# Project 3: Overlay Routing

CMSC 417 Spring 2015

Updated: April 2 2015

## 1 Deadline

This project will comprise multiple parts. Each part will have a separate due date.

## 2 Introduction

In this project, you will apply what you have learned throughout the semester to build an overlay circuit-switched traffic routing system that uses Link State routing to pass messages between arbitrary nodes.

Overlay routing is routing that happens at the application layer. The application layer processes on a given node, connect to application layer processes on other nodes to build routing tables and pass messages.

This project is intended to be "open ended" in the sense that we leave many of design decisions for you to make. Your grade will be based not only on your implementation, but on a short writeup, and a presentation to your classmates. We want to see that you possess the ability to not only design and implement a system, but to defend your design decisions. Do not underestimate the difficulty of this project.

## Environment

You will run this project in the CORE network environment. We will provide several network configurations for you to use. The basis of this project, is that you have several groups of nodes which are directly connected. When you load the network configuration, you should notice (and test this), nodes that are directly connected to each can ping each other. However, nodes that are not directly connected to each other cannot ping each other. Why is this?. The short answer is that they don't know how.

## Background

Let's discuss some network routing background (really this should be just a refresher; and will be presented at a high-level). When you connect your computer to your home network, what really happens? The router that you get from your ISP (really, it's a NAT Box), connects to the next hop router.

**EXAMPLE**: Open up a command line and run a traceroute to apps01.mywebapps.net. On windows you can use *tracert apps01.mywebapps.net* and on Unix (Linux or Mac) it should be *traceroute apps01.mywebapps.net*

In the example, you'll see all of the intermediate routers that the packets you sent from your machine to my server took. The first entry is the router (or access point) you are connected to. If you're on a home or residential network, chances are the first hop is just a consumer grade NAT Box (it doesn't really know how to route to much). Basically, all that router knows how to handle is: if traffic is for the local network, router it to that node, otherwise, just send it to the ISP and let the ISP handle it. That's the basics of routing.

Going back to the question in the CORE environment, why can't nodes that aren't directly connected ping each other? The longer answer is that there's not a routing protocol running on the nodes to tell them what to do with the traffic. Nodes basically know the identity of the node on the other end of a connection, but nothing beyond that.

### What you will be doing

Your task is to write an application layer protocol that uses the fact that nodes know how to communicate with their direct neighbors to facilitate communication between arbitrary nodes. Your code will run at the application layer and instances applications running on different nodes will be able to communicate. Put another way, instances of your application will be able to communicate across nodes.

### What you will NOT be doing

You are **NOT** writing a generalized routing protocol to facilitate arbitrary data transfer. That is to say, even after your project is finished, arbitrary nodes will not be able to ping each other using the Unix command "ping." However, they will be able to send "ping messages" to each other through your application.

## 3 Team Project

This project **MUST** be done in teams of at least 2 and no more than 3. There are no exceptions to this rule and we may take the liberty of re-assinging groups to meet this criterion. All students must work on all aspects of the project. We will verify this during the discussion meeting – we will be asking questions about all aspects of the project and all students need to be able to answer them.

## 4 Requirements

1. You must implement this project in Ruby. In this past, this project has been done in C, but the reality is that just getting the C code to work correctly consumes a huge amount of time. The goal of this project is to help you design protocols and design complete distributed system, not to debug C code.

2. You may **NOT** use any specialized libraries. (By this, we mean that we don't want you to pull an entire implementation of Dijkstra's algorithm from the Internet)

3. You **MUST** use CORE for this project.
4. You may **NOT** use RPC (Remote Procedure Calls) for any part of this project.
5. Nodes must communication using only network connections; i.e. they may not communicate via config files, etc.

# 5 Time Table

1. Design Specification: April 3, 2015 (due as PDF, via Submit Server)
2. Part 1: Routing Core: April 19th, 2015 (Due via Submit Server)
3. Part 2: Message Passing: April 28th, 2015 (Due via Submit Server)
4. Part 3: Security Extensions: May 6, 2015 (Due via Submit Server)
5. Final Presentation: May 7 and 12, 2015 (PPT Due via Submit Server May 6th)

## Design Specification

Due to the vastness of this project, we require you to first submit a design specification. The purpose of this specification is not only to demonstrate that you have read and understand the specification, but to help you think about the project at a conceptual level and ensure that you don't make early-on design decisions that will make later parts more difficult. Your design specification should include, at a minimum, the following:

1. Choice of algorithms: (don't simply say "use Dijkstra's algorithm for Link State routing", describe how the algorithm relates to the entire project as a whole)
2. Data Structures: This project requires keeping state for various systems, connections, etc. Your implementation will need ways to keep track of this and efficiently look it up. Decide what state you will need to keep, how long you will need it, and how it will be stored and addressed.
3. Message Formats: How will you format control messages?

Submit this as a PDF document on submit.cs.umd.edu

## Part 1: Routing Tables

You will first need to build routing tables in order for any message passing to occur.

### Link Costs

Each node will read a configuration file that contains the cost for each of its outgoing links. These costs will be part of the input to the routing algorithm. We will provide you with a ruby script (to be run on the host machine, run it in CORE, but not from a virtual node), that periodically changes the costs in the costs files. Nodes will need to periodically read in these files for subsequent rounds of the routing protocol.

### Routing Protocol

You must implement Link State routing as described in class. You are free to format the control messages as you see fit You should specify other parameters of the routing protocol (such as time between rounds), in a global configuration file (read by each node)

## Circuit Routing

Each node should have a path to every other (reachable) node in the network. The routing protocol should give each node the necessary information to route traffic toward an arbitrary destination. You will design a message passing service that builds a circuit between two nodes and passes message streams along this circuit.

### Control Messages

Your will need to design control messages to help build the circuit and carry traffic. **Note**: even though these are application layer messages, they are analogous to IP Layer packets. You will need to decide which fields to include in your control messages. This should be clearly specified in your design document; think carefully about what fields to include. Using the correct fields will make this project substantially easier.

### Circuit Setup

When a node receives a control message corresponding to a message the node must decide how to handle it. If a node is not the destination for the message, the node must decide which interface on which to extend the circuit to build the circuit closer to its destination. Since this is circuit switched, the node should not need to do lookups for data messages, only for circuit setup messages. You are free to pass messages between nodes however you like, but note that you must be able to defend your design choice. Before any message data is relayed, you must build the circuit completely.

## Part 2: Message Passing

Users will interact with your programs on each machine by entering commands to standard in, and reading the results from standard out.

### API

Your application should ultimately act as a server to the outside world. That is, it should listen for connections from other nodes and handle them accordingly. End nodes need to be able to accept messages from the end user (as well as deliver messages to the user) You should use STDIN and STDOUT for this.

### Operation

commands will be given in the form of: *command [args]* Here are some examples of messages:

- SENDMSG 192.168.1.2 "Hello World" : *send the message "Hello World" to the process running on 192.168.1.2*
- PING 192.168.52.12 3 5: *send 3 ping control messages to the process running on 192.168.52.12, separated by 5 seconds*

### Message Types [input]

You must support, at a minimum, the following messages between nodes:

- **SENDMSG [DST] [MSG]** This method will deliver the MSG text to the process running on DST. The sending host should print out "SENDMSG ERROR: HOST UNREACHABLE" if the message is not able to be successfully delivered.
- **PING [DST] [NUMPINGS] [DELAY]** This method will send NUMPINGS ping messages to the DST. There should be a delay of DELAY seconds between pings. The sending node should print information to standard out similar to that of the ping command on Unix (or Windows). For an exmaple open a command prompt (in windows) or a UNIX terminal and type 'ping apps01.mywebapps.net.' If a node does not receive a response within 5 seconds, it should print out "PING ERROR: HOST UNREACHABLE"
- **TRACEROUTE [DST]** This method will perform a traceroute to the DST. The host should print out information similar to a Unix or Windows traceroute.

### Message Types [output]

1. The destination node should print out: RECEIVED MSG SrcIP DATA . Where RECEIVED is just the text string RECEIVED, MSG is just the text string MSG, SrcIP is the IP address of the host that originated the message, and DATA is the string that the user entered.

## Config Parameters

You must support, at a minimum, the following config parameters (these should be read from a file that is specified as a single command line parameter).

- **Maximum Packet Size**
- **Name of Weights File** This is the file that contains the costs of the links between nodes.
- **Routing Update Interval** This specifies how often the routing protocol should run to calculate paths

The following two items were added April 2nd 2015

- **Routing Tables**: This specifies an (absolute) file path where the node will output its current routing tables. The output should be of the form Source,Dest,Cost,NextHop.
- **Dump Interval**: This interval (specified in seconds) indicates how often the node should dump its routing information to a text file.

## Part 3: Security Extensions

For the last part of this project, you will incorporate some security features into your routing protocol. This is very open-ended, but this is a 400 level class, so we expect that you will put a reasonable amount of effort into it. In your design spec, you should include an idea of what you intend to do for this part. The spec is not a final commitment, but should reflect what you actually intent to do.

Here are two ideas to get you started:

1. Adversarial Routing: Link-State routing, as implemented in this project is very vulnerable to attacks (imagine what would happen if a single node started advertising false information... Look into black hole attacks, and devise a way to defend against them.

2. Onion-Routing: TOR is a very popular onion router. Look into how TOR does message encryption, and implement a similar protocol to secure end-to-end message in your application.

## Part 4: Helpful (?) Hints

This section will include information about the provided starter files (the .imn, text files, and Ruby script)

## Deliverables

This section lists deliverables that you need to provide

This section was added April 2nd 2015

### Part 0: Design Spec

- A single PDF document containing the required specification. All group member names should be present on the document.

### Part 1: Routing CORE

- A zip file containing source code that supports all functionality for Part 1 (more details below). It is possible that you will progress beyond the features of part 1 before the due date of part 1. You are free to submit code that supports more functionality than the routing core, just make sure that you are supporting at least the required features.

- Source Code Files: you are free (and encouraged) to divide your project into multiple source files. However, name the main entry point (i.e. the part that starts your program) node.rb

- Shell Script: A shell script that runs a node with default parameters

- README: A text document (.txt, no PDF or Word documents please), that describes your code, how to run it, etc. This is for your benefit so that you can provide the grading TA with instructions on how to properly use your code.

## Clarifications

This section will contains clarification as they arise.

- (4/2/2015) An additional feature is required in the routing core (periodically writing routing tables to output file). *Don't worry about including information about this in your design specification, just support the feature when you get to it. I apologize for adding this requirement, but this project was previously given in one part, and we need a way to grade just the routing protocol.*
- (4/2/2015) Added section for deliverables.