Routing

Our network simulator implements some simplified versions of real-life functionalities, such as DNS and routing.

Nice to know:

- All clients and servers are connected to one router, and all routers have only one end node connected to them
- Because of this, when a router receives a packet that is addressed to it, it forwards the packet to the end node it is connected to.
- Packet receiver and sender ids are router ids (so if client id1 is connected to router id2, client's packet will be sent with sender_id = 2 because of reasons I've forgotten

DNS

When in the beginning the files that define the network are parsed, the parser creates and updates a C++ map of <std::string server_name, ushort router_id>
When creating a packet, server name is given, which is used as the key to DNS returning the router's id, to which server bearing this name is connected to.

Routing tables

As all the nodes and their links are known to us from the beginning, we can use the link state algorithm (Dijkstra's algorithm) to establish shortest paths between routers.

After the nodes and links have been created, the function create_routing_table() is called for every router before the user can start using the program.

The table creation happens in two parts: the "actual" dijkstra, and after that going so far the previous list that the root node is encountered.

Finding the lowest cost in this context means finding the fastest link, so link cost is calculated by this formula: 1000.0 / bandwidth, so that the largest bandwidth becomes the smallest cost so that Dijkstra's algorithm works nicely. Also, propagation delay is omitted here, cost is calculated only based on transmission delay.

As C++ priority_queue does not allow decreasing key, a slow, self-made "pseudo_heap" (cost_to_node, node_id pair) is used that allows decreasing key. Dijkstra is run normally, with the addition of keeping a previous-list. Now we have shortest paths.

Suppose we have previous list like this (root A, destination->where to send):

- A->A, B->B, C->B, D->C (assume we have a link to B from A)

Now we have to "backtrack" the list, until all where to send all have nodes that are A's neighbors. So D's previous is C and C's is B, that is A's neighbor, now we update

routing_table_ with values (D, B). This is done after the *trace back previous* comment

Routing

When a router receives a packet, its receiver_id is checked. If the destination id is the same as the router’s, the function send_to_end_node() is called. If not, the destination id is used as the key to the router's routing_table_ map, which returns the correct id where to send next. Then this id is used as the key to main_map, which gives the correct object where to send.