Додаток А. Лістинг моделюючого програмного забезпечення

```
package app.simulator;

public class Link implements Serializable{
	private static final long serialVersionUID = 1L;
	public mxCell edge;
	public static int packetcount=0;
	public static int routingpacketcount=0;
	List<Packet> PacketQueue = new LinkedList<Packet>();
	boolean removed= false;
	boolean Active = false;

	BlockingQueue<Packet> SendingQueue = new LinkedBlockingQueue<Packet>();
	BlockingQueue<Packet> ReceivingQueue = new
			LinkedBlockingQueue<Packet>();
	BlockingQueue<Packet> DataQueue = new LinkedBlockingQueue<Packet>();
	Packet p1,p2;
	static boolean GraphSynchro=false;
	private void ThreadTiming(Packet p) throws InterruptedException{
		if(GraphEditor.graph.getView()==null)System.out.println("f..!");
		if(edge==null)System.out.println("f..!");

		}
		while(Simulator.isPaused){
			Thread.sleep(200);
		}
		boolean isUpdater=false;
		synchronized(GraphEditor.graph){
			if(GraphSynchro==false){
				GraphSynchro=true;
				isUpdater=true;
			}
		}
		Thread.sleep(Restrictions.minPacketTransmitTime*
			Restrictions.Scale);
		synchronized(GraphEditor.graph){
			if(isUpdater){
				GraphSynchro=false;
				isUpdater=false;
				GraphEditor.graph.refresh();
			}
		}
	}
	Thread SendingThread = new Thread(new Runnable(){
		public void run(){
			while(!removed){
				try {

					p1 = SendingQueue.take();
					if (p1==null) continue;
					ThreadTiming(p1);
					Router1.receive(p1,Link.this);
					p1=null;
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		}
	});
	Thread ReceivingThread = new Thread(new Runnable(){
		public void run(){
```

```
            while(!removed){
                  try {
                        p2 = ReceivingQueue.take();
                        if (p2==null) continue;
                        ThreadTiming(p2);
                        Router2.receive(p2,Link.this);
                        p2=null;
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }
            }
      }
});
Router Router1;
Router Router2;
LinkState State;
public Router getLinkedRouter(Router Router){
      if(Router==Router1)return Router2;
      if(Router==Router2)return Router1;
      return null;
}
public Link(Router Router1,Router Router2,mxCell edge){
      this.Router1=Router1;
      this.Router2=Router2;
      Router1.registerLink(this,Router2);
      Router2.registerLink(this,Router1);
      State=LinkState.FREE;
      SendingThread.start();
      ReceivingThread.start();
}
public void setActive(){Active = true;}
public void setUnActive(){Active = false;}
public boolean isActive(){return Active;};
public void removeLink(){
      Router1.unregisterLink(Router2);
      Router2.unregisterLink(Router1);
      for(Packet p:SendingQueue){p.drop();}
      for(Packet p:ReceivingQueue){p.drop();}
      SendingQueue.clear();
      ReceivingQueue.clear();
      removed=true;
}
public void transmit(Packet packet,Router From){
      if(removed){
            System.out.println("link!");
            packet.drop();
            return;
      }
      if (packet instanceof RoutingPacket){
            synchronized(GraphEditor.graph){
                  routingpacketcount++;
            }
      }else{
            synchronized(GraphEditor.graph){
                  packetcount++;
            }
      }
      if(From==Router1)receive(packet);else
      if(From==Router2)send(packet);
}
private void send(Packet packet){SendingQueue.put(packet);}
private void receive(Packet packet){ReceivingQueue.put(packet);}
public String toString(){
      String S="";//this.Router1+"<-->"+this.Router2;
      if(p1!=null)S+="-->:"+p1+"\n";
```

```
            //if(!SendingQueue.isEmpty())S+=SendingQueue+"\n";
            if(p2!=null)S+="<--:"+p2+"\n";
            //if(!ReceivingQueue.isEmpty())S+=ReceivingQueue+"\n";
            return S;
      }
}
```

package app.simulator.packets;

```
public class Packet implements Serializable{
      public static int count=0;
      public static int activecount=0;
      public static int finalizedcount=0;
      public static int droppededcount=0;
      public int tree=1;
      Router From;
      Router To;
      public static final int BaseTTL=Restrictions.maxPathLegth;
      int TTL=BaseTTL;
      int Hops=0;
      public mxCell PacketCell;
      public Router getSender(){return From;}
      public Router getReceiver(){return To;}
      public void setTTL(int TTL){this.TTL=TTL;}
      public int getTTL(){return TTL;}
      public void setHops(int Hops){this.Hops=Hops;}
      public int getHops(){return Hops;}
      public void decTTL(){TTL--;Hops++;}
      public void drop() {
            if (!(this instanceof RoutingPacket)){
                  synchronized(GraphEditor.graph){
                        activecount--;
                        droppededcount++;
                        System.out.println("Dropped!"+this);
                  }
            }
      };
      public void finaliz() {
            //TODO Remove
            if(PacketCell!=null){
                  PacketCell.removeFromParent();
                  //GraphEditor.graph.refresh();
            }
            if (!(this instanceof RoutingPacket)){
                  synchronized(GraphEditor.graph){
                        activecount--;
                        finalizedcount++;
                  }
            }
      };

      public Packet(){
            if (!(this instanceof RoutingPacket)){
                  synchronized(GraphEditor.graph){
                        count++;
                        activecount++;
                  }
            }

      }
      public Packet(Router Router1,Router Router2){
            this.From=Router1;
            this.To=Router2;
            if (!(this instanceof RoutingPacket)){
                  synchronized(GraphEditor.graph){
```

```
                        count++;
                        activecount++;
                  }
            }
      }
      public Packet clone(){
            Packet p=null;
            try {
                  p = this.getClass().newInstance();
            } catch (InstantiationException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (IllegalAccessException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
            Class cl=this.getClass();
            while(cl!=Object.class){
            for(Field F:cl.getDeclaredFields()){
                  try {
                        if((F.getModifiers()&Modifier.FINAL)!=0)continue;
                        if((F.getModifiers()&Modifier.FINAL)!=0)continue;
                        //System.out.println(F);
                        F.set(p, F.get(this));
                              } catch (IllegalArgumentException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                  } catch (IllegalAccessException e) {
                        // TODO Auto-generated catch block
                        //e.printStackTrace();
                  }
            }
            cl=cl.getSuperclass();
            }
            return p;
      }
      public String toString(){return
            this.getClass().getSimpleName()+"("+From+"-->"+To+")TTL="+TTL;}
}

public class RoutingPacket extends Packet implements Serializable{
      public static int count=0;
      public RoutingPacket() {
            super();
      }
      public RoutingPacket(Router Router1, Router Router2) {
            super(Router1, Router2);
            synchronized(GraphEditor.graph){
                  count++;
            }
      }
}

public class AODVPacket extends RoutingPacket implements Serializable {
      int ID=0;
      public int router1num;
      public int router2num;
      public int getID(){return ID;}
      public void setID(int id){ID=id;}
      public void incID(){ID++;}
      public AODVPacket() {super();}
      public AODVPacket(Router Router1, Router Router2) {
            super(Router1, Router2);
            router1num= Router1.router_num;
            Route route = Router1.routing_protocol.RoutingTable.get(Router2);
```

```
            if (route == null || !(route instanceof AODVRoute)){
                  router2num = 0;
            }else{
                  if(this instanceof AODVRREQ){
                        router2num = ((AODVRoute)route).routernum+1;
                  }else{
                        router2num = ((AODVRoute)route).routernum;
                  }
            }
      }
}
public class AODVRREQ extends AODVPacket  implements BroadCastPacket,
Serializable {
      boolean D = false;
      public boolean getD(){return D;}
      public void setD(boolean d){D=d;}
      public AODVRREQ(){super();}
      public AODVRREQ(Router Router1, Router Router2, int ID) {
            super(Router1, Router2);
            this.setID(ID);
      }
}

public class AODVRREP extends AODVPacket implements Serializable {
      public AODVRREP(){super();}
      public AODVRREP(Router Router1, Router Router2) {
            super(Router1, Router2);
      }

      public AODVRREP(Router Router1, Router Router2, int metric) {
            super(Router1, Router2);
            this.Hops=metric;
      }
}

public class AODVERR extends AODVPacket {
      Router Destination;
      public int lasttl=0;
      Transit transit;
      public Router getDestination(){return Destination;}
      public AODVERR(){super();}
      public AODVERR(Router Router1, Router Router2,  Router Destination) {
            super(Router1, Router2);
            this.Destination=Destination;
      }
}

public interface BroadCastPacket {
      Router getSender();
      int getID();
}

public class MAODVRREQ extends AODVRREQ {
      public int angle;
      public Transit transit;
      public MAODVRREQ(){
            super();
      };
      public MAODVRREQ(Router Router1, Router Router2, int metric,
                  Transit transit){
            super(Router1,Router2,metric);
            this.transit=transit;
      }
}
```

```
public class MAODVRREP extends AODVRREP {
	public Transit transit;
	public MAODVRREP(){
		super();
	};
	public MAODVRREP(Router Router1, Router Router2, int metric, Transit
transit){	super(Router1,Router2,metric);
		this.transit=transit;
	}
}
package app.simulator.routing;

public class Router extends Peer implements Serializable {
	public int a;
	public int router_num=1;
	public static int last_num=0;
	public Router(){
		last_num++;
		a=last_num;
		routing_protocol=(RoutigProtocol)
				Restrictions.Algorythm.newInstance();
		routing_protocol.setRouter(this);
	}
	Map<Router,Link> LinkTable = new HashMap<Router,Link>();
	public RoutigProtocol routing_protocol= new AODV(this);
	synchronized public void registerLink(Link link,Router target){
		LinkTable.put(target,link);
		((AODV)routing_protocol).addRoute(target, link,
							0,Integer.MAX_VALUE);
		router_num++;
	}
	synchronized public void unregisterLink(Router target){
		routing_protocol.removeRoute(LinkTable.get(target));
		LinkTable.remove(target);
	}
	synchronized public void receive(Packet packet,Link link){
		if(packet instanceof AODVPacket){
			((AODV)routing_protocol).addRoute(packet.getSender(), link,
				packet.getHops(),((AODVPacket)packet).router1num);

		}
		if (packet.getReceiver()==this){
			packet.finaliz();
			if(packet instanceof RoutingPacket){
				routing_protocol.receive((RoutingPacket)packet,
				link);
			}
			return;
		}else{
			if(packet instanceof RoutingPacket
			&&!routing_protocol.proceed((RoutingPacket)packet,link))
			return;
		}
		send(packet,link);
	}
	synchronized public void send(Packet packet,Link link){
		packet.decTTL();
		if (packet.getTTL()<=0){
			packet.drop();
			return;
		}

		if(packet instanceof BroadCastPacket){
			routing_protocol.broadcast((BroadCastPacket)packet,link);
		}else{
```

```
                routing_protocol.route(packet);
            }
        }
        public String toString(){return ""+a;}

}

abstract public class RoutigProtocol implements Serializable{
        public Map<Router,Route> RoutingTable = new HashMap<Router,Route>();
        public Map<Router,List<Packet>> WaitingRoute =
                                        new HashMap<Router,List<Packet>>();
        public Router Router;
        Map<Router,Map<Router,Integer>> CheckBuffer=
                new HashMap<Router,Map<Router,Integer>>();
        public RoutigProtocol() {}
        RoutigProtocol(Router parent){Router = parent;}
        synchronized public void addRoute(Router router,Link link,int metric){
            if(!RoutingTable.containsKey(router)){
                RoutingTable.put(router, new Route(link,metric));
            }else if(RoutingTable.get(router).getMetric()>=metric){
                RoutingTable.remove(router);
                RoutingTable.put(router, new Route(link,metric));
            }
        }
        synchronized public void removeRoute(Link link){
            Router router=null;
            do{
                router=null;
                for(Router iRouter:RoutingTable.keySet()){
                    RoutingTable.get(iRouter).getLink();
                    if(RoutingTable.get(iRouter).getLink()==link){
                        router=iRouter;
                        break;
                    }
                }
                if(router!=null){
                    RoutingTable.remove(router);
                }
            }while(router!=null);
        }
        synchronized public void removeRoute(Router router)
            {RoutingTable.remove(router);}
        public boolean checkOld(AODVPacket packet){return true;}
        public boolean checkHorizont(BroadCastPacket packet){
            AODVPacket aodvpacket = (AODVPacket)packet;
            if(!checkOld(aodvpacket))return false;
            if(!CheckBuffer.containsKey(aodvpacket.getSender())){
                Map<Router,Integer> receivers =
                    new HashMap<Router,Integer>();
                CheckBuffer.put(aodvpacket.getSender(), receivers);
                receivers.put(aodvpacket.getReceiver(),
                                                    aodvpacket.getID());
                return true;
            }else if (!CheckBuffer.get(aodvpacket.getSender()).
                                containsKey(aodvpacket.getReceiver())) {
                CheckBuffer.get(aodvpacket.getSender()).
                                put(aodvpacket.getReceiver(),
                                        aodvpacket.getID());
                return true;
            }else if(CheckBuffer.get(aodvpacket.getSender()).
                get(aodvpacket.getReceiver())<aodvpacket.getID()){
                CheckBuffer.get(aodvpacket.getSender()).
                    remove(aodvpacket.getReceiver());
                CheckBuffer.get(aodvpacket.getSender()).
                    put(aodvpacket.getReceiver(), aodvpacket.getID());
```

```
                return true;
            }else if(aodvpacket.getID()==0){
                return false;
            }else{
                return false;
            }
        }
        abstract public void receive(RoutingPacket packet, Link link);
        abstract public boolean proceed(RoutingPacket packet, Link link);
        abstract public void findRoute(Router router1, Router router2);
        public void setRouter(Router r){this.Router=r;}
        synchronized public void broadcast(BroadCastPacket packet, Link link){
            if (!checkHorizont(packet)){
                return;
            }
            for(Link iLink:Router.LinkTable.values()){
                if(iLink!=link){
                    iLink.transmit(((Packet)packet).clone(),Router);
                }
            }

        }
        synchronized public void route(Packet packet){
            if (!RoutingTable.containsKey(packet.getReceiver())){
                if(!WaitingRoute.containsKey(packet.getReceiver())){
                    findRoute(Router,packet.getReceiver());
                    WaitingRoute.put(packet.getReceiver(),
                            new LinkedList<Packet>());
                }
                WaitingRoute.get(packet.getReceiver()).add(packet);
                return;
            }
            RoutingTable.get(packet.getReceiver()).getLink().
                transmit(packet,Router);
        }
}

public class AODV extends RoutigProtocol implements Serializable {
        public static int NeedRoutesCount=0;
        public static int FoundRoutesCount=0;
        AODV(Router parent) {super(parent);}
        AODV() {super();}
        Map<Router,Integer> AODVRREQcounter = new HashMap<Router,Integer>();
        public int getNewNumber(Router router){
            if(AODVRREQcounter.containsKey(router)){
                int n=AODVRREQcounter.remove(router)+1;
                AODVRREQcounter.put(router, n);
                return n;

            }else{
                AODVRREQcounter.put(router, 1);
                return 1;
            }
        }
        synchronized public void findRoute(final Router router1,
                                           final Router router2){
            synchronized(GraphEditor.graph){
                NeedRoutesCount++;
            }
            broadcast(new AODVRREQ(router1,
                        router2,getNewNumber(router2)),null);
            new Thread(){
                public void run(){
                    for(int i=0;
                          i<Restrictions.MaxRouteFindAttempts;i++){
```

```
                                    try {
                                          Thread.sleep
                                          (Restrictions.RouteFindPeriod*
                                                Restrictions.Scale);
                                          if(router1.
                                          routing_protocol.
                                          RoutingTable.
                                          containsKey(router2))return;
                                          if(!Simulator.isPaused)
                                          broadcast(new AODVRREQ(router1,
                                          router2,getNewNumber(router2)),
                                          null);
                                    } catch (InterruptedException e) {
                                          // TODO Auto-generated catch block
                                          e.printStackTrace();
                                    }
                              }
                        }
                  }.start();
      }
      Map<Router,Integer> CheckOld= new HashMap<Router,Integer>();
      public boolean checkOld(AODVPacket packet){return true;}
      synchronized public void receive(RoutingPacket packet, Link link){
            if(packet instanceof AODVRREQ){
                  Router.send(new AODVRREP(packet.getReceiver(),
                        packet.getSender()),null);
            }
            if(packet instanceof AODVERR){
                  RoutingTable.remove(((AODVERR)packet).getDestination());
                  findRoute(Router,((AODVERR)packet).getDestination());
            }
      }
      synchronized public boolean proceed(RoutingPacket packet, Link link){
            if((packet instanceof AODVRREQ)&&
                  ((AODVRREQ)packet).getD()&&
                  RoutingTable.containsKey(packet.getReceiver())){
                  if(!checkHorizont((AODVRREQ)packet))return false;
                  AODVRREP p = new AODVRREP(packet.getReceiver(),
                        packet.getSender(),
                        RoutingTable.get(packet.getReceiver()).
                              getMetric());
                  Router.send(p,null);
                  packet.finaliz();
                  return false;
            }
            return true;
      }
      synchronized public void addRoute(Router router,
                        Link link,int metric, int router_num){
            if(!RoutingTable.containsKey(router)){
                  RoutingTable.put(router,
                        new AODVRoute(link,metric,router_num));
            }else
if((((AODVRoute)RoutingTable.get(router)).routernum<router_num)||(RoutingTabl
e.get(router).getMetric()>=metric)){
                  RoutingTable.remove(router);
                  RoutingTable.put(router,
                        new AODVRoute(link,metric,router_num));

                  while(WaitingRoute.containsKey(router)){
                        synchronized(GraphEditor.graph){
                              FoundRoutesCount++;
                        }
                        for(Packet iPacket:WaitingRoute.get(router)){
                              route(iPacket);
```

```
                        }
                        WaitingRoute.remove(router);
                  }
            }
      }
      synchronized public void route(Packet packet){
            if (!RoutingTable.containsKey(packet.getReceiver())
            &&(packet.getSender()!=Router)){
                  Router.send(new AODVERR(Router, packet.getSender(),
                        packet.getReceiver()),null);
            }else{
                  super.route(packet);
            }

      }
}

public class MAODV extends AODV {
      public Map<Router,Map<Router,Transit>> TransitTable =
            new HashMap<Router,Map<Router,Transit>>();
      MAODV(Router parent) {super(parent);}
      MAODV() {super();}
       void addTransit(Router From, Router To, int Hops){
             if(!TransitTable.containsKey(From)){
                  TransitTable.put(From, new HashMap<Router,Transit>());}
             if(!TransitTable.containsKey(To)){
                  TransitTable.put(To, new HashMap<Router,Transit>());}
            Transit t = new Transit(From,To,Hops);
            TransitTable.get(From).put(To,t);
            TransitTable.get(To).put(From,t);
       }
       void addTransit(Transit t, int Hops){
             if(!TransitTable.containsKey(t.From)){
                  TransitTable.put(t.From, new HashMap<Router,Transit>());}
             if(!TransitTable.containsKey(t.To)){
                  TransitTable.put(t.To, new HashMap<Router,Transit>());}
            t.Hops+=Hops;
            TransitTable.get(t.From).put(t.To,t);
            TransitTable.get(t.To).put(t.From,t);
       }
      synchronized public void receive(RoutingPacket packet, Link link){
            if(packet instanceof MAODVRREQ){
                  Router.send(new MAODVRREP(packet.getReceiver(),
                        packet.getSender(),0,
                        ((MAODVRREQ)packet).transit),null);
                  return;
            }
            if(packet instanceof MAODVERR){
                  if(RoutingTable.containsKey(((AODVERR)packet).
                              getDestination()))
                        return;
            }
            if(packet instanceof AODVRREP){
                  if(packet instanceof MAODVRREP){
                        Transit t=null;
                        if(TransitTable.containsKey(packet.getSender())){
                              if(TransitTable.
                                    containsKey(packet.getReceiver())){
                                    t=TransitTable.
                                          get(packet.getSender()).
                                                get(packet.getReceiver());
                              }
                        }
                        Transit t2= ((MAODVRREP)packet).transit;
                        System.out.println(t);
```

```
                        if(t==null){
                              addTransit(
                                    ((MAODVRREP)packet).transit,
                                          packet.getHops());
                        }else{
                              if((t2.Hops-t.Hops<=packet.getHops())&&
                                    (t2.Hops-t.Hops>0)){
                                    t2.lastttl=2;
                                    addTransit(((MAODVRREP)packet).transit,
                                          packet.getHops()+t2.Hops);
                              }
                        return;
                  }else{
                        addTransit(packet.getSender(),
                              packet.getReceiver(),
                              packet.getHops());
                  }
            }
            if(packet instanceof AODVERR){
                  RoutingTable.remove(((AODVERR)packet).getDestination());
                  findRoute(Router,((AODVERR)packet).getDestination());
            }
            super.receive(packet, link);
      }

      synchronized public boolean proceed(RoutingPacket packet, Link link){
            if((packet instanceof MAODVRREQ)&&
            RoutingTable.containsKey(packet.getReceiver())){
                  Transit pt=((MAODVRREQ)packet).transit;
                  Transit t=null;
                  if(TransitTable.containsKey(pt.From)){
                        if(TransitTable.containsKey(pt.To)){
                              t=TransitTable.get(pt.From).get(pt.To);
                        }
                  }
                  if(t!=null&&t.Hops<pt.Hops&&pt.Num<=t.Num){

                        if(!checkHorizont((AODVRREQ)packet))return false;
                        MAODVRREP p = new MAODVRREP(packet.getReceiver(),
                              packet.getSender(),
                              RoutingTable.get(packet.getReceiver()).
                                    getMetric(),
                              ((MAODVRREQ)packet).transit);
                              Router.send(p,null);
                              packet.finaliz();
                              return false;
                  }
                  return true;
            }
            if(packet instanceof AODVRREP){
                  if(packet instanceof MAODVRREP){
                        addTransit(((MAODVRREP)packet).transit,
                              packet.getHops());
                  while(WaitingRoute.containsKey(((MAODVRREP)packet).
                  transit.From)){
                  synchronized(GraphEditor.graph){
                              FoundRoutesCount++;
                        }
                        for(Packet iPacket:
                        WaitingRoute.get(packet.getSender())){
                              route(iPacket);
                        }
                        WaitingRoute.remove(packet.getSender());
                  }
                  }else{
```

```
                        addTransit(packet.getSender(),
                        packet.getReceiver(),packet.getHops());
                  }

            }
            return super.proceed(packet, link);
      }
      synchronized public void route(Packet packet){

            if (!RoutingTable.containsKey(packet.getReceiver())
                  &&(packet.getSender()!=Router)){
                  //
            if(TransitTable.containsKey(packet.getSender())){


                  Transit t=null;
                  if(TransitTable.containsKey(packet.getSender())){
                        if(TransitTable.containsKey(packet.getReceiver())){
                              t=TransitTable.get(packet.getSender()).
                                    get(packet.getReceiver());
                        }
                  }
                  if(t==null){
                        super.route(packet);
                        return;
                  }
                  final Transit tt=t;
                  final MAODVRREQ p= new MAODVRREQ(this.Router,
                        packet.getReceiver(),
                        getNewNumber(packet.getReceiver()),t);
                  p.setD(true);
                  p.setTTL(2);
                  if(!WaitingRoute.containsKey(packet.getReceiver())){
                              Router.send(p,null);
                              WaitingRoute.put(packet.getReceiver(),
                              new LinkedList<Packet>());
                        }else{
                              tt.lastttl*=2;
                              p.setTTL(tt.lastttl);
                              if(p.getTTL()>8){
                                    Router.send(
                                          new AODVERR(Router,tt.To,tt.From),
                                          null);
                              }
                              else{
                                    Router.send(p,null);
                              }
                        }
                        WaitingRoute.get(packet.getReceiver()).add(packet);
                        return;
                  }
            }else{
                  super.route(packet);
            }
      }
}
```

```
public class Route implements Serializable{
	Link link;
	int Metric=0;
	boolean Active=true;
	public void setMetric(int Metric){this.Metric=Metric;}
	public int getMetric(){return Metric;}
	Route(Link link,int metric){this.link=link;this.Metric=metric;}
	Link getLink(){return link;}
	public String toString(){return ""+link+"("+Metric+")";}
}

public class AODVRoute extends Route{

	public int routernum;
	AODVRoute(Link link, int metric, int routernum) {
		super(link, metric);
		this.routernum=routernum;
	}
}

public class MAODVRoute extends AODVRoute {
	int FractureMetric;
	int FractureCount;
	int SumInfelicity;
	Router NearestFracture;
	MAODVRoute(Link link, int metric,int routernum,int FractureMetric,int
FractureCount,int SumInfelicity,Router NearestFracture ) {
		super(link, metric,routernum);
		this.FractureMetric=FractureMetric;
		this.FractureCount=FractureCount;
		this.SumInfelicity=SumInfelicity;
		this.NearestFracture=NearestFracture;
	}
}

public class Transit {
	public Router From;
	public Router To;
	public int FromNum;
	public int ToNum;
	public int Num=1;
	public boolean Active=true;
	public int ToTreeNum=0;
	public int Timer;
	public int Hops;
	public int Angle=0;
	public Transit(){}
	public Transit(Router From, Router To,int Hops){
		this.Hops=Hops;
		this.From=From;
		this.To=To;
	}
	public String toString(){
		return ""+From+"-->"+To+"("+Hops+")";
	}
	public int lastttl=3;
}
```

```
public  class Restrictions {
      public static double defaultradius=150;
      public static boolean isWiFi=false;//TODO realize
      public static int maxLinks=8;//TODO realize
      public static int maxPathLegth=30;
      public static int packetSize=25*1024;
      public static int maxSpeed=250*1024;
      public static int minPacketTransmitTime=1000*packetSize/maxSpeed;
      public static int maxRealRadius=70;//metr
      public static int ConnectionTime=30;//ms
      public static int Scale=100;//ms
      public static int MaxRouteFindAttempts=100;
      public static int RouteFindPeriod=10000;
      public static boolean autoremovelink=false;//TODO realize
      public static Class Algorythm = AODV.class;
      public static int SoftMoving=10;
      public static boolean Conus=false;
}
```

package app.gui;

```
public class GraphEditor extends BasicGraphEditor
{
      public static final NumberFormat numberFormat =
                  NumberFormat.getInstance();
      public static URL url = null;
      public GraphEditor(){
            this("mxGraph Editor",
                  new CustomGraphComponent(new CustomGraph()));
      }
      public static  mxGraph graph;
      public static mxCell[] getSortedEphire(final Object cellObj1){
            int n=Restrictions.maxLinks;
            java.util.ArrayList<mxCell> ephire =
                  new java.util.ArrayList<mxCell>();
            final mxCell cell1=(mxCell)cellObj1;
            if(cell1.getValue()==null){
                  cell1.setValue(new Router());
            }
            for(Object cellObj2:(Object[])graph.
                  getChildVertices(graph.getDefaultParent())){
                  mxCell cell2=(mxCell)cellObj2;
                  if(cell1.getGeometry().getPoint().
                              distance(cell2.getGeometry().getPoint())<
                        Restrictions.defaultradius){
                        if(cell1!=cell2)ephire.add(cell2);
                  }
            }
            mxCell[] SortedEphire = new mxCell[ephire.size()];
            ephire.toArray(SortedEphire);
            Arrays.sort(SortedEphire, new Comparator<mxCell>(){
                  public int compare(mxCell arg0, mxCell arg1) {
                        if(cell1.getGeometry().getPoint().
                              distance(arg0.getGeometry().getPoint())
                        >cell1.getGeometry().getPoint().
                              distance(arg1.getGeometry().getPoint())){
                              return 1;
                        }else{
                              return -1;
                        }
                  }

            });
            return SortedEphire;
```

```
}
synchronized public static void linksUpd(final Object cellObj1){
      int n=Restrictions.maxLinks;
      final mxCell cell1=(mxCell)cellObj1;
      if(cell1.getValue()==null){
            cell1.setValue(new Router());
      }
      for(Object edgeObj:graph.getEdges(cell1)){
            mxCell edge=(mxCell)edgeObj;
            edge.removeFromParent();
            edge.removeFromTerminal(true);
            edge.removeFromTerminal(false);
            mxCell cell = edge;
            if (cell.getValue() instanceof Link){
                  ((Link)cell.getValue()).removeLink();
            }else if(cell.getValue() instanceof Router){
                  ((Router)cell.getValue()).removeRouter();
            }
      }
      for(Object cellObj2:
            (Object[])graph.getChildVertices(graph.getDefaultParent()))
      {
            mxCell cell2=(mxCell)cellObj2;
            if (cell1==cell2)continue;
            int i=0;
            for(mxCell c:getSortedEphire(cell2)){
                  if(i<n){
                        if(graph.getEdgesBetween(c, cell2).length==0){
                              if(c.getEdgeCount()<n){
                                    graph.insertEdge(
                                          graph.getDefaultParent(),
                                          "Edge", null, c, cell2);
                                    i++;
                              }
                        }else{
                              i++;
                        }
                  }else{
                        if(graph.getEdgesBetween(c, cell2).length!=0){
                              removeEdgesBetween(c,cell2);
                        }
                  }
            }
      }
}
synchronized public static void removeEdgesBetween(
                  final mxCell cell1,final mxCell cell2){
      while(graph.getEdgesBetween(cell1, cell2).length!=0){
            mxCell edge = (mxCell) graph.getEdgesBetween(cell1,
                              cell2)[0];
            edge.removeFromParent();
            edge.removeFromTerminal(true);
            edge.removeFromTerminal(false);
            mxCell cell = edge;
            if (cell.getValue() instanceof Link){
                  ((Link)cell.getValue()).removeLink();
            }else if(cell.getValue() instanceof Router){
                  ((Router)cell.getValue()).removeRouter();
            }
      }

}
```

```
	synchronized public static void removeEdges(final Object cellObj1){
			mxCell cell1=(mxCell)cellObj1;
			if(cell1.getValue()==null){
				cell1.setValue(new Router());
			}

		for(Object
cellObj2:(Object[])graph.getChildVertices(graph.getDefaultParent())){

			mxCell cell2=(mxCell)cellObj2;
			if(cell1.getGeometry().getPoint().
				distance(cell2.getGeometry().getPoint())
				>Restrictions.defaultradius){
				if (((Object[])graph.
					getEdgesBetween(cell1, cell2)).length==0)
						continue;
				Object[] edges=graph.getEdgesBetween(cell1, cell2);
				mxCell edge = (mxCell)edges[0];
				mxCell lastedge =(mxCell)edges[edges.length-1];
			if(graph.getCellStyle(edge).get("shape").
				toString().equals("arrow"))
						continue;
				edge.removeFromParent();
				edge.removeFromTerminal(true);
				edge.removeFromTerminal(false);
				//linksUpd(cell2);
				mxCell cell = edge;
				if (cell.getValue() instanceof Link){
					((Link)cell.getValue()).removeLink();
				}else if(cell.getValue() instanceof Router){
					((Router)cell.getValue()).removeRouter();
				}


			}
		}
	}
	public GraphEditor(String appTitle, final mxGraphComponent component)
	{
		super("Wireless Modeling", component);
		graph = graphComponent.getGraph();
		graph.setCellsEditable(false);
		graph.setEdgeLabelsMovable(false);
		graph.setCellsBendable(false);
		graph.setCellsDisconnectable(false);
		graph.setCellsDeletable(false);
		graph.setCellsResizable(false);
		component.setAntiAlias(false);
		graph.setCellsCloneable(false);
		component.setBackground(new Color(204,204,204));
		graph.addListener(mxEvent.MOVE_CELLS,new mxIEventListener(){

			@Override
			public void invoke(Object sender, mxEventObject evt) {
				Point point=(Point)evt.getProperty("location");
				Object[] CellsArrayObj=
					(Object[])evt.getProperty("cells");
				//System.out.println(CellsArrayObj[0]);
				for(Object cellObj1:CellsArrayObj ){
					if(((mxCell)cellObj1).isVertex())
						linksUpd(cellObj1);
				}
			}
		});
		graph.addListener(mxEvent.CELLS_ADDED,new mxIEventListener(){
```

```
      public void invoke(Object sender, mxEventObject evt) {
            mxCell source = (mxCell)evt.getProperty("source");
            if(source==null)return;
            mxCell target = (mxCell)evt.getProperty("target");
            if(target==null)return;
      source.setConnectable(false);
      target.setConnectable(false);
            Object[] edges=graph.
                  getEdgesBetween((mxCell)evt.
                  getProperty("source"),
                  (mxCell)evt.getProperty("target"));
            if(edges.length==0)return;
            mxCell edge = (mxCell)edges[0];
            if(edge==null)return;
            int n= edges.length;
            mxCell lastedge =(mxCell)edges[n-1];
            if(graph.getCellStyle(lastedge).
                  get("shape").toString().equals("arrow")){
                  Packet p=new Packet((Router)source.getValue(),
                              (Router)target.getValue());
                  p.PacketCell=lastedge;
                  ((Router)source.getValue()).send(p, null);
                  return;
            }
            if(source.getValue()==null||
            !(source.getValue()instanceof Router)){
                  source.setValue(new Router());
            }
            if(target.getValue()==null||
            !(target.getValue()instanceof Router)||
            target.getValue()==source.getValue()){
                  target.setValue(new Router());
            }
            if(edge.getValue()==null||
                  !(edge.getValue()instanceof Link)){
                  Link link = new Link((Router)source.getValue(),
                              (Router)target.getValue(),edge);
                  edge.setValue(link);
            }
            graph.refresh();
      }
});

graph.addListener(mxEvent.REMOVE_CELLS,new mxIEventListener(){
      @Override
      public void invoke(Object sender, mxEventObject evt) {
            Object[] CellsArrayObj=
                  (Object[])evt.getProperty("cells");
            for(Object cellObj1:CellsArrayObj ){
                  mxCell cell = (mxCell)cellObj1;
                  if (cell.getValue() instanceof Link){
                        ((Link)cell.getValue()).removeLink();
                  }else if(cell.getValue() instanceof Router){
                        ((Router)cell.getValue()).removeRouter();
                  }
            }

      }
});

graph.addListener(mxEvent.REMOVE_CELLS_FROM_PARENT,
      new mxIEventListener(){
      @Override
      public void invoke(Object sender, mxEventObject evt) {
```

```
                        Object[] CellsArrayObj=
        (Object[])evt.getProperty("cells");
                        for(Object cellObj1:CellsArrayObj ){
                              mxCell cell = (mxCell)cellObj1;
                              if (cell.getValue() instanceof Link){
                                    ((Link)cell.getValue()).removeLink();
                              }else if(cell.getValue() instanceof Router){
                                    ((Router)cell.getValue()).removeRouter();
                              }
                        }
                  }
            });

      public static class CustomGraphComponent extends mxGraphComponent
      {
            public CustomGraphComponent(mxGraph graph)
            {
                  super(graph);
                  setPageVisible(false);
                  setGridVisible(false);
                  setToolTips(true);
                  getConnectionHandler().setCreateTarget(true);
                  mxCodec codec = new mxCodec();
                  Document doc = mxUtils.
                        loadDocument(GraphEditor.class.getResource(
                  "/com/mxgraph/examples/swing/resources/basic-style.xml")
                              .toString());
                  codec.decode(doc.getDocumentElement(),
                        graph.getStylesheet());
                  getViewport().setOpaque(false);
                  setBackground(Color.WHITE);
            }
      public static class CustomGraph extends mxGraph
      {
            protected Object edgeTemplate;
            public CustomGraph()
            {
                  setAlternateEdgeStyle("edgeStyle=
                        mxEdgeStyle.ElbowConnector;elbow=vertical");
            }
            public void setEdgeTemplate(Object template)
            {
                  edgeTemplate = template;
            }

      public static void main(String[] args)
      {
            try
            {

      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            }
            catch (Exception e1)
            {
                  e1.printStackTrace();
            }
            mxConstants.SHADOW_COLOR = Color.LIGHT_GRAY;
            GraphEditor editor = new GraphEditor();
            editor.createFrame(new EditorMenuBar(editor)).setVisible(true);
      }
}

public class SettingsPanel extends JPanel {
      public void add2(Component field, String Name){
            JLabel label = new JLabel(Name);
```

```
            add(label);
            add(field);
      }
      JTextField Algorythm =new JTextField("MAODV");
      JTextField Topology =new JTextField("Regular Mesh");
      JTextField Standart =new JTextField("802.15.4");
      JTextField Consistence   = new JTextField("0.9");
      JTextField Fluctuations   = new JTextField("0.5");
      JTextField Nodes =new JTextField("50");
      JTextField SendersCount = new JTextField("1");
      JTextField PacketsFreq = new JTextField("1");
      JTextField MinSendingDistance = new JTextField("3");
      JTextField MaxSendingDistance = new JTextField("30");
      JTextField MovingNodes= new JTextField("0");
      JTextField MovingSpeedPercent = new JTextField("0.5");
      JTextField MaxNodeConnections= new JTextField("6");
      JTextField TimeScale= new JTextField("5");
      JTextField From= new JTextField("1");
      JTextField To= new JTextField("29");
      public SettingsPanel(){
            super();
            this.setLayout(new GridLayout(
                  this.getClass().getDeclaredFields().length+1,2));
            for(Field f:this.getClass().getDeclaredFields()){
                        this.add2((Component)(f.get(this)),f.getName());
            }
            final JButton StartPause = new JButton("Start");
            final JButton Stop= new JButton("Stop");
            TimeScale.addActionListener(new ActionListener(){
                  public void actionPerformed(ActionEvent arg0) {
                        Restrictions.Scale=
                              Integer.parseInt(TimeScale.getText());
                  }
            });
            add(StartPause);
            add(Stop);
            final ActionListener StartPauseListener = new ActionListener(){

                  @Override
                  public void actionPerformed(ActionEvent arg0) {
                        if(StartPause.getText().matches("Start")){
      StartPause.setText("Pause");
      Simulator.isPaused=false;
      Restrictions.Scale=Integer.parseInt(TimeScale.getText());
      Restrictions.maxLinks=Integer.parseInt(MaxNodeConnections.getText());
      if(Standart.getText().matches("802.11"))
            Restrictions.isWiFi=true;else Restrictions.isWiFi=false;
      if(Algorythm.getText().matches("MAODV"))
            Restrictions.Algorythm=MAODV.class;
      else Restrictions.Algorythm=AODV.class;
      final Random random = new Random();
      final int maxCoord=(int)(
                  Double.parseDouble(Consistence.getText())*
                  Math.sqrt(Integer.parseInt(Nodes.getText()))*
                  Restrictions.defaultradius);
//Generate Nodes
if(Topology.getText().matches("Random Mesh")){
      for(int i=0;i<Integer.parseInt(Nodes.getText());i++){
            int x = Math.abs(random.nextInt())%maxCoord;
            int y = Math.abs(random.nextInt())%maxCoord;
            GraphEditor.graph.insertVertex(
                        GraphEditor.graph.getDefaultParent(),
                        "random point", null, x, y, 25, 25, "ellipse");
      }
}
```

```
final int CoordStep=(int) (Restrictions.defaultradius*
                                 Double.parseDouble(Consistence.getText()));
if(Topology.getText().matches("Random Regular Mesh")){
      for(int x=(int) (Restrictions.defaultradius*
                        Double.parseDouble(Consistence.getText()));
            x<maxCoord;
            x+=Restrictions.defaultradius*
                  Double.parseDouble(Consistence.getText())){
            for(int y=(int)(Restrictions.defaultradius*
                              Double.parseDouble(Consistence.getText()));
                  y<maxCoord;
                  y+=Restrictions.defaultradius*
                     Double.parseDouble(Consistence.getText())){
                        int dx = (int) (random.nextGaussian()*
                              Restrictions.defaultradius*
                              Double.parseDouble(Consistence.getText())*
                              Double.parseDouble(Fluctuations.getText()));
                        int dy = (int) (random.nextGaussian()*
                              Restrictions.defaultradius*
                              Double.parseDouble(Consistence.getText())*
                              Double.parseDouble(Fluctuations.getText()));

      GraphEditor.graph.insertVertex(GraphEditor.graph.getDefaultParent(),
             "random point", null, x+dx, y+dy, 25, 25, "ellipse");

            }
      }
}

if(Topology.getText().matches("Regular Mesh")){
      for(int x=(int) (Restrictions.defaultradius*
                       Double.parseDouble(Consistence.getText()));
            x<maxCoord;
            x+=Restrictions.defaultradius*
               Double.parseDouble(Consistence.getText())){
            for(int y=(int) (Restrictions.defaultradius*
                 Double.parseDouble(Consistence.getText()));
            y<maxCoord;
            y+=Restrictions.defaultradius*
               Double.parseDouble(Consistence.getText())){
      GraphEditor.graph.insertVertex(GraphEditor.graph.getDefaultParent(),
"random point", null, x, y, 25, 25, "ellipse");

            }
      }
}
mxCell v = (mxCell) GraphEditor.graph.getChildVertices(
            GraphEditor.graph.getDefaultParent())[0];
      GraphEditor.linksUpd(v);
      GraphEditor.graph.refresh();

      Object[] VerticesObj = GraphEditor.
            graph.getChildVertices(GraphEditor.graph.getDefaultParent());
//Generate Packets
int iMax = (int) (Integer.parseInt(SendersCount.getText()));
if( iMax == 1){
      int numSender = Integer.parseInt(From.getText())-1;
      int numReceiver = Integer.parseInt(To.getText())-1;
      Router Sender = (Router) ((mxCell)VerticesObj[numSender]).getValue();
      Router Receiver = (Router) ((mxCell)VerticesObj[numReceiver]).
                                                            getValue();
      new PacketThread(Sender,Receiver,(int) (Restrictions.Scale*1000/
                              Double.parseDouble(PacketsFreq.getText())));
}else{
```

```
int i=0;
while(i++<iMax){
      int numSender = Math.abs(random.nextInt())%VerticesObj.length;
      int numReceiver = Math.abs(random.nextInt())%VerticesObj.length;
      if(numSender==numReceiver){
            i--;
            continue;
      }
Router Sender = (Router) ((mxCell)VerticesObj[numSender]).getValue();
Router Receiver = (Router) ((mxCell)VerticesObj[numReceiver]).getValue();
Sender.send(new Packet(Sender, Receiver), null);
}
}
//Move Nodes
for(int i=0;i<Integer.parseInt(MovingNodes.getText());i++){
      final int num = Math.abs(random.nextInt())%VerticesObj.length;
      final mxCell cell = (mxCell)VerticesObj[num];
      final int sleeptime=(int) (1000*Restrictions.Scale/
            Double.parseDouble(MovingSpeedPercent.getText()));
      new Thread(){
            public void run(){
while (cell!=null&&cell.getParent()==GraphEditor.graph.getDefaultParent()){
      if(Simulator.isPaused){
            Thread.sleep(100);
            continue;
      }
      int oldx = (int) cell.getGeometry().getCenterX();
      int oldy = (int) cell.getGeometry().getCenterY();
      int x =Math.abs(random.nextInt())% (maxCoord);
      int y = Math.abs(random.nextInt())% (maxCoord);
      int dx =(x-oldx)/Restrictions.SoftMoving;
      int dy = (y-oldy)/Restrictions.SoftMoving;
      for(int i = 0;i<Restrictions.SoftMoving;i++){
      synchronized(cell){
            cell.getGeometry().setX(cell.getGeometry().getCenterX()+dx);
            cell.getGeometry().setY(cell.getGeometry().getCenterY()+dy);
      }
      Thread.sleep(sleeptime/Restrictions.SoftMoving);

}
}
}
}
}.start();
                              }
                        }else if(StartPause.getText().matches("Pause")){
                              StartPause.setText("Resume");
                              Simulator.isPaused=true;
                        }else if(StartPause.getText().matches("Resume")){
                              StartPause.setText("Pause");
                              Simulator.isPaused=false;
                        }
                  }
            };
            StartPause.addActionListener(StartPauseListener);
            Stop.addActionListener(new ActionListener(){
                  @Override
                  public void actionPerformed(ActionEvent arg0) {
                        mxCell parent = (mxCell) GraphEditor.
                                                graph.getDefaultParent();
                        for(Object CellObj:
                        GraphEditor.graph.getChildCells(parent)){
                              ((mxCell)CellObj).removeFromParent();
                        }
                        Router.last_num=0;
```

```
                        Packet.count=0;
                        Packet.finalizedcount=0;
                        Packet.droppededcount=0;
                        Packet.activecount=0;
                        AODV.FoundRoutesCount=0;
                        AODV.NeedRoutesCount=0;
                        if(StartPause.getText().matches("Pause")){
                              StartPauseListener.actionPerformed(null);
                        }
                        StartPause.setText("Start");
                        GraphEditor.graph.refresh();
                  }
            });
      }
}

class PacketThread{
      public PacketThread(final Router Sender, final Router Receiver,
                  final int sleeptime){
            new Thread(){
                  public void run(){
                        while(true){
                              if(!Simulator.isPaused)
                                    Sender.send(
                                          new Packet(Sender, Receiver),
                                          null);
                                    Thread.sleep(sleeptime);
                        }
                  }
            }.start();
      }
}

public class Statistics {
      public static String getStat(){
            String S;
            S= "Statisics:\n" +
                        "Packets Sum count: "+Packet.count+"\n" +
                        "Packets Current count: "+Packet.activecount+"\n"+
                        "Packets Finalized: "+Packet.finalizedcount+"\n" +
                        "Packets Dropped: "+Packet.droppededcount+"\n" +
                        "Service packet count: "+RoutingPacket.count+"\n"+
                        "Transmits count: "+Link.packetcount+"\n" +
                        "Service transmits count: "+
                                    Link.routingpacketcount+"\n"+
                        "Need routes: "+AODV.NeedRoutesCount+"\n" +
                        "Found routes: "+AODV.FoundRoutesCount+"\n" +
                        "Average Packets/Distance: \n"+
                        "Average MovingSpeed/Distance: \n"+
                        "Average Packet Transmit Latency: \n";
            mxCell parent = (mxCell) GraphEditor.graph.getDefaultParent();
            for(Object V:GraphEditor.graph.getChildVertices(parent)){
                  Router R = (Router) ((mxCell)V).getValue();
                  if (R==null) return S;
                  if(!R.routing_protocol.WaitingRoute.isEmpty()){
                        S+=R.routing_protocol.WaitingRoute+"\n";
                  }
            }
            return S;
}
```