CPE 701 – Project Design Report

David Leblanc
Spring 2014

The purpose of this report is to describe the design details of the emulated network project. The specifications for the project are referenced in the project description document. The layers of the network, including the protocols and implementations use at each level will be explained.

I – Overview

At a high level, the virtual network will be emulated locally by a network manager task which will layout the network topology, and handle the file structure and transfer requests. It will also monitor network traffic, and track interesting events that might occur in the process. It will hold and construct endpoint and core nodes with the links between them.

The emulated network will consist of a set of endpoint and core virtual nodes with connections between them. The endpoint nodes are virtual nodes which represent the local running applications. They will have the responsibility to handle connection and file transfer protocols. The requirement for endpoint nodes is that they must preserve the integrity of the files they send and receive. Each endpoint links to a single core node, through a unique port number (local network constraint), and will have a unique 32bit IP address to identify it.

The collection of core nodes essentially represent the core of the emulated network. Core nodes can have many links to other core nodes in the network. These nodes will handle the routing of packets through the network from their originating source to their desired destinations. Core nodes are also responsible for detecting and correcting (if time permits) any errors on packet passing through them.

Each link between nodes will handle the transfer of packets. The connection points contain the local and remote port numbers. Since the network is emulated locally, these port numbers will be unique for each link access point throughout the entire network. The transfer protocol for data sent through the link will be handled using a standard UDP socket. The sockets will listen asynchronously for incoming packets and fire events to notify upper layers (core nodes) upon receiving data.

At the lowest level, the data moved throughout the network will have the form of packets. The packets will have headers framing the data which will provide the information necessary for forwarding or routing. Each packet should hold basic data fields such as:

- Source Address
- Destination Address
- Type of Packet
- Time-to-live
- Header and Data Sizes
- Error Checksum
- Other

At the link layer, the packets will appear as simply bytes of data to be given to the network

nodes. The type of packets, routing or forwarding, error detection or correction, will be determined at the core layer. At the highest level (application), packets will be parsed to determine the data relevant to the end nodes of the network. These could be file transfer data request or acknowledgments, retransmission, or other higher level protocols. *Note: For the sake scope for this project, most of the focus for the design will be on file transfer protocols.*

There will also be a concept of the routing probe. This is basically a packet which travels throughout the network, keeping a history of nodes visited. The history will be used as a way to build the routing tables for the core nodes of the network. This can also be used for debugging purposes to makes sure that packets can travel throughout the network, and allow the user to track the individual routing packets themselves.

The next section describes the implementation of the network simulator using a top-down approach.

II – Design

The network simulator project will be implemented in C# using a WPF (time permitting) GUI interface which will allow the user to load or save networks, and track real-time information regarding the network.

1. User Interface

At the highest level, the user interface will be a WPF app which allows the user to get an overview of the network. The minimum functionality provided will be:

- **Load and Save networks:** The networks will be loaded and saved using XML files, because networks are hierarchical in nature and XML have the nice property of being human readable or writable
- **Initialize/Start networks:** This provides basic start/stop simulation functions to the user.
- **Set simulation parameters:** The parameters of the network will allow the user to set specific settings such as the error/drop rates, and slow down the simulation for observation and/or debugging.
- **Initiate and track file transfers:** There will be a file structure (in the app directory), in which each directory will correspond to a endpoint node. The user will be able to drop files into those directory and transfer select source and destination points to initiate file transfers.
- **Track/View traffic and general information at each individual nodes:** The user shall have the ability to select individual nodes or links anywhere in the network and see a log of the data flowing through it. This will be useful in the debugging process.

2. Network Emulation Layer

The network emulator is the interface which manages the network topography and construction. It will hold all the endpoint nodes and core nodes of the network and initiate and track file transfer progress. This is the layer between the user interface and the implementation of the network itself.

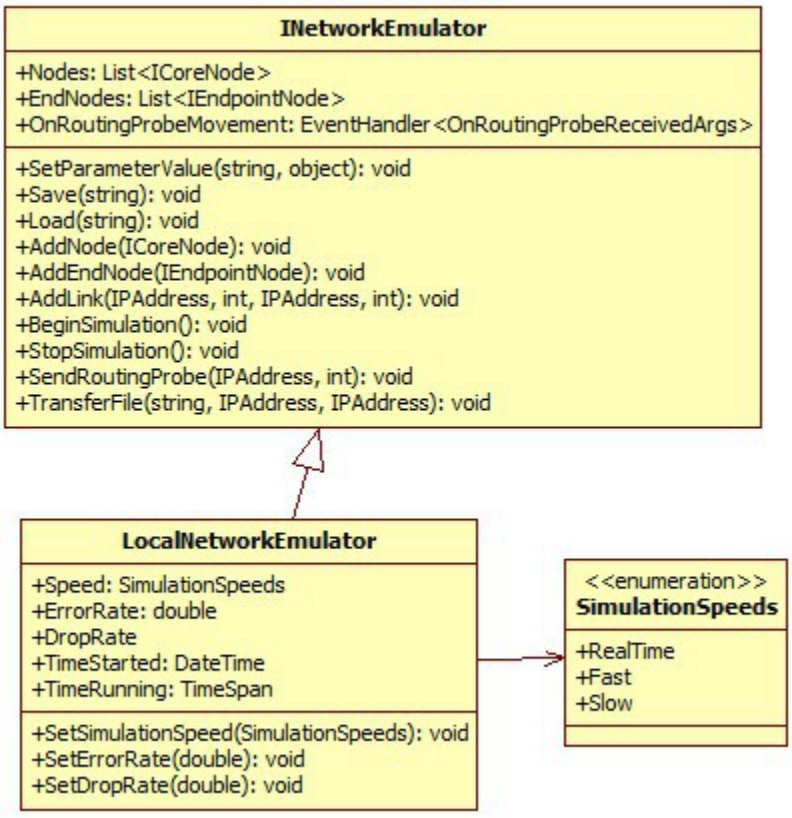

Figure 1: INetworkEmulator

The INetworkEmulator Interface is the interface which provides the functionality of the emulation and the LocalNetworkEmulator is the implementation of the network. The interface includes event handlers which will track progress of file transfers and routing probe movement throughout the network. *Note: More will be added to these interfaces as functionality gets implemented.*

3. Network Nodes Layer

In the Node layer, each node will be managed as an abstract INetworkNode object. The INetworkNode interface splits into two interfaces, ICoreNode and IEndpointNode. Each share similar behavior but also have different individual behaviors. The core nodes can have multiple connections to other nodes in the network. The will contain routing tables which will determine where traffic should go. The IEndpointNode handle the higher level endpoint functionality such as file transfer progress tracking requests and may only connect to a single core network node.

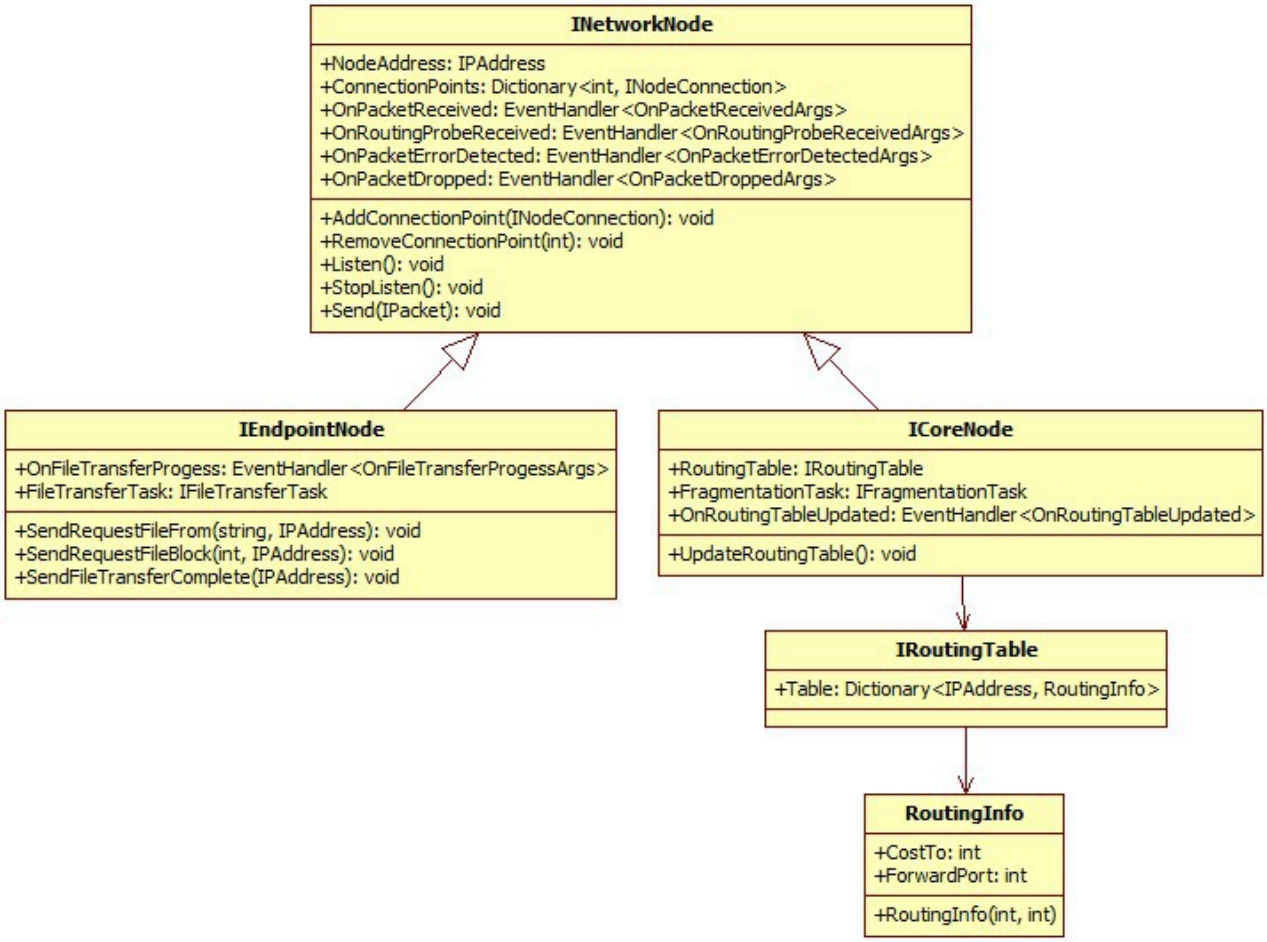

Figure 2: INetworkNode

3.1 INetworkNode

The INetworkNode interface is the base interface for the network node. It contains a globally unique IPAddress which will be used to identify it across the network. It also contains a list (Dictionary with keys corresponding to remote port numbers) of connections or links to other network nodes.

The INetworkNode interface will also include a set of event handlers which will track the arrival of data to trigger forwarding, error detection/correction, or routing table updating. It will also be used to notify the higher layer (the network manager, or user interface) than an event has occurred at the specific node in the network.

It contains methods which allow it initiate or stop listening to incoming data from its connections. It also contains a Send method which enables a outside caller to send a packet from the node in question. *Note: For the most part, the receiving and sending will be handled automatically (hidden to the user) by the node itself, but the 'send' method gives the network manager the ability to manually send their own defined packets.*

Finally, it also contains methods used in topology construction to add or remove virtual links.

3.2 ICoreNode

The ICoreNode interface inherits from the base INetworkNode interface, and provides

additional functionality specific to the core nodes within the network. In addition to its base methods, the core node interface includes a routing table, which will be used to forward packets to their desired destinations. Anytime the routing table gets update, the OnRoutingTableUpdated event will fire, which will be helpful to notify the user interface.

In addition, while passing packets through the nodes, fragmentation may occur, which will be handled by the fragmentation task. The fragmentation task will handle the mechanism which fragments the packets and manage the tracking of reconstruction.

3.2.1 IRoutingTable

The IRoutingTable is the interface which manages the routing tables within core nodes. The routing table is simply a dictionary for which the keys are the IP Addresses of destinations, and the value is simply the routing information. Currently, the routing information will simply have the cost to reach the IP Address, and which port the data should go through to reach the destination.

3.3 IEndpointNode

The IEndpointNode interface is used for endpoint nodes. The Endpoint nodes handle the higher level functionality of file transfers protocols. It includes additional functionality to send and receive files, and includes event handlers which notify when file transfer packets are received and when the file transfer has completed. The actual protocol which defines how files are transferred will be explained later.

The Endpoint nodes only connect to a single other netwwork node, so the AddConnectionPoint method will throw exceptions if an endpoint node attempts to connect to more than a single point.

4. Network Link Layer

The network link layer is the layer where data gets transferred between nodes. The base protocol used to do the transfer is UDP (as required by the project). The link layer is the simplest layer in the virtual network. It's role is to ensure that data can flow from one node to another. The links use UDP sockets at each end of the nodes, and listen for incoming data, which must be done asynchronously to ensure that data is processed through the link in real-time. Upon receiving the data, it will fire the event to notify the network node that a packet has arrived.

4.1 INodeConnection

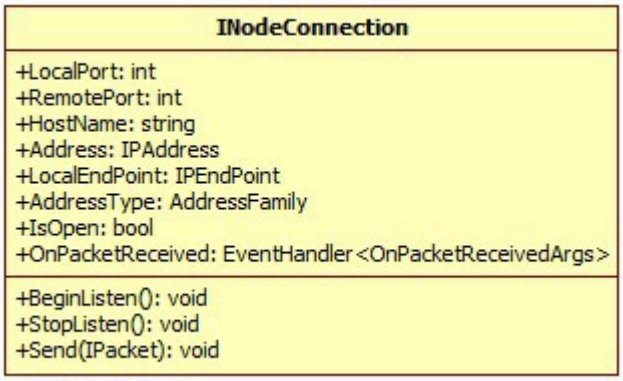

Figure 3: INodeConnection Interface

Another aspect to note for the link layer is that the link itself does not care about the content of the packets it sends or receives, it merely just passes it to and from the network nodes. But, to satisfy the requirements of the project, the links have a certain probability of either dropping a packet or randomly flipping some of the bits causing errors. These will happen upon sending or receiving packets.

5. Network Protocols

The protocols will be designed to perform file transfers from one endpoint node to another. There are many requirements that need to be met to perform the file transfers. The files must be transferred completely and without errors. To ensure these requirements are met, the file at the source must be broken up into multiple packets. The packets must be routed throughout the network and arrive successfully at the destination.

To ensure that the packets arrive at their destination properly, the nodes will use the routing tables to direct the flow of data. These tables can be build manually, or automatically updated. To accomplish automatic updating of the tables, there will be a routing packet which navigates from node to node and keeps a history of nodes visited. Upon reaching a core node, the routing packet will be checked and the routing tables updated if a better path has been found.

Upon arrival to the destination, the packets must be checked for errors. There are two basic ways to deal with errors within packets:

1. Acknowledge the error, drop the packet and request a retransmission of that packet.
2. Keep the packet and correct the error to repair the packet.

Option 1) will most likely be the preferred option for this project, but given enough time, option 2) could be explored. Upon receiving packets (and errors are dealt with), the file can be reconstructed at the destination. Common issues can arise when rebuilding a file, such as duplicate packets or out of order packets.

Rather that doing a back-and-forth request and acknowledgment type of file transfer (commonly used in TCP), the source will simply send packets in order and assume that it reached the destination. Upon receiving said packets, the destination will look at the index of the packet (which block of the file it belongs to), and populate that block in the output file. It will keep track of blocks missed (out of order, dropped, or errors on packets), and after a certain delay, send a request for retransmission.

5.1 IPacket

The IPacket interface will be the base interface for the packets. It provides basic functionality which is needed for the packets. All packets inherit from this base interface.

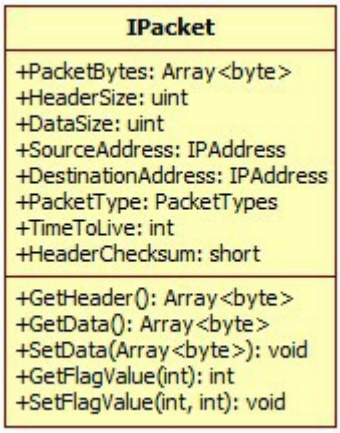

Figure 4: IPacketInterface

5.1.2 BasicPacket

The BasicPacket will be the generic basic IPacket implementation. This is just a default packet that can be used to retrieve the head and data from the packet.

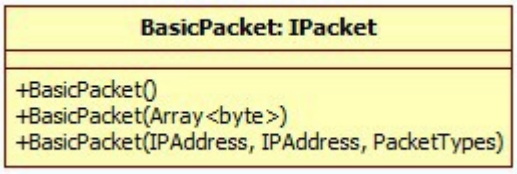

Figure 5: BasicPacket

5.1.3 RoutingProbePacket

The RoutingProbePacket is used to build the routing tables at the core nodes of the network. It inherits from the IPacket interface and adds a history list of the past nodes visited.

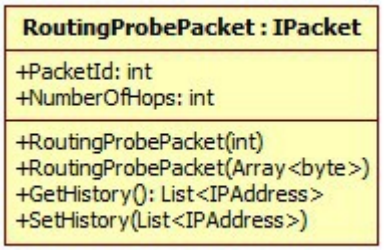

Figure 6: RoutingProbePacket

5.1.4 FileTransferPacket

The FileTransferPacket is used for the file transfer protocol. It inherits from the IPacket interface. It includes extra fields which will allow it to reliably transfer file bytes to its destination. This includes a code which describes the type of file transfer packet it is, and the block section of the file the packet corresponds to.

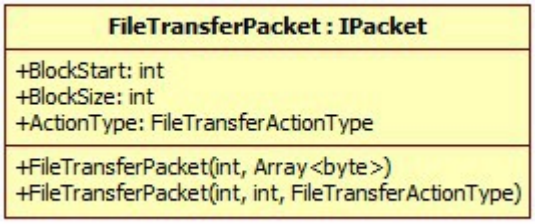

Figure 7: FileTransferPacket

Note: The FileTransferPacket may include more fields or functionality which will be better defined at implementation time.

5.2 Protocol Tasks

There will be a task object which will manage the file transfer tasks. This will keep track of blocks transferred and received, and manage the windows for retransmission of missing blocks. The implementation details of this are still under work.