MIT World Peace University

Wireless Devices and Mobile Security
Third Year B. Tech, Semester 5

Installation and Configuation of Network Simulator 2 (NS2)

Lab Assignment 1

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

September 24, 2023

Contents

1 Aim

Install and Configure Network Simulator tool such as Network Simulator 2 or NetSim or QualNet and study its components and eco system.

2 Theory

2.1 Simulation

Simulation is an invaluable methodology for comprehending complex real-world phenomena by creating and studying theoretical models in a controlled digital environment. It facilitates learning through experimentation and can be applied across various domains.

2.2 Computer Simulation

Computer simulation is a specialized branch of simulation that focuses on designing theoretical physical systems and processes on digital computers. Key aspects include:

1. **Model Creation** Computer simulation begins with the creation of mathematical or algorithmic models that represent the behavior of a real-world system. These models define the relationships between various components and variables within the system.

2. **Execution** The model is implemented in software, allowing it to run on a computer. The simulation software calculates the system's behavior over time, considering inputs, parameters, and initial conditions.

3. **Analysis** Simulations generate vast amounts of data, enabling in-depth analysis. Researchers can observe the system's behavior, identify patterns, and draw conclusions about its performance and characteristics.

4. **Applications** Computer simulation finds applications in diverse fields, including physics, engineering, economics, biology, and social sciences. It helps researchers, engineers, and decision-makers test hypotheses, optimize designs, and make informed choices without costly real-world experimentation.

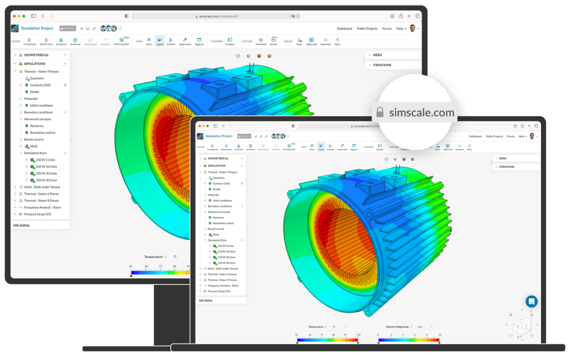


Figure 1: Computer Simulation

2.3 Network Simulation

Network simulation is a subset of computer simulation that concentrates on modeling and assessing the behavior of computer networks, such as local area networks (LANs) and wide area networks (WANs). Key considerations in network simulation include:

1. **Topology Modeling** Network simulators allow users to create intricate network topologies by defining nodes, links, routers, switches, and their interconnections. These models closely mimic real-world network structures.

2. **Traffic Generation** Simulators enable the generation of network traffic, simulating data transmission, routing, and congestion scenarios. Users can specify the type and volume of traffic to assess network performance.

3. **Protocol Evaluation** Network simulation tools assist in evaluating network protocols, including routing, transport, and application-layer protocols. Researchers can analyze how different protocols interact and impact network behavior.

4. **Scenario Testing** Network simulation provides a controlled environment for testing various network scenarios, such as load balancing, fault tolerance, and security measures. This helps in identifying vulnerabilities and optimizing network configurations.

5. **Predictive Analysis** By running simulations with different parameters and configurations, network professionals can predict how changes will affect network performance, aiding in informed decision-making.

2.4 Network Simulator (NS2)

Network Simulator, commonly referred to as NS2, is a widely used open-source software tool designed for network simulation. It has become an essential platform for researchers, engineers, and educators working on network-related projects. NS2 offers an array of features and capabilities, including:

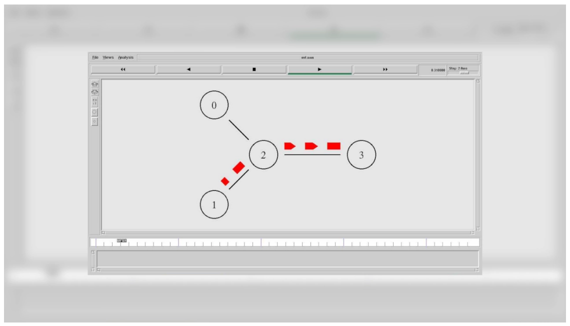


Figure 2: Network Simulator 2

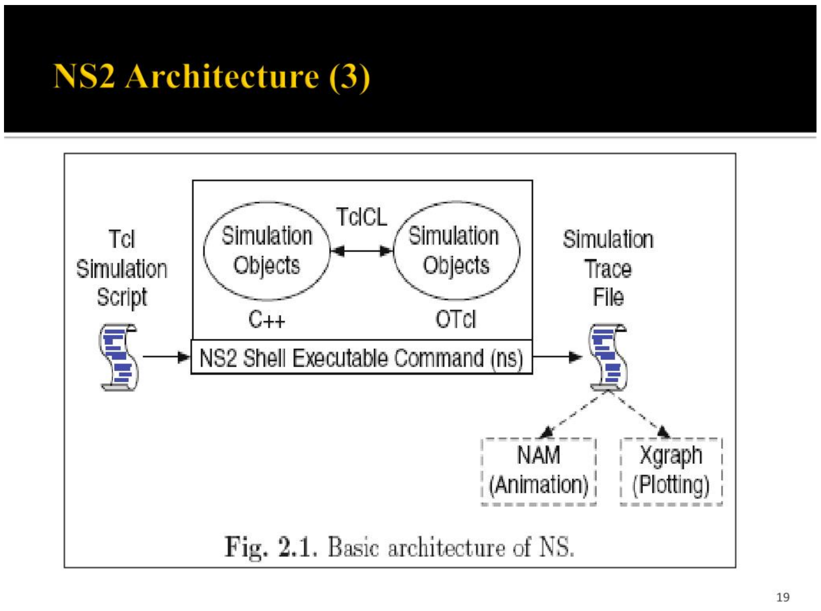


Figure 3: Network Simulator 2 Architecture

2.4.1 Features of NS2

NS2 encompasses a range of features that make it a powerful choice for network simulation:

1. **Protocol Support** NS2 supports a wide variety of network protocols, making it versatile for simulating different types of networks, including wired and wireless.
2. **Modularity** The simulator's modular architecture allows users to extend and customize its functionality by adding new modules and protocols.
3. **Visualization Tools** NS2 includes visualization tools like NAM (Network Animator), which aids in visualizing network topologies and real-time simulation results.
4. **Community Support** NS2's open-source nature has fostered a vibrant community of users and developers who contribute scripts, patches, and extensions, enhancing its capabilities.
5. **Integration** NS2 integrates both C++ for internal mechanisms and OTcl (Object-oriented Tool Command Language) for configuration and simulation setup. This integration provides flexibility and efficiency.
6. **Backward Compatibility** NS2 maintains backward compatibility with scripts written for its predecessor, NS-1, ensuring the continued use of existing simulation scenarios.

NS2 is widely used for research, educational purposes, and network protocol development, making it an indispensable tool in the field of network simulation.

2.5 TCL Scripting

TCL (Tool Command Language) is a powerful scripting language commonly used for various tasks, including configuration, automation, and simulation. In the context of NS2 (Network Simulator-2) and network simulations, TCL scripting plays a crucial role in defining and configuring simulation scenarios. Here are key aspects of TCL scripting:

1. **Scripting Language** TCL is a versatile scripting language known for its simplicity and flexibility. It provides a convenient way to specify actions and configurations in a readable and concise manner.
2. **Simulation Scenario Definition** In NS2, TCL scripts are used to define simulation scenarios, including network topologies, node configurations, traffic patterns, and simulation parameters. Researchers and network engineers write TCL scripts to create custom simulations.
3. **Modular Configuration** TCL scripts in NS2 allow for modular configuration. Users can define different components and their properties, making it easy to modify and reuse parts of the simulation setup.
4. **Event Scheduling** TCL scripting enables event scheduling within NS2 simulations. Events such as packet transmissions, node movements, and protocol behaviors can be scheduled at specific simulation times.
5. **Customization** TCL scripts can be customized to simulate a wide range of network scenarios, from simple point-to-point connections to complex multi-hop wireless networks. Customization is essential for studying specific network behaviors.
6. **Integration with NS2** TCL scripting seamlessly integrates with NS2's C++ backend. TCL scripts configure and control the simulation, while C++ handles the internal mechanisms and communication between simulation objects.
7. **Error Handling** TCL provides error-handling mechanisms to catch and respond to errors during simulation execution. This ensures that simulations run smoothly and accurately.
8. **Visualization** TCL scripts often include commands for visualization tools like NAM (Network Animator). These tools allow users to visualize the simulation in real-time, helping with debugging and analysis.
9. **Community Contributions** The NS2 community has created a wealth of TCL scripts and libraries that can be used as building blocks for various network simulations. These scripts cover a wide range of network scenarios and applications.
10. **Learning TCL** Proficiency in TCL scripting is essential for NS2 users. Learning resources, tutorials, and documentation are available to help users become proficient in TCL scripting for network simulations.

TCL scripting in NS2 empowers researchers and network professionals to create and customize network simulations, enabling them to study, analyze, and optimize network behaviors and protocols effectively.

2.6 Steps to Install NS2 on Ubuntu

Installing NS2 on Ubuntu involves several steps. This guide will walk you through the process to ensure a successful installation.

1. **Update Package List:**

```
- Open a terminal by pressing Ctrl + Alt + T.
- Update the package list to ensure you have the latest
information about available packages. Run the following command:
    sudo apt-get update
```

2. **Install Essential Packages:**

```
- NS2 requires essential packages to be installed on your
system. Install them by running the following command:
    sudo apt-get install build-essential autoconf automake
    libxmu-dev
```

3. **Download NS2 Source Code:**

```
- Visit the official NS2 website to download the NS2 source
code: [NS2 Official Website](https://www.nsnam.org/)
- Choose the version of NS2 that you want to install and
download the source code archive.
```

4. **Extract the Downloaded Archive:**

```
- Navigate to the directory where you downloaded the NS2 source
code archive.
- Extract the archive using the tar command. Replace ns2-version.tar.gz
with the actual filename you downloaded.
    tar -xzvf ns2-version.tar.gz
```

5. **Navigate to the NS2 Directory:**

```
- Change your current directory to the NS2 source code directory
using the cd command. Replace ns2-version with the actual directory name.
    cd ns2-version
```

6. **Build and Install NS2:**

```
- NS2 uses the autoconf and automake tools to configure and build
the software. Run the following commands:
    ./configure
    make
    sudo make install
- This process may take some time as NS2 is compiled and installed
on your system.
```

7. **Test the Installation:**

```
- After the installation is complete, you can test NS2 by running
a simple simulation. Create a TCL script (e.g., mysimulation.tcl)
that defines a basic network scenario and run it using NS2:
    ns mysimulation.tcl
- If NS2 runs the simulation without errors, your installation was
 successful.
```

8. **Optional: Install NAM (Network Animator):**

```
- NAM is a visualization tool for NS2 simulations. You can install
 it using the following command:
    sudo apt-get install nam
- NAM allows you to visualize your network simulations in real-time.
```

9. **You're Done:**

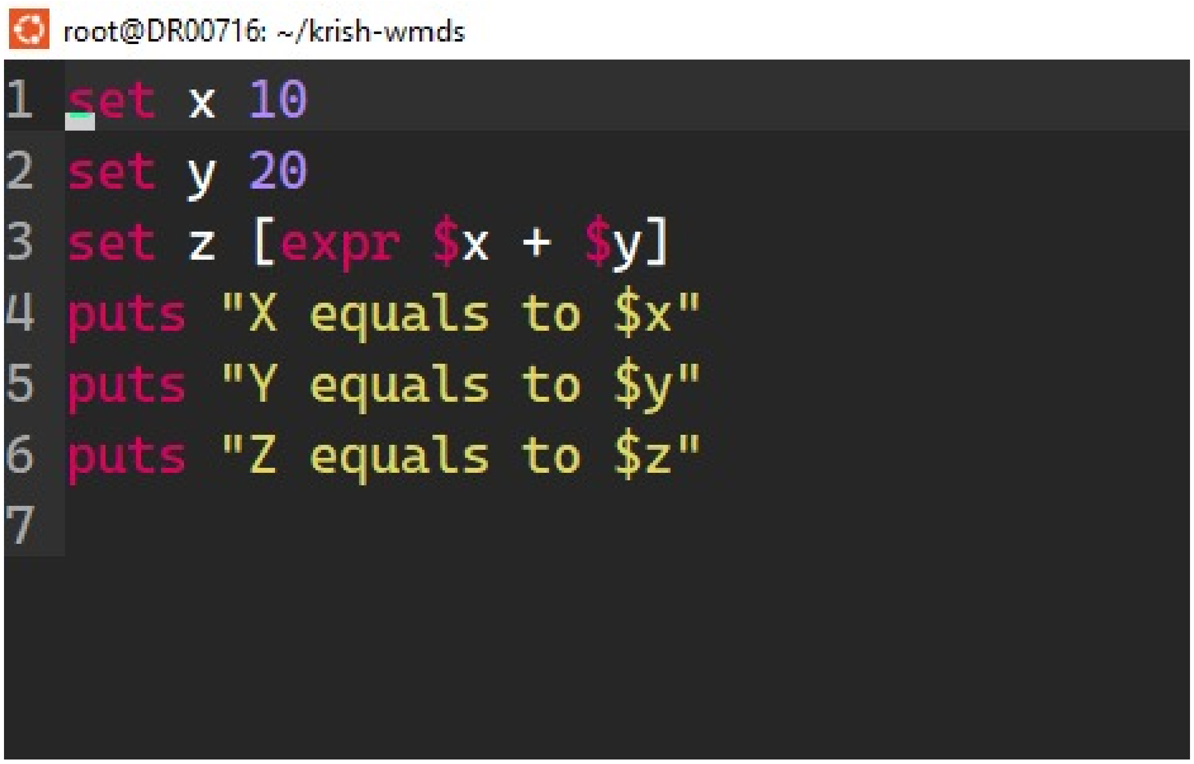
```
- NS2 is now installed on your Ubuntu system, and you're ready to
use it for network simulations.
```

Please note that NS2 installation can be a complex process, and you may encounter dependencies or issues specific to your system. Refer to NS2 documentation and forums for troubleshooting if needed.

3 Platform

Operating System: Ubuntu 22.04 x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters: NS2, NAM 1.4

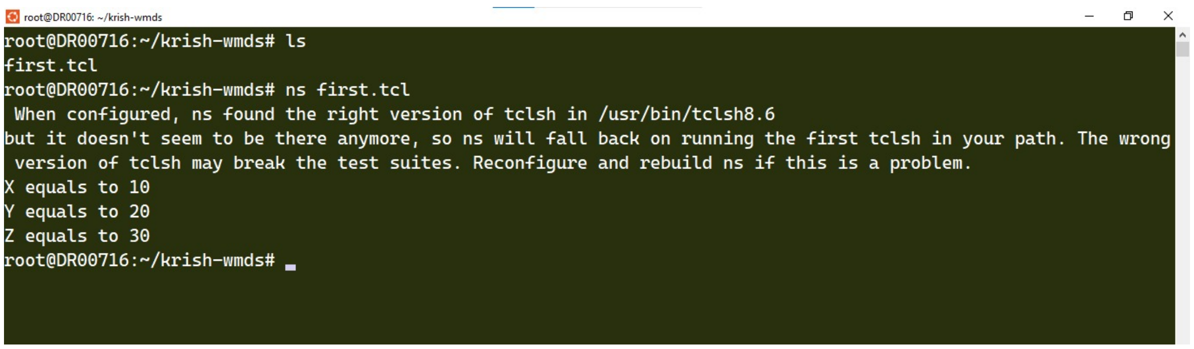
4 Screenshots



root@DR00716: ~/krish-wmds

```
set x 10
set y 20
set z [expr $x + $y]
puts "X equals to $x"
puts "Y equals to $y"
puts "Z equals to $z"
```

Figure 4: First TCL Script



root@DR00716: ~/krish-wmds

```
root@DR00716:~/krish-wmds# ls
first.tcl
root@DR00716:~/krish-wmds# ns first.tcl
 When configured, ns found the right version of tclsh in /usr/bin/tclsh8.6
but it doesn't seem to be there anymore, so ns will fall back on running the first tclsh in your path. The wrong
 version of tclsh may break the test suites. Reconfigure and rebuild ns if this is a problem.
X equals to 10
Y equals to 20
Z equals to 30
root@DR00716:~/krish-wmds# 
```

Figure 5: Running the First TCL Script

5 Conclusion

Thus, we studied, install and configure NS2 on ubuntu.

6 FAQ

1. **Distinguish Emulation and Simulation with Example**

   Emulation and simulation are both techniques used in computer science and engineering but have distinct differences.

   - **Emulation** Emulation involves replicating the behavior of one system or device using another, typically with different hardware or software. It aims to make one system mimic the functions of another. For example, running a PlayStation game on a computer using an emulator software is emulation. The emulator mimics the PlayStation's hardware and software to run the game.

   - **Simulation** Simulation, on the other hand, involves creating a model or representation of a real-world system or process to study its behavior under various conditions. It doesn't necessarily aim to replicate a specific existing system. For example, simulating traffic flow in a city to analyze congestion patterns is simulation. There's no emulation of existing traffic systems; instead, a model is created to understand traffic behavior.

2. **Compare Compiler and Interpreter**

   Compilers and interpreters are both tools used to process and execute code, but they function differently:

   - **Compiler** - Converts the entire source code into machine code or an intermediate code at once. - Produces a separate executable file. - Typically has a longer initial compilation time. - Executes the code faster since it's already translated. - Examples include GCC (GNU Compiler Collection) for C/C++ and Java Compiler for Java.

   - **Interpreter** - Processes and executes code line by line or statement by statement. - Does not produce a separate executable file; it directly interprets the source code. - Usually has a shorter startup time. - Slower execution as it translates and executes code simultaneously. - Examples include Python Interpreter for Python and JavaScript Engine for JavaScript.

3. **List Different Simulation Tools and Compare with NS-2**

   There are several simulation tools available, each with its strengths and weaknesses. Let's compare some of them with NS-2:

   - **NS-2 (Network Simulator-2)** - Focuses on network simulations, making it suitable for studying network protocols and behaviors. - Offers flexibility in creating custom network scenarios. - Provides a large library of built-in network components. - Utilizes both OTcl and C++ for configuration and implementation. - Open-source and widely used in academia and research.

   - **MATLAB/Simulink** - General-purpose simulation tool used for various domains, including control systems and signal processing. - Provides a graphical user interface for model building. - Extensive toolboxes for various applications. - Proprietary software with licensing costs.

   - **AnyLogic** - A multimethod simulation tool suitable for modeling complex systems, including agent-based, discrete event, and system dynamics simulations. - Offers 3D simulation and animation capabilities. - Supports Java-based modeling. - Commercial software with a free Personal Learning Edition.

   - **OMNeT++** - A discrete event simulation framework for network simulations. - Focuses on modeling communication networks. - Supports multiple programming languages, including

C++ and NED (Network Description Language). - Open-source and widely used in the networking community.

- **SimPy** - A Python library for discrete event simulation. - Lightweight and easy to use. - Suitable for small to medium-sized simulations. - Ideal for Python enthusiasts and quick prototyping.

4. **Why Two Languages (OTcl and C++) Used by NS2**

   NS-2 employs both OTcl (Object-oriented Tool Command Language) and C++ for specific purposes:

   - **OTcl** Used for simulation configuration and setup. It provides a high-level, easy-to-use interface for defining network topologies, scenarios, and simulation parameters. OTcl allows quick experimentation and configuration changes without recompiling the entire simulator.

   - **C++** Used for implementing the internal mechanisms and behaviors of network components. C++ provides the necessary performance and low-level control required for accurately simulating network protocols and behaviors. The combination of OTcl and C++ in NS-2 leverages the strengths of both languages for efficient network simulations.

5. **Advantages and Disadvantages of NS-2**

   NS-2 has several advantages and disadvantages:

   Advantages - Widely used in network research and academia. - Open-source and free to use. - Provides a flexible environment for creating custom network scenarios. - Supports various network protocols and components. - Offers visualization tools like NAM for real-time simulation visualization.

   Disadvantages - Learning curve, especially for beginners. - Steeper initial setup compared to some other simulation tools. - Performance limitations for large-scale simulations. - Limited support for wireless network modeling. - Lack of official user-friendly graphical interface.

6. **Draw and Explain Three Kinds of Formats for Wired Networks in NS-2**

   In NS-2, wired networks can be represented in various formats. Here are three common formats:

   1. **Star Topology** - In a star topology, a central node (hub) connects to multiple peripheral nodes (spokes). - All communication between peripheral nodes passes through the central hub. - It is straightforward to implement and provides centralized control. - However, if the hub fails, the entire network can become non-functional.

   2. **Ring Topology** - In a ring topology, each node connects to exactly two other nodes, forming a closed loop. - Data circulates in a unidirectional or bidirectional manner around the ring. - Ring topologies are resilient, as data can take alternate paths if a link or node fails. - However, adding or removing nodes can be complex, and the entire ring can be disrupted if not properly managed.

   3. **Mesh Topology** - Mesh topologies involve multiple nodes interconnected in a complex manner, where each node connects to multiple others. - Data can take multiple paths to reach its destination, enhancing fault tolerance and redundancy. - Mesh topologies are suitable for robust and fault-tolerant networks but can be challenging to manage and configure. - They provide high reliability but can be costly to implement due to the number of connections required.

These are just a few examples of wired network topologies in NS-2, and they demonstrate the flexibility of the simulator in