CSE 473 – Introduction to Computer Networks

Lab 5 Report – 140 Points

Your name: Jiaming Qiu; Richard Wu

Part A. **[20 points]** Paste a copy of the completed source code for the *Forwarder* class below. Highlight your changes by making them **bold** (you may omit sections of the original program that contain no added code). Remember to also place a complete copy in the repository before you make your final commit. *Your* committed version should have no extraneous *print* statements.

```
   /**
    * This is the main thread for the Forwarder object.
    *
    * This forwarder thread grab packets from substrate and send them either
    * to the application level or to the router based on the protocol number
    * in the packets. It also put packets in the substrate if the application
    * level and router have packets to send.
    */
   public void run() {
      now = 0;
      double t0 = System.nanoTime() / sec;

      while (!quit) {
         now = System.nanoTime() / sec - t0;
         // TODO
         // if the Substrate has an incoming packet
         if (sub.incoming()) {
            Pair<Packet, Integer> pp = sub.receive();
            Packet pkt = pp.left;
            // if it's addressed to this overlay router,
            // send to the SrcSnk or the Router
            if (pkt.destAdr == myIp) {
               if (pkt.protocol == 1) {
                  try {
                     toSnk.put(pkt);
                  } catch (Exception e) {
                     System.err.println("Forwarder:run: cannot write" + " to
toSnk " + e);
                     System.exit(1);
                  }
               } else if (pkt.protocol == 2) {
                  try {
                     toRtr.put(pp);
                  } catch (Exception e) {
                     System.err.println("Forwarder:run: cannot write" + " to
toRtr " + e);
                     System.exit(1);
                  }
```

```
            }
         }
         // else
         else {
            pkt.ttl--;
            if (pkt.ttl == 0) {
               if (debug > 0)
                  System.out.println("expired ttl");
            } else {
               // forward it to the next hop
               int lnk = lookup(pkt.destAdr);
               if (lnk != -1 && sub.ready(lnk)) {
                  sub.send(pkt, lnk);
               }
            }
         }
      }
      // else if we have a packet from the Router to send
      else if (fromRtr.size() > 0) {
         // send it to the Substrate
         Pair<Packet, Integer> pp = fromRtr.peek();
         if (sub.ready(pp.right)) {
            sub.send(pp.left, pp.right);
            try {
               fromRtr.take();
            } catch(Exception e) {
               System.err.println("Forwarder:run: fromRtr take exception"
+e);
               System.exit(1);
            }
         }
      }
      // else if we have a payload from the SrcSnk to send
      else if (fromSrc.size() > 0) {
         // lookup the outgoing link using dest IP address
         // format a packet containing the payload and
         // pass it to the Substrate
         Packet pkt = fromSrc.peek();
         int lnk = lookup(pkt.destAdr);
         if (lnk != -1 && sub.ready(lnk)) {
            sub.send(pkt, lnk);
            try {
               fromSrc.take();
            } catch(Exception e) {
               System.err.println("Forwarder:run: fromSrc take exception"
+e);
               System.exit(1);
            }
         }
      }
      // else, nothing to do, so take a nap
      else {
         try {
            Thread.sleep(1);
```

```
            } catch (Exception e) {
               System.out.println("Forwarder:run: " + "can't sleep" + e);
               System.exit(1);
            }
         }
      }
   }

   /**
    * Add a route to the forwarding table.
    *
    * @param nuPrefix is a prefix to be added
    * @param nuLnk    is the number of the link on which to forward packets
    *                 matching the prefix
    *
    *                 If the table already contains a route with the specified
    *                 prefix, the route is updated to use nuLnk. Otherwise, a
route
    *                 is added.
    *
    *                 If debug>0, print the forwarding table when done
    */
   public synchronized void addRoute(Prefix nuPrefix, int nuLnk) {
      // TODO
      // if table contains an entry with the same prefix,
      // just update the link; otherwise add an entry
      boolean update = false;
      Pair<Prefix, Integer> nuRte = new Pair<Prefix, Integer>(nuPrefix,
nuLnk);
      for (int i = 0; i < fwdTbl.size(); i++) {
         Pair<Prefix, Integer> rte = fwdTbl.get(i);
         if (rte.left.equals(nuPrefix)) {
            update = true;
            fwdTbl.set(i, nuRte);
            break;
         }
      }
      if (!update) {
         fwdTbl.add(nuRte);
      }
      if (debug > 0) {
         printTable();
      }
   }


   /**
    * Lookup route in fwding table.
    *
    * @param ip is an integer representing an IP address to lookup
    * @return nextHop link number or -1, if no matching entry.
    */
   private synchronized int lookup(int ip) {
      // TODO lookup function
```

```
    int maxLeng = 0;
    int nextHop = -1;
    for (Pair<Prefix, Integer> rte : fwdTbl) {
       Prefix p = rte.left;
       int num = rte.right;
       if (p.matches(ip) && p.leng >= maxLeng) {
          maxLeng = p.leng;
          nextHop = num;
       }
    }
    return nextHop;
}
```

***Part B*. [30 points]** Paste a copy of the completed source code for the *Router* class below. Highlight your changes by making them **bold** (you may omit sections of the original program that contain no added code). Remember to also place a complete copy in the repository before you make your final commit. *Your* committed version should have no extraneous *print* statements.

```
   /**
    * This is the main thread for the Router object.
    *
    * The router thread sends hello and advertisement packets periodically
    * (1 second and 10 seconds respectively) by calling specific functions.
    * It also retrieves incoming packets and processes it.
    */
   public void run() {
      double t0 = System.nanoTime() / sec;
      now = 0;
      double helloTime, pvSendTime;
      helloTime = pvSendTime = now;
      while (!quit) {
         // TODO
         now = System.nanoTime() / sec - t0;
         double helloDelta = 1;
         double advertDelta = 10;
         // if it's time to send hello packets, do it
         if (now > helloTime + helloDelta && fwdr.ready4pkt()) {
            sendHellos();
            helloTime += helloDelta;
         }
         // else if it's time to send advertisements, do it
         else if (now > pvSendTime + advertDelta && fwdr.ready4pkt()) {
            sendAdverts();
            pvSendTime += advertDelta;
         }
         // else if the forwarder has an incoming packet
         // to be processed, retrieve it and process it
         else if (fwdr.incomingPkt()) {
            handleIncoming();
         }
         // else nothing to do, so take a nap
         else {
            try {
               Thread.sleep(1);
            } catch (Exception e) {
               System.out.println("Router:run: " + "can't sleep" + e);
               System.exit(1);
            }
         }
      }
      String s = String.format("Router link cost statistics\n" + "%8s %8s %8s
%8s %8s\n", "peerIp", "count",
            "avgCost", "minCost", "maxCost");
      for (LinkInfo lnk : lnkVec) {
         if (lnk.count == 0)
```

```
            continue;
         s += String.format("%8s %8d %8.3f %8.3f %8.3f\n",
Util.ip2string(lnk.peerIp), lnk.count,
               lnk.totalCost / lnk.count, lnk.minCost, lnk.maxCost);
      }
      System.out.println(s);
   }



   /**
    * Lookup route in routing table.
    *
    * @param pfx is IP address prefix to be looked up.
    * @return a reference to the Route that matches the prefix or null
    */
   private Route lookupRoute(Prefix pfx) {
      // TODO lookup function
      Route mtRte = null;
      for (Route rte : rteTbl) {
         if (rte.pfx.equals(pfx)) {
            mtRte = rte;
            break;
         }
      }
      return mtRte;
   }



   /**
    * Add a route to the routing table.
    *
    * @param rte is a route to be added to the table; no check is done to make
sure
    *            this route does not conflict with an existing route
    */
   private void addRoute(Route rte) {
      // TODO add route
      rteTbl.add(rte);
   }



   /**
    * Update a route in the routing table.
    *
    * @param rte   is a reference to a route in the routing table.
    * @param nuRte is a reference to a new route that has the same prefix as
rte
    * @return true if rte is modified, else false
    *
    *         This method replaces certain fields in rte with fields in nuRte.
    *         Specifically,
```

```
    *
    *         if nuRte has a link field that refers to a disabled link, ignore
it
    *         and return false
    *
    *         else, if the route is invalid, then update the route and return
true,
    *
    *         else, if both routes have the same path and link, then the
timestamp
    *         and cost fields of rte are updated
    *
    *         else, if nuRte has a cost that is less than .9 times the cost of
rte,
    *         then all fields in rte except the prefix fields are replaced
with the
    *         corresponding fields in nuRte
    *
    *         else, if nuRte is at least 20 seconds newer than rte (as
indicated by
    *         their timestamps), then all fields of rte except the prefix
fields
    *         are replaced
    *
    *         else, if the link field for rte refers to a link that is
currently
    *         disabled, replace all fields in rte but the prefix fields
    */
   private boolean updateRoute(Route rte, Route nuRte) {
      // TODO update route
      if (lnkVec.get(nuRte.outLink).helloState == 0) {
         return false;
      }
      else if (!rte.valid) {
         rte.cost = nuRte.cost;
         rte.outLink = nuRte.outLink;
         rte.path = nuRte.path;
         rte.timestamp = nuRte.timestamp;
         rte.valid = true;
         return true;
      }
      else if (rte.path.equals(nuRte.path)) {
         rte.cost = nuRte.cost;
         rte.timestamp = nuRte.timestamp;
         return true;
      }
      else if (nuRte.cost < 0.9*rte.cost
            || nuRte.timestamp >= rte.timestamp + 20
            || lnkVec.get(rte.outLink).helloState == 0) {
         rte.cost = nuRte.cost;
         rte.outLink = nuRte.outLink;
         rte.path = nuRte.path;
         rte.timestamp = nuRte.timestamp;
         rte.valid = true;
```

```
            return true;
         }
         return false;
      }



      /**
       * Send hello packet to all neighbors.
       *
       * First check for replies. If no reply received on some link, update the
link
       * status by subtracting 1. If that makes it 0, the link is considered
down, so
       * we mark all routes using that link as invalid. Also, if certain routes
are
       * marked as invalid, we will need to print the table if debug larger than
1,
       * and we need to send failure advertisement by calling sendFailureAdvert
if
       * failure advertisement is enable.
       */
      public void sendHellos() {
         int lnk = 0;
         for (LinkInfo lnkInfo : lnkVec) {
            //TODO
            boolean update = false;
            // if no reply to the last hello, subtract 1 from
            // link status if it's not already 0
            if (!lnkInfo.gotReply && lnkInfo.helloState != 0) {
               lnkInfo.helloState--;
            }
            if (lnkInfo.helloState == 0) {
            // go through the routes to check routes
            // that contain the failed link
               for (Route rte : rteTbl) {
                  if (rte.outLink == lnk && rte.valid) {
                     rte.valid = false;
                     update = true;
                  }
               }
            }
            if (update) {
            // print routing table if debug is enabled
            // and valid field of route is changed
               if (debug > 1) {
                  printTable();
               }
            // send link failure advertisement if enFA is enabled
            // and valid field of route is changed
               if (enFA) {
                  sendFailureAdvert(lnk);
               }
            }
```

```
            // send new hello, after setting gotReply to false
            lnkInfo.gotReply = false;
            Packet p = new Packet();
            p.protocol = 2; p.ttl = 100;
            p.srcAdr = myIp;
            p.destAdr = lnkInfo.peerIp;
            p.payload = String.format("RPv0\ntype: hello\n"
               + "timestamp: %.3f\n", now);
            fwdr.sendPkt(p,lnk);
            lnk++;
         }
      }

      /** Send initial path vector to each of our neighbors. */
      public void sendAdverts() {
         // TODO send advertisement
         for (Prefix pfx : pfxList) {
            for (int lnk = 0; lnk < nborList.size(); lnk++) {
               if (lnkVec.get(lnk).helloState == 0)
                  continue;
               Packet p = new Packet();
               p.protocol = 2;
               p.ttl = 100;
               p.srcAdr = myIp;
               p.destAdr = lnkVec.get(lnk).peerIp;
               p.payload = String.format("RPv0\ntype: advert\n" +
               "pathvec: %s %.3f %.3f %s\n", pfx.toString(), now, 0.0,
               myIpString);
               fwdr.sendPkt(p, lnk);
            }
         }
      }



      /**
       * Retrieve and process packet received from Forwarder.
       *
       * For hello packets, we simply echo them back. For replies to our own
hello
       * packets, we update costs. For advertisements, we update routing state
and
       * propagate as appropriate.
       */
      public void handleIncoming() {
         // parse the packet payload
         Pair<Packet,Integer> pp = fwdr.receivePkt();
         Packet p = pp.left; int lnk = pp.right;

         String[] lines = p.payload.split("\n");
         if (!lines[0].equals("RPv0")) return;

         String[] chunks = lines[1].split(":");
         if (!chunks[0].equals("type")) return;
```

```
      String type = chunks[1].trim();

      // TODO
      // if it's an route advert, call handleAdvert
      if (type.equals("advert")) {
         handleAdvert(lines, lnk);
      }
      // if it's a link failure advert, call handleFailureAdvert
      else if (type.equals("fadvert")) {
         handleFailureAdvert(lines, lnk);
      }
      // if it's a hello, echo it back
      else if (type.equals("hello")) {
         p.srcAdr = myIp;
         p.destAdr = lnkVec.get(lnk).peerIp;
         p.ttl--;
         p.payload = String.format("RPv0\ntype: hello2u\n"
               + lines[2]);
         if (fwdr.ready4pkt()) {
            fwdr.sendPkt(p,lnk);
         }
      }
      // else it's a reply to a hello packet
      else if (type.equals("hello2u")) {
      // use timestamp to determine round-trip delay
         String[] message = lines[2].split(":");
         if (!message[0].equals("timestamp")) return;
         double timestamp = Double.parseDouble(message[1].trim());
         double delay = (now - timestamp)/2;
      // use this to update the link cost using exponential
      // weighted moving average method
         LinkInfo linkInfo = lnkVec.get(lnk);
         final double alpha = 0.1;
         linkInfo.cost = (1-alpha)*linkInfo.cost + alpha*delay;
      // also, update link cost statistics
         linkInfo.count++;
         linkInfo.totalCost += delay;
         linkInfo.minCost = Math.min(linkInfo.minCost,delay);
         linkInfo.maxCost = Math.max(linkInfo.maxCost,delay);
      // also, set gotReply to true
         linkInfo.gotReply = true;
         linkInfo.helloState = 3;
      }
   }


   /**
    * Handle an advertisement received from another router.
    *
    * @param lines is a list of lines that defines the packet; the first two
lines
    *              have already been processed at this point
    *
    * @param lnk   is the number of link on which the packet was received
```

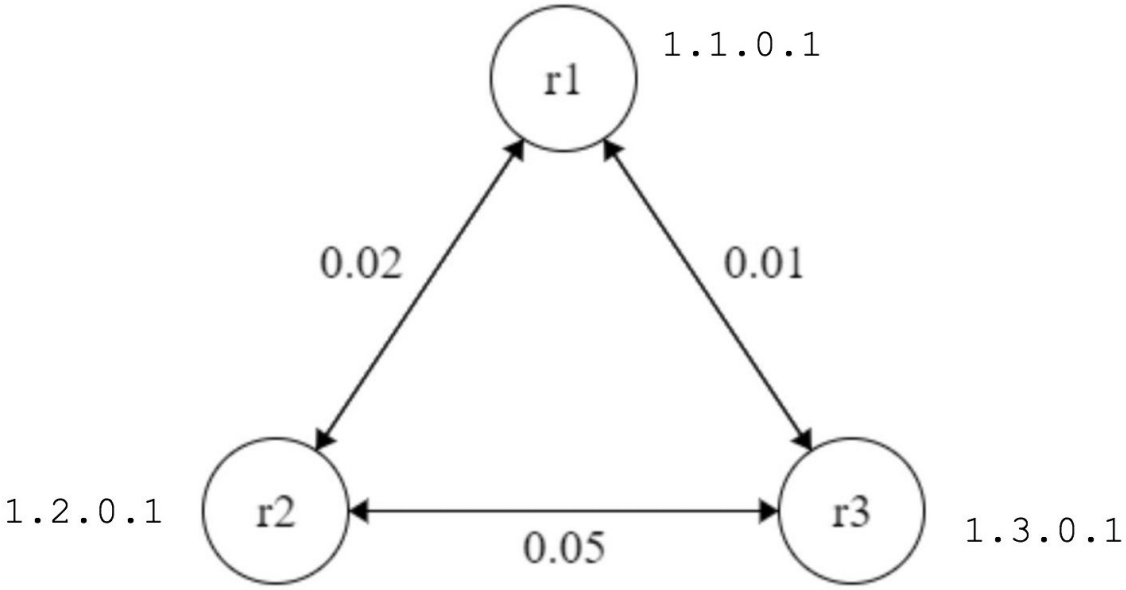
```
 */
private void handleAdvert(String[] lines, int lnk) {
   // example path vector line
   // pathvec: 1.2.0.0/16 345.678 .052 1.2.0.1 1.2.3.4
        //
   // TODO
     // Parse the path vector line.
   String[] message = lines[2].split(":");
   if (!message[0].equals("pathvec")) return;
   String[] info = message[1].trim().split(" ");
   if (info.length < 4) return;
   Prefix pfx = new Prefix(info[0].trim());
   double timestamp = Double.parseDouble(info[1].trim());
   double cost = Double.parseDouble(info[2].trim());
   LinkedList<Integer> path = new LinkedList<Integer>();
   for (int i = 3; i < info.length; i++) {
      int rteIp = Util.string2ip(info[i].trim());
      path.addLast(rteIp);
   }
     // If there is loop in path vector, ignore this packet.
   if (path.contains(myIp)) return;
     // Form a new route, with cost equal to path vector cost
     // plus the cost of the link on which it arrived.
   Route nuRte = new Route();
   nuRte.pfx = pfx;
   nuRte.timestamp = timestamp;
   nuRte.cost = cost + lnkVec.get(lnk).cost;
   nuRte.path = path;
   nuRte.outLink = lnk;
   nuRte.valid = true;
     // Look for a matching route in the routing table
     // and update as appropriate; whenever an update
     // changes the path, print the table if debug>0;
   boolean addNew = false;
   boolean update = false;
   int oldLnk = -1;
   Route rte = lookupRoute(pfx);
   if (rte == null) {
      addRoute(nuRte);
      addNew = true;
   }
   else {
      oldLnk = rte.outLink;
      update = updateRoute(rte, nuRte);
   }
   if (addNew || update) {
     // whenever an update changes the output link,
     // update the forwarding table as well.
      if (oldLnk != lnk) {
         fwdr.addRoute(pfx, lnk);
      }
     // If the new route changed the routing table,
     // extend the path vector and send it to other neighbors.
      String newPath = myIpString;
```

```
         for (int rteIp : path) {
            newPath += " " + Util.ip2string(rteIp);
         }
         for (int lnkNum = 0; lnkNum < nborList.size(); lnkNum++) {
            if (lnk == lnkNum
                  || lnkVec.get(lnkNum).helloState == 0) continue;
            Packet p = new Packet();
            p.protocol = 2; p.ttl = 100;
            p.srcAdr = myIp;
            p.destAdr = lnkVec.get(lnkNum).peerIp;
            p.payload = String.format("RPv0\ntype: advert\n"
               + "pathvec: %s %.3f %.3f %s\n",
               pfx.toString(), timestamp, nuRte.cost, newPath);
            if (fwdr.ready4pkt()) {
               fwdr.sendPkt(p,lnkNum);
            }
         }
         if (debug > 0) {
            printTable();
         }
      }
   }
   /**
    * Handle the failure advertisement received from another router.
    *
    * @param lines is a list of lines that defines the packet; the first two
lines
    *              have already been processed at this point
    *
    * @param lnk   is the number of link on which the packet was received
    */
   private void handleFailureAdvert(String[] lines, int lnk) {
      // example path vector line
      // fadvert: 1.2.0.1 1.3.0.1 345.678 1.4.0.1 1.2.0.1
      // meaning link 1.2.0.1 to 1.3.0.1 is failed
           //
      // TODO
        // Parse the path vector line.
      String[] message = lines[2].split(":");
      if (!message[0].equals("linkfail")) return;
      String[] info = message[1].trim().split(" ");
      if (info.length < 4) return;
      int failFrom = Util.string2ip(info[0].trim());
      int failTo = Util.string2ip(info[1].trim());
      double timestamp = Double.parseDouble(info[2].trim());
      LinkedList<Integer> path = new LinkedList<Integer>();
      for (int i = 3; i < info.length; i++) {
         int rteIp = Util.string2ip(info[i].trim());
         path.addLast(rteIp);
      }
        // If there is loop in path vector, ignore this packet.
      if (path.contains(myIp)) return;
        // go through routes to check if it contains the link
      // set the route as invalid (false) if it does
```

```
      // update the time stamp if route is changed
      boolean change = false;
      for (Route rte : rteTbl) {
         LinkedList<Integer> rtePath = rte.path;
         for (int i = 0; i < rtePath.size()-1; i++) {
            if (rtePath.get(i) == failFrom
                  && rtePath.get(i+1) == failTo) {
               rte.valid = false;
               rte.timestamp = timestamp;
               change = true;
               break;
            }
         }
      }
      if (change) {
      // print route table if route is changed and debug is enabled
         if (debug > 0) {
            printTable();
         }
      // If one route is changed, extend the message
      // and send it to other neighbors.
         String newPath = myIpString;
         for (int rteIp : path) {
            newPath += " " + Util.ip2string(rteIp);
         }
         for (int lnkNum = 0; lnkNum < nborList.size(); lnkNum++) {
            if (lnk == lnkNum
                  || lnkVec.get(lnkNum).helloState == 0) continue;
            Packet p = new Packet();
            p.protocol = 2; p.ttl = 100;
            p.srcAdr = myIp;
            p.destAdr = lnkVec.get(lnkNum).peerIp;
            p.payload = String.format("RPv0\ntype: fadvert\n"
                  + "linkfail: %s %s %.3f %s\n",
                  Util.ip2string(failFrom), Util.ip2string(failTo), timestamp,
newPath);
            if (fwdr.ready4pkt()) {
               fwdr.sendPkt(p,lnkNum);
            }
         }
      }
   }
```

Part C. **[20 points]** Put your files for this lab in the directory `~/473/lab5`. In this part, you will be running some tests using the configuration and script you will find in the *net1* sub-directory. Commit all log files to your repository after finishing this part.

1. (5 points) Draw a diagram showing the logical links joining the three routers in the overlay network defined by the configuration files *r1*, *r2* and *r3* in the *net1* sub-directory. Label the inter-router links with their assigned link costs.



2. (10 points) Run *script1* in the *net1* sub-directory by typing

```
./script1 1 .333 20 static
```

Paste a copy of the output below.

```
jim@onlusr:~/cse_473/lab5/net1$ ./script1 1 .333 20 static
delta= .333  runlength= 20 static
*********** log  1  *************
Final Report

Routing table (28.572)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.3.0.0/16     20.002    0.012     1      valid 1.3.0.1
1.2.0.0/16     20.002    0.022     0      valid 1.2.0.1

Forwarding table (28.578)
0.0.0.0/0 0
1.3.0.0/16 1
1.2.0.0/16 0

Router link cost statistics
```

```
  peerIp    count  avgCost  minCost  maxCost
 1.2.0.1       26    0.022    0.021    0.047
 1.3.0.1       26    0.013    0.011    0.037

SrcSnk statistics
  destIp    count avgDelay minDelay maxDelay
 1.2.0.1       24    0.029    0.021    0.044
 1.3.0.1       37    0.023    0.011    0.044

********** log  2  *************
Final Report

Routing table (28.232)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.1.0.0/16     20.002    0.022     1      valid 1.1.0.1
1.3.0.0/16     20.002    0.034     1      valid 1.1.0.1 1.3.0.1

Forwarding table (28.232)
0.0.0.0/0 0
1.1.0.0/16 1
1.3.0.0/16 1

Router link cost statistics
  peerIp    count  avgCost  minCost  maxCost
 1.3.0.1       25    0.052    0.050    0.061
 1.1.0.1       25    0.023    0.021    0.031

SrcSnk statistics
  destIp    count avgDelay minDelay maxDelay
 1.1.0.1       28    0.030    0.021    0.043
 1.3.0.1       33    0.036    0.032    0.043

********** log  3  *************
Final Report

Routing table (28.309)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.1.0.0/16     20.002    0.012     0      valid 1.1.0.1
1.2.0.0/16     20.002    0.034     0      valid 1.1.0.1 1.2.0.1

Forwarding table (28.309)
0.0.0.0/0 0
1.1.0.0/16 0
1.2.0.0/16 0

Router link cost statistics
  peerIp    count  avgCost  minCost  maxCost
 1.1.0.1       26    0.012    0.011    0.021
 1.2.0.1       26    0.053    0.051    0.059

SrcSnk statistics
  destIp    count avgDelay minDelay maxDelay
 1.1.0.1       37    0.023    0.011    0.044
 1.2.0.1       24    0.036    0.031    0.043
```

For each pair of routers r_i and r_j, write down the shortest path from r_i to r_j and the total cost of that path. Verify that the final routing tables and forwarding tables printed by *script1* are consistent with these shortest paths.

For r1, r2, the shortest path is r1 -> r2, with the total cost of 0.02.

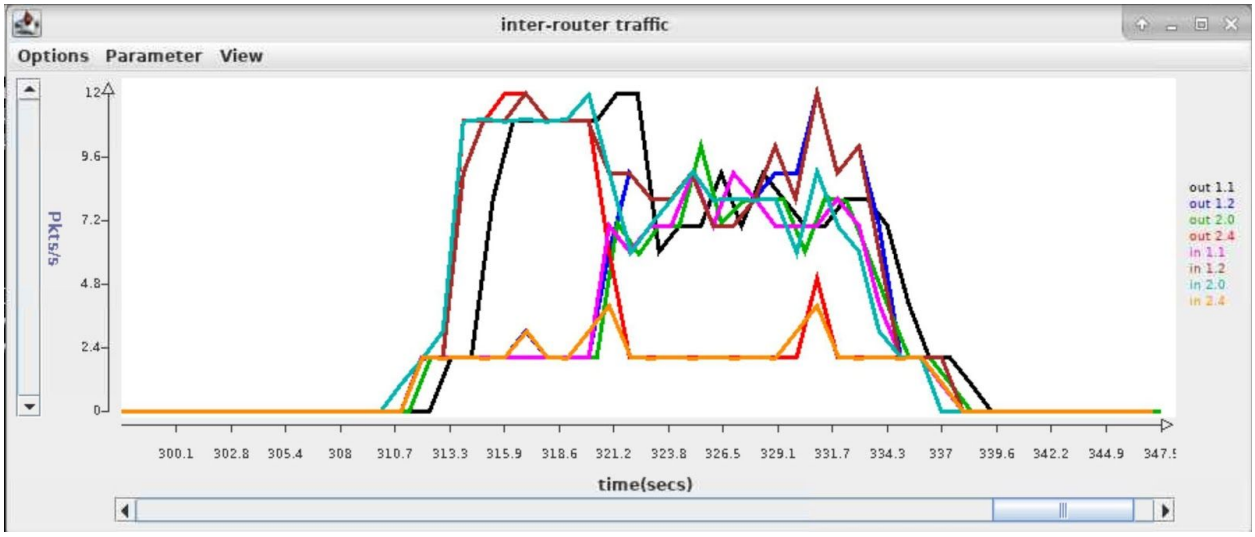
For r1, r3, the shortest path is r1 -> r3, with the total cost of 0.01

For r2, r3, the shortest path is r2->r1->r3, with the total cost of 0.02+0.01=0.03.

Since the costs of links are identical in both directions, the shortest paths in reverse order will be the shortest paths for each of the reversed pair, with the same amount of total cost.

The paths recorded in each of the routing table are exactly the shortest paths we found for each pair of routers. The records in forwarding tables are also consistent with these results because they use the next hop of each of these paths as the link to forward packets.

Paste a screenshot of the monitoring window from your *script1* run below.



Note how the bandwidth on some links changes part way through the run. Explain why this happens. How are packets routed during the first few seconds of the run? Why does this happen?

Before t=10 seconds, there is no advertisement happening. Therefore, the routing table in the router is empty and the forwarding table only contains 0.0.0.0/0 0, which means that every application level packet will be sent to link 0. Hence, every packet in r1 will be sent to r2, i.e. out1.1 and in2.0; every packet in r2 will be sent to r3, i.e. out2.4; every packet in r3 will be sent to r1, i.e. in1.2. The rest of the bandwidth is because of the hello packets and hello2u packets. After t = 10, all the shortest path get updated in every router, thus traffic get redistributed.

How many packets per second should be sent on the link from *onl* router port 1.1 to *onl* router port 2.0 during the first part of the run? Your answer should include all packets sent by the routing algorithm and all packets sent by the *SrcSnk* that would travel over this link. Explain your answer. Does your answer match the observed packet rates?

11 packets.

1 for hello (from r1 to r2); 1 for hello2u (reply r2's hello message)

3 for ping packets (r1 to r2 or r3); 1.5 (on average) for packets from r3 to r1 to r2

3 for ping reply (r1 back to r2 or r3); 1.5 (on average) for ping reply packets from r3 to r1 to r2

The total number of packets should be around 11 (Srcsnk generates packets to random destinations, so 11 is not exact)

Our answer matches the observed packet rates in the graph.

How many packets per second should be sent on the link from the *onl* router port 1.1 to *onl* router port 2.0 during the second part of the run? Explain your answer. Does your answer match the observed packet rates

8 packets.

1 for hello (from r1 to r2); 1 for hello2u (reply r2's hello message)

1.5 (on average) for packets from r1 to r2; 1.5 (on average) for packets from r3 to r1 to r2

1.5 (on average) for ping reply packets from r1 to r2; 1.5 (on average) for ping reply packets from r3 to r1 to r2

The total number of packets should be around 8 (Srcsnk generates packets to random destinations, so 8 is not exact)

Our answer matches the observed packet rates in the graph.

3. (5 points) Run *script1* again by typing

```
./script1 2 .333 20 static debugg
```

and paste a copy of the resulting *log2_2* file below. Add comments to the output in bold to explain how the advertisements trigger changes in the routing table. Also, explain why some received advertisements do not trigger changes to the routing table.

```
/192.168.7.1:31313 sending to /192.168.4.2:31313 at 10.023
protocol=2 ttl=100 srcAdr=1.2.0.1 destAdr=1.1.0.1
RPv0
type: advert
pathvec: 1.2.0.0/16 10.001 0.000 1.2.0.1

// add a new route to both the routing and forwarding table after a new prefix
is received (from r1)
/192.168.7.1:31313 received from /192.168.4.2:31313 at 10.033
protocol=2 ttl=100 srcAdr=1.1.0.1 destAdr=1.2.0.1
RPv0
type: advert
pathvec: 1.1.0.0/16 10.001 0.000 1.1.0.1

/192.168.7.1:31313 sending to /192.168.2.4:31313 at 10.052
protocol=2 ttl=100 srcAdr=1.2.0.1 destAdr=1.3.0.1
RPv0
type: advert
pathvec: 1.2.0.0/16 10.001 0.000 1.2.0.1

// add a new route to both the routing and forwarding table after a new prefix
is received (from r3)
/192.168.7.1:31313 received from /192.168.2.4:31313 at 10.051
protocol=2 ttl=100 srcAdr=1.3.0.1 destAdr=1.2.0.1
RPv0
type: advert
pathvec: 1.3.0.0/16 10.001 0.000 1.3.0.1

Forwarding table (10.047)
0.0.0.0/0 0
1.1.0.0/16 1

Routing table (10.041)
    prefix  timestamp     cost  link  VLD/INVLD  path
1.1.0.0/16     10.001    0.022     1      valid 1.1.0.1

// update an existing route in both the routing and forwarding table after a
shorter path to the same prefix is introduced (by r1)
/192.168.7.1:31313 received from /192.168.4.2:31313 at 10.058
protocol=2 ttl=100 srcAdr=1.1.0.1 destAdr=1.2.0.1
RPv0
type: advert
pathvec: 1.3.0.0/16 10.001 0.012 1.1.0.1 1.3.0.1

Forwarding table (10.066)
0.0.0.0/0 0
```

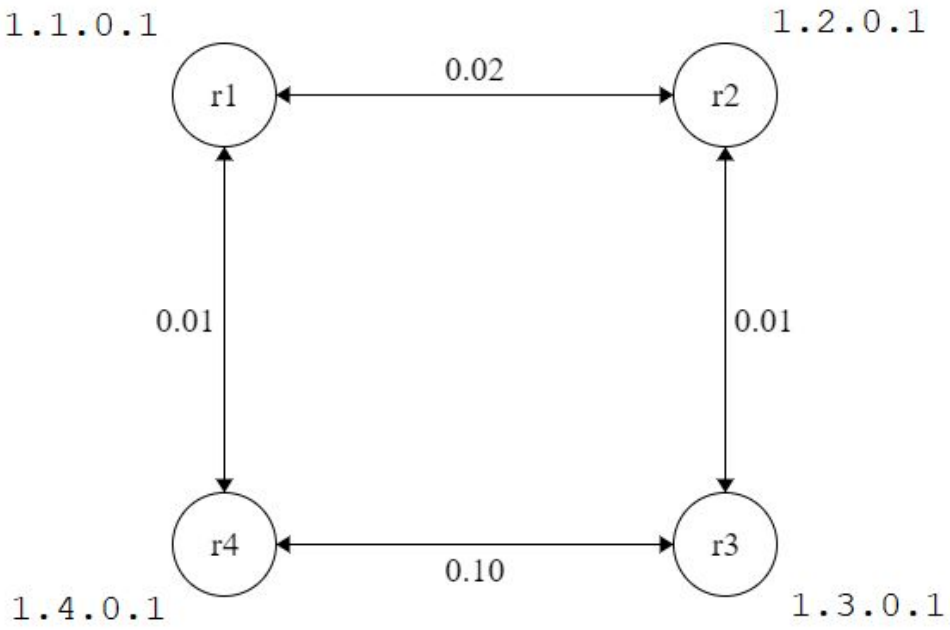
```
1.1.0.0/16 1
1.3.0.0/16 0

Routing table (10.066)
    prefix  timestamp     cost  link  VLD/INVLD  path
1.1.0.0/16     10.001    0.022     1      valid 1.1.0.1
1.3.0.0/16     10.001    0.052     0      valid 1.3.0.1

Forwarding table (10.081)
0.0.0.0/0 0
1.1.0.0/16 1
1.3.0.0/16 1

/192.168.7.1:31313 sending to /192.168.4.2:31313 at 10.096
protocol=2 ttl=100 srcAdr=1.2.0.1 destAdr=1.1.0.1
RPv0
type: advert
pathvec: 1.3.0.0/16 10.001 0.052 1.2.0.1 1.3.0.1

Routing table (10.081)
    prefix  timestamp     cost  link  VLD/INVLD  path
1.1.0.0/16     10.001    0.022     1      valid 1.1.0.1
1.3.0.0/16     10.001    0.034     1      valid 1.1.0.1 1.3.0.1

// cause no change to the routing table because of higher cost to an existing
route using a different path
/192.168.7.1:31313 received from /192.168.2.4:31313 at 10.082
protocol=2 ttl=100 srcAdr=1.3.0.1 destAdr=1.2.0.1
RPv0
type: advert
pathvec: 1.1.0.0/16 10.001 0.012 1.3.0.1 1.1.0.1

/192.168.7.1:31313 sending to /192.168.2.4:31313 at 10.113
protocol=2 ttl=100 srcAdr=1.2.0.1 destAdr=1.3.0.1
RPv0
type: advert
pathvec: 1.1.0.0/16 10.001 0.022 1.2.0.1 1.1.0.1

// discard the packet because a loop is detected
/192.168.7.1:31313 received from /192.168.2.4:31313 at 10.118
protocol=2 ttl=100 srcAdr=1.3.0.1 destAdr=1.2.0.1
RPv0
type: advert
pathvec: 1.2.0.0/16 10.001 0.036 1.3.0.1 1.1.0.1 1.2.0.1

/192.168.7.1:31313 sending to /192.168.2.4:31313 at 10.144
protocol=2 ttl=100 srcAdr=1.2.0.1 destAdr=1.3.0.1
RPv0
type: advert
pathvec: 1.3.0.0/16 10.001 0.034 1.2.0.1 1.1.0.1 1.3.0.1

// update timestamp and cost of an existing route using the same path
/192.168.7.1:31313 received from /192.168.4.2:31313 at 20.022
protocol=2 ttl=100 srcAdr=1.1.0.1 destAdr=1.2.0.1
```

```
RPv0
type: advert
pathvec: 1.1.0.0/16 20.001 0.000 1.1.0.1

/192.168.7.1:31313 sending to /192.168.4.2:31313 at 20.031
protocol=2 ttl=100 srcAdr=1.2.0.1 destAdr=1.1.0.1
RPv0
type: advert
pathvec: 1.2.0.0/16 20.002 0.000 1.2.0.1

Routing table (20.028)
    prefix  timestamp     cost  link  VLD/INVLD  path
1.1.0.0/16     20.001    0.023     1      valid 1.1.0.1
1.3.0.0/16     10.001    0.034     1      valid 1.1.0.1 1.3.0.1

// update timestamp and cost of an existing route using the same path
/192.168.7.1:31313 received from /192.168.4.2:31313 at 20.033
protocol=2 ttl=100 srcAdr=1.1.0.1 destAdr=1.2.0.1
RPv0
type: advert
pathvec: 1.3.0.0/16 20.001 0.013 1.1.0.1 1.3.0.1

Routing table (20.045)
    prefix  timestamp     cost  link  VLD/INVLD  path
1.1.0.0/16     20.001    0.023     1      valid 1.1.0.1
1.3.0.0/16     20.001    0.036     1      valid 1.1.0.1 1.3.0.1

/192.168.7.1:31313 sending to /192.168.2.4:31313 at 20.053
protocol=2 ttl=100 srcAdr=1.2.0.1 destAdr=1.3.0.1
RPv0
type: advert
pathvec: 1.2.0.0/16 20.002 0.000 1.2.0.1

// cause no change to the routing table because of higher cost to an existing
route using a different path
/192.168.7.1:31313 received from /192.168.2.4:31313 at 20.053
protocol=2 ttl=100 srcAdr=1.3.0.1 destAdr=1.2.0.1
RPv0
type: advert
pathvec: 1.3.0.0/16 20.001 0.000 1.3.0.1

// cause no change to the routing table because of higher cost to an existing
route using a different path
/192.168.7.1:31313 received from /192.168.2.4:31313 at 20.061
protocol=2 ttl=100 srcAdr=1.3.0.1 destAdr=1.2.0.1
RPv0
type: advert
pathvec: 1.1.0.0/16 20.001 0.013 1.3.0.1 1.1.0.1

/192.168.7.1:31313 sending to /192.168.2.4:31313 at 20.080
protocol=2 ttl=100 srcAdr=1.2.0.1 destAdr=1.3.0.1
RPv0
type: advert
pathvec: 1.1.0.0/16 20.001 0.023 1.2.0.1 1.1.0.1
```

```
/192.168.7.1:31313 sending to /192.168.2.4:31313 at 20.097
protocol=2 ttl=100 srcAdr=1.2.0.1 destAdr=1.3.0.1
RPv0
type: advert
pathvec: 1.3.0.0/16 20.001 0.036 1.2.0.1 1.1.0.1 1.3.0.1

// discard the packet because a loop is detected
/192.168.7.1:31313 received from /192.168.2.4:31313 at 20.078
protocol=2 ttl=100 srcAdr=1.3.0.1 destAdr=1.2.0.1
RPv0
type: advert
pathvec: 1.2.0.0/16 20.002 0.035 1.3.0.1 1.1.0.1 1.2.0.1

Final Report

Routing table (28.225)
    prefix  timestamp     cost  link  VLD/INVLD  path
1.1.0.0/16     20.001    0.023     1      valid 1.1.0.1
1.3.0.0/16     20.001    0.036     1      valid 1.1.0.1 1.3.0.1

Forwarding table (28.230)
0.0.0.0/0 0
1.1.0.0/16 1
1.3.0.0/16 1

Router link cost statistics
  peerIp    count  avgCost  minCost  maxCost
 1.3.0.1       25    0.053    0.050    0.067
 1.1.0.1       25    0.024    0.021    0.048

SrcSnk statistics
  destIp    count avgDelay minDelay maxDelay
 1.1.0.1       34    0.027    0.019    0.045
 1.3.0.1       27    0.037    0.031    0.043
```

Part D. **[30 points]** In this part, you will be running some tests using the configuration and script you will find in the *net2* sub-directory. Commit all log files to your repository after finishing this part.

1. (5 points) Draw a diagram showing the logical links joining the four routers in the overlay network defined by the configuration files *r1*, *r2*, *r3*, and *r4*. Label the inter-router links with their assigned link costs.



2. (10 points) In this part, you will disable and re-enable one of the links while *script2* is running. This is done using a filter that is installed on *onl* router port 1.1. Click on this port in the RLI and select "Filter Table" from the menu. This will show you a "delete filter" which causes all packets received on this link to be discarded. At the right end of the filter table entry you will see a check box. Click on the check box and select "Commit" from the file menu in order to turn on the filter (effectively disabling the link). To turn off the filter (and re-enable the link), uncheck the box and select "Commit" again. Based on the configuration files, predict what this filter will do to the routing tables.

When the filter is enabled, r1 can no longer send advertisement packet to r2, which means r2 can no longer forward it to r3. Therefore, after 20 seconds, r3 will start accepting r4's advertisement about itself and r4's advertisement about r1, thus changes its routing table. After the filter is disabled again, r3 will receive the advertisement of r1 and r4 from r2, then it will set its routing table to the original state.

Now, run *script2* again (with the filter turned off) by typing

```
./script2 1 .333 100 static debug
```

after the script has run for about 30 seconds, turn on the filter. Then wait another 30 seconds and turn off the filter. Paste a copy of the content of *log1_3* file and add comments in bold, explaining the changes to the routing table at *r3*.

```
Forwarding table (10.032)
0.0.0.0/0 0
1.2.0.0/16 1

// When r3 receives r2's advertisement about itself, it adds the path to r2
in its routing table
Routing table (10.024)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     10.003    0.012     1      valid 1.2.0.1

Forwarding table (10.064)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 1

// When r3 receives r2's advertisement about r1, it adds the path to r1 in
its routing table
Routing table (10.063)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     10.003    0.012     1      valid 1.2.0.1
1.1.0.0/16     10.001    0.035     1      valid 1.2.0.1 1.1.0.1

Forwarding table (10.080)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 1
1.4.0.0/16 1

// When r3 receives r2's advertisement about r4, it adds the path to r4 in
its routing table
Routing table (10.080)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     10.003    0.012     1      valid 1.2.0.1
1.1.0.0/16     10.001    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     10.001    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1

// When r3 receives r2's advertisement about itself again with nothing
changed, it updates the timestamp
Routing table (20.021)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     20.004    0.013     1      valid 1.2.0.1
1.1.0.0/16     10.001    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     10.001    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1

// When r3 receives r2's advertisement about r1 again with nothing changed,
it updates the timestamp
Routing table (20.037)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     20.004    0.013     1      valid 1.2.0.1
1.1.0.0/16     20.001    0.035     1      valid 1.2.0.1 1.1.0.1
```

```
1.4.0.0/16     10.001    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r4 again with nothing changed, it updates the timestamp

```
Routing table (20.051)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     20.004    0.013     1      valid 1.2.0.1
1.1.0.0/16     20.001    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     20.002    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (30.024)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     30.001    0.013     1      valid 1.2.0.1
1.1.0.0/16     20.001    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     20.002    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r1 again with nothing changed, it updates the timestamp

```
Routing table (30.041)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     30.001    0.013     1      valid 1.2.0.1
1.1.0.0/16     30.001    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     20.002    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r4 again with nothing changed, it updates the timestamp

```
Routing table (30.049)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     30.001    0.013     1      valid 1.2.0.1
1.1.0.0/16     30.001    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     30.000    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (40.022)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     40.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     30.001    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     30.000    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (50.024)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     50.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     30.001    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     30.000    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1

Forwarding table (50.105)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 1
```

```
1.4.0.0/16 0
```

// When r3 receives r4's advertisement about itself, r3 chooses to use this path because timestamp has passed 20 seconds, it updates the route

```
Routing table (50.105)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     50.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     30.001    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     50.001    0.102     0      valid 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (60.028)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     60.001    0.013     1      valid 1.2.0.1
1.1.0.0/16     30.001    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     50.001    0.102     0      valid 1.4.0.1
```

// When r3 receives r4's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (60.107)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     60.001    0.013     1      valid 1.2.0.1
1.1.0.0/16     30.001    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     60.001    0.102     0      valid 1.4.0.1

Forwarding table (60.117)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 0
1.4.0.0/16 0
```

// When r3 receives r4's advertisement about r1, r3 chooses to use this path because timestamp has passed 20 seconds, it updates the route

```
Routing table (60.117)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     60.001    0.013     1      valid 1.2.0.1
1.1.0.0/16     60.000    0.115     0      valid 1.4.0.1 1.1.0.1
1.4.0.0/16     60.001    0.102     0      valid 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (70.021)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     70.001    0.013     1      valid 1.2.0.1
1.1.0.0/16     60.000    0.115     0      valid 1.4.0.1 1.1.0.1
1.4.0.0/16     60.001    0.102     0      valid 1.4.0.1

Forwarding table (70.044)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 1
1.4.0.0/16 0
```

// When the filter is disabled, r3 receives r2's advertisement about r1 with less cost, it updates the path

```
Routing table (70.044)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     70.001    0.013     1      valid 1.2.0.1
1.1.0.0/16     70.005    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     60.001    0.102     0      valid 1.4.0.1

Forwarding table (70.048)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 1
1.4.0.0/16 1
```

// When the filter is disabled, r3 receives r2's advertisement about r4 with less cost, it updates the path

```
Routing table (70.048)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     70.001    0.013     1      valid 1.2.0.1
1.1.0.0/16     70.005    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     70.001    0.048     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (80.025)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     80.004    0.012     1      valid 1.2.0.1
1.1.0.0/16     70.005    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     70.001    0.048     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r1 again with nothing changed, it updates the timestamp

```
Routing table (80.039)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     80.004    0.012     1      valid 1.2.0.1
1.1.0.0/16     80.001    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     70.001    0.048     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r4 again with nothing changed, it updates the timestamp

```
Routing table (80.050)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     80.004    0.012     1      valid 1.2.0.1
1.1.0.0/16     80.001    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     80.000    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (90.021)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     90.004    0.012     1      valid 1.2.0.1
1.1.0.0/16     80.001    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     80.000    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r1 again with nothing changed, it updates the timestamp

```
Routing table (90.042)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     90.004    0.012     1      valid 1.2.0.1
1.1.0.0/16     90.005    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     80.000    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r4 again with nothing changed, it updates the timestamp

```
Routing table (90.049)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     90.004    0.012     1      valid 1.2.0.1
1.1.0.0/16     90.005    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     90.001    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (100.020)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16    100.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     90.005    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     90.001    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r1 again with nothing changed, it updates the timestamp

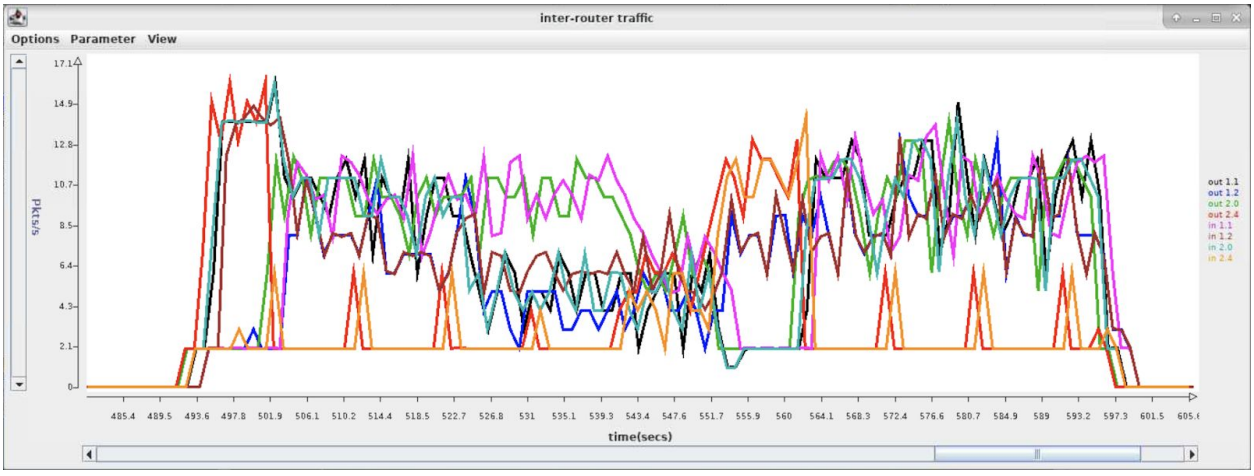
```
Routing table (100.040)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16    100.001    0.012     1      valid 1.2.0.1
1.1.0.0/16    100.000    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     90.001    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r4 again with nothing changed, it updates the timestamp

```
Routing table (100.053)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16    100.001    0.012     1      valid 1.2.0.1
1.1.0.0/16    100.000    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16    100.001    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1

Final Report

Routing table (108.209)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16    100.001    0.012     1      valid 1.2.0.1
1.1.0.0/16    100.000    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16    100.001    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1

Forwarding table (108.210)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 1
1.4.0.0/16 1
```

```
Router link cost statistics
  peerIp     count   avgCost   minCost   maxCost
 1.4.0.1       105     0.102     0.101     0.119
 1.2.0.1       105     0.013     0.010     0.031

SrcSnk statistics
  destIp     count avgDelay minDelay maxDelay
 1.1.0.1        76     0.050     0.031     0.115
 1.2.0.1        87     0.016     0.011     0.074
 1.4.0.1        90     0.061     0.041     0.104
```

Paste a screenshot of the monitoring window from your run below. To make everything fit in the window, click on the arrow on the x-axis repeatedly until everything is visible.



3. (15 points) Run *script2* again, but this time you need to enable link failure advertisement, run script2 with command

```
./script2 2 .333 100 static debug enFA
```

after the script has run for about 30 seconds, turn on the filter. Then wait another 30 seconds and turn off the filter. Paste a copy of the content of *log2_3* file and add comments in bold, explaining all changes to the routing table at *r3*.

```
Forwarding table (10.010)
0.0.0.0/0 0
1.2.0.0/16 1

// When r3 receives r2's advertisement about itself, it adds the path to r2
in its routing table
Routing table (10.005)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     10.001    0.012     1      valid 1.2.0.1

Forwarding table (10.060)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 1

// When r3 receives r2's advertisement about r1, it adds the path to r1 in
its routing table
Routing table (10.060)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     10.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     10.001    0.034     1      valid 1.2.0.1 1.1.0.1

Forwarding table (10.122)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 1
1.4.0.0/16 1

// When r3 receives r2's advertisement about r4, it adds the path to r4 in
its routing table
Routing table (10.122)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     10.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     10.001    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     10.015    0.048     1      valid 1.2.0.1 1.1.0.1 1.4.0.1

// When r3 receives r2's advertisement about itself again with nothing
changed, it updates the timestamp
Routing table (20.004)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     20.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     10.001    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     10.015    0.048     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r1 again with nothing changed, it updates the timestamp

Routing table (20.037)

prefix	timestamp	cost	link	VLD/INVLD	path
1.2.0.0/16	20.001	0.012	1	valid	1.2.0.1
1.1.0.0/16	20.001	0.034	1	valid	1.2.0.1 1.1.0.1
1.4.0.0/16	10.015	0.048	1	valid	1.2.0.1 1.1.0.1 1.4.0.1

// When r3 receives r2's advertisement about r4 again with nothing changed, it updates the timestamp

Routing table (20.101)

prefix	timestamp	cost	link	VLD/INVLD	path
1.2.0.0/16	20.001	0.012	1	valid	1.2.0.1
1.1.0.0/16	20.001	0.034	1	valid	1.2.0.1 1.1.0.1
1.4.0.0/16	20.001	0.046	1	valid	1.2.0.1 1.1.0.1 1.4.0.1

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

Routing table (30.004)

prefix	timestamp	cost	link	VLD/INVLD	path
1.2.0.0/16	30.001	0.013	1	valid	1.2.0.1
1.1.0.0/16	20.001	0.034	1	valid	1.2.0.1 1.1.0.1
1.4.0.0/16	20.001	0.046	1	valid	1.2.0.1 1.1.0.1 1.4.0.1

// When r3 receives r2's advertisement about r1 again with nothing changed, it updates the timestamp

Routing table (30.038)

prefix	timestamp	cost	link	VLD/INVLD	path
1.2.0.0/16	30.001	0.013	1	valid	1.2.0.1
1.1.0.0/16	30.001	0.035	1	valid	1.2.0.1 1.1.0.1
1.4.0.0/16	20.001	0.046	1	valid	1.2.0.1 1.1.0.1 1.4.0.1

// When r3 receives r2's advertisement about r4 again with nothing changed, it updates the timestamp

Routing table (30.107)

prefix	timestamp	cost	link	VLD/INVLD	path
1.2.0.0/16	30.001	0.013	1	valid	1.2.0.1
1.1.0.0/16	30.001	0.035	1	valid	1.2.0.1 1.1.0.1
1.4.0.0/16	30.007	0.048	1	valid	1.2.0.1 1.1.0.1 1.4.0.1

// When r3 receives r2's failure advertisement about the link from r1 to r2, it immediately sets all the paths containing this link as invalid

Routing table (34.006)

prefix	timestamp	cost	link	VLD/INVLD	path
1.2.0.0/16	30.001	0.013	1	valid	1.2.0.1
1.1.0.0/16	34.001	0.035	1	invalid	1.2.0.1 1.1.0.1
1.4.0.0/16	34.001	0.048	1	invalid	1.2.0.1 1.1.0.1 1.4.0.1

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

Routing table (40.008)

prefix	timestamp	cost	link	VLD/INVLD	path
1.2.0.0/16	40.002	0.012	1	valid	1.2.0.1
1.1.0.0/16	34.001	0.035	1	invalid	1.2.0.1 1.1.0.1

```
1.4.0.0/16     34.001    0.048     1    invalid     1.2.0.1 1.1.0.1 1.4.0.1

Forwarding table (40.115)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 0
1.4.0.0/16 1
```

// When r3 receives r4's advertisement about r1, it accepts this advertisement and choose this path (because the original one is invalid)

```
Routing table (40.115)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     40.002    0.012     1      valid 1.2.0.1
1.1.0.0/16     40.002    0.114     0      valid 1.4.0.1 1.1.0.1
1.4.0.0/16     34.001    0.048     1    invalid     1.2.0.1 1.1.0.1 1.4.0.1

Forwarding table (40.153)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 0
1.4.0.0/16 0
```

// When r3 receives r4's advertisement about itself, it accepts this advertisement and choose this path (because the original one is invalid)

```
Routing table (40.153)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     40.002    0.012     1      valid 1.2.0.1
1.1.0.0/16     40.002    0.114     0      valid 1.4.0.1 1.1.0.1
1.4.0.0/16     40.001    0.102     0      valid 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (50.005)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     50.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     40.002    0.114     0      valid 1.4.0.1 1.1.0.1
1.4.0.0/16     40.001    0.102     0      valid 1.4.0.1
```

// When r3 receives r4's advertisement about r1 again with nothing changed, it updates the timestamp

```
Routing table (50.113)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     50.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     50.001    0.114     0      valid 1.4.0.1 1.1.0.1
1.4.0.0/16     40.001    0.102     0      valid 1.4.0.1
```

// When r3 receives r4's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (50.158)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     50.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     50.001    0.114     0      valid 1.4.0.1 1.1.0.1
1.4.0.0/16     50.001    0.102     0      valid 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (60.012)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     60.006    0.012     1      valid 1.2.0.1
1.1.0.0/16     50.001    0.114     0      valid 1.4.0.1 1.1.0.1
1.4.0.0/16     50.001    0.102     0      valid 1.4.0.1
```

// When r3 receives r4's advertisement about r1 again with nothing changed, it updates the timestamp

```
Routing table (60.111)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     60.006    0.012     1      valid 1.2.0.1
1.1.0.0/16     60.001    0.114     0      valid 1.4.0.1 1.1.0.1
1.4.0.0/16     50.001    0.102     0      valid 1.4.0.1
```

// When r3 receives r4's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (60.155)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     60.006    0.012     1      valid 1.2.0.1
1.1.0.0/16     60.001    0.114     0      valid 1.4.0.1 1.1.0.1
1.4.0.0/16     60.001    0.102     0      valid 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (70.006)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     70.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     60.001    0.114     0      valid 1.4.0.1 1.1.0.1
1.4.0.0/16     60.001    0.102     0      valid 1.4.0.1

Forwarding table (70.035)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 1
1.4.0.0/16 0
```

// The filter is disabled. When r3 receives r2's advertisement about r1, it chooses this path because it has a lower cost.

```
Routing table (70.035)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     70.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     70.001    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     60.001    0.102     0      valid 1.4.0.1

Forwarding table (70.098)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 1
1.4.0.0/16 1
```

// When r3 receives r2's advertisement about r4, it chooses this path because it has a lower cost.

```
Routing table (70.098)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     70.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     70.001    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     70.001    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (80.006)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     80.001    0.013     1      valid 1.2.0.1
1.1.0.0/16     70.001    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     70.001    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r1 again with nothing changed, it updates the timestamp

```
Routing table (80.038)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     80.001    0.013     1      valid 1.2.0.1
1.1.0.0/16     80.000    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     70.001    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r4 again with nothing changed, it updates the timestamp

```
Routing table (80.098)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     80.001    0.013     1      valid 1.2.0.1
1.1.0.0/16     80.000    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     80.001    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (90.005)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     90.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     80.000    0.035     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     80.001    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r1 again with nothing changed, it updates the timestamp

```
Routing table (90.031)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     90.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     90.000    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     80.001    0.047     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r4 again with nothing changed, it updates the timestamp

```
Routing table (90.097)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16     90.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     90.000    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     90.001    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about itself again with nothing changed, it updates the timestamp

```
Routing table (100.005)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16    100.001    0.012     1      valid 1.2.0.1
1.1.0.0/16     90.000    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     90.001    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

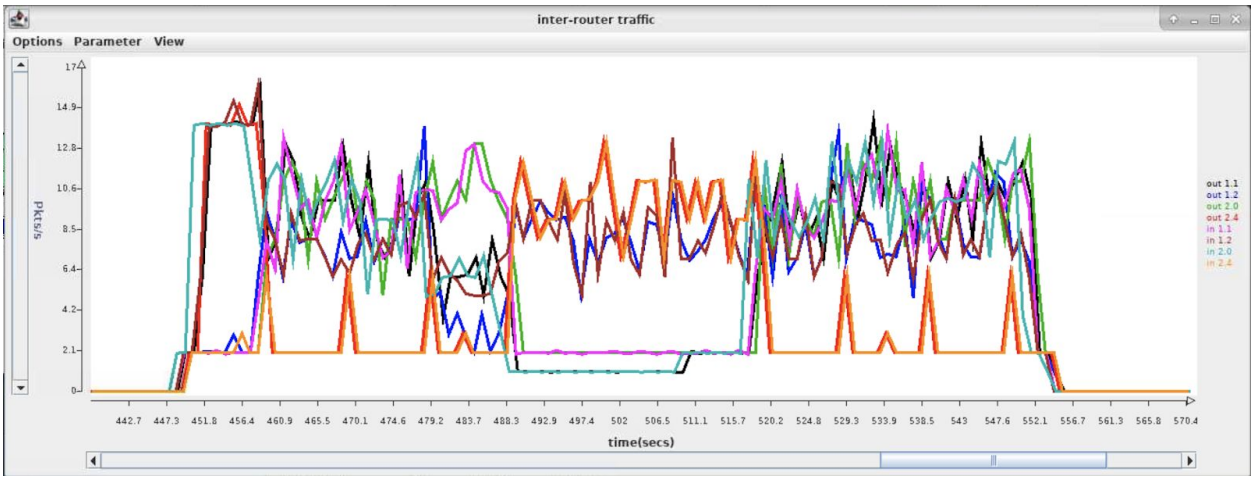
// When r3 receives r2's advertisement about r1 again with nothing changed, it updates the timestamp

```
Routing table (100.036)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16    100.001    0.012     1      valid 1.2.0.1
1.1.0.0/16    100.001    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16     90.001    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1
```

// When r3 receives r2's advertisement about r4 again with nothing changed, it updates the timestamp

```
Routing table (100.104)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16    100.001    0.012     1      valid 1.2.0.1
1.1.0.0/16    100.001    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16    100.006    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1

Final Report

Routing table (108.208)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.2.0.0/16    100.001    0.012     1      valid 1.2.0.1
1.1.0.0/16    100.001    0.034     1      valid 1.2.0.1 1.1.0.1
1.4.0.0/16    100.006    0.046     1      valid 1.2.0.1 1.1.0.1 1.4.0.1

Forwarding table (108.209)
0.0.0.0/0 0
1.2.0.0/16 1
1.1.0.0/16 1
1.4.0.0/16 1

Router link cost statistics
  peerIp    count  avgCost  minCost  maxCost
 1.4.0.1      105    0.102    0.100    0.117
 1.2.0.1      105    0.012    0.010    0.028

SrcSnk statistics
  destIp    count avgDelay minDelay maxDelay
 1.1.0.1       95    0.062    0.031    0.116
 1.2.0.1      107    0.016    0.011    0.074
 1.4.0.1       82    0.066    0.042    0.105
```

Paste a screenshot of the monitoring window from your run below. To make everything fit in the window, click on the arrow on the x-axis repeatedly until everything is visible.



From both the screenshots and content of log files above, what's the benefit of using link failure advertisement?

The link failure advertisement can let all the other routers immediately know the failure of this link, then set paths containing this link as invalid, and accept other valid advertisement in time. Without failure advertisement, even though the link from r1 to r2 is down, r3 does not know it until waiting for 20 seconds, and then accepting r4's advertisement. With failure advertisement, r3 will know the failure only 3 seconds later. It then is able to set its paths as invalid and accept r4's advertisement in time at t = 40. As shown above, in the first graph, all the new valid paths are set at roughly t = 60, whereas in the second graph, all the new valid paths are set at roughly t = 40.

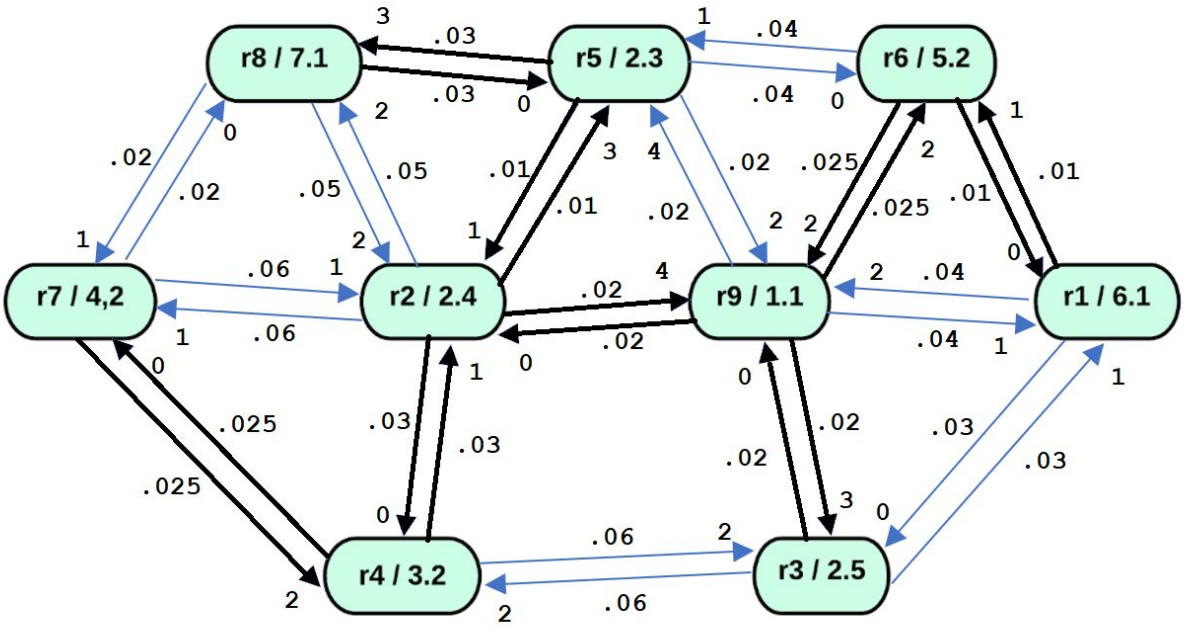
Part E. **[40 points] In this part, you will be using the configuration in the *net3* subdirectory. Commit all log files** to your repository after finishing this part.

1. (10 points) Using the information in the provided configuration files draw a network graph that represents this network. Each node in the graph should be labeled with the router number (e.g. r_1, r_2, ...) and the last two components of the IP address of its ONL host (so, for example, r_1 runs on the host whose address is 192.168.6.1, so label its node in the graph with "r_1/6.1"). Each link should be labeled with its cost and for each router, the endpoints of the links incident to it should be labeled 0, 1, 2,... where these local link numbers are determined by the order in which the neighbors are listed in the configuration file. For example, here is the relevant section from the configuration file for r_1.

```
neighbor: 1.3.0.1 192.168.2.5 .03
neighbor: 1.6.0.1 192.168.5.2 .01
neighbor: 1.9.0.1 192.168.1.1 .04
```

The link connecting to r_3 (which has IP address 1.3.0.1 in the overlay) would have an index of 0 at r_1. The link connecting to r_6 (which has IP address 1.6.0.1 in the overlay) would have an index of 1 at r_1. The link connecting to r_9 (which has IP address 1.9.0.1 in the overlay) would have an index of 2 at r_1.

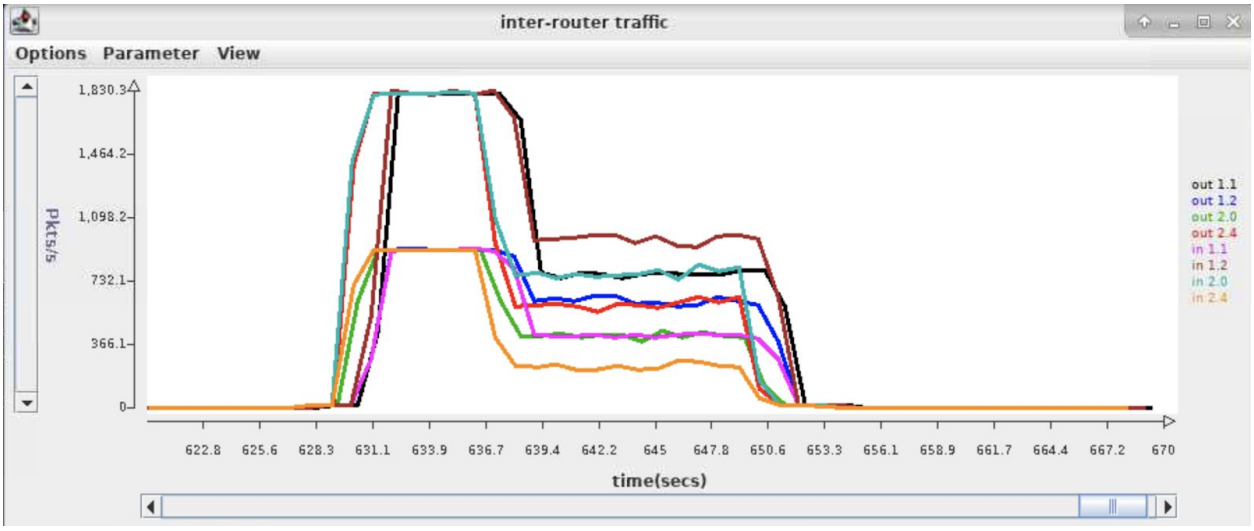
Find a shortest path tree in your network graph, rooted at router 2. Show the edges in the shortest path tree using heavy weight lines.



2. (10 points) Run the provided *script3* by typing

```
./script3 1 .01 20 static
```

Paste a screenshot of the monitoring window from your run here.



Paste the portion of the output from *log1_2* showing the final routing table at r_2.

```
Routing table (28.409)
    prefix  timestamp     cost  link  VLD/INVLD     path
1.9.0.0/16      20.001    0.022     4      valid 1.9.0.1
1.5.0.0/16      20.002    0.012     3      valid 1.5.0.1
1.4.0.0/16      20.001    0.032     0      valid 1.4.0.1
1.8.0.0/16      20.002    0.044     3      valid 1.5.0.1 1.8.0.1
1.3.0.0/16      20.002    0.044     4      valid 1.9.0.1 1.3.0.1
1.1.0.0/16      20.005    0.064     4      valid 1.9.0.1 1.1.0.1
1.7.0.0/16      20.001    0.062     1      valid 1.7.0.1
1.6.0.0/16      20.001    0.049     4      valid 1.9.0.1 1.6.0.1
```

Do the routes in your routing table match the shortest path tree in your network graph? If not, explain why not.

They do not match for the paths going to r1 and going to r7.

*The shortest path to r1 should be from r2 to r9 to r6 to r1, i.e. 0.02+0.025+0.01 = 0.055. However, the path in routing table is r2 to r9 to r1, i.e. 0.02+0.04 = 0.06.This is because r2 first received the direct advertisement of r1 from r9, it then first recorded 0.06 as the cost to r1. According to our code, a different path will be used only when the cost of new path is less than 0.9 times the original cost. When it received the advertisement of the "actual shortest path", it calculated: 0.06*0.9 = 0.054 and 0.055>0.054. Therefore, even though the new path has lower cost, r2 will still use the path from r9 to r1, instead of r9 to r6 to r1.*

*The same case for r7. The shortest path to r7 should be from r2 to r4 to r7, i.e. 0.03+0.025 = 0.055. However, the path in routing table is r2 directly to r7, i.e. 0.06.This is because r2 first received the direct advertisement of r7 from itself, it then first recorded 0.06 as the cost to r7. According to our code, a different path will be used only when the cost of new path is less than 0.9 times the original cost. When it received the advertisement of the "actual shortest path to r7" from r4, it calculated: 0.06*0.9 = 0.054 and 0.055>0.054. Therefore, even though the new path has lower cost, r2 will still use the path directly to r7, instead of r4 to r7.*

3. (10 points) Run *script3* by typing

```
./script3 2 .01 20 static debugg
```

Check the content of *log2_5* log file, show all advertisements for prefix 1.7.0.0/* that are *received* by r5 during the first round of advertisements (the ones that occur at around 10 seconds), and paste a copy of them here.

```
/192.168.2.3:31313 received from /192.168.7.1:31313 at 10.097
protocol=2 ttl=100 srcAdr=1.8.0.1 destAdr=1.5.0.1
RPv0
type: advert
pathvec: 1.7.0.0/16 10.002 0.022 1.8.0.1 1.7.0.1

/192.168.2.3:31313 received from /192.168.2.4:31313 at 10.147
protocol=2 ttl=100 srcAdr=1.2.0.1 destAdr=1.5.0.1
RPv0
type: advert
pathvec: 1.7.0.0/16 10.002 0.064 1.2.0.1 1.7.0.1

/192.168.2.3:31313 received from /192.168.1.1:31313 at 10.216
protocol=2 ttl=100 srcAdr=1.9.0.1 destAdr=1.5.0.1
RPv0
type: advert
pathvec: 1.7.0.0/16 10.002 0.122 1.9.0.1 1.2.0.1 1.7.0.1
```

Which of r_5's neighbors send it advertisements for this prefix? Why do these neighbors send advertisements and the others do not? Do your best to explain your observations based on the delays that advertisements will experience as they pass through the network.

In our experiment, r8, r2, and r9 send r5 the advertisement packet about r7. r6 does not send the packet. A host will forward an advertisement packet only when this advertisement packet changed its own routing table. For r8, it first receives an advertisement from r7. This triggers r8 to add the path to r7 in its routing table. Therefore, it forwards this advertisement of r7 to r5. The same case for r2 and r9. However, for r6, it first receives the advertisement packet of r7 from r5. Then it adds the path to r7 in its routing table and only forwards this packet to r9 and r1. (according to our policy, the advertisement packet will not be sent back). Later when r6 receives advertisement of r7 from r9 or r1, because it has already archived the shortest path, this will not trigger the modification of routing table, i.e. no forwarding of the advertisement.

4. (10 points) Run *script3* which exercises variable link delays by typing

```
./script3 3 .01 120 debug
```

Paste a screenshot of the monitoring window below. To get the entire run on the display, you will need to zoom out, by clicking repeatedly on the arrowhead at the right end of the horizontal axis.

Type the command

```
grep "1.1.0.0.16....." log3_*
```

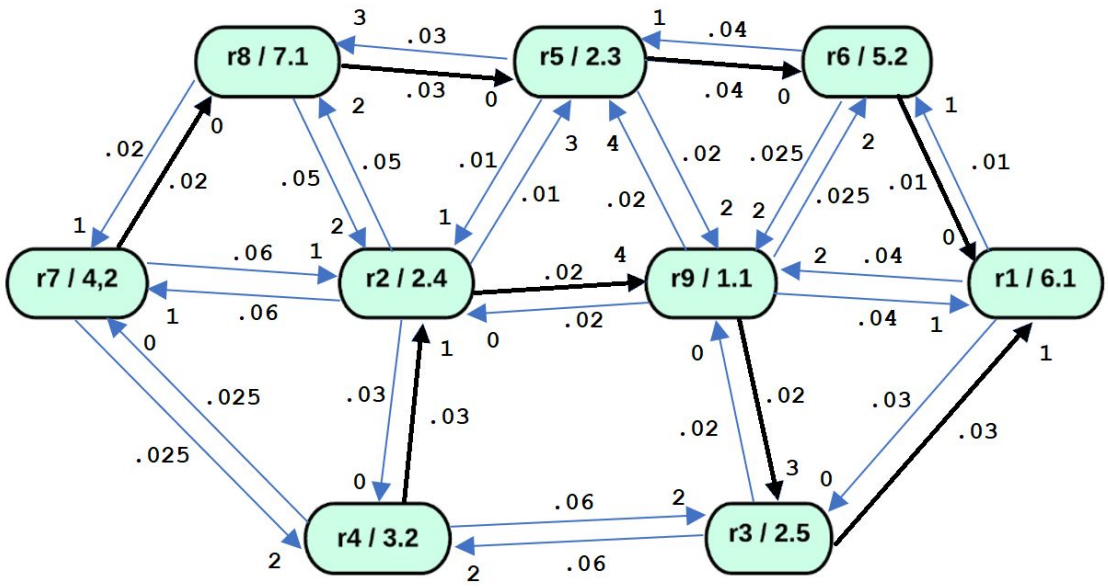
and paste the results below. Remove all lines that are *identical* to the one above them. Highlight all the places after time 10, where the path to 1.1.0.0/16 changes in a particular router, by making them bold.

```
log3_2:1.1.0.0/16     10.002    0.266     4      valid 1.9.0.1 1.3.0.1 1.1.0.1
log3_2:1.1.0.0/16     20.000    0.406     4      valid 1.9.0.1 1.3.0.1 1.1.0.1
log3_2:1.1.0.0/16     30.001    0.454     4      valid 1.9.0.1 1.3.0.1 1.1.0.1
log3_2:1.1.0.0/16     40.001    0.435     4      valid 1.9.0.1 1.3.0.1 1.1.0.1
log3_2:1.1.0.0/16     50.001    0.473     4      valid 1.9.0.1 1.3.0.1 1.1.0.1
log3_2:1.1.0.0/16     70.001    0.622     0      valid 1.4.0.1 1.3.0.1 1.1.0.1
log3_2:1.1.0.0/16     70.001    0.379     4      valid 1.9.0.1 1.1.0.1
log3_2:1.1.0.0/16     80.004    0.462     4      valid 1.9.0.1 1.1.0.1
log3_2:1.1.0.0/16     90.003    0.486     4      valid 1.9.0.1 1.1.0.1
log3_2:1.1.0.0/16    100.000    0.602     4      valid 1.9.0.1 1.1.0.1
log3_2:1.1.0.0/16    110.001    0.574     4      valid 1.9.0.1 1.1.0.1
log3_2:1.1.0.0/16    120.001    0.505     4      valid 1.9.0.1 1.1.0.1
log3_3:1.1.0.0/16     10.002    0.067     1      valid 1.1.0.1
log3_3:1.1.0.0/16     20.000    0.139     1      valid 1.1.0.1
log3_3:1.1.0.0/16     30.001    0.145     1      valid 1.1.0.1
log3_3:1.1.0.0/16     40.001    0.097     1      valid 1.1.0.1
log3_3:1.1.0.0/16     50.001    0.111     1      valid 1.1.0.1
log3_3:1.1.0.0/16     60.001    0.171     1      valid 1.1.0.1
log3_3:1.1.0.0/16     70.001    0.195     1      valid 1.1.0.1
log3_3:1.1.0.0/16     80.004    0.157     1      valid 1.1.0.1
log3_3:1.1.0.0/16     90.003    0.102     1      valid 1.1.0.1
log3_3:1.1.0.0/16    100.000    0.080     1      valid 1.1.0.1
log3_3:1.1.0.0/16    110.001    0.130     1      valid 1.1.0.1
log3_3:1.1.0.0/16    120.001    0.198     1      valid 1.1.0.1
log3_4:1.1.0.0/16     10.002    0.334     1      valid 1.2.0.1 1.9.0.1 1.3.0.1
1.1.0.1
log3_4:1.1.0.0/16     20.000    0.546     1      valid 1.2.0.1 1.9.0.1 1.3.0.1
1.1.0.1
log3_4:1.1.0.0/16     30.001    0.678     1      valid 1.2.0.1 1.9.0.1 1.3.0.1
1.1.0.1
log3_4:1.1.0.0/16     30.001    0.459     2      valid 1.3.0.1 1.1.0.1
log3_4:1.1.0.0/16     40.001    0.311     2      valid 1.3.0.1 1.1.0.1
log3_4:1.1.0.0/16     50.001    0.436     2      valid 1.3.0.1 1.1.0.1
log3_4:1.1.0.0/16     60.001    0.438     2      valid 1.3.0.1 1.1.0.1
log3_4:1.1.0.0/16     70.001    0.522     2      valid 1.3.0.1 1.1.0.1
log3_4:1.1.0.0/16     80.004    0.390     2      valid 1.3.0.1 1.1.0.1
log3_4:1.1.0.0/16     90.003    0.417     2      valid 1.3.0.1 1.1.0.1
log3_4:1.1.0.0/16    100.000    0.314     2      valid 1.3.0.1 1.1.0.1
log3_4:1.1.0.0/16    110.001    0.445     2      valid 1.3.0.1 1.1.0.1
log3_4:1.1.0.0/16    120.001    0.433     2      valid 1.3.0.1 1.1.0.1
log3_5:1.1.0.0/16     10.002    0.555     3      valid 1.8.0.1 1.7.0.1 1.4.0.1
1.2.0.1 1.9.0.1 1.3.0.1 1.1.0.1
log3_5:1.1.0.0/16     10.002    0.376     2      valid 1.9.0.1 1.3.0.1 1.1.0.1
log3_5:1.1.0.0/16     10.002    0.131     0      valid 1.6.0.1 1.1.0.1
```

```
log3_5:1.1.0.0/16     20.000    0.270     0      valid 1.6.0.1 1.1.0.1
log3_5:1.1.0.0/16     30.001    0.419     0      valid 1.6.0.1 1.1.0.1
log3_5:1.1.0.0/16     40.001    0.450     0      valid 1.6.0.1 1.1.0.1
log3_5:1.1.0.0/16     50.001    0.354     0      valid 1.6.0.1 1.1.0.1
log3_5:1.1.0.0/16     60.001    0.237     0      valid 1.6.0.1 1.1.0.1
log3_5:1.1.0.0/16     70.001    0.179     0      valid 1.6.0.1 1.1.0.1
log3_5:1.1.0.0/16     80.004    0.256     0      valid 1.6.0.1 1.1.0.1
log3_5:1.1.0.0/16     90.003    0.393     0      valid 1.6.0.1 1.1.0.1
log3_5:1.1.0.0/16    100.000    0.487     0      valid 1.6.0.1 1.1.0.1
log3_5:1.1.0.0/16    110.001    0.519     0      valid 1.6.0.1 1.1.0.1
log3_5:1.1.0.0/16    120.001    0.458     0      valid 1.6.0.1 1.1.0.1
log3_6:1.1.0.0/16     10.002    0.047     0      valid 1.1.0.1
log3_6:1.1.0.0/16     20.000    0.112     0      valid 1.1.0.1
log3_6:1.1.0.0/16     30.001    0.182     0      valid 1.1.0.1
log3_6:1.1.0.0/16     40.001    0.208     0      valid 1.1.0.1
log3_6:1.1.0.0/16     50.001    0.156     0      valid 1.1.0.1
log3_6:1.1.0.0/16     60.001    0.094     0      valid 1.1.0.1
log3_6:1.1.0.0/16     70.001    0.046     0      valid 1.1.0.1
log3_6:1.1.0.0/16     80.004    0.073     0      valid 1.1.0.1
log3_6:1.1.0.0/16     90.003    0.137     0      valid 1.1.0.1
log3_6:1.1.0.0/16    100.000    0.201     0      valid 1.1.0.1
log3_6:1.1.0.0/16    110.001    0.202     0      valid 1.1.0.1
log3_6:1.1.0.0/16    120.001    0.156     0      valid 1.1.0.1
log3_7:1.1.0.0/16     10.002    0.389     2      valid 1.4.0.1 1.2.0.1 1.9.0.1
1.3.0.1 1.1.0.1
log3_7:1.1.0.0/16     10.002    0.309     0      valid 1.8.0.1 1.5.0.1 1.6.0.1
1.1.0.1
log3_7:1.1.0.0/16     20.000    0.620     0      valid 1.8.0.1 1.5.0.1 1.6.0.1
1.1.0.1
log3_7:1.1.0.0/16     30.001    0.835     0      valid 1.8.0.1 1.5.0.1 1.6.0.1
1.1.0.1
log3_7:1.1.0.0/16     40.001    0.452     2      valid 1.4.0.1 1.3.0.1 1.1.0.1
log3_7:1.1.0.0/16     50.001    0.588     2      valid 1.4.0.1 1.3.0.1 1.1.0.1
log3_7:1.1.0.0/16     60.001    0.578     2      valid 1.4.0.1 1.3.0.1 1.1.0.1
log3_7:1.1.0.0/16     70.001    0.657     2      valid 1.4.0.1 1.3.0.1 1.1.0.1
log3_7:1.1.0.0/16     70.001    0.528     1      valid 1.2.0.1 1.9.0.1 1.1.0.1
log3_7:1.1.0.0/16     80.004    0.590     1      valid 1.2.0.1 1.9.0.1 1.1.0.1
log3_7:1.1.0.0/16     90.003    0.580     1      valid 1.2.0.1 1.9.0.1 1.1.0.1
log3_7:1.1.0.0/16    100.000    0.505     2      valid 1.4.0.1 1.3.0.1 1.1.0.1
log3_7:1.1.0.0/16    110.001    0.601     2      valid 1.4.0.1 1.3.0.1 1.1.0.1
log3_7:1.1.0.0/16    120.001    0.693     2      valid 1.4.0.1 1.3.0.1 1.1.0.1
log3_8:1.1.0.0/16     10.002    0.453     1      valid 1.7.0.1 1.4.0.1 1.2.0.1
1.9.0.1 1.3.0.1 1.1.0.1
log3_8:1.1.0.0/16     10.002    0.244     0      valid 1.5.0.1 1.6.0.1 1.1.0.1
log3_8:1.1.0.0/16     20.000    0.474     0      valid 1.5.0.1 1.6.0.1 1.1.0.1
log3_8:1.1.0.0/16     30.001    0.609     0      valid 1.5.0.1 1.6.0.1 1.1.0.1
log3_8:1.1.0.0/16     40.001    0.650     0      valid 1.5.0.1 1.6.0.1 1.1.0.1
log3_8:1.1.0.0/16     50.001    0.532     0      valid 1.5.0.1 1.6.0.1 1.1.0.1
log3_8:1.1.0.0/16     60.001    0.421     0      valid 1.5.0.1 1.6.0.1 1.1.0.1
log3_8:1.1.0.0/16     70.001    0.358     0      valid 1.5.0.1 1.6.0.1 1.1.0.1
log3_8:1.1.0.0/16     80.004    0.482     0      valid 1.5.0.1 1.6.0.1 1.1.0.1
log3_8:1.1.0.0/16     90.003    0.586     0      valid 1.5.0.1 1.6.0.1 1.1.0.1
log3_8:1.1.0.0/16    100.000    0.677     0      valid 1.5.0.1 1.6.0.1 1.1.0.1
log3_8:1.1.0.0/16    110.001    0.691     0      valid 1.5.0.1 1.6.0.1 1.1.0.1
```

```
log3_8:1.1.0.0/16     120.001     0.632      0       valid 1.5.0.1 1.6.0.1 1.1.0.1
log3_9:1.1.0.0/16      10.002     0.157      3       valid 1.3.0.1 1.1.0.1
log3_9:1.1.0.0/16      20.000     0.260      3       valid 1.3.0.1 1.1.0.1
log3_9:1.1.0.0/16      30.001     0.289      3       valid 1.3.0.1 1.1.0.1
log3_9:1.1.0.0/16      40.001     0.256      3       valid 1.3.0.1 1.1.0.1
log3_9:1.1.0.0/16      50.001     0.283      3       valid 1.3.0.1 1.1.0.1
log3_9:1.1.0.0/16      50.001     0.250      1       valid 1.1.0.1
log3_9:1.1.0.0/16      60.001     0.252      1       valid 1.1.0.1
log3_9:1.1.0.0/16      70.001     0.159      1       valid 1.1.0.1
log3_9:1.1.0.0/16      80.004     0.269      1       valid 1.1.0.1
log3_9:1.1.0.0/16      90.003     0.272      1       valid 1.1.0.1
log3_9:1.1.0.0/16     100.000     0.375      1       valid 1.1.0.1
log3_9:1.1.0.0/16     110.001     0.336      1       valid 1.1.0.1
log3_9:1.1.0.0/16     120.001     0.254      1       valid 1.1.0.1
```

Paste a copy of your network graph below and highlight the shortest path tree defined by the routes going to r_1 at time 15, by making the links heavy weight.



Find a time when the shortest path tree to r_1 differs from the one at time 15. Paste another copy of your network graph below and highlight the links in the shortest path tree at that time. During what time period is this shortest path tree used?

The shortest path tree shown below is used from time __100____ to time __120___.