In this lab, you will be implementing an overlay network that uses a distributed routing algorithm and testing it in ONL. The program will be structured using five components, as illustrated at right. The main component (not shown) reads command-line arguments and a configuration file, then starts the other components that do the actual work. The Substrate sends packets over emulated links with specified (and possibly variable) delays. The SrcSnk acts as a user-level application, which sends packets at random to other nodes in the network and echos back packets that it receives from other nodes. The Forwarder forwards packets in a hop-by-hop fashion through the network based on its internal forwarding table. The Router implements the distributed routing algorithm, and modifies its own routing table and the Forwarder’s table, as appropriate. These components all have one or more internal threads and these threads communicate with other components through queues. You will be required to complete the implementation of the Forwarder and Router.

The overlay network you will implement is a simplified IPv4 network. Specifically, the overlay packets will have just four fields, source and destination addresses, a protocol field and a TTL. The protocol field takes on values of 1 or 2, where 1 designates a “data packet” sent by a SrcSnk thread, while 2 designates a routing protocol packet sent by a Router thread. When you test your program, you will be using multiple instances of this program, each running on a separate host and acting as a single node in an overlay network. Each overlay node will have one or more neighbors and will be “connected” to its neighbors by logical overlay links.

You will be implementing this network using ONL hosts. This can be a little confusing as each of your overlay IP packets will be carried within an ONL IP packet with it’s own source and destination address fields. That is, your overlay packets will be carried in the payload portion of UDP packets carried between hosts by the ONL “substrate”. These substrate packets will have addresses of the form 192.168.x.y (as usual, for ONL), while your overlay packets will have addresses of the form 1.a.b.c. Make sure you understand the difference between the overlay IP packets and the substrate packets and their relationship to each other. It’s important to keep this
distinction clearly in mind, as you work on the lab. Note that only the Substrate really needs to
know about the ONL addresses. The other components just work with the overlay addresses.

The Forwarder contains a forwarding table that consists of entries that take the form (prefix,
output link number). When a packet is received from the Substrate, the Forwarder first checks to
see if the packet is addressed to “this” overlay node. If so, the packet is passed on to either the
SrcSink or the Router, based on the value of the protocol field. If the packet is not addressed to
this overlay node, the Forwarder does a lookup in its forwarding table to find the longest
matching prefix for the destination address in the packet. It then sends the packet back to the
Substrate along with the link number of the overlay link that the packet should be sent to. For
simplicity, you may implement the forwarding table as a simple, ArrayList list. When doing a
lookup, just search the entire list to find the longest matching prefix.

The Router in this lab will use a path-vector style protocol, similar to what BGP does. Unlike
BGP, our protocol will be an intra-domain protocol and will include a path cost equal to the
sum of the link costs. The Router thread sends “hello packets” to its neighbors once every second,
and the neighbors echo these packets back. The hello packets contain a timestamp that the
Router uses to measure the round trip delay for the link, and the link cost is then set to half the
round trip delay. The Router uses the exponential weighted moving average method (with a
parameter $\alpha = 0.1$) to smooth out variations in the individual delay measurements. The link costs
should be set based on the “smoothed” delay values. If a Router fails to get a response to three
consecutive hello packets from a given neighbor, it will change the status of that link to “failed”.
It continues to send hello packets to failed links, and restores the link as soon as soon as it gets a
response.

The payloads of routing packets are formatted as ASCII text. A hello packet is simply

```
RPv0
  type: hello
  timestamp: 123.456
```

The reply to a hello packet is

```
RPv0
  type: hello2u
  timestamp: 123.456
```

The timestamp in the hello packet is the time in seconds from an arbitrary starting point at the
sending router. Timestamps are echoed back in the replies to hello packets.

The other type of packet used by the routing protocol is a route advertisement packet. An
example is shown below.

```
RPv0
  type: advert
  pathvec: 1.5.0.0/16 345.678 .346 1.2.0.1 1.2.3.4 1.5.4.3
```

Each path vector starts with an advertised prefix, followed by a timestamp for the vector and
the cost of the path (in seconds). The remainder of the path vector is a list of the IP addresses of
the routers along the path, ending with the router that originated this advertisement (1.5.4.3 in
the example). The timestamp for a path vector represents the time at which the advertising
router first sent this path vector. It is forwarded without change by other routers and used by
routers when deciding how to update their routing table entries.
Each *Router* sends an advertisement for each of its own prefixes to each of its neighbors periodically (every 10 seconds). The path vectors for these advertisements will have a cost of zero and a path consisting only the IP address of the sending *Router*.

When a *Router* receives an advertisement from one of its neighbors, it first checks to see if its own IP address appears in the path vector. If so, it just discards this advertisement. Otherwise, it decides whether or not to update its routing table, based on the contents of the path vector. Before describing the update process, let’s look at the contents of a routing table entry. Each entry contains the following fields:

- *prefix*, *timestamp*, *cost*, *path*, *output link*

The *prefix* is an IP address prefix for the subnet that this route tells us how to reach. The *timestamp* is the timestamp of the most recent advertisement packet that caused an update to this routing table entry. The *cost* is the sum of the link costs in the path to the destination subnet. The *path* is a list of IP address strings defining the path to the destination subnet. The *output link* is the link used to reach the first router on the path.

Now, let’s look at how a received advertisement affects the routing table. There are two main cases to consider:

- First, if the routing table has no entry for the subnet whose prefix is specified in the advertisement, then a new route is added to the table based on the information in the advertisement. The *cost* field of the routing table entry is obtained by adding the cost in the received advertisement to the cost of the link on which the advertisement was received.
- If the routing table already had an entry for the subnet whose prefix is specified in the advertisement, then we may need to modify the existing route. There are several sub-cases to consider.
  - If the advertisement arrived on a link that is currently disabled, then the new advertisement is ignored.
  - Otherwise, if the new advertisement uses the same path as the current routing table entry, then we simply update the timestamp field and the cost of the existing routing table entry, based on the information in the advertisement.
  - Otherwise, the new advertisement defines a new route that uses a different path than the old one. We’ll update the current entry based on this new route if any of the following three conditions is true:
    - the cost of the new route is at least 10% smaller than the cost of the current route
    - the new route is at least 20 seconds newer than the old route (as defined by the timestamp of the current route and the timestamp of the advertisement)
    - the current route uses a link that is disabled

If a new route is added to the routing table, or if the link field of an existing entry is changed, then the *Router* should also change the corresponding entry in the *Forwarder*’s internal table.

To aid in debugging, the contents of the routing table should be printed whenever debugging is enabled and a received advertisement either causes us to add a route to the table or modify the path used by an existing route.
Whenever a received advertisement causes any change at all to the routing table, then that advertisement should be extended and forwarded to all the neighboring routers (with the exception of the router that sent the advertisement to this router). When extending an advertisement a Router does two things. First, it modifies the cost field in the advertisement to include the cost of the link on which it received the advertisement. Second, it adds its own IP address to the front of the path portion of the advertisement.

You will find a partial implementation in your code repository along with configuration files for two different overlay network topologies. As usual, a lab report template with more detailed instructions is provided.