iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 4 **Issue:** VIII **Month of publication:** August 2016

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

Volume 4 Issue VIII, August 2016
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Modeling wireless network topology with NS3

Farruh Muzafarov[1], Javlonbek Abdujalilov[2]
Department of Telecommunication Engineering, Tashkent University of Information Technologies

Abstract – Data transmission between client-server communications has been modeled using NS3 in this paper. Facilities and elements of ns3 simulation software contents are given. Algorithm, software and configuration of imitation model in NS3 environment are given.
Keywords – Channel, client-server, network, algorithm, layer, topology.

I. INTRODUCTION

This work aims to introduce ns-3 basics to get started using in laboratory of telecommunication subjects. The models that are present in ns-3 cover many more layers of the protocol stack: the most current version of ns-3 supports a complete IPv4, Address Resolution Protocol (ARP), User Datagram Protocol (UDP), and TCP stack. ns-3 contains a few simple models to generate and collect trace but more importantly provides many MAC and Physical (PHY) layers. The oldest such model is a detailed IEEE 802.11 implementation that features both infrastructure and ad-hoc mode, multiple rate control algorithms, interference and propagation loss models, many classic mobility models such as random walk, random direction, etc. More recent additions include a Wimax model, a model of underwater communication systems, and a spectrum-aware channel and PHY modeling framework.

In the real world, only two major protocol interface abstractions are in wide use: the most obvious one is the socket programming interface that defines how user space applications can send and receive trace over layer 2, 3, or 4 protocols. The second abstraction is less widely seen by application developers but crucially important for network simulators: it is the kernel-level interface that defines how the layer 3 protocols and the network cards present in the host communicate. This abstraction is defined by the very similar *net_device* and *ifnet* data structures on Linux and BSD respectively.

For efficiency and performance reasons, the other protocol interfaces have not yet converged to a simple set of common features. For example, the interface between the UDP and the IP layer exposes a lot of details of the IP layer to optimize the case where IP datagrams need to be fragmented to make sure that the UDP layer does not allocate a big buffer to hold the user datagram only to create later a set of smaller fragments in the IP layer.

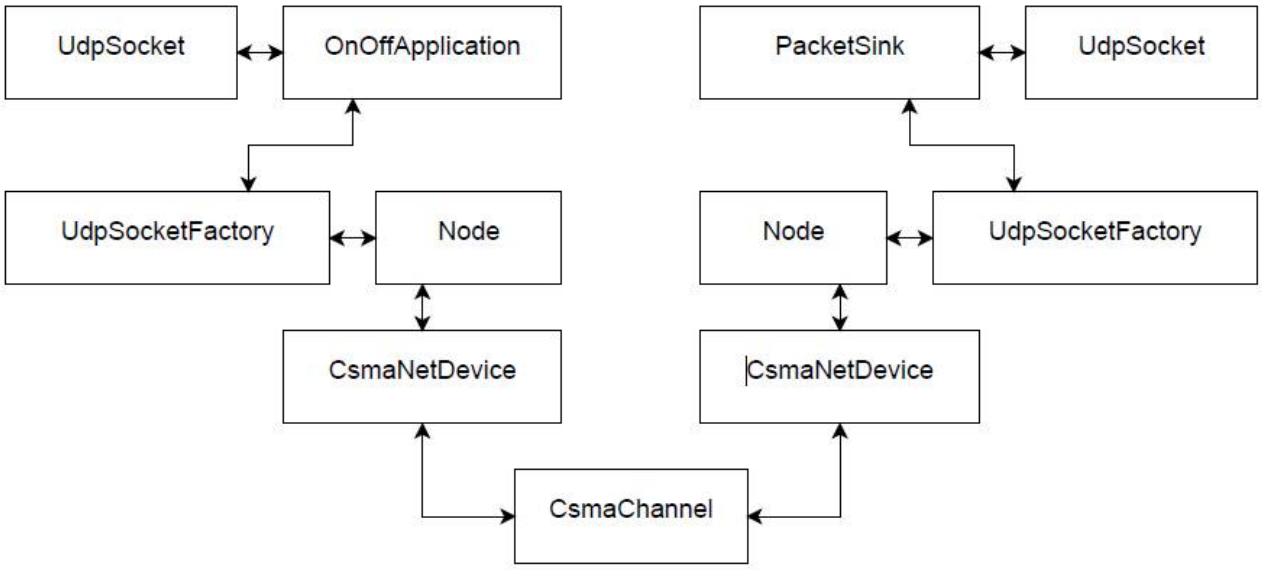


Fig. 1 An ns-3 Network topology

Figure 1. summarizes the relationship between these objects and illustrates how a simple network simulation could be put together to simulate a single link interconnecting a node that sends traffic with a node that receives it.

The *NetDevice* class is based on the Linux kernel-level net device data structure: it represents a single hardware device and allows

©IJRASET: All Rights are Reserved

the upper layers to queue packets for transmission by the hardware and makes it possible for the hardware device to notify the upper layers that a new packet has been received.

A. Channels interconnect NetDevices: each Channel object contains the list of NetDevice objects it is connected to. It represents a network cable, a wireless transmission medium, or a fiber cable.

B. The Node class matches the GTNetS concept of a host system. A Node contains a list of NetDevice and Application objects.

C. The Socket class represents a communication endpoint that can be created by applications and that are used to send and receive traffic to the protocol stacks that are attached to a node.

D. The SocketFactory class can be used to create sockets of various kinds by applications: each protocol that is attached (aggregated) to a node needs to also aggregate a new type of SocketFactory to the node so that applications interested in sending or receiving traffic cover this new protocol can request this socket factory and create sockets from it.

ns-3 adopts a similar structure: most inter-layer interfaces except for the *socket* and net device abstractions are left unspecified to leave as much flexibility as possible to model developers, yet make it easy to inter operate with protocol implementations that expect a *socket* or a net_device interference to be present.

Nodes may/may not have mobility, other traits. Nodes have “network devices”. the motherboard of a computer with RAM, CPU, and, IO interfaces:

1) Network devices transfer packets over channels

2) Incorporating Layer 1 (Physical) & Layer 2 (Link)

E. Devices interface w/ Layer 3 (Network: IP, ARP)

F. Layer 3 supports Layer 4 (Transport: UDP, TCP)

G. Layer 4 is used by Layer 5 (Application) objects. The packet generator and consumer which can run on a Node and talk to a set of network stacks.

H. Socket: the interface between an application and a network stack.

I. NetDevice: a network card which can be plugged in an IO interface of a Node

J. Channel: a physical connector between a set of NetDevice objects

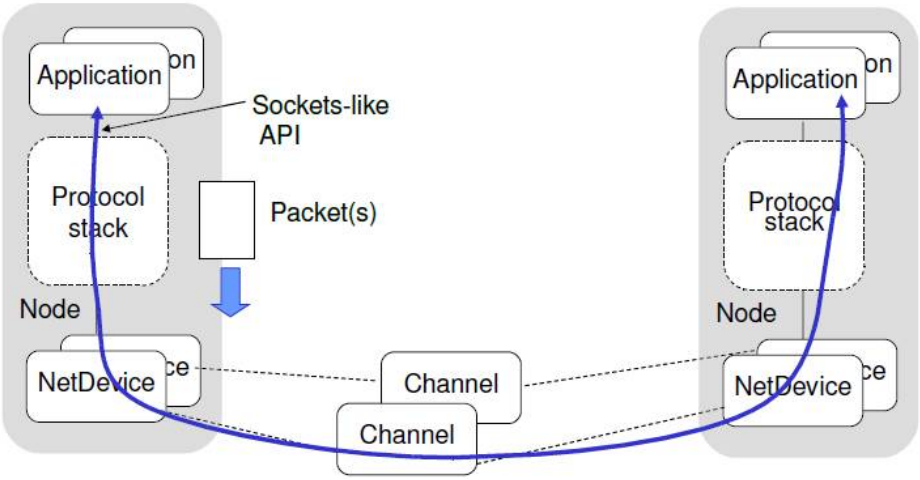


Fig. 2 Simulation Network Architecture

ns-3 is a discrete-event network simulator in which the simulation core and models are implemented in C++. ns-3 is built as a library which may be statically or dynamically linked to a C++ main program that defines the simulation topology and starts the simulator.

II. NETWORK DESIGN AND CONFIGURATION

When you create new topology is being set into the various phases: first you have to choose how many nodes you want to create then you have to install few things on that nodes, like you have to choose the channels, set the attributes on that channel, and then on that channel you have to install few devices, set few rules for those devices which are the internet rules, like protocols and all of that. After that you can do client-server communication and whatever you want to do.

©IJRASET: All Rights are Reserved

Volume 4 Issue VIII, August 2016
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

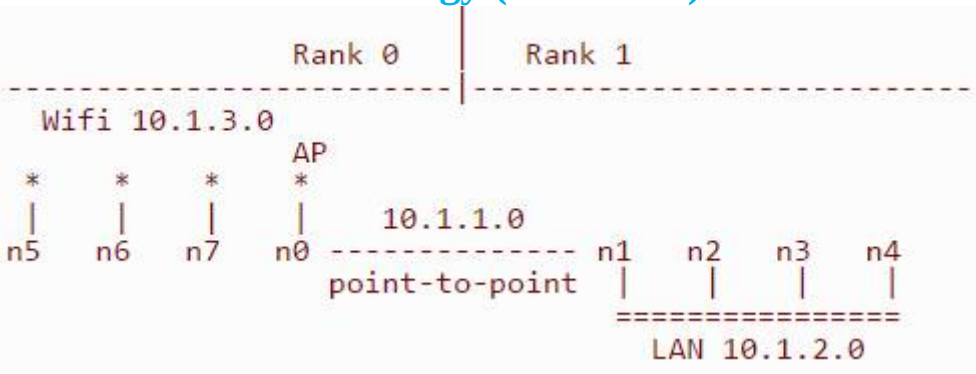


Fig. 1 WiFi network topology

You can see that we are adding a new network device to the node on the left side of the point-to-point link that becomes the access point for the wireless network. A number of wireless STA nodes are created to fill out the new 10.1.3.0 network as shown on the left side of the illustration.

III. SIMULATION DESCRIPTION

In this section we are going to further expand our knowledge of ns-3 network devices and channels to cover an example of a wireless network. Ns-3 provides a set of 802.11 models that attempt to provide an accurate MAC-level implementation of the 802.11 specification and a “not-so-slow” PHY-level model of the 802.11a specification.

As we have seen both point-to-point and CSMA topology helper objects when constructing point-to-point topologies[1], we will see equivalent Wifi topology helpers in this section. The appearance and operation of these helpers should look quite familiar.

We provide an entire script and examine some of the output. Take a look at the ASCII art reproduced below in figure-1 that shows the WiFi network topology constructed. You can see that we are going to further extend our example by hanging a wireless network off of the left side. Notice that this is a default network topology since you can actually vary the number of nodes created on the wired and wireless networks.

We can set nWifi to control how many STA (station) nodes are created in the simulation. There will always be one AP (access point) node on the wireless network. By default there are three “extra” CSMA nodes and three wireless STA nodes.

The code begins by loading module include files. There are a couple of new includes corresponding to the Wifi module and the mobility module which we will discuss below.

```
#include "ns3/core-module.h"
#include "ns3/simulator-module.h"
#include "ns3/node-module.h"
#include "ns3/helper-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
```

The ns-3 namespace is used and a logging component is defined.

```
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("ThirdScriptExample");
```

The main program begins by adding some command line parameters for enabling or disabling logging components and for changing the number of devices created.

```
bool verbose = true;
  uint32_t nCsma = 3;
  uint32_t nWifi = 3;
  CommandLine cmd;
  cmd.AddValue (``nCsma", ``Number of \"extra\" CSMA nodes/devices", nCsma);
  cmd.AddValue (``nWifi", ``Number of wifi STA devices", nWifi);
  cmd.AddValue (``verbose", ``Tell echo applications to log if true", verbose);
  cmd.Parse (argc,argv);
```

©IJRASET: All Rights are Reserved

```
if (verbose)
  {
   LogComponentEnable(``UdpEchoClientApplication", LOG_LEVEL_INFO);
   LogComponentEnable(``UdpEchoServerApplication", LOG_LEVEL_INFO);
}
```

the next step is to create two nodes that we will connect via the point-to-point link.

```
NodeContainer p2pNodes;
p2pNodes.Create (2);
```

Next, We instantiate a PointToPointHelper and set the associated default Attributes so that we create a five megabit per second transmitter on devices created using the helper and a two millisecond delay on channels created by the helper. We then Intall the devices on the nodes and the channel between them.

```
PointToPointHelper pointToPoint;
 pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
 pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
 NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
```

Next, we declare another NodeContainer to hold the nodes that will be part of the bus (CSMA) network.

```
NodeContainer csmaNodes;
 csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);
```

The next line of code Gets the first node (as in having an index of one) from the point-to-point node container and adds it to the container of nodes that will get CSMA devices. The node in question is going to end up with a point-to-point device and a CSMA device. We then create a number of “extra” nodes that compose the remainder of the CSMA network.

We then instantiate a CsmaHelper and set its Attributes as we did in the previous example. We create a NetDeviceContainer to keep track of the created CSMA net devices and then we Install CSMA devices on the selected nodes.

```
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));
NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
```

Next, we are going to create the nodes that will be part of the Wifi network. We are going to create a number of “station” nodes as specified by the command line argument, and we are going to use the “leftmost” node of the point-to-point link as the node for the access point.

```
NodeContainer wifiStaNodes;
 wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode = p2pNodes.Get (0);
```

The next bit of code constructs the wifi devices and the interconnection channel between these wifi nodes. First, we configure the PHY and channel helpers:

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
```

For simplicity, this code uses the default PHY layer configuration and channel models which are documented in the API doxygen documentation for the YansWifiChannelHelper::Default and YansWifiPhyHelper::Default methods. Once these objects are created, we create a channel object and associate it to our PHY layer object manager to make sure that all the PHY layer objects created by the YansWifiPhyHelper share the same underlying channel, that is, they share the same wireless medium and can communication and interfere:

```
phy.SetChannel (channel.Create ());
```

Once the PHY helper is configured, we can focus on the MAC layer. Here we choose to work with non-Qos MACs so we use a NqosWifiMacHelper object to set MAC parameters.

©IJRASET: All Rights are Reserved

Volume 4 Issue VIII, August 2016
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

```
WifiHelper wifi = WifiHelper::Default ();
 wifi.SetRemoteStationManager ("ns3::AarfWifiManager");
NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
```

The SetRemoteStationManager method tells the helper the type of rate control algorithm to use. Here, it is asking the helper to use the AARF algorithm details are, of course, available in Doxygen.

Next, we configure the type of MAC, the SSID of the infrastructure network we want to setup and make sure that our stations don't perform active probing:

IV. SIMULATION RESULTS

Figure 4,5. shows client-server communication point-to-point channel. The time spent for data sent and received by client to server. Client sent 1024 bytes to server's port 9 at time 2 second and server received at a time 2.00369 second from client.

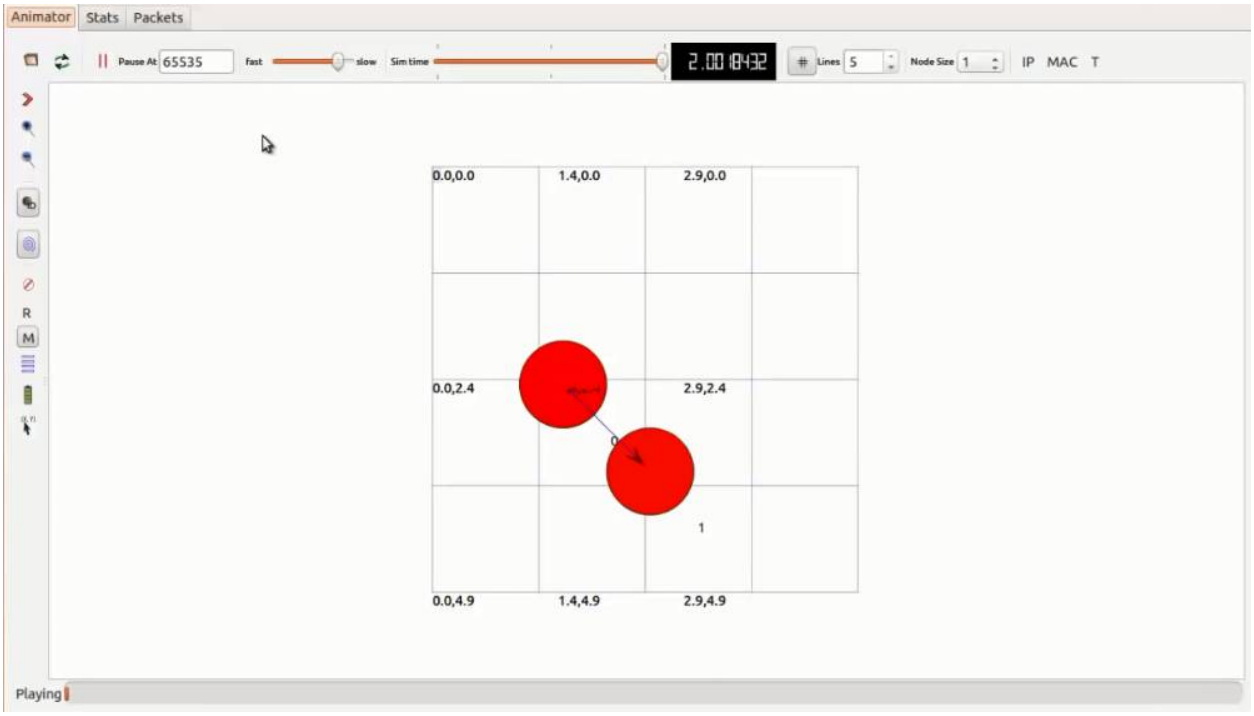


Fig. 4 Client sent 1024 bytes to server

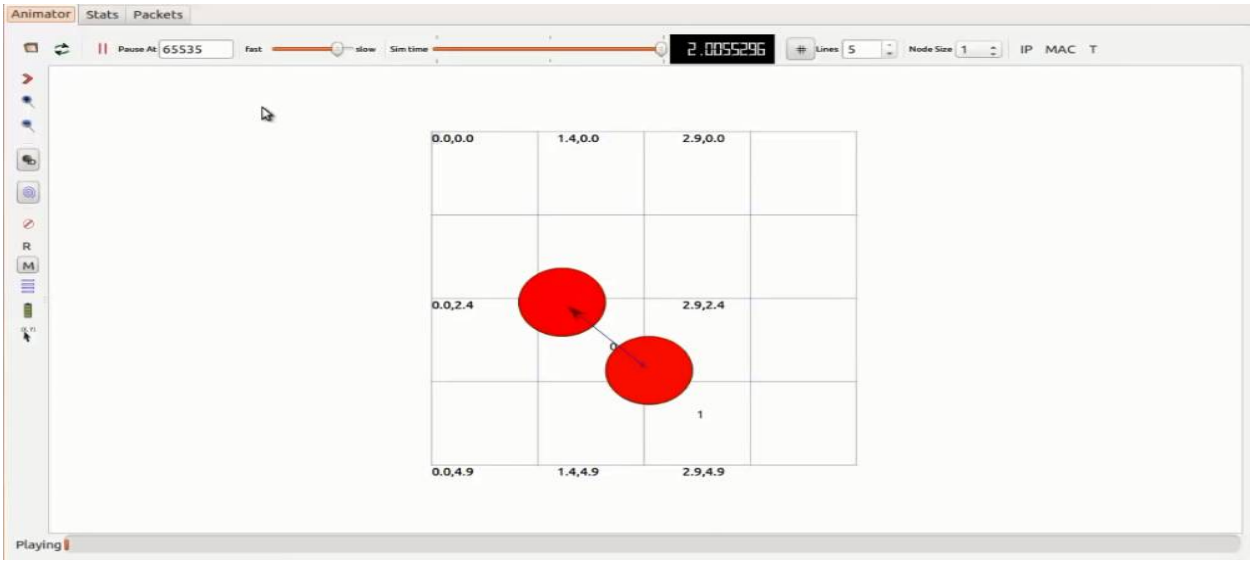


Fig. 5 Server received 1024 bytes from client

©IJRASET: All Rights are Reserved

At time 2s client sent 1024 bytes to 10.1.1.2 port 9
At time 2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.00369 server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.00737s client received 1024 bytes from 10.1.1.2 port 9
When we run the simulation using NetAnim animation tool we can see the two nodes exchanges data between each other. In the simulation we can see time consumed for transmitting data between hosts.

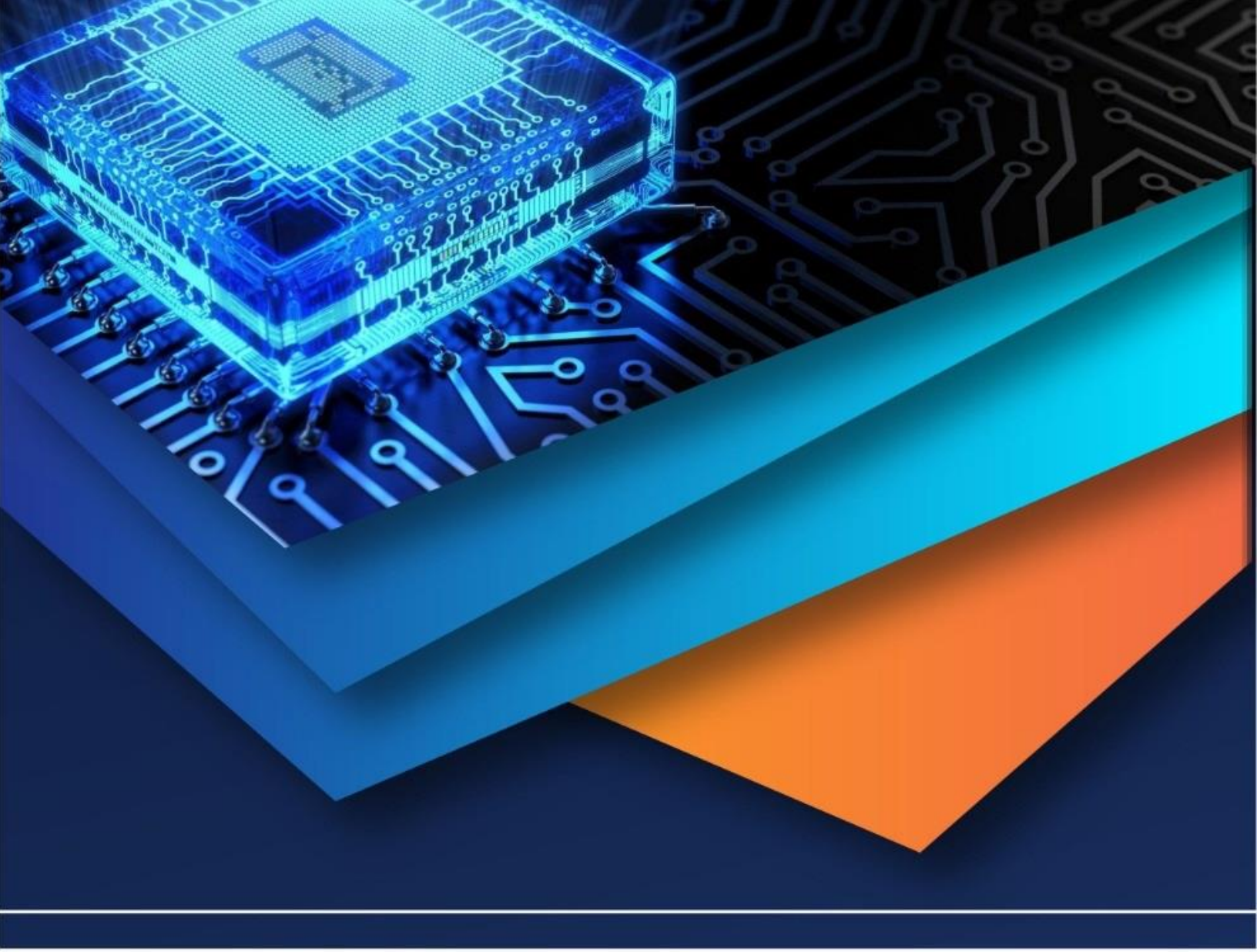
V. CONCLUSION

A. Simulation is a key component of network research
1) Debug ability
2) Reproducibility
3) Parameter exploration
4) No dependency on existing hardware/software
B. ns-3 has a strong focus on realism:
1) Makes models closer to the real world: easier to validate
2) Allows direct code execution: no model validation
3) Allows robust emulation for large-scale and mixed experiments
C. ns-3 also cares about good software engineering:
1) Single-language architecture is more robust in the long term
2) Open source community ensures long lifetime to the project

REFERENCES

[1] Ns-3 Tutorial Hands-On: Point-to-point and CSMA -Ethernet. 2011/ http://iwing.cpe.ku.ac.th
[2] http://nsnam.org/
[3] Mathieu Lacage. Experimentation with ns-3. 2009
[4] Tutorial: http://www.nsnam.org/docs/tutorial/tutorial.html
[5] Wiki: http://www.nsnam.org/wiki/index.php/Main_Page

©IJRASET: All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)