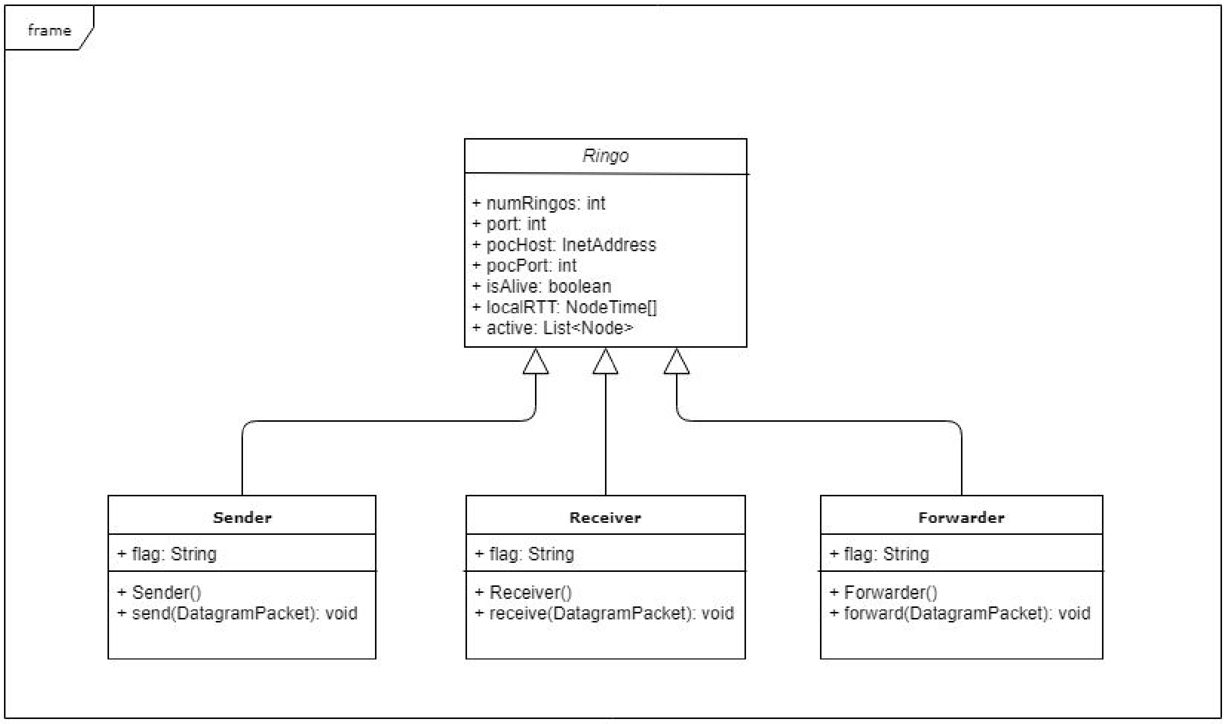
Names: Rudy Lowenstein and Kenneth Scharm
Emails: rlowenstein3@gatech.edu and kscharm3@gatech.edu
CS 3251 - Computer Networks I
16 March 2018
Constantine Dovrolis

Ringo Project: Milestone 2

I. Design Report

Overall Architecture



We plan on implementing the Ringo Protocol by abstracting the idea of a Ringo. We decided on an abstract class because a Ringo by itself cannot exist and does not exist in the system. Every Ringo in the system will have the number of total Ringos, its own port number and IP address, the port number and IP address of it's Point of Contact (PoC), a boolean that represents the alive state of that Ringo, and a local Round-Trip Time (RTT) vector. The main application is called RingoApp, where all communication takes place.

We classify the categories of Ringo, or sub-classes, as the following: Senders, Receivers, and Forwarders. The type of Ringo is implemented asa String that designates that particular Ringo's role. Sender Ringos can send data to other Ringos via the send method.

Forwarder Ringos can forward data from one ringo to the next via the forward method. Receiver Ringos can receive data from a Ringo via its receive method.

There are two classes we use to model Ringo nodes: Node and NodeTime. A Node is created with a String IP address and a port number. A NodeTime is composed of a Node and a long RTT value. We use NodeTime

Packet Header Structures

- Keep-Alive Packet

String Destination IP Address	Int Destination UDP Port Number	String Source IP Address	Int Source UDP Port Number

- Round-Trip-Time Vector

String Destination IP Address	Int Destination UDP Port Number	Round-Trip Time (ms)

- Data Packets

Int Sequence number	Int Destination UDP Port Number	String Destination IP Address	String Source IP Address	Int Source UDP Port Number

- ACK Packets

String Destination IP Address	Int Destination UDP Port Number

Algorithms

Peer Discovery

Each Ringo keeps track of a list of all active Ringos. When a Ringo comes online it adds itself to the list of active Ringos. If it is initialized with a PoC, it sends this active list to it’s PoC, who then adds the new information to its local active Ringo list. As more and more Ringos come online, they continue to exchange information with all Ringos in their active list. This means that whenever a Ringo receives new information, it broadcasts it to all other Ringos. The algorithm converges when the size of every Ringo's active list is equal to the number of Ringos specified in the command line.

Optimal Ring Calculation

To compute an optimal ring, we will be using a brute force approach. We plan on achieving this by testing all permutations of Ringos, and picking the sequence with the minimum sum of RTT time. With brute force, the solution can be computed in O(n!) time. This does not work well when n is large, but n is very small for this assignment, so the brute force approach is actually more efficient than the Held-Karp dynamic programming solution, which has $O(n^2 2^n)$ running time.

Key Data Structures

- Each Ringo has an active list of all known Ringos
- Each Ringo object has a local RTT array of size N, where each entry is a NodeTime representing the destination node and the RTT to reach that node.
- Global RTT map from Node to a NodeTime array of size N, such that when you query the map, you will get the RTT vector from that node to all other nodes
- Global optimal ring structure backed by a circular doubly linked list, where each node points to two other nodes, namely a previous and a next

Thread Architecture

Each Ringo will be on its own thread to send and receive packets. We will be extending Java's Thread class to implement our own threads. Each Ringo has a sending thread AND a receiving thread. Additionally, we will have a main thread for parsing arguments and running the user interface through the command line. This thread is the main thread for any given Ringo. This is where we will implement the reliable data transport protocol over UDP. If a Forwarder goes down during transfer, we will use the alternate path to complete reliable transfer. After transfer is complete, we will use the main thread to recalculate the optimal ring. No separate thread is needed because data transfer will halt until the ring is rebuilt.

II. README

Project Files

RingoApp.java: main Ringo application (**run this**)

Ringo.java: abstract class that defines all the components of a Ringo

Sender.java: subclass of Ringo that defines a Sender

Forwarder.java: subclass of Ringo that defines a Forwarder

Receiver.java: subclass of Ringo that defines a Receiver

README.pdf: includes design report and README

sample.txt: contains sample output from RingoApp

Steps to Compile and Run

i) **Compiling Source Files**

To compile all source files, run the command:

```
javac *.java
```

in the directory where all the source files are stored.

ii) **Running the Application**

To launch any given Ringo, use the command:

```
java RingoApp <flag> <local-port> <PoC-name> <PoC-port> <N>
```

<flag>: S if this Ringo peer is the Sender, R if it is the Receiver, and F if it is a Forwarder
<local-port>: the UDP port number that this Ringo should use (at least for peer discovery)
<PoC-name>: the host-name of the PoC for this Ringo. Set to 0 if this Ringo does not have a PoC.

<PoC-port>: the UDP port number of the PoC for this Ringo. Set to 0 if this Ringo does not have a PoC.
<N>: the total number of Ringos (when they are all active).

Example Ringo launch where PoC is local machine:

```
java RingoApp S 13001 localhost 13002 5
```

Example Ringo launch where PoC is another machine:

```
java RingoApp S 13001 networklab3.cc.gatech.edu 13002 5
```

Known Bugs / Limitations

- Currently, the Ringos do not fully complete the transfer of RTT vectors to create the RTT Matrix
- The show-ring command does not show the optimal ring, but rather a route between all Ringos. This is to show that the network has been fully built during peer discovery.