پیاده سازی شبکه MANET
با استفاده از OMNETPP

تهیه و تنظیم: مسعود شیخ حسینی

مقدمه

ابتدا در این مقاله ، حرکت های مختلف گره ها را که پیاده سازی شده اند بررسی میکنیم و سپس به کالبد شکافی الگوریتم مسیریابی {ای-او-دی-وی} میپردازیم و در ادامه با انجام تغییراتی در این الگوریتم ، امکان معرفی و عضو شدن در سرویس های مختلف توسط گره ها را ممکن میکنیم.
نکته مهمی که باید به آن توجه شود این است که بدون افزودن فریمورک {آی-نت} به نرم افزار {آم-نت} امکان انجام موارد گفته شده ممکن نیست.

حرکت - Mobility

حرکت های متنوع و جالبی در مثال های فریمورک {آی-نت} پیاده سازی شده اند اما فقط مواردی که در پروژه پیاده کردم را توضیح میدهم.

- **LinearMobility**

```
**.host*.mobility.typename = "LinearMobility"
**.host*.mobility.initFromDisplayString = false
**.host*.mobility.speed = 50mps
**.host*.mobility.angle = 30deg
**.host*.mobility.initialX = uniform(100m,150m)
**.host*.mobility.initialY = uniform(100m,200m)
```

حرکت به صورت خطی میباشد. گروه گره ها دسته جمعی حرکت میکنند.
برای این حرکت نیاز است دو مشخصه تعیین شود:
۱. سرعت گره ها: که عدد را نوشته و جلوی آن واحد مورد نظر را قرار میدهیم.
۲. زاویه: زاویه حرکتی را مینویسیم و واحد مدنظر را هم قرار میدهیم.

- **CircleMobility**

```
**.host*.mobility.typename = "CircleMobility"
# SMALL CIRCLE
**.host[0..11].mobility.cx = 400m
**.host[0..11].mobility.cy = 400m
**.host[0..11].mobility.r = 100m
**.host[0..11].mobility.speed = 2000mps
```

حرکت به صورت دایره ای میباشد. در اینجا هم گره ها دست جمعی حرکت میکنند.

چندین مشخصه باید تعیین شود:

۱. نقطه مرکزی دایره ما در محور افقی مختصات.

۲. نقطه مرکزی دایره ما در محور عمودی مختصات.

۳. شعاع دایره

۴. سرعت حرکت

سپس موقعیت هر گره را مشخص میکنیم. موقعیت را میتوانیم خیلی راحت تر با زاویه نسبت به مرکز دایره مشخص کنیم.

من تک تک گره ها را جدا موقعیت مکانی کرده ام ، اما امکان تعیین موقعیت دسته ای گره ها هم ممکن است.

```
**.host[1].mobility.startAngle = 0deg
**.host[2].mobility.startAngle = 30deg
**.host[3].mobility.startAngle = 60deg
**.host[4].mobility.startAngle = 90deg
**.host[5].mobility.startAngle = 120deg
**.host[6].mobility.startAngle = 150deg
**.host[7].mobility.startAngle = 180deg
**.host[8].mobility.startAngle = 210deg
**.host[9].mobility.startAngle = 240deg
**.host[10].mobility.startAngle = 270deg
**.host[11].mobility.startAngle = 300deg
**.host[0].mobility.startAngle = 330deg
```

- **AttachedMobility**

حرکت به صورت متصل میباشد. یعنی قادریم یک حرکت واحد را برای فقط یک گره مشخص کنیم و به اصطلاح بقیه گره ها را مجبور به تقلید از حرکت آن گره کنیم.
در اینجا تعریف کمی متفاوت است:

```
*.visualizer.mobilityVisualizer.moduleFilter = "**.mobility"

**.host[0].mobility.typename = "CircleMobility"
**.host[0].mobility.cx = 300m
**.host[0].mobility.cy = 200m
**.host[0].mobility.r = 150m
**.host[0].mobility.speed = 40mps

# other hosts are also moving around in a larger circle following host[0]
**.host[*].mobility.typename = "AttachedMobility"
**.host[*].mobility.mobilityModule = "^.^.host[0].mobility"
```

یک حرکت دایره ای برای گره ۰ام مشخص کرده ایم. حال برای بقیه گره ها با فرمتی که در کد مشخص شده ، حرکت را طوری تعریف میکنیم که گره ها دقیقا حرکت گره ۰ را انجام دهند.
اما در اینجا مشکلی وجود دارد ، در این حالت و تا اینجا دقیقا همه گره ها در مکان گره ۰ متمرکز شده اند و فقط یک نقطه پر از گره داریم. باید برای باقی گره ها فاصله مشخص کنیم تا حالت یک دنباله (قطاری) از گره ها داشته باشیم که پشت سر گره ۰ در حرکت اند.

به این صورت مشکل را حل میکنیم.
برای هرگره یک فاصله مشخص از محور افقی و یک فاصله زاویه ای تعیین میکنیم. به این صورت تا وقتی که یک دور کامل از دایره را مقدار دهی نکرده باشیم ، میتوانیم گره هایی با فاصله های مشخص و زیاد داشته باشیم.

```
**.host[1].mobility.offsetX = 10m
**.host[1].mobility.offsetHeading = 90deg

**.host[2].mobility.offsetX = -50m
**.host[2].mobility.offsetHeading = -120deg

**.host[3].mobility.offsetX = 80m
**.host[3].mobility.offsetHeading = 130deg

**.host[4].mobility.offsetX = 150m
**.host[4].mobility.offsetHeading = 150deg

**.host[5].mobility.offsetX = -120m
**.host[5].mobility.offsetHeading = 180deg

**.host[6].mobility.offsetX = 190m
**.host[6].mobility.offsetHeading = 200deg

**.host[7].mobility.offsetX = 200m
**.host[7].mobility.offsetHeading = 230deg

**.host[8].mobility.offsetX = 220m
**.host[8].mobility.offsetHeading = 250deg

**.host[9].mobility.offsetX = -160m
**.host[9].mobility.offsetHeading = 290deg
```

- **SuperpositioningMobility**

این حرکت ترکیبی میباشد. یعنی ما قادریم برای گره ها چندین حرکت مشخص کنیم و فقط محدود به یک حرکت مشخص نباشیم.
در اینجا تعریف کمی متفاوت است:

۱. ابتدا تعداد حرکت ها برای هر گره را مشخص میکنیم.

```
**.host[*].mobility.numElements = 2
```

حال باید دو حرکت را
مشخص کنیم و حتما باید به تعدادی که در بالا گفتیم حرکت ها را تعریف کنیم وگرنه برنامه به مشکل میخورد.

اولین حرکت:
همانطور که قبلا با این حرکت آشنا شدیم ، میدانیم که حرکت دایره ای است.

```
# ---- 1st movement
**.host*.mobility.element[0].typename = "CircleMobility"
**.host[0..11].mobility.element[0].cx = 200m
**.host[0..11].mobility.element[0].cy = 200m
**.host[0..11].mobility.element[0].r = 90m
**.host[0..11].mobility.element[0].speed = 2000mps
**.host[1].mobility.element[0].startAngle = 0deg
**.host[2].mobility.element[0].startAngle = 30deg
**.host[3].mobility.element[0].startAngle = 60deg
**.host[4].mobility.element[0].startAngle = 90deg
**.host[5].mobility.element[0].startAngle = 120deg
**.host[6].mobility.element[0].startAngle = 150deg
**.host[7].mobility.element[0].startAngle = 180deg
**.host[8].mobility.element[0].startAngle = 210deg
**.host[9].mobility.element[0].startAngle = 240deg
**.host[10].mobility.element[0].startAngle = 270deg
**.host[11].mobility.element[0].startAngle = 300deg
**.host[0].mobility.element[0].startAngle = 330deg
```

دومین حرکت:

```
# ---- 2nd movement
**.host*.mobility.element[1].typename = "LinearMobility"
**.host*.mobility.element[1].initFromDisplayString = false
**.host*.mobility.element[1].speed = 50mps
**.host*.mobility.element[1].angle = 30deg
**.host*.mobility.element[1].initialX = uniform(100m,150m)
**.host*.mobility.element[1].initialY = uniform(100m,200m)
```

حرکت خطی میباشد و مشخصه های لازم آن را هم تعیین کردیم.

تمام حرکت های تعریف شده در این داکیومنت ، پیوست شده اند.

سفارشی سازی یک الگوریتم مسیریابی - AODV in MANET

توضیحات مربوط به طرز کار این الگوریتم حتما لازمه کار است اما در اینجا به دلیل تکراری بودن مباحث و تمرکز روی کدها به آن نمیپردازیم و مستقیم سراغ هسته این الگوریتم میرویم.
فایلها در این مسیر وجود دارند:
/src/inet/routing/aodv/

توجه: در همه کدهای تغییر داده/اضافه شده ، پیشوند zz وجود دارد که نشانی اختصاری من در تمام پروژه هایی است که من در توسعه آنها نقش داشتم.

با توجه به سناریو تعریف شده ، برای شروع ابتدا :

۱. به پروتکل قابلیت تعریف کردن سرویس های مورد نیاز و سرویس های تامین شده را میدهیم.

```
std::string zzAvailableServices = "0";
std::string zzRequestedServices = "0";
```

اگر از دیدگاه برنامه نویسی شی گرا نگاه کنیم ، در حقیقت متغیر های جدید برای کلاس تعریف کرده ایم.

و علاوه بر هدر در فایل اصلی هم تعریف میکنیم:

```
zzAvailableServices = par("zzAvailableServices");
zzRequestedServices = par("zzRequestedServices");
```

۲. حال طبق استانداردی که در کدها تعریف شده میبینیم، برای این متغیرهای جدید اضافه شده ، در تمامی ۳ کلاس اصلی (RREQ,RREP,RERR) برای آنها getter-setter تعریف میکنیم.
برای مثال در کلاس RREQ داریم:

```
std::string Rreq::getZZRequestedServices_req() const
{
    return this->zzRequestedServices_req;
}

void Rreq::setZZRequestedServices_req(std::string requestedServices)
{
    handleChange();
    this->zzRequestedServices_req = requestedServices;
}
```

۳. در ۳ کلاس اصلی گفته شده ، تابعی داریم که بسته ها را کپی میکند و بسته های جدیدی از آنها تولید میکند تا برای پخش های مختلف استفاده کند. باید در این تابع (copy) ، متغیر های تعریف شده جدید را هم کپی میکنیم:

```
void Rreq::copy(const Rreq& other)
{
    this->joinFlag = other.joinFlag;
    this->repairFlag = other.repairFlag;
    this->gratuitousRREPFlag = other.gratuitousRREPFlag;
    this->destOnlyFlag = other.destOnlyFlag;
    this->unknownSeqNumFlag = other.unknownSeqNumFlag;
    this->hopCount = other.hopCount;
    this->rreqId = other.rreqId;
    this->destAddr = other.destAddr;
    this->destSeqNum = other.destSeqNum;
    this->originatorAddr = other.originatorAddr;
    this->originatorSeqNum = other.originatorSeqNum;

    this->zzRequestedServices_req = other.zzRequestedServices_req;
}
```

باید به این نکته توجه کرد که فقط سرویس های درخواست شده (موردنیاز) در بسته ها جابه جا میشوند و سرویس های تامین شده (سرویس دهنده) در روتر ها تعریف شده اند و نباید در بسته ها جا به جا شوند.

۴. منطق مهم تصمیم گیری باید با توجه به سناریو جدید ما انجام شود:
اگر روتر سرویس خواسته شده در بسته را میتواند تامین کند -> ادامه مسیریابی را ادامه بده. پس گره مبدا مسیریابی اش به گره مقصد انجام میشود.

اگر روتر نمیتواند پاسخگوی سرویس های خواسته شده باشد -> مانند یک مسیریابی ناموفق عمل کند تا اصلا مسیریابی بین این دو گره انجام نشود.

تمام این عملیات در این تابع اتفاق می افتد. handleMessage
این تابع بسته ها را کالبدشکافی میکند و درنهایت با تشخیص اینکه بسته کدام مرحله AODV را شامل میشود ، یکی از توابع زیر را برای تصمیم گیری نهایی صدا میزند:

```
switch (ctrlPacket->getPacketType()) {
    case RREQ:
        handleRREQ(dynamicPtrCast<Rreq>(ctrlPacket->dupShared()), sourceAddr, arrivalPacketTTL);
        break;

    case RREP:
        handleRREP(dynamicPtrCast<Rrep>(ctrlPacket->dupShared()), sourceAddr);
        break;

    case RERR:
        handleRERR(dynamicPtrCast<const Rerr>(ctrlPacket), sourceAddr);
        break;

    case RREPACK:
        handleRREPACK(dynamicPtrCast<const RrepAck>(ctrlPacket), sourceAddr);
        break;

    default:
        throw cRuntimeError("AODV Control Packet arrived with undefined packet type: %d", ctrlPacket->getPacketType());
}
delete udpPacket;
```

پس باید تغییرات لازم برای تصمیم گیری جدید را در این توابع پیاده سازی کنیم.
و در همه آنها روند کار به همان صورتی که گفته شد میباشد.
برای مثال در کلاس RREQ:

```
void Aodv::handleRREQ(const Ptr<Rreq>& rreq, const L3Address& sourceAddr, unsigned int timeToLive)
{
    EV_INFO << "AODV Route Request arrived with source addr: " << sourceAddr << " originator addr: " << rreq->getOriginatorAddr()
            << " destination addr: " << rreq->getDestAddr() << endl;


    if (rreq->getZZRequestedService_req() != this->zzNeededService) {
        EV_INFO << "The sender requested service: " << rreq->getZZRequestedService_req() << " is not available in this node." << endl;
        return;
    }
    // A node ignores all RREQs received from any node in its blacklist set.
```

همانطور که مشاهده میکنید ، سرویس های موجود در روتر با سرویس خواسته شده در بسته مقایسه میشود و اگر روتر قادر به سرویس دهی نباشد. بسته رها میشود. (مانند در نظر گرفتن مبدا بسته به در لیست سیاه که باید مسیریابی انجام نشود ، که در کدهای پایینتر نوشته شده بود ، من هم همان روش را برای رها کردن بسته ها در نظر گرفتم)

و در نهایت میتوانیم خیلی راحت سرویس ها را در فایل اصلی ini. اینگونه تعریف کنیم:

```
*.host[0].aodv.zzAvailableServices = "1,2"
*.host[1].aodv.zzAvailableServices = "1"
*.host[1].aodv.zzRequestedServices = "3"
```

مثلا میزبان ۰ سرویس های ۱و۲ را ارایه میدهد و میزبان یک سرویس ۱ را ارایه میدهد و سرویس ۳ را میخواهد.

- با سپاس از توجه شما