# Network simulator

The network consists of three kinds of node-objects (client, servers, routers) and are connected by links, which have latency and bandwidth. Nodes transmit packet-objects. A simplified DNS.

## Scope

All data is sent in one packet, no congestion control, no TCP-style reliability, no TCP/IP-style encapsulation.

## Features

- load a network from text files
- establish lowest cost routes between nodes
- send simple packets between clients and servers
- print useful information eg. current packets in a node, helpful error messages etc.
- Qt GUI and network traffic animations
- maybe something more (adding and removing a node)

## Modules

- main, contains DNS and main map functionalities
- nodes
- packet

## Division of work

- network functionalities (routing tables, packet sending logic etc.) (Rasmus)
- full-time UI guy (Santeri)
- parsing text files, parsing commands, Packet-class, tests, cmake (TBD)

## External libraries

- QT

# Preliminary schedule and milestones

**20.11.** Basic class structures done, example files and file parsing implemented. Possibly some UI features implemented. Code and different features can be tested by running the code.

**27.11.** All main features ready

**4.12.** Project done before final adjustments

**9.12. FINAL SUBMISSION DL**

# Planned commands

- **G** <client name> <server name> <dummy content>; see below (**G**et)
- **R** <router id>; prints router's routing table (**R**outing table)
- **P** <node id>; prints queued packets' information (**P**ackets)
- **S** <node id>; sends a packet forward from the queue (**S**end)

- **nodes** <file name>; loads nodes, updates DNS and main map
- **links** <file name>; loads links, updates nodes
- **route**; calculates routing tables for routers

- **DNS**; prints DNS map

Maybe something more…

The network will be loaded from two text files that look something like this:

**nodes.txt**
<type>;<id>;<max_capacity>;<end_node_id>;<name>;<content_type>;
<content_size>;<connection_id>

Type 0 client, 1, router, 2 server. Routers don't have end node ids
Parsing this will create the objects and update DNS <name_of_end_node, its_ and the map of id, ptr_node pairs. There will also be a global map of <node_id, pointer_to_node_obj> pairs, **"main map"** created here.

Parsing: if type == 0, update DNS in addition to updating the main map

If type == 1, set connection_id_ to 0 if not connected to a end node
If type == 2, parse content size and type also.

**links.txt**
<node_id>;<transmission_speed>;<propagation_ delay>;<neighbor_id>;

Parsing: get the correct objects from the main map and update their neighbors_ map (the routing tables will be created based on these).

Optional thought: there could also be a button to create a node

After everything has been loaded, the routers will establish their routing tables. Now you can type commands.
All end-nodes are connected to the network by one router, all of this end-node's packets are sent to it and the router will handle the rest.

## Use scenario

Routing tables etc. exist. Three end-nodes, client myComputer, servers Netflix and Youtube, trivial routers.

DNS map:
myComputer - 3
Netflix - 2
Youtube - 1

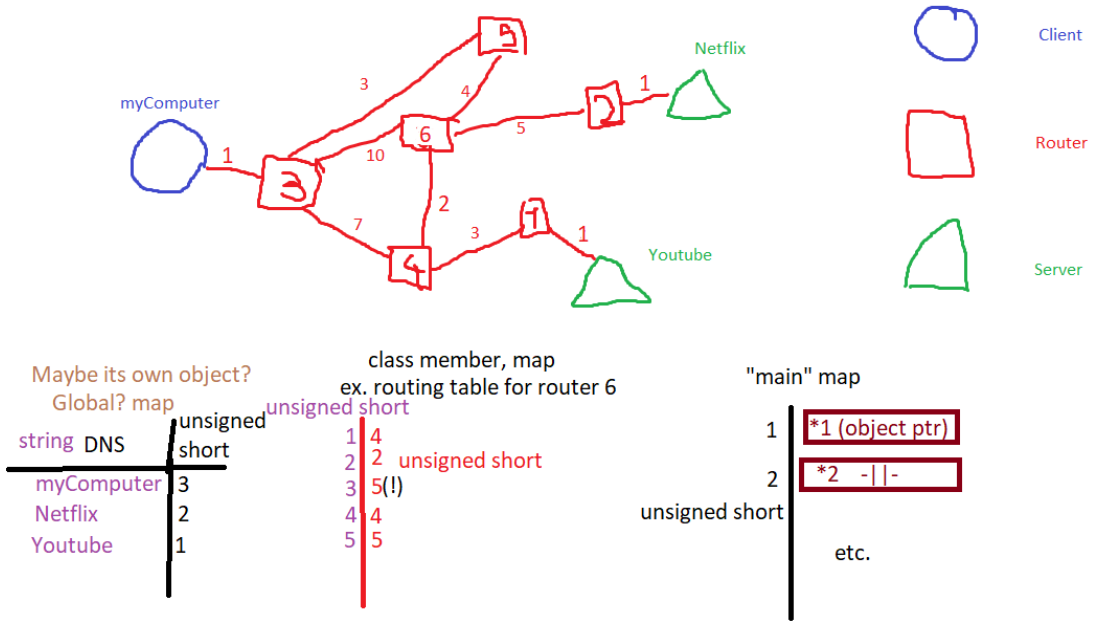
Command line: **G myComputer Youtube Gangnam style;** "GET" packet
This command will create a packet object with the following values:
Content (string): Gangnam style
Size: something quite small
Flags: 0 (in bits 0000 0000) (this might not be needed)
Sender_id: 3 (from myComputer connection_id_)
Receiver_id: 1 (from DNS)

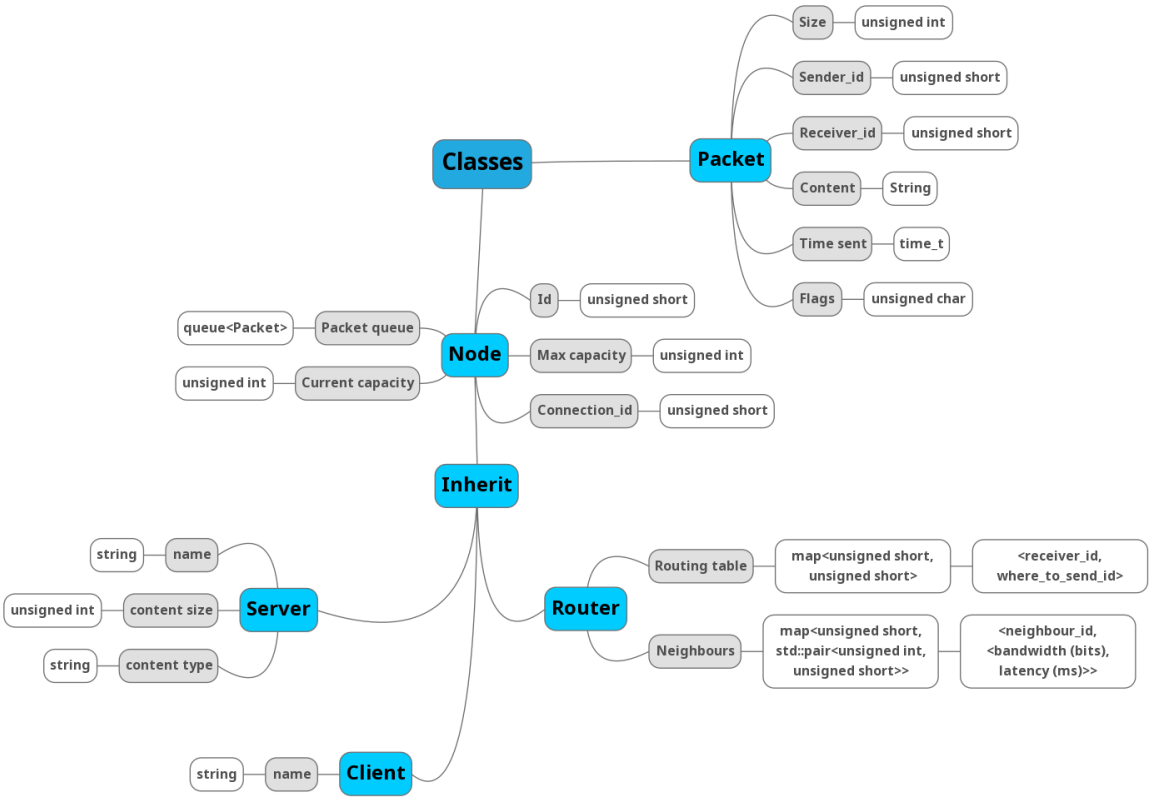
How is receiver_id acquired? The arguments are parsed, and the third one is used as a key in the DNS map, which now returns 1.

The packet is created and transmitted to router 3, ie. is copied to its packets_ queue, and removed from myComputer's queue (the router is accessed by using myComputer's connection_id_ as key to the main map, which gives a pointer to

router 3). Router 3 then checks where packets addressed to id 1 are sent, using receiver_id_ as key to its member routing_table, which gives value 4. Again, this is used as a key to the main map, which gives a pointer to router 4.

As the packet arrives in router 1, it checks that the packet is addressed here, and is given to the Youtube server. Here, the packet addresses are switched, the content_ string is appended with ".mp4" (as defined in content_type_ member), and size is switched to something that's defined in content_size_ of Youtube server object.






**The global maps**

std::map<unsigned short, Node*> main_map;
std::map<std::string, unsigned short> DNS;

```
class Packet {

    private:
    unsigned int size_; // bytes
    unsigned short sender_id_;
    unsigned short receiver_id_;
    std::string content_;
    unsigned char flags_; // reverse order
    time_t time_sent_; //

    public:

    // getters, setters etc.

};
```

```
class Node {
    private:
    unsigned short id_;
    std::queue<Packet> packets_;
    unsigned int max_capacity_;
    unsigned int current_capacity_;
    unsigned short connection_id_; // for routers end node id, end nodes router_id

    public:

    // getters, setters etc.

};




class Router : public Node {

    private:
    std::map<unsigned short, unsigned short> routing_table_; // <receiver_id,
where_to_send_id>
    std::map<unsigned short, std::pair<unsigned int, unsigned short>> neighbors_; //
<neighbor_id, <bandwidth, latency>>bits and ms


    public:

    // getters, setters etc.

};

class Client : public Node {

    private:
    std::string name_;

    public:

    // getters, setters etc.
};

class Server : public Node {
```

```
    private:
    std::string name_;
    std::string content_type_; // .mp4 for Youtube, .jpg for Instagram etc.
    unsigned int content_size_; // small for pictures, large for videos

    public:

    // getters, setters etc.

};
```