

**TÀI LIỆU KỸ THUẬT**

# **NTU-ATTACK**

## **NỀN TẢNG MÔ PHỎNG TẤN CÔNG MẠNG MANET**

**Mai Cường Thọ**

(thomc@ntu.edu.vn)

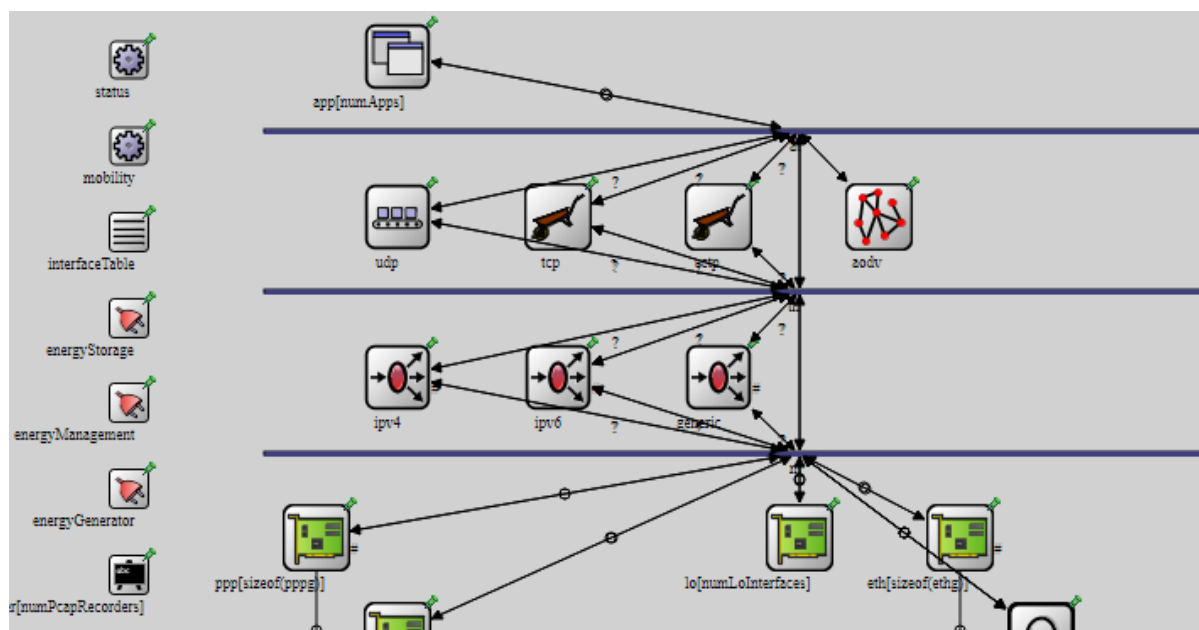
[https://github.com/maicuongtho/NTU\\_Attack\\_MANET](https://github.com/maicuongtho/NTU_Attack_MANET)

## 1. Phân tích thiết kế nền tảng NTU-Attack

Hệ thống NTU-Attack được chúng tôi thiết kế dựa trên việc thừa kế và phát triển từ ý tưởng, kiến trúc và cách tổ chức mã của NETA framework [16], [29], tuân thủ nguyên tắc thiết kế sau:

- Không hiệu chỉnh các mã các framework nền INET và OMNeT++
- Sửa đổi ít nhất có thể các module được hack để thực hiện hành vi độc hại.
- Thực hiện nguyên tắc của lập trình hướng đối tượng trong việc kế thừa các thành phần sẵn có của OMNeT++ và INET ghi đè phương thức để thực hiện xây dựng hệ thống.

NTU-Attack nhằm mục tiêu hoạt động tương thích trên nền tảng OMNeT++ 5.6 và INET 4.x, giúp mô phỏng một số tấn công mạng MANET, thử nghiệm các giao thức mới và các kỹ thuật phòng chống và phát hiện tấn công.



**Hình 1. Một nút mạng MANET bình thường sử dụng AODV**

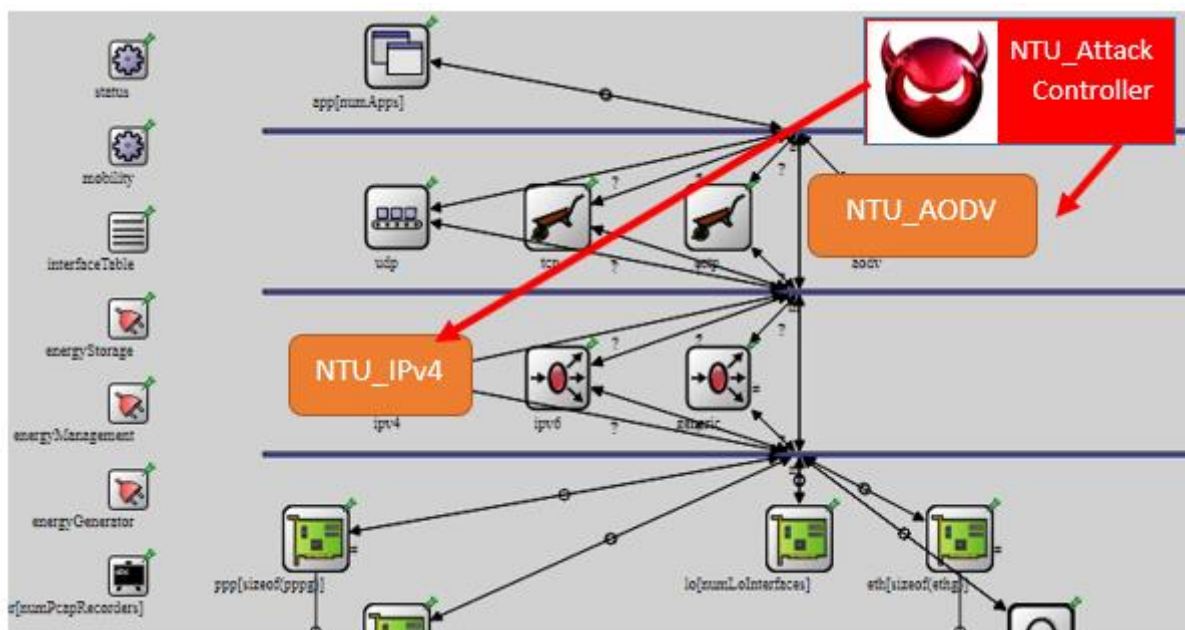
Để mô phỏng tấn công, ta cần xây dựng được nút mạng di động hoạt động được như nút bình thường nhưng có thể thực hiện thêm được hành vi độc hại.

Tấn công mạng nói chung, mạng MANET nói riêng có thể được thực hiện ở tầng khác nhau trên mô hình TCP/IP (tấn công tầng ứng dụng, tầng vận chuyển, tầng mạng, và liên kết dữ liệu. NTU-Attack nhằm mục tiêu trước hết là mô phỏng tấn công vào tầng

mạng, mà cụ thể là vào giao thức xây dựng lộ trình khi có yêu cầu trao đổi dữ liệu (giao thức định tuyến AODV).

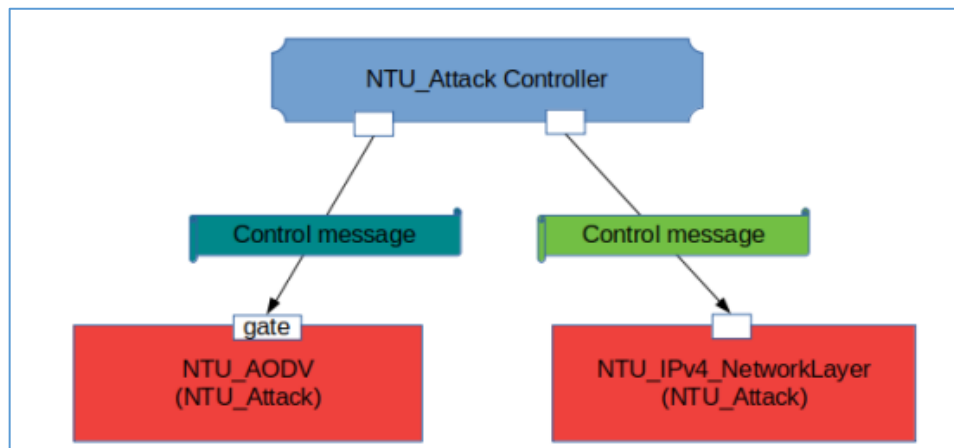
Do vậy, để xây dựng nút mạng mạo danh là nút đích hoặc có đường đi tốt nhất tới đích (nút độc hại) và thực hiện các hành động như hủy gói (dropping) hoàn toàn, hủy gói theo tần suất nào đó, hủy gói theo giao thức tầng ứng dụng khi nhận được gói, trì hoãn việc chuyển gói (delay) thì buộc phải hiệu chỉnh (hack) hai module con thành phần là *IPv4* và *AODV* của INET.

Tuân thủ nguyên tắc thiết kế, việc triển khai ta không được hiệu chỉnh trực tiếp trên các module gốc, mà nhân bản các module này và thêm các mã lệnh xử lý cần thiết thực hiện được hành vi tấn công (được đặt tên tương ứng là *NTU\_IPv4* và *NTU\_AODV*). Nút độc hại này ít nhất hoạt động như một nút bình thường trong mạng, và chỉ thực hiện tấn công vào thời điểm nhất định với những tham số nhất định, vì vậy mỗi hình thức tấn công cần có một bộ/module điều khiển việc tấn công này.



**Hình 2. Cấu trúc và thành phần node mạng thực hiện hành vi độc hại**

### Mô hình truyền thông điệp giữa các module

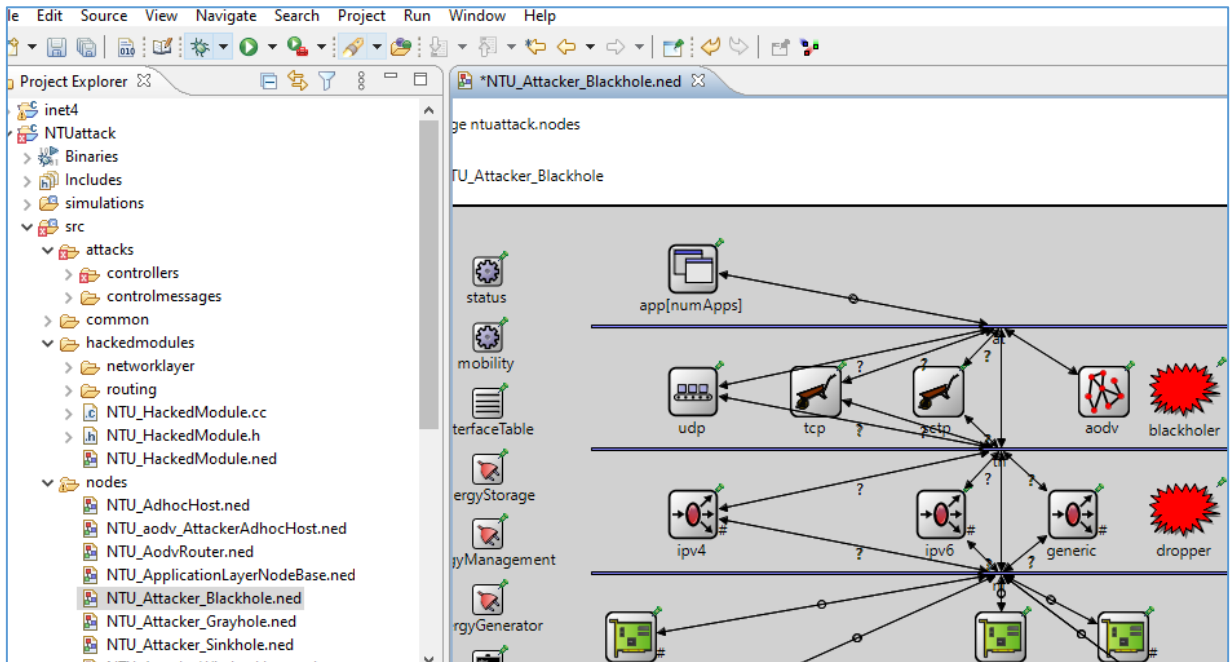


**Hình 3. Mô hình truyền thông điệp điều khiển tấn công**

Mỗi hình thức tấn công khi được thực hiện, NTU\_attack controller sẽ gửi thông điệp điều khiển (có cấu trúc riêng tùy theo từng hình thức tấn công) đến các module NTU\_AODV và NTU\_IPv4 để kích hoạt và truyền tham số.

Tổ chức dự án được thực hiện trên OMNeT++5.6 như sau:

- Môi trường cài đặt: Hệ điều hành Ubuntu 18
- Dự án nền phải có: INET framework 4
- Tên dự án của đề tài: **NTU\_Attack**
  - Dự án phải tham chiếu đến dự án nền tảng INET4
  - *Simulation*: Chứa các thư mục con cho mỗi mô phỏng
  - *Src*: thư mục mã nguồn
    - *Attacker*
      - Controller : chức các controller tấn công
      - ControlMessage: cấu trúc thông điệp tấn công
    - *HackedModule*
      - Networklayer: các giai thức tầng mạng “sửa đổi”
      - Routing: các giao thức định tuyến sửa đổi, xây dựng mới
    - *Nodes*: Chứa các node mạng độc hại được xây dựng
      - Blackhole Attacker node
      - Grayhole Attacker node
      - ..



**Hình 4. Tổ chức dự án thực tế NTU\_Attack trên OMNeT**

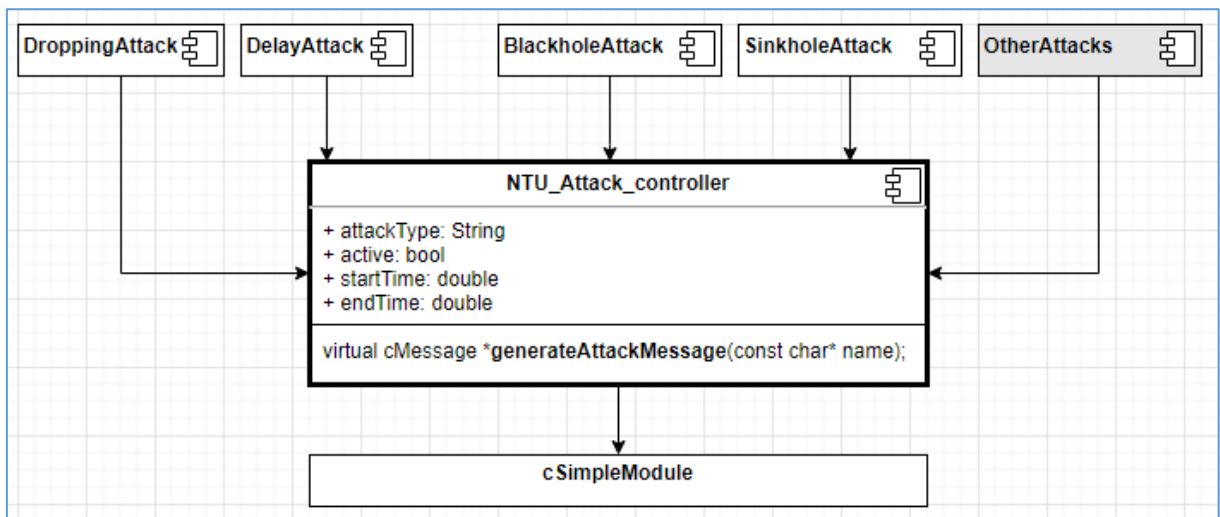
## 2. Module NTU\_Attack\_Controller

Như đã trình bày ở trên hình 3, NTU\_Attack\_Controller là module điều khiển việc thực hiện các hành vi độc hại, các tham số cho thực hiện hành vi gồm có: thời điểm kích hoạt, thời điểm ngừng kích hoạt, xác suất hủy gói, giao thức vận chuyển dữ liệu bị hủy gói.

Để thực hiện các kiểu tấn công cụ thể, từng module điều khiển cụ thể sẽ được xây dựng dựa trên việc kế thừa module NTU\_Attack\_Controller và xây dựng cấu trúc thông điệp điều khiển riêng, tạo thông điệp và gửi cho các module độc hại NTU\_AODV và NTU\_IPv4 (các controller cụ thể đã xây dựng bao gồm: sinkhole, blackhole, grayhole, dropping).

Các nút độc hại thực hiện các hành vi tương ứng sẽ được xây dựng bằng tạo bản sao của nút bình thường rồi thay thế các module con bình thường bằng các module độc hại, đồng thời gắn thêm các bộ điều khiển thực hiện hành vi độc hại.

Hình 5 trình bày mô hình thiết kết các bộ điều khiển thực hiện từng hành vi tấn công, dựa trên module cơ sở NTU\_Attack\_controller.



Hình 5. Thiết kế và sử dụng của module NTU\_Attack\_controller

Thiết kế cụ thể triển khai qua 2 bước cụ thể như sau:

Bước 1. Định nghĩa module controller bằng ngôn ngữ mô tả **.ned**

```

NTU_Attack.ned
package ntuattack.attacks.controllers;
simple NTU_Attack
{
    parameters:
        @display("i=misc/cloud3,red,100;is=1");
        string attackType @enum("dropping","sinkhole","blackhole",\
                                "grayhole","wormhole","delay") = default("");
        bool active = default(false);
        double startTime @unit("s") = default(0s);
        double endTime @unit("s") = default(0s);
}

```

Hình 6. Định nghĩa controller cơ sở NTU\_Attack\_Controller

- Thuộc tính *active* mặc định là *false* được sử dụng để qui định khả năng cho phép tấn công hoặc không.
- Thuộc tính *startTime* và *endTime* giúp xác định thời điểm bắt đầu và kết thúc tấn công
- Thuộc tính *attackType* chứa tên danh mục các hình thức tấn công sẽ thực hiện, mặc định là chuỗi rỗng.

Bước 2: Lập trình hoạt động cho module này thông qua 2 tệp **.h** và **.cc**

```

NTU_Attack.h
#ifndef ATTACKS_CONTROLLERS_NTU_ATTACK_H_
#define ATTACKS_CONTROLLERS_NTU_ATTACK_H_
#define NTU_ATTACK_ACTIVATE_TAG "Activate"
#define NTU_ATTACK_DEACTIVATE_TAG "Deactivate"
#define NTU_ATTACK_END_MESSAGE "EndAttack"
#define NTU_ATTACK_START_MESSAGE "StartAttack"
#define NTU_ATTACK_TYPE "attackType"
#define NTU_ATTACK_ACTIVE "active"
#define NTU_ATTACK_END_TIME "endTime"
#define NTU_ATTACK_START_TIME "startTime"
class NTU_Attack: public cSimpleModule {
private:
    char* attackType;
    vector<cModule *> modList; //List of modules for activation of the attack
protected:
    virtual void initialize();
    void getAttackModules();
    void scheduleAttack();
    void sendMessageToHackedModules(cMessage *msg);
    virtual void activateModules();
    virtual void deactivateModules();
    virtual void handleMessage(cMessage *msg);
    virtual cMessage *generateAttackMessage(const char* name);
};
#endif /* ATTACKS_CONTROLLERS_NTU_ATTACK_H_ */

```

Hình 3.7. Tập tiêu đề của Controller

```

NTU_Attack.cc
+ * NTU_Attack.cc
#include "NTU_Attack.h"
#include "NTU_HackedModule.h"

Define_Module (NTU_Attack);
+ void NTU_Attack::initialize() {}
+ void NTU_Attack::getAttackModules() {}
+ void NTU_Attack::scheduleAttack() {}
+ void NTU_Attack::handleMessage(cMessage *msg) {}
+ void NTU_Attack::activateModules() {}
+ void NTU_Attack::deactivateModules() {}
+ void NTU_Attack::sendMessageToHackedModules(cMessage *msg) {}
- cMessage *NTU_Attack::generateAttackMessage(const char *name) {
    LOG << "ERROR: EN NTU_ATTACK GENERATE ATTACK MESSAGE\n";
    cMessage *msg = new cMessage(name);
    return msg;
}

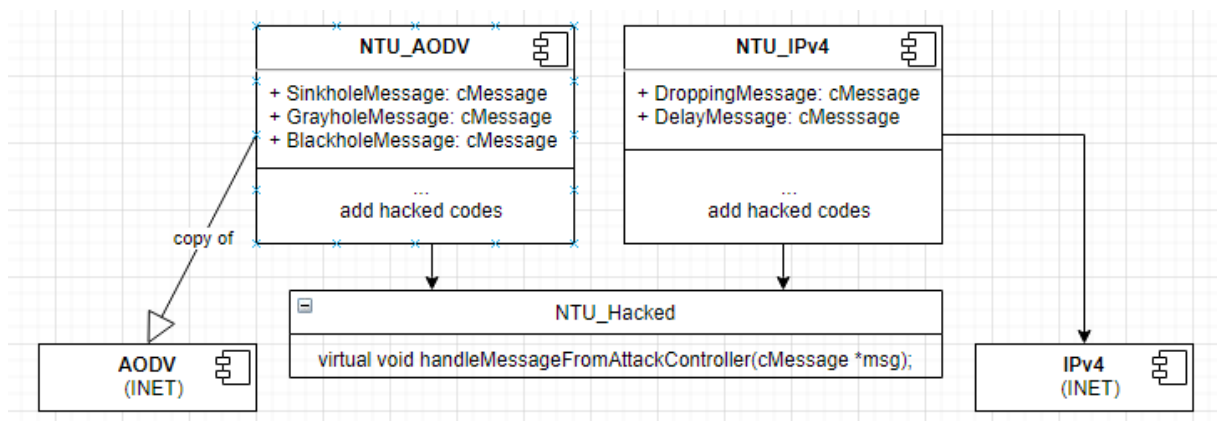
```

Hình 3. 8. Tập chứa mã xử lý của Controller

### 3. Module NTU\_IPv4 và NTU\_AODV

Về nguyên tắc hoạt động, các module bị tấn công sẽ đều phải nhận được thông điệp tấn công từ controller, do đó chúng đều phải có phương thức xử lý thông điệp này và dựa vào đó để thực hiện hành vi độc hại tương ứng.

Các phương thức trong AODV của INET không thiết kế để cho phép ghi đè phương thức, vậy nên hình thức xây dựng module NTU\_AODV ở đây sẽ là tạo ra các bản sao tương ứng và bổ sung mã lệnh độc hại vào các phương thức cần thiết.



Hình 9. Thiết kế các module có hành vi độc hại NTU\_AODV và NTU\_Ipv4

Theo trên, thiết kế mô đun cơ sở (NTU\_Hacked) cũng được triển khai cụ thể qua 2 bước như sau:

- Bước 1. Khai báo module bằng ngôn ngữ .ned

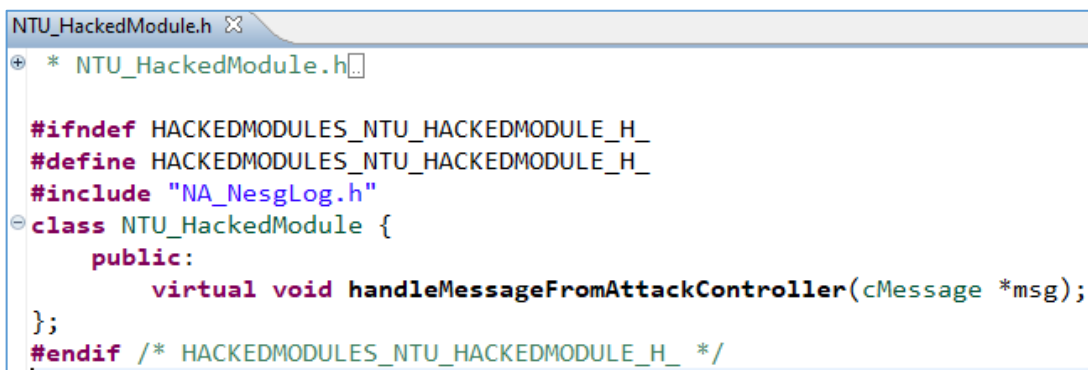
```
NTU_HackedModule.ned
package ntuattack.hackedmodules;
simple NTU_HackedModule
{
    // no thing
}
```

Hình 10. Khai báo module NTU\_Hacked

Ở module này không cần thuộc tính gì thêm, mục tiêu chỉ để định nghĩa phương thức ảo xử lý thông điệp được gửi đến từ Controller, và phải được code cụ thể khi triển khai các hacked mô đun kế thừa.

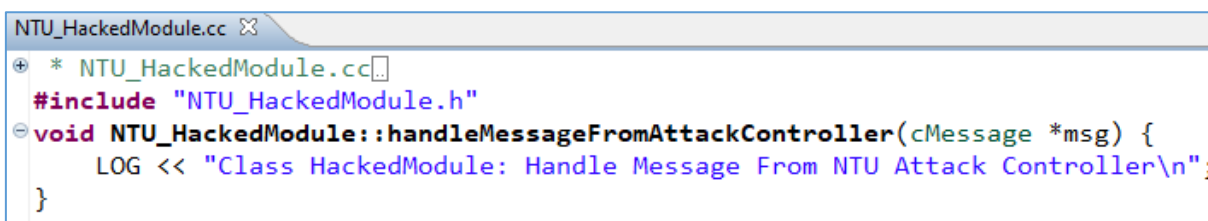


Bước 2: Xây dựng phần mã xử lý của module qua hai tệp **.h** và **.cc**



```
NTU_HackedModule.h
+ * NTU_HackedModule.h
+
+ #ifndef HACKEDMODULES_NTU_HACKEDMODULE_H_
+ #define HACKEDMODULES_NTU_HACKEDMODULE_H_
+ #include "NA_NesgLog.h"
+ class NTU_HackedModule {
+ public:
+     virtual void handleMessageFromAttackController(cMessage *msg);
+ };
+ #endif /* HACKEDMODULES_NTU_HACKEDMODULE_H_ */
```

Hình 11. Tệp tiêu đề của module NTU\_Hacked



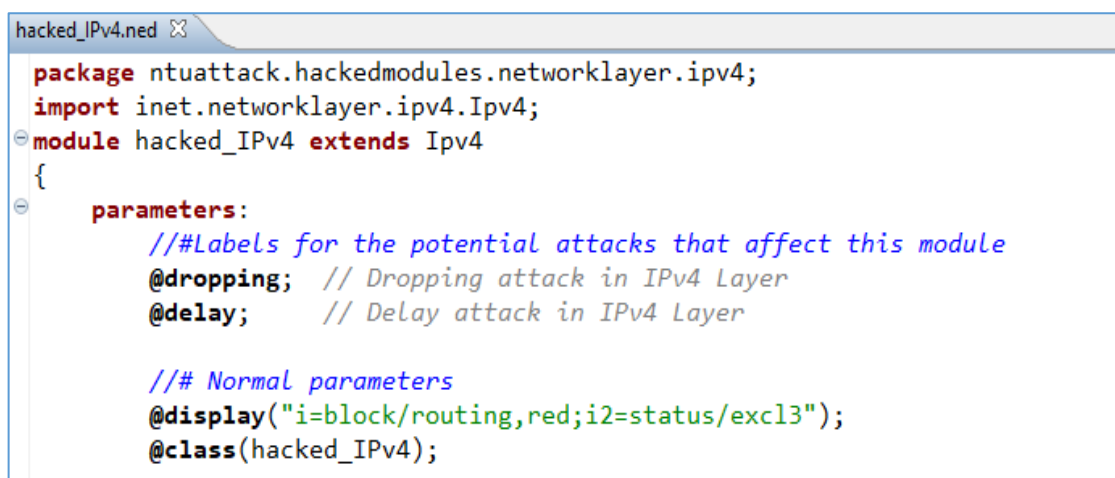
```
NTU_HackedModule.cc
+ * NTU_HackedModule.cc
+ #include "NTU_HackedModule.h"
+ void NTU_HackedModule::handleMessageFromAttackController(cMessage *msg) {
+     LOG << "Class HackedModule: Handle Message From NTU Attack Controller\n";
+ }
```

Hình 12. Tệp chứa mã xử lý của module NTU\_Hacked

### 3.1. Module NTU\_IPv4

Module sẽ thực hiện hành vi độc hại hủy gói hoặc trì hoãn việc chuyển tiếp gói tin khi nhận được thông điệp tấn công từ controller. Nghiên cứu module IPv4 của INET4 chúng tôi nhận thấy, các phương thức cần thiết cho thực hiện hành vi độc hại đều cho phép ghi đè, do vậy module này (NTU\_IPv4) sẽ được xây dựng theo cách sau:

Bước 1. Định nghĩa module sử dụng kế thừa module IPv4 bằng ngôn ngữ .ned



```
hacked_IPv4.ned
package ntuattack.hackedmodules.networklayer.ipv4;
import inet.networklayer.ipv4.Ipv4;
module hacked_IPv4 extends Ipv4
{
    parameters:
        // #Labels for the potential attacks that affect this module
        @dropping; // Dropping attack in IPv4 Layer
        @delay; // Delay attack in IPv4 Layer

        // # Normal parameters
        @display("i=block/routing,red;i2=status/excl3");
        @class(hacked_IPv4);
}
```

Hình 13. Định nghĩa module hacked\_IPv4 (NTU\_IPv4)

Mô đun này có 2 tham số qua trọng là 2 nhãn để xác định hình thức độc hại có tác dụng trên nó, đó là dropping (hủy gói) và delay (làm trễ).

Bước 2. Triển khai các xử lý cần thiết cho module qua tệp .h và .cc

```
hacked_IPv4.h

#ifndef HACKEDMODULES_NETWORKLAYER_IPV4_HACKED_IPV4_H_
#define HACKEDMODULES_NETWORKLAYER_IPV4_HACKED_IPV4_H_
using namespace inet;
using namespace omnetpp;
class INET_API hacked_IPv4 : public Ipv4, public NTU_HackedModule {
private:
    //...other codes
    bool droppingAttackIsActive;
    double droppingAttackProbability;
    bool delayAttackIsActive;
    double delayAttackProbability;
public:
    void handleMessageFromAttackController(cMessage *msg);
protected:
    /*----- DROPPING ATTACK -----*/
    virtual void handleIncomingDatagram(Packet *packet);
    /*----- DELAY ATTACK -----*/
    virtual void sendPacketToNIC(Packet *packet);
    //.. other methods
};
#endif /* HACKEDMODULES_NETWORKLAYER_IPV4_HACKED_IPV4_H_ */
```

Hình 14. Tệp tiêu đề xử lý của module NTU\_IPv4

- Phương thức cần ghi đè thứ 1 (kế thừa từ NTU\_HackedModule) để đọc thông tin trong thông điệp tấn công được gửi tới từ controller:

*virtual void handleMessageFromAttackController(cMessage \*msg);*

- Phương thức cần ghi đè thứ 2 (kế thừa từ Ipv4) để thực hiện thành vi độc hại (hủy gói) thay vì hành vi xử lý bình thường:

*virtual void handleIncomingDatagram(Packet \*packet)*

- Phương thức cần ghi đè thứ 3 (kế thừa từ Ipv4) để thực hiện thành vi độc hại (làm trễ gói) thay vì hành vi xử lý bình thường:

*virtual void sendPackeToNIC (Packet \*packet)*

```

void NTU_IPv4::handleMessageFromAttackController(cMessage *msg) {

    /*----- DROPPING ATTACK -----*/
    if ( not strcmp(msg->getFullName(), "droppingActivate")) {
        droppingAttackIsActive = true;
        ..
        // get other infos is encapsulated in message
        ..
    } else if ( not strcmp(msg->getFullName(), "droppingDeactivate")) {
        droppingAttackIsActive = false;
    }
    /*----- DELAY ATTACK -----*/
    if ( not strcmp(msg->getFullName(), "delayActivate")) {
        delayAttackIsActive = true;
        ..
        // get other infos is encapsulated in message
        ..
    } else if ( not strcmp(msg->getFullName(), "delayDeactivate")) {
        delayAttackIsActive = false;
    }
}

```

**Hình 15. Mã xử lý thông điệp tấn công căn bản từ Controller**

```

void hacked_IPv4::handleIncomingDatagram(Packet *packet)
{
    //-----Dropng UDP and TCP and ICMP-----
    if ((droppingUDPdata==true)&& (droppingTCPdata==true) && (droppingPINGdata==true)) {
        if (droppingAttackIsActive) {
            if (!strcmp(packet->getName(), UDP_DATA, 3)|| !strcmp(packet->getName(), TCP_D
            {
                if (uniform(0, 1) < droppingAttackProbability) {
                    // .. other codes
                    delete packet;
                } else {
                    Ipv4::handleIncomingDatagram(packet);
                }
            } else { //Packet is not a data packet --> normal behavior
                Ipv4::handleIncomingDatagram(packet);
            }
        }
        else { // --> Normal behavior., it means droppingAttack Is NOT Active: droppingAttackI
            Ipv4::handleIncomingDatagram(packet);
        }
    }
    //-----Dropng UDP and TCP-----
} else if ((droppingUDPdata==true)&& (droppingTCPdata==true) && (droppingPINGdata==false)) {
    //...more proccessing codes
    ...
}

```

**Hình 16. Mã xử lý căn bản khi một Packet đến tầng mạng của module NTU\_IPv4**

Từ xử lý trên 15, nút độc hại sẽ có thông tin cho việc thực hiện hành vi của mình ở hình 16. Xử lý ở đây là xóa gói (drop/delete packet) toàn bộ, chọn lọc, theo xác suất.

```

void hacked_IPv4::sendPacketToNIC(Packet *packet)
{
    if(delayAttackIsActive){
        if (!strcmp(packet->getName(), PING_DATA, 4) || !strcmp(packet->getName(), UDP_DATA))
        {
            if (uniform(0, 1) < delayAttackProbability) {
                packet->addTagIfAbsent<PacketProtocolTag>()->setProtocol(&Protocol::ipv4);
                packet->addTagIfAbsent<DispatchProtocolInd>()->setProtocol(&Protocol::ipv4);
                delete packet->removeTagIfPresent<DispatchProtocolReq>();
                ASSERT(packet->findTag<InterfaceReq>() != nullptr);
                // Perform send packet out with delay times is 0.1s
                sendDelayed(packet, 0.1, "queueOut");
                EV_INFO << "DELAY_END:" << simTime() << endl;
                return;
            }
        }
    }
    Ipv4::sendPacketToNIC(packet);
}

```

**Hình 17. Mã xử lý delay Packet ở tầng mạng của module NTU\_IPv4**

Đoạn mã hình 3.17 thực hiện kiểm tra trạng thái có kích hoạt tấn công delay không, và gói dữ liệu có tên là UDP, TCP, PING hay không. Nếu thỏa mãn thì sinh ngẫu nhiên một xác suất, nếu nhỏ hơn xác suất cho phép thì thực hiện kỹ thuật phát trễ gói.

Hàm chính cho phát trễ gói trong xử lý này là *sendDelayed*. Đoạn mã trên thực hiện phát trễ 0.1 giây.

### 3.2. Module NTU-AODV

Như đã phân tích ở trên, giao thức AODV gốc được thiết kế không cho phép ghi đè các phương thức xử lý, như vậy cần xây dựng một “bản sao” của nó và chỉnh code để thực hiện các xử lý độc hại.

Module NTU-AODV sẽ được xây dựng dựa trên việc copy bản AODV gốc, và chèn thêm các đoạn mã độc vào các phương thức phù hợp để thực hiện hành vi độc hại làm sai lệch kết quả định tuyến theo hướng thiết lập lộ trình kết quả đi qua nó, để từ đó có thể thực hiện các hành vi tấn công độc hại khác.

Trong các cuộc tấn công tấn công lỗ đen vào AODV cũng như các hình thức tấn công khác vào nó trên tầng mạng như: tấn công lỗ chìm, tấn công lỗ xám và tấn công lỗ sau. Kỹ thuật thực hiện gồm 2 bước:

**Bước 1**, Khi nhận gói yêu cầu tuyến RREQ, các nút độc hại trả lời ngay lập tức cho nút nguồn (thông qua gói RREP giả mạo) rằng nó có lộ trình tốt nhất đến đích (số chặng

tới đích Hopcount = 1 và số SN = MAX), nhằm lừa nút nguồn chọn đây là tuyến tốt nhất tới đích và sẽ truyền dữ liệu đến đích thông qua nó.

**Bước 2**, Thực hiện hành vi độc hại: xóa bỏ tất cả các gói, xóa theo một tỉ lệ nhất định, xóa theo loại gói nhất định...Bước 2 được thực hiện thông qua việc triển khai thêm hình thức tấn công dropping và delay vào giao thức IPv4 (module NTU\_IPv4).

Thực hiện 1, cần xử lý trên 3 phương thức sau:

- Phương thức xử lý khi nhận được gói yêu cầu tuyến: **handleRREQ**.
- Phương tạo gói trả lời: **CreateRREP**
- Phương thức xử lý gói RREP: **handleRREP**

Phương thức **handleRREQ** là hàm được gọi khi có một RREQ gửi đến node. Thêm đoạn mã sau vào đầu hàm, đảm bảo tạo ra gói trả lời yêu cầu tuyến giả mạo RREP và gửi trả lời ngay và không xử lý gì thêm khi có hành động tấn công phù hợp xảy ra.

```
void NTU_Aodv::handleRREQ(const Ptr<Rreq>& rreq, const L3Address& sourceAddr, unsign
{
    ....
    /*-----Attack code begin-----
    if ( ((sinkholeAttackIsActive ==true)|| (grayholeAttackIsActive==true) ||
        (blackholeAttackIsActive==true)|| (wormholeAttackIsActive==true) ) )
    {
        auto rrep = createRREP(rreq, destRoute, reverseRoute, sourceAddr);
        // send to the originator
        sendRREP(rrep, rreq->getOriginatorAddr(), 255);
        //delete rreq;
        return;    // discard RREQ, in this case, we do not forward it.
    }
    /*-----Attack code -----
    ....
}
```

**Hình 18.** Xử lý RREQ tại node độc hại trong module NTU\_AODV

+ Phương thức **CreateRREP** được gọi ở hàm trên phải thực hiện được hành vi sinh ra gói giả mạo nếu việc tấn công đã được kích hoạt. Có 2 thông tin cần giả mạo để thực hiện được hành vi độc hại là Số chặng (HopCount) =1 đảm bảo là đường đi ngắn nhất và số Destination Sequence Number = MAX\_SEQUENCE\_NUM đảm bảo là đủ mới. Mã xử lý như hình dưới đây.

```

const Ptr<Rrep> NTU_Aodv::createRREP(const Ptr<Rreq>& rreq, IRoute *destRoute, IRO
{
    auto rrep = makeShared<Rrep>(); // TODO: "AODV-RREP");
    ....

    // sinkhole/grayhole/blackhole/wormhole attack -----
    if ( ((sinkholeAttackIsActive==true)|| (grayholeAttackIsActive==true) ||
        (blackholeAttackIsActive==true)|| (wormholeAttackIsActive==true) ) )
    {
        rrep->setDestSeqNum(MAX_SEQUENC_NUM);
        rrep->setHopCount(1);
        rrep->setLifeTime(myRouteTimeout.trunc(SIMTIME_MS));
        return rrep;
    }
    //-----
    .....
}

```

**Hình 19. Xử lý tạo RREP giả mạo tại node độc hại trong module NTU\_AODV**

Để nhận thông tin của mỗi hành vi tấn công, ta override phương thức để xử lý lấy thông tin, hàm *virtual void handleMessageFromAttackController(cMessage \*msg)*.

```

void NTU_Aodv::handleMessageFromAttackController(cMessage *msg){
    if (not strcmp(msg->getFullName(), "blackholeActivate")) { //BLACKHOLE ATTACK IS ACTIVATED
        blackholeAttackIsActive = true;
        // get other infos from attacker control message
        ..
    } else if (not strcmp(msg->getFullName(), "blackholeDeactivate")) { //BLACKHOLE ATTACK IS deACTIVATED
        blackholeAttackIsActive = false;
        ..
    } else if (not strcmp(msg->getFullName(), "sinkholeActivate")) { //SINKHOLE ATTACK IS ACTIVATED
        sinkholeAttackIsActive = true;
        // get other infos from attacker control message
        ..
    } else if (not strcmp(msg->getFullName(), "sinkholeDeactivate")) { //BLACKHOLE ATTACK IS deACTIVATED
        sinkholeAttackIsActive = false;
        ..
    } else if (not strcmp(msg->getFullName(), "grayholeActivate")) { //GRAYHOLE ATTACK IS ACTIVATED
        grayholeAttackIsActive = true;
        // get other infos from attacker control message
        ..
    } else if (not strcmp(msg->getFullName(), "grayholeDeactivate")) { //GRAYHOLE ATTACK IS deACTIVATED
        grayholeAttackIsActive = false;
        ..
    } else if (not strcmp(msg->getFullName(), "wormholeActivate")) { //WORMHOLE ATTACK IS ACTIVATED
        wormholeAttackIsActive = true;
    } else if (not strcmp(msg->getFullName(), "wormholeDeactivate")) { //WORMHOLE ATTACK IS deACTIVATED
        wormholeAttackIsActive = false;
        // get other infos from attacker control message
        ...
    }
}

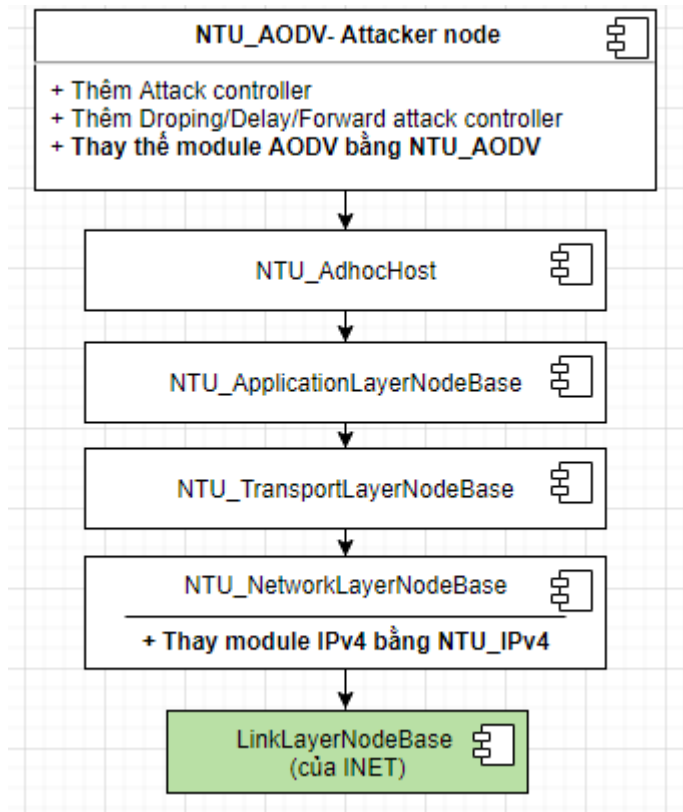
```

**Hình 20. Xử lý của NTU\_AODV khi nhận được thông điệp tấn công**

#### 4. Xây dựng các nút mạng độc hại

Theo như thiết kế, ứng với mỗi hình thức tấn công cần xây dựng một controller và một thông điệp tương ứng. Các bước xây như sau:

- *Bước 1:* Xây dựng controller kế thừa từ NTU\_Attack\_controller
- *Bước 2:* Xây dựng một cấu trúc dữ liệu mang thông điệp cho cuộc tấn công
- *Bước 3:* Các nút tấn công được tạo ra bởi việc thay thế các module IPv4 và AODV bởi NTU\_IPv4 và NTU\_AODV tương ứng, và gắn thêm controller vào.

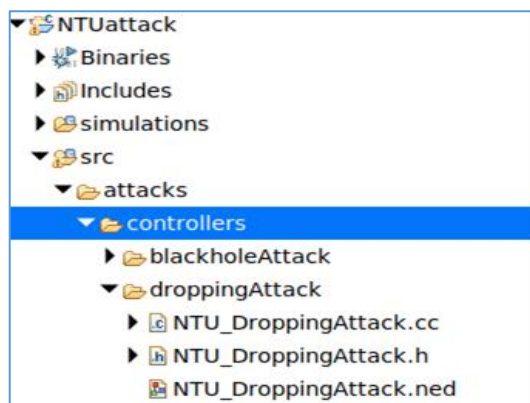


Hình 21. Thiết kế node độc hại

##### 4.1. Xây dựng Controller cho nút thực hiện hành vi các hành vi độc hại

Mục này trình bày cụ thể việc xây dựng một nút thực hiện tấn công hủy gói (Dropping Attack) như một hình mẫu cho việc xây dựng các nút độc hại khác.

- Bước 1: Xây dựng **Controller**: được xây dựng bởi 3 tệp (.h, .cc, và .ned)



Hình 22. Tổ chức dự án của NTUAttack với thư mục chứa controller của các tấn công



```

NTU_DroppingAttack.ned
1 package ntuattack.attacks.controllers.droppingAttack;
2 import ntuattack.attacks.controllers.NTU_Attack;
3
4 simple NTU_DroppingAttack extends NTU_Attack
5 {
6     parameters:
7         @class(NTU_DroppingAttack);
8         @display("i=misc/cloud3,red,100;is=l");
9         attackType = "dropping";
10        double droppingAttackProbability = default(0); //Probability of dropping a packet.
11        bool droppingUDPdata=default(false);           //Drop data transfer by UDP protocol
12        bool droppingTCPdata=default(false);           //Drop data transfer by UDP protocol
13        bool droppingPINGdata=default(false);          //Drop data transfer by ICMP protocol
14 }

```

Hình 23. Mô tả controller NTU\_DroppingAttack bằng ngôn ngữ ned

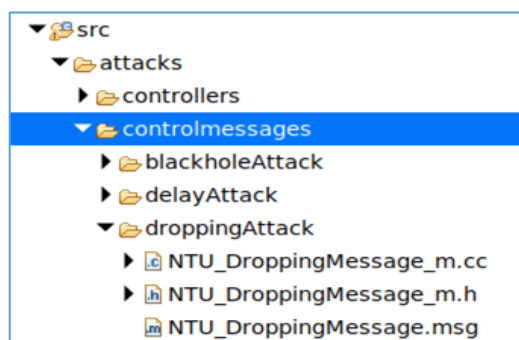
```

NTU_DroppingAttack.h
1 /*
2  * NTU_DroppingAttack.h
3  *
4  * Created on: Mar 6, 2020
5  * Author: thomc
6  */
7
8 #ifndef ATTACKS_CONTROLLERS_DROPPINGATTACK_NTU_DROPPINGATTACK_H_
9 #define ATTACKS_CONTROLLERS_DROPPINGATTACK_NTU_DROPPINGATTACK_H_
10 #include "NTU_Attack.h"
11 #include "NTU_DroppingMessage_m.h"
12
13 class NTU_DroppingAttack: public NTU_Attack {
14
15 protected:
16     /**
17      * Overridden function
18      */
19     virtual cMessage *generateAttackMessage(const char* name);
20 };
21
22 #endif /* ATTACKS_CONTROLLERS_DROPPINGATTACK_NTU_DROPPINGATTACK_H_ */

```

Hình 24 Tập tiêu đề của xử lý của controller NTU\_Dropping\_Attack

#### 4.2. Xây dựng cấu trúc thông điệp cho nút



Hình 25. Tổ chức dự án của NTUAttack với thư mục ControlMessage chứa cấu trúc thông điệp của các hình thức tấn công



```

1  *NTU_DroppingMessage_m.h
2  #ifndef NTU_DROPPINGMESSAGE_M_H
3  #define NTU_DROPPINGMESSAGE_M_H
4  class NTU_DroppingMessage : public ::omnetpp::cMessage
5  {
6  protected:
7      double droppingAttackProbability;
8      bool droppingUDPdata;
9      bool droppingTCPdata;
10     bool droppingPINGdata;
11 private:
12     void copy(const NTU_DroppingMessage& other);
13 protected:
14     bool operator==(const NTU_DroppingMessage&);
15 public:
16     NTU_DroppingMessage(const char *name=NULLptr, short kind=0);
17     NTU_DroppingMessage(const NTU_DroppingMessage& other);
18     virtual ~NTU_DroppingMessage();
19     NTU_DroppingMessage& operator=(const NTU_DroppingMessage& other);
20     virtual NTU_DroppingMessage *dup() const override {return new NTU_DroppingMessage(*this);}
21     virtual void parsimPack(omnetpp::cCommBuffer *b) const override;
22     virtual void parsimUnpack(omnetpp::cCommBuffer *b) override;
23
24     // field getter/setter methods
25     virtual double getDroppingAttackProbability() const;
26     virtual void setDroppingAttackProbability(double droppingAttackProbability);
27     virtual bool getDroppingUDPdata() const;
28     virtual void setDroppingUDPdata(bool droppingUDPdata);
29     virtual bool getDroppingTCPdata() const;
30     virtual void setDroppingTCPdata(bool droppingTCPdata);
31     virtual bool getDroppingPINGdata() const;
32     virtual void setDroppingPINGdata(bool droppingPINGdata);
33 }

```

Hình 26. Tập tiêu đề của thông điệp tấn công hủy gói

```

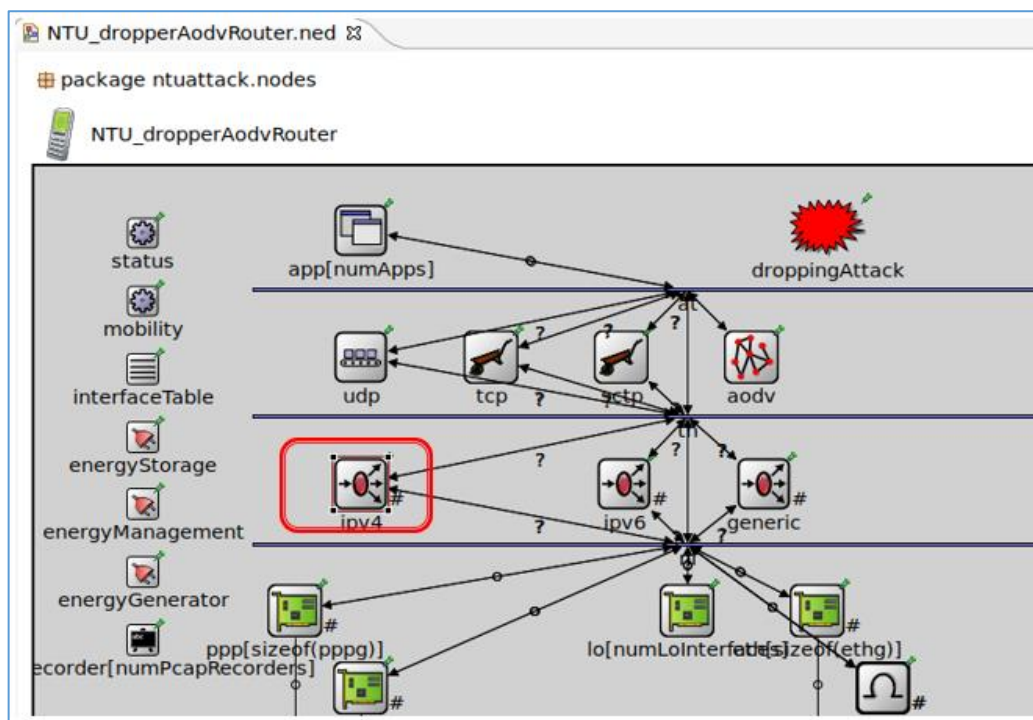
1  NTU_DroppingAttack.cc
2  #include "NTU_DroppingAttack.h"
3  #include "NTU_DroppingMessage_m.h"
4
5  Define_Module (NTU_DroppingAttack);
6
7  cMessage *NTU_DroppingAttack::generateAttackMessage(const char *name) {
8
9      LOG << "NTU_DroppingAttack: generateAttackMessage\n";
10
11      NTU_DroppingMessage *msg = new NTU_DroppingMessage(name);
12      msg->setDroppingAttackProbability(par("droppingAttackProbability").doubleValue());
13      msg->setDroppingPINGdata(par("droppingPINGdata").boolValue());
14      msg->setDroppingTCPdata(par("droppingTCPdata").boolValue());
15      msg->setDroppingUDPdata(par("droppingUDPdata").boolValue());
16      return msg;
17 }

```

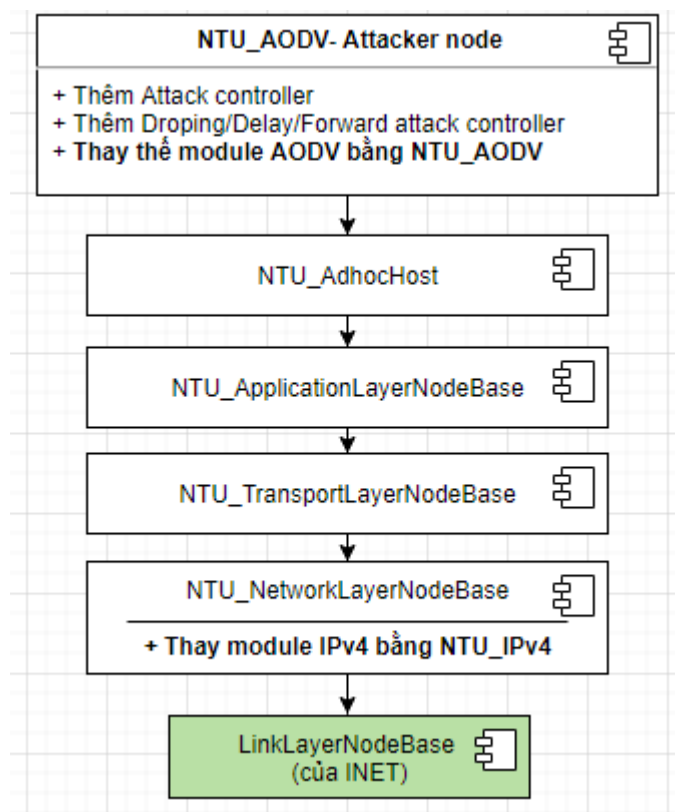
Hình 27. Tập mã nguồn của thông điệp tấn công hủy gói

#### 4.3. Tạo nút mạng tấn công

Được xây dựng bằng cách gắn Controller vào node đã được hack module IPv4 theo qui trình mô tả trên hình 20



Hình 28. Trục quan kiến trúc nút độc hại thực hiện hành vi hủy gói



Hình 29. Xây dựng node từ kế thừa các node cơ sở

```

1
2 package ntuattack.nodes;
3 import ntuattack.nodes.NTU_AodvRouter;
4 import ntuattack.attacks.controllers.droppingAttack.NTU_DroppingAttack;
5
6 module NTU_dropperAodvRouter extends NTU_AodvRouter
7 {
8     submodules:
9         // Include dropping attack controller
10        droppingAttack: NTU_DroppingAttack
11        {
12            @display("p=911.925,74.024994");
13            active = true;
14            startTime = 0s;
15            endTime = 15s;
16            droppingAttackProbability = 1;
17            droppingUDPdata = true;
18            droppingTCPdata = true;
19            droppingPINGdata = true;
20        }
21 }

```

**Hình 30. Mã ngôn ngữ ned mô tả nút**

Hình trên 29 mô tả việc gắn Dropping Attack Controller và nút, hành vi độc hại kích hoạt ngay từ đầu, bắt đầu tấn công ngay giây đầu tiên và kết thúc ở giây thứ 15. Tấn suất thực hiện là 1 (100%). Hủy toàn bộ dữ liệu sử dụng giao thức nên TCP, UDP, và ICMP

**Các hình thức tấn công khác như: tấn công lỗ đen, tấn công lỗ xám, tấn công lỗ chìm và các hình thức tấn công khác sẽ được thực hiện tương tự**

## 5. Xây dựng kịch bản và mô phỏng

Việc mô phỏng và đánh giá mô phỏng được thực hiện như trên lõi OMNeT++ thông thường, do đây là nền tảng được xây dựng để chạy trên OMNeT++5.6 sử dụng hệ thư viện INET4. Với mỗi mô phỏng cần triển khai 2 bước: Xây dựng module mạng mô phỏng, và qui định giá trị các tham số cho cấu hình mô phỏng.

- *Bước 1.* Xây dựng module mạng mô phỏng

Việc xây dựng module mạng mô phỏng có thể được thực hiện theo 2 cách: sử dụng phương pháp kéo thả trực quan hoặc phương pháp viết trực tiếp mã ngôn ngữ .ned.

Hình dưới đây trình bày mã xây dựng mạng mô phỏng với 2 nút gửi AodvSender AodvSender1 và các nút nhận tương ứng là AodvReceiver và AodvReceiver1, 2 node

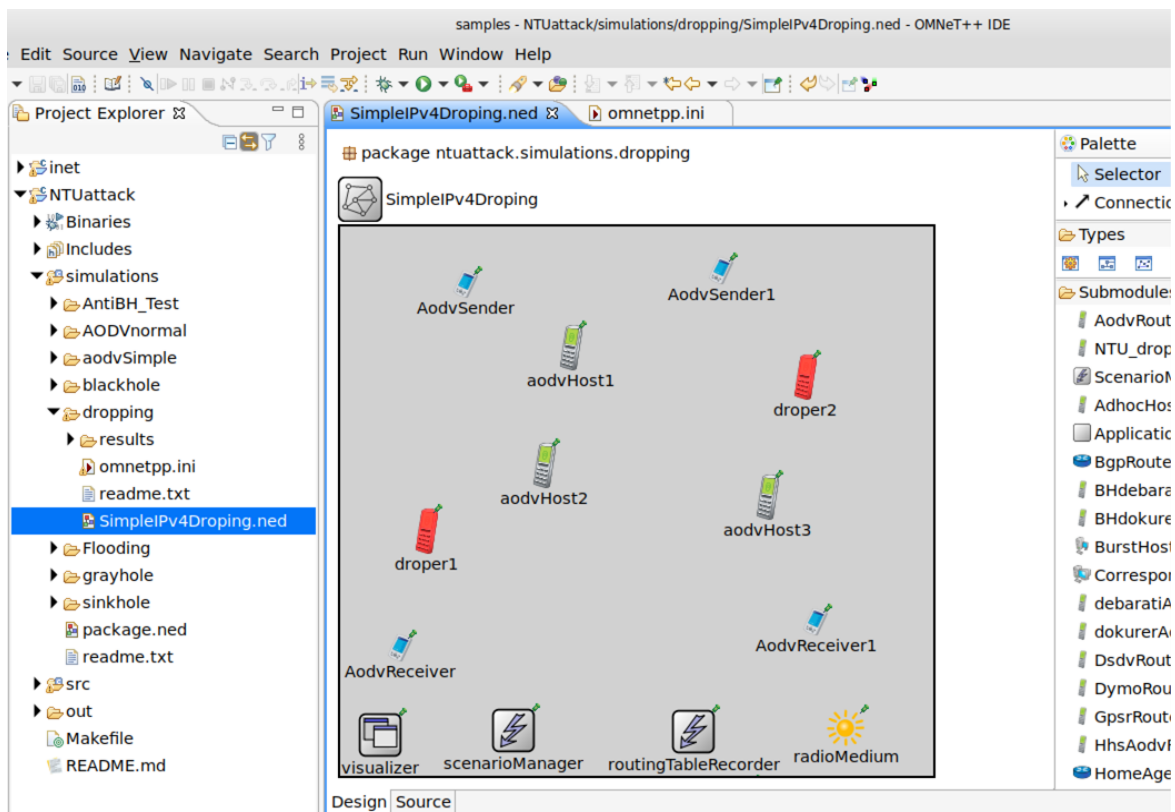
thực hiện tấn công hủy gói là droper1 và droper2, thêm 3 nút mạng bình thường aodvHost1, aodvHost2, aodvHost3.

```

SimpleIPv4Dropping.ned  omnetpp.ini
1 package ntuattack.simulations.dropping;
2 import inet.common.scenario.ScenarioManager;
3 import inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurator;
4 import inet.networklayer.ipv4.RoutingTableRecorder;
5 import inet.node.aodv.AodvRouter;
6 import inet.physicallayer.unitdisk.UnitDiskRadioMedium;
7 import inet.visualizer.contract.IIntegratedVisualizer;
8 import ntuattack.nodes.NTU_AodvRouter;
9 import ntuattack.nodes.NTU_dropperAodvRouter;
10 network SimpleIPv4Dropping {
11     parameters:
12         @figure[title](type=label; pos=0,0; anchor=sw; color=darkblue);
13         @display("bgb=534,443");
14     submodules:
15         visualizer: <default("IntegratedCanvasVisualizer")> like IIntegratedVisualizer if hasVisualizer() { @display("p=284,491;is=s");
16         radioMedium: UnitDiskRadioMedium { parameters: @display("p=363,500;is=s");
17         configurator: Ipv4NetworkConfigurator { parameters: @display("p=190,492;is=s");
18             config = xml("<config><interface hosts='*' address='145.236.x.x' netmask='255.255.0.0' /></config>");
19         routingTableRecorder: RoutingTableRecorder { parameters: @display("p=106,496;is=s");
20         scenarioManager: ScenarioManager { parameters: script = default(xml("<scenario/>")); @display("p=106,496;is=s");
21
22         AodvSender: AodvRouter { parameters: @display("i=device/pocketpc_s;r=,,#707070;p=113,51");
23         AodvSender1: AodvRouter { parameters: @display("i=device/pocketpc_s;r=,,#707070;p=343,39");
24         AodvReceiver: AodvRouter { parameters: @display("i=device/pocketpc_s;r=,,#707070;p=57,393");
25         AodvReceiver1: AodvRouter { parameters: @display("i=device/pocketpc_s;r=,,#707070;p=428,349");
26
27         droper1: NTU_dropperAodvRouter { @display("p=82,297;i=,EF2929,90");
28         droper2: NTU_dropperAodvRouter { @display("p=418,133;i=,EF2929,90");
29
30         aodvHost1: AodvRouter { @display("p=207,107;b=5,10");
31         aodvHost2: AodvRouter { @display("p=185,228");
32         aodvHost3: AodvRouter { @display("p=384,239");
33     connections allowunconnected:
34 }

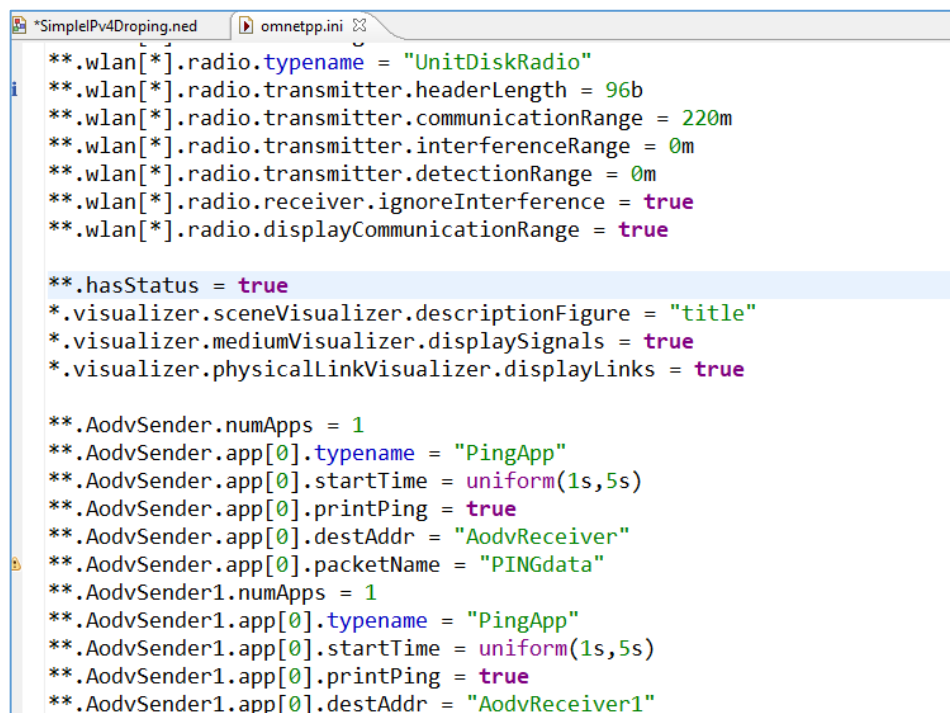
```

Hình 31. Xây dựng mạng mô phỏng bằng ngôn ngữ .ned



Hình 32. Trực quan mạng mô phỏng theo như 15

- Bước 2. Qui định các tham số cho mô phỏng (file .ini)



```

**wlan[*].radio.typename = "UnitDiskRadio"
**wlan[*].radio.transmitter.headerLength = 96b
**wlan[*].radio.transmitter.communicationRange = 220m
**wlan[*].radio.transmitter.interferenceRange = 0m
**wlan[*].radio.transmitter.detectionRange = 0m
**wlan[*].radio.receiver.ignoreInterference = true
**wlan[*].radio.displayCommunicationRange = true

**hasStatus = true
*.visualizer.sceneVisualizer.descriptionFigure = "title"
*.visualizer.mediumVisualizer.displaySignals = true
*.visualizer.physicalLinkVisualizer.displayLinks = true

**.AodvSender.numApps = 1
**.AodvSender.app[0].typename = "PingApp"
**.AodvSender.app[0].startTime = uniform(1s,5s)
**.AodvSender.app[0].printPing = true
**.AodvSender.app[0].destAddr = "AodvReceiver"
**.AodvSender.app[0].packetName = "PINGdata"
**.AodvSender1.numApps = 1
**.AodvSender1.app[0].typename = "PingApp"
**.AodvSender1.app[0].startTime = uniform(1s,5s)
**.AodvSender1.app[0].printPing = true
**.AodvSender1.app[0].destAddr = "AodvReceiver1"

```

**Hình 33.** Qui định giá trị tham số mô phỏng