###### *CSE 473 – Introduction to Computer Networks*

Lab 5 Report – 140 Points

##### *Your name: Gilad Gabriel Tsehori, Gai Ashkenazy*

***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.

\* we can receive multiple types of packets from the Substrate layer

\* this thread can do the following

\* this method keeps track of time since the program started to run, called

\* now in seconds.

\* if there is an incoming message from sub:

\* if it is for myIP address pass to application if protocol is 1,

\* if protocol is 2 and to myIP pass to router

\* else if it is not for this IP address, find next hop from routing table

\* and pass to next hop

\* else if we have an incoming message from router pass it to the link the

\* router specifies

\* else if we have an incoming message from application, pass to next hop by

\* a look up operation

\* else take a 1ms nap

\* repeat until quit=true

\*/

public void run() {

**now = 0;**

**double t0 = System.nanoTime() / sec;**

while (!quit) {

now = System.nanoTime() / sec - t0;

**// if the Substrate has an incoming packet**

**if (sub.incoming()) {**

**// extract information from the subtrate**

**Pair<Packet, Integer> inPair = sub.receive();**

**Packet inPacket = inPair.left;**

**int inLink = inPair.right;**

**// if it's addressed to this overlay node**

**if (inPacket.destAdr == myIp) {**

**if (inPacket.protocol == 1) {**

**// send to the SrcSnk**

**toSnk.add(inPacket);**

**} else {**

**// send to the Router**

**toRtr.add(inPair);**

**}**

**}**

**// not to this overlay node: forward it**

**else {**

**int nextHopLink = this.lookup(inPacket.destAdr);**

**inPacket.ttl--;**

**if (inPacket.ttl == 0) {**

**// do nothing; discard the packet**

**} else if (nextHopLink == -1) {**

**//next hop is not in our forwarding table**

**// forward to defult router**

**} else {**

**// forwarder is not ready or full**

**while (!this.ready()) { /\* busy wating \*/ }**

**// send packet to next hop**

**sub.send(inPacket, nextHopLink);**

**}**

**}**

**}**

**// else if we have a packet from the Router to send**

**else if (fromRtr.size() > 0) {**

**// send it to the Substrate**

**Pair<Packet, Integer> routerPair = null;**

**try {**

**routerPair = fromRtr.take();**

**} catch (Exception e) {**

**System.out.println("Forwarder:send: take exception" + e);**

**System.exit(1);**

**}**

**// forwarder is not ready or full**

**while (!this.ready()) { /\* busy wating \*/ }**

**// if forwarder is ready send to subtrate**

**sub.send(routerPair.left, routerPair.right);**

**}**

**// else if we have a payload from the SrcSnk to send**

**else if (fromSrc.size() > 0) {**

**Packet outPkt = null;**

**try {**

**outPkt = fromSrc.take();**

**} catch (Exception e) {**

**System.out.println("Forwarder:send: take exception" + e);**

**System.exit(1);**

**}**

**// lookup the outgoing link using dest IP address**

**int destLink = this.lookup(outPkt.destAdr);**

**// format a packet containing the payload and pass it to the Substrate**

**sub.send(outPkt, destLink);**

**} else {**

**// else nothing to do, so take a nap**

**try {**

**myThread.sleep(1);**

**} catch (Exception e) {**

**System.err.println("Forwarder:send: put exception" + e);**

**System.exit(1);**

}

}

}

}

public synchronized void addRoute(Prefix nuPrefix, int nuLnk) {

**// if table contains an entry with the same prefix, just update the link;**

**boolean flag = false;**

**Pair<Prefix, Integer> pp = new Pair<>(nuPrefix, nuLnk);**

**for (int i = 0; i < fwdTbl.size(); i++) {**

**if (nuPrefix.equals(fwdTbl.get(i).left)) {**

**fwdTbl.set(i, pp);**

**flag = true;**

**break;**

**} else if (fwdTbl.get(i).left.leng < nuPrefix.leng) { // optimization**

**fwdTbl.add(i, pp);**

**flag = true;**

**break;**

**}**

**}**

**// otherwise add an entry**

**if (!flag) fwdTbl.add(pp);**

**if (debug > 0) this.printTable();**

}

private synchronized int lookup(int ip) {

**// assuming fwdTbl is sorted from longest prefix to shortest, first match**

**// will be longest match**

**for (int i = 0; i < fwdTbl.size(); i++) {**

**if (fwdTbl.get(i).left.matches(ip)) {**

**return fwdTbl.get(i).right;**

**}**

**}**

**return -1;**

}

***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.

\* <p>

\* handle 3 clocks now - time since start, helloTime - time since last hello,

\* and pvSendTime - time since last advertisment was initialized

\* if hello wasn't sent for 1 sec send hellos

\* else if advertisement wasn't initialized for 10secm send advertisements

\* else if incoming packet from forwarder - handle incoming (method)

\* else take a nap for 1ms

\*/

public void run() {

double t0 = System.nanoTime() / sec;

double helloTime, pvSendTime;

now = (System.nanoTime() / sec) - t0;

helloTime = pvSendTime = now;

while (!quit) {

**now = (System.nanoTime() / sec) - t0;**

**// if it's time to send hello packets, do it**

**if (now - helloTime > 1) {**

**helloTime = now;**

**this.sendHellos();**

**}**

**// else if it's time to send advertisements, do it**

**else if (now - pvSendTime > 10) {**

**pvSendTime = now;**

**this.sendAdverts();**

**}**

**// 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 {**

**myThread.sleep(1);**

**} catch (Exception e) {**

**System.err.println("Forwarder:send: put exception" + 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);

private Route lookupRoute(Prefix inputPfx) {

**for (int i = 0; i < rteTbl.size(); i++) {**

**if (rteTbl.get(i).pfx.equals(inputPfx)) {**

**return rteTbl.get(i);**

**}**

**}**

**return null;**

}

private void addRoute(Route rte) {

**rteTbl.add(rte);**

}

private boolean updateRoute(Route rte, Route nuRte) {

**// nuRte link is disabled**

**boolean flag = false;**

**if (!nuRte.valid) {**

**flag = false;**

**// rte is invalid, replace rte with nuRte**

**} else if (!rte.valid) {**

**flag = true;**

**//rte path and link are the same as in nuRte, update cost and timestamp**

**} else if (rte.path.equals(nuRte.path) && (rte.outLink == nuRte.outLink)) {**

**flag = true;**

**} else if (nuRte.cost <= (0.9 \* rte.cost)) {**

**flag = true;**

**} else if (nuRte.timestamp >= (rte.timestamp + 20)) {**

**flag = true;**

**}**

**if (flag) {**

**rte.path = nuRte.path;**

**rte.cost = nuRte.cost;**

**rte.outLink = nuRte.outLink;**

**rte.pfx = nuRte.pfx;**

**rte.timestamp = nuRte.timestamp;**

**rte.valid = nuRte.valid;**

**}**

**return flag;**

}

public void sendHellos() {

for (LinkInfo lnkInfo : lnkVec) {

**int lnk = lnkVec.indexOf(lnkInfo);**

**boolean routeHasChanged = false;**

**// if no reply to the last hello, subtract 1 from**

**// link status if it's not already 0**

**if (!lnkInfo.gotReply) {**

**if (lnkInfo.helloState == 1) {**

**// go through the routes to check routes**

**// that contain the failed link**

**for (Route rte : rteTbl) {**

**boolean prevVal = (rte.valid) ? true : false;**

**rte.valid = (rte.outLink == lnk) ? false : prevVal;**

**routeHasChanged = (rte.valid != prevVal) ? true : routeHasChanged;**

**}**

**}**

**// substract hello state if it is more than 0**

**lnkInfo.helloState = (lnkInfo.helloState == 0) ?**

**0 : lnkInfo.helloState - 1;**

**}**

**// print routing table if debug is enabled**

**// and valid field of route is changed**

**if (routeHasChanged && debug > 0) {**

**printTable();**

**}**

**// send link failure advertisement if enFA is enabled**

**// and valid field of route is changed**

**if (routeHasChanged && enFA) this.sendFailureAdvert(lnk);**

**// send new hello, after setting gotReply to false**

**lnkInfo.gotReply = false;**

**while (!fwdr.ready4pkt()) { /\* busy waiting \*/ }**

**// make an hello packet p**

**Packet p = new Packet();**

**p.protocol = 2;**

**p.ttl = 99;**

**p.srcAdr = this.myIp;**

**p.destAdr = lnkInfo.peerIp;**

**p.payload = "RPv0\ntype: hello\ntimestamp: " + now + "\n";**

**fwdr.sendPkt(p, lnk);**

**}**

}

public void sendAdverts() {

**// for each prefix, advertise it**

**for (int i = 0; i < pfxList.size(); i++) {**

**// send advertisement**

**for (int j = 0; j < lnkVec.size(); j++) {**

**// make an advert packet p**

**Packet p = new Packet();**

**p.protocol = 2;**

**p.ttl = 99;**

**p.srcAdr = this.myIp;**

**//update destAdress**

**p.destAdr = lnkVec.get(j).peerIp;**

**p.payload = String.format("RPv0\ntype: advert\n"**

**+ "pathvec: %s %.3f 0 %s\n",**

**this.pfxList.get(i), now, this.myIpString);**

**//send packet in the right link**

**int outLink = j;**

**while (!fwdr.ready4pkt()) { /\* busy waiting \*/ }**

**fwdr.sendPkt(p, outLink);**

**}**

**}**

}

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();

**// 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")) {**

**//make a hello2u packet**

**int destAdress = p.srcAdr;**

**byte newProtocol = p.protocol;**

**String newPayload = lines[0] + "\ntype: hello2u\n" + lines[2] + "\n";**

**Packet newP = new Packet();**

**newP.srcAdr = myIp;**

**newP.destAdr = destAdress;**

**newP.ttl = 99;**

**newP.protocol = newProtocol;**

**newP.payload = newPayload;**

**while (!fwdr.ready4pkt()) { /\* busy waiting \*/ }**

**//send the hello2u packet**

**fwdr.sendPkt(newP, lnk);**

**}**

**// else it's a reply to a hello packet**

**else if (type.equals("hello2u")) {**

**// use timestamp to determine round-trip delay**

**double incomingTimestamp =**

**Double.parseDouble(lines[2].split(":")[1].trim());**

**double rtt = now - incomingTimestamp;**

**double linkCost = rtt / 2;**

**// use this to update the link cost using exponential**

**// weighted moving average method**

**LinkInfo lnkInfo = lnkVec.get(lnk);**

**double alpha = 0.1;**

**lnkInfo.cost = alpha \* linkCost + (1 - alpha) \* lnkInfo.cost;**

**// also, update link cost statistics**

**lnkInfo.count++;**

**lnkInfo.totalCost += lnkInfo.cost;**

**lnkInfo.maxCost = (lnkInfo.cost > lnkInfo.maxCost) ?**

**lnkInfo.cost : lnkInfo.maxCost;**

**lnkInfo.minCost = (lnkInfo.cost < lnkInfo.minCost) ?**

**lnkInfo.cost : lnkInfo.minCost;**

**// also, set gotReply to true**

**lnkInfo.gotReply = true;**

**lnkInfo.helloState = 3;**

}

}

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

// Parse the path vector line.

**// if the advert came from an invalid link disrigard this advert**

**if (lnkVec.get(lnk).helloState == 0) {**

**return;**

**}**

**// process the lines array**

**String[] pathvec = lines[2].split(" ");**

**Prefix pfx = new Prefix(pathvec[1]);**

**double inTs = Double.parseDouble(pathvec[2]);**

**double inCost = Double.parseDouble(pathvec[3]);**

**LinkedList<Integer> newPath = new LinkedList<>();**

**// If there is loop in path vector, ignore this packet.**

**// in the same process build a new path**

**int receivedAdvertFrom = -1;**

**for (int i = 4; i < pathvec.length; i++) {**

**newPath.add(Util.string2ip(pathvec[i].trim()));**

**if (i == 4) {**

**receivedAdvertFrom = Util.string2ip(pathvec[i].trim());**

**}**

**if (myIpString.equals(pathvec[i].trim())) {**

**return;**

**}**

**}**

**// add myIP to the head of the newPath Lnked List**

**newPath.addFirst(this.myIp);**

**// Form a new route, with cost equal to path vector cost**

**// plus the cost of the link on which it arrived.**

**Route newRte = new Route();**

**newRte.pfx = pfx;**

**newRte.timestamp = now;**

**newRte.path = newPath;**

**// add cost field of router**

**double newPathCost = inCost + lnkVec.get(lnk).cost;**

**newRte.cost = newPathCost;**

**newRte.outLink = lnk;**

**newRte.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;**

**Route rteInTbl = this.lookupRoute(pfx);**

**boolean rteTblHasChanged = false;**

**boolean addToFwrd = false;**

**LinkedList<Integer> pathBeforeUpdate = (this.lookupRoute(pfx) == null) ?**

**null : this.lookupRoute(pfx).path;**

**int outlinkBeforeUpdate = (this.lookupRoute(pfx) == null) ?**

**(-1) : this.lookupRoute(pfx).outLink;**

**if (rteInTbl == null) {**

**this.addRoute(newRte);**

**addToFwrd = true;**

**rteTblHasChanged = true;**

**} else {**

**rteTblHasChanged = this.updateRoute(rteInTbl, newRte);**

**}**

**// whenever an update changes the output link,**

**// update the forwarding table as well.**

**LinkedList<Integer> pathAfterUpdate = this.lookupRoute(pfx).path;**

**int outlinkAfterUpdate = this.lookupRoute(pfx).outLink;**

**boolean outLinkHasChanged = outlinkBeforeUpdate != outlinkAfterUpdate;**

**boolean pathHasChanged = (pathBeforeUpdate == null) ?**

**true : (!pathBeforeUpdate.equals(pathAfterUpdate));**

**if (debug > 0 && pathHasChanged) {**

**this.printTable();**

**}**

**if (outLinkHasChanged || addToFwrd) fwdr.addRoute(pfx, lnk);**

**// If the new route changed the routing table,**

**// extend the path vector and send it to other neighbors.**

**if (rteTblHasChanged || pathHasChanged || outLinkHasChanged) {**

**// send advertisement**

**for (int i = 0; i < lnkVec.size(); i++) {**

**int tempLnk = i;**

**// make an advert packet p**

**Packet p = new Packet();**

**p.protocol = 2;**

**p.ttl = 99;**

**p.srcAdr = this.myIp;**

**//change destAdr**

**p.destAdr = lnkVec.get(i).peerIp;**

**p.payload = String.format("RPv0\ntype: advert\n"**

**+ "pathvec: %s %.3f %.3f",**

**pfx, now, newPathCost);**

**//format the ip path of this packet**

**for (int ip : newPath) {**

**p.payload += " " + Util.ip2string(ip);**

**}**

**p.payload += "\n";**

**// dont send advert to the router that sent you**

**if (receivedAdvertFrom == lnkVec.get(i).peerIp) {**

**continue;**

**}**

**//send packet on the right link**

**while (!fwdr.ready4pkt()) { /\* busy waiting \*/ }**

**fwdr.sendPkt(p, tempLnk);**

**}**

**}**

}

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

// Parse the path vector line.

**String[] pathvec = lines[2].split(" ");**

**int startLinkFailure = Util.string2ip(pathvec[1]);**

**int endLinkFailure = Util.string2ip(pathvec[2]);**

**String startLinkFailureString = pathvec[1];**

**String endLinkFailureString = pathvec[2];**

**double inTs = Double.parseDouble(pathvec[3]);**

**LinkedList<Integer> path = new LinkedList<>();**

**// If there is loop in path vector, ignore this packet.**

**int receivedFAdvertFrom = -1;**

**for (int i = 4; i < pathvec.length; i++) {**

**path.add(Util.string2ip(pathvec[i]));**

**if (i == 4) {**

**receivedFAdvertFrom = Util.string2ip(pathvec[i].trim());**

**}**

**if (myIpString.equals(pathvec[i].trim())) {**

**return;**

**}**

**}**

**// go through routes to check if it contains the link**

**// set the route as invalid (false) if it does**

**boolean atLeastOneRouteUpdated = false;**

**// loop on all routes in rte table**

**for (int i = 0; i < rteTbl.size(); i++) {**

**// loop on the path of this entery in the rte table**

**for (int j = 0; j < rteTbl.get(i).path.size() - 1; j++) {**

**//if the entery contains the path that faild make this path invalid**

**if (rteTbl.get(i).path.get(j) ==**

**startLinkFailure && rteTbl.get(i).path.get(j + 1) == endLinkFailure) {**

**// if the path was valid and now it is going to change to not valid**

**if (rteTbl.get(i).valid) {**

**atLeastOneRouteUpdated = true;**

**}**

**rteTbl.get(i).valid = false;**

**}**

**}**

**}**

**// update the time stamp if route is changed**

**if (atLeastOneRouteUpdated) {**

**inTs = now;**

**}**

**// print route table if route is changed and debug is enabled**

**if (atLeastOneRouteUpdated && debug > 0) {**

**printTable();**

**}**

**//If one route is changed, extend the message and send it to other neighbors.**

**if (atLeastOneRouteUpdated) {**

**// add own IP address to front of path**

**path.addFirst(this.myIp);**

**// send failed advertisement packet**

**for (LinkInfo lnkInfo : lnkVec) {**

**// make an fadvert packet p**

**Packet p = new Packet();**

**p.protocol = 2;**

**p.ttl = 99;**

**p.srcAdr = this.myIp;**

**p.destAdr = lnkInfo.peerIp;**

**p.payload = String.format("RPv0\ntype: fadvert\n"**

**+ "linkfail: %s %s %.3f",**

**startLinkFailureString, endLinkFailureString, inTs);**

**//add ips to the path vector**

**for (int ip : path) {**

**p.payload += " " + Util.ip2string(ip);**

**}**

**p.payload += "\n";**

**int tempLnk = lnkVec.indexOf(lnkInfo);**

**// don't send to router that sent the advert to me**

**if (receivedFAdvertFrom == lnkInfo.peerIp) {**

**continue;**

**}**

**//send packet of the right link**

**while (!fwdr.ready4pkt()) { /\* busy waiting \*/ }**

**fwdr.sendPkt(p, tempLnk);**

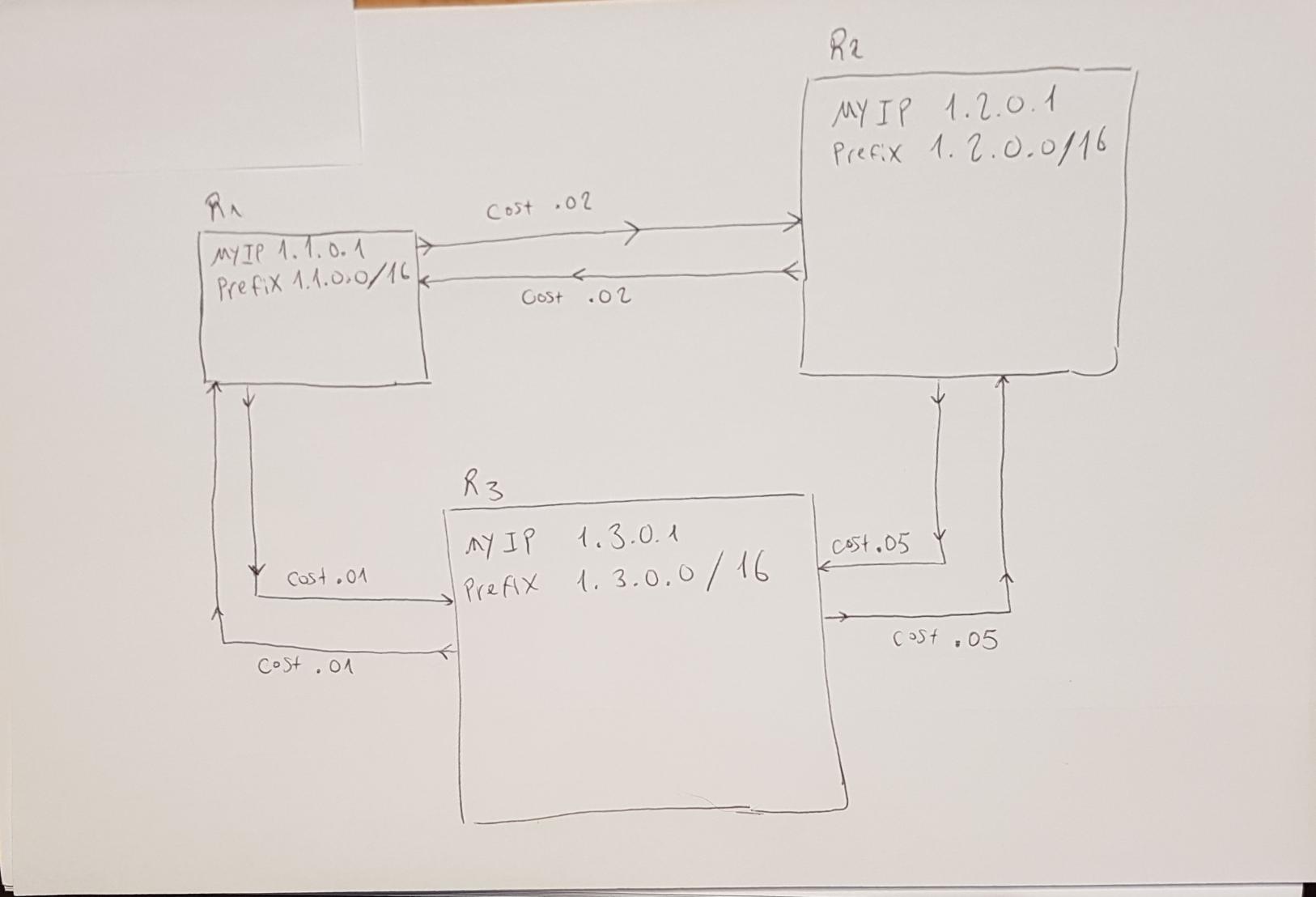
**}**

**}**

}

***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.

**

1. (10 points) Run *script1* in the *net1* sub-directory by typing

./script1 1 .333 20 static

Paste a copy of the output below.

kaiashk@onlusr:~/cse473s-f19-students-gai.ashkenazy/lab5/net1$ ./script1 1 0.333 20 static

delta= 0.333 runlength= 20 static

\*\*\*\*\*\*\*\*\*\*\* log 1 \*\*\*\*\*\*\*\*\*\*\*\*\*

Final Report

Routing table (28.222)

prefix timestamp cost link VLD/INVLD path

1.3.0.0/16 20.015 0.014 1 valid 1.1.0.1 1.3.0.1

1.2.0.0/16 20.012 0.026 0 valid 1.1.0.1 1.2.0.1

Forwarding table (28.233)

1.3.0.0/16 1

1.2.0.0/16 0

0.0.0.0/0 0

Router link cost statistics

peerIp count avgCost minCost maxCost

1.2.0.1 25 0.031 0.024 0.046

1.3.0.1 25 0.017 0.013 0.026

SrcSnk statistics

destIp count avgDelay minDelay maxDelay

1.2.0.1 23 0.028 0.021 0.045

1.3.0.1 38 0.024 0.011 0.045

\*\*\*\*\*\*\*\*\*\*\* log 2 \*\*\*\*\*\*\*\*\*\*\*\*\*

Final Report

Routing table (28.274)

prefix timestamp cost link VLD/INVLD path

1.1.0.0/16 20.038 0.022 1 valid 1.2.0.1 1.1.0.1

1.3.0.0/16 20.057 0.036 1 valid 1.2.0.1 1.1.0.1 1.3.0.1

Forwarding table (28.275)

1.1.0.0/16 1

1.3.0.0/16 1

0.0.0.0/0 0

Router link cost statistics

peerIp count avgCost minCost maxCost

1.3.0.1 24 0.052 0.050 0.052

1.1.0.1 24 0.022 0.020 0.023

SrcSnk statistics

destIp count avgDelay minDelay maxDelay

1.1.0.1 21 0.033 0.021 0.046

1.3.0.1 40 0.035 0.030 0.043

\*\*\*\*\*\*\*\*\*\*\* log 3 \*\*\*\*\*\*\*\*\*\*\*\*\*

Final Report

Routing table (28.267)

prefix timestamp cost link VLD/INVLD path

1.1.0.0/16 20.013 0.012 0 valid 1.3.0.1 1.1.0.1

1.2.0.0/16 20.027 0.038 0 valid 1.3.0.1 1.1.0.1 1.2.0.1

Forwarding table (28.278)

1.1.0.0/16 0

1.2.0.0/16 0

0.0.0.0/0 0

Router link cost statistics

peerIp count avgCost minCost maxCost

1.1.0.1 24 0.012 0.010 0.013

1.2.0.1 25 0.055 0.053 0.060

SrcSnk statistics

destIp count avgDelay minDelay maxDelay

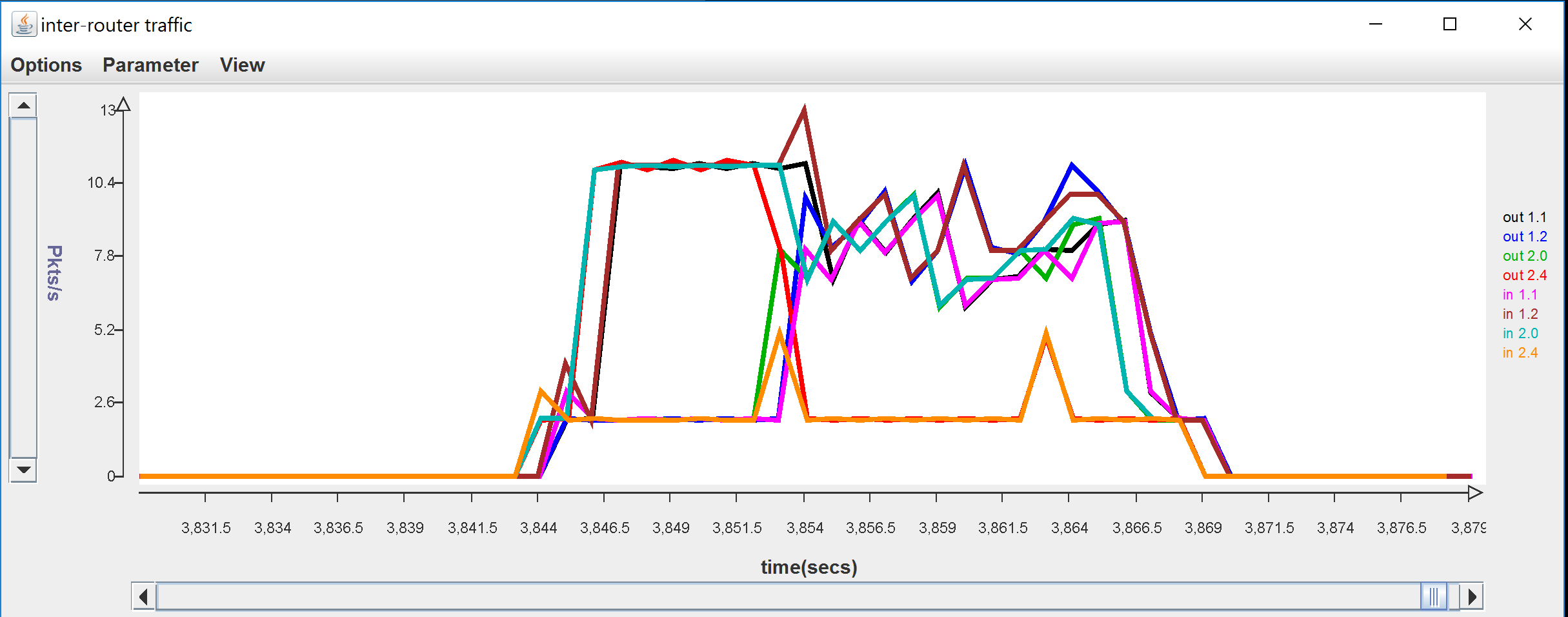
1.1.0.1 27 0.022 0.011 0.044

1.2.0.1 34 0.037 0.031 0.044

For each pair of routers *ri* and *rj*, write down the shortest path from *ri* to *rj* 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.

* *r1 to r2: r1 r2, with cost of 0.026*
* *r1 to r3: r1 r3, with cost of 0.014*
* *r2 to r1: r2 r1, with cost of 0.022*
* *r2 to r3: r2 r1 r3 , with cost of 0.036*
* *r3 to r1: r3 r1, with cost of 0.012*
* *r3 to r2: r3 r1 r2 , with cost of 0.038*

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?

*The script runs a total of 25 - 30 seconds, where the first 10 seconds include application packets and router protocol packets being sent only over the default route (so it could take a costly link to the destination, and not the cheapest way), in the next 10 seconds we see a change where application and routing protocol packets are being sent to the right next hop (in a cheaper way), and in the last 5 - 10 seconds the last application packet get to their destination and router protocol packets are being sent between the routers, still in the cheaper way.*

*The bandwidth of the links seems to change where we suspect it to; around the 10 seconds mark. At this point of time, we can see that the link connecting r2 and r3 (out 2.4) is completely avoided (only hello packets are sent on it) and the application packets take the cheaper way, we can see a small increase in packets coming in and out of r1 (port 1.1).*

*After this first few seconds, the router protocol continues to operate, so the bandwidth does not drop completely, but rather works and “converges” (according to our defined metrics) and the routing table stabilizes and does not change for a few seconds (3~5 seconds), which allows the Sender and Receiver instances of all routers to halt and quit.*

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?

*We will count the number of packets out on a link to one direction (form router port 1.1 to router port 2.0, not on the other way around). Also notice that the default link of router 1.1 is 0 which is the link that connects 1.1 with 2.0 (r1 to r2).*

*On router port 1.1, we have the host with the overlay address 1.1.0.1 (and with ONL network address of 192.168.4.2), while on router port 2.0 we have the host with overlay address 1.1.0.2 (and with ONL network address of 192.168.7.1).*

*We will calculate the rate of packets sent per second for each of the two sources of the packets:*

* *From SrcSnk: we have a delta of 0.333, which is the time (in seconds) that the application waits between it sends packets. Thus, a packet is sent every seconds, so so r1 sends packets to r2 and r3 through this link so in total packets from the application of r1 are*

*Also packet from r3 to r2 are going through this link so that is 30 more packets which makes a total of*

* *From router (router protocol packets): we have different kinds of routing protocol packets which are sent over the link.*
  + *Advert packets: because in the first part we don’t have any advert packet, we will have 0 of them.*
  + *Hello packets: every second, the router sends a hello packet to every link; so one hello packet is sent over the link.  
    In the meanwhile (before or after the hello packet is sent), an hello2u packet is sent over the link, as response to a hello packet from 1.1.0.2.  
    So in total, 2 hello protocol packets are being sent per seconds, or packets overall.*

*In total, 90 + 20 = 110 packets are sent on the link from router port 1.1 to router port 2.0.*

*To see whether this result corresponds to the observation we have in the graph, we first calculate the rate: packets per second.  
Looking at the black links in the graph (out of 1.1), we see that this corresponds to the calculations.*

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

*The second part of the run are the 10 seconds after the routing tables have changed to their optimal polices.*

*We will calculate the rate of packets sent per second for each of the two sources of the packets:*

* *From SrcSnk: we have a delta of 0.333, which is the time (in seconds) that the application waits between it sends packets. Thus, a packet is sent every seconds, so*

*Packets that r1 sends to r2 – 30 packets.*

*Packets that r3 sends to r2 – 30 packets.*

*Total of 60 packets in the second part of the run.*

* *From router (router protocol packets): we have different kinds of routing protocol packets which are sent over the link.*
  + *Advert packets: this time we will take into consideration the first round of advert packets, there are 2 adverts on this link. One advertising r1 and the second advertising r3. Both advertisements will be chosen by r2.*
  + *Hello packets: every second, the router sends a hello packet to every link; so one hello packet is sent over the link.  
    In the meanwhile (before or after the hello packet is sent), an hello2u packet is sent over the link, as response to a hello packet from 1.1.0.2.  
    So in total, 2 hello protocol packets are being sent per seconds, or packets overall.*

*In total, 60 + 2 + 20 = 82 packets are sent on the link from router port 1.1 to router port 2.0.*

*To see whether this result corresponds to the observation we have in the graph, we first calculate the rate: packets per second.  
Looking at the black links in the graph (out of 1.1) on the second part, we see that this corresponds to the calculations (on average).*

1. (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 10.040

protocol=2 ttl=99 srcAdr=1.2.0.1 destAdr=1.1.0.1

RPv0

type: advert

pathvec: 1.2.0.0/16 10.000 0 1.2.0.1

**// (1) The following advert has changed the routing table of router**

**1.1.0.2; a path to host 1.1.0.1 has been added to the routing table**

/192.168.7.1:31313 received from /192.168.4.2:31313 at 10.031

protocol=2 ttl=99 srcAdr=1.1.0.1 destAdr=1.2.0.1

RPv0

type: advert

pathvec: 1.1.0.0/16 10.000 0 1.1.0.1

Routing table (10.044)

prefix timestamp cost link VLD/INVLD path

1.1.0.0/16 10.044 0.022 1 valid 1.2.0.1 1.1.0.1

Forwarding table (10.057)

1.1.0.0/16 1

0.0.0.0/0 0

/192.168.7.1:31313 sending to /192.168.2.4:31313 at 10.069

protocol=2 ttl=99 srcAdr=1.2.0.1 destAdr=1.3.0.1

RPv0

type: advert

pathvec: 1.2.0.0/16 10.000 0 1.2.0.1

**// (2) The following advert has changed the routing table of router**

**1.1.0.2; a path to host 1.3.0.1 has been added to the routing table.**

**Change is seen in the table in the bottom of this page.**

/192.168.7.1:31313 received from /192.168.2.4:31313 at 10.045

protocol=2 ttl=99 srcAdr=1.3.0.1 destAdr=1.2.0.1

RPv0

type: advert

pathvec: 1.3.0.0/16 10.001 0 1.3.0.1

**// (3) The following advert has changed the routing table of router**

**1.1.0.2; a path to host 1.3.0.1 has been added to the routing table; this path goes through host 1.1.0.1 and it is cheaper than (2).**

**The result of this change is seen in the beginning of the next page.**

/192.168.7.1:31313 received from /192.168.4.2:31313 at 10.078

protocol=2 ttl=99 srcAdr=1.1.0.1 destAdr=1.2.0.1

RPv0

type: advert

pathvec: 1.3.0.0/16 10.044 0.012 1.1.0.1 1.3.0.1

Routing table (10.076)

prefix timestamp cost link VLD/INVLD path

1.1.0.0/16 10.044 0.022 1 valid 1.2.0.1 1.1.0.1

1.3.0.0/16 10.076 0.051 0 valid 1.2.0.1 1.3.0.1

Forwarding table (10.090)

1.1.0.0/16 1

1.3.0.0/16 0

0.0.0.0/0 0

Routing table (10.097)

prefix timestamp cost link VLD/INVLD path

1.1.0.0/16 10.044 0.022 1 valid 1.2.0.1 1.1.0.1

1.3.0.0/16 10.097 0.034 1 valid 1.2.0.1 1.1.0.1 1.3.0.1

Forwarding table (10.110)

1.1.0.0/16 1

1.3.0.0/16 1

0.0.0.0/0 0

**//The following advert didn’t change the routing table of router**

**1.1.0.2; because this path is more expensive than (1)**

/192.168.7.1:31313 received from /192.168.2.4:31313 at 10.111

protocol=2 ttl=99 srcAdr=1.3.0.1 destAdr=1.2.0.1

RPv0

type: advert

pathvec: 1.1.0.0/16 10.035 0.012 1.3.0.1 1.1.0.1

/192.168.7.1:31313 sending to /192.168.4.2:31313 at 10.119

protocol=2 ttl=99 srcAdr=1.2.0.1 destAdr=1.1.0.1

RPv0

type: advert

pathvec: 1.3.0.0/16 10.076 0.051 1.2.0.1 1.3.0.1

/192.168.7.1:31313 sending to /192.168.2.4:31313 at 10.122

protocol=2 ttl=99 srcAdr=1.2.0.1 destAdr=1.3.0.1

RPv0

type: advert

pathvec: 1.1.0.0/16 10.044 0.022 1.2.0.1 1.1.0.1

/192.168.7.1:31313 sending to /192.168.2.4:31313 at 10.171

protocol=2 ttl=99 srcAdr=1.2.0.1 destAdr=1.3.0.1

RPv0

type: advert

pathvec: 1.3.0.0/16 10.097 0.034 1.2.0.1 1.1.0.1 1.3.0.1

**// this path contains a loop and will be discarded**

/192.168.7.1:31313 received from /192.168.2.4:31313 at 10.146

protocol=2 ttl=99 srcAdr=1.3.0.1 destAdr=1.2.0.1

RPv0

type: advert

pathvec: 1.2.0.0/16 10.103 0.035 1.3.0.1 1.1.0.1 1.2.0.1

/192.168.7.1:31313 sending to /192.168.4.2:31313 at 20.023

protocol=2 ttl=99 srcAdr=1.2.0.1 destAdr=1.1.0.1

RPv0

type: advert

pathvec: 1.2.0.0/16 20.000 0 1.2.0.1

**// This path is already in the routing table, timestamp will be updated**

/192.168.7.1:31313 received from /192.168.4.2:31313 at 19.999

protocol=2 ttl=99 srcAdr=1.1.0.1 destAdr=1.2.0.1

RPv0

type: advert

pathvec: 1.1.0.0/16 20.001 0 1.1.0.1

**// This path is already in the routing table, timestamp will be updated**

/192.168.7.1:31313 received from /192.168.4.2:31313 at 20.038

protocol=2 ttl=99 srcAdr=1.1.0.1 destAdr=1.2.0.1

RPv0

type: advert

pathvec: 1.3.0.0/16 20.028 0.013 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=99 srcAdr=1.2.0.1 destAdr=1.3.0.1

RPv0

type: advert

pathvec: 1.2.0.0/16 20.000 0 1.2.0.1

**// There is a better path in routing table, (3)**

/192.168.7.1:31313 received from /192.168.2.4:31313 at 20.044

protocol=2 ttl=99 srcAdr=1.3.0.1 destAdr=1.2.0.1

RPv0

type: advert

pathvec: 1.3.0.0/16 20.001 0 1.3.0.1

**// There is a better path in routing table, (1)**

/192.168.7.1:31313 received from /192.168.2.4:31313 at 20.066

protocol=2 ttl=99 srcAdr=1.3.0.1 destAdr=1.2.0.1

RPv0

type: advert

pathvec: 1.1.0.0/16 20.012 0.013 1.3.0.1 1.1.0.1

/192.168.7.1:31313 sending to /192.168.2.4:31313 at 20.087

protocol=2 ttl=99 srcAdr=1.2.0.1 destAdr=1.3.0.1

RPv0

type: advert

pathvec: 1.1.0.0/16 20.032 0.023 1.2.0.1 1.1.0.1

/192.168.7.1:31313 sending to /192.168.2.4:31313 at 20.100

protocol=2 ttl=99 srcAdr=1.2.0.1 destAdr=1.3.0.1

RPv0

type: advert

pathvec: 1.3.0.0/16 20.045 0.036 1.2.0.1 1.1.0.1 1.3.0.1

**// this path contains a loop and will be discarded**

/192.168.7.1:31313 received from /192.168.2.4:31313 at 20.078

protocol=2 ttl=99 srcAdr=1.3.0.1 destAdr=1.2.0.1

RPv0

type: advert

pathvec: 1.2.0.0/16 20.043 0.035 1.3.0.1 1.1.0.1 1.2.0.1

Final Report

Routing table (28.248)

prefix timestamp cost link VLD/INVLD path

1.1.0.0/16 20.032 0.023 1 valid 1.2.0.1 1.1.0.1

1.3.0.0/16 20.045 0.036 1 valid 1.2.0.1 1.1.0.1 1.3.0.1

Forwarding table (28.248)

1.1.0.0/16 1

1.3.0.0/16 1

0.0.0.0/0 0

Router link cost statistics

peerIp count avgCost minCost maxCost

1.3.0.1 25 0.052 0.051 0.053

1.1.0.1 25 0.023 0.021 0.024

SrcSnk statistics

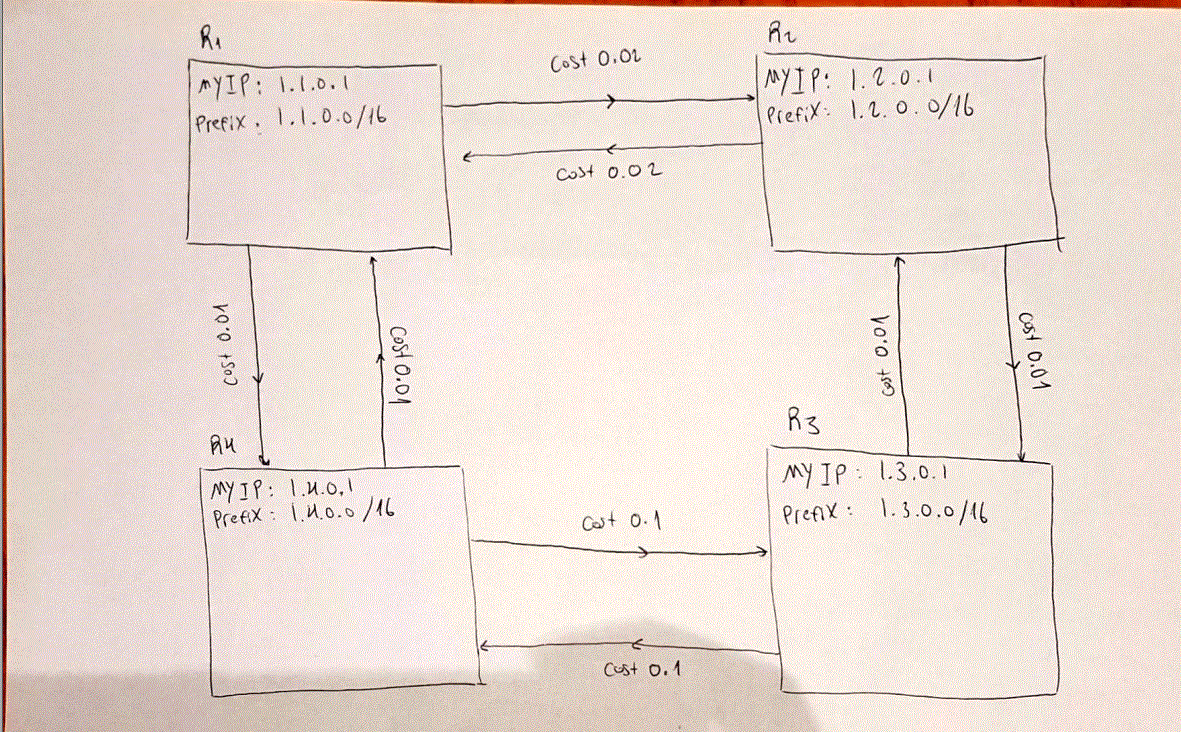
destIp count avgDelay minDelay maxDelay

1.1.0.1 25 0.028 0.021 0.044

1.3.0.1 36 0.037 0.031 0.044

***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.

**

1. (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.

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*.

kaiashk@onlusr:~/cse473s-f19-students-gai.ashkenazy/lab5/net2$ cat log1\_3

**// This route is added because it is the first route to advertise prefix 1.2.0.0**

Routing table (10.027)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 10.027 0.012 1 valid 1.3.0.1 1.2.0.1

Forwarding table (10.044)

1.2.0.0/16 1

0.0.0.0/0 0

**// This route is added because it is the first route to advertise prefix 1.1.0.0**

Routing table (10.086)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 10.027 0.012 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 10.086 0.035 1 valid 1.3.0.1 1.2.0.1 1.1.0.1

Forwarding table (10.101)

1.2.0.0/16 1

1.1.0.0/16 1

0.0.0.0/0 0

**// This route is added because it is the first route to advertise prefix 1.4.0.0**

Routing table (10.104)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 10.027 0.012 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 10.086 0.035 1 valid 1.3.0.1 1.2.0.1 1.1.0.1

1.4.0.0/16 10.104 0.047 1 valid 1.3.0.1 1.2.0.1 1.1.0.1 1.4.0.1

Forwarding table (10.113)

1.2.0.0/16 1

1.1.0.0/16 1

1.4.0.0/16 1

0.0.0.0/0 0

**// As of timestamp 30 (effective 33), the link was disabled.**

**// Route advertising prefix 1.4.0.1 is changed because 20sec has passed since // it’s been updated, notice that the router is still thinking the path //1.2.0.1 – 1.1.0.1 is valid, although it’s not. This changes slowly because // enFA is off (no Fadverts, this took as required about 20 seconds)**

Routing table (50.115)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 50.015 0.012 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 30.046 0.034 1 valid 1.3.0.1 1.2.0.1 1.1.0.1

1.4.0.0/16 50.115 0.102 0 valid 1.3.0.1 1.4.0.1

Forwarding table (50.128)

1.2.0.0/16 1

1.1.0.0/16 1

1.4.0.0/16 0

0.0.0.0/0 0

**// Route advertising prefix 1.1.0.1 is changed because 20sec has passed since // it’s been updated, notice that the router is still thinking the path //1.2.0.1 – 1.1.0.1 is valid, although it’s not. This changes slowly because // enFA is off (no Fadverts, this took as required about 20 seconds)**

Routing table (50.133)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 50.015 0.012 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 50.133 0.114 0 valid 1.3.0.1 1.4.0.1 1.1.0.1

1.4.0.0/16 50.115 0.102 0 valid 1.3.0.1 1.4.0.1

Forwarding table (50.148)

1.2.0.0/16 1

1.1.0.0/16 0

1.4.0.0/16 0

0.0.0.0/0 0

**// Route advertising prefix 1.1.0.1 is changed because the filter is off and // now traffic can flow on the link 1.2.0.1 – 1.1.0.1, and this is a cheaper // path than the previous one.**

Routing table (70.055)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 70.014 0.012 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 70.055 0.034 1 valid 1.3.0.1 1.2.0.1 1.1.0.1

1.4.0.0/16 60.112 0.102 0 valid 1.3.0.1 1.4.0.1

Forwarding table (70.057)

1.2.0.0/16 1

1.1.0.0/16 1

1.4.0.0/16 0

0.0.0.0/0 0

**// Route advertising prefix 1.4.0.1 is changed because the filter is off and // now traffic can flow on the link 1.2.0.1 – 1.1.0.1, and this is a cheaper // path than the previous one.**

Routing table (70.077)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 70.014 0.012 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 70.055 0.034 1 valid 1.3.0.1 1.2.0.1 1.1.0.1

1.4.0.0/16 70.077 0.046 1 valid 1.3.0.1 1.2.0.1 1.1.0.1 1.4.0.1

Forwarding table (70.086)

1.2.0.0/16 1

1.1.0.0/16 1

1.4.0.0/16 1

0.0.0.0/0 0

Final Report

Routing table (108.182)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 100.015 0.012 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 100.055 0.034 1 valid 1.3.0.1 1.2.0.1 1.1.0.1

1.4.0.0/16 100.062 0.046 1 valid 1.3.0.1 1.2.0.1 1.1.0.1 1.4.0.1

Forwarding table (108.182)

1.2.0.0/16 1

1.1.0.0/16 1

1.4.0.0/16 1

0.0.0.0/0 0

Router link cost statistics

peerIp count avgCost minCost maxCost

1.4.0.1 105 0.102 0.101 0.102

1.2.0.1 105 0.012 0.011 0.013

SrcSnk statistics

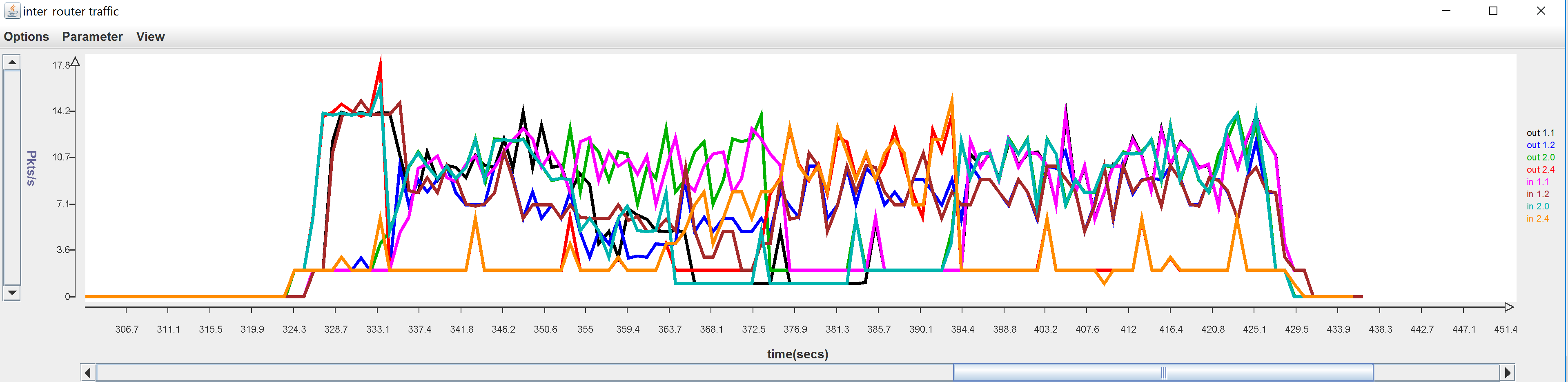
destIp count avgDelay minDelay maxDelay

1.1.0.1 82 0.056 0.031 0.115

1.2.0.1 94 0.017 0.011 0.075

1.4.0.1 84 0.061 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.



1. (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*.

kaiashk@onlusr:~/cse473s-f19-students-gai.ashkenazy/lab5/net2$ cat log2\_3

**// This route is added because it is the first route to advertise prefix 1.2.0.0**

Routing table (10.024)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 10.024 0.012 1 valid 1.3.0.1 1.2.0.1

Forwarding table (10.040)

1.2.0.0/16 1

0.0.0.0/0 0

**// This route is added because it is the first route to advertise prefix 1.1.0.0**

Routing table (10.082)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 10.024 0.012 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 10.082 0.035 1 valid 1.3.0.1 1.2.0.1 1.1.0.1

Forwarding table (10.094)

1.2.0.0/16 1

1.1.0.0/16 1

0.0.0.0/0 0

**// This route is added because it is the first route to advertise prefix 1.4.0.0**

Routing table (10.106)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 10.024 0.012 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 10.082 0.035 1 valid 1.3.0.1 1.2.0.1 1.1.0.1

1.4.0.0/16 10.106 0.047 1 valid 1.3.0.1 1.2.0.1 1.1.0.1 1.4.0.1

Forwarding table (10.121)

1.2.0.0/16 1

1.1.0.0/16 1

1.4.0.0/16 1

0.0.0.0/0 0

**// Both paths that contain the path 1.2.0.1 – 1.1.0.1 are invalid, this //happened at time 33 because r3 received a Fadvert after 3 consecutive //hellos got lost on the filtered link.**

Routing table (33.045)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 30.010 0.013 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 30.048 0.035 1 invalid 1.3.0.1 1.2.0.1 1.1.0.1

1.4.0.0/16 30.056 0.046 1 invalid 1.3.0.1 1.2.0.1 1.1.0.1 1.4.0.1

**// The route advertising prefix 1.4.0.0 is changed because the previews path // was invalid**

Routing table (40.100)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 40.010 0.013 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 30.048 0.035 1 invalid 1.3.0.1 1.2.0.1 1.1.0.1

1.4.0.0/16 40.100 0.102 0 valid 1.3.0.1 1.4.0.1

Forwarding table (40.106)

1.2.0.0/16 1

1.1.0.0/16 1

1.4.0.0/16 0

0.0.0.0/0 0

**// The route advertising prefix 1.1.0.0 is changed because the previews path // was invalid**

Routing table (40.135)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 40.010 0.013 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 40.135 0.114 0 valid 1.3.0.1 1.4.0.1 1.1.0.1

1.4.0.0/16 40.100 0.102 0 valid 1.3.0.1 1.4.0.1

Forwarding table (40.141)

1.2.0.0/16 1

1.1.0.0/16 0

1.4.0.0/16 0

0.0.0.0/0 0

**// at this time the filter is back off and traffic can flow on the path //1.2.0.1 – 1.1.0.1, so adverts are sent advertising this path to 1.1.0.1, //and because it is cheaper the routing table changes**

Routing table (60.054)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 60.011 0.012 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 60.054 0.034 1 valid 1.3.0.1 1.2.0.1 1.1.0.1

1.4.0.0/16 50.101 0.102 0 valid 1.3.0.1 1.4.0.1

Forwarding table (60.058)

1.2.0.0/16 1

1.1.0.0/16 1

1.4.0.0/16 0

0.0.0.0/0 0

**// at this time the filter is back off and traffic can flow on the path //1.2.0.1 – 1.1.0.1, so adverts are sent advertising this path to 1.4.0.1, //and because it is cheaper the routing table changes**

Routing table (60.072)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 60.011 0.012 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 60.054 0.034 1 valid 1.3.0.1 1.2.0.1 1.1.0.1

1.4.0.0/16 60.072 0.046 1 valid 1.3.0.1 1.2.0.1 1.1.0.1 1.4.0.1

Forwarding table (60.081)

1.2.0.0/16 1

1.1.0.0/16 1

1.4.0.0/16 1

0.0.0.0/0 0

Final Report

Routing table (108.174)

prefix timestamp cost link VLD/INVLD path

1.2.0.0/16 100.014 0.012 1 valid 1.3.0.1 1.2.0.1

1.1.0.0/16 100.048 0.034 1 valid 1.3.0.1 1.2.0.1 1.1.0.1

1.4.0.0/16 100.050 0.046 1 valid 1.3.0.1 1.2.0.1 1.1.0.1 1.4.0.1

Forwarding table (108.175)

1.2.0.0/16 1

1.1.0.0/16 1

1.4.0.0/16 1

0.0.0.0/0 0

Router link cost statistics

peerIp count avgCost minCost maxCost

1.4.0.1 105 0.102 0.101 0.103

1.2.0.1 105 0.012 0.011 0.013

SrcSnk statistics

destIp count avgDelay minDelay maxDelay

1.1.0.1 103 0.051 0.031 0.113

1.2.0.1 91 0.016 0.011 0.073

1.4.0.1 81 0.061 0.042 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.



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

*The benefits of link failure advertisements are clearly shown in the logs and graphs, when link failure advertisements are sent r3 finds out that some path he uses is invalid, this makes him change the routing table entries to invalid, which makes him find alternative paths when advertisements are sent. We see on the logs and graphs that without failure advertisements it takes the router 20 seconds to find a valid path, and with them it takes less than 10 seconds.*

* *A note – the instructions tells to wait roughly 30 seconds, this could cause different times stamps on some routes.*

***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.

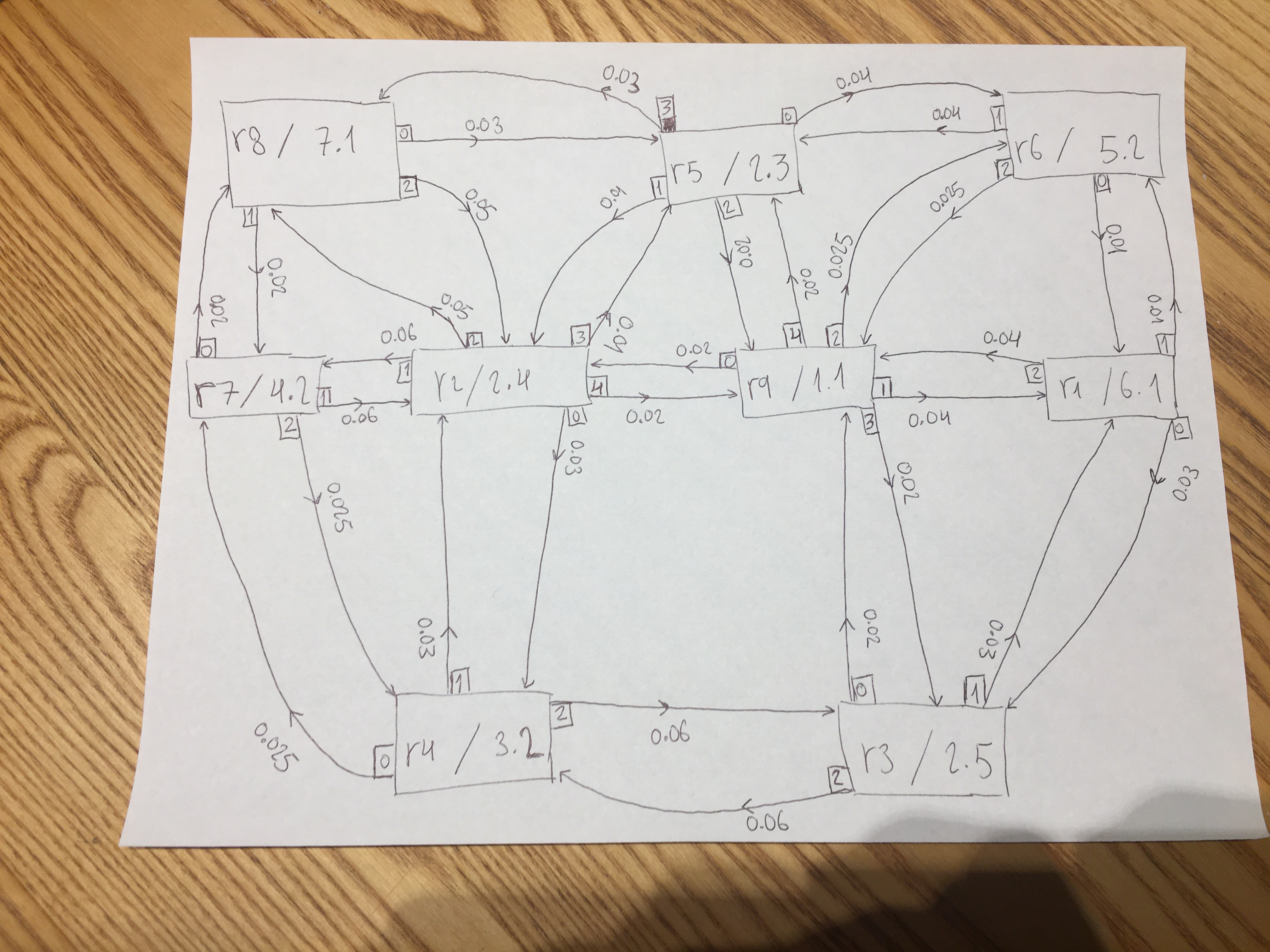
1. (10 points) Using the information in the provided configuration files draw a network graph that represents this network. Each node in the graph should be labeled with the router number (e.g. *r*1, *r*2, ...) and the last two components of the IP address of its ONL host (so, for example, *r*1 runs 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

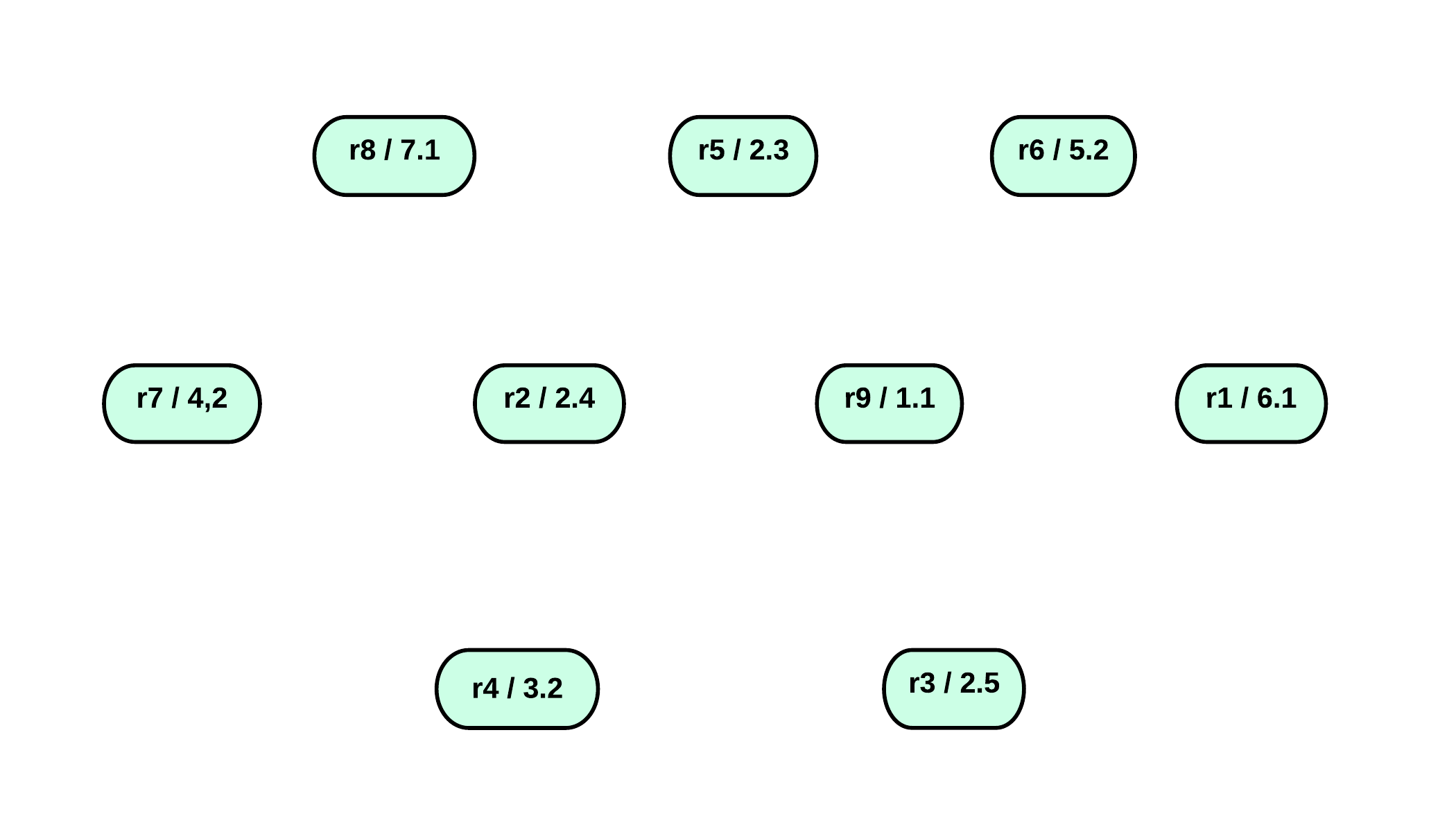
neighbor: 1.6.0.1 192.168.5.2 .01

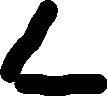
neighbor: 1.9.0.1 192.168.1.1 .04

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.

**

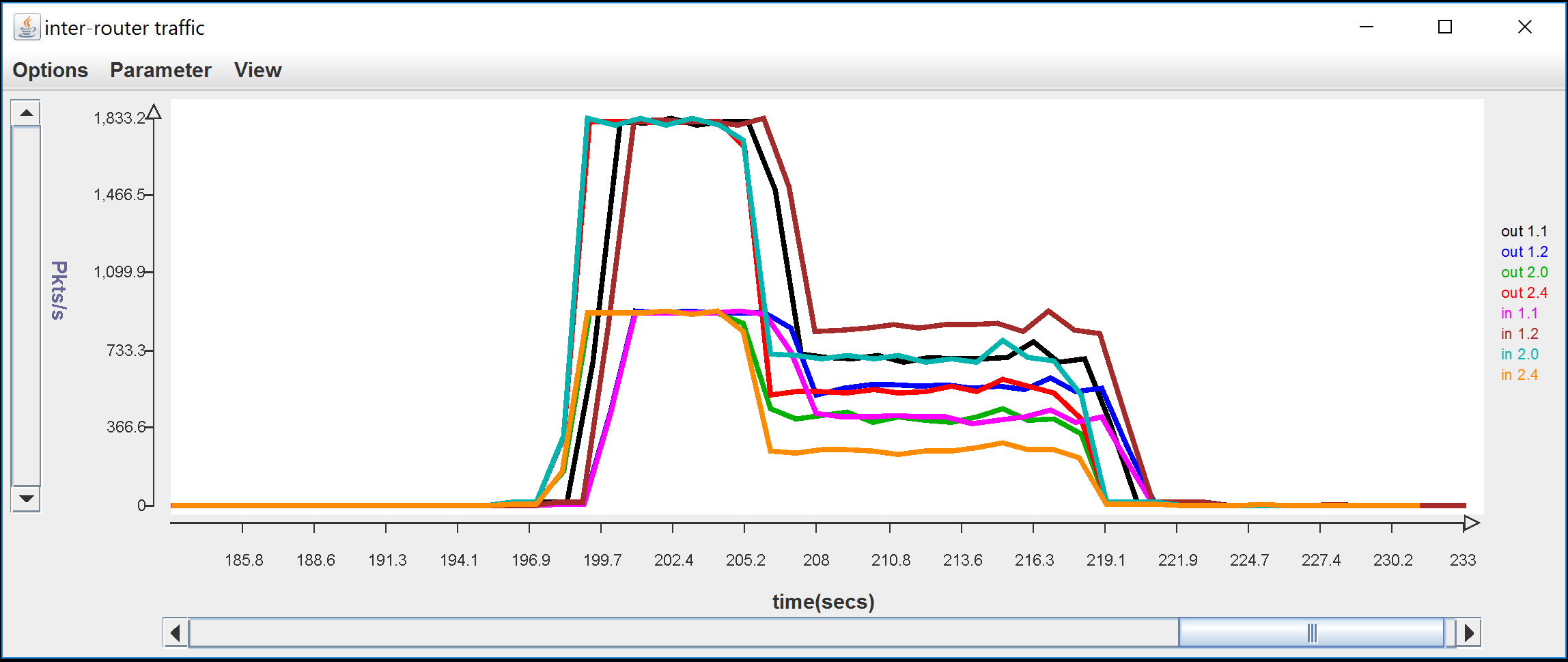


These are the shortes paths from r2 to all the other routers, the output links and initial costs are in the drawing above.

1. (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 (28.373)

prefix timestamp cost link VLD/INVLD path

1.5.0.0/16 19.997 0.012 3 valid 1.2.0.1 1.5.0.1

1.4.0.0/16 20.037 0.032 0 valid 1.2.0.1 1.4.0.1

1.9.0.0/16 20.028 0.023 4 valid 1.2.0.1 1.9.0.1

1.7.0.0/16 20.058 0.062 1 valid 1.2.0.1 1.7.0.1

1.6.0.0/16 20.062 0.054 3 valid 1.2.0.1 1.5.0.1 1.6.0.1

1.8.0.0/16 20.082 0.045 3 valid 1.2.0.1 1.5.0.1 1.8.0.1

1.3.0.0/16 20.051 0.046 4 valid 1.2.0.1 1.9.0.1 1.3.0.1

1.1.0.0/16 20.192 0.063 4 valid 1.2.0.1 1.9.0.1 1.6.0.1 1.1.0.1

Forwarding table (28.392)

1.5.0.0/16 3

1.4.0.0/16 0

1.9.0.0/16 4

1.7.0.0/16 1

1.6.0.0/16 3

1.8.0.0/16 3

1.3.0.0/16 4

1.1.0.0/16 4

0.0.0.0/0 0

Router link cost statistics

peerIp count avgCost minCost maxCost

1.4.0.1 26 0.032 0.031 0.032

1.7.0.1 26 0.062 0.061 0.063

1.8.0.1 26 0.052 0.051 0.052

1.5.0.1 26 0.012 0.011 0.012

1.9.0.1 26 0.023 0.021 0.024

SrcSnk statistics

destIp count avgDelay minDelay maxDelay

1.6.0.1 260 0.078 0.051 0.122

1.1.0.1 254 0.079 0.058 0.120

1.7.0.1 240 0.078 0.057 0.120

1.9.0.1 232 0.057 0.020 0.120

1.8.0.1 266 0.068 0.040 0.121

1.5.0.1 236 0.046 0.009 0.119

1.4.0.1 254 0.062 0.029 0.120

1.3.0.1 258 0.070 0.038 0.123

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

*The following routers matches the shortes path: r1, r3, r4, r5, r8, r9.*

*These do not match: r6, r7.*

*This is due to the fact that when we get a new route advertised to us, we check if it’s cost is less than 0.9 the cost of the old route, this can cause paths that are just a little bit more expensive to be chosen over less priced paths.*

*We will show an example where this could happened with r6 and r7, and their initial costs (this is even more likely to happened when the network is full of traffic).*

*The path to r6 that our program chose costs initialy*

*< , where 0.002 is the an additional price that is approximetly added in every hop (this is due to queueing and processing delay, in the first script this delay was exactly 2ms, because there is much more traffic in this script, there will be a larger delay).*

*So if our program received the path through r5 first it will be chosen over the cheapest path through r9.*

1. (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 10.131

protocol=2 ttl=99 srcAdr=1.8.0.1 destAdr=1.5.0.1

RPv0

type: advert

pathvec: 1.7.0.0/16 10.073 0.022 1.8.0.1 1.7.0.1

/192.168.2.3:31313 received from /192.168.2.4:31313 at 10.180

protocol=2 ttl=99 srcAdr=1.2.0.1 destAdr=1.5.0.1

RPv0

type: advert

pathvec: 1.7.0.0/16 10.117 0.062 1.2.0.1 1.7.0.1

/192.168.2.3:31313 received from /192.168.1.1:31313 at 10.320

protocol=2 ttl=99 srcAdr=1.9.0.1 destAdr=1.5.0.1

RPv0

type: advert

pathvec: 1.7.0.0/16 10.257 0.118 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.

*r8, r2, and r9 sent advertisements of r7 to router r5.*

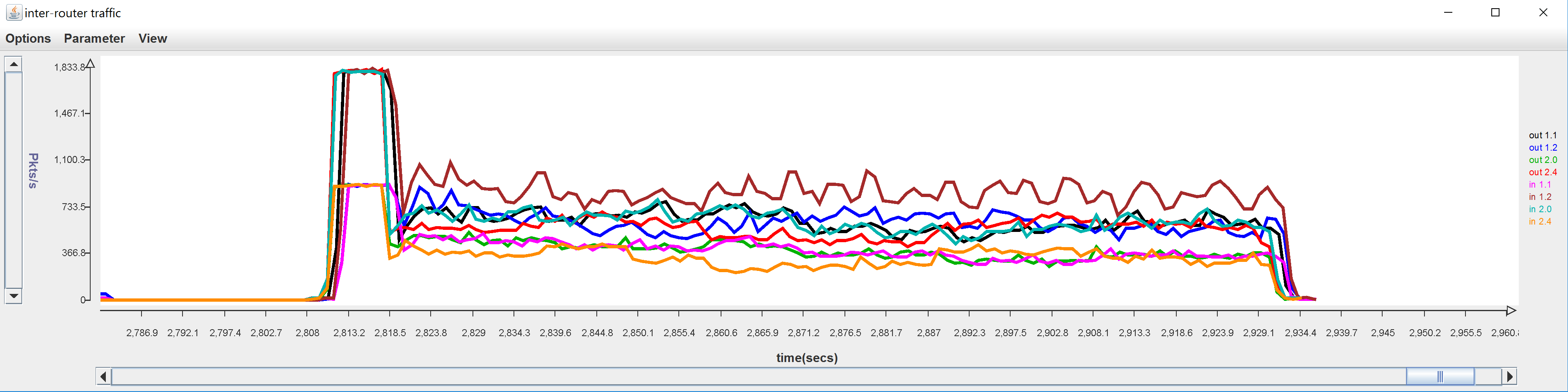
*This happened because all of these options were shortest (cheapest) paths at those routers, we only advertise a path that is cheaper (there are some other options but they are not relevant to this senerio like if a link is invalid or if 20 seconds have passed etc.)*

*r6 did not send an advertisment because it received an advertisment for r7 from r5 (we checked this in the full log), and then r6 had a path that is cheaper to r7 than the other advertisment that he got for r7, so he did not pass non-relevant advertisments to r5.*

1. (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.

kaiashk@onlusr:~/cse473s-f19-students-gai.ashkenazy/lab5/net3$ grep "1.1.0.0.16....." log3\_\*

log3\_2:1.1.0.0/16 10.201 0.253 4 valid 1.2.0.1 1.9.0.1 1.3.0.1 1.1.0.1

log3\_2:1.1.0.0/16 30.085 0.411 4 valid 1.2.0.1 1.9.0.1 1.3.0.1 1.1.0.1

**log3\_2:1.1.0.0/16 40.055 0.356 4 valid 1.2.0.1 1.9.0.1 1.1.0.1**

**log3\_2:1.1.0.0/16 80.196 0.406 4 valid 1.2.0.1 1.9.0.1 1.3.0.1 1.1.0.1**

**log3\_2:1.1.0.0/16 100.089 0.382 4 valid 1.2.0.1 1.9.0.1 1.1.0.1**

log3\_2:1.1.0.0/16 120.047 0.332 4 valid 1.2.0.1 1.9.0.1 1.1.0.1

log3\_3:1.1.0.0/16 10.064 0.049 1 valid 1.3.0.1 1.1.0.1

log3\_3:1.1.0.0/16 30.033 0.114 1 valid 1.3.0.1 1.1.0.1

log3\_3:1.1.0.0/16 40.012 0.178 1 valid 1.3.0.1 1.1.0.1

log3\_3:1.1.0.0/16 110.114 0.250 1 valid 1.3.0.1 1.1.0.1

log3\_3:1.1.0.0/16 120.135 0.253 1 valid 1.3.0.1 1.1.0.1

log3\_4:1.1.0.0/16 10.256 0.320 1 valid 1.4.0.1 1.2.0.1 1.9.0.1 1.3.0.1 1.1.0.1

**log3\_4:1.1.0.0/16 20.163 0.262 2 valid 1.4.0.1 1.3.0.1 1.1.0.1**

log3\_4:1.1.0.0/16 30.464 0.431 2 valid 1.4.0.1 1.3.0.1 1.1.0.1

log3\_4:1.1.0.0/16 40.054 0.418 2 valid 1.4.0.1 1.3.0.1 1.1.0.1

**log3\_4:1.1.0.0/16 60.248 0.491 1 valid 1.4.0.1 1.2.0.1 1.9.0.1 1.1.0.1**

**log3\_4:1.1.0.0/16 70.083 0.414 2 valid 1.4.0.1 1.3.0.1 1.1.0.1**

log3\_4:1.1.0.0/16 100.514 0.438 2 valid 1.4.0.1 1.3.0.1 1.1.0.1

log3\_4:1.1.0.0/16 110.223 0.526 2 valid 1.4.0.1 1.3.0.1 1.1.0.1

log3\_4:1.1.0.0/16 120.566 0.522 2 valid 1.4.0.1 1.3.0.1 1.1.0.1

log3\_5:1.1.0.0/16 10.410 0.575 3 valid 1.5.0.1 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 10.429 0.358 2 valid 1.5.0.1 1.9.0.1 1.3.0.1 1.1.0.1

log3\_5:1.1.0.0/16 10.444 0.132 0 valid 1.5.0.1 1.6.0.1 1.1.0.1

log3\_5:1.1.0.0/16 40.496 0.320 0 valid 1.5.0.1 1.6.0.1 1.1.0.1

log3\_5:1.1.0.0/16 50.494 0.332 0 valid 1.5.0.1 1.6.0.1 1.1.0.1

log3\_5:1.1.0.0/16 120.179 0.371 0 valid 1.5.0.1 1.6.0.1 1.1.0.1

log3\_6:1.1.0.0/16 10.197 0.047 0 valid 1.6.0.1 1.1.0.1

log3\_6:1.1.0.0/16 20.331 0.120 0 valid 1.6.0.1 1.1.0.1

log3\_6:1.1.0.0/16 80.282 0.167 0 valid 1.6.0.1 1.1.0.1

log3\_6:1.1.0.0/16 90.442 0.243 0 valid 1.6.0.1 1.1.0.1

log3\_6:1.1.0.0/16 110.316 0.282 0 valid 1.6.0.1 1.1.0.1

log3\_6:1.1.0.0/16 120.157 0.257 0 valid 1.6.0.1 1.1.0.1

log3\_7:1.1.0.0/16 10.283 0.412 2 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\_7:1.1.0.0/16 10.991 0.309 0 valid 1.7.0.1 1.8.0.1 1.5.0.1 1.6.0.1 1.1.0.1

log3\_7:1.1.0.0/16 20.965 0.507 0 valid 1.7.0.1 1.8.0.1 1.5.0.1 1.6.0.1 1.1.0.1

**log3\_7:1.1.0.0/16 50.414 0.543 1 valid 1.7.0.1 1.2.0.1 1.9.0.1 1.1.0.1**

**log3\_7:1.1.0.0/16 80.541 0.548 2 valid 1.7.0.1 1.4.0.1 1.3.0.1 1.1.0.1**

**log3\_7:1.1.0.0/16 120.276 0.513 1 valid 1.7.0.1 1.2.0.1 1.9.0.1 1.1.0.1**

log3\_8:1.1.0.0/16 10.330 0.476 1 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\_8:1.1.0.0/16 10.776 0.244 0 valid 1.8.0.1 1.5.0.1 1.6.0.1 1.1.0.1

log3\_8:1.1.0.0/16 20.603 0.369 0 valid 1.8.0.1 1.5.0.1 1.6.0.1 1.1.0.1

log3\_8:1.1.0.0/16 40.619 0.457 0 valid 1.8.0.1 1.5.0.1 1.6.0.1 1.1.0.1

log3\_8:1.1.0.0/16 90.959 0.639 0 valid 1.8.0.1 1.5.0.1 1.6.0.1 1.1.0.1

log3\_8:1.1.0.0/16 120.755 0.548 0 valid 1.8.0.1 1.5.0.1 1.6.0.1 1.1.0.1

log3\_9:1.1.0.0/16 10.120 0.141 3 valid 1.9.0.1 1.3.0.1 1.1.0.1

log3\_9:1.1.0.0/16 20.083 0.197 3 valid 1.9.0.1 1.3.0.1 1.1.0.1

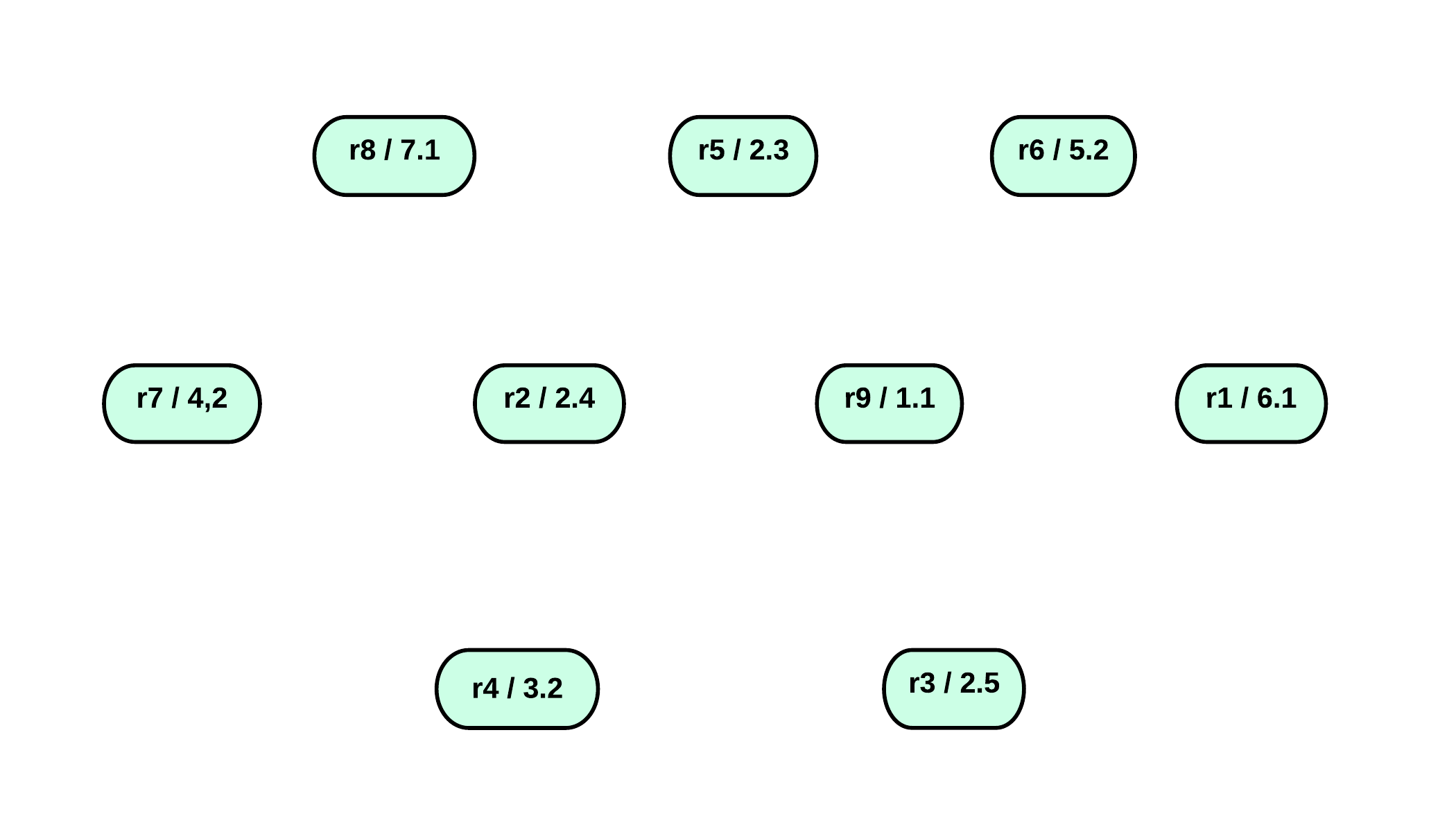
**log3\_9:1.1.0.0/16 40.014 0.203 1 valid 1.9.0.1 1.1.0.1**

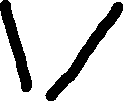
**log3\_9:1.1.0.0/16 80.109 0.233 3 valid 1.9.0.1 1.3.0.1 1.1.0.1**

**log3\_9:1.1.0.0/16 100.045 0.221 1 valid 1.9.0.1 1.1.0.1**

log3\_9:1.1.0.0/16 120.019 0.184 1 valid 1.9.0.1 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.

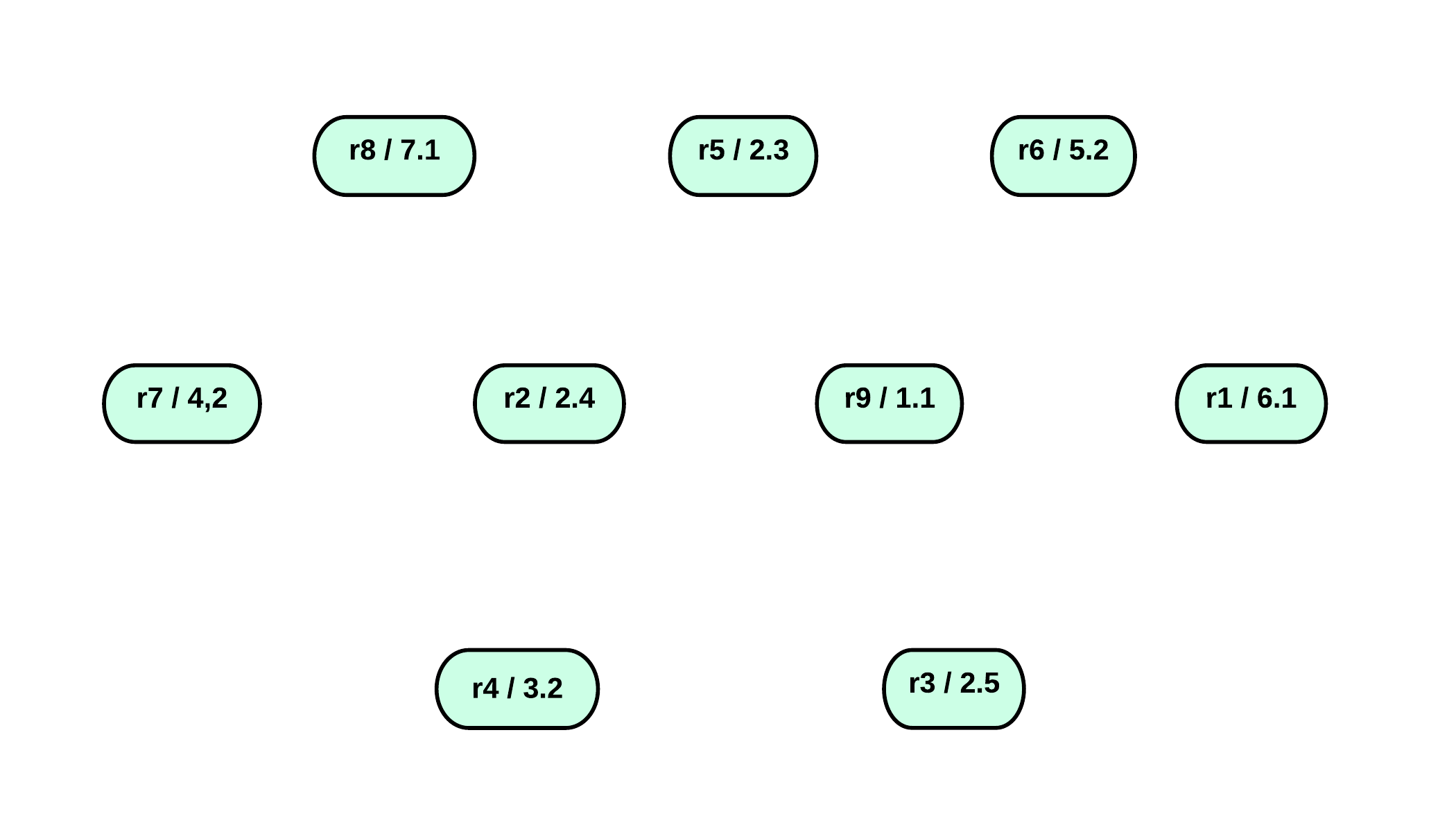
**

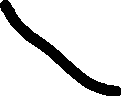


These are the shortes paths from all routers to r1, all the other routes and output links are in the drawing above (page 25).

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 \_101\_ to time \_108\_.*

**



These are the shortes paths from all routers to r1, all the other routes and output links are in the drawing above (page 25).