



---

## ELEC-H-417: BitTorrent-like Project

---

**Author:** Jérémy Pagé

**Professor:** Jean-Michel Dricot

Academic Year 2016–2017

# Contents

<b>1</b>	<b>Goal</b>	<b>2</b>
<b>2</b>	<b>Description of the Steps</b>	<b>3</b>
2.1	Step 1 . . . . .	3
2.2	Step 2 . . . . .	4
2.3	Step 3 (optional) . . . . .	4
<b>3</b>	<b>Execution</b>	<b>5</b>
3.1	Step 1 . . . . .	5
3.2	Step 2 . . . . .	5
3.3	Step 3 . . . . .	5
<b>4</b>	<b>Report Content</b>	<b>5</b>
<b>5</b>	<b>Submission</b>	<b>6</b>
<b>A</b>	<b>Project Structure</b>	<b>7</b>
<b>B</b>	<b>Configuration Files</b>	<b>8</b>
<b>C</b>	<b>Useful Scripts</b>	<b>8</b>
<b>D</b>	<b>Message Format</b>	<b>9</b>
D.1	DISCOVER_TRACKER . . . . .	10
D.2	TRACKER_INFO . . . . .	10
D.3	GET_FILE_INFO . . . . .	10
D.4	FILE_INFO . . . . .	11
D.5	GET_CHUNK . . . . .	11
D.6	CHUNK . . . . .	12
D.7	ERROR . . . . .	12

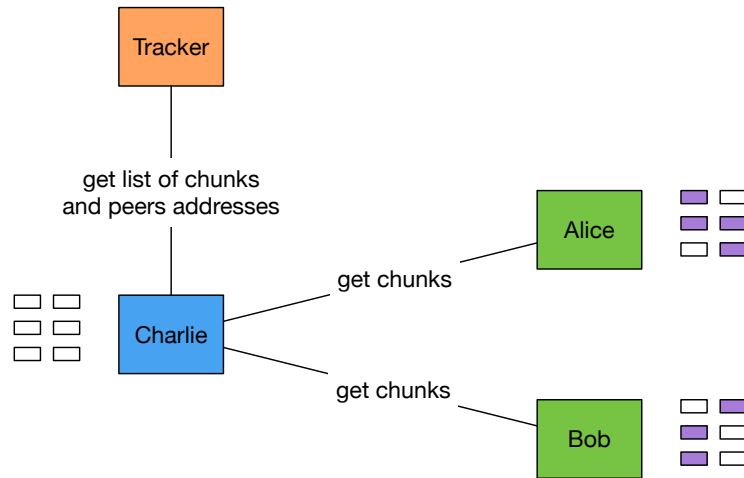


Figure 1: Schema Example

## 1 Goal

By groups of 3, you will create a *BitTorrent*<sup>1</sup>-like client and tracker. The project will have to be finished for December 21 at 12:21 PM. The grade obtained for the project will count for 25% of the final grade for the course. In addition to the source code, you will make a report explaining your solution. The choice for the programming language is left to you, but we highly recommend to use Python<sup>2</sup> (version 3).

You will implement 2 peers: **Alice** and **Bob**, the client: **Charlie**, and the **Tracker**. The tracker provides Charlie with the list of **chunks** of a file, and the peers which have some of these chunks, Alice and Bob. Charlie will download each chunk from Alice or Bob in order to recover the file (see Figure 1).

The project is divided in 3 steps (last one is optional):

1. Creation of the client *Charlie* that will download chunks of a file from 2 peers: *Alice* and *Bob*.
2. Creation of the tracker that will provide your client *Charlie* with the list of chunks to download and where these chunks are located.
3. Discover the tracker with a *broadcast* message.

In order to implement your solution, we will provide you with:

- A basic project structure (see Appendix A).
- Configuration files (see Appendix B).

<sup>1</sup><https://en.wikipedia.org/wiki/BitTorrent>

<sup>2</sup><https://www.python.org/>

- Useful scripts (see Appendix C).
- The definition of the protocol messages format (see Appendix D).

Besides those requirements, you will have the possibility to earn *bonus* points by completing 2 challenges:

1. You will earn **1** extra point (over 20) by implementing the third step.
2. You will earn **2** extra points (over 20) by contacting our own tracker running in the lab and successfully download the file we will provide in chunks.

## 2 Description of the Steps

Be sure to complete one step at a time and to be sure that the step is correctly implemented before going any further.

### 2.1 Step 1

You need to implement 3 entities: Alice, Bob and Charlie. Alice and Bob will have the same job: they will provide Charlie with the chunks they have. Charlie will ask Alice and Bob for missing chunks.

**Alice and Bob** Alice and Bob need to read the configuration file `config/peers.ini` to know their IP address and port number where they can be contacted. They do not need to read `config/file.ini`. Indeed, they can know which chunks they have by looking at their own directory: `chunks/alice/*.bin` and `chunks/bob/*.bin`.

Alice and Bob will listen for incoming requests (on their IP address and port number):

- If the request is malformed (invalid message format), they will send back an **ERROR** (see Appendix D.7) message with the error code `INVALID_MESSAGE_FORMAT`.
- If the request is not of type `GET_CHUNK` (see Appendix D.5), they will send back an **ERROR** (see Appendix D.7) message with the error code `INVALID_REQUEST`.
- If the chunk cannot be found (look to the directory content), they will send back an **ERROR** (see Appendix D.7) message with the error code `CHUNK_NOT_FOUND`.
- Otherwise, they will send back a **CHUNK** (see Appendix D.6) message with the content obtained from the file.

**Note:** If in `config/file.ini`, Alice is set to have a specific chunk, it does not mean that she really has this chunk! Indeed, if you remove the chunk from Alice's directory, she must signal that she does not have the chunk. If Bob has the chunk, it **should** work without any problem. Do not remove a chunk if the other peer does not have it, otherwise it will not be possible to recover the file.

**Charlie** Charlie needs to read the 2 configuration files to know which chunks to download and which peers have it. It will connect to the peer and ask for a chunk by sending a `GET_CHUNK` (see Appendix D.5) message with the chunk hash. If the peer has the chunk, it will send it back, otherwise, Charlie needs to ask to another peer.

**Note:** Try to minimize the opening and closing of *sockets*.

## 2.2 Step 2

You need to implement the tracker. Its job is to provide to Charlie the list of chunks and the peers for each chunk. Charlie will not use the configuration files anymore, except to get the IP address and port number of the tracker (`config/peers.ini`).

**Tracker** The tracker needs to read the 2 configuration files to know its IP address and port number, and to know the list of chunks and peers for each chunk.

The tracker will listen for incoming requests:

- If the request is malformed (invalid message format), it will send back an `ERROR` (see Appendix D.7) message with the error code `INVALID_MESSAGE_FORMAT`.
- If the request is not of type `GET_FILE_INFO` (see Appendix D.3), it will send back an `ERROR` (see Appendix D.7) message with the error code `INVALID_REQUEST`.
- Otherwise, it will send back a `FILE_INFO` (see Appendix D.4) message with the list of chunks and peers for each chunk.

**Charlie** Charlie will connect to the tracker and send a `FILE_INFO` (see Appendix D.4) message. The tracker will respond with the description of the chunks, and Charlie can know contact Alice and Bob to get the chunks (same as the previous step).

## 2.3 Step 3 (optional)

You need to add the possibility to discover a tracker. The tracker will respond to discover requests and Charlie will broadcast these requests to discover the IP address and port number of the tracker. Charlie will not use the configuration files, no exception.

**Note:** There can be several available trackers with different IP addresses. You will need to choose one if this is the case.

**Tracker** The tracker will listen for broadcasting messages (UDP) on port **9000**:

- If the request is malformed (invalid message format), it will send back an `ERROR` (see Appendix D.7) message with the error code `INVALID_MESSAGE_FORMAT`.
- If the request is not of type `DISCOVER_TRACKER` (see Appendix D.1), it will send back an `ERROR` (see Appendix D.7) message with the error code `INVALID_REQUEST`.
- Otherwise, it will send back a `TRACKER_INFO` (see Appendix D.2) message with its IP address and port number (from the configuration file).

**Charlie** Charlie will broadcast a `DISCOVER_TRACKER` (see Appendix D.1) message on port **9000** and listen for responses. Amongst the responses, Charlie will choose one tracker and request for file information (same as the previous step).

### 3 Execution

If you use Python as programming language, you will execute your application with the following commands.

#### 3.1 Step 1

Run Alice and Bob in 2 different terminals:

```
python3 src/alice.py
python3 src/bob.py
```

Then, run Charlie:

```
python3 src/charlie.py 1
```

where `1` is the step number.

**Note:** The parsing of the step number argument is already done for you in the file `src/charlie.py`.

#### 3.2 Step 2

Run Alice, Bob and the tracker in 3 different terminals:

```
python3 src/alice.py
python3 src/bob.py
python3 src/tracker.py
```

Then, run Charlie:

```
python3 src/charlie.py 2
```

#### 3.3 Step 3

Run Alice, Bob and the tracker in 3 different terminals:

```
python3 src/alice.py
python3 src/bob.py
python3 src/tracker.py
```

Then, run Charlie:

```
python3 src/charlie.py 3
```

### 4 Report Content

Your report will contain:

- Functional description of your solution.
- 1 sequence diagram for each step.

- Difficulties encountered and how you solved them.
- Configuration of every host, *i.e.*, for every *socket*, include:
  - IP address and port number (if defined by you)
  - UDP or TCP
  - Mode: *client* or *server*

## 5 Submission

You will create a ZIP<sup>3</sup> containing your source code and your report. Make sure to include everything and that your code is working!

You will send the created ZIP to the email address: `jerepage@ulb.ac.be` with the following subject:

[ELEC-H-417] FIRSTNAME1 LASTNAME1 - FIRSTNAME2 LASTNAME2 - FIRSTNAME3 LASTNAME3

---

<sup>3</sup>[https://en.wikipedia.org/wiki/Zip\\_\(file\\_format\)](https://en.wikipedia.org/wiki/Zip_(file_format))

## A Project Structure

You will receive<sup>4</sup> this basic structure:

```
elec-h-417-project
├── chunks
│   ├── alice
│   ├── bob
│   └── charlie
├── config
│   └── peers.ini
├── files
├── src
│   ├── lib
│   ├── alice.py
│   ├── bob.py
│   ├── charlie.py
│   ├── check_file.py
│   ├── create_chunks.py
│   ├── merge_chunks.py
│   └── tracker.py
├── MESSAGES.md
├── README.md
└── statement.pdf
```

You will only modify those files:

- `config/peers.ini` : configure the IP addresses and port numbers of the entities.
- `src/alice.py` , `src/bob.py` : provide the chunks of the file.
- `src/charlie.py` : download the chunks of the file.
- `src/tracker.py` : provide the list of chunks and the peers for each chunk.

If you need additional files (highly recommended to split your code!), you will add them to the `src/lib/` directory and import them.

### Note:

- `MESSAGES.md` <sup>a</sup> describes the protocol messages format as in Section D.
- `README.md` <sup>b</sup> explains how to use the scripts and how to run the different steps.
- `statement.pdf` is this document.

<sup>a</sup><https://github.com/jpagex/elec-h-417-project/blob/master/MESSAGES.md>

<sup>b</sup><https://github.com/jpagex/elec-h-417-project/blob/master/README.md>

<sup>4</sup>Available online at <https://github.com/jpagex/elec-h-417-project>.



## B Configuration Files

There is 2 different configuration files:

- `config/peers.ini` : Define the IP addresses and port numbers of Alice, Bob and the tracker. You can edit this file to change these values.
- `config/file.ini` : This file list the chunks in the order with their hash and the peers (Alice or Bob) that have the chunk. You do not need to edit this file, it is auto-generated with the script `src/create_chunks.py`.

The default `config/peers.ini` file is:

```
[tracker]
ip_address = 127.0.0.1
port_number = 8000

[alice]
ip_address = 127.0.0.1
port_number = 8001

[bob]
ip_address = 127.0.0.1
port_number = 8002
```

Here is an example of a `config/file.ini` file:

```
[description]
filename = test.txt
chunks_count = 3

[chunks]
0 = 7b1c9c5713f25e3ff8f1d84b6dfa01045f67fcc7
1 = 9473c0216c709b1d24e8089626900a0343cdee36
2 = bd0be5d2b74e26ec4430c23c806dfd646e610cbb

[chunks_peers]
0 = alice
1 = alice, bob
2 = bob
```

## C Useful Scripts

3 Python scripts have been made for you:

- `src/create_chunks.py` : Create chunks of 512 kB from a file inside the `files/` directory, and create the configuration file `config/file.ini`. The first third of chunks will be added to Alice, the second third to Alice **and** Bob, and the last third to Bob.
- `src/merge_chunks.py` : Merge the chunks from `chunks/charlie/` directory into a single file.

- `src/check_file.py` : Check if the file obtained from the chunks is identical to the one provided in `files/`.

To use them, run these commands:

```
python3 src/create_chunks.py [filename]
python3 src/merge_chunks.py
python3 src/check_file.py
```

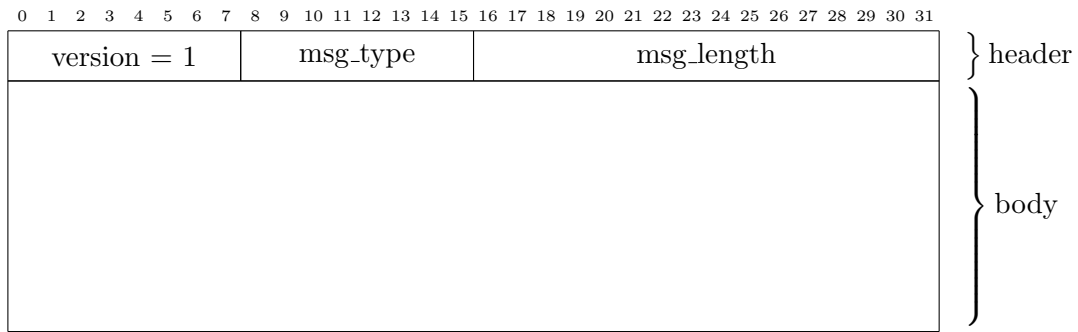
where `filename` is the filename of the file in the `files/` directory.

**Note:** to create a file of a specific size with random data, use this command:

```
head -c [bytes_count] < /dev/urandom > files/[filename]
```

## D Message Format

The general format of a packet is comprised of a header of 4-bytes length and a body of variable length.



- `version` must be 1.
- `msg_type` must be between 0 and 6 (included) depending on the body type.
- `msg_length` is the total length of the message in *dwords*<sup>5</sup> (e.g., if the message has a body of 12 bytes, the `msg_length` will be  $(4 + 12)/4 = 4$ ). It means that the message length in bytes must be a multiple of 4.

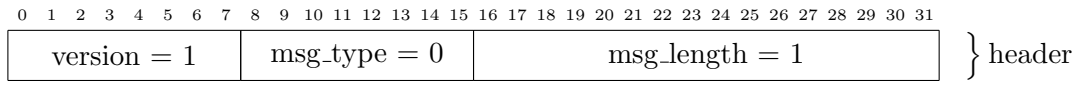
The possible values for `msg_type` are:

- 0 = `DISCOVER_TRACKER`
- 1 = `TRACKER_INFO`
- 2 = `GET_FILE_INFO`
- 3 = `FILE_INFO`
- 4 = `GET_CHUNK`
- 5 = `CHUNK`
- 6 = `ERROR`

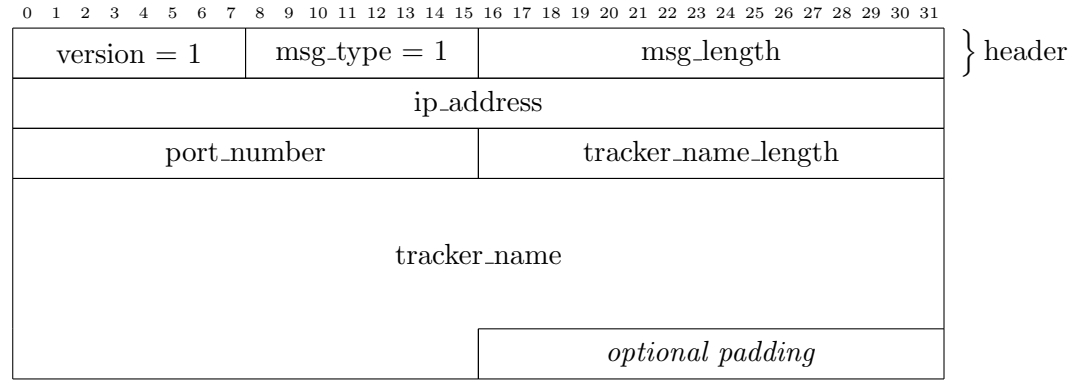
---

<sup>5</sup>1 *dword* = 4 bytes

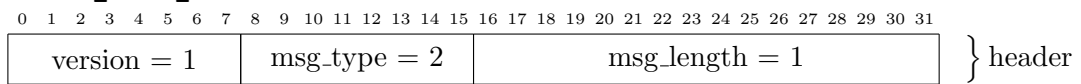
### D.1 DISCOVER\_TRACKER



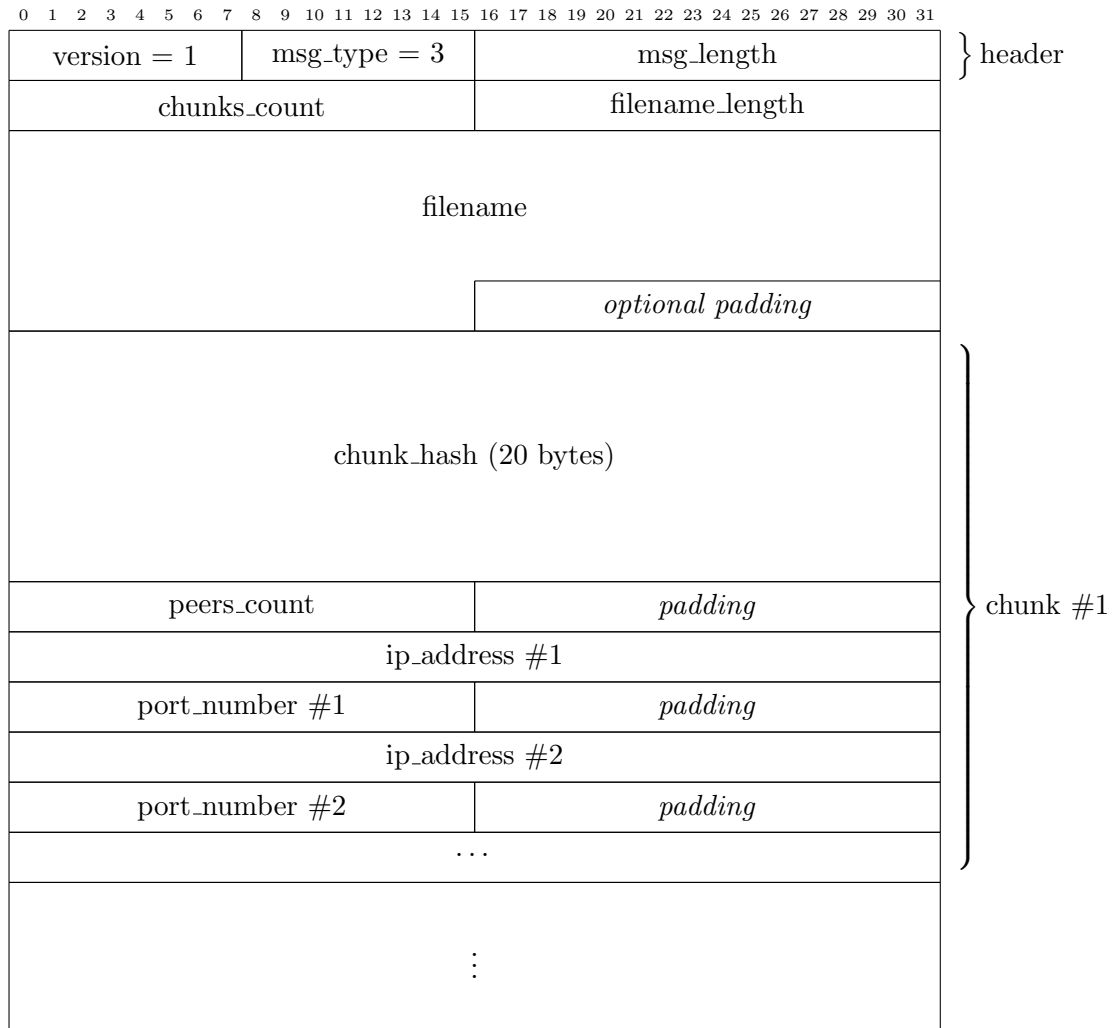
### D.2 TRACKER\_INFO



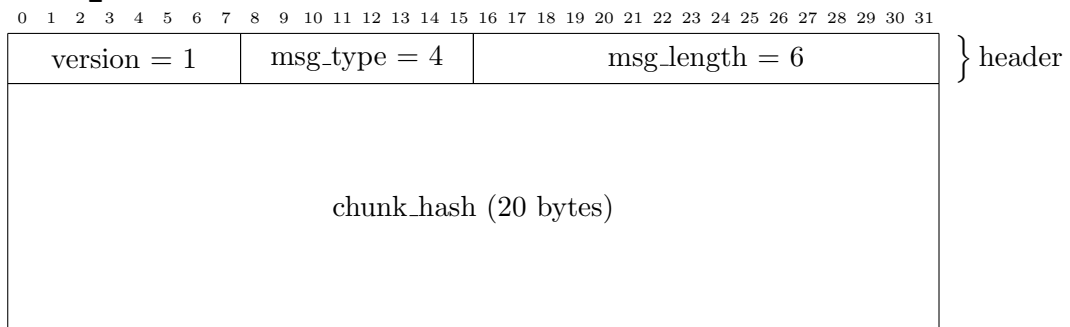
### D.3 GET\_FILE\_INFO



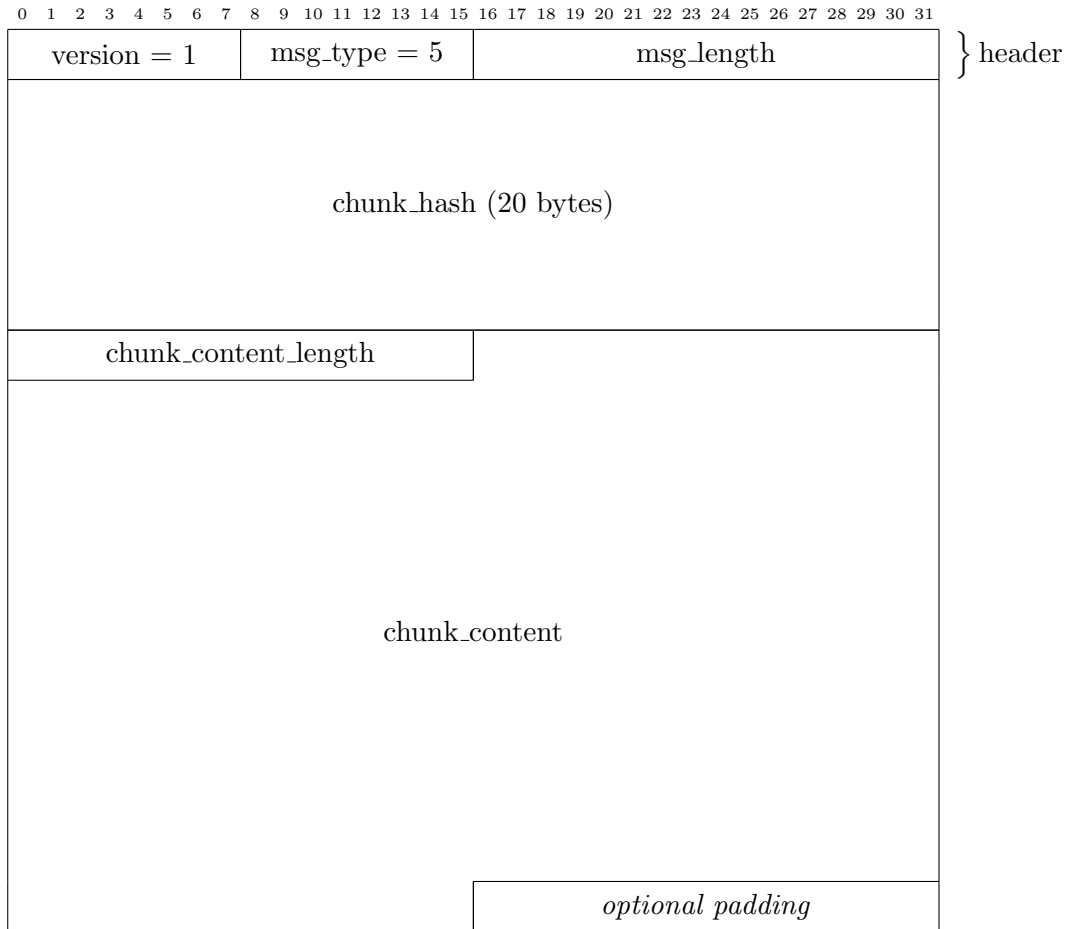
#### D.4 FILE\_INFO



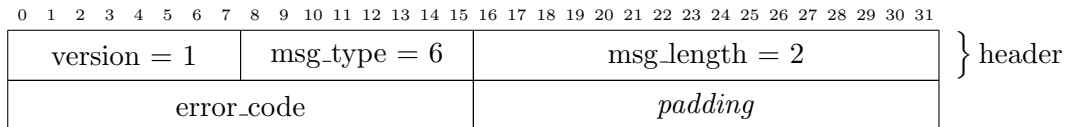
#### D.5 GET\_CHUNK



## D.6 CHUNK



## D.7 ERROR



The error codes are:

- 0 = INVALID\_MESSAGE\_FORMAT
- 1 = INVALID\_REQUEST
- 2 = CHUNK\_NOT\_FOUND