LULEÅ UNIVERITY OF TECHNOLOGY

COMPUTER SCIENCE

D0021E

MOBILE IP

Author
S. JONSSON
setjon-7@student.ltu.se
M. LARSSON ANDERSSON
magany-7@student.ltu.se
M. JONSSON
micjon-5@student.ltu.se

Supervisor
A. Prof. KARAN MITRA
SAGUNA SAGUNA

March 20, 2020

1 Introduction

In the fourth lab we were supposed to learn about a network layer mobility managment protocol through implementing a Mobile IP home agent that registers current location for a mobile node.

2 Methods

In the simulator we implemented

- **Home Agent** that all traffic between correspondent node and a mobile node traversed through and registred the mobile nodes current location.
- **IPv6 Stateless Address Autoconfiguration** For configuring addresses for nodes connecting to a router
- **Node migration between routers**
- **Advertisement and Solicitation**
- **Triangle routing problem**

2.1 Handoff process

As can be seen in fig. 1 our network is built upon 2 routers and 2 hosts, referenced as nodes in this paper. all devices in the network is identified by it's IP address which is constructed such that "<Network>.<Suffix>".

In Fig. 1 Node 0.1 is intending to move to a new network, in this case being network 1. To accomplish this Node 0.1 sends a solicitation request to Router 1.0, after receiving an advertisement from Router 1.0 Node 0.1 begins the IPV6 autoconfiguration process.

The autoconfiguration process is built on the principle that the nodes autogenerate an IP and proceeds to ask the Router if the IP is unique. This process is repeated until the nodes ip is unique in which case it starts using it as a "Care of" address. Node 0.1 will henceforth be referenced to as Node 1.2, as that is its new address in Fig. 2.

After establishing a CO address Node 1.2 sends a binding update message to it's home network, which is Router 0.0 and waits for a Binding Acknowledgement before sending any more messages. Upon receiving a BUM Router 0.0 updates the routing table of it's home agent, which in Fig. 2 is symbolized as a node but it is just a module in the router software. The HA handles forwarding information for nodes that have migrated to new addresses, so in our case the HA routing table will look like Table 4. After binding the original IP to the CO IP the router sends a Binding Acknowledgement to CO 1.2, upon receiving the BA from Router 0.0 CO 1.2 sends a newAddress message to 1.1 informing it of it's new address and thus the triangle routing problem is solved.

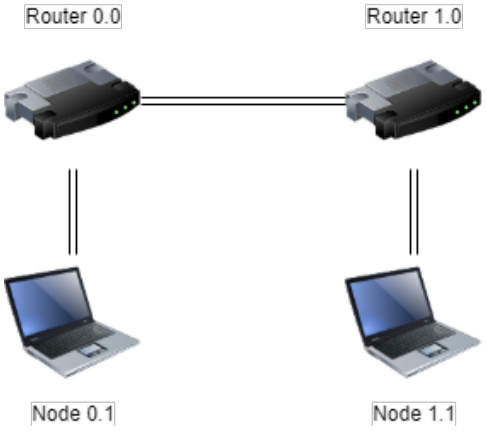

Figure 1: Migration step 1

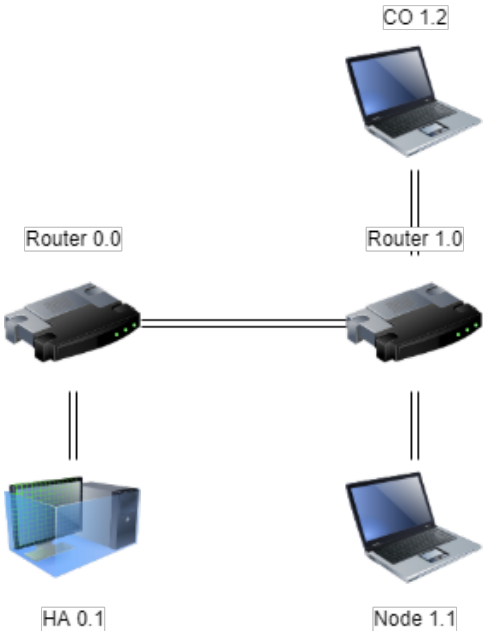

Figure 2: Migration step 2

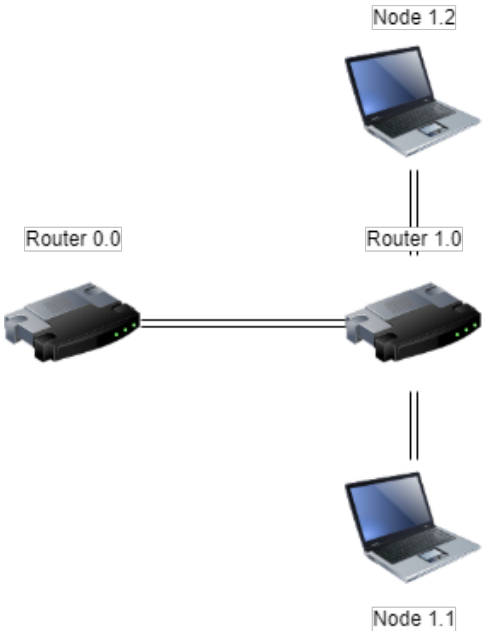

Figure 3: Migration step 3

Original IP	CO IP
0.1	1.2

Figure 4: Home agent routing table

2.2 Code

```
1
2  public class Router extends SimEnt{
3
4    Router(int interfaces, int networkId) {
5      _routingTable = new RouteTableEntry[interfaces];
6      _interfaces=interfaces;
7      setNetworkAddr(networkId, 0);
8      _HA = new HomeAgent(this);
9    }
10
11   public void advertise() {
12     System.out.println(this.toString() + " Sending advertisement to all connected interfaces:");
13     for(RouteTableEntry addr : _routingTable) {
14       if(addr == null)
15         continue;
16       System.out.println(this.toString() + " Sending advertisement to " +
               addr.networkAddress.toString());
17       send (addr.link, new Advertisement(this._id, addr.networkAddress, 0), _now);
18     }
19   }
20
21   public int getEmptyInterface() {
22     for(int i = 0; i<_routingTable.length; i++) {
23       if(_routingTable[i] == null) {
24         return i;
25       }
26     }
27     return -1;
28   }
29   // This method connects links to the router and also informs the
30   // router of the host connects to the other end of the link
31   public void connectInterface(int interfaceNumber, SimEnt link, SimEnt node) {
32     if(!uniqueAddress(node._id)) {
33       System.out.println("IP Address is not unique");
34       return;
35     }
36     if (interfaceNumber<_interfaces && _routingTable[interfaceNumber] == null) {
37       _routingTable[interfaceNumber] = new RouteTableEntry(node._id, link);
38       System.out.println(node.toString() + " Successfully connected to " + this.toString());
39     }
40     else {
41       System.out.println("Trying to connect to port not in router");
42       return;
43     }
44     ((Link) link).setConnector(this);
45   }
46   /**
47    * Checks whether the IP Address already exists in this router
48    * @param nA NetworkAddr to check
49    * @return true if address is unique, false if address is not
50    */
51   private boolean uniqueAddress(NetworkAddr nA) {
52     if(nA.nodeId() == 0 && nA.networkId() == getAddr().networkId())
53       return false;
54     for(RouteTableEntry addr : _routingTable) {
55       if(addr == null)
56         continue;
57       if(addr.networkAddress.SameAddress(nA))
58         return false;
59     }
60     return true;
61   }
```

Figure 5: Router part 1/2

```
1  // When messages are received at the router this method is called
2  public void recv(SimEnt source, Event event) {
3    if(event instanceof Migrate) {
4      System.out.println("Router attempts to change interface for node " +((Migrate)
           event).source().networkId() + " to  interface " +((Migrate) event).newInterface());
5      ((Migrate) event).newLink(moveInterface(((Migrate) event).source() ,((Migrate)
           event).newInterface()));
6      send(getInterface(((Migrate) event).source()),event,0);
7    }
8    else if(event instanceof Disconnect) {
9      Link link = (Link) removeFromInterface(((Disconnect) event).source());
10     link.removeConnector(this);
11   }
12   else if(event instanceof UniqueAddr) {
13     System.out.println(toString() +" handles a request to test if address is unique");
14     ((UniqueAddr)event).setUnique(uniqueAddress(((UniqueAddr)event).getAddr()));
15     send(source, event, 0);
16   }
17   else if(event instanceof Solicit) {
18     System.out.println(this.toString() + " Received Solicitation request from Node " +
           ((Solicit)event).source());
19     advertise();
20   }
21   else if(event instanceof Advertisement) {
22     System.out.println(this.toString() + " Received Advertisement from Router " +
           ((Advertisement)event).source());
23   }
24   else if(event instanceof BindingRequest) {
25     BindingRequest br = (BindingRequest) event;
26     if(br.destination().SameAddress(getAddr())) {
27       System.out.println(this.toString()+ " received a binding request");
28       send(_HA, br, 0);
29     }
30     else {
31       System.out.println("Router " + this.getAddr().networkId() + "." + this.getAddr().nodeId()
             +  " handles packet with seq: " + ((Message) event).seq()+" from node: "+((Message)
             event).source().networkId()+"." + ((Message) event).source().nodeId() );
32       SimEnt sendNext = getInterface(br.destination());
33       System.out.println("Router sends to node: " + ((Message)
             event).destination().networkId()+"." + ((Message) event).destination().nodeId());
34       send (sendNext, event, _now);
35     }
36   }
37   else if (event instanceof Message) {
38     System.out.println("Router " + this.getAddr().networkId() + "." + this.getAddr().nodeId() +
           " handles packet with seq: " + ((Message) event).seq()+" from node: "+((Message)
           event).source().networkId()+"." + ((Message) event).source().nodeId() );
39     SimEnt sendNext = getInterface(((Message)  event).destination());
40     System.out.println("Router sends to node: " + ((Message)
           event).destination().networkId()+"." + ((Message) event).destination().nodeId());
41     send (sendNext, event, _now);
42   }
43 }
44 }
```

Figure 6: Router part 2/2

```
public class Node extends SimEnt {

  public Node (int network, int node) {
    super();
    setNetworkAddr(network, node);
  }

  private int swapInterfaceAfter = 0;
  private int swapRouterAfter = 0;
  private int swapTo;
  private Router _newRouter;
  /**
   * After non zero number of messages it will attempt swap to given interface
   * @param numberOfMessages
   * @param swapToInterface
   */
  public void changeInterfaceAfter(int numberOfMessages, int swapToInterface) {
    swapInterfaceAfter= numberOfMessages;
    swapTo = swapToInterface;
  }

  public void changeRouterAfter(int numberOfMessages, Router newRouter) {
    _newRouter = newRouter;
    swapRouterAfter = numberOfMessages;
  }

  public void sendSolicitationRequest() {
    System.out.println(this.toString() + " sends a solicitation request");
    send(_peer, new Solicit(this._id, 0), 0);
  }

  public void recv(SimEnt src, Event ev) {
    if (ev instanceof TimerEvent) {
      if (setup && _stopSendingAfter > _sentmsg && ((_sentmsg != swapRouterAfter&&_sentmsg !=
          swapInterfaceAfter)||_sentmsg == 0)) {
        _sentmsg++;
        send(_peer, new Message(_id, new NetworkAddr(_toNetwork, _toHost),_seq),0);
        send(this, new TimerEvent(),_timeBetweenSending);
        System.out.println("Node "+_id.networkId()+ "." + _id.nodeId() +" sent message with seq:
            "+_seq + " at time "+SimEngine.getTime());
        _seq++;
      }
      else if(setup && _sentmsg == swapInterfaceAfter) {
        _sentmsg++;
        System.out.println("Node "+_id.networkId()+ "." + _id.nodeId() + " sends a request to
            change interface to interface " + swapTo);
        send(_peer, new Migrate(getAddr(), swapTo),0);
        send(this, new TimerEvent(), _timeBetweenSending);
      }
      else if(setup && _sentmsg == swapRouterAfter{
        _sentmsg++;
        send(_peer,new Disconnect(getAddr()), 0);
        Link newLink = new Link();
        newLink.setConnector(this);
        _peer = newLink;
        int i = _newRouter.getEmptyInterface();
        if(i>=0) {
          _newRouter.connectInterface(i, newLink, this);
        }
        setup = false;
        sendSolicitationRequest();
      }
    }
    else if(ev instanceof ProvideNewAddr) {
      ProvideNewAddr pna = (ProvideNewAddr)ev;
      _toNetwork = pna.source().networkId();
      _toHost = pna.source().nodeId();
    }
```

Figure 7: Node part 1/2

```
    else if(ev instanceof Migrate) {
      Migrate mig = (Migrate)ev;
      if(mig.getNewLink() != null) {
        _peer = mig.getNewLink();
        System.out.println("Node migrated to new interface");
      }
      else
        System.out.println("Node did not migrate");
    }
    else if(ev instanceof UniqueAddr) {
      if(((UniqueAddr) ev).isUnique()) {
        NetworkAddr addr = ((UniqueAddr) ev).getAddr();
        oldAddress = new NetworkAddr(_id.networkId(), _id.nodeId());
        _id.updateAddr(addr.networkId(), addr.nodeId());
        send(_peer, new BindingRequest(_id, new NetworkAddr(oldAddress.networkId(), 0), _sentmsg,
              oldAddress),0);
        _sentmsg++;
        System.out.println("Node is ready to communicate with new address " +toString());
        setup = true;
      }
      else {
        System.out.println(toString() +" did not find unique address testing new address");
        ((UniqueAddr)ev).getAddr().incrementAddr();
        send(_peer, ev, 0);
      }
    }
    else if(ev instanceof BindingAck) {
      System.out.println("Binding complete");
    }
    else if(ev instanceof Advertisement) {
      if(!setup) {
        NetworkAddr addr = new
              NetworkAddr(((Advertisement)ev).source().networkId(),((Advertisement)ev).source().networkId()+1
              );
        send(_peer, new UniqueAddr(addr), 0);
      }
      System.out.println(this.toString() + " Received Advertisement from Router " +
            ((Advertisement)ev).source());
    }
    else if (ev instanceof Message) {
      System.out.println("Node "+_id.networkId()+ "." + _id.nodeId() +" receives message with seq:
            "+((Message) ev).seq() + " at time "+SimEngine.getTime());
    }
    if(ev instanceof ForwardMessage) {
      ForwardMessage fm = (ForwardMessage) ev;
      send(_peer, new ProvideNewAddr(_id,fm.getOriginalSource(),_seq) , 0);
      System.out.println("Send new address to other host");
    }
  }
}
```

Figure 8: Node part 2/2

```java
public class HomeAgent extends SimEnt {

  private HashMap<NetworkAddr, NetworkAddr> remoteNodes; //Remote address as value with key as
       localAddress
  private Router _router;

  public HomeAgent(Router router) {
    _router = router;
    this._id = router.getAddr();
    remoteNodes = new HashMap<NetworkAddr, NetworkAddr>();
  }
  /**
   * Checks if given nodeId is used in routing to a host on a remote network
   * @param nodeId
   * @return if nodeId in homeAgent table of remote Address or not
   */
  public boolean inHomeAgent(int nodeId) {
    for(NetworkAddr addr: remoteNodes.keySet() ) {
      if(addr.nodeId() == nodeId) {
        return true;
      }
    }
    return false;
  }

  public boolean hasBeenConnected(NetworkAddr remoteAddr) {
    return remoteNodes.containsValue(remoteAddr);
  }

  public NetworkAddr getRemoteAddress(NetworkAddr localAddress) {
    return remoteNodes.get(localAddress);
  }

  private void removeAddr(NetworkAddr remoteAddr) {
    remoteNodes.remove(remoteAddr);
  }

  private void setRemoteNode(NetworkAddr localAddr, NetworkAddr remoteAddr) {
    remoteNodes.put(localAddr, remoteAddr);
  }

  @Override
  public void recv(SimEnt src, Event ev) {
    if(ev instanceof BindingRequest) {
      BindingRequest br = (BindingRequest)ev;
      System.out.println("HomeAgent of "+ _router.toString() +" setting up new binding to node " +
           br.source() );
      setRemoteNode(br.localAddr(), br.source());
      send(_router, new BindingAck(_router.getAddr(), br.source(), 0), 0);
    }
    else if(remoteNodes.isEmpty()) {
      System.out.println("Homeagent dropped packet no host on foreign network");
    }
    else if(ev instanceof Message) {
      Message msg = (Message)ev;
      System.out.println("Homeagent Forwarding message");

      send(_router, new ForwardMessage(this.getAddr(), getRemoteAddress(msg.destination()),
           msg.seq(), msg.source()), 0);
    }
  }
}
```

Figure 9: HomeAgent

3 Results

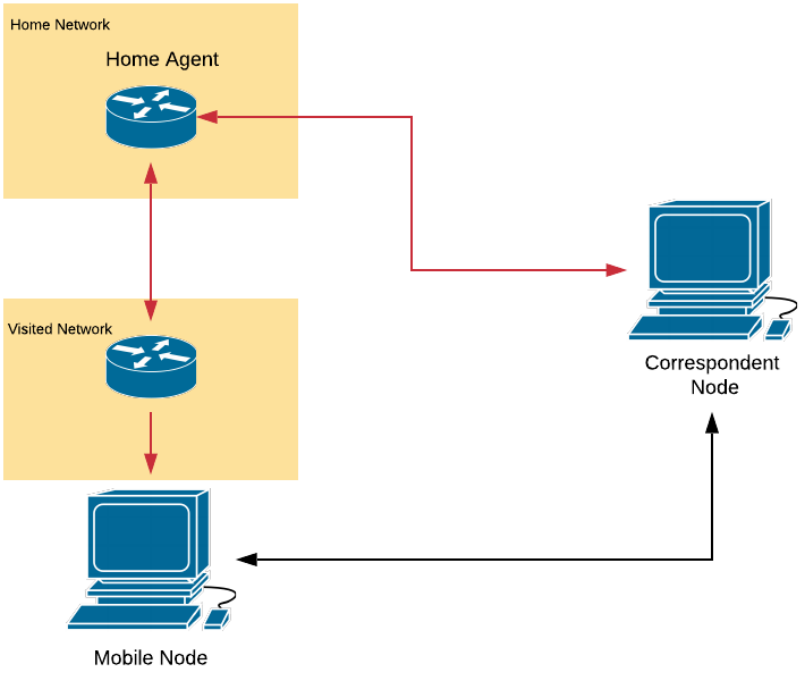

Figure 10: *Chart showing the overall picture for the laboration*

```
1  Node 0.1 Successfully connected to Router 1.0
2  Node 0.1 sends a solicitation request
3  Router 1.0 handles packet with seq: 5 from node: 1.1
4  Router sends to node: 0.1
5  Router 1.0 Received Solicitation request from Node 0.1
6  Router 1.0 Sending advertisement to all connected interfaces:
7  Router 1.0 Sending advertisement to 0.0
8  Router 1.0 Sending advertisement to 1.1
9  Router 1.0 Sending advertisement to 0.1
10 Node 0.1 receives message with seq: 5 at time 5.0
11 Router 0.0 Received Advertisement from Router 1.0
12 Node 1.1 Received Advertisement from Router 1.0
13 Node 0.1 Received Advertisement from Router 1.0
14 Router 1.0 handles a request to test if address is unique
15 Node is ready to communicate with new address Node 1.2
16 Router 1.0 handles packet with seq: 6 from node: 1.2
17 Router sends to node: 0.0
18 Router 0.0 received a binding request
19 HomeAgent of Router 0.0 setting up new binding to node 1.2
20 Router 0.0 handles packet with seq: 0 from node: 0.0
21 Router sends to node: 1.2
22 Router 1.0 handles packet with seq: 0 from node: 0.0
23 Router sends to node: 1.2
24 Binding complete
25 Node 1.1 sent message with seq: 6 at time 6.0
26 Router 1.0 handles packet with seq: 6 from node: 1.1
27 Router sends to node: 0.1
28 Router 0.0 handles packet with seq: 6 from node: 1.1
29 Router sends to node: 0.1
30 Homeagent Forwarding message
```

Figure 11: Results in console when running the simulator

As shown in Fig. 11 on row 1, the node has successfully been connected to a new router and therefore sends a solicitation request. On row 5 you can see that the router has received the request and then sends an advertisement to all the connected interfaces. Then the router checks if the nodes address is unique and if there is no problem, node is ready to receive messages but not communicate with its new address. On row 18, the router receives a binding request from the same node and the homeagent for that router starts the binding. The binding is completed at row 24 and can now communicate with itself.

4 Discussion

Something we realised at the end of the laboration was that we should have planned the project beforehand. Many problems occured that could have been avoided if we had a clear picture of how we should implement all the functionalities from the very beginning.

Our IPV6 autoconfiguration is not entirely correct, instead of asking connected nodes for collisions we simply let the router handle the lookup process.