

## Contents

<b>1</b>	<b>Goal</b>	<b>1</b>
1.1	Provided Files . . . . .	1
<b>2</b>	<b>The Network Models</b>	<b>2</b>
<b>3</b>	<b>The Node Model</b>	<b>2</b>
3.1	Router Functionality . . . . .	3
3.2	Data Packet Generators . . . . .	3
<b>4</b>	<b>Your Process Model</b>	<b>3</b>
<b>5</b>	<b>Testing Your Implementation</b>	<b>4</b>
<b>6</b>	<b>Reporting Your Results</b>	<b>5</b>

## 1 Goal

In this assignment, you will implement the Distributed Asynchronous Bellman-Ford routing algorithm, as discussed in class. For simplicity, there will be a single destination node. You are required to demonstrate the correctness your implementation on two network models which have been provided (see accompanying figure).

### 1.1 Provided Files

In the `public/` sub-directory of the class umbrella directory, there is a sub-directory named `lab-bf/` which contains all the models provided. A unix archive (tar file) of this subdirectory can also be downloaded from the class web page.

## 2 The Network Models

There are two network models provided, named `lab-bf-test1` and `lab-bf-test2`. Each model consists of 12 nodes in a random mesh topology, with bi-directional links connecting the nodes. The nodes are named `node_0` through

---

<sup>1</sup>There is a companion document entitled “Notes on OPNET” which should be helpful for completing this assignment.

node\_11, and node\_0 is the destination node. The address of node\_ $i$  is the integer  $i$ , which is indicated by setting the “user id” attribute of node\_ $i$  to the integer  $i$ . Thus, the destination node in each network has address 0. Each link has a transmission rate of 1000 bits/sec. For simplicity, the weights of links in either direction between two nodes are defined to be the same. In particular, the weight of a link between two nodes with addresses  $i$  and  $j$  is defined to be  $i + j$ . Hopefully, this should allow you easily to manually calculate the shortest paths to the destination node by examining the accompanying figure.

**Every node in these two networks, including the destination node in each network, are instances of the *same* node model, which is also provided.**

### 3 The Node Model

The provided node model is named lab-bf, and is illustrated in the accompanying figure. As illustrated, the node model provided consists of 4 receivers, 4 transmitters, a packet generator, a packet sink, and a “router”. The receiver and transmitters are paired together for connection to bi-directional links. Thus, each node can connect to at most four other nodes through bi-directional links. **Your primary task is to construct a process model for the “router” which implements the distributed asynchronous Bellman-Ford algorithm.** The router resides inside a queue module, though it is not actually necessary to use the queuing resources of the module to store packets (rather, packets can be transparently queued in the transmitters). The queue module (named “router”) has 5 input packet streams, with indices 0 to 4, and 5 output packet streams, with indices 0 to 4. Input packet stream  $i$  is driven by receiver  $i$ , and output packet stream  $i$  drives transmitter  $i$ , where  $0 \leq i \leq 3$ . Receiver  $i$  and transmitter  $i$  are paired together as described above. (This simplifies your design problem, since if a packet arrives to the router on input stream  $i$  from an adjacent node, the router can send packets back to that same node on output packet stream  $i$ .) The output packet stream of the packet generator drives input packet stream 4 of the queue module. Output packet stream 4 of the queue module drives the packet sink module. The address of the node is determined by the “user id” attribute of the node. The destination node has address 0.

### 3.1 Router Functionality

The process model that you develop for the queue module should generate control packets (and process such control packets received from neighboring nodes) that implement the distributed Bellman-Ford algorithm. **You will need to define a packet format** within the parameter editor to create these control packets. The process model you develop will also need to process data packets received from the packet generator module. The process model you develop should encapsulate such data packets received from the packet generator and route the encapsulated packets to the appropriate neighboring node along a shortest path to the destination node. Obviously, your process model should also route data packets received from adjacent nodes to the appropriate neighboring node which is on the shortest path to the destination. For the destination node, the process model you develop should de-encapsulate received data packets and send the de-encapsulated packets to the packet sink module. The destination node should also keep a count of the number of data packets it forwards to the sink.

### 3.2 Data Packet Generators

The packet generator module within the node model generates data packets of length 10 bits each, according to a Poisson process with rate one packet/second. The generation of packets does not start until the simulation time is  $t = 2.0$  seconds. This simplification allows the shortest path computations to converge before data packets are generated (in practice, data packets might not travel along shortest paths until the path computations converge).

## 4 Your Process Model

As mentioned above, your main task is to construct a process model that resides in the queue module named “router” within each node. To simplify your task, a partially completed process model is provided for you, called “labBellmanFord”.

The provided process model accomplishes some initialization tasks. In particular, it reads the “user id” attribute, which contains the address of the node which hosts the process, and assigns this to a state variable named `local_address`. The provided code also initializes an array of 4 variables,

called `link_cost`. In particular, `link_cost[i]` is initialized to the link cost of the link which is attached to receiver  $i$ . This is accomplished by identifying the link attached to receiver  $i$ , and reading the “cost” attribute of the link. If there is no link attached to receiver  $i$ , then `link_cost[i]` is assigned the dummy value  $-1.0$ . Thus, your process model can determine if a link is attached to receiver  $i$  by examining the value of `link_cost[i]` (i.e. if it has the value  $-1.0$ , then there is no link attached).

You should modify<sup>2</sup> the provided process model `labBellmanFord`, so that it provides the functionality as described in Section 3.1.

*Even though you need not be concerned with all the details of the provided code in “labBellmanFord”, it is worthwhile to give it some study, as it provides an example of how labelled traces work with the debugger. If you run a simulation using the provided process model verbatim, and run it through the debugger, you can turn on the trace label named “test” (in the debugger, type the command `ltrace test`). The trace label “test” is referred to in the provided code in `labBellmanFord`. When the simulation is run with this trace on (type the command `cont` inside the debugger after turning the “test” trace on), it causes the addresses of each node to be printed out, along with the link costs for each link.*

## 5 Testing Your Implementation

In order to verify your implementation, create an animation of your simulation and view it. (10 simulated seconds should be enough time to see something interesting). You should initially see the nodes exchanging control packets to calculate shortest paths, and then see actual data packets being routed towards the destination node along shortest paths. You can also probe the packet counter at the destination node. Since there are 12 nodes (including the destination) generating packets for the destination at a rate of 1 packet/sec each, you should see about  $12(T - 2)$  packets arrive if you run the simulation for  $T$  seconds (recall no data packets are generated in the first 2 seconds of the simulation), if your implementation is working correctly.

---

<sup>2</sup>If you rename the process model, do not use the name “lab-bf”. Dashes in the filename of a process model confuse the compiler and will result in errors at compile time.

## 6 Reporting Your Results

- For each of the provided network models, animate your simulation, but start your animation at  $t = 2.0$  seconds, so that only data packets are seen. About 10 simulated seconds should be sufficient to see several packets from each node travel to the destination. At the end of your animation, you should see red arrows on all links which are along shortest paths. Capture a bitmap of the final picture of the animation (for each network model), print it out, and include it with your report. Each printout should depict the shortest path tree rooted at the node\_0, if your implementation is working correctly.
- For each network model, run a simulation for  $T = 52$  seconds, and probe the packet counter at the destination node. Display the count versus time in the Analysis Editor, and include printouts of the graph for each network. (In both cases, you should get approximately 600 packets arriving at the destination<sup>3</sup>.)
- Include in your report a thorough documentation of your process model, discussing theory of operation. Include an OPNET generated report of your process model, and a picture of the state diagram of your process model.
- Include a discussion of the problems encountered, debugging techniques that were used, and anything else that you might have learned or observed in completing the task.

---

<sup>3</sup>This includes packets generated at the destination