Wireless Network Modeling and Simulation

UGE: M2 SIA - MSQue Project Report

Luca Uckermann

2025-02-02

This project focuses on the modeling and simulation of wireless networks using the INET framework within the OMNeT++ simulation environment. Following the wireless tutorial provided by the INET framework, the project explores key concepts such as transmission rates, protocol behavior, event logs and performance metrics. Configurations such as two hosts communicating wirelessly, introducing interference and using CSMA with and without acknowledgment mechanisms are analyzed.

Table of contents

This project is based on the INET wireless tutorial (INET Framework n.d.b). To install the INET framework (INET Framework n.d.a) the following steps were taken:

```
pip install opp_env
opp_env install --init\
  -w inet-workspace inet-4.5.4 omnetpp-6.1.0
```

The INET framework was installed using the `opp_env` tool (OMNeT++ n.d.b), which simplifies the installation process by managing dependencies and environment variables. The INET framework version `4.5.4` and OMNeT++ version `6.1.0` were installed in the `inet-workspace/` directory.

To run the OMNeT++ IDE (OMNeT++ n.d.a), the following commands were executed:

```
cd inet-workspace
opp_env shell
omnetpp
```

The simulation was run using the IDE, which provides a graphical interface for configuring and running simulations. The `wireless` tutorial was selected and the simulation started (`tutorials/wireless/omnetpp.ini`).

Each of the following steps corresponds to a specific configuration in the tutorial. The results and analysis for each configuration are presented in detail.

1 Two hosts communicating wirelessly

1.1 Simulate and measure the transmission rate.

After each packet transmitted, the following output is generated:

```
UDPData-1 (8.504 ms 1063 B)
```

The output shows that the packet `UDPData-1` was transmitted after `8.504 ms` with a size of `1063 bytes`.

1.2 Is the inter-arrival time for the packets memoryless? Explain your answer.

One line of the `omnetpp.ini` configuration file is as follows:

```
*.hostA.app[0].sendInterval =\
  exponential(12ms)
```

From the configuration it is clear that the inter-arrival times are memoryless because the `sendInterval` is explicitly defined as `exponential(12ms)`. This is a key property of exponential distributions, which are memoryless.

Another approach is to analyze the output of the simulation:

```
Event #17
Reception ended: successfully
  (8.504 ms 1063 B)

Event #6122
Reception ended: successfully
  (8.504 ms 1063 B)
```

Even after many events, the packet arrival time is still constant at `8.504 ms`, which is a characteristic of memoryless processes.

1.3 Compute the total number of packets transmitted during the simulation time. Explain why the transmission rate was around 660 kbps.

The simulation stops after `20` seconds with the following output:

```
Simulation time limit reached
  -- at t=20s, event #20746
Transmission count = 1596

WirelessA.hostB.app[0]:
  received 1596 packets
```

A total of `1596` packets were transmitted during the simulation time.

```
totalLengthField = 1028
```

The total length of a packet is `1028` bytes.

The transmission rate is calculated as follows:

$$\begin{aligned}\text{Transmission Rate} &= \frac{\text{totalLengthField} \times \text{Transmission count}}{\text{Simulation time}} \\ &= \frac{1028\text{bytes} \times 1596}{20\text{s}} = 82034.4\text{bytes/s} \\ &= 82034.4\text{bytes/s} \times 8\text{bits/byte} \\ &= 656275.2\text{bits/s} \\ &= 656.2752\text{kbps}\end{aligned} \tag{1}$$

Equation Equation 1 shows that the transmission rate is about `656 kbps`, which is close to the expected value of `660 kbps`.

1.4 Reproduce the simulation and confirm that the results remain consistent.

All simulation runs give the same result:

```
Simulation time limit reached
  -- at t=20s, event #20746

Transmission count = 1596
Signal send count = 1596
Reception computation count = 1596

WirelessA.hostB.app[0]:
  received 1596 packets
```

The `transmission count` and the packets received by `hostB` are always `1596` packets, confirming the consistency of the results. The last event number `#20746` is also the same for all runs, indicating that the simulation is deterministic.

1.5 Modify the random number generator seed in `omnetpp.ini`. How does changing the seed impact the simulation results?

To modify the seed of the random number generator, add the following line to `omnetpp.ini`:

```
seed-set = 1337
```

After changing the seed, the simulation results are as follows:

```
Simulation time limit reached --
  at t=20s, event #21022

Transmission count = 1617
Signal send count = 1617
Reception computation count = 1617

WirelessA.hostB.app[0]:
  received 1617 packets
```

The `transmission count` as well as the packets received by `hostB` are now `1617` packets, which is different from the previous result. This shows that changing the seed affects the simulation results. In this case, the number of transmitted packets increased by `21`.

From here on, the seed is set to `1337` for all simulations to allow reproducibility.

1.6 List all parameters of the MAC protocol used in `AckingWirelessInterface` with references to the source code.

The parameters in the `omnetpp.ini` file are as follows:

```
*.host*.wlan[0].\
  typename = "AckingWirelessInterface"
*.host*.wlan[0].\
  mac.useAck = false
*.host*.wlan[0].\
  mac.fullDuplex = false
*.host*.wlan[0].\
  radio.transmitter.communicationRange =\
    500m
*.host*.wlan[0].\
  radio.receiver.ignoreInterference = true
*.host*.wlan[0].\
  mac.headerLength = 23B
```

and in the file `AckingWirelessInterface.ned` (l. 30-36):

```
parameters:
  string interfaceTableModule;
  string energySourceModule =\
    default("");
  double bitrate @unit(bps);
  *.interfaceTableModule =\
    default(
      absPath(this.interfaceTableModule)
    );
  *.energySourceModule =\
    default(
      absPath(this.energySourceModule)
    );
  **.bitrate = this.bitrate;
```

1.7 Explain the `throughput:vector` statistic in the analysis file by citing its definition and implementation in the source code.

Information can be found in `/applications/udpapp/UdpBasicApp.ned` (l. 55-57):

```
@signal[packetReceived]
  (type=inet::Packet);
@statistic[packetReceived]
  (title="packets received";
    source=packetReceived;
    record=count,
      "sum(packetBytes)",
      "vector(packetBytes)"
    ;
    interpolationmode=none
  );
@statistic[throughput]
  (title="throughput";
    unit=bps;
    source="throughput(packetReceived)";
    record=vector
  );
```

The `packetReceived` signal, of type `inet::Packet`, is emitted whenever a packet is received by the module and serves as the basis for related metrics. The `packetReceived` statistic uses this signal to track three key metrics: the total number of packets received (`count`), the cumulative size of packets received (`sum(packetBytes)`) and the individual sizes of packets as a time-series vector (`vector(packetBytes)`), with no interpolation applied to the discrete events. The `throughput` statistic, also derived from the `packetReceived` signal, calculates and records the throughput (rate of data received) in bits per second (`bps`) as a time series vector. This setup allows detailed tracking and analysis of packet reception and throughput performance over time during the simulation.

The analysis file (`Wireless.vec`) contains the following lines related to the `throughput:vector` statistic:

```
vector 82 WirelessA.hostB.app[0]
  throughput:vector ETV
attr source throughput(packetReceived)
attr title throughput
```

These lines show that the `throughput:vector` statistic is associated with the `packetReceived` signal from the `hostB` application module, providing insight into the throughput performance of the network.

2 Setting up some animations

2.1 Visualize the radio transmission range.

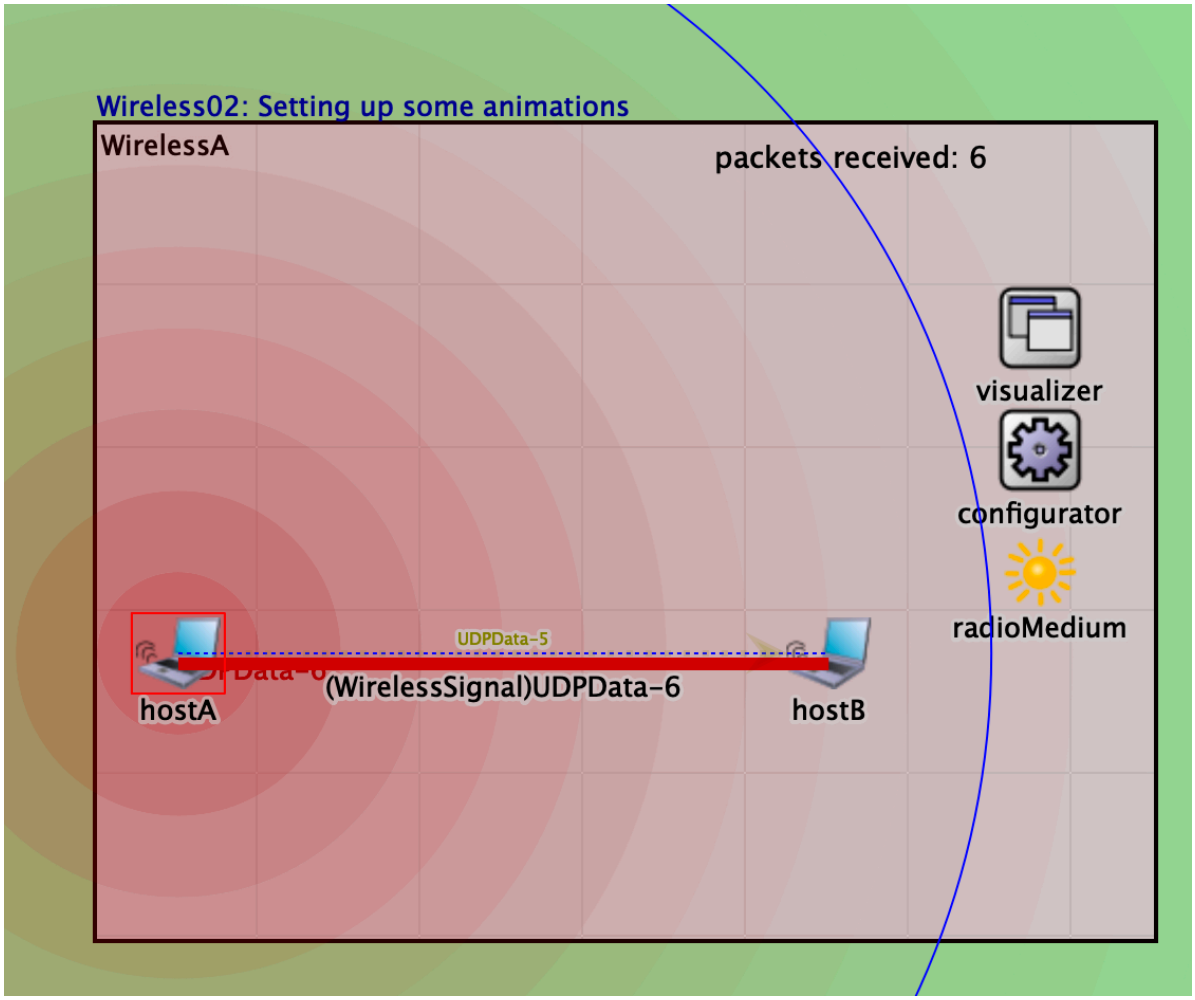

Figure 1: Transmission Range

Figure 1 visualizes the radio transmission range, showing the communication radius around `hostA`, represented by the blue circle. The range is set to `500m` as specified in the `omnetpp.ini` configuration file:

```
*.host*.wlan[0].radio.transmitter.\
  communicationRange = 500m
```

2.2 Change the communication range so that host B is no longer reachable from host A. Visualize the new range.

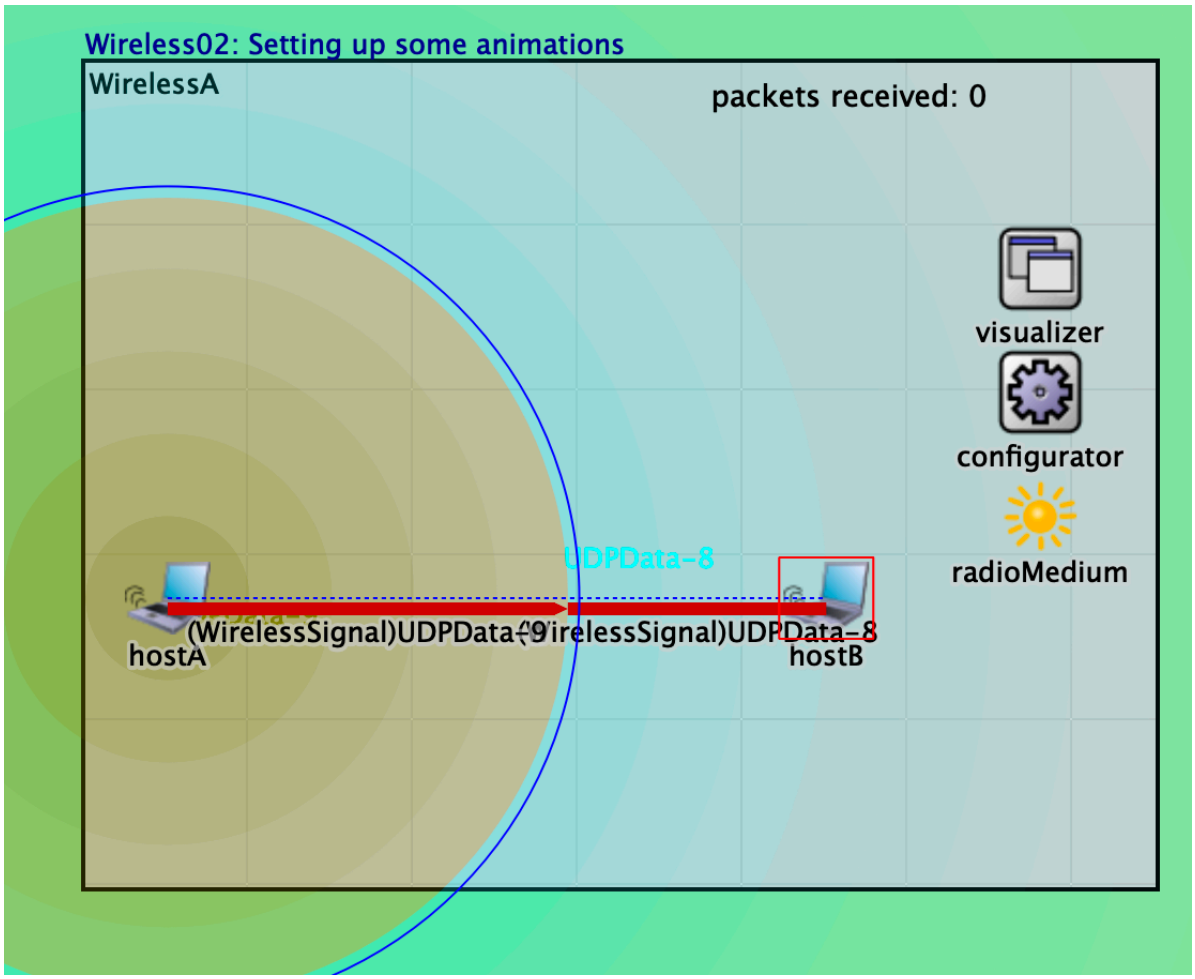

Figure 2: Reduced Transmission Range

Figure 2 illustrates the reduced communication range, where the blue circle around `hostA` no longer reaches `hostB`. This change was achieved by modifying the communication range parameter in the `omnetpp.ini` file:

```
*.host*.wlan[0].radio.transmitter.\
  communicationRange = 250m
```

2.3 Provide simulation results for the transmission rate and explain them.

To provide results, the range is set back to `500m` in the `omnetpp.ini` file. The simulation is run and the following output is generated:

```
Simulation time limit reached
  -- at t=20s, event #21022

Radio signal
  arrival computation count = 1617
Transmission count = 1617
Signal send count = 1617
Reception computation count = 1617

WirelessA.hostB.app[0]:
  received 1617 packets
```

The transmission count is `1617` packets, which is the same as the previous result (with `seed-set = 1337`). The animation only visualizes the network topology and does not interfere with the simulation logic.

Leaving the communication range at `250m` would result in `hostB` being unreachable from `hostA`:

```
WirelessA.hostB.app[0]: received 0 packets
```

The output confirms that no packets were received from `hostB`, indicating that the reduced communication range successfully prevented communication between the two hosts.

3 Adding more nodes and decreasing the communication range

3.1 Identify the type of wireless interfaces for the new hosts. Explain your answer with references to the source code.

The three new hosts (`hostR1`, `hostR2` and `hostR3`) are defined in the `WirelessB.ned` network configuration:

```
network WirelessB extends WirelessA
{
  submodules:
    hostR1: <default("WirelessHost")>\
      like INetworkNode {
        @display("p=250,300");
    }
    hostR2: <default("WirelessHost")>\
      like INetworkNode {
        @display("p=150,450");
    }
    hostR3: <default("WirelessHost")>\
      like INetworkNode {
        @display("p=350,450");
    }
}
```

The `WirelessHost` is defined in `/node/inet/WirelessHost.ned`:

```
module WirelessHost extends StandardHost
{
    parameters:
        numWlanInterfaces = default(1);
        @display("i=device/wifilaptop");
}
```

"Models a host with (default) one wireless (802.11) card in infrastructure mode" (l. 11).

The wireless interfaces for these hosts are specified in the `omnetpp.ini` file:

```
*.host*.wlan[0].typename =\
  "AckingWirelessInterface"
```

Again, the `AckingWirelessInterface` type is used for the wireless interfaces of all hosts in the network.

4 Setting up static routing

4.1 Use the runtime GUI to capture a screenshot showing the IP addresses of all hosts in the network.

Set the following option in the `omnetpp.ini` file to enable this debugging information:

```
INFO: hostA [...]
  {inet_addr:10.0.0.1/24[...]}
INFO: hostB [...]
  {inet_addr:10.0.0.2/24[...]}
INFO: hostR1 [...]
  {inet_addr:10.0.0.3/24[...]}
INFO: hostR2 [...]
  {inet_addr:10.0.0.4/24[...]}
INFO: hostR3 [...]
  {inet_addr:10.0.0.5/24[...]}
```

Alternatively, the routing table can be viewed in the runtime GUI by selecting the `hostA` module and clicking on `Routing Table` in the `ipv4` section:

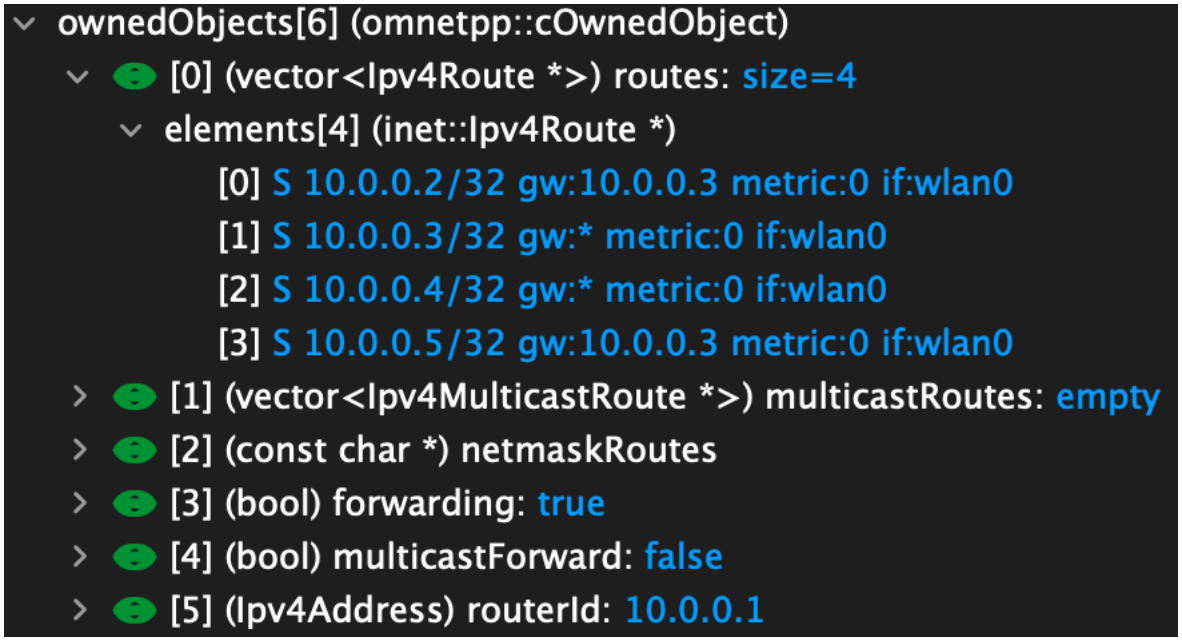

Figure 3: Routing Table Host A

Figure 3 displays the routing table for `hostA`, showing the IP addresses of all hosts in the network.

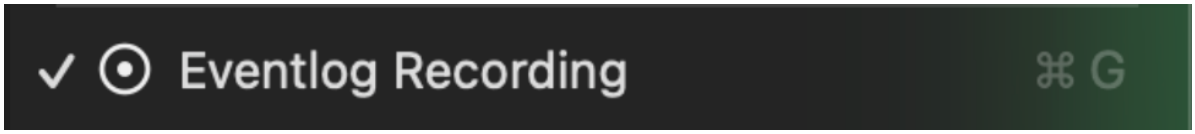

Figure 4: Enable Event Logging

5 Taking interference into account

5.1 Enable event log recording in the runtime GUI.

Figure 4 shows that the "Eventlog Recording" option is enabled in the runtime GUI.

5.2 Open the `.elog` file with a text editor and explain its contents.

An excerpt from the `.elog` file is as follows:

```
CM id 243061 tid 243061 eid 243061
  etid 243061 c
    omnetpp::cMessage n UDPData-0 pe -1
BS id 243061 tid 243061 eid 243061
  etid 243061 c
    inet::Packet n
      UDPData-0 l 8000 m 54 pe 4

SH sm 54 sg 3
CMB sm 54 tm 55 m arrived

ES id 242948 tid 242948 eid 242948
  etid 242948 c
    inet::ClockEvent n
      sendTimer k 2 sm 54 st
        0.007276585767 am 54 at
          0.026732847636 pe 14

CMB sm 51 tm 25 m packetReceivedFromUpper
CME
CMB sm 51 tm 27 m packetReceivedFromUpper
CME
CMB sm 51 tm 30 m packetReceivedFromUpper

CMB sm 61 tm 4 m transmitPacket
```

```
BS id 242927 tid 242927 eid 242927
  etid 242927 c
    omnetpp::cMessage n
      removeNonInterferingTransmissions
        sm 4 st 0.002868038578
          am 4 at 0.021373372834
            pe 9

CMB sm 182 tm 30 m packetReceivedFromLower
CME

MDC id 74 d
  "t=processed 1 pk (1008 B);
    p=550,300;b=600,5,,,,1"

CMB sm 75 tm 69 m
  findBestMatchingRoute(10.0.0.2)
CME
CMB sm 75 tm 76 m
  resolveL3Address
CME
```

In general, the file captures the detailed sequence of events and interactions between modules during the simulation. Key events include:

1. **Message Creation:** A packet (`UDPData-0`) is created as an instance of `inet::Packet` with a size of `8000` bits and processed by submodules (`CM` and `BS` events).
2. **Signal Handling and Arrival:** Signals are processed by submodules (`SH` events) and packets arrive at target submodules (`CMB` events).
3. **Event Scheduling:** Timers and network events (`ES`) are scheduled for execution, such as the `sendTimer` for managing transmissions.
4. **Packet Processing and Transmission:** Packets are received from upper layers, processed and transmitted at various stages (`CMB` with `packetReceivedFromUpper` and `transmitPacket`).
5. **Address Resolution:** Submodules resolve layer 3 addresses and identify routes (`CMB findBestMatchingRoute`).
6. **Visualization Updates:** Visualization logs (`MDC`) show processed packets and their visual representation on the simulation canvas.

This information provides a comprehensive view of packet flow and module interactions that can be used to analyze system behavior and performance.

5.3 Analyze the event log using the sequence chart and event log table tools. Filter events for `hostA+udp`, `hostB+udp` and `hostR1+ip`.

Name	ID	Type	Path
hostA	5	inet.node.inet.WirelessHost	WirelessB.hostA
app[0]	54	inet.applications.udpapp.UdpBasicApp	WirelessB.hostA.app[0]
at	55	inet.common.MessageDispatcher	WirelessB.hostA.at
bl	44	inet.common.MessageDispatcher	WirelessB.hostA.bl
cb	43	inet.common.MessageDispatcher	WirelessB.hostA.cb
interfaceTable	42	inet.networklayer.common.InterfaceTable	WirelessB.hostA.interfaceTable
ipv4	49	inet.networklayer.ipv4.Ipv4NetworkLayer	WirelessB.hostA.ipv4
li	45	inet.common.MessageDispatcher	WirelessB.hostA.li
lo[0]	46	inet.linklayer.loopback.LoopbackInterface	WirelessB.hostA.lo[0]
mobility	41	inet.mobility.static.StationaryMobility	WirelessB.hostA.mobility
nl	50	inet.common.MessageDispatcher	WirelessB.hostA.nl
tcp	52	inet.transportlayer.tcp.Tcp	WirelessB.hostA.tcp
tn	53	inet.common.MessageDispatcher	WirelessB.hostA.tn
udp	51	inet.transportlayer.udp.Udp	WirelessB.hostA.udp
wlan[0]	47	inet.linklayer.acking.AckingWirelessInterface	WirelessB.hostA.wlan[0]
hostB	6	inet.node.inet.WirelessHost	WirelessB.hostB
hostR1	7	inet.node.inet.WirelessHost	WirelessB.hostR1
at	136	inet.common.MessageDispatcher	WirelessB.hostR1.at
bl	126	inet.common.MessageDispatcher	WirelessB.hostR1.bl
cb	125	inet.common.MessageDispatcher	WirelessB.hostR1.cb
interfaceTable	124	inet.networklayer.common.InterfaceTable	WirelessB.hostR1.interfaceTable
ipv4	131	inet.networklayer.ipv4.Ipv4NetworkLayer	WirelessB.hostR1.ipv4
arp	157	inet.networklayer.arp.ipv4.GlobalArp	WirelessB.hostR1.ipv4.arp
configurator	149	inet.networklayer.configurator.ipv4.Ipv4NodeConfigurator	WirelessB.hostR1.ipv4.configurator
icmp	154	inet.networklayer.ipv4.Icmp	WirelessB.hostR1.ipv4.icmp
igmp	153	inet.networklayer.ipv4.Igmpv2	WirelessB.hostR1.ipv4.igmp
ip	156	inet.networklayer.ipv4.Ipv4	WirelessB.hostR1.ipv4.ip

Figure 5: Filter Event Log

Figure 5 shows the event log filtering options for `hostA+udp`, `hostB+udp` and `hostR1+ip` in the runtime GUI.

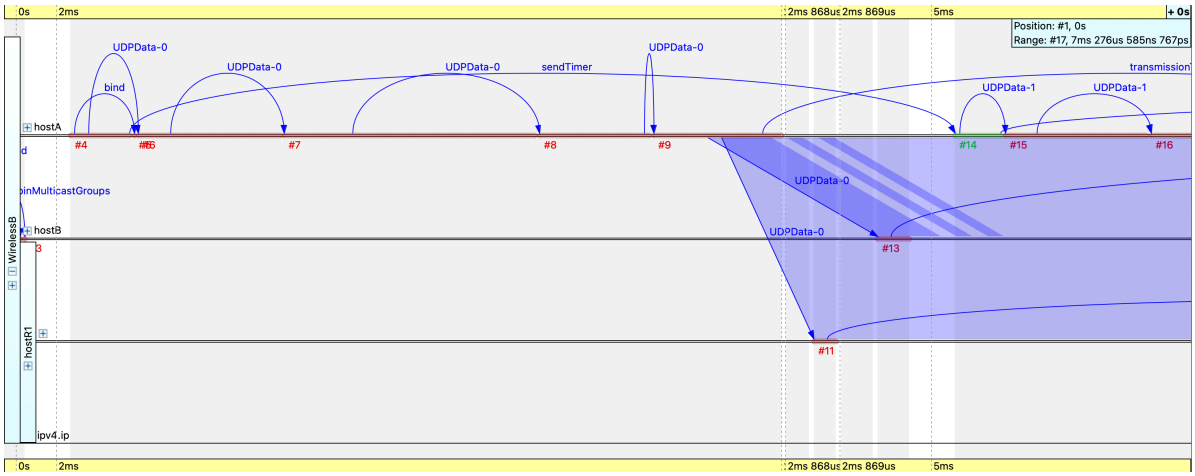

Figure 6: Event Log Start

5.4 Take screenshots and comment on both a complete and an incomplete transmission sequence chart.

Figure 6 shows the first events of the simulation. At `Event #4` it starts with the `bind` operation for `UDPData-0` in `hostA`. This step represents the initialization of the UDP communication, including the creation of the packet and its preparation for transmission. At this point, the `UDPData-0` message is scheduled for delivery, marking the start of its journey from `hostA`.

The illustration is further simplified by using the *Network Interface* option in the *Preset Configuration*:

As shown in Figure 7, the sequence chart shows the status of three packets (`UDPData-813`, `UDPData-814` and `UDPData-815`):

- **`UDPData-813`:** Successfully reaches `hostB` after being forwarded by `hostR1`. The arcs indicate a complete flow from `hostA` to `hostB` via `hostR1`.
- **`UDPData-814`:** Is transmitted from `hostA` to `hostR1` but does not reach `hostB`. The absence of arcs from `hostR1` to `hostB` indicates that the packet was either dropped or lost.
- **`UDPData-815`:** Also fails to reach `hostB` after being sent to `hostR1`. Similar to `UDPData-814`, the packet appears to have been dropped or interrupted before reaching its final destination.

This simplified view highlights the occurrence of packet loss or forwarding problems at `hostR1`, indicating the need for further analysis of routing or congestion effects on the network.

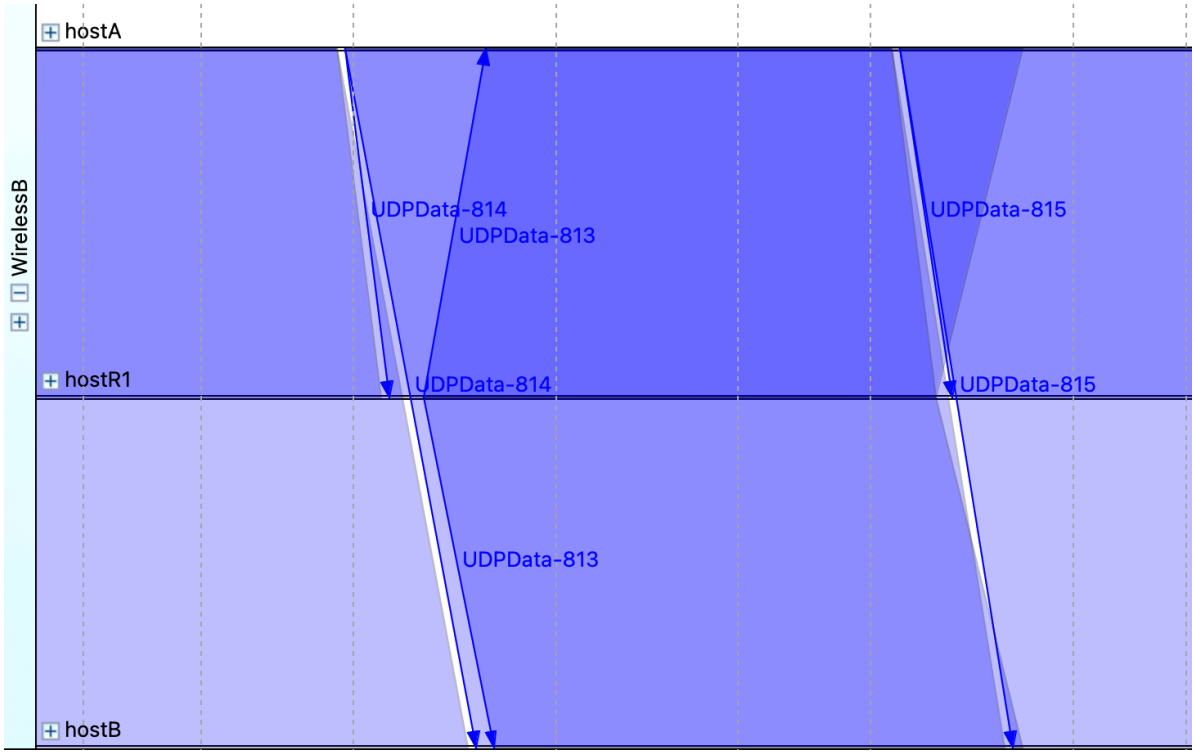

Figure 7: Event Log Simplified

6 Using CSMA to better utilize the medium

6.1 1. Compare the number of packets received by host B in this configuration with the results from *Configuration Step 5* (Section 5). Explain the difference.

Running the simulation with the CSMA configuration produces the following output:

```
WirelessB.hostB.app[0]:
  received 1090 packets
```

In Section 5 it was:

```
WirelessB.hostB.app[0]:
  received 139 packets
```

The number of packets received by `hostB` in this configuration is `1090`, which is significantly higher than the `139` packets received in the previous configuration. This difference can be attributed to the use of CSMA in the current configuration, which allows for better utilization of the medium by coordinating access to the channel among multiple nodes. The CSMA protocol helps reduce collisions and interference, resulting in more successful packet transmissions and higher throughput compared to the previous configuration.

7 Turning on ACKs in CSMA

7.1 Measure the `numRetry` statistic for *Configuration Steps 6* (Section 6) and *7*.

The `numRetry` statistic is measured in the simulation output (`Wireless06.sca`):

```
scalar WirelessB.hostA.wlan[0].mac
  numRetry 0
scalar WirelessB.hostB.wlan[0].mac
  numRetry 0
scalar WirelessB.hostR1.wlan[0].mac
  numRetry 0
scalar WirelessB.hostR2.wlan[0].mac
  numRetry 0
scalar WirelessB.hostR3.wlan[0].mac
  numRetry 0
```

and (`Wireless07.sca`):

```
scalar WirelessB.hostA.wlan[0].mac
  numRetry 67
scalar WirelessB.hostB.wlan[0].mac
  numRetry 0
scalar WirelessB.hostR1.wlan[0].mac
  numRetry 67
scalar WirelessB.hostR2.wlan[0].mac
  numRetry 0
scalar WirelessB.hostR3.wlan[0].mac
  numRetry 0
```

7.2 Cite the source code where the logic for `numRetry` is defined.

The logic for `numRetry` is implemented in `inet/linklayer/csmaca/CsmaCaMac.cc`:

```
numRetry = 0;
WATCH(numRetry);

void CsmaCaMac::finish()
{
    recordScalar("numRetry", numRetry);
    ...
}

void CsmaCaMac::retryCurrentTransmission()
{
    ASSERT(retryCounter < retryLimit);
    retryCounter++;
    numRetry++;
    generateBackoffPeriod();
}
```

At the start it is initialized with `0` and tracked using `WATCH`. In the `finish()` method, `numRetry` is recorded as a scalar statistic using `recordScalar()`. Each time a packet is not successfully transmitted and a retry is initiated, the `retryCurrentTransmission()` method increments `numRetry`.

7.3 Explain the differences in this metric between the two configurations.

The `numRetry` metric for `Configuration Step 6` is `0` for all hosts because of the lack of ACKs. In `Configuration Step 7`, the `numRetry` statistic is `67` for `hostA` and `hostR1`, indicating the number of retransmissions due to failed packet delivery attempts. The increase in `numRetry` when ACKs are enabled is due to the acknowledgment mechanism requiring retries when packets are lost or ACKs are not received. While this increases the reliability of the network, it also results in higher retransmission counts, especially in scenarios with higher traffic or interference.

8 Configuring node movements

8.1 Plot the `throughput vector` at `hostB` and identify the time when the transmission stops.

The following plot is generated using the *Line Chart with Matplotlib* tool in the runtime GUI:

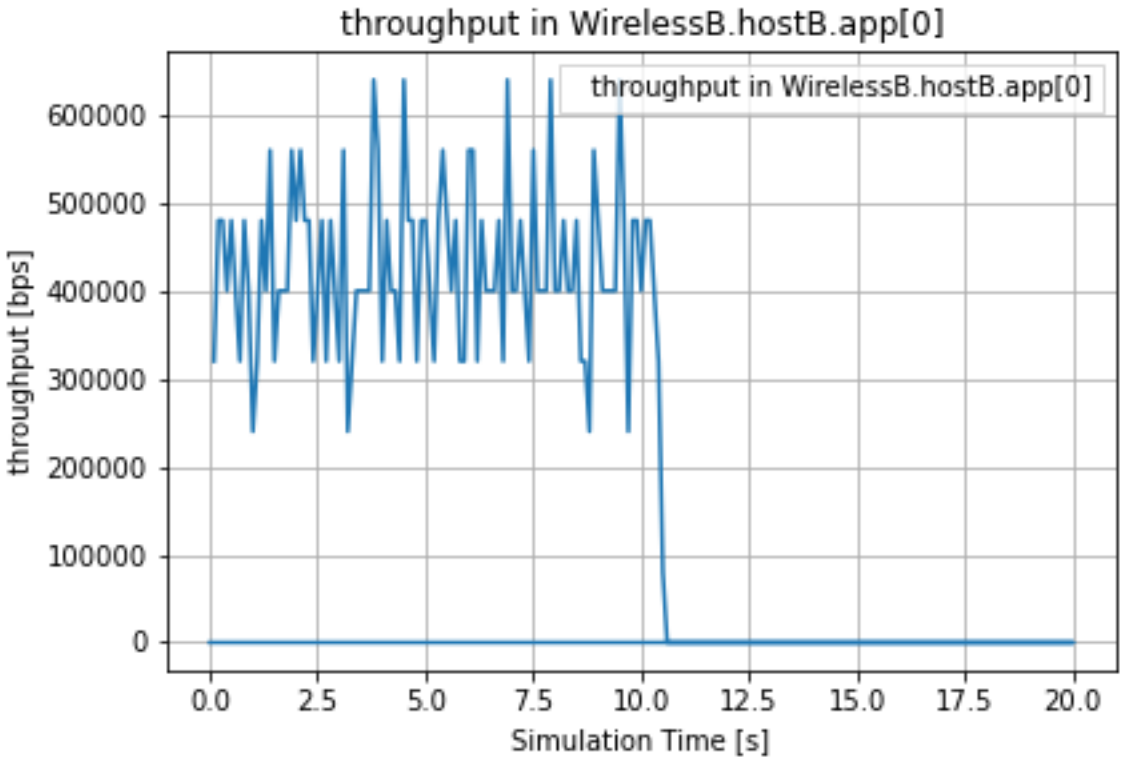

Figure 8: Throughput Vector Host B

Figure 8 shows the `throughput vector` at `hostB` over time. Transmission stops at about `t=10.5s`, as indicated by the sharp drop in throughput to `0 bps`. This corresponds to the moment in the simulation when `hostR1` leaves the communication range of `hostA`, leading to the termination of packet transmission.

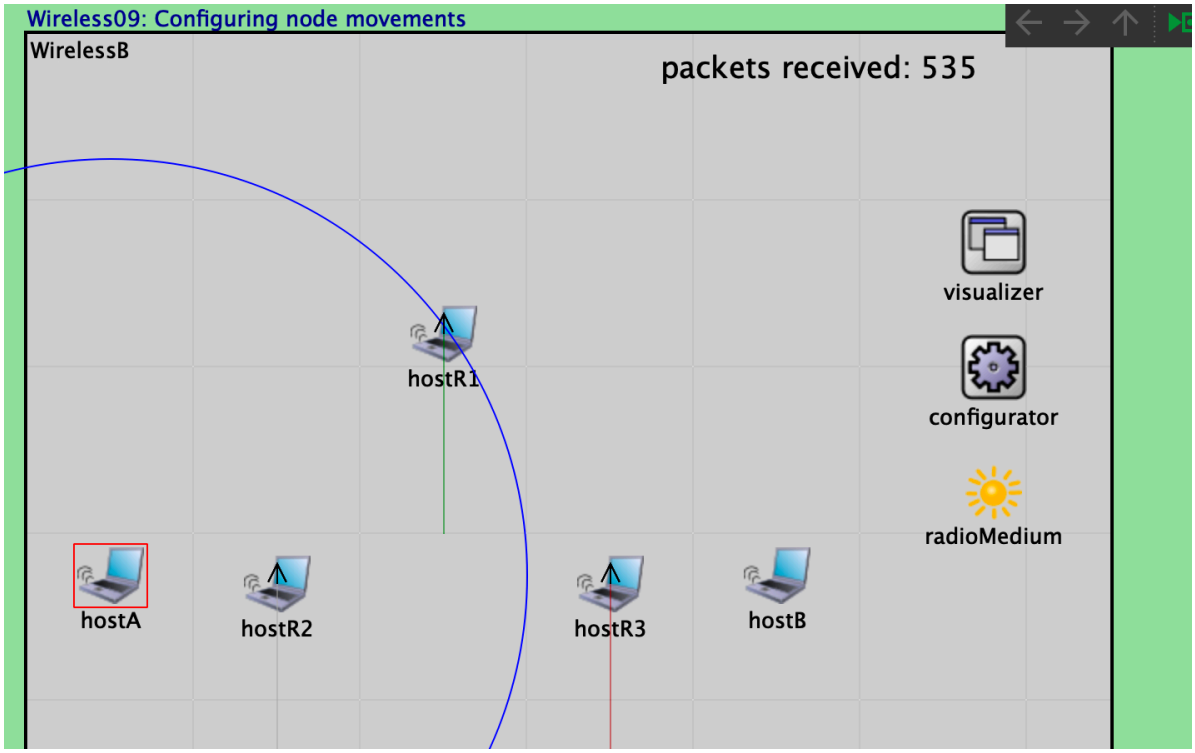

Figure 9: Simulation at t=10

Figure 9 shows the simulation at `t=10s`, where the nodes have moved to new positions. Here, `hostR1` is barely within the communication range of `hostA`, leading to the resulting reduction in throughput and eventual termination of transmissions.

Figure 10 shows the simulation at `t=12s`, where `hostR1` has moved further away from `hostA`, resulting in complete loss of communication and termination of packet transmission. A total of `557` packets were received by `hostB` before the transmission stopped.

9 Conclusion

The wireless network simulations performed in this project demonstrate the effectiveness of OMNeT++ and the INET framework in modeling complex network scenarios. Key findings include the importance of proper protocol configuration, such as enabling acknowledgment mechanisms in CSMA, to ensure reliability despite the increased retransmission overhead.

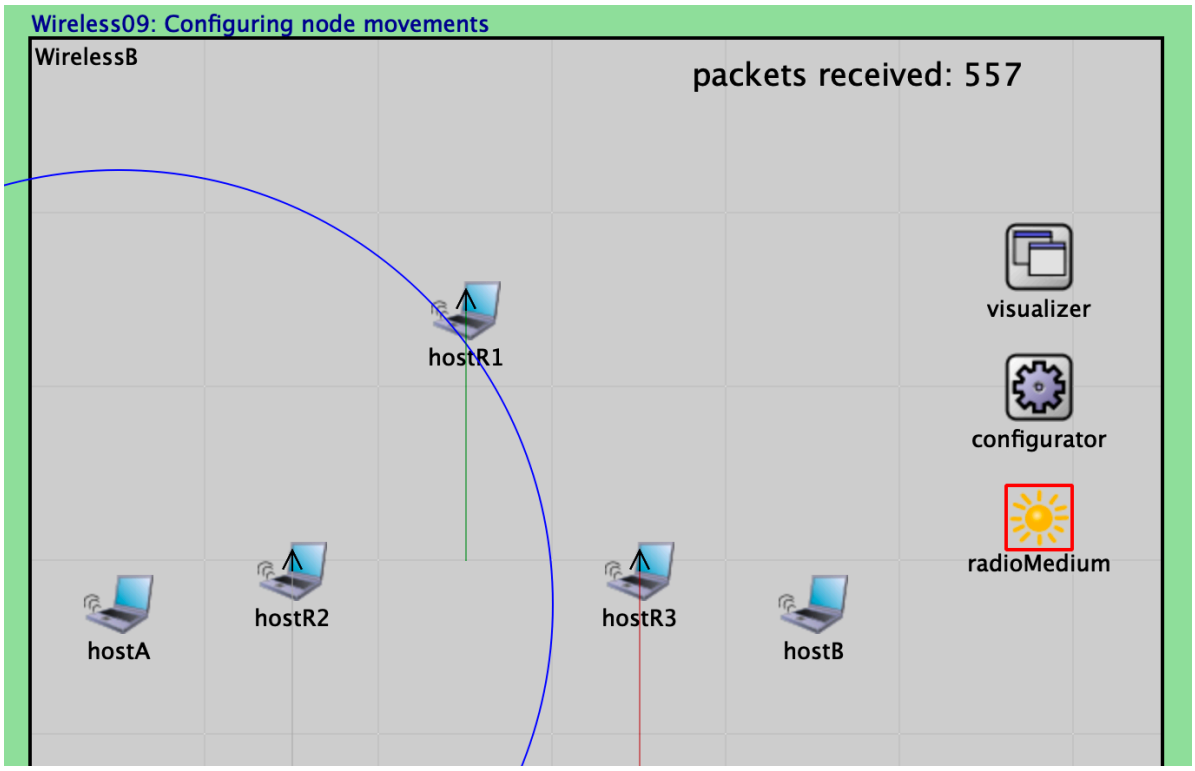

Figure 10: Simulation at t=12

Throughput measurements and event log analysis revealed the impact of communication range, interference and mobility on network performance. In addition, static and dynamic routing configurations were successfully implemented and visualized, providing a clear understanding of node interactions. Overall, the project highlights the versatility and utility of simulation tools for studying wireless network behavior and optimizing protocol design.

10 References

INET Framework. n.d.a. "INET Framework." Accessed January 14, 2025. https://inet.omnetpp.org.

———. n.d.b. "INET Framework - Wireless Tutorial." Accessed January 14, 2025. https://inet.omnetpp.org/docs/tutorials/wireless/doc/index.html.

OMNeT++. n.d.a. "OMNeT++." Accessed January 14, 2025. https://omnetpp.org.

———. n.d.b. "OMNeT++ Environment." Accessed January 14, 2025. https://omnetpp.org/opp_env.