verkkosimu

Generated by Doxygen 1.9.5

Chapter 1

Source content

This folder should contain only hpp/cpp files of your implementation. You can also place hpp files in a separate directory `include`.

You can create a summary of files here. It might be useful to describe file relations, and brief summary of their content.

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

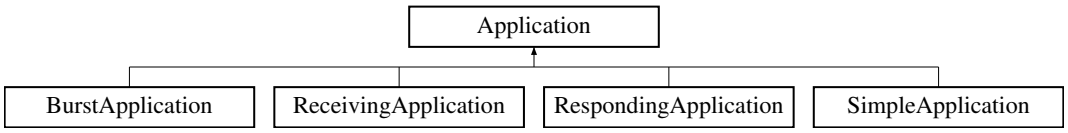
Class Documentation

5.1 Application Class Reference

The Application class is an abstract class that is inherited by different types of applications.

`#include <Application.h>`

Inheritance diagram for Application:



Public Member Functions

- **Application** (std::vector< int > destinationAddresses, int transmissionInterval, int packetSize)
- virtual Packet * packetGenerator (int source, Packet *currentPacket=nullptr)=0

  The packetGenerator function is called on every tick and returns a vector of Packet pointers created or an empty vector if none were created.

Protected Attributes

- std::vector< int > **destinationAddresses_**
- int **transmissionInterval_**
- int **packetSize_**

5.1.1 Detailed Description

The Application class is an abstract class that is inherited by different types of applications.

5.1.2 Member Function Documentation

5.1.2.1 packetGenerator()

```
virtual Packet * Application::packetGenerator (
            int source,
            Packet * currentPacket = nullptr )  [pure virtual]
```

The packetGenerator function is called on every tick and returns a vector of Packet pointers created or an empty vector if none were created.

Parameters

source	address of the node that is creating the packet

Implemented in SimpleApplication, BurstApplication, RespondingApplication, and ReceivingApplication.

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

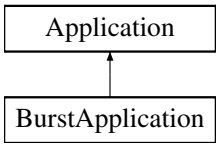
- Application.h
- Application.cpp

5.2 BurstApplication Class Reference

The BurstApplication class sends packets in bursts where packets are first sent for multiple ticks in a row and then packet sending goes on a break.

```
#include <Application.h>
```

Inheritance diagram for BurstApplication:



Public Member Functions

- **BurstApplication** (std::vector< int > destinationAddresses, int transmissionInterval, int packetSize)
- Packet * packetGenerator (int source, Packet *currentPacket)

  BurstApplication::packetGenerator sends a burst of packets every transmissionInterval_, otherwise sends a nullptr. Updates how many packets are left in this burst or how long until next burst, depending on whether a burst is ongoing when called.

Additional Inherited Members

5.2.1 Detailed Description

The BurstApplication class sends packets in bursts where packets are first sent for multiple ticks in a row and then packet sending goes on a break.

5.2.2 Member Function Documentation

5.2.2.1 packetGenerator()

```
Packet * BurstApplication::packetGenerator (
            int source,
            Packet * currentPacket )  [virtual]
```

BurstApplication::packetGenerator sends a burst of packets every transmissionInterval_, otherwise sends a nullptr. Updates how many packets are left in this burst or how long until next burst, depending on whether a burst is ongoing when called.

Implements Application.

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

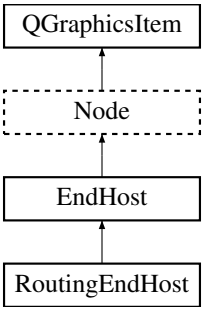
- Application.h
- Application.cpp

5.3 EndHost Class Reference

The EndHost class inherits Node. It does not route packets, only sends them based on its application's behavior.

```
#include <EndHost.h>
```

Inheritance diagram for EndHost:



Public Member Functions

- EndHost (int address, std::vector< int > application, std::vector< int > queue)

  EndHost constructor.
- ~**EndHost** ()

  EndHost::~EndHost destructor to delete application; virtual destructor of parent Node will delete packets.
- void processPacket (Packet *packet) override

  EndHost::processPacket forwards current packet to application, if no packets lets application know that, too.
- void paint (QPainter *painter, QStyleOptionGraphicsItem const *option, QWidget *widget) override

  EndHost::paint paints endhost specific.

Additional Inherited Members

5.3.1 Detailed Description

The EndHost class inherits Node. It does not route packets, only sends them based on its application's behavior.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 EndHost()

```
EndHost::EndHost (
            int address,
            std::vector< int > application,
            std::vector< int > queue )
```

EndHost constructor.

Parameters

address	network address
application	vector of variable length that defines a application. First item defines application type, rest are application specific parameters

5.3.3 Member Function Documentation

5.3.3.1 paint()

```
void EndHost::paint (
            QPainter * painter,
```

```
QStyleOptionGraphicsItem const * option,
QWidget * widget )  [override]
```

EndHost::paint paints endhost specific.

Parameters

painter	qt painter
option	unused qt stuff
widget	unused qt stuff

5.3.3.2 processPacket()

```
void EndHost::processPacket (
            Packet * packet = nullptr )  [override], [virtual]
```

EndHost::processPacket forwards current packet to application, if no packets lets application know that, too.

Parameters

packet	current packet this EndHost is handling, default value is nullptr (no packets)

Implements Node.

Reimplemented in RoutingEndHost.

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

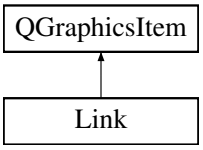
- EndHost.h
- EndHost.cpp

5.4 Link Class Reference

The Link class is a one-directional edge between nodes. Goes from node1 to node 2.

`#include <Link.h>`

Inheritance diagram for Link:



Public Member Functions

- **Link** (Node *node1, Node *node2, double transmissionSpeed, double propagationDelay)
- void **runOneTick** ()

  runOneTick advances pakcets in packets_ according to propagation delay. If the firs packet in the queue is transmitted, move it to the next node
- int receive (Packet *packet)

  receive sets the value of inTransmission to packet if there is no packet in transmission
- void **receivePackets** ()

  receivePackets advances the packet in inTransmission and moves it to packets_ when it is completely transmitted from the sender node to this link.
- const Node * **getDestination** () const
- const double **getTransmissionSpeed** () const
- const double **getPropagationDelay** () const
- int getCumulativeThroughput () const

  getCumulativeThroughput
- double getUtilization () const

  getUtilization
- QRectF **boundingRect** () const override
- void paint (QPainter *painter, QStyleOptionGraphicsItem const *option, QWidget *widget) override

  Link::paint paints a link.
- int dummyStat () const

  Link::dummyStat Generates dummy statistic.

5.4.1 Detailed Description

The Link class is a one-directional edge between nodes. Goes from node1 to node 2.

5.4.2 Member Function Documentation

5.4.2.1 dummyStat()

```
int Link::dummyStat ( ) const
```

Link::dummyStat Generates dummy statistic.

Returns

returns random integer

5.4.2.2 getCumulativeThroughput()

```
int Link::getCumulativeThroughput ( ) const
```

getCumulativeThroughput

Returns

amount of bits that have gone through this link

5.4.2.3 getUtilization()

```
double Link::getUtilization ( ) const
```

getUtilization

Returns

percentage of the link throughput that is currently used (1.0 = 100%)

5.4.2.4 paint()

```
void Link::paint (
            QPainter * painter,
            QStyleOptionGraphicsItem const * option,
            QWidget * widget )  [override]
```

Link::paint paints a link.

Parameters

painter	the painter to use
option	unused qt stuff
widget	unused qt stuff

5.4.2.5 receive()

```
int Link::receive (
            Packet * packet )
```

receive sets the value of inTransmission to packet if there is no packet in transmission

Parameters

packet	

Returns

1 if packet was received, 0 if packet couldn't be received

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

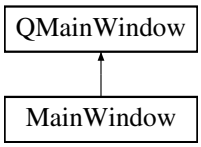
- Link.h
- Link.cpp

5.5 MainWindow Class Reference

The MainWindow class. Represents the only window of the application.

`#include <mainwindow.h>`

Inheritance diagram for MainWindow:



Signals

- void **invSimSignal** (bool state)
- void **updateTickLcd** (int tick)

Public Member Functions

- MainWindow (QWidget ∗parent=nullptr)
  MainWindow::MainWindow constructs Qt MainWindow.

Protected Member Functions

- void timerEvent (QTimerEvent ∗event) override
  MainWindow::timerEvent receives events.

5.5.1 Detailed Description

The MainWindow class. Represents the only window of the application.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 MainWindow()

```
MainWindow::MainWindow (
            QWidget * parent = nullptr )
```

MainWindow::MainWindow constructs Qt MainWindow.

Parameters

parent	parent relation to other widgets

5.5.3 Member Function Documentation

5.5.3.1 timerEvent()

```
void MainWindow::timerEvent (
            QTimerEvent * event )  [override], [protected]
```

MainWindow::timerEvent receives events.

Parameters

event	event pointer

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

- mainwindow.h
- mainwindow.cpp

5.6 Network Class Reference

The Network class class that takes ownership of nodes and links. Implements methods for address based link and node initialization.

`#include <Network.h>`

Public Member Functions

- Network (QString filename)

  Network::Network constructs the network based on a JSON file in resources.qrc.
- ~**Network** ()

  Network::~Network deconstructor deletes all Nodes and Links, and their respective deconstructors destroy packets.
- void **runOneTick** ()

  Network::runOneTick For all Nodes and Links, first calls runOneTick() and then, when all are done, calls receive↩ Packets() and this is done to make sure Nodes don't receive and pass forward a packet within the same tick.
- void addNode (Node ∗n)

  Network::addNode adds node to the network.
- void addLink (int a, int b, double bandwidth, double delay)

  Network::addLink adds link to the netowrk based on addresses, populates node's links_ information.
- void **initializeRoutingTables** () const

  Network::initializeRoutingTables Initialize all nodes' routing tables.
- void populateScene (QGraphicsScene ∗scene)

  Network::populateScene populates scene with this network.
- int getCurrentTick () const

  Network::getCurrentTick.

5.6.1 Detailed Description

The Network class class that takes ownership of nodes and links. Implements methods for address based link and node initialization.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Network()

```
Network::Network (
            QString filename )
```

Network::Network constructs the network based on a JSON file in resources.qrc.

Parameters

filename	name of the JSON file from which the network blueprint is read

5.6.3 Member Function Documentation

5.6.3.1 addLink()

```
void Network::addLink (
            int a,
            int b,
            double bandwidth,
            double delay )
```

Network::addLink adds link to the netowrk based on addresses, populates node's links_ information.

Parameters

a	node a address
b	node b address
bandwidth	link bandwidth
delay	link delay

5.6.3.2 addNode()

```
void Network::addNode (
```

```
            Node * node )
```

Network::addNode adds node to the network.

Parameters

node	pointer to a network node

5.6.3.3 getCurrentTick()

```
int Network::getCurrentTick ( ) const
```

Network::getCurrentTick.

Returns

current tick of the simulation

5.6.3.4 populateScene()

```
void Network::populateScene (
            QGraphicsScene * scene )
```

Network::populateScene populates scene with this network.

Parameters

scene	QGraphicsScene to populate

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

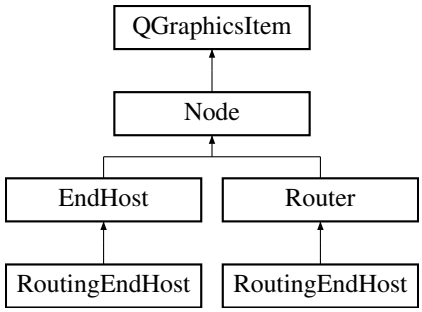
- Network.h
- Network.cpp

5.7 Node Class Reference

The Node class Abstract class that represents a node in a network.

```
#include <Node.h>
```

Inheritance diagram for Node:



Public Member Functions

- Node (int address, std::vector< int > queue)

  Node::Node constructor will parse queue type and initialize Queue packets_.*
- ~**Node** ()

  Node::~Node deconstructor will delete all packets in received and delete packets_, which is a Queue and Queue's own deconstructor will in turn destroy its own packets.*
- void **runOneTick** ()

  Node::runOneTick processes the next packet in line and sends it to processPacket; will route packets not meant for this Node to links according to lookupTable_.
- void receive (Packet *packet)

  Node::receive adds received packet to received_, from where it will be processed.
- void **receivePackets** ()

  Node::receivePackets transfers received packets to the packets_ queue, from where they will be processed one by one. The queue does not necessarily accept all of them, it may drop packets depending on its chosen behavior. The purpose of this is so that Nodes cannot process packets received this "tick".
- void **initializeRoutingTable** ()

  Node::initializeRoutingTable calculates the next link from this node for all destinations in the network using modified Dijkstra's algorithm, will only route packets through routers.
- virtual void processPacket (Packet *packet=nullptr)=0
- int getAddress () const

  Node::getAddress returns node's address.
- int dummyStat () const

  Node::dummyStat returns random integer from node's scope.
- void **addLink** (Link *link)
- int getLastPacketAge () const

  getLastPacketAge
- int getBufferSize () const

  getBufferSize
- void **hoverMoveEvent** (QGraphicsSceneHoverEvent *event) override
- QRectF **boundingRect** () const override

Protected Member Functions

- void drawTopText (QPainter *, QString)

  Node::drawTopText draw text in a box above the node.
- void drawBottomText (QPainter *, QString)

  Node::drawBelowText draw text in a box below the node - black if latest addition to queue was a success, red if packet was dropped.

Protected Attributes

- std::vector< Link * > **links_**
- Queue * **packets_**
- int **address_**
- std::vector< Packet * > **received_**
- std::map< int, Link * > **lookupTable_**
- Network * **network_**
- int **lastPacketAge_** = 0
- Application * **application_**
- bool **lastPacketStatus_** = true

5.7.1 Detailed Description

The Node class Abstract class that represents a node in a network.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Node()

```
Node::Node (
            int address,
            std::vector< int > queue )
```

Node::Node constructor will parse queue type and initialize Queue* packets_.

Parameters

address	integer address of this node
queue	vector which contains id (type) and parameter(s) or queue

5.7.3 Member Function Documentation

5.7.3.1 drawBottomText()

```
void Node::drawBottomText (
            QPainter * painter,
            QString text )  [protected]
```

Node::drawBelowText draw text in a box below the node - black if latest addition to queue was a success, red if packet was dropped.

Parameters

painter	qpainter to use
text	qstring of the text

5.7.3.2 drawTopText()

```
void Node::drawTopText (
            QPainter * painter,
            QString text )  [protected]
```

Node::drawTopText draw text in a box above the node.

Parameters

painter	qpainter to use
text	qstring of the text

5.7.3.3 dummyStat()

```
int Node::dummyStat ( ) const
```

Node::dummyStat returns random integer from node's scope.

Returns

the random integer

5.7.3.4 getAddress()

```
int Node::getAddress ( ) const
```

Node::getAddress returns node's address.

Returns

the integer address

5.7.3.5 getBufferSize()

```
int Node::getBufferSize ( ) const
```

getBufferSize

Returns

number of packets waiting their turn in the node's buffer

5.7.3.6 getLastPacketAge()

```
int Node::getLastPacketAge ( ) const
```

getLastPacketAge

Returns

age of last received packet in ticks

5.7.3.7 processPacket()

```
virtual void Node::processPacket (
            Packet * packet = nullptr )  [pure virtual]
```

Implemented in EndHost, Router, and RoutingEndHost.

5.7.3.8 receive()

```
void Node::receive (
            Packet * packet )
```

Node::receive adds received packet to received_, from where it will be processed.

Parameters

packet	packet from received

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

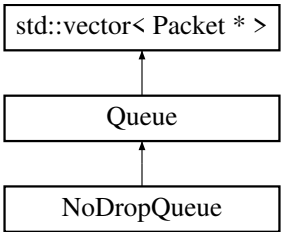
- Node.h
- Node.cpp

5.8 NoDropQueue Class Reference

The NoDropQueue class acts like a regular vector; no packets are dropped.

`#include <Queue.h>`

Inheritance diagram for NoDropQueue:



Public Member Functions

- bool maybe_push_back (Packet *) override

  NoDropQueue::maybe_push_back will always add packet; nothing is dropped.

5.8.1 Detailed Description

The NoDropQueue class acts like a regular vector; no packets are dropped.

5.8.2 Member Function Documentation

5.8.2.1 maybe_push_back()

```
bool NoDropQueue::maybe_push_back (
            Packet * packet )  [override], [virtual]
```

NoDropQueue::maybe_push_back will always add packet; nothing is dropped.

Parameters

packet	to be added to the queue

Returns

true always (never drops packet)

Implements Queue.

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

- Queue.h
- Queue.cpp

5.9 Packet Class Reference

The Packet class represents a network packet.

`#include <Packet.h>`

Public Member Functions

- **Packet** (int source, int destination, int size)
- void **runOneTick** ()

  Packet::runOneTick increments age of the packet by one, called by Packet's owner (a Link or a Node)
- int getAge () const

  Packet::getAge.

Public Attributes

- const int **sourceAddress**
- const int **destinationAddress**
- const int **size**
- double **transmitted** = 0.0
- double **received** = 0.0

5.9.1 Detailed Description

The Packet class represents a network packet.

5.9.2 Member Function Documentation

5.9.2.1 getAge()

```
int Packet::getAge ( ) const
```

Packet::getAge.

Returns

interger packet's age in ticks

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

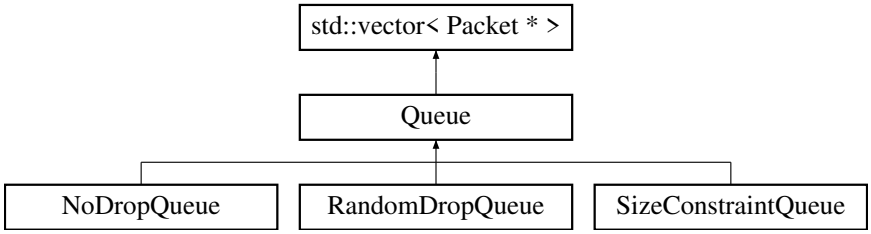
- Packet.h
- Packet.cpp

5.10 Queue Class Reference

The Queue class is abstract. It inherits std::vector and is otherwise similar, but includes (pure) virtual maybe_↩ push_back which will only call push_back if certain conditions are met.

`#include <Queue.h>`

Inheritance diagram for Queue:



Public Member Functions

- virtual ∼**Queue** ()

  Queue::∼Queue deconstructor will destoy all packets in the queue.
- virtual bool maybe_push_back (Packet ∗)=0

5.10.1 Detailed Description

The Queue class is abstract. It inherits std::vector and is otherwise similar, but includes (pure) virtual maybe_↩ push_back which will only call push_back if certain conditions are met.

5.10.2 Member Function Documentation

5.10.2.1 maybe_push_back()

```
virtual bool Queue::maybe_push_back (
            Packet *  )  [pure virtual]
```

Implemented in NoDropQueue, RandomDropQueue, and SizeConstraintQueue.

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

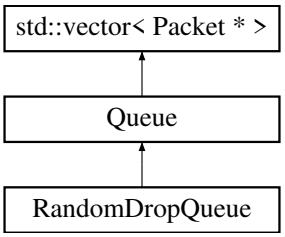
- Queue.h
- Queue.cpp

5.11 RandomDropQueue Class Reference

The RandomDropQueue class will drop packets with int chanceOfDrop probability, for example a value of 50 means half of the packets are dropped on average.

`#include <Queue.h>`

Inheritance diagram for RandomDropQueue:



Public Member Functions

- **RandomDropQueue** (int chanceOfDrop)
- bool maybe_push_back (Packet ∗) override

  RandomDropQueue::RandomDropQueue drops packet with 0-100 % probability (given when initializing a RandomDropQueue)

5.11.1 Detailed Description

The RandomDropQueue class will drop packets with int chanceOfDrop probability, for example a value of 50 means half of the packets are dropped on average.

5.11.2 Member Function Documentation

5.11.2.1 maybe_push_back()

```
bool RandomDropQueue::maybe_push_back (
            Packet * packet )  [override], [virtual]
```

RandomDropQueue::RandomDropQueue drops packet with 0-100 % probability (given when initializing a RandomDropQueue)

Parameters

packet	to be added or dropped

Returns

true if successfully added, false if dropped

Implements Queue.

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

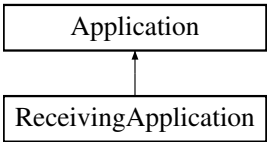
- Queue.h
- Queue.cpp

5.12 ReceivingApplication Class Reference

The ReceivingApplication class only receives packets.

`#include <Application.h>`

Inheritance diagram for ReceivingApplication:



Public Member Functions

- **ReceivingApplication** (std::vector< int > destinationAddresses={}, int transmissionInterval=0, int packet↩ Size=0)
- Packet * packetGenerator (int source, Packet *currentPacket)

  ReceivingApplication::packetGenerator will always return nullptr; this application does not send anything.

Additional Inherited Members

5.12.1 Detailed Description

The ReceivingApplication class only receives packets.

5.12.2 Member Function Documentation

5.12.2.1 packetGenerator()

```
Packet * ReceivingApplication::packetGenerator (
            int source,
            Packet * currentPacket )  [virtual]
```

ReceivingApplication::packetGenerator will always return nullptr; this application does not send anything.

Returns

always a nullptr

Implements Application.

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

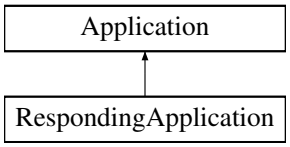
- Application.h
- Application.cpp

5.13 RespondingApplication Class Reference

The RespondingApplication class sends packets when it is sent packets.

`#include <Application.h>`

Inheritance diagram for RespondingApplication:



Public Member Functions

- **RespondingApplication** (std::vector< int > destinationAddresses, int transmissionInterval, int packetSize)
- Packet * packetGenerator (int source, Packet *currentPacket)

  RespondingApplication::packetGenerator will generate a new packet only when it receives a packet for itself; it responds to packets.

Additional Inherited Members

5.13.1 Detailed Description

The RespondingApplication class sends packets when it is sent packets.

5.13.2 Member Function Documentation

5.13.2.1 packetGenerator()

```
Packet * RespondingApplication::packetGenerator (
            int source,
            Packet * currentPacket )  [virtual]
```

RespondingApplication::packetGenerator will generate a new packet only when it receives a packet for itself; it responds to packets.

Returns

new packet to be sent or nullptr, if it itself received a nullptr

Implements Application.

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

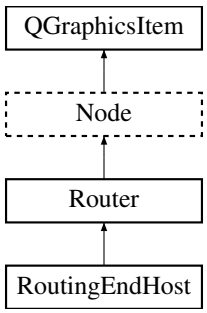
- Application.h
- Application.cpp

5.14 Router Class Reference

The Router class. A type of netowork node that is capable of routing packages forward.

`#include <Router.h>`

Inheritance diagram for Router:



Public Member Functions

- **Router** (int address, std::vector< int > queue)
- void processPacket (Packet *packet) override

  Router::processPacket processes packet; will simply destroy packet if its destination is this Router (which it shouldn't be), otherwise does nothing.
- void paint (QPainter *painter, QStyleOptionGraphicsItem const *option, QWidget *widget) override

  Router::paint paints router specifically.

Additional Inherited Members

5.14.1 Detailed Description

The Router class. A type of netowork node that is capable of routing packages forward.

5.14.2 Member Function Documentation

5.14.2.1 paint()

```
void Router::paint (
            QPainter * painter,
            QStyleOptionGraphicsItem const * option,
            QWidget * widget )  [override]
```

Router::paint paints router specifically.

Parameters

painter	painter to use
option	unused qt stuff
widget	unused qt stuff

5.14.2.2 processPacket()

```
void Router::processPacket (
            Packet * packet )  [override], [virtual]
```

Router::processPacket processes packet; will simply destroy packet if its destination is this Router (which it shouldn't be), otherwise does nothing.

Parameters

packet	

Implements Node.

Reimplemented in RoutingEndHost.

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

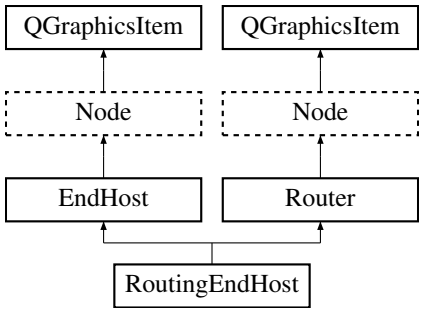
- Router.h
- Router.cpp

5.15 RoutingEndHost Class Reference

The RoutingEndHost class. Represents a type of network node that is capable of running applications like a endhost and forwarding packages like a router.

`#include <RoutingEndHost.h>`

Inheritance diagram for RoutingEndHost:



Public Member Functions

- **RoutingEndHost** (int address, std::vector< int > application, std::vector< int > queue)
- void processPacket (Packet *packet) override

  RoutingEndHost::processPacket forwards packet to application.
- void paint (QPainter *painter, QStyleOptionGraphicsItem const *option, QWidget *widget) override

  RoutingEndHost::paint paints routingendhost specific.

Additional Inherited Members

5.15.1 Detailed Description

The RoutingEndHost class. Represents a type of network node that is capable of running applications like a endhost and forwarding packages like a router.

5.15.2 Member Function Documentation

5.15.2.1 paint()

```
void RoutingEndHost::paint (
            QPainter * painter,
            QStyleOptionGraphicsItem const * option,
            QWidget * widget )  [override]
```

RoutingEndHost::paint paints routingendhost specific.

Parameters

painter	qt painter
option	unused qt stuff
widget	unused qt stuff

5.15.2.2 processPacket()

```
void RoutingEndHost::processPacket (
            Packet * packet = nullptr )  [override], [virtual]
```

RoutingEndHost::processPacket forwards packet to application.

Parameters

packet	to be processed, default value nullptr (no packets available)

Reimplemented from EndHost.

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

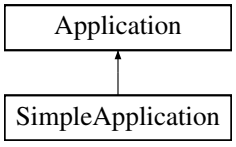
- RoutingEndHost.h
- RoutingEndHost.cpp

5.16 SimpleApplication Class Reference

The SimpleApplication class sends a single packet every tranmissionInterval_ ticks.

```
#include <Application.h>
```

Inheritance diagram for SimpleApplication:



Public Member Functions

- **SimpleApplication** (std::vector< int > destinationAddresses, int transmissionInterval, int packetSize)
- Packet * packetGenerator (int source, Packet *currentPacket)

  SimpleApplication::packetGenerator will generate packets depending on a predetermined interval; ignores the incoming packet.

Additional Inherited Members

5.16.1 Detailed Description

The SimpleApplication class sends a single packet every tranmissionInterval_ ticks.

5.16.2 Member Function Documentation

5.16.2.1 packetGenerator()

```
Packet * SimpleApplication::packetGenerator (
            int source,
            Packet * currentPacket )  [virtual]
```

SimpleApplication::packetGenerator will generate packets depending on a predetermined interval; ignores the incoming packet.

Returns

new packet to be sent or nullptr if nothing is sent

Implements Application.

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

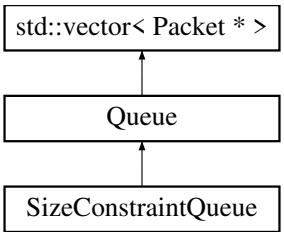
- Application.h
- Application.cpp

5.17 SizeConstraintQueue Class Reference

The SizeConstraintQueue class will drop new packets if queue size is at maxSize.

```
#include <Queue.h>
```

Inheritance diagram for SizeConstraintQueue:



Public Member Functions

- **SizeConstraintQueue** (int maxSize)
- bool maybe_push_back (Packet ∗) override

  SizeConstraintQueue::maybe_push_back pushes packet to queue if queue size is not at maxSize, which is given during initialization; otherwise drops packet.

5.17.1 Detailed Description

The SizeConstraintQueue class will drop new packets if queue size is at maxSize.

5.17.2 Member Function Documentation

5.17.2.1 maybe_push_back()

```
bool SizeConstraintQueue::maybe_push_back (
            Packet * packet )  [override], [virtual]
```

SizeConstraintQueue::maybe_push_back pushes packet to queue if queue size is not at maxSize, which is given during initialization; otherwise drops packet.

Parameters

packet	to be added or dropped

Returns

true if successfully added, false if dropped

Implements Queue.

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

- Queue.h
- Queue.cpp

Chapter 6

File Documentation

6.1 Application.h

```
1 #ifndef APPLICATION_H
2 #define APPLICATION_H
3
4 #include "Packet.h"
5 #include <vector>
6 #include <cstdlib>
7
12 class Application
13 {
14 public:
15     Application(std::vector<int> destinationAddresses, int transmissionInterval, int packetSize);
16     virtual ~Application(){};
17
24     virtual Packet* packetGenerator(int source, Packet* currentPacket = nullptr) = 0;
25 protected:
26     std::vector<int> destinationAddresses_;
27     int transmissionInterval_;
28     int packetSize_;
29 };
30
34 class SimpleApplication : public Application
35 {
36 public:
37     SimpleApplication(std::vector<int> destinationAddresses, int transmissionInterval, int packetSize);
38     ~SimpleApplication(){}
39     Packet* packetGenerator(int source, Packet* currentPacket);
40 private:
41     int counter_;
42 };
43
44
49 class BurstApplication : public Application
50 {
51 public:
52     BurstApplication(std::vector<int> destinationAddresses, int transmissionInterval, int packetSize);
53     ~BurstApplication(){}
54     Packet* packetGenerator(int source, Packet* currentPacket);
55 private:
60     int currentDestination_;
61     int packetsLeftThisBurst_ = 0;
62     int counter_;
63 };
64
65
69 class RespondingApplication : public Application
70 {
71 public:
72     RespondingApplication(std::vector<int> destinationAddresses, int transmissionInterval, int
       packetSize);
73     ~RespondingApplication(){}
74     Packet* packetGenerator(int source, Packet* currentPacket);
75 };
76
81 class ReceivingApplication : public Application
82 {
83 public:
84     ReceivingApplication(std::vector<int> destinationAddresses = {}, int transmissionInterval = 0, int
       packetSize = 0);
```

```
85      ~ReceivingApplication(){}
86      Packet* packetGenerator(int source, Packet* currentPacket);
87  };
88
89
90  #endif // APPLICATION_H
```

6.2 EndHost.h

```
1  #ifndef ENDHOST_H
2  #define ENDHOST_H
3
4  #include "Node.h"
5  #include "Application.h"
6
11 class EndHost : virtual public Node
12 {
13 public:
14      EndHost(int address, std::vector<int> application, std::vector<int> queue);
15      ~EndHost();
16      void processPacket(Packet *packet) override;
17      void paint(QPainter *painter, QStyleOptionGraphicsItem const *option, QWidget *widget) override;
18
19 private:
20      Application* application_;
21 };
22
23
24 #endif // ENDHOST_H
```

6.3 Link.h

```
1  #ifndef LINK_H
2  #define LINK_H
3
4  #include "Packet.h"
5  #include <QGraphicsItem>
6  #include <QQueue>
7  #include <cmath>
8  #include <QDebug>
9
10 class Node; // forward definition, can not include normally
11
16 class Link : public QGraphicsItem
17 {
18 public:
19      Link(Node* node1, Node* node2, double transmissionSpeed, double propagationDelay);
20      ~Link();
21
22      void runOneTick();
23
24
25      int receive(Packet* packet);
26
27      void receivePackets();
28
29      const Node* getDestination() const;
30      const double getTransmissionSpeed() const;
31      const double getPropagationDelay() const;
32
33
34      int getCumulativeThroughput() const;
35      double getUtilization() const;
36
37      // virtual mehtods inherited from Qt that must be implemented
38      QRectF boundingRect() const override;
39      void paint(QPainter *painter, QStyleOptionGraphicsItem const *option, QWidget *widget) override;
40
41      int dummyStat() const;
42
43 private:
44      double transmissionSpeed_;
45      double propagationDelay_;
46      Node* node1_;
47      Node* node2_;
48      QQueue<Packet*> packets_;
49      Packet* inTransmission_ = nullptr;
50
54      double maxThroughput_;
```

```
55
60      int cumulativeThroughput_ = 0;
65      int currentThroughput_ = 0;
66
67 };
68
69 #endif // LINK_H
```

6.4 mainwindow.h

```
1 #ifndef MAINWINDOW_H
2 #define MAINWINDOW_H
3
4 #include "Network.h"
5 #include <QMainWindow>
6 #include <QGraphicsScene>
7 #include <QLCDNumber>
8
9 QT_BEGIN_NAMESPACE
10 namespace Ui { class MainWindow; }
11 QT_END_NAMESPACE
12
16 class MainWindow : public QMainWindow
17 {
18     Q_OBJECT
19
20 public:
21     MainWindow(QWidget *parent = nullptr);
22     ~MainWindow();
23 signals:
24     void invSimSignal(bool state);
25     void updateTickLcd(int tick);
26
27 protected:
28     void timerEvent(QTimerEvent *event) override;
29
30 private slots:
31     void on_actionExit_triggered();
32
33     void on_actionLoad_Simulation_triggered();
34
35     void on_pushButton_2_clicked();
36
37     void on_pushButton_clicked(bool checked);
38
39     void invSimState(bool state) {
40         emit invSimSignal(!state);
41     };
42
43
44
45 private:
46     Ui::MainWindow *ui;
47     QGraphicsScene *scene_;
48     QLCDNumber *ticklcd_;
49     Network *network_;
50     qint64 simulationtimerid_;
51     void replaceNetwork(Network* network);
52     void runOneTick();
53 };
54
55
56 #endif // MAINWINDOW_H
```

6.5 Network.h

```
1 #ifndef NETWORK_H
2 #define NETWORK_H
3
4 #include "Link.h"
5 #include <QGraphicsItem>
6 #include <vector>
7 #include <iostream>
8 #include <QString>
9 #include <QMap>
10
14 class Network
15 {
16 public:
```

```
17     Network() {};
18     Network(QString filename);
19     ~Network();
20     void runOneTick();
21     void addNode(Node* n);
22     void addLink(int a, int b, double bandwidth, double delay);
23     void initializeRoutingTables() const;
24     void populateScene(QGraphicsScene* scene);
25     int getCurrentTick() const;
26
27 private:
28     QMap<int, Node*> nodes_; // address-node
29     std::vector<Link*> links_;
30     int tick_ = 0;
31 };
32
33 #endif // NETWORK_H
```

6.6 Node.h

```
1 #ifndef NODE_H
2 #define NODE_H
3
4 #include "Link.h"
5 #include "Packet.h"
6 #include "Network.h"
7 #include "Application.h"
8 #include "Queue.h"
9 #include <QGraphicsItem>
10 #include <vector>
11 #include <queue>
12 #include <QPainter>
13
14 constexpr double sizeconst = 25;
15
19 class Node : public QGraphicsItem
20 {
21 public:
22     Node(int address, std::vector<int> queue);
23     ~Node();
24
25     void runOneTick();
26     void receive(Packet* packet);
27     void receivePackets();
28     void initializeRoutingTable();
29     virtual void processPacket(Packet *packet = nullptr) = 0;
30
31     int getAddress() const;
32     int dummyStat() const;
33     void addLink(Link* link);
34
39     int getLastPacketAge() const;
40
45     int getBufferSize() const;
46
47     void hoverMoveEvent(QGraphicsSceneHoverEvent *event) override;
48     QRectF boundingRect() const override { return QRectF(-sizeconst, -sizeconst, sizeconst*2,
       sizeconst*2); }
49
50 protected:
51     void drawTopText(QPainter*, QString);
52     void drawBottomText(QPainter*, QString);
53
54     std::vector<Link*> links_;
55     Queue* packets_;
56     int address_;
57     std::vector<Packet*> received_;
58     std::map<int, Link*> lookupTable_;
59     Network* network_;
60     int lastPacketAge_ = 0;
61     Application* application_;
62     bool lastPacketStatus_ = true;
63 };
64
65 #endif // NODE_H
```

6.7 Packet.h

```
1 #ifndef PACKET_H
```

```
2 #define PACKET_H
3
4 #include <QGraphicsItem>
5
9 class Packet
10 {
11 public:
12     Packet(int source, int destination, int size);
13     void runOneTick();
14     const int sourceAddress;
15     const int destinationAddress;
16     const int size;
17     int getAge() const;
18     double transmitted = 0.0;
19     double received = 0.0;
20 private:
21     int age_ = 0;
22 };
23
24 #endif // PACKET_H
```

6.8 Queue.h

```
1 #ifndef QUEUE_H
2 #define QUEUE_H
3 #include <vector>
4 #include <Packet.h>
5
11 class Queue : public std::vector<Packet*>
12 {
13 public:
14     Queue(){};
15     virtual ~Queue();
16     virtual bool maybe_push_back(Packet*) = 0;
17 };
18
22 class NoDropQueue : public Queue
23 {
24 public:
25     NoDropQueue();
26     ~NoDropQueue(){};
27     bool maybe_push_back(Packet*) override;
28 };
29
34 class RandomDropQueue : public Queue
35 {
36 public:
37     RandomDropQueue(int chanceOfDrop);
38     ~RandomDropQueue(){};
39     bool maybe_push_back(Packet*) override;
40 private:
41     int chanceOfDrop_;
42 };
43
47 class SizeConstraintQueue : public Queue
48 {
49 public:
50     SizeConstraintQueue(int maxSize);
51     ~SizeConstraintQueue(){};
52     bool maybe_push_back(Packet*) override;
53 private:
54     double maxSize_;
55 };
56
57
58
59 #endif // QUEUE_H
```

6.9 Router.h

```
1 #ifndef ROUTER_H
2 #define ROUTER_H
3
4 #include "Packet.h"
5 #include "Node.h"
6
10 class Router : virtual public Node
11 {
12 public:
```

```
13     Router(int address, std::vector<int> queue);
14     ~Router(){}
15     void processPacket(Packet *packet) override;
16     void paint(QPainter *painter, QStyleOptionGraphicsItem const *option, QWidget *widget) override;
17
18 private:
19     void nexthop(Packet* packet);
20 };
21
22 #endif // ROUTER_H
```

6.10 RoutingEndHost.h

```
1 #ifndef ROUTINGENDHOST_H
2 #define ROUTINGENDHOST_H
3
4 #include "Router.h"
5 #include "EndHost.h"
6
10 class RoutingEndHost : public EndHost, public Router
11 {
12 public:
13     RoutingEndHost(int address, std::vector<int> application, std::vector<int> queue);
14     ~RoutingEndHost(){}
15     void processPacket(Packet *packet) override;
16     void paint(QPainter *painter, QStyleOptionGraphicsItem const *option, QWidget *widget) override;
17 };
18
19
20 #endif // ROUTINGENDHOST_H
```

Index