 6c 6c 6f 20 74 68 65 72 65
```

- `read messages from console, send them with your username`

udp demo pt2

- extend previous part
- same format, receive on same port
- print received messages

udp broadcast

- "fire and forget"
 - what if we fired to "everyone"?
- special address for "everyone" – broadcast address
 - highest address in the subnet
 - `192.168.x.255` for `/24` networks
 - `'<broadcast>'` in python
- broadcast sockets need to be marked as such – `so_broadcast`

```
sock.setopt(SOL_SOCKET, SO_BROADCAST, 1)
```

udp demo pt3

- extend the previous example to use broadcast addresses