Lab 1 report

109550087 單宇晟

```
// TODO 1: create one ap node and one wifi node
ap.Create(1);
stas.Create(1);
```

在 TODO 1 的部分，這裡就很簡單的將兩個 node create 出來。

```
// TODO 2: change propagation model here
YansWifiPhyHelper wifiPhy;
YansWifiChannelHelper wifiChannel;

wifiChannel.AddPropagationLoss("ns3::LogDistancePropagationLossModel",
                               "Exponent", DoubleValue(PLE),
                               "ReferenceDistance", DoubleValue(1.0),
                               "ReferenceLoss", DoubleValue(46.677));
                                  // spec
wifiChannel.AddPropagationLoss("ns3::NakagamiPropagationLossModel",
                                 "m0", DoubleValue(1.0),
                                 "m1", DoubleValue(1.0),
                                 "m2", DoubleValue(1.0));
                                | // Rayleigh
wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
// TODO 2
```

接著是 TODO 2，可以看到這裡有兩個 model，一個是 log distance path loss model，這部分我是參考 spec 上寫的，所以我在這個 model 裡使用的參數都跟 spec 上的一樣(PLE=3)；另一個則是 Rayleigh fading model，最後在統一把 propagation delay 設成一個 constant delay 的 model。

```
// TODO 3: Initialize AP & wifi node position and specify movement
MobilityHelper mobility;

Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator>();
positionAlloc->Add(Vector(0.0, 0.0, 0.0));

mobility.SetPositionAllocator(positionAlloc);
mobility.SetMobilityModel("ns3::ConstantVelocityMobilityModel");

mobility.Install(stas);

Ptr<ConstantVelocityMobilityModel> cvmm = stas.Get(0)->GetObject<ConstantVelocityMobilityModel>();
cvmm->SetVelocity(Vector(20, 0, 0));

mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(ap);
// TODO 3
```

再來是 TODO 3，這裡我也參考了 spec 上的 code，不過我把 position allocator 改成了 list position allocator、mobility 的 model 從 random 改成了 constant velocity，並把 velocity 設成(20, 0, 0)，因為這樣在之後的作圖環節比較方便，也比較好觀察輸出的數值。要注意的是，上面這些 mobility 是屬於 stas 這個 mode 的，ap 這個 node 我設定成 constant position，也就是固定位置

```
// TODO 4: hook SNR and throughput related trace source
Config::Connect("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Phy/MonitorSnifferRx", MakeCallback(&Monitor));
// TODO 4
```

```
void Position(std::string context, Ptr<const MobilityModel> model)
{
    // std::cout << context << std::endl;
    // Vector position = model->GetPosition();
    // std::cout << " x: " << position.x
    //           << " y: " << position.y
    //           << " z: " << position.z << std::endl;
}

void Monitor(std::string context, Ptr<const Packet> packet, uint16_t channelFreqMhz, WifiTxVector txVector, MpduInfo aMpdu, SignalNoiseDbm signalNoise, uint16_t staId)
{
    Ptr<MobilityModel> apMobility = ap.Get(0)->GetObject<MobilityModel>();
    Vector position_ap = apMobility->GetPosition();
    // std::cout << " x: " << position_ap.x
    //           << " y: " << position_ap.y
    //           << " z: " << position_ap.z << std::endl;

    Ptr<MobilityModel> stasMobility = stas.Get(0)->GetObject<MobilityModel>();
    Vector position_sta = stasMobility->GetPosition();
    // std::cout << " x: " << position_sta.x
    //           << " y: " << position_sta.y
    //           << " z: " << position_sta.z << std::endl;

    double distance = CalculateDistance(position_sta, position_ap);
    std::cout << "distance:" << distance << std::endl;
    std::cout << "signal:" << signalNoise.signal << std::endl;
}
```

最後來到了 TODO 4，我首先把 position 裡的 code 都 comment 起來，並在 monitor()這個 function 裡先找到 stas 和 ap 的位置，並計算兩個 node 之間的 distance，會這樣做的原因是如果在 position 裡輸出位置，同一個位置會輸出很多 signal strength，我覺得這樣作圖會比較麻煩，所以才改成了現在這個寫法。程式的輸出會是 distance:xxx signal:yyy 的形式。而如果要做 figure 2，就在 Monitor()裡加上下面這行

```
sum += packet->GetSize();
```

最後再用 sum 去計算不同 PLE 下的 throughput 就完成了。

Figure 1:

在 figure 1 中，藍線代表 log distance path loss model，紅色則是加上 Rayleigh fading model 的結果，可以看到兩者的都會有隨著距離增加，趨勢其實十分接近，只是紅線的 signal strength 會有一些幅度的震盪，因為 Rayleigh model 代表不存在 LOS，所以才會有這種情況。

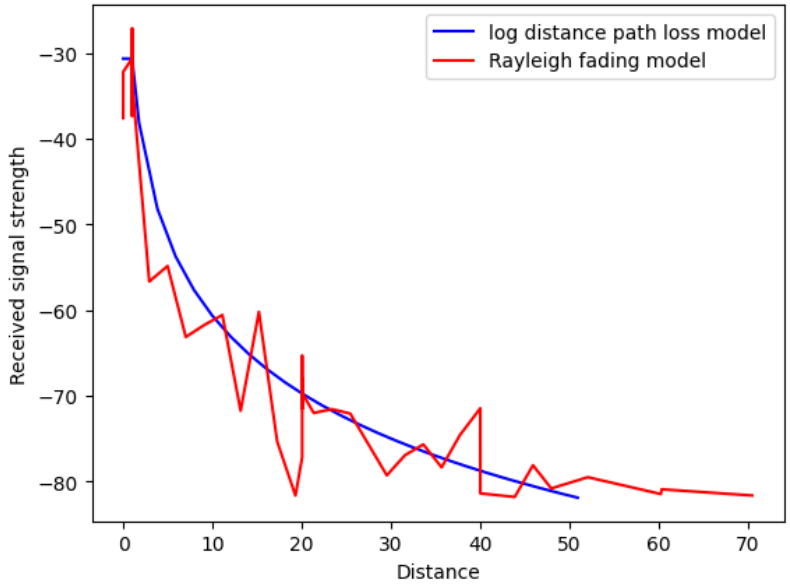

Figure 2:

在 figure 2 中，可以看到在 PLE 超過一定的 threshold 前，throughput 都會是一個固定的數值，而在超過 threshold 之後，就會呈現負相關。