ns-3 Network Simulator

Complete Setup and Basic Simulation Guide

A Comprehensive Introduction to MANET Simulation

Network Simulation Laboratory
Department of Computer Science and Engineering

February 5, 2026

Contents

1 Introduction

1.1 What is ns-3?

ns-3 (Network Simulator 3) is a discrete-event network simulator designed primarily for research and educational purposes. It is a free, open-source software project written in C++ with optional Python bindings. ns-3 is not an extension of the older ns-2 simulator but rather a completely new implementation built from the ground up to address the architectural limitations of its predecessor.

The simulator provides a robust framework for modeling the behavior of computer networks, including wired and wireless networks, Internet protocols, and various network topologies. ns-3 enables researchers and students to simulate complex network scenarios without requiring physical hardware, thereby reducing costs and enabling reproducible experimentation.

1.2 Why Use ns-3?

There are several compelling reasons to use ns-3 for network simulation:

- **Cost-Effective Research:** Setting up physical network testbeds with multiple nodes, routers, and wireless access points is expensive. ns-3 allows researchers to simulate large-scale networks on a single computer.
- **Reproducibility:** Simulations can be precisely controlled and repeated, ensuring consistent results across experiments. This is crucial for scientific research where reproducibility is a key requirement.
- **Flexibility:** ns-3 supports a wide range of network protocols and technologies, including TCP/IP, UDP, WiFi, LTE, and various routing protocols such as AODV, DSDV, and OLSR.
- **Scalability:** Users can simulate networks ranging from a few nodes to thousands of nodes, testing scenarios that would be impractical to implement physically.
- **Active Community:** ns-3 has an active development community, extensive documentation, and numerous tutorials available online.
- **Integration with Real Systems:** Advanced features allow ns-3 to interact with real network stacks and hardware, enabling hybrid simulation-emulation scenarios.

1.3 Common Use Cases

ns-3 is widely used in both academic and industrial research settings. Some common applications include:

- **Mobile Ad-hoc Networks (MANETs):** Studying routing protocols like AODV, DSR, and OLSR in dynamic, infrastructure-less networks where nodes communicate directly with each other.
- **Wireless Sensor Networks (WSNs):** Evaluating energy-efficient protocols and data aggregation techniques for resource-constrained sensor deployments.

- **Vehicular Ad-hoc Networks (VANETs):** Simulating vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication for intelligent transportation systems.
- **Internet of Things (IoT):** Testing low-power communication protocols such as LoRaWAN, Zigbee, and 6LoWPAN.
- **Quality of Service (QoS) Analysis:** Evaluating network performance metrics such as throughput, latency, packet delivery ratio, and jitter under various traffic conditions.
- **Protocol Development:** Designing and testing new routing, transport, or application-layer protocols before implementation in real systems.
- **Network Security:** Simulating attacks such as denial-of-service (DoS), eavesdropping, and evaluating countermeasures.

This guide focuses primarily on MANET simulation using ns-3, providing students with hands-on experience in setting up, configuring, and running basic wireless ad-hoc network experiments.

2 System Requirements

Before proceeding with the installation of ns-3, it is essential to ensure that your system meets the minimum hardware and software requirements. This section outlines the necessary specifications and provides recommendations for optimal performance.

2.1 Operating System Requirements

ns-3 is primarily developed and tested on Unix-like operating systems. The officially supported platforms include:

- **Ubuntu Linux:** Version 20.04 LTS, 22.04 LTS, or later (recommended)
- **Debian:** Version 10 or later
- **Fedora:** Recent versions
- **macOS:** Version 10.14 or later
- **Windows:** Via Windows Subsystem for Linux (WSL 2) running Ubuntu

For beginners, Ubuntu 22.04 LTS is highly recommended due to its stability, extensive community support, and availability of pre-packaged dependencies. This guide primarily focuses on Ubuntu-based installations, including native Ubuntu, WSL, and cloud-based environments.

2.2 Hardware Recommendations

While ns-3 can run on modest hardware configurations, the following specifications are recommended for comfortable development and simulation:

Component	Recommended Specification
Processor	Multi-core CPU (Intel Core i5/i7 or AMD equivalent)
RAM	4 GB minimum, 8 GB or more recommended
Storage	10 GB free disk space for ns-3 and dependencies
Network	Internet connection for downloading packages

Table 1: Recommended hardware specifications for ns-3

For large-scale simulations involving thousands of nodes or long simulation durations, additional RAM and processing power will significantly improve performance. Simulations can be CPU-intensive, so a multi-core processor enables parallel builds and faster compilation times.

2.3 Cloud-Based and Virtual Environments

Many students may not have access to physical Linux machines or may prefer cloud-based development environments. ns-3 works well in the following setups:

- **Windows Subsystem for Linux (WSL 2):** Allows Windows users to run a complete Linux environment without dual-booting. WSL 2 provides better performance than WSL 1 and supports graphical applications with appropriate configuration.
- **GitHub Codespaces:** Cloud-based development environment that provides a preconfigured Ubuntu container accessible via web browser. Ideal for students without local computational resources. Note that GUI applications are not supported, so analysis must be command-line based.
- **Virtual Machines:** VirtualBox or VMware can be used to run Ubuntu as a guest OS on Windows or macOS. Allocate at least 2 CPU cores and 4 GB RAM to the virtual machine.
- **Cloud Computing Platforms:** AWS, Google Cloud, or Azure instances can be used for large-scale simulations. Students should be aware of potential costs associated with extended usage.

2.4 Software Dependencies

ns-3 requires several software packages and libraries to compile and run successfully. These include:

- **C++ Compiler:** GCC version 7 or later, or Clang version 6 or later
- **Python:** Version 3.6 or later (for build system and optional bindings)
- **CMake:** Version 3.10 or later (build system)
- **Git:** For version control and downloading ns-3 from repositories
- **Development Libraries:** libxml2, libsqlite3, and others depending on optional features

Detailed installation instructions for these dependencies are provided in the following section.

3 Installation and Environment Setup

This section provides comprehensive step-by-step instructions for installing ns-3 on various platforms. Each subsection targets a specific environment, ensuring that students can successfully set up ns-3 regardless of their operating system or infrastructure.

3.1 Installation on Native Ubuntu Linux

For users running Ubuntu natively on their machines, the installation process is straightforward. Open a terminal and follow these steps:

3.1.1 Step 1: Update System Packages

Before installing any new software, ensure that your system's package index is up to date:

```
sudo apt update
sudo apt upgrade -y
```

This ensures that you have the latest security patches and package information.

3.1.2 Step 2: Install Essential Build Tools

ns-3 requires a C++ compiler, build tools, and version control software:

```
sudo apt install -y build-essential g++ cmake git
```

- `build-essential`: Meta-package that includes GCC, G++, make, and other essential compilation tools
- `g++`: GNU C++ compiler required for compiling ns-3 source code
- `cmake`: Cross-platform build system used by ns-3
- `git`: Version control system for cloning the ns-3 repository

3.1.3 Step 3: Install Python and Required Libraries

ns-3's build system relies on Python. Install Python 3 and related development packages:

```
sudo apt install -y python3 python3-dev python3-pip
```

Some optional ns-3 features require additional Python packages, which can be installed later if needed.

3.1.4 Step 4: Install Additional Dependencies

Several libraries enhance ns-3 functionality and enable specific features:

```
sudo apt install -y libxml2 libxml2-dev libsqlite3-dev
sudo apt install -y libboost-all-dev libeigen3-dev
sudo apt install -y gir1.2-goocanvas-2.0 python3-gi
sudo apt install -y python3-gi-cairo python3-pygraphviz
```

Purpose of each dependency:

- `libxml2`: XML parsing library used for configuration files
- `libsqlite3`: Lightweight database library for storing simulation results
- `libboost`: C++ libraries that extend standard library functionality
- `libeigen3`: Template library for linear algebra operations
- `gir1.2-goocanvas-2.0`: Canvas widget for GTK+ (used in visualizations)
- `python3-gi`: Python bindings for GObject introspection
- `python3-pygraphviz`: Graph visualization library

Note that some of these packages are optional and primarily support graphical visualizations, which may not be available in headless cloud environments.

3.1.5 Step 5: Download ns-3

Clone the latest stable version of ns-3 from the official GitLab repository:

```
cd ~
git clone https://gitlab.com/nsnam/ns-3-dev.git ns-3
cd ns-3
```

Alternatively, download a specific release version:

```
git clone https://gitlab.com/nsnam/ns-3-dev.git -b ns-3.40 ns-3
```

Replace `ns-3.40` with the desired version number. Using stable releases is recommended for beginners.

3.2 Installation on Windows via WSL 2

Windows Subsystem for Linux (WSL 2) allows Windows users to run a Linux environment directly on Windows without the overhead of a traditional virtual machine.

3.2.1 Step 1: Enable WSL 2

Open PowerShell as Administrator and run:

```
wsl --install
```

This command installs WSL 2 and the default Ubuntu distribution. Restart your computer when prompted.

3.2.2 Step 2: Launch Ubuntu

After restarting, launch Ubuntu from the Start menu. You will be prompted to create a UNIX username and password.

3.2.3 Step 3: Follow Ubuntu Installation Steps

Once inside the Ubuntu terminal, follow the same installation steps outlined in Section 3.1 for native Ubuntu. All commands are identical.

3.2.4 Step 4: Access Windows Files from WSL

Windows drives are mounted under `/mnt/`. For example, your C: drive is accessible at `/mnt/c/`. However, it is recommended to keep ns-3 files within the WSL file system for better performance.

3.3 Installation on GitHub Codespaces

GitHub Codespaces provides cloud-based development environments with Ubuntu containers. This is ideal for students without local resources.

3.3.1 Step 1: Create a Codespace

Navigate to a GitHub repository (or create a new one) and click on **Code** > **Codespaces** > **Create codespace on main**. A new Ubuntu-based container will be provisioned in the cloud.

3.3.2 Step 2: Open Terminal

Once the codespace loads, open the integrated terminal. You now have access to a full Ubuntu environment.

3.3.3 Step 3: Install Dependencies

Run the same dependency installation commands as in Section 3.1. Since you don't have sudo password-less access by default, you may need to enter your codespace password when prompted (or it may already be configured).

```
sudo apt update
sudo apt install -y build-essential g++ cmake git python3 python3-dev
sudo apt install -y libxml2 libxml2-dev libsqlite3-dev
```

3.3.4 Step 4: Clone ns-3

Download ns-3 as described in Section 3.1, Step 5.

3.3.5 Step 5: Note on GUI Applications

Codespaces run in a headless environment, meaning graphical applications will not display. All analysis must be performed using command-line tools, log files, and PCAP traces that can be downloaded for local analysis.

3.4 Common Installation Errors and Fixes

During installation, students may encounter several common issues. This subsection provides solutions to frequent problems.

Error	Solution
`E: Unable to locate package`	Run `sudo apt update` to refresh package lists
`Permission denied`	Ensure you have sudo privileges or are in the correct user group
`Git clone fails`	Check internet connection; verify repository URL; try using HTTPS instead of SSH
`CMake version too old`	Install a newer version manually or use a newer Ubuntu release
`Python not found`	Explicitly install `python3` and create a symbolic link if necessary
`Missing library errors during build`	Install missing development packages using `apt search <library-name>`

Table 2: Common installation errors and their solutions

4 ns-3 Directory Structure

Understanding the directory structure of ns-3 is crucial for navigating the codebase, adding custom simulations, and utilizing built-in examples. This section provides an overview of the most important directories and their purposes.

After cloning or extracting ns-3, the root directory contains numerous folders and files. The key directories are:

4.1 Important Directories

- `scratch/`: This is the primary working directory for users developing custom simulations. Any C++ source file placed in this directory can be compiled and run using the ns-3 build system without modifying configuration files. This is where beginners should place their MANET simulation scripts.

- `examples/`: Contains official example programs demonstrating various ns-3 features. These are organized by module (e.g., `wireless/`, `routing/`, `tutorial/`). Examining these examples is an excellent way to learn ns-3 APIs and best practices.

- `src/`: The source code for all ns-3 modules resides here. Each subdirectory represents a module (e.g., `wifi/`, `internet/`, `mobility/`, `aodv/`). Users typically do not modify these files unless contributing to ns-3 development or implementing custom protocol modules.

- `build/`: Generated during the build process, this directory contains compiled object files, libraries, and executables. Users generally do not interact with this directory directly.

- `doc/`: Contains ns-3 documentation, including the tutorial, manual, and API reference in various formats (HTML, PDF). This is a valuable resource for understanding advanced features.

- `contrib/`: Optional contributed modules not part of the core ns-3 distribution. Users can add third-party modules here.

- `utils/`: Utility scripts for testing, benchmarking, and code formatting.
- `testpy/`: Python scripts used by the ns-3 testing framework.

4.2 The Build System: `ns3`

ns-3 uses a Python-based wrapper script called `ns3` (or `./ns3` when executed from the ns-3 root directory) to manage building, configuring, and running simulations. This script simplifies interaction with the underlying CMake build system.

Common `ns3` commands include:

- `./ns3 configure`: Configure the build system with specified options
- `./ns3 build`: Compile ns-3 and all simulations in the `scratch/` directory
- `./ns3 run <program>`: Compile (if needed) and execute a simulation program
- `./ns3 clean`: Remove build artifacts
- `./ns3 -help`: Display available commands and options

4.3 Configuration Files

Configuration files allow users to customize build options:

- `CMakeLists.txt`: Primary CMake configuration file defining build targets and dependencies
- `ns3rc`: User-specific configuration file for setting default build options

For most beginner use cases, modifying these files is unnecessary.

4.4 Directory Structure Summary

Directory	Purpose
`scratch/`	User simulation scripts
`examples/`	Official example programs
`src/`	ns-3 core modules and protocol implementations
`build/`	Compiled binaries and object files
`doc/`	Documentation and manuals
`contrib/`	Third-party contributed modules
`utils/`	Utility and helper scripts

Table 3: Summary of important ns-3 directories

5 Build and Verification

After installing ns-3 and its dependencies, the next step is to configure, build, and verify the installation. This ensures that the simulator is correctly set up and ready for custom simulations.

5.1 Configuration

Navigate to the ns-3 root directory and configure the build:

```
cd ~/ns-3
./ns3 configure --enable-examples --enable-tests
```

Configuration Options:

- `-enable-examples`: Compiles example programs, which are useful for learning
- `-enable-tests`: Enables the test suite for verifying correctness
- `-build-profile=optimized`: Builds optimized binaries for faster execution (default is debug mode)
- `-disable-python`: Disables Python bindings if not needed

For beginners, the default configuration is sufficient. Advanced users may customize further based on specific requirements.

The configuration step checks for dependencies and generates build files. Expected output includes messages confirming the availability of required libraries and compiler versions.

5.2 Building ns-3

Compile ns-3 using the build command:

```
./ns3 build
```

This process may take several minutes depending on system performance. The build system compiles all ns-3 modules, examples, and tests. Progress is displayed in the terminal with messages indicating which modules are being compiled.

Common Build Issues:

- **Compilation errors due to missing headers:** Install missing development libraries
- **Out of memory errors:** Reduce parallel build jobs using `./ns3 build -j2` (limits to 2 parallel jobs)
- **Permission errors:** Ensure write permissions in the ns-3 directory

5.3 Running Built-in Examples

Verify the installation by running a simple built-in example. The `hello-simulator` example is a minimal program that prints a message:

```
./ns3 run hello-simulator
```

Expected Output:

```
Hello Simulator
```

If this message appears, ns-3 is correctly installed and functional.
Next, test a wireless example to ensure WiFi modules are working:

```
./ns3 run wifi-simple-adhoc
```

This example simulates a basic ad-hoc WiFi network. If no errors occur and the program completes successfully, wireless components are operational.

5.4 Verifying Successful Installation

A successful installation should meet the following criteria:

1. Configuration completes without fatal errors
2. Build process finishes without compilation failures
3. Built-in examples execute and produce expected output
4. No missing library warnings during execution

To perform a comprehensive verification, run the ns-3 test suite:

```
./test.py
```

This executes hundreds of unit tests covering various modules. While not all tests may pass on every system (due to platform-specific quirks), the majority should succeed. Review the test report for any critical failures.

5.5 Post-Build Configuration

After a successful build, consider setting up environment variables for convenience:

```
echo 'export NS3_HOME=~/ns-3' >> ~/.bashrc
echo 'export PATH=$NS3_HOME:$PATH' >> ~/.bashrc
source ~/.bashrc
```

This allows running `ns3` commands from any directory (though simulations should still be placed in the `scratch/` directory).

6 Basic MANET Simulation

Mobile Ad-hoc Networks (MANETs) are self-configuring networks of mobile devices connected by wireless links. Unlike traditional networks with fixed infrastructure, MANETs operate without centralized access points or routers. Each node acts as both a host and a router, forwarding packets for other nodes.

6.1 Understanding MANET Concepts

6.1.1 Key Characteristics

MANETs exhibit several unique characteristics:

- **Dynamic Topology:** Node mobility causes frequent topology changes
- **Decentralized Architecture:** No fixed infrastructure or central controller
- **Limited Resources:** Nodes often have constraints in power, bandwidth, and processing
- **Multi-hop Communication:** Nodes communicate through intermediate nodes when out of direct range
- **Autonomous Operation:** Each node independently makes routing decisions

6.1.2 Routing in MANETs

Routing protocols for MANETs differ fundamentally from traditional routing protocols due to network dynamics. They are categorized as:

- **Proactive (Table-driven):** Routes are computed and maintained continuously (e.g., DSDV, OLSR)
- **Reactive (On-demand):** Routes are discovered only when needed (e.g., AODV, DSR)
- **Hybrid:** Combines proactive and reactive approaches (e.g., ZRP)

This guide demonstrates **AODV (Ad-hoc On-Demand Distance Vector)**, a popular reactive routing protocol that discovers routes only when a node needs to send data.

6.2 Components of the Simulation

Our basic MANET simulation includes the following components:

1. **Nodes:** Representing mobile devices (e.g., smartphones, laptops)
2. **WiFi Ad-hoc Configuration:** Setting up wireless communication without access points
3. **Mobility Model:** Defining how nodes move in the simulation area

4. **Internet Stack:** Installing TCP/IP protocols on nodes
5. **Routing Protocol:** AODV for discovering and maintaining routes
6. **Applications:** UDP traffic generation to test connectivity
7. **Simulation Events:** Controlling start, stop, and data flow

6.3 Complete MANET Simulation Code

The following C++ program implements a basic MANET simulation with 10 nodes moving randomly. Nodes 0 and 9 communicate via UDP, with intermediate nodes forwarding packets using AODV routing.

File: Save this code as `manet-basic.cc` in the `scratch/` directory.

```
/*
 * Basic MANET Simulation with AODV Routing
 *
 * This simulation creates a mobile ad-hoc network with 10 nodes
 * moving randomly in a 1000x1000m area. Node 0 sends UDP traffic
 * to Node 9, and AODV routing discovers the multi-hop path.
 *
 * Outputs: PCAP trace files for packet analysis
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"
#include "ns3/aodv-module.h"
#include "ns3/applications-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("BasicManetSimulation");

int main(int argc, char *argv[])
{
    // ========== Simulation Parameters ==========
    uint32_t numNodes = 10;              // Number of mobile nodes
    double simTime = 100.0;              // Total simulation time
       in seconds
    double txRange = 250.0;              // Transmission range in
       meters
    std::string phyMode = "DsssRate1Mbps"; // Physical layer data
       rate

    // Allow command-line arguments to override defaults
    CommandLine cmd;
    cmd.AddValue("numNodes", "Number of nodes", numNodes);
    cmd.AddValue("simTime", "Simulation time", simTime);
```

```
35      cmd.Parse(argc, argv);
36
37      // Enable logging for debugging (optional, can be commented
            out)
38      // LogComponentEnable("BasicManetSimulation", LOG_LEVEL_INFO);
39      // LogComponentEnable("AodvRoutingProtocol", LOG_LEVEL_INFO);
40
41      // ========== Create Nodes ==========
42      // NodeContainer is a collection of network nodes
43      NodeContainer adhocNodes;
44      adhocNodes.Create(numNodes);
45      NS_LOG_INFO("Created " << numNodes << " mobile nodes");
46
47      // ========== Configure WiFi Physical Layer ==========
48      // YansWifiPhyHelper manages the physical layer (radio
            characteristics)
49      // YansWifiChannelHelper manages the wireless channel (
            propagation model)
50
51      WifiHelper wifi;
52      wifi.SetStandard(WIFI_STANDARD_80211b); // Use 802.11b
            standard
53
54      YansWifiPhyHelper wifiPhy;
55      YansWifiChannelHelper wifiChannel;
56
57      // Set propagation delay model (constant speed of light)
58      wifiChannel.SetPropagationDelay("ns3::
            ConstantSpeedPropagationDelayModel");
59
60      // Set propagation loss model (Friis for open space)
61      // Range is determined by transmission power and receiver
            sensitivity
62      wifiChannel.AddPropagationLoss("ns3::FriisPropagationLossModel
            ");
63
64      wifiPhy.SetChannel(wifiChannel.Create());
65
66      // ========== Configure WiFi MAC Layer ==========
67      // WifiMacHelper configures the MAC layer for ad-hoc mode
68      WifiMacHelper wifiMac;
69      wifiMac.SetType("ns3::AdhocWifiMac"); // Ad-hoc mode (no AP
            required)
70
71      // Set remote station manager (rate control algorithm)
72      wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",
73                                   "DataMode", StringValue(phyMode
                                       ),
74                                   "ControlMode", StringValue(
                                       phyMode));
75
```

```
76      // Install WiFi devices on all nodes
77      NetDeviceContainer adhocDevices = wifi.Install(wifiPhy,
            wifiMac, adhocNodes);
78      NS_LOG_INFO("WiFi devices installed on all nodes");
79
80      // ========== Configure Mobility Model ==========
81      // MobilityHelper assigns mobility patterns to nodes
82      MobilityHelper mobility;
83
84      // Set initial positions: random within a 1000x1000 meter area
85      mobility.SetPositionAllocator("ns3::
            RandomRectanglePositionAllocator",
86                                    "X", StringValue("ns3::
                                        UniformRandomVariable[Min
                                        =0.0|Max=1000.0]"),
87                                    "Y", StringValue("ns3::
                                        UniformRandomVariable[Min
                                        =0.0|Max=1000.0]"));
88
89      // Set mobility model: Random Waypoint
90      // Nodes move to random positions at random speeds with pause
            times
91      mobility.SetMobilityModel("ns3::RandomWaypointMobilityModel",
92                                "Speed", StringValue("ns3::
                                    UniformRandomVariable[Min=5.0|
                                    Max=15.0]"), // 5-15 m/s
93                                "Pause", StringValue("ns3::
                                    ConstantRandomVariable[Constant
                                    =2.0]"),     // 2 second pause
94                                "PositionAllocator", PointerValue(
                                    CreateObject<
                                    RandomRectanglePositionAllocator
                                    >()));
95
96      mobility.Install(adhocNodes);
97      NS_LOG_INFO("Random Waypoint mobility model installed");
98
99      // ========== Install Internet Stack ==========
100     // InternetStackHelper installs TCP/IP protocols on nodes
101     InternetStackHelper internet;
102
103     // Use AODV routing protocol
104     AodvHelper aodv;
105     internet.SetRoutingHelper(aodv); // Install AODV as the
            routing protocol
106     internet.Install(adhocNodes);
107     NS_LOG_INFO("Internet stack with AODV routing installed");
108
109     // ========== Assign IP Addresses ==========
110     // Ipv4AddressHelper assigns IPv4 addresses to network devices
111     Ipv4AddressHelper address;
```

```
112     address.SetBase("10.1.1.0", "255.255.255.0"); // Network:
            10.1.1.0/24
113     Ipv4InterfaceContainer adhocInterfaces = address.Assign(
            adhocDevices);
114     NS_LOG_INFO("IP addresses assigned to all nodes");
115
116     // ========== Set Up Applications ==========
117     // Application 1: UDP Server on Node 9 (receives packets)
118     uint16_t port = 9; // UDP port number
119
120     // PacketSink receives and counts packets
121     PacketSinkHelper sinkHelper("ns3::UdpSocketFactory",
122                                 InetSocketAddress(Ipv4Address::
                                        GetAny(), port));
123     ApplicationContainer sinkApp = sinkHelper.Install(adhocNodes.
            Get(9));
124     sinkApp.Start(Seconds(0.0));    // Server starts at t=0
125     sinkApp.Stop(Seconds(simTime)); // Server stops at end of
            simulation
126
127     // Application 2: UDP Client on Node 0 (sends packets)
128     OnOffHelper onOffHelper("ns3::UdpSocketFactory",
129                             InetSocketAddress(adhocInterfaces.
                                    GetAddress(9), port));
130
131     // Configure traffic pattern: constant bit rate
132     onOffHelper.SetConstantRate(DataRate("250Kbps")); // 250
            kilobits per second
133     onOffHelper.SetAttribute("PacketSize", UintegerValue(512)); //
             512-byte packets
134
135     ApplicationContainer clientApp = onOffHelper.Install(
            adhocNodes.Get(0));
136     clientApp.Start(Seconds(10.0));  // Client starts at t=10 (
            allow time for routing)
137     clientApp.Stop(Seconds(simTime)); // Client stops at end of
            simulation
138
139     NS_LOG_INFO("UDP client-server applications configured");
140
141     // ========== Enable PCAP Tracing ==========
142     // PCAP files allow packet-level analysis using tools like
            Wireshark
143     // Generate PCAP trace for all WiFi devices
144     wifiPhy.EnablePcapAll("manet-basic");
145     NS_LOG_INFO("PCAP tracing enabled");
146
147     // ========== Run Simulation ==========
148     NS_LOG_INFO("Starting simulation for " << simTime << " seconds
            ...");
149     Simulator::Stop(Seconds(simTime));
```

```
150     Simulator::Run();
151
152     // ========== Cleanup and Finish ==========
153     Simulator::Destroy();
154     NS_LOG_INFO("Simulation complete");
155
156     return 0;
157 }
```

6.4 Code Explanation

The simulation code is structured into logical sections:

1. **Parameter Definition:** Configurable parameters like number of nodes, simulation duration, and transmission range
2. **Node Creation:** `NodeContainer` creates a collection of network nodes
3. **WiFi Configuration:** Physical and MAC layers are configured for ad-hoc mode without access points
4. **Mobility:** Nodes are assigned initial positions and movement patterns using Random Waypoint model
5. **Internet Stack:** TCP/IP protocols and AODV routing are installed on all nodes
6. **IP Addressing:** Each node receives a unique IP address in the 10.1.1.0/24 subnet
7. **Applications:** UDP client on Node 0 sends traffic to UDP server on Node 9
8. **Tracing:** PCAP files are generated for detailed packet analysis
9. **Simulation Execution:** The discrete-event simulator runs for the specified duration

7 Running the Simulation

After creating the MANET simulation script, the next step is to compile and execute it. This section walks through the process and explains expected behaviors.

7.1 Compilation

Navigate to the ns-3 root directory and compile the simulation:

```
cd ~/ns-3
./ns3 run manet-basic
```

The build system automatically detects new files in the `scratch/` directory, compiles them, and executes the resulting binary.

Expected Console Output:

During execution, you should see log messages (if logging is enabled) indicating:

- Node creation
- WiFi device installation
- Mobility configuration
- Application start times
- Simulation progress

The simulation runs for 100 seconds (simulated time), which typically completes in a few seconds of real time on modern hardware.

7.2 Command-Line Arguments

The simulation accepts optional command-line arguments to modify parameters without editing the code:

```
./ns3 run "manet-basic --numNodes=15 --simTime=200"
```

This command runs the simulation with 15 nodes for 200 seconds.

To see available arguments:

```
./ns3 run "manet-basic --PrintHelp"
```

7.3 Expected Behavior

The simulation demonstrates the following behavior:

1. **Initialization (0-10s):** Nodes are positioned randomly and begin moving. The AODV protocol remains idle as no traffic is generated yet.
2. **Route Discovery (10s):** Node 0 starts sending UDP traffic to Node 9. Since no route exists, AODV initiates route discovery by broadcasting Route Request (RREQ) messages.

3. **Route Establishment:** Intermediate nodes forward RREQs. When Node 9 receives the RREQ, it sends a Route Reply (RREP) back to Node 0, establishing the path.

4. **Data Transmission (10-100s):** UDP packets flow from Node 0 to Node 9 along the discovered route. As nodes move, routes may break, triggering new route discoveries.

5. **Route Maintenance:** AODV monitors route status. If a link breaks (nodes move out of range), error messages (RERR) propagate, and new routes are discovered.

7.4 Common Runtime Issues

Issue	Solution
Compilation errors	Ensure code syntax is correct; check for missing semicolons or brackets
No output files generated	Verify PCAP tracing is enabled; check file permissions in ns-3 directory
Simulation runs but no packets received	Increase transmission range or reduce area size to ensure connectivity
Segmentation fault	Check for null pointer access; ensure all objects are properly initialized
Extremely slow execution	Reduce number of nodes or simulation time; use optimized build profile

Table 4: Common runtime issues and solutions

7.5 Debugging Tips

To debug simulations:

- Enable logging for specific components by uncommenting `LogComponentEnable` lines
- Use `NS_LOG` environment variable:

```
NS_LOG="AodvRoutingProtocol=level_info" ./ns3 run manet-basic
```

- Add custom print statements using `std::cout` or `NS_LOG_INFO`
- Examine PCAP files to verify packet transmission

8 Output and Analysis

After running the simulation, ns-3 generates several output files that allow detailed analysis of network behavior. This section explains how to interpret and analyze simulation results.

8.1 PCAP Files

PCAP (Packet Capture) files record all packets transmitted and received by network devices. These files are compatible with packet analysis tools like Wireshark, tcpdump, and tshark.

8.1.1 Locating PCAP Files

After running `manet-basic`, PCAP files are created in the ns-3 root directory with names following the pattern:

```
manet-basic-X-Y.pcap
```

where:

- `X` is the node ID (0 to numNodes-1)
- `Y` is the device ID (typically 0 for the first WiFi interface)

Example files:

```
manet-basic-0-0.pcap  # Node 0's WiFi device
manet-basic-9-0.pcap  # Node 9's WiFi device
```

8.1.2 Analyzing PCAP Files with Wireshark

For local installations with GUI support, open PCAP files using Wireshark:

```
wireshark manet-basic-0-0.pcap
```

In Wireshark, you can:

- Filter packets by protocol (e.g., `udp`, `aodv`, `icmp`)
- Examine packet headers and payloads
- View sequence diagrams showing communication flow
- Calculate statistics like throughput, packet loss, and delay

8.1.3 Command-Line Analysis with tshark

For headless environments (WSL, Codespaces), use `tshark`:
Install tshark:

```
sudo apt install tshark
```

Count total packets:

```
tshark -r manet-basic-0-0.pcap | wc -l
```

Filter UDP packets:

```
tshark -r manet-basic-0-0.pcap -Y "udp"
```

Extract AODV routing packets:

```
tshark -r manet-basic-0-0.pcap -Y "aodv"
```

Generate statistics:

```
tshark -r manet-basic-0-0.pcap -q -z io,stat,1
```

8.2 Log Files

If logging is enabled during simulation, console output can be redirected to a file for later analysis:

```
./ns3 run manet-basic > simulation.log 2>&1
```

Log files contain timestamped events showing:

- Route discovery messages
- Packet transmission and reception
- Routing table updates
- Application-level events

8.3 Performance Metrics

Key performance metrics for MANET evaluation include:

- **Packet Delivery Ratio (PDR):** Percentage of successfully delivered packets
- **End-to-End Delay:** Average time for packets to travel from source to destination
- **Routing Overhead:** Number of control packets (RREQ, RREP, RERR) relative to data packets
- **Throughput:** Amount of data successfully received per unit time
- **Route Discovery Time:** Time taken to establish a route after traffic initiation

These metrics can be extracted from PCAP files or calculated using custom instrumentation in the simulation code.

8.4 Exporting Results

To analyze PCAP files on a different machine, download them from your simulation environment:

From WSL:

```
cp ~/ns-3/manet-basic-*.pcap /mnt/c/Users/YourUsername/Desktop/
```

From Codespaces: Use the VS Code interface to download files, or use `scp` if SSH access is configured.

8.5 Limitations in Cloud Environments

When working in headless environments (Codespaces, remote servers):

- GUI-based visualization tools like NetAnim cannot be used
- Wireshark requires X11 forwarding or VNC for remote display
- Analysis must rely on command-line tools and downloaded traces

Despite these limitations, comprehensive analysis is still possible using `tshark`, Python scripts, and custom logging.

9 Common Mistakes and Troubleshooting

This section catalogs frequent errors encountered by ns-3 beginners and provides actionable solutions. Understanding these pitfalls helps accelerate the learning process and reduces frustration.

9.1 Compilation and Build Errors

Problem	Cause	Solution
Undefined reference to `ns3::`	Missing module in includes	Add required `#include` directives and ensure module is built
`error: 'LogComponentEnable' was not declared`	Missing `core-module.h`	Include `ns3/core-module.h` at top of file
`Pure virtual function called`	Incorrect object initialization	Ensure all Ptr<> objects are created with CreateObject<> or factory methods
Linker errors about missing symbols	Module not linked	Verify all required modules are included in header files
`Waf: command not found`	Outdated ns-3 version	Use `./ns3` instead of `./waf` for newer versions

Table 5: Common compilation errors

9.2 Runtime Errors

Problem	Cause	Solution
Nodes never communicate	Transmission range too small	Increase transmission power or reduce simulation area
Routing protocol not working	Internet stack installed before routing helper	Set routing helper before installing Internet stack
Applications don't start	Start time equals simulation stop time	Ensure app start time < stop time < simulation duration
Segmentation fault at runtime	Accessing uninitialized pointer	Check that all Ptr<> objects are properly created
No PCAP files generated	Tracing not enabled or permissions issue	Verify EnablePcapAll() call and directory write permissions
Simulation hangs indefinitely	Infinite loop or deadlock	Add debug print statements to identify stuck section

Table 6: Common runtime errors

9.3 Conceptual Misunderstandings

Problem	Cause	Solution
Poor packet delivery in MANET	Network partitioning due to mobility	Increase node density, reduce mobility speed, or use smaller area
High routing overhead	Frequent topology changes	Adjust mobility parameters or use proactive routing for stable scenarios
Unexpected packet loss	Collision or buffer overflow	Examine PCAP for MAC-layer retransmissions; adjust transmission rate
Different results on each run	Random number generation	Use SetSeed() and SetRun() for reproducible simulations
IP address conflicts	Manual address assignment error	Use Ipv4AddressHelper for automatic, conflict-free addressing

Table 7: Conceptual issues and solutions

9.4 Performance Issues

Problem	Cause	Solution
Extremely slow simulation	Debug build profile	Reconfigure with `-build-profile=optimized`
High memory usage	Too many nodes or long simulation	Reduce scale or use batch processing for multiple runs
Excessive PCAP file size	Tracing all nodes	Enable tracing only for specific nodes of interest
Build takes very long	All modules being compiled	Build only required modules using `-enable-modules` flag

Table 8: Performance optimization tips

9.5 Debugging Strategies

When encountering errors, follow this systematic debugging approach:

1. **Read error messages carefully:** Compiler and runtime errors often indicate exactly what's wrong
2. **Enable logging:** Use `NS_LOG` to trace execution flow
3. **Simplify the scenario:** Reduce number of nodes, disable features to isolate the problem

4. **Verify assumptions:** Check that routes exist, nodes are within range, timing is correct
5. **Consult documentation:** ns-3 manual and API reference contain detailed explanations
6. **Search online:** ns-3 users mailing list archive and Stack Overflow have solutions to common issues
7. **Use version control:** Git allows reverting to known working states

9.6 Getting Help

If problems persist after attempting these solutions:

- **ns-3 Users Mailing List:** https://groups.google.com/g/ns-3-users
- **Official Documentation:** https://www.nsnam.org/documentation/
- **Stack Overflow:** Tag questions with `ns-3`
- **GitHub Issues:** Report bugs at https://gitlab.com/nsnam/ns-3-dev/-/issues

When asking for help, provide:

- ns-3 version
- Operating system and version
- Complete error messages
- Minimal reproducible example
- Steps already attempted

10 Conclusion

This guide has provided a comprehensive introduction to ns-3, covering installation, configuration, directory structure, and the implementation of a basic MANET simulation. By following the steps outlined in this document, students have gained the foundational knowledge necessary to begin conducting network simulation experiments.

10.1 Summary of Achievements

Upon completing this guide, students have:

- Successfully installed and configured ns-3 on their chosen platform (Ubuntu, WSL, or Codespaces)
- Understood the ns-3 directory structure and build system
- Learned the fundamental concepts of Mobile Ad-hoc Networks
- Implemented a complete MANET simulation with AODV routing
- Executed the simulation and generated PCAP trace files
- Gained familiarity with packet analysis and performance evaluation techniques
- Identified common pitfalls and troubleshooting strategies

10.2 Extending This Foundation

The basic MANET simulation presented here serves as a starting point for more advanced experiments. Students can extend this work in several directions:

10.2.1 Protocol Comparison

Compare different routing protocols (AODV, DSDV, OLSR, DSR) under identical conditions to evaluate:

- Packet delivery ratio
- Average end-to-end delay
- Routing overhead
- Route convergence time

10.2.2 Mobility Models

Experiment with different mobility patterns:

- Constant Position (static network)
- Random Walk
- Gauss-Markov
- Reference Point Group Mobility (for modeling group movement)

10.2.3 Traffic Patterns

Modify application layer to simulate realistic scenarios:

- Multiple concurrent flows
- Variable bit rate traffic
- TCP-based applications
- Bursty traffic patterns

10.2.4 Network Conditions

Introduce challenging conditions to test protocol robustness:

- Varying node density
- Obstacles using building propagation models
- Energy constraints and node failures
- Malicious nodes (security research)

10.2.5 Custom Metrics and Instrumentation

Enhance analysis capabilities by:

- Implementing custom trace sources
- Collecting application-level statistics
- Exporting data to CSV for processing in Python/MATLAB
- Visualizing results with graphs and charts

10.3 Best Practices for Research

When using ns-3 for academic research or projects:

- **Document thoroughly:** Maintain detailed notes on simulation parameters and assumptions
- **Version control:** Use Git to track changes and collaborate effectively
- **Reproducibility:** Set random seeds to ensure consistent results across runs
- **Statistical rigor:** Run multiple simulations with different seeds and report confidence intervals
- **Validation:** Verify results against analytical models or published research when possible
- **Cite properly:** Reference ns-3 in publications using official citation format

10.4 Further Resources

To deepen understanding of ns-3 and network simulation:

- **ns-3 Tutorial:** Comprehensive tutorial available in the `doc/` directory
- **ns-3 Manual:** Detailed documentation of all modules and APIs
- **Example Programs:** Explore `examples/` directory for advanced use cases
- **Research Papers:** Read papers using ns-3 to understand experiment design
- **Online Courses:** Platforms like Coursera and edX offer network simulation courses
- **Books:** "Introduction to Network Simulator NS2" covers foundational concepts applicable to ns-3

10.5 Final Remarks

Network simulation is an invaluable skill for computer science students specializing in networking, wireless communications, and distributed systems. ns-3 provides a powerful, flexible platform for investigating complex network behaviors that would be difficult or impossible to study through purely theoretical approaches or physical experimentation.

While the learning curve can be steep initially, persistence and systematic study yield significant dividends. The ability to model, simulate, and analyze network protocols empowers students to contribute meaningfully to research, understand real-world network deployments, and design next-generation communication systems.

This guide has equipped you with the essential tools and knowledge to begin your journey with ns-3. Continue exploring, experimenting, and pushing the boundaries of what you can achieve with network simulation. The skills developed through working with ns-3 extend far beyond the simulator itself, fostering critical thinking, problem-solving abilities, and deep understanding of network protocols that will serve you throughout your career.

Good luck with your simulations and future research endeavors!