Telecommunications Network

Practice 2

File transfer

Open file in binary mode

```
with open ('input.txt', 'rb') as f:
  f.read(128)
```

- read(x):
  - read x byte (if binary mode)
  - read x character (if text mode)

> “When size is omitted or negative, the entire contents of the file will be read and returned; it’s your problem if the file is twice as large as your machine’s memory.”
> – python.org

Struct transfer

- Convert the data to binary and then back

```
import struct
values = (1, 'ab'.encode(), 2.7)
packer = struct.Struct('i 2s f') # Int, char[2], float
packed_data = packer.pack(*values)
print(packed_data)

unpacker = struct.Struct('i 2s f')
unpacked_data = unpacker.unpack(packed_data)
print(unpacked_data)
```

```
b'\x01\x00\x00\x00ab\x00\x00\xcd\xcc,@'
(1, b'ab', 2.700000047683716)
```

Note

int is usually 4 bytes, char is usually 1 byte, so a small number might be transferred as a string to save space

Struct properties

“Xs” (e.g. “2s”) denotes an X long byte object (pl. b'abc')

```
import struct
values = (1, 'ab', 2.7)
packer = struct.Struct('i 2s f')
packed_data = packer.pack(*values)
```

error: argument for 's' must be a bytes object

Correctly

```
values = (1, 'ab'.encode(), 2.7) # vagy values = (1, b'ab', 2.7)
packed_data = packer.pack(*values)
print(packed_data)
```

b'\x01\x00\x00\x00ab\x00\x00\xcd\xcc,@'

Struct properties

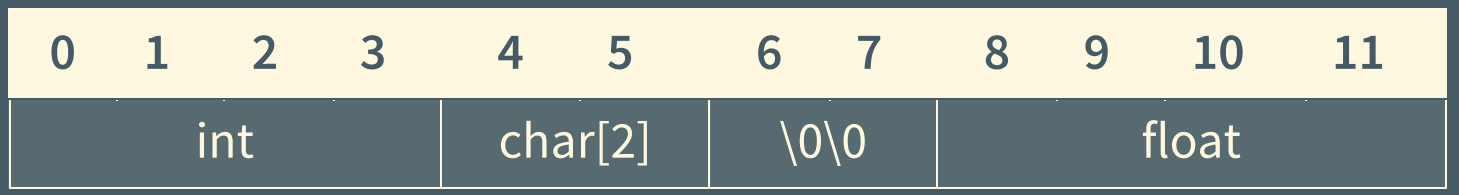
- Size of the struct in bytes:

```
import struct
packer = struct.Struct('i 2s f')
print(struct.calcsize('i 2s f'))
print(packer.size)
```

```
12
12
```

- i: int = 4, 2s: char[2] = 2, f: float = 4 $\Rightarrow 4 + 2 + 4 \neq 12$???
- Aligns the data members so that their starting position is divisible by the machine word length size (here 4)
- The Lost Art of Structure Packing

Struct memory layout



Struct

Character	Byte order
@	native
=	native
<	little-endian
>	big-endian
!	network(=big-endian)

Reference

Format	C Type	Python type	Size
x	pad byte	no value	
c	char	bytes of length 1	1
b	signed char	integer	1
B	unsigned char	integer	1
?	_Bool	bool	1
h	short	integer	2
H	unsigned short	integer	2
i	int	integer	4
I	unsigned int	integer	4
l	long	integer	4
L	unsigned long	integer	4
q	long long	integer	8
Q	unsigned long long	integer	8
n	ssize_t	integer	
N	size_t	integer	
e	(6)	float	2
f	float	float	4
d	double	float	8
s	char[]	bytes	
p	char[]	bytes	
P	void*	integer	

File handling

- seek(offset, whence): change position

```
with open('alma.txt', 'r') as f:
  sor = f.readline()
  print('jelenlegi sor:', sor.strip()) # jelenlegi sor 1. sor

  sor = f.readline()
  print('jelenlegi sor:', sor.strip()) # jelenlegi sor 2. sor

  f.seek(0, 0)                        # f.seek(offset, whence)

  sor = f.readline()
  print('jelenlegi sor:', sor.strip()) # jelenlegi sor 1. sor
```

- offset: read/write cursor position in the file
- whence: 0: absolute, 1: relative, 2: relative to the file's end

File handling

- Seek in binary file

```
import struct

packer = struct.Struct('i3si')

with open('dates.bin', 'wb') as f:
  for i in range(5):
    values = (2020 + i, b'jan', 10 + i)
    packed_data = packer.pack(*values)
    f.write(packed_data)

with open('dates.bin', 'rb') as f:
  f.seek(packer.size * 3)
  data = f.read(packer.size)
  print(packer.unpack(data))
```

Array of bytes vs string

- String → Array of bytes

```
import struct
str = 'hello'
print(str.encode())
print(struct.pack('8s', str.encode() ))
```

```
b'hello'
b'hello\x00\x00\x00'
```

- Array of bytes → String

```
import struct
d = struct.pack('8s', str.encode())
print(d)
print(d.decode().strip('\x00'))
```

```
b'hello\x00\x00\x00'
hello
```

Python socket, hostname resolution

- Use socket module

```
import socket
```

- gethostname()

```
hostname = socket.gethostname()
```

- gethostbyname()

```
hostip = socket.gethostbyname('www.example.org')
```

- gethostbyname_ex()

```
hostname, aliases, addresses = socket.gethostbyname_ex(host)
```

- gethostbyaddr()

```
hostname, aliases, addrs = socket.gethostbyaddr('157.181.161.79')
```

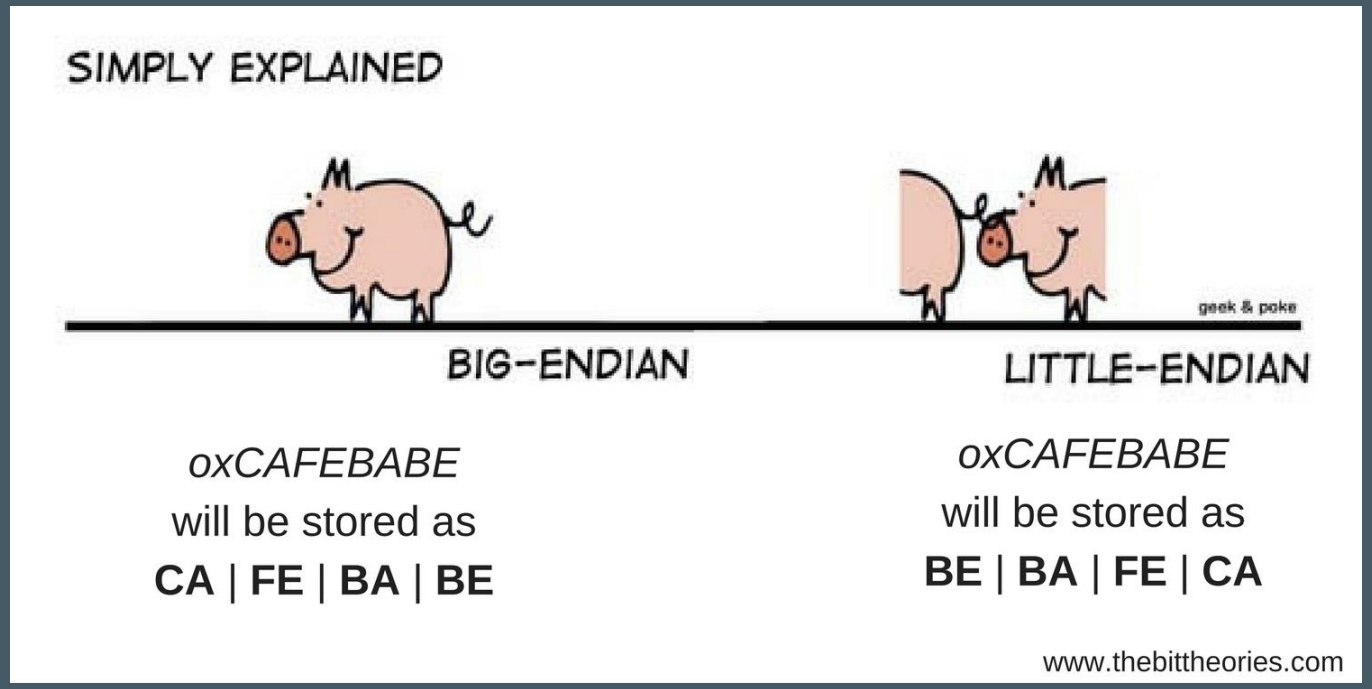
Port numbers

- Certain protocols have fixed port numbers and constants (transport protocols)!
- getservbyport()

```
import socket
print(socket.getservbyport(22, 'tcp'))
```

- Write the ports from 1 to 100 and their associated protocols/service names!

Little endian, big endian



- Encoding of 16 and 32-bit unsigned numbers
  - htons(), htonl() – host to network short / long
  - ntohs(), ntohl() – network to host short / long

Practice I.

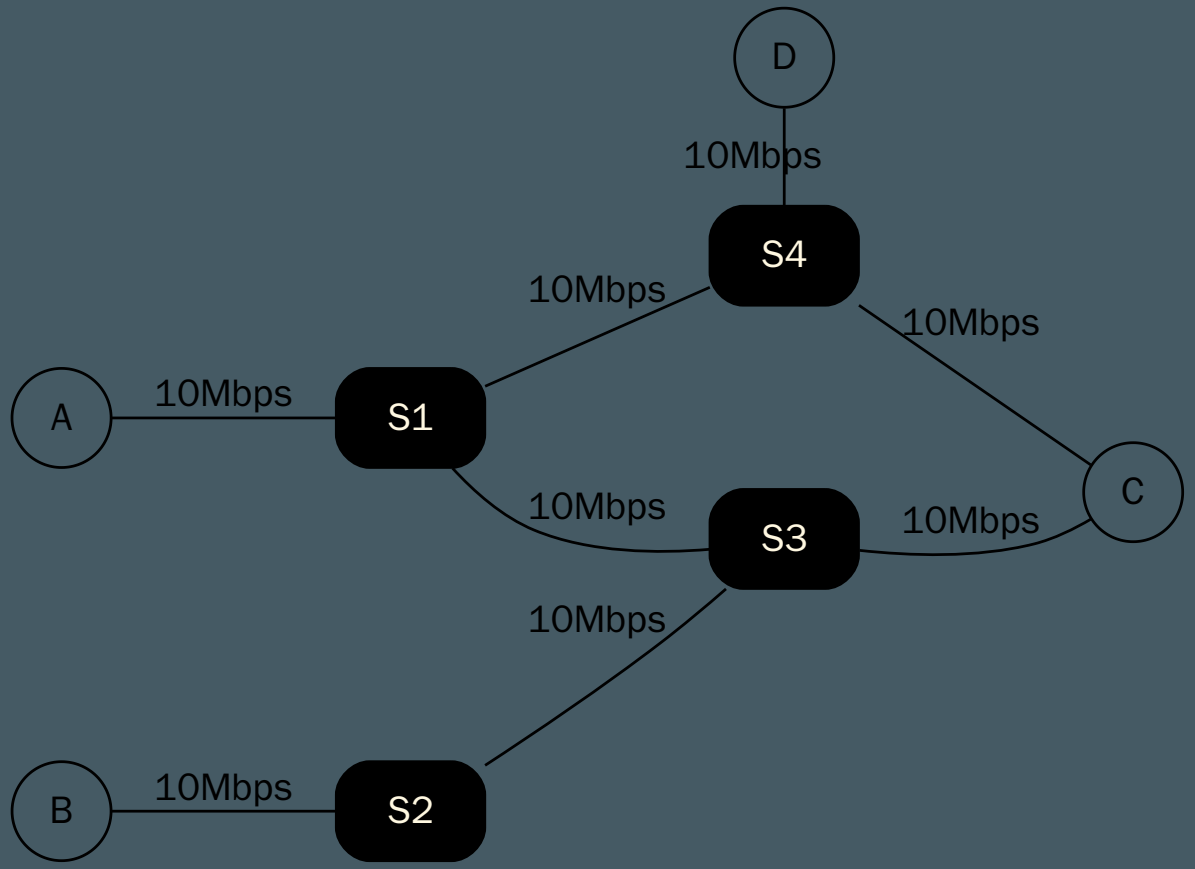
- Given a binary file, with the following structure:
  - Domain (20s), port (i)
- Write a python script supporting the following command line arguments:
  - port <line>: outputs what service belongs to the port of the line given as parameter
  - domain <line>: outputs the resolved IP address of the domain of the line given as parameter
  - If there is no parameter, outputs the host's name

Assignment I.

Circuit-switched networks

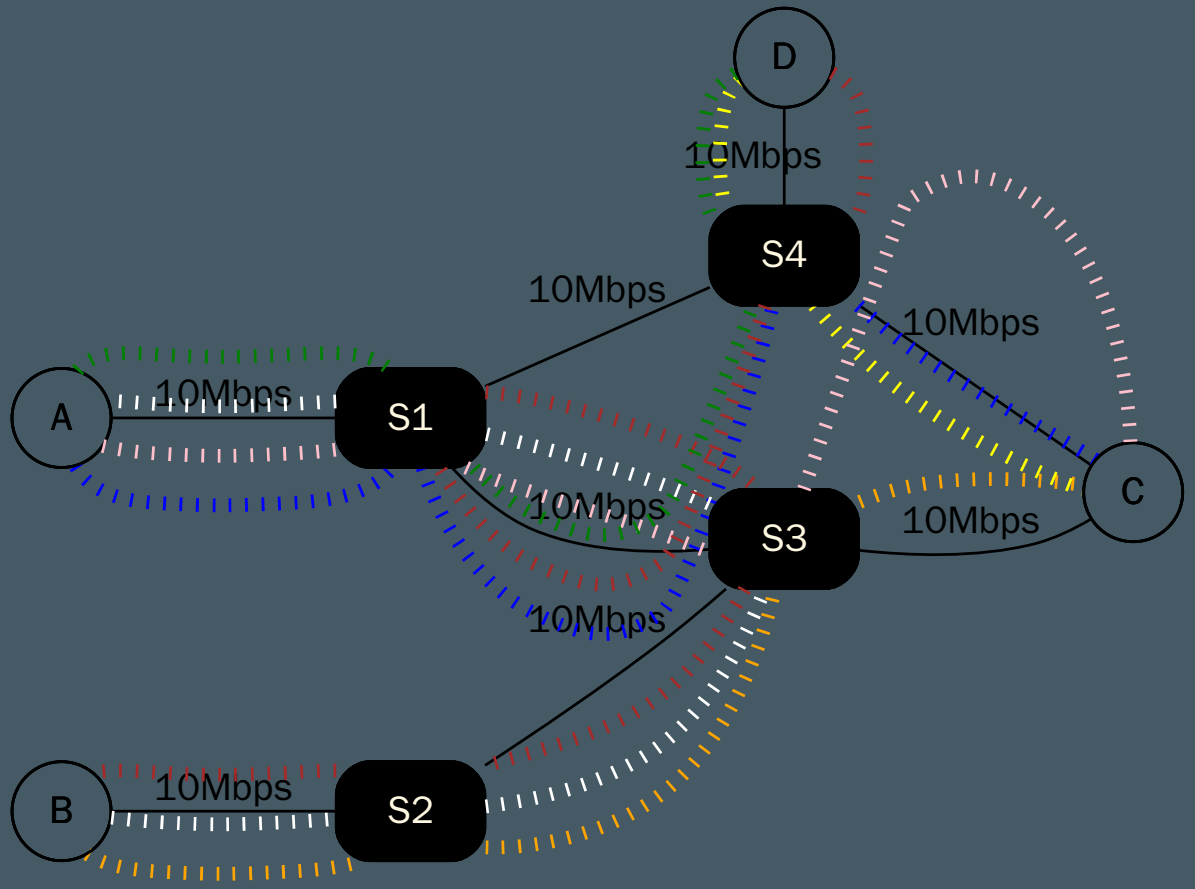
- Description
- Status of the test server

Topology - cs1.json



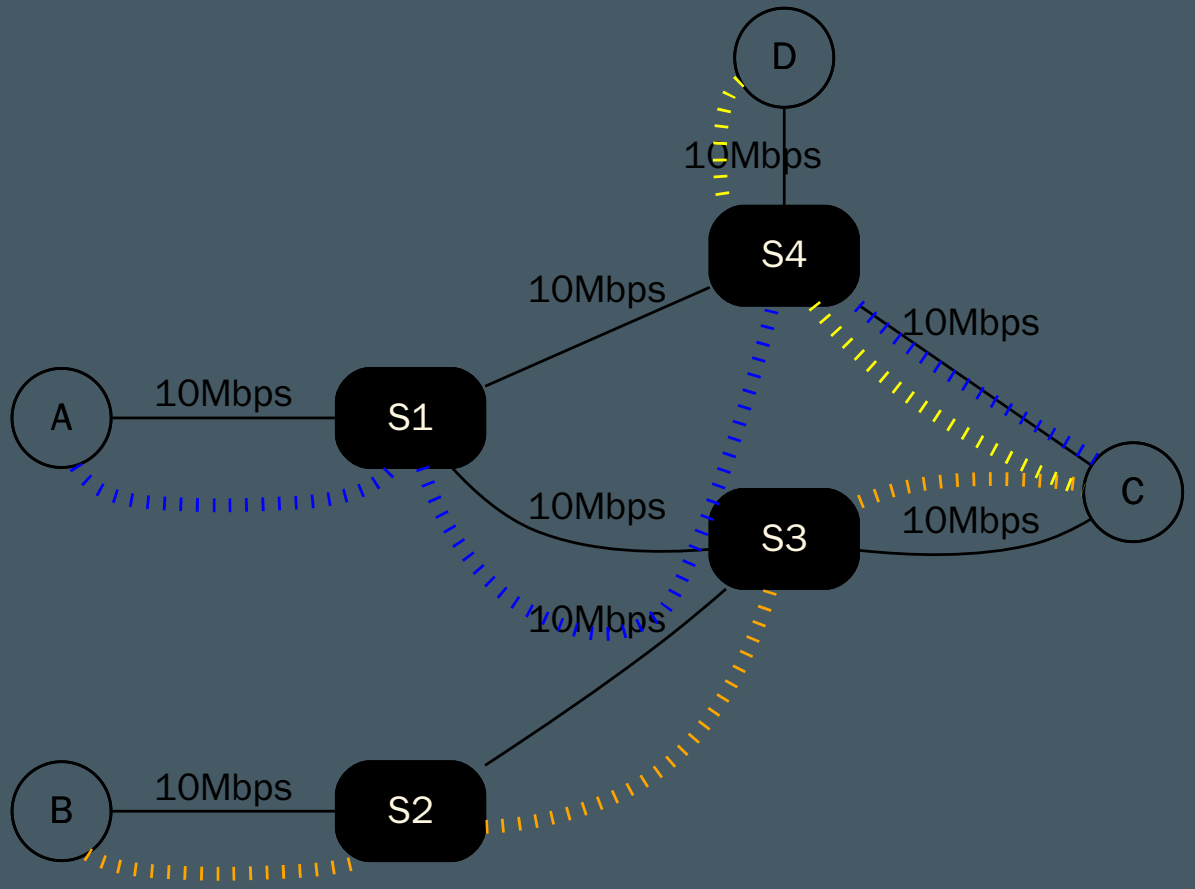

```
"end-points": [ "A", "B", "C", "D" ],
"switches": [ "S1", "S2", "S3", "S4" ],
"links" : [
  {
    "points" : [ "A", "S1" ],
    "capacity" : 10.0
  },
  {
    "points" : [ "B", "S2" ],
    "capacity" : 10.0
  },
  {
    "points" : [ "D", "S4" ],
    "capacity" : 10.0
  },
  {
    "points" : [ "S1", "S4" ],
    "capacity" : 10.0
  },
  {
    "points" : [ "S1", "S3" ],
    "capacity" : 10.0
  },
  {
    "points" : [ "S2", "S3" ],
    "capacity" : 10.0
  },
  {
    "points" : [ "S4", "C" ],
    "capacity" : 10.0
  },
  {
    "points" : [ "S3", "C" ],
    "capacity" : 10.0
  }
],
```

Possible circuits - cs1.json




```
"possible-circuits" : [
  ["D", "S4", "C"],
  ["C", "S4", "D"],
  ["A", "S1", "S4", "C"],
  ["A", "S1", "S3", "C"],
  ["C", "S4", "S1", "A"],
  ["C", "S3", "S1", "A"],
  ["B", "S2", "S3", "C"],
  ["C", "S3", "S2", "B"],
  ["B", "S2", "S3", "S1", "A"],
  ["A", "S1", "S3", "S2", "B"],
  ["D", "S4", "S1", "S3", "S2", "B"],
  ["B", "S2", "S3", "S1", "S4", "D"],
  ["A", "S1", "S4", "D"],
  ["D", "S4", "S1", "A"]
],
```

Demands - cs1.json




```
"simulation" : {
  "duration" : 11,
  "demands" : [
    {
      "start-time" : 1,
      "end-time" : 5,
      "end-points" : ["A", "C"],
      "demand" : 10.0
    },
    {
      "start-time" : 2,
      "end-time" : 10,
      "end-points" : ["B", "C"],
      "demand" : 10.0
    },
    {
      "start-time" : 6,
      "end-time" : 10,
      "end-points" : ["D", "C"],
      "demand" : 10.0
    },
    {
      "start-time" : 7,
      "end-time" : 10,
      "end-points" : ["A", "C"],
      "demand" : 10.0
    }
  ]
}
```

Task

Given the file cs1.json, which contains the description of an undirected graph. The graph contains end-points and switch nodes (switches). Edges (links) have capacity (real number). Let's say that we are in a circuit-switched network and we are using an RRP-like resource reservation protocol. Assuming Links are shared and narrow resources. The json contains the possible routes that can be created (possible-circuits), as well as the incoming circuit requests connecting two endpoints with a starting and ending point. The simulation starts at time t=1 and ends at time t=duration.

Write a program which simulates the allocation and deallocation of resources according to the topology, capacities and demands outlined in the given JSON file!

e.g.:

```
1. demand allocation: A<->C st:1 – successful
2. demand allocation: B<->C st:2 – successful
3. demand deallocation: A<->C st:5
4. demand allocation: D<->C st:6 – successful
5. demand allocation: A<->C st:7 – unsuccessful
...
```

Assignment I.

Arguments:

```
python3 program.py <cs1.json>
```

Output:

```
<event number>. <event name>: <node1><-><node2> st:<simulation time> [- (successful|unsuccessful)]
```

Submission: The program should be submitted through the TMS system in .zip format, which contains a single client.py file!

Deadline: See TMS