Telecommunications Network

Practice 6

Exercise I.

- Write a proxy server.
  - The proxy should receive the requests from the TCP calculator client.
  - It should forward all requests to the UDP calculator server.
  - It should also forward all responses coming from the server to the client.

Exercise II.

- Write a calculator client that asks a UDP server for the address of the TCP server.
  - The client should send the b”Hello Server” bytestring to the UDP server.
  - The server should reply with the address of the TCP server, to which the client can send the numbers and the operator.

Exercise III.

- Write a proxy that will forward messages between a web browser and a web server.
  - The proxy should forward the browser's requests to the server without change.
  - In the default case, the proxy should forward the server's response to the browser as well.
  - If the server's response contains the string "SzamHalo", then it should send a 404 error code instead.

Execution example:

```
python3 netProxy.py ggombos.web.elte.hu 9000
```

In the browser: localhost:9000

Exercise IV.

- Consider the following parity technique:
  - Interpret n data bit as a k x l bit matrix.
  - Compute a partiy bit (odd parity) for every column and extend our table with a row containing these new bits.
  - Send the data row by row.
- Example (k = 2, l = 3):

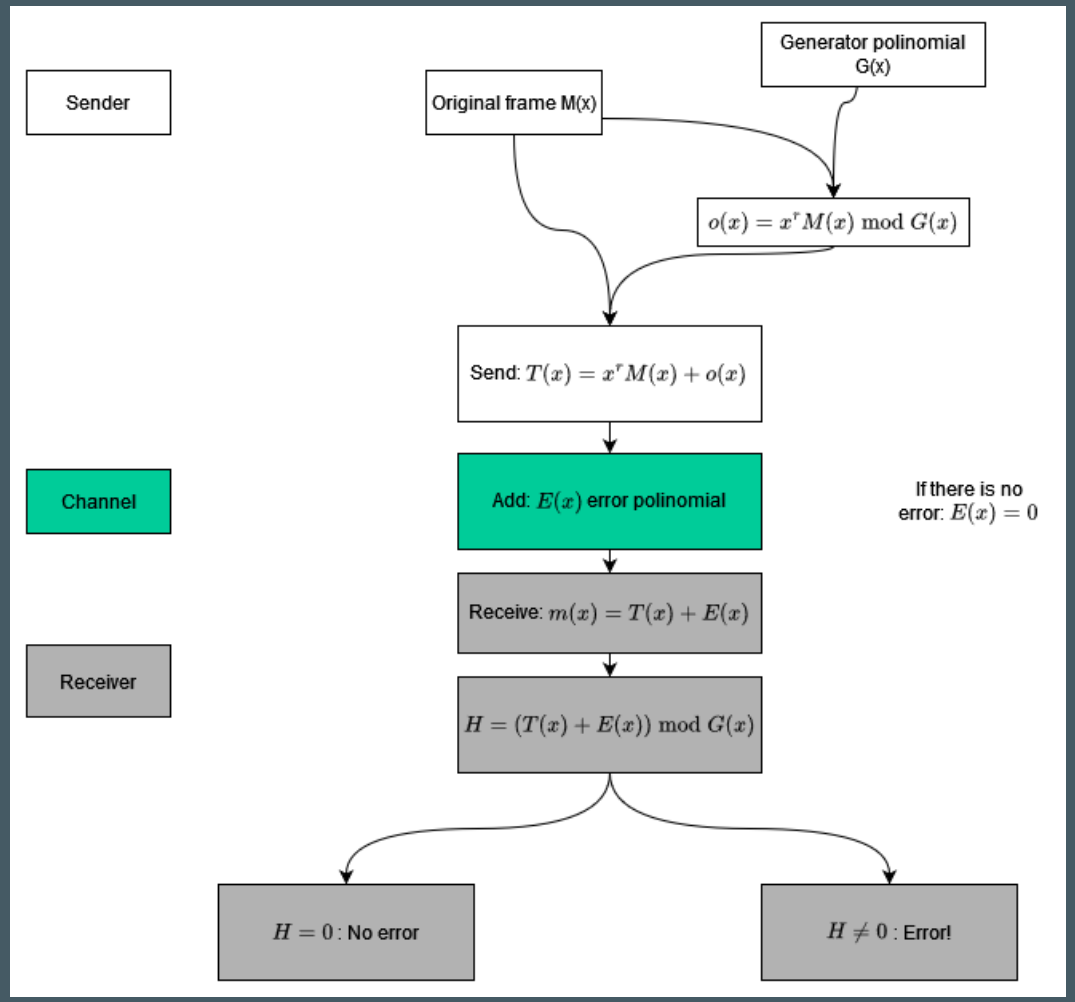
1	**0**	**1**
0	1	1
0	**0**	**1**

Exercise IV.

- How does this method behave in the case of simple bit errors and in the case of burst-like errors if k = 3 and l = 4? What is the maximal length of a bit sequence for which we can guarantee the detection of errors?
- Burst-like: Consequtive bits are faulty.
- Extend the matrix with a new column containing a parity bit for each row (two dimensional parity technique). How can we use this technique to fix a single bit error? Can we use it in the case of multiple faulty bits and burst-like errors?

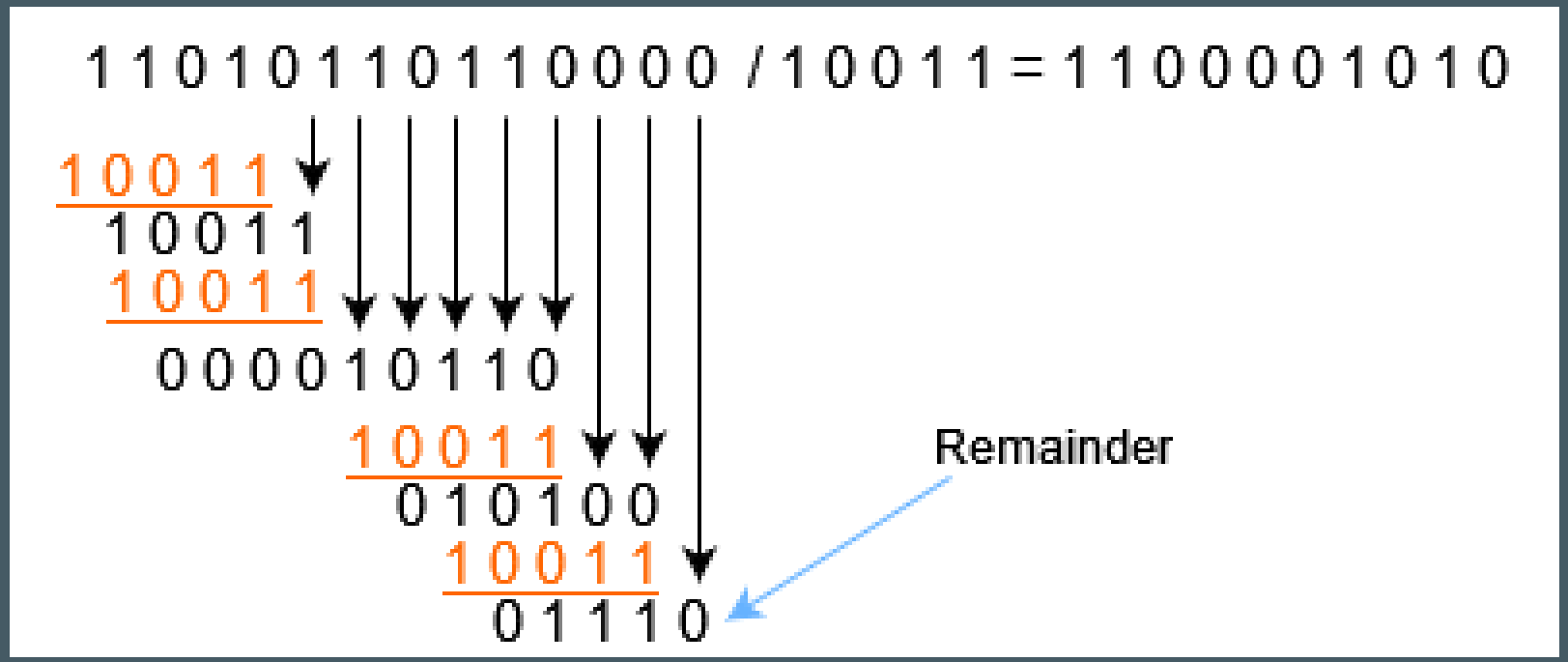
CRC error detection

Source: Based on Tamás Lukovszki's lectures



An example CRC calculation

- Frame ($M(x)$): 1101011011
- Generator ($G(x)$): 10011
- Let's perform the following division: $\frac{1101011011_2}{10011_2}$
- The remainder will be the CRC result.



Exercise V.

- We have a generator polinomial $G(x) = x^4 + x^3 + x + 1$.
- Compute the 4 bit CRC value of the following input: 1100101011101100_2.
- This messages gets altered during transmission. The receiver's data link layer gets the following bit sequence: 11001010110110100100_2.
- Can we detect the errors that have happened with our generator polinomial? If no, why not?

CRC and hashing in python

- CRC

```
import binascii, zlib
test_string= "You must cut down the mightiest tree in the forest with a herring".encode('utf-8')
print(hex(binascii.crc32(bytearray(test_string))))
print(hex(zlib.crc32(test_string)))
```

- MD5

```
import hashlib
test_string= "You must cut down the mightiest tree in the forest with a herring".encode('utf-8')
m = hashlib.md5()
m.update(test_string)
print(m.hexdigest())
```

- SHA1/SHA256

```
import hashlib
test_string= "You must cut down the mightiest tree in the forest with a herring".encode('utf-8')
m = hashlib.sha1()      #or hashlib.sha256
m.update(test_string)
print(m.hexdigest())
```

NIST approved algorithms

Assignment IV.

Netcopy

- Description

Assignment IV.

- Write a netcopy client-server application, which is makes us able to transfer a file and then check it's integrity with CRC or MD5. We need to create three components/scripts:
  - Checksum server: Stores (file ID, checksum length, checksum, expiration (in seconds)) entries.
  - Netcopy client: sends a file (given to it as a command line argument) to the server. Throughout (or at the end of the) transfer it computes the CRC or MD5 checksum of the file, then sends it to the Checksum server. The expiration should be 60 seconds. The file ID is an integer, which should be given as a command line argument as well.
  - Netcopy server: It waits for the client to connect. Once the connection is established it receives the bytes sent to it and stores them in a file given to it as a command line argument. At the end of the transfer it asks the Checksum server for the checksum of the file (based on the file ID) and checks the integrity of the file. It writes the result of the check to the standard output. The file ID should be given as a command line argument here as well.

Checksum server - TCP

- Insert message
  - Format: text
  - Structure: BE|<file ID>|<expiration in seconds>|<checksum length in bytes>|<bytes of the checksum>
  - “|” is the delimiter character
  - Example: BE|1237671|60|12|abcdefabcdef
    - In this case: the file ID: 1237671, the expiration is 60 seconds, the checksum is 12 bytes long and abcdefabcdef is the checksum itself
  - Response: OK
- Query message
  - Format: text
  - Structure: KI|
  - “|” is the delimiter character

Checksum server - TCP

- Query message (cont.)
  - Example: KI|1237671
  - We ask for the checksum that belongs to the file with file ID 1237671
- Response: |
  - Example: 12|abcdefabcdef
- If we don’t have a checksum for the given file ID, we send the following: 0|
- The server runs in an infinte loop and is able to talk to multiple clients at the “same time”. Use TCP for all communication and handle only the listed message types.
- After the expiration the checksum entries should be deleted, but it’s okay to delete them only when they are queried.
- Arguments:

```
python3 checksum_srv.py <ip> <port>
```

- <ip> - e.g. localhost the ip address of the server
- <port> - the port of the server

Netcopy client – TCP

- Operation:
  - Connects to the server. It gets it’s ip and port as command line arguments.
  - It sends the file’s bytes to the server.
  - Communicates with the Checksum server in the already described way.
  - After transfering the file and the checksum it terminates the connection and then itself.
- Arguments:

```
python3 netcopy_cli.py <srv_ip> <srv_port> <chsum_srv_ip> <chsum_srv_port> <file ID> <filename with path>
```

- <file ID>: integer
- <srv_ip> <srv_port>: the address of the netcopy server
- <chsum_srv_ip> <chsum_srv_port>: the address of the Checksum server

Netcopy server – TCP

- Operation:
  - It binds it’s socket to the address given to it as a command line argument.
  - Waits for a client.
  - Once the connection is established it receives the bytes of the file and stores them in the file given to it as a command line argument.
  - Once it received the whole file it asks the Checksum server for the checksum and checks the integrity of the file.
  - It communicates with the Checksum server in the already described way.
  - In case the checksums don’t match it writes the following to the standard output: CSUM CORRUPTED
  - In case the checksums do match it writes the following: CSUM OK
  - Once it has received and checked the file it terminates.

Netcopy server – TCP

- Arguments:

```
python3 netcopy_srv.py <srv_ip> <srv_port> <chsum_srv_ip> <chsum_srv_port> <file ID> <filename with path>
```

- <file ID>: integer, same as in the case of the client – we use it to query the server for the checksum
- <srv_ip> <srv_port>: the address of the netcopy server – used for the binding of the socket
- <chsum_srv_ip> <chsum_srv_port>: the address of the Checksum server
- <filename> : it writes the received bytes to this file

Submission: The program should be submitted through the TMS system in .zip format, which contains a checksum_srv.py, a netcopy_cli.py and a netcopy_srv.py file.
Deadline: See TMS