# TrustMod Manual

Muhammad Shadi Hajar
m.hajar@rgu.ac.uk

July 2021

## 1 Adding TrustMod to NS-3

The name of the module is trust. The easiest way to add the trust module to NS-3 simulator is to create a new module named Trust. Below are the steps to create a new module named trust:

1. Create the module skeleton:
   NS-3 is provided with a python program to create the skeleton of new modules. This program is available in the *utils* directory. Therefore, in order to create the Trust module, do the following:
   `$ ./utils/create-module.py trust`
   Now, a new directory named *trust* will be created in the following path: `../ns-allinone-3.30/ns-3.30/src` (It depends on your NS-3 version).
2. Copy and replace the *trust* directory into *src* directory.

## 2 Linking to Other Modules

As the trust module should receive cross-layer information from sending and receiving paths, the following implementation should be added:

### 2.1 Sending Path

You should add the following code to the `../ns-allinone-3.30/ns-3.30/src/wifi/model/wifi-net-device.cc` file.

- Import the trust module:

```
#include "ns3/trust.h"
```

- In the *send* method, add the following code:

```
Ptr<Node> n = GetNode();
Ptr<Trust> trust = n->GetObject<Trust> ();
if(trust)
{
    trust->SentInput(packet, realTo);
}
```

This code should be added directly after:

```
Mac48Address realTo = Mac48Address::ConvertFrom (dest);
```

Of course, you can easily copy the file wifi-net-device.cc to the path: `../ns-allinone-3.30/ns-3.30/src/wifi/model/`

## 2.2 Receiving Path

The AODV module has been chosen to get information from the receiving path.
In the `../ns-allinone-3.30/ns-3.30/src/aodv/model/aodv-routing-protocol.h` file, mainly two methods should be declared as follows:

```
bool PromiscReceive (Ptr<NetDevice> device, Ptr<const Packet>
    packet, uint16_t protocol, const Address &from, const Address
     &to, NetDevice::PacketType packetType);

bool IsNeighbor(Ipv4Address x)
{
    return m_nb.IsNeighbor(x);
}
```

Moreover, the following methods should also be added in order to launch malicious activities:

```
void SetMalicious(int att)
{
    maliciousAttribute = att;
}

int GetMalicious() const
{
   return maliciousAttribute;
}
```

```
void SetOnOffAttackPeriod(int att)
{
   onOffAttackAttribute = att;
}

int GetOnOffAttackPeriod () const
{
   return onOffAttackAttribute;
}

void SetIsForwarded(Ipv4Address att)
{
   forwardedAttribute = att;
}

Ipv4Address GetIsForwarded() const
{
   return forwardedAttribute;
}

void SetPromiscImp (bool att)
{
   promiscAttribute = att;
}

bool GetPromiscImp () const
{
   return promiscAttribute;
}

void SetOption(int att)
{
   optionAttribute = att;
}

int GetOption() const
{
   return optionAttribute;
}

void SetOnOffPercent(double att)
{
   onOffPercentAttribute = att;
}
```

```
double GetOnOffPercent () const
{
   return onOffPercentAttribute;
}

void SetInitPeriod(int att)
{
   initPeriodAttribute = att;
}

int GetInitPeriod() const
{
   return initPeriodAttribute;
}
```

In the `../ns-allinone-3.30/ns-3.30/src/aodv/model/aodv-routing-protocol.cc` file, you have to do the following:

- Register *SetPromiscReceiveCallback* on *netDevice*. Add the following code to the end of the *RoutingProtocol* :: *Start* method:

```
if (GetPromiscImp ())
{
m_ipv4->GetNetDevice (1)->SetPromiscReceiveCallback (
    MakeCallback (&RoutingProtocol::PromiscReceive, this));
}
```

- Next, you have to write the implementation of *PromiscReceive* method as follows. This code could be added to the end of the file.

```
bool RoutingProtocol::PromiscReceive (Ptr<NetDevice> device,
     Ptr<const Packet> packet, uint16_t protocol, const
    Address &from, const Address &to, NetDevice::PacketType
    packetType)
{
   Ipv4Header packetCopyHeader;
   Ptr<Packet> packetCopy = packet->Copy();
   packetCopy->RemoveHeader (packetCopyHeader);
   Ptr<WifiNetDevice> wifiNetDevice = DynamicCast<
    WifiNetDevice> (device);
   Ptr<Node> node = wifiNetDevice->GetNode();
   Ptr<Trust> trust = node->GetObject<ns3::Trust> ();
   if (trust)
   {
```

```
      trust->ForwardedInput(packet, from);
   }
   return true;
}
```

- Add the malicious attributes to the end of *RoutingProtocol* :: *GetTypeId* method as follows:

```
.AddAttribute("maliciousAttribute", "An␣attribute␣to␣set␣up␣
    malicious␣activity", UintegerValue(0),
    MakeUintegerAccessor(&RoutingProtocol::SetMalicious,&
    RoutingProtocol::GetMalicious), MakeUintegerChecker<
    uint16_t>())

.AddAttribute("onOffAttackAttribute", "An␣attribute␣to␣set␣
    the␣period␣of␣on␣off␣attack", UintegerValue(0),
    MakeUintegerAccessor(&RoutingProtocol::
    SetOnOffAttackPeriod, &RoutingProtocol::
    GetOnOffAttackPeriod), MakeUintegerChecker<uint16_t>())

.AddAttribute("forwardedAttribute", "An␣attribute␣to␣check␣
    if␣the␣sent␣packet␣is␣forwarded␣or␣not", Ipv4AddressValue
    ("1.1.1.1"), MakeIpv4AddressAccessor(&RoutingProtocol::
    SetIsForwarded,
&RoutingProtocol::GetIsForwarded), MakeIpv4AddressChecker())

.AddAttribute("PromiscAttribute", "An␣attribute␣to␣enable␣
    the␣implementation␣of␣method␣PromiscReceive",
    BooleanValue(false), MakeBooleanAccessor(&RoutingProtocol
    ::SetPromiscImp,
&RoutingProtocol::GetPromiscImp), MakeBooleanChecker ())

.AddAttribute("optionAttribute", "An␣attribute␣to␣set␣up␣
    malicious␣activity␣option", UintegerValue(0),
    MakeUintegerAccessor(&RoutingProtocol::SetOption, &
    RoutingProtocol::GetOption), MakeUintegerChecker<uint16_t
    >())
.AddAttribute("onOffPercentAttribute", "An␣attribute␣to␣set␣
    the␣percentage␣of␣the␣on␣phase␣to␣the␣off␣phase",
    DoubleValue(0), MakeDoubleAccessor(&RoutingProtocol::
    SetOnOffPercent, &RoutingProtocol::GetOnOffPercent),
    MakeDoubleChecker<double>())

.AddAttribute("initPeriodAttribute", "An␣attribute␣to␣set␣
    the␣initialization␣period", UintegerValue(0),
```

```
    MakeUintegerAccessor(&RoutingProtocol::SetInitPeriod, &
    RoutingProtocol::GetInitPeriod), MakeUintegerChecker<
    uint16_t>())
;
return tid;
}
```

- Now, the malicious activity functionalities has to be added to the end of *RoutingProtocol* :: *RouteInput* method as follows:

```
 if   (GetMalicious())
  {
      // First scenario when the node is developig good
    behaviour
      if (GetMalicious()== 1)
      {
         if(ns3::Simulator::Now().GetSeconds ()>1) // This
    first second of simulation is for initializaton
          {
             Ptr<UniformRandomVariable> x = CreateObject<
    UniformRandomVariable> ();
             x->SetAttribute ("Min", DoubleValue (0));
             x->SetAttribute("Max", DoubleValue(300));
             uint16_t temp = x->GetInteger ();

             if((temp%36)+ns3::Simulator::Now().GetSeconds ()
    -2<18)
             {
                NS_LOG_ERROR("D␣"<<ns3::Simulator::Now().
    GetSeconds()<<"␣Source:␣"<<header.GetSource ()<<"␣
    Destination:␣"<<header.GetDestination ());
                //std::cout <<"D "<< ns3::Simulator::Now().
    GetSeconds ()<<" "<<header.GetSource ()<<" " <<header.
    GetDestination ()<<" " <<"malicious activity"<<std::endl;
                return false;
             }
          }
      }
//************************************************************
      else if(GetMalicious()== 2) // Second scenario when
    the node is developig bad behaviour
      {
         if(ns3::Simulator::Now().GetSeconds ()>1) // This
    first second of simulation is for initializaton
```

```
        {
           Ptr<UniformRandomVariable> x = CreateObject<
    UniformRandomVariable> ();
           x->SetAttribute ("Min", DoubleValue (0));
           x->SetAttribute("Max", DoubleValue(300));
           uint16_t temp = x->GetInteger ();

           if((temp%36)+ns3::Simulator::Now().GetSeconds ()
    -2>=18)
           {
              NS_LOG_ERROR("D␣"<<ns3::Simulator::Now().
    GetSeconds()<<"␣Source:␣"<<header.GetSource ()<<"␣
    Destination:␣"<<header.GetDestination ());
              //std::cout <<"D "<< ns3::Simulator::Now().
    GetSeconds ()<<" "<<header.GetSource ()<<" " <<header.
    GetDestination ()<<" " <<"malicious activity"<<std::endl;
              return false;
           }
        }

      }
//*************************************************************
      else if(GetMalicious()== 3) // Third scenario when the
     node change its behaviour from good to bad suddenly
      {
         if(ns3::Simulator::Now().GetSeconds ()>1) // This
    first second of simulation is for initializaton
        {
           Ptr<UniformRandomVariable> x = CreateObject<
    UniformRandomVariable> ();
           x->SetAttribute ("Min", DoubleValue (0));
           x->SetAttribute("Max", DoubleValue(300));
           uint16_t temp = x->GetInteger ();
           //std::cout<<"The temp variable is: "<<temp<<std
    ::endl;
           if (ns3::Simulator::Now().GetSeconds ()<20) //
    Less than 20 seconds, the node is behaving good
           {
              if((temp%36)+ns3::Simulator::Now().GetSeconds
     ()-2<18)
              {
                 NS_LOG_ERROR("D␣"<<ns3::Simulator::Now().
    GetSeconds()<<"␣Source:␣"<<header.GetSource ()<<"␣
    Destination:␣"<<header.GetDestination ());
                 //std::cout <<"D "<< ns3::Simulator::Now()
    .GetSeconds ()<<" "<<header.GetSource ()<<" " <<header.
```

```
    GetDestination ()<<" " <<"malicious activity"<<std::endl;
                   return false;
               }
           }
           else // The node after the 20 seconds starts its
     bad behaviour
           {
               Ptr<UniformRandomVariable> y = CreateObject<
    UniformRandomVariable> ();
               y->SetAttribute ("Min", DoubleValue (0));
               y->SetAttribute ("Max", DoubleValue(300));
               uint16_t temp1 = y->GetInteger ();
               //std::cout<<"The temp1 variable is: "<<temp1
    <<std::endl;
               if (temp1%4<3)
               {
                   NS_LOG_ERROR("D␣"<<ns3::Simulator::Now().
    GetSeconds()<<"␣Source:␣"<<header.GetSource ()<<"␣
    Destination:␣"<<header.GetDestination ());
                   return false;
               }

           }
       }

     }
//***********************************************************
      else if (GetMalicious()== 4 && GetOnOffAttackPeriod())
     // The fourth scenario the fixed on off attack ( the
    period of the on and off phases are preset)
     {
        if(ns3::Simulator::Now().GetSeconds ()>1) // This
    first second of simulation is for initializaton
        {
           Ptr<UniformRandomVariable> x = CreateObject<
    UniformRandomVariable> ();
           x->SetAttribute ("Min", DoubleValue (0));
           x->SetAttribute("Max", DoubleValue(300));
           uint16_t temp = x->GetInteger ();
           //std::cout<<"The temp variable is: "<<temp<<std
    ::endl;
           if (ns3::Simulator::Now().GetSeconds ()<
    GetInitPeriod()) // Less than the initialization period,
    the node is behaving good
           {
               //if((temp%36)+ns3::Simulator::Now().
```

```
GetSeconds ()-2<GetInitPeriod()-2)
            if (temp%10<1)
            {
               NS_LOG_ERROR("D␣"<<ns3::Simulator::Now().
GetSeconds()<<"␣Source:␣"<<header.GetSource ()<<"␣
Destination:␣"<<header.GetDestination ());
               //std::cout <<"D "<< ns3::Simulator::Now()
.GetSeconds ()<<" "<<header.GetSource ()<<" " <<header.
GetDestination ()<<" " <<"malicious activity"<<std::endl;
               return false;
            }
         }
         else // The node after the initial period starts
 its on off attack
         {
            uint16_t onOffPeriod = GetOnOffAttackPeriod()
;
            uint16_t loc = (ns3::Simulator::Now().
GetSeconds ()-GetInitPeriod())/onOffPeriod; //a variable
to specify where we are located in on of off period
            Ptr<UniformRandomVariable> z = CreateObject<
UniformRandomVariable> ();
            z->SetAttribute ("Min", DoubleValue (0));
            z->SetAttribute ("Max", DoubleValue(300));
            uint16_t temp2 = z->GetInteger ();
            if (loc%2==0)
            {
               if (temp2%4<3)  // Change the malicious
drop rate default (temp2%4<3)
               {
                  NS_LOG_ERROR("D␣"<<ns3::Simulator::Now
().GetSeconds()<<"␣Source:␣"<<header.GetSource ()<<"␣
Destination:␣"<<header.GetDestination ());
                  return false;
               }
            }
            else
            {
               if (temp2%10<1)
               {
                  //NS_LOG_ERROR("D AODV Malicious: "<<
ns3::Simulator::Now().GetSeconds()<<" Source: "<<header.
GetSource ()<<" Destination: "<<header.GetDestination ())
;
                  NS_LOG_ERROR("D␣"<<ns3::Simulator::Now
().GetSeconds()<<"␣Source:␣"<<header.GetSource ()<<"␣
```

```
    Destination:␣"<<header.GetDestination ());
                        return false;
                    }
                }

            }
        }
      }
//*************************************************************

      else if (GetMalicious()== 5 && GetOnOffAttackPeriod())
     // The fifth scenario the percentage on off attack ( the
     percentage between on and off phases are preset)
      {
         if(ns3::Simulator::Now().GetSeconds ()>1) // This
    first second of simulation is for initialization
         {
            Ptr<UniformRandomVariable> x = CreateObject<
    UniformRandomVariable> ();
            x->SetAttribute ("Min", DoubleValue (0));
            x->SetAttribute("Max", DoubleValue(300));
            uint16_t temp = x->GetInteger ();
            //std::cout<<"The temp variable is: "<<temp<<std
    ::endl;
            if (ns3::Simulator::Now().GetSeconds ()<
    GetInitPeriod()) // Less than the initialization period,
    the node is behaving good
            {
               //if((temp%36)+ns3::Simulator::Now().
    GetSeconds ()-2<GetInitPeriod()-2)
               if (temp%10<1)
               {
                  NS_LOG_ERROR("D␣"<<ns3::Simulator::Now().
    GetSeconds()<<"␣Source:␣"<<header.GetSource ()<<"␣
    Destination:␣"<<header.GetDestination ());
                  //std::cout <<"D "<< ns3::Simulator::Now()
    .GetSeconds ()<<" "<<header.GetSource ()<<" " <<header.
    GetDestination ()<<" " <<"malicious activity"<<std::endl;
                  return false;
               }
            }
            else // The node after the initial period
    seconds starts its on off attack
            {
               uint16_t offPeriod = GetOnOffAttackPeriod();
               float onPeriod =  offPeriod*GetOnOffPercent()
```

```
;
            uint16_t totalPeriod = onPeriod + offPeriod;
            std::cout<<"The␣totalPeriod␣value␣is:␣"<<
totalPeriod<<std::endl;
            uint16_t loc = ((ns3::Simulator::Now().
GetSeconds ()-20)/totalPeriod)+1; //a variable to specify
 where we are located in on or off period
            std::cout<<"The␣loc␣value␣is:␣"<<loc<<std::
endl;
            Ptr<UniformRandomVariable> z = CreateObject<
UniformRandomVariable> ();
            z->SetAttribute ("Min", DoubleValue (0));
            z->SetAttribute ("Max", DoubleValue(300));
            uint16_t temp2 = z->GetInteger ();
            if (ns3::Simulator::Now().GetSeconds () <=
(20+loc*totalPeriod-offPeriod) )// if it is true then we
are in the on phase
            {
               if (temp2%4<3)  //temp2%4<3  //temp2%10<8
               {
                  NS_LOG_ERROR("D␣"<<ns3::Simulator::Now
().GetSeconds()<<"␣Source:␣"<<header.GetSource ()<<"␣
Destination:␣"<<header.GetDestination ());
                  return false;
               }
            }
            else
            {
               if (temp2%10<1)
               {
                  //NS_LOG_ERROR("D AODV Malicious: "<<
ns3::Simulator::Now().GetSeconds()<<" Source: "<<header.
GetSource ()<<" Destination: "<<header.GetDestination ())
;
                  NS_LOG_ERROR("D␣"<<ns3::Simulator::Now
().GetSeconds()<<"␣Source:␣"<<header.GetSource ()<<"␣
Destination:␣"<<header.GetDestination ());
                  return false;
               }
            }

         }
      }
   }

}
```

```
return Forwarding(p, header, ucb, ecb);
}
```

Of course, you can easily copy and replace the aodv-routing-protocol.h and aodv routing-protocol.cc files in `../ns-allinone-3.30/ns-3.30/src/aodv/model/`

# 3 TrustMod Usage

In order to use the trust module, you can instantiate a trust helper object in the simulation file, assign the required attributes, and then install it in the required nodes, as follows:

1. Import the trust module:

```
#include "ns3/trust-module.h"
```

2. Instantiate a trust helper object:

```
TrustHelper trustHelper;
```

3. Set the required attributes:

```
trustHelper.SetTrustAttribute ("MaxPerHopDelay", TimeValue (
    Seconds (0.0)));
trustHelper.SetTrustAttribute ("SimulationTimeAttribute",
    UintegerValue (202));
trustHelper.SetTrustAttribute ("TimeUnitAttribute",
    UintegerValue (1));
trustHelper.SetTrustAttribute ("
    RecommendationsListenerPortAttribute", UintegerValue
    (49999));
```

4. Create an application container and install the trust module in the nodes:

```
ApplicationContainer c = trustHelper.Install(allNodes.Get(0)
    );
```

In order to launch malicious activities, use the following steps:

1. Import the aodv module:

```
#include "ns3/aodv-module.h"
```

2. Instantiate an aodv helper object:

```
AodvHelper maliciousAodv;
```

3. Set the required attributes:

```
maliciousAodv.Set("maliciousAttribute", UintegerValue(4));
maliciousAodv.Set("onOffAttackAttribute", UintegerValue(30))
    ;
maliciousAodv.Set("onOffPercentAttribute", DoubleValue(1));
maliciousAodv.Set("initPeriodAttribute",UintegerValue(50));
```

4. Instantiate an internet stack helper object:

```
InternetStackHelper maliciousStack;
```

5. Set the routing to the malicious aodv:

```
maliciousStack.SetRoutingHelper(maliciousAodv);
```

6. Install the malicious stack into the nodes:

```
maliciousStack.Install(maliciousNodes);
```

# 4 Trust Output

The trust statistics and results are serialized to text file. The default path of the trust output is `../ns-allinone-3.30/ns-3.30`, and the default name is *TrustData.txt*. The name of the trust output could be changed by setting the following attribute:

```
trustHelper.SetTrustAttribute ("FileAttribute", StringValue ("
    TrustData.txt"));
```

*******