Network simulator

Generated by Doxygen 1.9.5

Chapter 1

readme

#TODOS The waiting until calling receive in send() and in packet.h needs to either be done in a separate thread or with a signal through QTimer or something so that the gui and animations don't get blocked.

To create a handler that calls send() when and where appropriate.

1.1 Requirements with Qt Creator

There shouldn't be any other requirements if you use Qt Creator.

On windows go to `https://www.qt.io/download` and find an installer if you just want a binary. You probably want the open source version. Alternatively, you can compile from source. Instructions here: `https://doc.qt.io/qt-6/build-sources.html`

On macOS you could use homebrew to install Qt Creator. Run `brew install --cask qt-creator`

On linux just use your package manager of choice.

1.2 Requirements without Qt Creator

1.2.1 A compiler

On windows you should probably just install MinGW. Using other compilers through WSL 2 is probably possible as well.

On macOS, you could use clang. Install Xcode and its command line tools.

1.2.2 Qt

This application was built with Qt version 6.4.0

On windows go to `https://www.qt.io/download` and find an installer if you just want a binary. You probably want the open source version. Alternatively, you can compile from source. Instructions here: `https://doc.qt.io/qt-6/build-sources.html`

On macOS, you could use homebrew to install qt. Run `brew install qt`

On linux just use your package manager of choice.

1.2.3 cmake

On windows, follow instructions from `https://cmake.org/install/`

On macOS, you could use homebrew to install cmake. Run `brew install cmake`

On linux just use your package manager of choice.

1.2.4 make

1.3 Building the gui without Qt Creator

Run cmake on this directory and make. If you don't mind where build files and the application binary are placed run \ `cmake .` \ `make`

If you want build files and the application binary somewhere else, navigate to your desired directory and run \ `cmake $PATH_TO_GUI`, where `$PATH_TO_GUI` is the path to this directory. \ `make`

Now you should have a runnable binary

1.4 Building with Qt Creator

Open the directory in Qt Creator, build and run.

1.5 Source content

1.5.1 MainWindow

MainWindow contains the source code for the main window of the program. mainwindow.ui holds the layout of the window.

1.5.2 NodeItem

Is a visual representation of nodes and their connections.

1.5.3 PacketItem

Describes the animation item that moves when packets are sent between nodes.

1.5.4 Resources.qrc

Is a container that holds and links resources external to the source code.

1.5.5 main.cpp

Entry point of the program.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

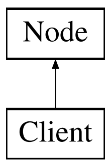
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 Client Class Reference

Inheritance diagram for Client:



Public Member Functions

- **Client** (unsigned short id, unsigned int max_capacity, unsigned short connection_id, std::pair< unsigned int, unsigned short > connection_cost, std::string name, NodeItem ∗nodeitem)
- void **create_packet** (unsigned short sender_id, unsigned short receiver_id, std::string content, unsigned int flag)
- void send ()
- void send (Packet ∗packet)
- void receive (Packet ∗packet)
- void print ()
- std::string getName () const

Additional Inherited Members

5.1.1 Member Function Documentation

5.1.1.1 getName()

```
std::string Client::getName ( ) const  [inline], [virtual]
```

Implements Node.

5.1.1.2 print()

```
void Client::print ( )  [virtual]
```

Implements Node.

5.1.1.3 receive()

```
void Client::receive (
            Packet * packet )  [virtual]
```

Implements Node.

5.1.1.4 send() [1/2]

```
void Client::send ( )  [virtual]
```

Implements Node.

5.1.1.5 send() [2/2]

```
void Client::send (
            Packet * packet )  [virtual]
```

Implements Node.

The documentation for this class was generated from the following files:

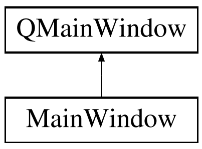
- nodes.h
- client.cpp

5.2 MainWindow Class Reference

The MainWindow class The main window of the program.

```
#include <mainwindow.h>
```

Inheritance diagram for MainWindow:



Public Member Functions

- MainWindow (std::map< unsigned short, Node ∗ > ∗map1, std::map< std::string, unsigned short > ∗map2, QWidget ∗parent=nullptr)

  MainWindow Consructor for the MainWindow class.
- void addServer (unsigned short id, std::string name, float x, float y)

  addServer Creates a graphical server nodeitem and adds it to the scene.
- void addRouter (unsigned short id, std::string name, float x, float y)

  addRouter Creates a graphical router noditem and adds it to the scene.
- void addClient (unsigned short id, std::string name, float x, float y)

  addClient Creates a graphical client nodeitem and adds it to the scene.
- void addLine (NodeItem ∗node1, NodeItem ∗node2)

  addLine Adds a graphical line between two nodeitems.
- void sendPacket (NodeItem ∗node1, NodeItem ∗node2)

  sendPacket Animates a moving packet between two nodeitems
- void dropPacket (NodeItem ∗node)

  dropPacket Animates a dropped packet by changing the color of the node to black.
- NodeItem ∗ getNode (unsigned short id)

  getNode Getter for the graphical nodeitem.
- NodeItem ∗ getNodeByName (std::string name)

  getNodeByName Getter for the graphical nodeitem.

5.2.1 Detailed Description

The MainWindow class The main window of the program.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 MainWindow()

```
MainWindow::MainWindow (
            std::map< unsigned short, Node ∗ > ∗ map1,
            std::map< std::string, unsigned short > ∗ map2,
            QWidget ∗ parent = nullptr )
```

MainWindow Consructor for the MainWindow class.

Parameters

map1	A map with id keys and Node values
map2	A DNS map
parent	An optional pointer to a parent QObject

5.2.3 Member Function Documentation

5.2.3.1 addClient()

```
void MainWindow::addClient (
             unsigned short id,
             std::string name,
             float x,
             float y )
```

addClient Creates a graphical client nodeitem and adds it to the scene.

Parameters

id	The id of the client node the created nodeitem corresponds to.
name	The name of the client.
x	The x coordinate of the created nodeitem.
y	The y coordinate of the created nodeitem.

5.2.3.2 addLine()

```
void MainWindow::addLine (
             NodeItem * node1,
             NodeItem * node2 )
```

addLine Adds a graphical line between two nodeitems.

Parameters

node1	The first nodeitem to be connected.
node2	The second nodeitem to be connected.

5.2.3.3 addRouter()

```
void MainWindow::addRouter (
             unsigned short id,
             std::string name,
             float x,
             float y )
```

addRouter Creates a graphical router noditem and adds it to the scene.

Parameters

id	The id of the router node the created nodeitem corresponds to.
name	The name of the router
x	The x coordinate of the created nodeitem.
y	The y coordinate of the created nodeitem.

5.2.3.4 addServer()

```
void MainWindow::addServer (
             unsigned short id,
             std::string name,
             float x,
             float y )
```

addServer Creates a graphical server nodeitem and adds it to the scene.

Parameters

id	The id of the server node the created nodeitem corresponds to.
name	The name of the server
x	The x coordinate of the created nodeitem.
y	The y coordinate of the created nodeitem.

5.2.3.5 dropPacket()

```
void MainWindow::dropPacket (
             NodeItem * node )
```

dropPacket Animates a dropped packet by changing the color of the node to black.

Parameters

node	The node which drops a packet.

5.2.3.6 getNode()

```
NodeItem * MainWindow::getNode (
             unsigned short id )
```

getNode Getter for the graphical nodeitem.

Parameters

id	The id of the nodeitem.

Returns

A pointer to the nodeitem.

5.2.3.7 getNodeByName()

```
NodeItem * MainWindow::getNodeByName (
             std::string name )
```

getNodeByName Getter for the graphical nodeitem.

Parameters

name	The name of the node.

Returns

A pointer to the nodeitem.

5.2.3.8 sendPacket()

```
void MainWindow::sendPacket (
             NodeItem * node1,
             NodeItem * node2 )
```

sendPacket Animates a moving packet between two nodeitems

Parameters

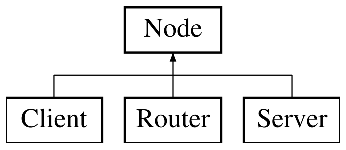
node1	The node the packet leaves from.
node2	The node the packet approaches.

The documentation for this class was generated from the following files:

- mainwindow.h
- mainwindow.cpp

5.3 Node Class Reference

Inheritance diagram for Node:



Public Member Functions

- **Node** (unsigned short id, unsigned int max_capacity, unsigned short connection_id, std::pair< unsigned int, unsigned short > connection_cost, NodeItem ∗nodeitem)
- virtual void **send** ()=0
- virtual void **send** (Packet ∗packet)=0
- virtual void **receive** (Packet ∗packet)=0
- virtual void **print** ()=0
- bool **not_full_after_add** (unsigned int added_size)
- void **change_max_capacity** (unsigned int new_capacity)
- void **change_current_capacity** (unsigned int change)
- unsigned short **getId** () const
- unsigned int **getCurrentCapacity** () const
- unsigned int **getMaxCapacity** () const
- size_t **queue_packet_count** () const
- unsigned short **getConnectionId** () const
- virtual std::string **getName** () const =0
- NodeItem ∗ **getNodeItem** ()

Protected Attributes

- unsigned short **id_**
- std::queue< Packet ∗ > **packets_**
- unsigned int **max_capacity_**
- unsigned int **current_capacity_**
- unsigned short **connection_id_**
- std::pair< unsigned int, unsigned short > **connection_cost_**
- NodeItem ∗ **nodeitem_**

The documentation for this class was generated from the following file:

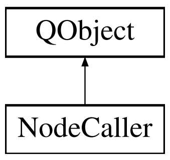
- nodes.h

5.4 NodeCaller Class Reference

The NodeCaller class An object for interacting between nodeitems and mainwindow.

`#include <nodeitem.h>`

Inheritance diagram for NodeCaller:



Public Slots

- void change_max_capacity (unsigned short id)

  change_max_capacity Signals mainwindow to change nodeitems max capacity.
- void call_print (unsigned short id)

  call_print Signals mainwindow to print out nodeitem's information.
- void call_print_routing_table (unsigned short id)

  call_print_routing_table Signals mainwindow to call print_routing_table()
- void send_packet (unsigned short id)

  send_packet Signals the mainwindow to call the send() method of class Node.

Signals

- void **print_called** ()

  print_called A request to call the print() method of the Node class was made.
- void **change_max_capacity_called** ()

  change_max_capacity_called A request to call the change_max_capacity() method of the Node class was made.
- void **print_routing_table_called** ()

  print_routing_table_called A request was made to call the print_routing_table() method of the Router class.
- void **send_packet_called** ()

  send_packet_called A request was made to call the send() method of the Node class.

Public Member Functions

- unsigned short getId ()

  getId A getter for the id of the nodeitem signalling to mainwindow.

5.4.1 Detailed Description

The NodeCaller class An object for interacting between nodeitems and mainwindow.

5.4.2 Member Function Documentation

5.4.2.1 call_print

```
void NodeCaller::call_print (
            unsigned short id )  [inline], [slot]
```

call_print Signals mainwindow to print out nodeitem's information.

Parameters

id	The id of the calling nodeitem.

5.4.2.2 call_print_routing_table

```
void NodeCaller::call_print_routing_table (
             unsigned short id )  [inline], [slot]
```

call_print_routing_table Signals mainwindow to call print_routing_table()

Parameters

id	The id of the calling router.

5.4.2.3 change_max_capacity

```
void NodeCaller::change_max_capacity (
             unsigned short id )  [inline], [slot]
```

change_max_capacity Signals mainwindow to change nodeitems max capacity.

Parameters

id	The id of the calling nodeitem.

5.4.2.4 getId()

```
unsigned short NodeCaller::getId ( )  [inline]
```

getId A getter for the id of the nodeitem signalling to mainwindow.

Returns

id of the nodeitem.

5.4.2.5 send_packet

```
void NodeCaller::send_packet (
             unsigned short id )  [inline], [slot]
```

send_packet Signals the mainwindow to call the send() method of class Node.

Parameters

id	The id of the calling node.

The documentation for this class was generated from the following file:

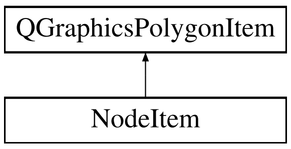
- nodeitem.h

5.5 NodeItem Class Reference

The NodeItem class A grapchical item which represents a node.

```
#include <nodeitem.h>
```

Inheritance diagram for NodeItem:



Public Member Functions

- NodeItem (unsigned short id, std::string name, double x, double y, NodeCaller ∗caller, QGraphicsItem ∗parent=nullptr)

  NodeItem Constructor of nodeitem.
- QVariant itemChange (GraphicsItemChange change, const QVariant &value)

  itemChange Describes how this nodeitem should react to a change in position.
- void addConnection (std::pair< NodeItem ∗, NodeItem ∗ > connection, QGraphicsLineItem ∗line)

  addConnection Adds a new connection to static collections.
- void updateConnections ()

  updateConnections Updates the graphical lines between each connected nodeitem.
- QPointF position ()

  position Returns the centre of this nodeitem.
- void addName (std::string name)

  addName Adds a graphical textitem to this nodeitem.
- void moveTextItem (qreal x, qreal y)

  moveTextItem Moves this nodeitems textitem by (x,y)
- QGraphicsSimpleTextItem ∗ getNameItem ()

  getNameItem A getter for the graphical textitem underneath each nodeitem.
- void **resetTicks** ()

  resetTicks Reset ticks_. Used for calculating how long the animation ought to be.
- void setColor (QBrush brush)

  setColor Changes the color of this NodeItem.
- void **dropPacket** ()

  dropPacket Setter for isDroppingPacket_. Makes this NodeItem change its color to black.
- void changeSpeed (double multiplier)

*changeSpeed Multiplies the animationSpeed_*

- void **setSpeed** ()

  *setSpeed Setter for animationSpeed_*
- void operator<< (NodeItem ∗other)

  operator << Animates a packetitem between this and other nodeitems.
- void contextMenuEvent (QGraphicsSceneContextMenuEvent ∗event)

  contextMenuEvent Opens a contextmenu and emits a signal to MainWindow if an action was chosen.
- std::string **getName** () const
- unsigned short **getID** () const

Protected Member Functions

- void advance (int phase)

  advance Animates the nodeitem. Describes what the nodeitem ought to do in each tick of the timer.

5.5.1 Detailed Description

The NodeItem class A grapchical item which represents a node.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 NodeItem()

```
NodeItem::NodeItem (
            unsigned short id,
            std::string name,
            double x,
            double y,
            NodeCaller * caller,
            QGraphicsItem * parent = nullptr )   [inline]
```

NodeItem Constructor of nodeitem.

Parameters

id	The id of the nodeitem.
name	The name of the nodeitem.
x	The x coordinate of the top left corner of the nodeitem.
y	The y coordinate of the top left corner of the nodeitem.
caller	A QObject for signalling between nodeitems and mainwindow.
parent	An optional pointer to a parent QObject.

5.5.3 Member Function Documentation

5.5.3.1 addConnection()

```
void NodeItem::addConnection (
             std::pair< NodeItem *, NodeItem * > connection,
             QGraphicsLineItem * line )  [inline]
```

addConnection Adds a new connection to static collections.

Parameters

connection	The connection to be added.
line	A pointer to a lineitem.

5.5.3.2 addName()

```
void NodeItem::addName (
             std::string name )  [inline]
```

addName Adds a graphical textitem to this nodeitem.

Parameters

name	The desired name.

5.5.3.3 advance()

```
void NodeItem::advance (
             int phase )  [inline], [protected]
```

advance Animates the nodeitem. Describes what the nodeitem ought to do in each tick of the timer.

Parameters

phase	Describes whether the function was called before or after the change occurred.

5.5.3.4 changeSpeed()

```
void NodeItem::changeSpeed (
             double multiplier )  [inline]
```

changeSpeed Multiplies the animationSpeed_

Parameters

multiplier	

5.5.3.5 contextMenuEvent()

```
void NodeItem::contextMenuEvent (
             QGraphicsSceneContextMenuEvent * event )  [inline]
```

contextMenuEvent Opens a contextmenu and emits a signal to MainWindow if an action was chosen.

Parameters

event	Right click pressed on nodeitem.

5.5.3.6 getNameItem()

```
QGraphicsSimpleTextItem * NodeItem::getNameItem ( )  [inline]
```

getNameItem A getter for the graphical textitem underneath each nodeitem.

Returns

A pointer to the textitem.

5.5.3.7 itemChange()

```
QVariant NodeItem::itemChange (
             GraphicsItemChange change,
             const QVariant & value )  [inline]
```

itemChange Describes how this nodeitem should react to a change in position.

Parameters

change	The type of change which occurred.
value	The value (magnitude) of the change.

Returns

Changed item.

5.5.3.8 moveTextItem()

```
void NodeItem::moveTextItem (
            qreal x,
            qreal y )  [inline]
```

moveTextItem Moves this nodeitems textitem by (x,y)

Parameters

x	The change in x.
y	The change in y.

5.5.3.9 operator<<()

```
void NodeItem::operator<< (
            NodeItem * other )  [inline]
```

operator << Animates a packetitem between this and other nodeitems.

Parameters

other	The other nodeitem.

5.5.3.10 position()

```
QPointF NodeItem::position ( )  [inline]
```

position Returns the centre of this nodeitem.

Returns

Returns the centre coordinate of this nodeitem.

5.5.3.11 setColor()

```
void NodeItem::setColor (
            QBrush brush )  [inline]
```

setColor Changes the color of this NodeItem.

Parameters

brush	The wanted color.

5.5.3.12 updateConnections()

```
void NodeItem::updateConnections ( )  [inline]
```

updateConnections Updates the graphical lines between each connected nodeitem.

The documentation for this class was generated from the following file:

- nodeitem.h

5.6 Packet Class Reference

Public Member Functions

- **Packet** (unsigned int size, unsigned short sender_id, unsigned short receiver_id, std::string content)
- unsigned short **getSenderId** () const
- unsigned short **getReceiverId** () const
- bool **isDebug** () const
- bool **isCheat** () const
- unsigned int **getSize** () const
- time_t **getTimeSent** () const
- void **setSenderId** (unsigned short new_id)
- void **setReceiverId** (unsigned short new_id)
- void **setDebug** ()
- void **setCheat** ()
- std::string & **getContent** ()
- void **setSize** (unsigned int new_size)
- std::string **getContent_print** () const
- void **wait** (unsigned int bandwidth, unsigned short latency)
- void **print** ()

The documentation for this class was generated from the following file:

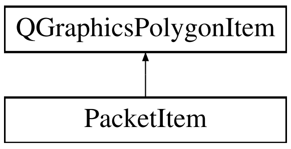
- packet.h

5.7 PacketItem Class Reference

The PacketItem class Is the graphical item which moves in the animation of packages.

```
#include <packetitem.h>
```

Inheritance diagram for PacketItem:



Public Member Functions

- PacketItem (QPointF start, QPointF end, QGraphicsScene ∗scene)

  PacketItem Constructor of the animation item.
- void **resetTicks** ()

  resetTicks Resets the animation counter to 0.
- void setSpeed (qreal magnitude)

  setSpeed Sets the speed of the packetitem. Depends on the distance that needs to be covered.
- void changeSpeed (double multiplier)

  changeSpeed Changes the animation speed of the packetitem.

Protected Member Functions

- void advance (int phase)

  advance Describes how the packetitem should act in each tick of the animation.

5.7.1 Detailed Description

The PacketItem class Is the graphical item which moves in the animation of packages.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 PacketItem()

```
PacketItem::PacketItem (
            QPointF start,
            QPointF end,
            QGraphicsScene ∗ scene )  [inline]
```

PacketItem Constructor of the animation item.

Parameters

start	The start position of the packetitem.
end	The end position of the packetitem.
scene	The scene in which the packetitem appears in.

5.7.3 Member Function Documentation

5.7.3.1 advance()

```
void PacketItem::advance (
            int phase )  [inline], [protected]
```

advance Describes how the packetitem should act in each tick of the animation.

Parameters

phase	Describes whether the change has happened or not.

5.7.3.2 changeSpeed()

```
void PacketItem::changeSpeed (
            double multiplier )  [inline]
```

changeSpeed Changes the animation speed of the packetitem.

Parameters

multiplier	

5.7.3.3 setSpeed()

```
void PacketItem::setSpeed (
            qreal magnitude )  [inline]
```

setSpeed Sets the speed of the packetitem. Depends on the distance that needs to be covered.

Parameters

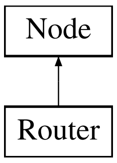
magnitude	The distance moved.

The documentation for this class was generated from the following file:

- packetitem.h

5.8 Router Class Reference

Inheritance diagram for Router:



Public Member Functions

- **Router** (unsigned short id, unsigned int max_capacity, unsigned short connection_id, std::pair< unsigned int, unsigned short > connection_cost, NodeItem ∗nodeitem)
- void **create_routing_table** (const std::map< unsigned short, Router ∗ > &routers)
- std::map< unsigned short, std::pair< unsigned int, unsigned short > > & **getNeighbors** ()
- void **print_routing_table** () const
- void **print_connections** () const
- void **addNeighbor** (unsigned short id, const std::pair< unsigned int, unsigned short > &pair)
- void send ()
- void send (Packet ∗packet)
- void receive (Packet ∗packet)
- void **send_to_end_node** (Packet ∗packet)
- void print ()
- std::string getName () const

Additional Inherited Members

5.8.1 Member Function Documentation

5.8.1.1 getName()

```
std::string Router::getName ( ) const  [inline], [virtual]
```

Implements Node.

5.8.1.2 print()

```
void Router::print ( )  [virtual]
```

Implements Node.

5.8.1.3 receive()

```
void Router::receive (
             Packet * packet )   [virtual]
```

Implements Node.

5.8.1.4 send() [1/2]

```
void Router::send ( )  [virtual]
```

Implements Node.

5.8.1.5 send() [2/2]

```
void Router::send (
             Packet * packet )   [virtual]
```

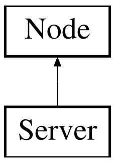
Implements Node.

The documentation for this class was generated from the following files:

- nodes.h
- router.cpp

5.9 Server Class Reference

Inheritance diagram for Server:



Public Member Functions

- **Server** (unsigned short id, unsigned int max_capacity, unsigned short connection_id, std::pair< unsigned int, unsigned short > connection_cost, std::string name, std::string content_type, unsigned int content_size, NodeItem ∗nodeitem)
- void send ()
- void send (Packet ∗packet)
- void receive (Packet ∗packet)
- void print ()
- void **add** (Packet ∗packet)
- std::string **getContentType** () const
- unsigned int **getContentSize** () const
- std::string getName () const

Additional Inherited Members

5.9.1 Member Function Documentation

5.9.1.1 getName()

```
std::string Server::getName ( ) const  [inline], [virtual]
```

Implements Node.

5.9.1.2 print()

```
void Server::print ( )  [virtual]
```

Implements Node.

5.9.1.3 receive()

```
void Server::receive (
            Packet * packet )  [virtual]
```

Implements Node.

5.9.1.4 send() [1/2]

```
void Server::send ( )  [virtual]
```

Implements Node.

5.9.1.5 send() [2/2]

```
void Server::send (
            Packet * packet )  [virtual]
```

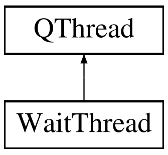
Implements Node.

The documentation for this class was generated from the following files:

- nodes.h
- server.cpp

5.10 WaitThread Class Reference

Inheritance diagram for WaitThread:



Public Member Functions

- **WaitThread** (Node ∗node, Packet ∗packet, unsigned int bandwidth, unsigned short latency, QObject ∗parent=nullptr)

The documentation for this class was generated from the following file:

- nodes.h

Chapter 6

File Documentation

6.1 mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QPainter>
#include <QGraphicsView>
#include <QTimer>


#include <map>

#include "nodeitem.h"
#include "nodes.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE


class MainWindow : public QMainWindow
{
    Q_OBJECT // necessary for slots and signals

public:
    MainWindow(std::map<unsigned short, Node*> *map1, std::map<std::string, unsigned short> *map2,
    QWidget *parent = nullptr);
    ~MainWindow();

    void addServer(unsigned short id, std::string name, float x, float y);

    void addRouter(unsigned short id, std::string name, float x, float y);

    void addClient(unsigned short id, std::string name, float x, float y);

    void addLine(NodeItem *node1, NodeItem *node2);

    void sendPacket(NodeItem *node1, NodeItem *node2);

    void dropPacket(NodeItem *node);

    NodeItem* getNode(unsigned short id);

    NodeItem* getNodeByName(std::string name);


private slots:

    void on_actionLoad_triggered();


    void on_actionClient_triggered();
    void on_actionRouter_triggered();
    void on_actionServer_triggered();
    void on_actionAdd_a_connection_triggered();

    void on_actionCreate_a_packet_triggered();

    void on_actionSend_packets_triggered();
    void sendPackages();

    void send_package();

    void change_node_max_capacity();

    void print_node();

    void print_routing_table();

    void on_actionChange_animation_speed_triggered();

private:
    Ui::MainWindow *ui;
    QGraphicsScene *scene;

    std::map<unsigned short, NodeItem *> nodeitems;

    QTimer *timer;

    NodeCaller *caller_;

    std::map<unsigned short, Client*> client_nodes;
    std::map<unsigned short, Router*> router_nodes;

    std::map<unsigned short, Node*> *main_map; // all nodes in simulation <node_id, Node>
    std::map<std::string, unsigned short> *DNS;


    void addLink(Router* first, unsigned int bandwidth, unsigned short latency, Router* second);

    bool readLinks();

    std::pair<float, float> nextPos();

    bool readNodes();

    void create_routing_tables();

    void create_lineitems();
};



#endif // MAINWINDOW_H
```

6.2 nodeitem.h

```
#ifndef NODEITEM_H
#define NODEITEM_H

#include <QGraphicsSimpleTextItem>
#include <QGraphicsScene>
#include <QMenu>
#include <QGraphicsSceneContextMenuEvent>
#include <QInputDialog>

#include "packetitem.h"

class NodeCaller : public QObject {
    Q_OBJECT // necessary for signals and slots

public:
    unsigned short getId() { return id_; }

public slots:
    void change_max_capacity(unsigned short id)
    {
        id_ = id;
        emit change_max_capacity_called();
    }

    void call_print(unsigned short id)
    {
        id_ = id;
        emit print_called();
    }

    void call_print_routing_table(unsigned short id)
    {
        id_ = id;
        emit print_routing_table_called();
    }

    void send_packet(unsigned short id)
    {
        id_ = id;
        emit send_packet_called();
    }

signals:

    void print_called();

    void change_max_capacity_called();

    void print_routing_table_called();

    void send_packet_called();

private:
    unsigned short id_;
};


class NodeItem : public QGraphicsPolygonItem
{
public:
    NodeItem(unsigned short id, std::string name, double x, double y, NodeCaller *caller, QGraphicsItem
   *parent = nullptr) : QGraphicsPolygonItem(parent),
        caller_(caller), name_(name), id_(id)
    {
        position_ = QPointF(0,0);
        offset_ = QPointF(x+12,y+12);
        setFlag(QGraphicsItem::ItemIsMovable);
        setFlag(QGraphicsItem::ItemSendsScenePositionChanges);
        QBrush brush(Qt::white);
        color = brush;
        isDroppingPacket_ = false;
        ticks_ = 0;
    }

    QVariant itemChange(GraphicsItemChange change, const QVariant &value)
    {
        if (change == ItemPositionChange && scene()) {
            QPointF newPos = value.toPointF();
            position_ = newPos;
            updateConnections();
        }
        return QGraphicsItem::itemChange(change, value);
    }

    void addConnection(std::pair<NodeItem *, NodeItem *> connection, QGraphicsLineItem* line)
    {
        lines_.push_back(line);
        connections_.push_back(connection);
    }


    void updateConnections()
    {
        int i = 0;
        for (auto connection : connections_)
        {
            lines_[i]->setLine(QLineF(connection.first->position(),
                                      connection.second->position()));
            i++;
        }
    }

    QPointF position() { return position_ + offset_; }

    void addName(std::string name)
    {
        nameItem_ = new QGraphicsSimpleTextItem(QString::fromStdString(name), this);
        QPointF center = this->boundingRect().center();
        QPointF newPos(this->mapToScene(center).x() - (nameItem_->boundingRect().width() / 2),
                       this->mapToScene(center).y() - (nameItem_->boundingRect().height() / 2));
        nameItem_->setPos(newPos);
    }

    void moveTextItem(qreal x, qreal y)
    {
        nameItem_->moveBy(x,y);
    }

    QGraphicsSimpleTextItem* getNameItem() { return nameItem_; }

    void resetTicks() { ticks_ = 0; }

    void setColor(QBrush brush)
    {
        color = brush;
    }

    void dropPacket()
    {
        isDroppingPacket_ = true;
    }

    void changeSpeed(double multiplier)
    {
        if (multiplier > 0) animationSpeed_ = multiplier;
    }

    void setSpeed()
    {
        animationSpeed_ = 1;
    }

    void operator«(NodeItem *other)
    {
        new PacketItem(this->position(), other->position(), this->scene());
    }


    void contextMenuEvent(QGraphicsSceneContextMenuEvent *event)
    {
        QMenu *menu = new QMenu();
        menu->setAttribute(Qt::WA_DeleteOnClose);
        menu->clear();
        menu->addAction("Send a packet");
        menu->addAction("Change maximum capacity");
        menu->addAction("Print information");
        if (this->name_ == std::to_string(this->id_))
            menu->addAction("Print routing table");
        QAction *a = menu->exec(event->screenPos());
        if (a == nullptr) return;
        else if (a->text().toStdString() == "Send a packet") {
            caller_->send_packet(id_);
        }
        else if (a->text().toStdString() == "Change maximum capacity") {
            caller_->change_max_capacity(id_);
        }
        else if (a->text().toStdString() == "Print information") {
            caller_->call_print(id_);
        }
        else if (a->text().toStdString() == "Print routing table") {
            caller_->call_print_routing_table(id_);
        }
    }

    std::string getName() const { return name_; }
    unsigned short getID() const { return id_; }

protected:
    void advance(int phase)
    {
        if (!phase) return;
        if (this->isDroppingPacket_) this->setBrush(Qt::black);
        else this->setBrush(color);

        ticks_ += animationSpeed_;
        if (ticks_ > 30) {
            isDroppingPacket_ = false;
            ticks_ = 0;
        }
    }


private:
    inline static std::vector<QGraphicsLineItem *> lines_;

    inline static std::vector<std::pair<NodeItem *, NodeItem *» connections_;

    inline static double animationSpeed_;

    bool isDroppingPacket_;

    double ticks_;


    unsigned short id_;

    std::string name_;

    QBrush color;

    QPointF offset_;

    QPointF position_;

    QGraphicsSimpleTextItem* nameItem_;

    NodeCaller *caller_;
};

#endif // NODEITEM_H
```

6.3 nodes.h

```
#ifndef NODES_HPP
#define NODES_HPP

#include "packet.h"
#include "nodeitem.h"

#include <string>
#include <map>
#include <queue>
#include <iostream>


#include <QThread>




extern std::map<std::string, unsigned short> DNS;

class Node {

protected:
    unsigned short id_;
    std::queue<Packet*> packets_;
    unsigned int max_capacity_; // KILObytes
    unsigned int current_capacity_; // KILObytes
    unsigned short connection_id_; // for routers end node id, end nodes router id
    std::pair<unsigned int, unsigned short> connection_cost_;
    NodeItem *nodeitem_;



public:
    Node(unsigned short id, unsigned int max_capacity, unsigned short connection_id, std::pair<unsigned
    int, unsigned short> connection_cost, NodeItem *nodeitem)
        :   id_(id), max_capacity_(max_capacity), connection_id_(connection_id), current_capacity_(0),
    connection_cost_(connection_cost), nodeitem_(nodeitem) {}

    virtual void send() = 0; // pop from queue, call receiver receive()
    virtual void send(Packet* packet) = 0; // cheat version
    virtual void receive(Packet* packet) = 0; // different for different nodes

    virtual void print() = 0;

    virtual ~Node() = default;

    // getters, setters etc.

    // COMPARING HAPPENS IN KILOBYTES
    bool not_full_after_add(unsigned int added_size) { // arg = KB
        return current_capacity_  + added_size <= max_capacity_;
    }


    void change_max_capacity(unsigned int new_capacity) { // kilobytes
        max_capacity_ = new_capacity;
    }

    void change_current_capacity(unsigned int change) {
        current_capacity_ += change;
    }

    unsigned short getId() const { return id_; }
    unsigned int getCurrentCapacity() const { return current_capacity_; }
    unsigned int getMaxCapacity() const { return max_capacity_; }
    size_t queue_packet_count() const { return packets_.size(); }
    unsigned short getConnectionId() const { return connection_id_; }

    virtual std::string getName() const = 0;
    NodeItem* getNodeItem() { return nodeitem_; }

};

class WaitThread : public QThread
{
    Q_OBJECT
    void run() override
    {
        unsigned int transmission_delay = size_ * 8 / bandwidth_; // KB -> Kb

        if (transmission_delay < 1) {
            double t_d = size_ * 8 / bandwidth_; // same as above, seconds
            t_d *= 1000000; // seconds to microseconds
            transmission_delay = t_d; // to unsigned int
            QThread::usleep(transmission_delay); // sleep microseconds
        } else {
            QThread::sleep(transmission_delay); // sleep seconds
        }

        QThread::usleep(1000 * latency_); // sleep microseconds
        node_->receive(packet_);
        return;
    }

public:
    WaitThread(Node* node, Packet* packet, unsigned int bandwidth, unsigned short latency, QObject* parent = nullptr) : QThread(parent), node_(node), packet_(packet), bandwidth_(bandwidth), latency_(latency) { size_ = packet->getSize(); }


private:
    unsigned int size_;
    unsigned int bandwidth_;
    unsigned short latency_;
    Node *node_;
    Packet *packet_;
};


extern std::map<unsigned short, Node*> main_map; // all nodes in simulation <node_id, Node>

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»KILObits and ms


public:
    Router(unsigned short id, unsigned int max_capacity, unsigned short connection_id, std::pair<unsigned int, unsigned short> connection_cost, NodeItem *nodeitem)
    :   Node(id, max_capacity, connection_id, connection_cost, nodeitem) {}

    void create_routing_table(const std::map<unsigned short, Router*>& routers);

    std::map<unsigned short, std::pair<unsigned int, unsigned short»& getNeighbors() { return neighbors_; }

    void print_routing_table() const;
    void print_connections() const;

    void addNeighbor(unsigned short id, const std::pair<unsigned int, unsigned short>& pair) { neighbors_.insert({id, pair}); }

    void send(); // normal version
    void send(Packet* packet); // cheat version
    void receive(Packet* packet);

    void send_to_end_node(Packet* packet);

    void print();

    std::string getName() const { return std::to_string(this->getId()); }


};

class Client : public Node {

private:
    std::string name_;

public:
    Client(unsigned short id, unsigned int max_capacity, unsigned short connection_id,
    std::pair<unsigned int, unsigned short> connection_cost, std::string name, NodeItem *nodeitem)
        : name_(name), Node(id, max_capacity, connection_id, connection_cost, nodeitem) {}

    void create_packet(unsigned short sender_id, unsigned short receiver_id, std::string content,
                                            unsigned int flag);

    void send(); // normal version
    void send(Packet* packet); // cheat version
    void receive(Packet* packet);
    void print();

    std::string getName() const { return name_; }

};

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, KILObytes

public:
    Server(unsigned short id, unsigned int max_capacity,  unsigned short connection_id,
    std::pair<unsigned int, unsigned short> connection_cost, std::string name, std::string content_type,
    unsigned int content_size, NodeItem *nodeitem)
    :   name_(name), content_type_(content_type), content_size_(content_size), Node(id, max_capacity,
    connection_id, connection_cost, nodeitem) {}

    void send(); // normal version
    void send(Packet* packet); // cheat version
    void receive(Packet* packet);
    void print();

    void add(Packet* packet) { packets_.push(packet); }


    std::string getContentType() const { return content_type_; }
    unsigned int getContentSize() const { return content_size_; }

    std::string getName() const { return name_; }

};

#endif
```

6.4 packet.h

```
#ifndef PACKET_H
#define PACKET_H

#include <string>
#include <time.h>
#include <unistd.h>
#include <iostream>


class Packet {

    private:
    unsigned int size_; // !!!CHANGED!!! KILObytes
    unsigned short sender_id_;
    unsigned short receiver_id_;
    std::string content_;
    time_t time_sent_;
    bool debug_; // print useful data when moving
    bool cheat_; // this will bypass the queue

    public:

    Packet(unsigned int size, unsigned short sender_id, unsigned short receiver_id, std::string content)
    :
    size_(size), sender_id_(sender_id), receiver_id_(receiver_id), content_(content), debug_(false),
    cheat_(false) {
        time_sent_ = time(NULL);
    }

    ~Packet() = default;

    // getters, setters etc.
    unsigned short getSenderId() const { return sender_id_; }
    unsigned short getReceiverId() const { return receiver_id_; }
    bool isDebug() const { return debug_; }
    bool isCheat() const { return cheat_; }
    unsigned int getSize() const { return size_; }
    time_t getTimeSent() const { return time_sent_; }

    void setSenderId(unsigned short new_id) { sender_id_ = new_id; }
    void setReceiverId(unsigned short new_id) { receiver_id_ = new_id; }
    void setDebug() { debug_ = true; }
    void setCheat() { cheat_ = true; }


    std::string& getContent() { return content_; }
    void setSize(unsigned int new_size) { size_ = new_size; }

    std::string getContent_print() const { return content_; }

    void wait(unsigned int bandwidth, unsigned short latency) { // BW: Mb/s, latency: ms, size_ KB
        // wait transfer
        unsigned int transmission_delay = size_ * 8 / bandwidth; // KB -> Kb

        if (transmission_delay < 1) {
            double t_d = size_ * 8 / bandwidth; // same as above, seconds
            t_d *= 1000000; // seconds to microseconds
            transmission_delay = t_d; // to unsigned int
            usleep(transmission_delay); // sleep microseconds
        } else {
            sleep(transmission_delay); // sleep seconds
        }

        usleep(1000 * latency); // sleep microseconds

    }

    void print() {
        std::cout « "Packet from " « sender_id_ « " to " « receiver_id_
        « "\nContent: " « content_ « ", size: " « size_ « "KB"
        « "\nDebug packet: " « debug_
        « "\nTime elapsed since creation: " « time(NULL) - time_sent_ « std::endl;
}

};


#endif

```

6.5 packetitem.h

```
#ifndef PACKETITEM_H
#define PACKETITEM_H

#include <QGraphicsScene>
#include <QGraphicsItem>
#include <QtMath>

class PacketItem : public QGraphicsPolygonItem
{

public:
    PacketItem(QPointF start, QPointF end, QGraphicsScene *scene) : QGraphicsPolygonItem()
    {
        qreal x = 0, y = 0;

        QPolygonF Triangle;
        Triangle.append(QPointF(x,y));
        Triangle.append(QPointF(x-10,y+13));
        Triangle.append(QPointF(x+10,y+13));
        Triangle.append(QPointF(x,y));

        QBrush redBrush(Qt::yellow);
        QPen blackpen(Qt::black);

        this->setPolygon(Triangle);
        this->setPen(blackpen);
        this->setBrush(redBrush);
        this->setZValue(1);

        scene->addItem(this);

        QPointF vector = end - start;
        qreal dot = QPointF::dotProduct(vector, QPointF(0, vector.ry()));
        qreal mvector = qSqrt(qPow(vector.rx(), 2) + qPow(vector.ry(), 2));

        qreal angle;
        if (vector.rx() >= 0)
            angle = qRadiansToDegrees(qAcos(-(dot/(mvector*vector.ry()))));
        else
            angle = qRadiansToDegrees(qAcos((dot/(mvector*vector.ry())))) - 180;

        this->setRotation(angle);
        this->setSpeed(mvector);
        this->setPos(start + (vector/3));
    }

    void resetTicks() { ticks = 0; }

    void setSpeed(qreal magnitude)
    {
        speed = animationSpeed * magnitude / 75;
    }

    void changeSpeed(double multiplier)
    {
        if (multiplier > 0) animationSpeed = multiplier;
    }

protected:
    void advance(int phase)
    {
        if (!phase) return;
        this->setPos(mapToParent(0,-(speed)));
        ticks += animationSpeed;
        if (ticks > 30) {
            this->scene()->removeItem(this);
            delete this;
        }
    }

private:
    qreal angle;

    qreal speed;

    double ticks;

    inline static double animationSpeed;
};

#endif // PACKETITEM_H
```

Index