Project NetSim

Adaptive Systems

Routing protocol simulation

Christoph Hauer

Puchberg am Schneeberg, September 10, 2016

Contents

Abbreviations

.NET	Microsoft .NET Framework
App	Application
API	Application Programming Interface
AODV	Ad hoc On-Demand Distance Vector Routing
DSDV	Destination-Sequenced Distance-Vector Routing
DSR	Dynamic Source Routing
OLSR	Optimized Link State Routing Protocol
CPU	Central Processing Unit
JSON	Java Script Object Notation
MVVM	Model-View-ViewModel
WPF	Windows Presentation Foundation

Chapter 1

Introduction

This project is created as part of the course Adaptive Systems. The goal of the project is to simulate various dynamic routing protocols. The following chapter will explain the problem description of the project, the investigation methods and the taken assumtions based on the problem description.

1.1. Problem description

The task for this project is as follows. Simulate various dynamic routing protocols for a varying number of participating nodes. You must provide a visualization of your results which demonstrates the necessary steps for handling the routing mechanism (e.g., the creation of routing tables). You must provide realization parts for all routing protocols. The endresult should be a prototype with test procedures and a documentation file.

- Implementation of varying node count and edges
- Correct realization of DSDV/OLSR/AODV/DSR
- Correct handling of churns
- Visualization of the network
- Visualization of the protocol specific aspects
- Describe your idea(s) and used algorithm(s)

1.2. Investigation

The following research methods were used to obtain information for this work:

- Literature review
- Own experience of programming training

For the literature review scientific papers and RFC papers were used and analyzed subsequently.

1.3. Assumptions

Following assumptions were taken based on the problem description.

1.3.1. Message formats

Based on the problem description the project will not simulate the exact details of the protocols but only the basic operations. That means the project uses not the exact messages formats of the protocols.

1.3.2. Address Identification

Since not stated otherwise in the problem description, the simulator uses upercase letters as identifiers for the nodes in the network. This can be taken as analogy to the IP Protocol. Each identifier in the network has to be unique to ensure proper operation of the simulator.

1.3.3. Broken connection detection

Based on the problem description the project will not simulate the accurate broken connection detection. That means the simulator knows based on his architecture is a link is broken or offline and has not use the basic link sensing features of the protocols to detect this connection.

1.3.4. Metric simplifying

Since not stated otherwise in the problem description, for simplifying the simulation, the simulator will handle the metric of routes as hopcount. That means the metric between one-hop neighbour A and B will always be one.

1.3.5. Bi-directional connections

Since not stated otherwise in the problem description, the simulator can only create bi-directional connections.

1.3.6. Single connections

Since not stated otherwise in the problem description, the simulator can only create single connections between network nodes.

1.3.7. Message sizes

Based on the problem description the project will not handle message sizes or message fragmenting.

1.3.8. Connection Protocol

Based on the problem description the simulator will use an analogy to the enthernet frame message. This frame messages handles the end to end transmission of messages between two nodes. Each messages gets encapsulated in the so called connection frame message. This method has to be used because of the possibility to forward messages from Node A to C over the Node B.

Chapter 2

Implementation

This chapter gets down to the architectural and implementation part of the simulation project.

2.1. Architecture

This sections provides an overview for architecture of the project. The figure 2.1 shows the base classes which provide the base for creating the simulator and the specific routing protocol implementations. The figure in Appendix A.1 shows the hole archicture of the NetSim simulator project.

2.1.1. Network

The network is created with NetSimClient objects for the nodes and NetSimConnection objects for the edges of the network. Each of this classes is inherited from the NetSimItem class which provides an Id and Location for the visualization of the simulation (see 2.1.5).

2.1.2. Simulator

At the top of figure 2.1 there is the NetSimSimulator class which provides the logic for adding nodes or clients and edges or connections. It holds references to all clients and connections in the simulator.

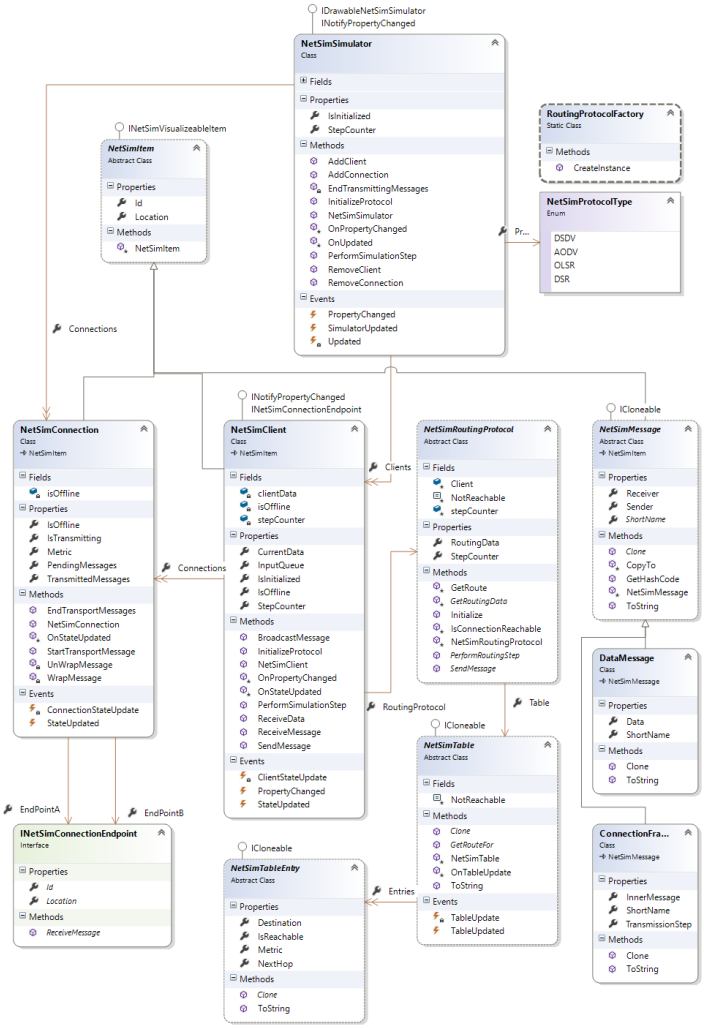

Figure 2.1.: Base classes of the architecture of the NetSim Project

The simulation is performed in steps. Each step is executed by each client in a random order. A simulation step is executed through the method PerformSimulationStep and gets forwarded to each client.

2.1.3. Messages

The abstract NetSimMessage class provides a base class for every messages sent via the network of the simulator. It provides information about the sender, receiver and the ShortName (e.g. Data) of the specific message. The DataMessage is a protocol unspecific message type which is intended to send data from a node to another.

Additionally in some protocols it's used to trigger the route search mechanisms. The ConnectionFrame message is an equivalent to the Ethernet frame message and can wrap each other message. This message gets used when transmitting a message via a connection to map the local transmission endpoints.

2.1.4. Routing Protocol

Figure 2.1 shows the abstract NetSimProtocol class under the simulator class. This class provides the possibility to create an inherited version for each specific routing protocol. It holds also a reference to an NetSimTable class which is an abstract base class for the routing tables of the protocols.

2.1.5. Visualizer

The visualizer part of the project creates based on the current state of the simulation elements on a canvas. The positioning of the elements is done with the help of the location saved in the NetSimItem base class. The visualizer redraws every time the simulator recognizes a change on something.

2.2. Technology

This section describes the used technolgies for the project.

- The simulator client is written in C# 6.0 and .NET Framework 4.5.1.
- The WPF Framework was used for the Graphical User Interface.
- MVVM Light [1] was used as Model-View-ViewModel Framework when creating the GUI.
- JSON and Newtonsoft JSON.NET [2] was used for the save and load functioniality of a network.

2.2.1. Coding Guidelines

The project was verified for code style guidelines with StyleCop 4.7.54.

Chapter 3

Routing Protocols

This chapter describes the implemented protocols, ideas and algorithms' used to create the simulation project.

3.1. Dynamic Source Routing

The DSR Protocol was realized based on the teaching material [3] of the FH Wiener Neustadt course Adative Systeme and the information found in [4].

The basic message type of the DSR protocol implementation are:

- Route request message - for route discovery or search
- Route reply - as a route information found message
- Route error - for failure messages

As seen in figure 3.1 the route request and reploy messages save the path to the destination in the message itself. To handle e.g. Data Messages in this routing protocol a additional message type had to be created. The DSR Frame messages handles the wrapping of a normal messages and provides the additional route information.

3.1.1. Assumptions

Since not stated otherwise in the the problem description, the simulator will only provide the basic correct functioning like in [3] and not every detail of the protocol realization as stated in [4].

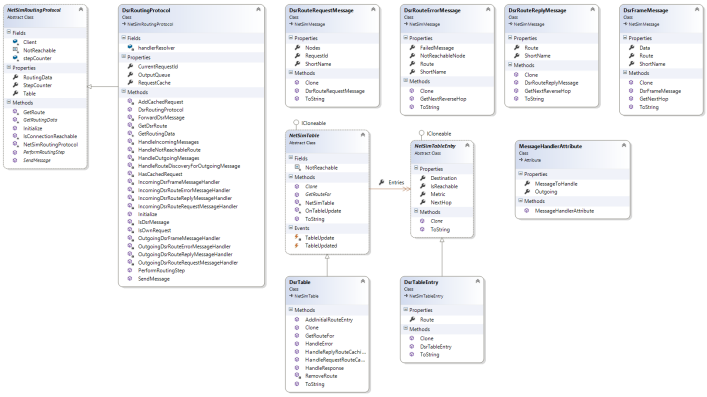

Figure 3.1.: Implementation of the DSR protocol.

3.1.2. Message handling

The message handling in the DSR protocol implementation was realized with a MessageHandler attribute as seen in listing 3.1. This attribute is assigned to the message handler method in the DSR routing protocol class. When a new incoming message is detected the protocol class searches the right handler for the message dynamically. This way of handling messages was also applied to outgoing messages.

```
[MessageHandler(typeof(DsrFrameMessage), Outgoing = false)]
private void IncomingDsrFrameMessageHandler(NetSimMessage message)
```

Listing 3.1: Message handler attribute and method signature.

3.2. Ad hoc On-Demand Distance Vector

The AODV Protocol was realized based on the teaching material [3] of the FH Wiener Neustadt course Adative Systeme and the information found in [5].

The messaing handling of the AODV protocol was realized the same the way as in the DSR protocol implementation (see section 3.1.2). Figure 3.2 shows the implementation of the AODV protocol.

The basic message type of the DSR protocol implementation are:

- Route request message - for route discovery or search
- Route reply - as a route information found message
- Route error - for failure messages

- Hello Message - for route maintenance

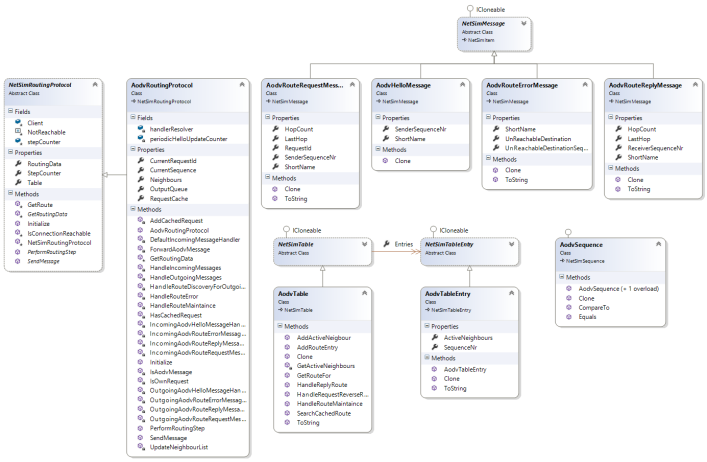

Figure 3.2.: Implementation of the AODV protocol.

The hello message as seen in 3.2 is used for route maintenance and not neighbour detection like in OLSR protocol (see section 3.3). This is important because AODV should be an reactive not proactive routing protocol [3, 5]

3.2.1. Neighbours

The AODV protocol has a few specific features. The first feature is the storage of the neighbors. The AODV protocol differs between two types of neighbours. Neighbours that are directly connected and neighbours on an active path.

The neighbours directly connected have to answer the broadcasted Hello messages in a period of time, otherwise the protocol handles the route to the neighbour as unreachable. [5]

The active neighbours are neighbours along an active path. That means every neighbour that sends messages over a given route is an active neighbour of this route. The protocol stores this neighbours directly to the route. [5]

3.2.2. Assumptions

Since not stated otherwise in the the problem description, the simulator will only provide the basic correct functioning as in [3] and not every detail of the protocol realization as stated in [5].

Following simplifications has been applied:

- The detailed realization of the protocol neighbour detection was simplified.
- The error message caching was not implemented.
- Acknoledgement messages were not implemented.

3.3. Optimized Link State Routing

The OLSR Protocol was realized based on the teaching material [3] of the FH Wiener Neustadt course Adative Systeme and the information found in [6].

The messaing handling of the OLSR protocol was realized the same the way as in the DSR protocol implementation (see section 3.1.2) with the only difference that is was only applied to incoming messages.

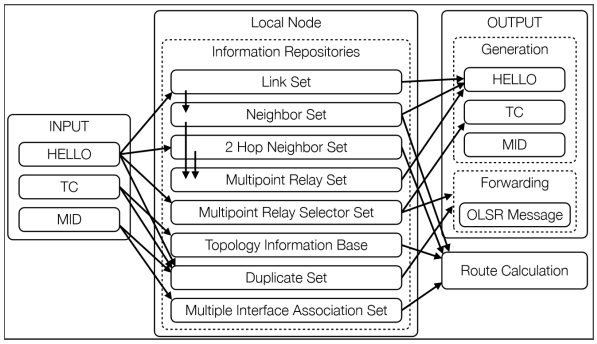

Figure 3.3.: Diagramm for the OLSR protocol data flow. [7]

As seen in figure 3.4 the OLSR routing protocol class has a state property. This state property represents the current state of the OLSR node in simulator.

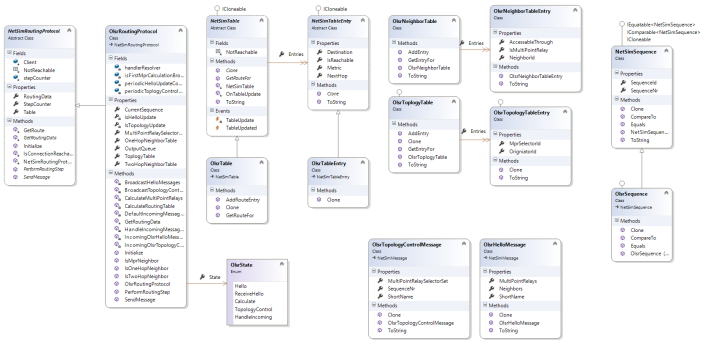

Figure 3.4.: Implementation of the OLSR protocol.

Following states are implemented:

- Hello - Broadcast your own neighbour (and mpr if calculated) information.
- Receive Hello - Wait for incoming hello messages and set state to hello again if incoming hello changes something on the neighbour tables.
- Calculate - The nodes calculates his multipoint relays. After finishing returns to state hello again to share information.
- Topology Control - Only nodes that are selcted as MPR by other nodes broadcast TC messages. [6]
- Handle Incoming - If node is not a MPR node, it has only to handle incoming messages and forward it if needed.

3.3.1. MPR Selection

Figure 3.5 shows how the simulator marks the specific neighbours nodes for node A.

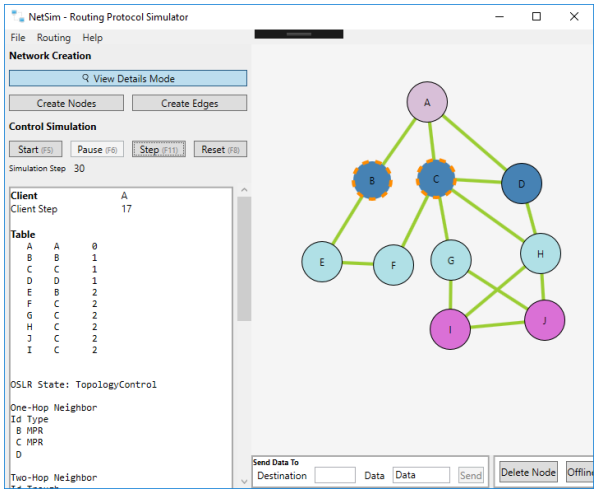

Figure 3.5.: Screenshot of simulator with as MPR marked nodes.

One-hop neighbours are marked as dark blue. Two-hop neighbours are marked as light blue. Multipoint relays have a dashed orange boarder around the node.

3.3.2. Assumptions

Since not stated otherwise in the the problem description, the simulator will only provide the basic correct functioning as in [3] and not every detail of the protocol realization as stated in [6].

Following simplifications has been applied:

- The detailed realization of the protocol one hop neighbour and two hop neighbour detection was simplified.
- MID functionality was not implemented (including MID messages).
- Acknoledgement messages were not implemented.

3.4. Destination-Sequenced Distance-Vector Routing

The DSDV Protocol was realized based on the teaching material [3] of the FH Wiener Neustadt course Adative Systeme and the information found in [8].

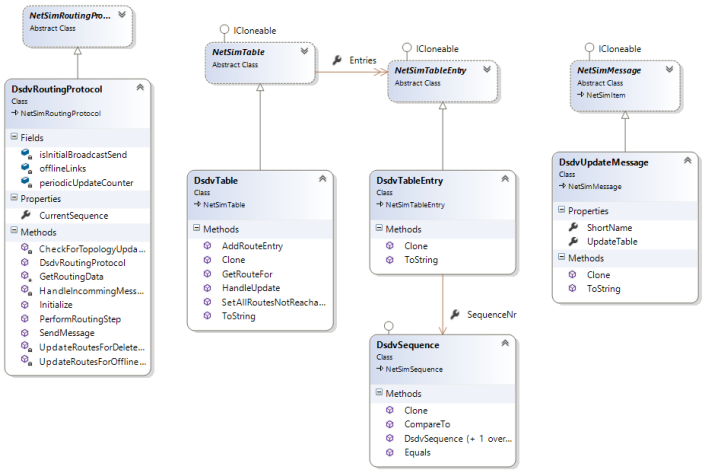

Figure 3.6.: Implementation of the DSDV protocol.

3.4.1. Assumptions

Since not stated otherwise in the the problem description, the simulator will only provide the basic correct functioning as in [3] and not every detail of the protocol realization as stated in [8].

Chapter 4

Help

This chapter provides a brief introduction to the use of simulation project. Figure 4.1 shows the User Interface of the NetSim Routing Protocol Simulator project.

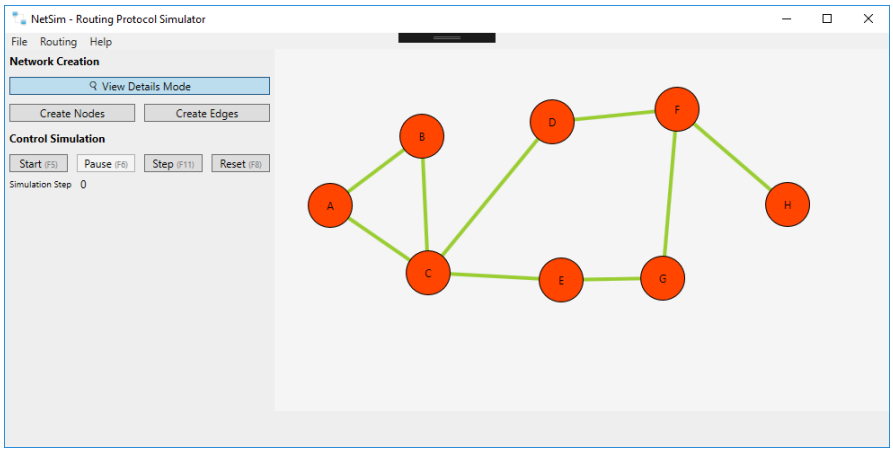

Figure 4.1.: Screenshot of the simulation project UI.

On the left side there is the network and simulation control part of the UI. The light gray area on the right side is the network canvas. On this canvas nodes and edges are created.

4.1. Network Creation and Management

The section takes care of creating network nodes and edges. The network creation has three modes:

- View Mode

- Creating nodes
- Creating edges

4.1.1. Create nodes

To create nodes click on the Create Nodes button as seen in figure 4.2. This starts the nodes creation mode. After clicking somewhere on the light gray canvas a network node gets created. The naming of the nodes start with the capital letter A.

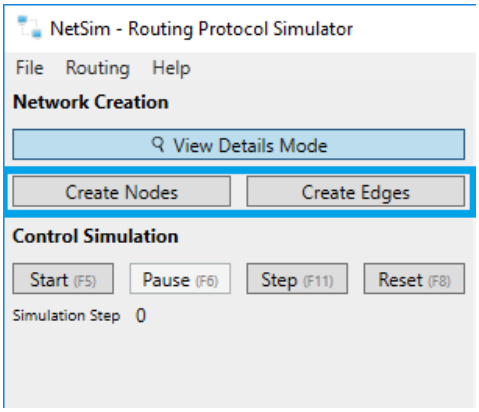

Figure 4.2.: Create Nodes and Create Edges Button of the simulator UI.

4.1.2. Create edges

To create edges click on the Create Edges button as seen in figure 4.2. This starts the edges creation mode. After clicking and holding the left mouse button on an already created node on the light gray canvas a red line gets displayed as seen in Figure 4.3. To finish an edge drag this line to another created node. When the connection is possible the display line gets green and you can stop holding the left mouse button.

4.1.3. View element details

The simulator has a the ability to view the details of every node an edge on the network canvas as seen in figure 4.4. To view the details of an element the view details mode has to be enabled. This is accomplished with the view details mode

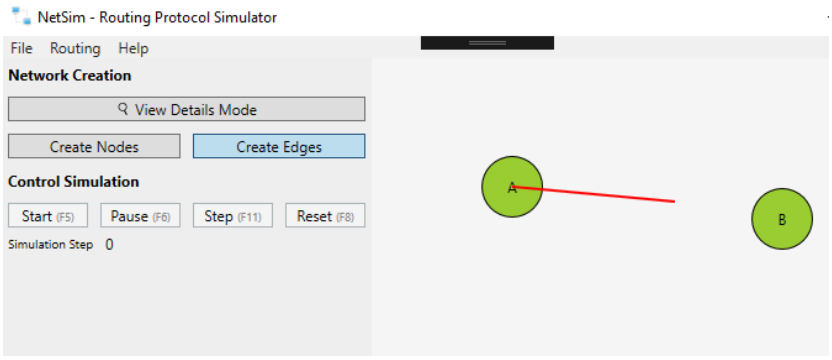

Figure 4.3.: Screenshot of creating an edge.

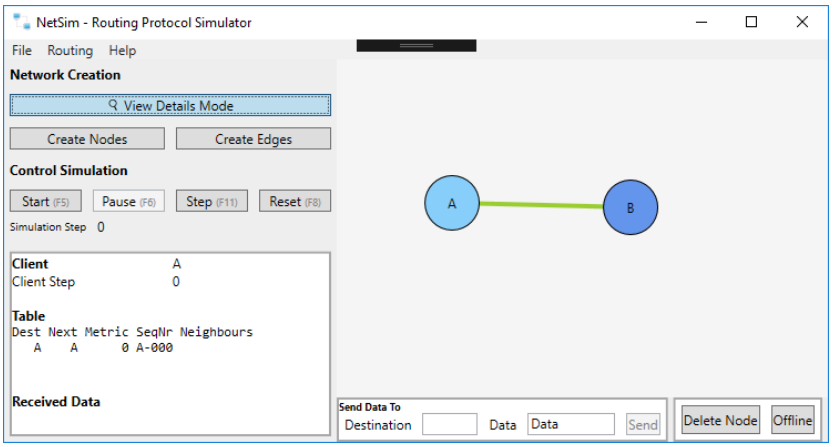

Figure 4.4.: Screenshot of the details view mode of the simulator ui.

button on the top of the left part of the UI. To view the details of an specific element click on the this element.

The details on the left bottom part of the UI as seen in figure 4.4 displaying things like the client name, the current simulation step, received data messages and the protocol specific information like routing tables.

On bottom part of the UI, it's possible to send a data message to an other node on the network.

4.1.4. Mark element as offline

To mark an element like a node or an edge as offline get into the view details mode of the simulator and click on the specific element on the network canvas. To mark the element as offline click on the button offline on the bottom right part of the UI.

If a node gets marked as offline each direct connected edge gets marked as offline to.

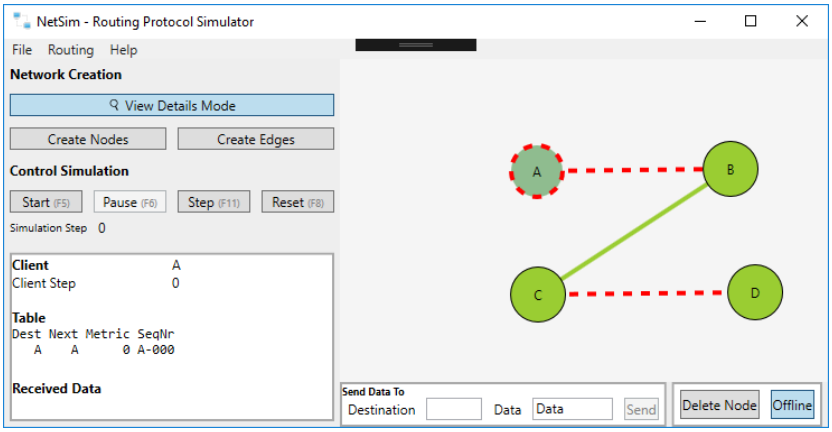

Figure 4.5.: Screenshot from offline nodes ans edges on the network canvas.

As seen in figure 4.5 the connection between node C and node D is marked as offline. Also the node A is marked as offline and so every direct connection of the node is marked as offline to.

4.1.5. Delete a element

To delete an element like a node or an edge get into the view details mode of the simulator and click on the specific element on the network canvas. To delete the element click on the delete node or delete edge button on the bottom right part of the UI as seen in figure 4.6. If a node gets deleted each direct connected edge gets marked as offline to.

Figure 4.6.: Screenshot of the delete and offline button of the UI.

4.2. Simulate Routing

To control the simulation the UI provides four buttons as seen in figure 4.7.

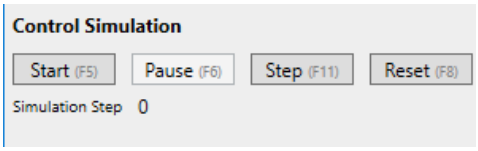

Figure 4.7.: Screenshot of the delete and offline button of the UI.

The start button starts the simulation, which means a simulation step gets executed about each half second. To pause the simulatio run use the pause button. The step button performs only one simulation step. The reset button sets the simulation to the initial state. It doesn't delete the created network!

Appendix A

Appendix

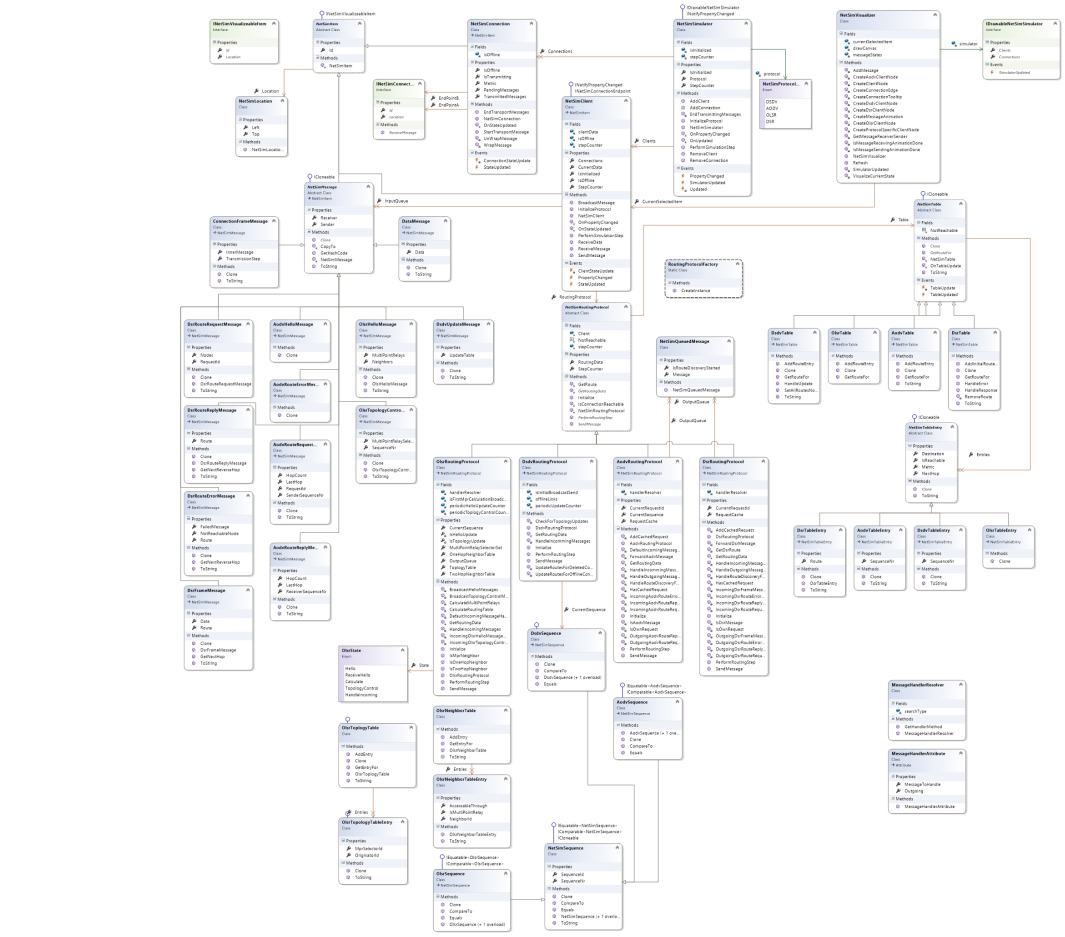

Figure A.1.: Architecture of the NetSim Project

List of Figures

Listings

Bibliography

[1] L. Bugnion, "MVVM Light Toolkit." Website. Available online http://www.mvvmlight.net/doc; retrieved on 3. September 2016.

[2] "Json.NET - Popular high-performance JSON framework for .NET." Website. Available online http://www.newtonsoft.com/json; retrieved on 3. September 2016.

[3] M. Szvetits, "Adaptive architectures." Teaching material, 2016.

[4] D. Johnson, Y. Hu, and D. Maltz, "The dynamic source routing protocol (dsr) for mobile ad hoc networks for ipv4," 2 2007. RFC 4728 Experimental.

[5] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," 7 2003. RFC 3561 Experimental.

[6] T. Clausen and P. Jacquet, "Optimized link state routing protocol (olsr)," 10 2003. RFC 3626 Experimental.

[7] "Optimized Link State Routing Protocol." Website. Available online https://en.wikipedia.org/wiki/Optimized_Link_State_Routing_Protocol; retrieved on 9. September 2016.

[8] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers)," 1994.