Code of dynamic source routing:

void forwardRouteReply(string source\_ip, string destination\_ip,

unsigned long long int packet\_id, string route\_list)

{

vector<string> ip\_list = StringOperations::split(route\_list, ","

);

int self\_pos = -1;

int count = ip\_list.size();

int i;

for(i = 0; i < ip\_list.size() - 1; i++)

{

if(ip\_list[i] == ip\_list[i + 1])

{

ip\_list.erase(ip\_list.begin() + i + 1);

}

}

for(i = 0; i < count; i++)

{

if(ip\_list[i] == ip)

{

self\_pos = i;

break;

}

}

for(i = 0; i < self\_pos; i++)

{

if(ip\_list[i] != ip && ip\_list[self\_pos - 1] != ip)

{

cache.updateRoute(ip\_list[i], ip\_list[self\_pos - 1]);

}

}

for(i = self\_pos + 1; i < count; i++)

{

if(ip\_list[i] != ip && ip\_list[self\_pos + 1] != ip)

{

cache.updateRoute(ip\_list[i], ip\_list[self\_pos + 1]);

}

}

if(destination\_ip != ip) // Or self\_pos != 0

{

string arr[] =

{

string(ROUTE\_REPLY), source\_ip, destination\_ip,

StringOperations::to\_string(packet\_id), route\_list

};

string message = StringOperations::join(vector<string>(arr, arr

+5), DELIMITER);

sendDataImmediate(ip\_list[self\_pos - 1], message);

return;

}

else

{

if(packet\_queues.find(source\_ip) != packet\_queues.end())

{

for(int i = 0; i < packet\_queues[source\_ip].size(); i++)

{

pthread\_t th;

pthread\_create(&th, NULL, sendDataPacket, packet\_queues[

source\_ip][i]);

}

}

}

}

void forwardRouteRequest(string source\_ip, string destination\_ip,

unsigned long long int packet\_id, string route\_list)

{

pair<string, unsigned long long int> pkt\_info = make\_pair(

source\_ip, packet\_id);

if(processed\_packets.find(pkt\_info) != processed\_packets.end())

// Already processed

{

return;

}

processed\_packets.insert(pkt\_info);

if(ip == destination\_ip)

{

sendRouteReply((route\_list == "")? ip: ((route\_list.find(ip) ==

string::npos)? route\_list + "," + ip: route\_list));

return;

}

if(cache.isRouteCached(destination\_ip))

{

if(route\_list == "")

{

route\_list = ip;

}

else if(route\_list.find(ip) == string::npos)

{

route\_list += "," + ip;

}

if(route\_list.find(cache.fetchRoute(destination\_ip)) == string

::npos)

{

route\_list += "," + cache.fetchRoute(destination\_ip);

}

#ifdef DEBUG

cerr << "Path is taken from cache: " << route\_list << endl;

#endif

sendRouteReply(route\_list);

return;

}

if(route\_list == "")

{

route\_list = ip;

}

else if(route\_list.find(ip) == string::npos)

{

route\_list = route\_list + "," + ip;

}

broadcastRouteRequest(ROUTE\_REQUEST, source\_ip, destination\_ip,

packet\_id, route\_list);

}

void forwardData(Data \*data, string sender\_ip)

{

pair<string, unsigned long long int> pkt\_info = make\_pair(data->

getPacket()->getSource(), data->getPacket()->getPacketID());

sendDataImmediate(sender\_ip, string(MAC\_ACK) + DELIMITER +

pkt\_info.first + DELIMITER + data->getPacket()->

getPacketIDString() + DELIMITER + data->getPacket()->

getDestination());

if(processed\_packets.find(pkt\_info) == processed\_packets.end())

{

processed\_packets.insert(pkt\_info);

if(data->getPacket()->getDestination() == ip)

{

cout << "Message from " << data->getPacket()->getSource() <<

": ’" << data->getPacket()->getContent() << "’" << endl

<< PROMPT\_STRING;

fflush(stdout);

if(cache.isRouteCached(pkt\_info.first))

{

sendDataImmediate(cache.fetchRoute(pkt\_info.first), string(

UDP\_ACK) + DELIMITER + pkt\_info.first + DELIMITER +

data->getPacket()->getPacketIDString() + DELIMITER + ip

);

}

else

{

sendRouteRequest(pkt\_info.first);

}

return;

}

else

{

addToQueue(data);

if(cache.isRouteCached(data->getPacket()->getDestination()))

{

pthread\_t th;

pthread\_create(&th, NULL, sendDataPacket, data);

}

else

{

char id[25];

sprintf(id, "%llu", next\_packet\_id++);

sendDataImmediate(cache.fetchRoute(data->getPacket()->

getSource()), string(ROUTE\_ERROR) + DELIMITER + ip +

DELIMITER + string(id) + DELIMITER + data->getPacket()

->getSource() + DELIMITER + ip);

}

}

}

else

{

return;

}

}

void forwardUACK(string orig\_src, string orig\_id, string orig\_dest

)

{

if(orig\_src == ip)

{

unsigned long long int id = atoll(orig\_id.c\_str());

cout << "Message with id " << orig\_id << " delivered to " <<

orig\_dest << endl << PROMPT\_STRING;

fflush(stdout);

for(int i = 0; i < packet\_queues[orig\_dest].size(); i++)

{

if(packet\_queues[orig\_dest][i]->getPacket()->getPacketID() ==

id)

{

packet\_queues[orig\_dest][i]->acknowledge();

packet\_queues[orig\_dest][i]->in\_queue = false;

packet\_queues[orig\_dest].erase(packet\_queues[orig\_dest].

begin() + i);

break;

}

}

}

else

{

if(cache.isRouteCached(orig\_src)) // To avoid pointer error.

{

sendDataImmediate(cache.fetchRoute(orig\_src), string(UDP\_ACK)

+ DELIMITER + orig\_src + DELIMITER + orig\_id + DELIMITER

+ orig\_dest);

}

}

}

void forwardRouteError(string rerr\_src, unsigned long long int

rerr\_pkt\_id, string data\_src, string path)

{

pair<string, unsigned long long int> pkt\_info = make\_pair(

rerr\_src, rerr\_pkt\_id);

if(processed\_packets.find(pkt\_info) != processed\_packets.end())

// Already processed

{

return;

}

processed\_packets.insert(pkt\_info);

vector<string> ip\_list = StringOperations::split(path, ",");

cache.removePath(ip\_list[0]);

for(int i = 0; i < ip\_list.size(); i++)

{

cache.updateRoute(ip\_list[i], ip\_list[0]);

}

if(ip != data\_src)

{

char id[25];

sprintf(id, "%llu", rerr\_pkt\_id);

sendDataImmediate(cache.fetchRoute(data\_src), string(

ROUTE\_ERROR) + DELIMITER + rerr\_src + DELIMITER + string(id

) + DELIMITER + data\_src + DELIMITER + ip + "," + path);

}

Link for this code :

<http://www.cse.iitd.ac.in/~mcs142144/documents/DSR_thesis.pdf>

explanation is also provided in it.