ns-3

NETWORK SIMULATOR

Protocols – AODV, DSDV

Wireless Communications and Mobile Computing Code: CSCI 256

Instructor: Dr. Ming Li

Semester: Spring 2019

Submitted By:

Monalisa Sha,

Abhijit Dubal,

Himanshu Gupta

Table of Contents

1. Introduction to NS-3:

- Ns-3 has been developed to provide an open, extensible network simulation platform, for networking research and education. In brief, ns-3 provides models of how packet data networks work and perform and provides a simulation engine for users to conduct simulation experiments. Some of the reasons to use ns-3 include to perform studies that are more difficult or not possible to perform with real systems, to study system behavior in a highly controlled, reproducible environment, and to learn about how networks work. Many simulation tools exist for network simulation studies. Below are a few distinguishing features of ns-3 in contrast to other tools.

- Ns-3 is designed as a set of libraries that can be combined together and also with other external software libraries. While some simulation platforms provide users with a single, integrated graphical user interface environment in which all tasks are carried out, ns-3 is more modular in this regard. Several external animators and data analysis and visualization tools can be used with ns-3. However, users should expect to work at the command line and with C++ and/or Python software development tools.

- Ns-3 is primarily used on Linux systems, although support exists for FreeBSD, Cygwin (for Windows), and native Windows Visual Studio support is in the process of being developed.

1.2 For NS-2 Users:

For those familiar with ns-2 (a popular tool that preceded ns-3), the most visible outward change when moving to ns-3 is the choice of scripting language. Programs in ns-2 are scripted in OTcl and results of simulations can be visualized using the Network Animator nam. It is not possible to run a simulation in ns-2 purely from C++ (i.e., as a main() program without any OTcl). Moreover, some components of ns-2 are written in C++ and others in OTcl. In ns-3, the simulator is written entirely in C++, with optional Python bindings. Simulation scripts can therefore be written in C++ or in Python. New animators and visualizers are available and under current development. Since ns-3 generates pcap packet trace files, other utilities can be used to analyze traces as well.

1.3 Downloading and Installing NS-3 using Bake

Bake is a tool for distributed integration and building, developed for the ns-3 project. Bake can be used to fetch development versions of the ns-3 software, and to download and build extensions to the base ns-3 distribution, such as the Direct Code Execution environment, Network Simulation Cradle, ability to create new Python bindings, and others.
You can get the most recent copy of bake by typing the following into your Linux shell (assuming you have installed Mercurial):

```
$ cd
$ mkdir workspace
$ cd workspace
$ hg clone http://code.nsnam.org/bake
```

After the clone command completes, you should have a directory called bake, the contents of which should look something like the following:

```
$ ls bake bakeconf.xml doc generate-binary.py
TODO bake.py examples test
```

After the step, you should change into the 'bake' directory, and then set the following environment variables

```
$ export BAKE_HOME=`pwd`
$ export PATH=$PATH:$BAKE_HOME:$BAKE_HOME/build/bin
$ export
PYTHONPATH=$PYTHONPATH:$BAKE_HOME:$BAKE_HOME/build/lib
```

After configuration of Bake, we have to use the following commands:
Step into the workspace directory and type the following into your shell:

```
$ ./bake.py configure -e ns-3.27
```

Next, we'll ask bake to check whether we have enough tools to download various components.
Type: `$ ./bake.py check`

You should see something like the following,

```
> Python - OK
> GNU C++ compiler – OK
> Mercurial - OK
> CVS - OK
> GIT - OK
> Bazaar – OK
> Tar tool - OK
> Unzip tool - OK
> Unrar tool - is missing
> 7z data compression utility - OK
> XZ data compression utility – OK
> Make - OK
> cMake - OK
> patch tool - OK
> autoreconf tool - OK
> Path searched for tools: /usr/lib64/qt-3.3/bin /usr/lib64/ccache /usr/local/bin /bin
/usr/bin /usr/local/sbin /usr/sbin /sbin bin
```

Next, try to download the software:

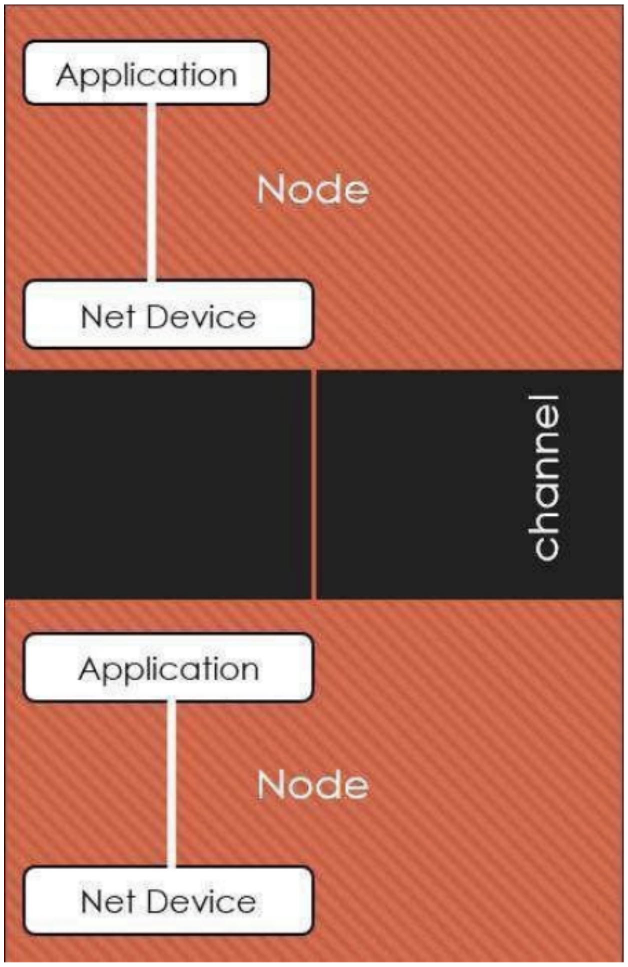
```
$ ./bake.py download
```

should yield something like:

```
>> Searching for system dependency setuptools - OK
>> Searching for system dependency pygoocanvas - OK
>> Searching for system dependency pygraphviz - OK
>> Searching for system dependency python-dev - OK
>> Searching for system dependency libxml2-dev - OK
>> Searching for system dependency clang-dev - OK
>> Downloading click-ns-3.25 - OK
>> Downloading BRITE - OK
>> Searching for system dependency qt4 - OK
>> Downloading nsc-0.5.3 - OK
>> Searching for system dependency g++ - OK
>> Downloading castxml - OK
>> Downloading openflow-ns-3.25 - OK
>> Downloading netanim-3.108 - OK
>> Downloading pygccxml-1.9.1 - OK
>> Downloading pygccxml - OK
>> Downloading pybindgen-ns3.27-castxml (target directory:pybindgen) - OK
>> Downloading ns-3.27 – OK
```

You are now ready to build the ns-3 distribution.

1.3 NS-3 components



1.3.1 Figure 1: Main Components of NS-3

- Node – denotes hosts/end systems.
- Application – Applications run on nodes.
- It consumes hosts resources
- Channel – media over which data flows.
- Large Ethernet Switch
- Three-dimensional space with obstruction.
- (wireless)
- NetDevice – A network interface attached to a node.
- Topology helpers – simplifies networks configuration process.

More on topology helpers:

- NodeContainer – create a set of nodes referred to be one reference.
- PointToPointHelper – Set channel and device attributes.
- NetDeviceContainer – holds network interfaces in one object.
- InternetStackHelper – To install an Internet stack (TCP/UDP/IP, etc) for a NodeContainer
- Ipv4AddressHelper – managing IP4 assignment
- NS3 has many components….you can search for them in the NS3 documentation website
- Under NS3 directory, go to src, all components are defined there.
- You can find a folder in each called examples, so that you have a good starting point to learn how to use a certain component.
- Topology helpers are also listed in each directory under a directory called helpers.

1.4 Compiling and Running in NS-3

- Execute ./waf command on top ns3 directory o Top directory, in my case, is ns-3
- ~/workspace/NS3Work/source/ns-3.26# ./waf
- ./waf goes over all projects under scratch and compiles them then o links them. Executables will be placed in the build directory.
- To run the scratch-simulator project(sample project in NS-3) we type o ./waf –run scratch-simulator

- If you use a C++ debugger like gdb or llvm, you can debug your NS3 projects.

- For GDB ./waf --run --commandtemplate "gdb -args %s”

- For LLVM ./waf --run --commandtemplate "lldb %s"

2. Introduction to DSDV, AODV

2.1 DSDV

- Each node maintains a routing table which stores.
- next hop, cost metric towards each destination.
- a sequence number that is created by the destination itself.
- Each node periodically forwards routing table to neighbors.
- Each node increments and appends its sequence number when sending its local routing table.
- greater sequence numbers are preferred.
- Each node advertises a monotonically increasing even sequence number for itself.

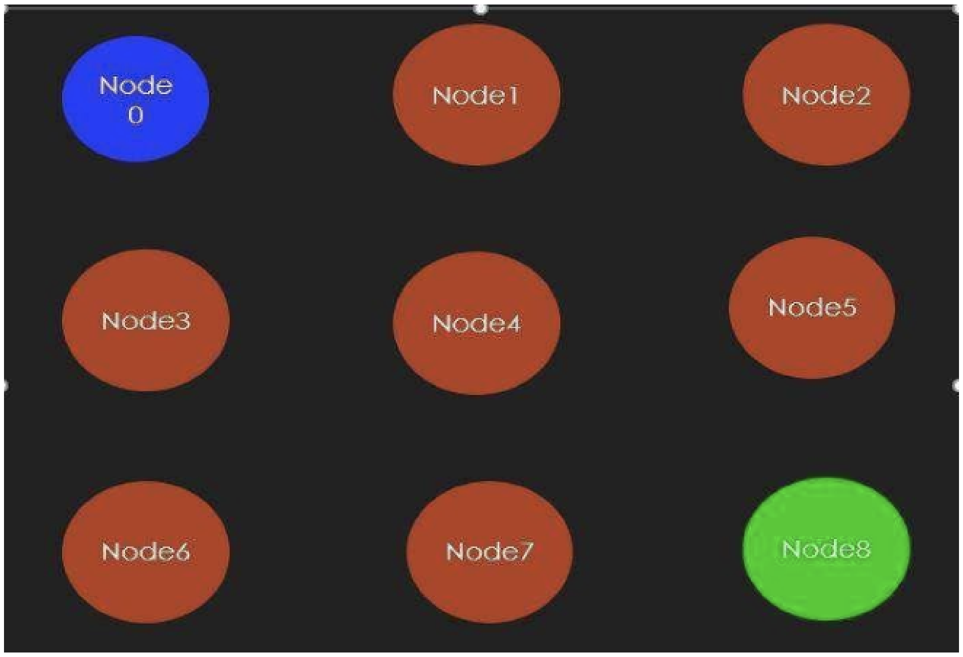
2.2 AODV

- Source floods route request in the network.
- Reverse paths are formed when a node hears a route request.
- Each node forwards the request only once (pure flooding).
- Uses hop-by-hop routing.
- Route reply forwarded via the reverse path.

3. Topology 1(Nodes without mobility)

- Nodes placed in Grid with mobility constant position, i.e. nodes are static, no movement at all
- Source: Node 0/10.0.0.1
- Destination: Node 8/10.0.0.9

- Total # Nodes: 9
- Tx Range: 100m
- Data Rate: 1 Mbps



3 Figure 2: Topology Structure(with no mobility)

3.1 Results of AODV, DSDV

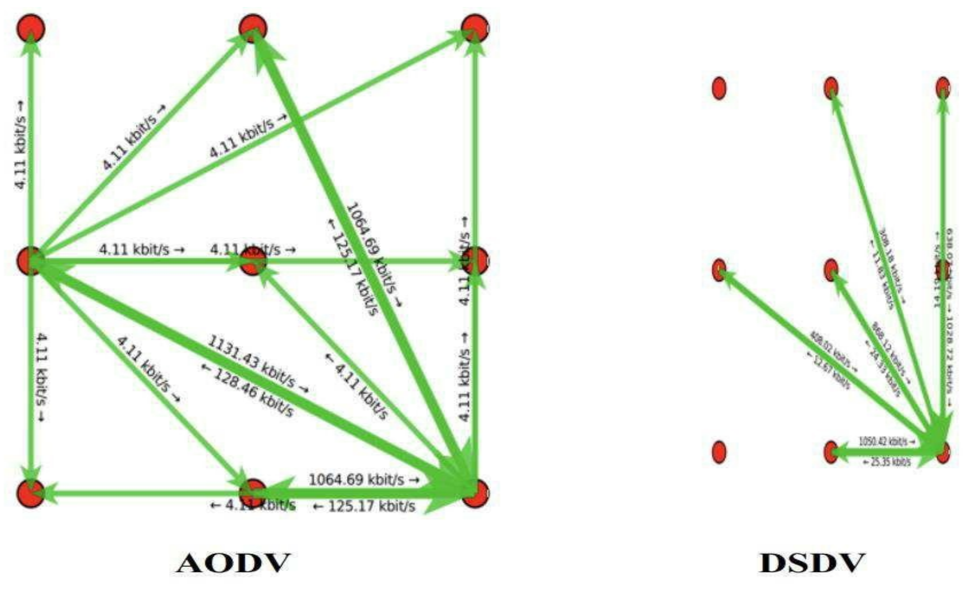
3.1.1 AODV

- Total Packets Sent: 59746
- Total Packets Received: 29037
- Total Packets Lost: 18076
- Throughput: 234.01kbps
- Packet Delivery Ratio: 69%

3.1.2 DSDV

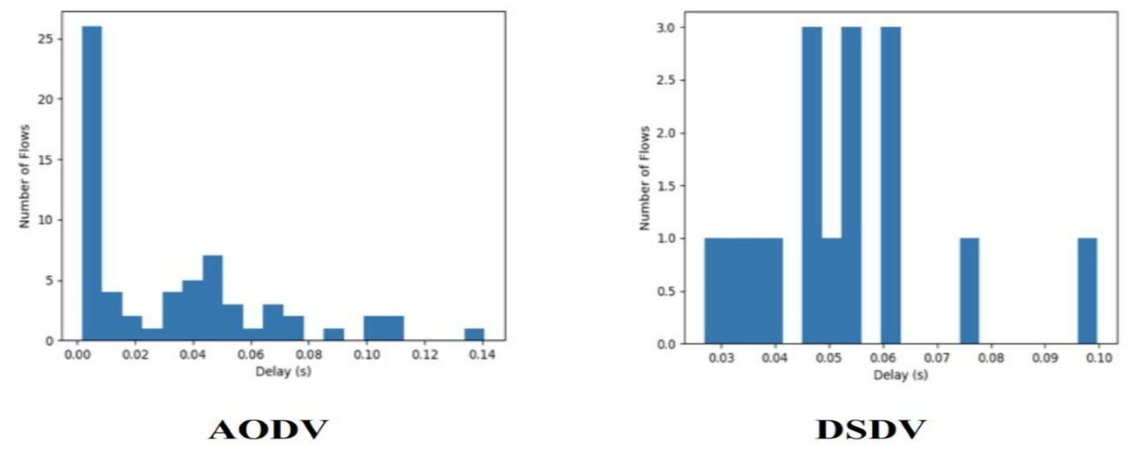
- Total Packets Sent: 58847
- Total Packets Received: 33462
- Total Packets Lost: 13980
- Throughput: 923.07kbps
- Packet Delivery Ratio: 76%

3.2 Simulation of AODV, DSDV



3.2.1 Figure 3: Simulation of AODV, DSDV (snapshots are different because of different time signatures)

3.3 Delay Stats of AODV, DSDV

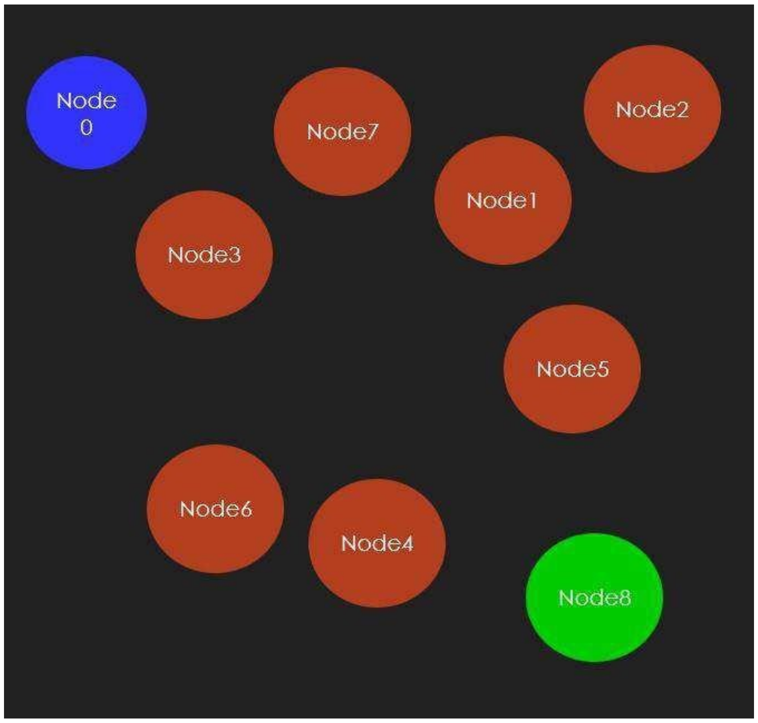


3.3.1 Figure 4: Delay Stats of AODV, DSDV

Above delay stats indicates that, AODV performs fast as it has the highest number of flows with minimum amount of delay, the other protocols like DSDV performance almost identical.

4. Topology 2(node with mobility)

- Nodes placed in Grid with motion, i.e. nodes are static, no movement at all
- Source: Node 0/10.0.0.1
- Destination: Node 8/10.0.0.9
- Total # Nodes: 9
- Tx Range: 100m
- Data Rate: 1 Mbps



4. Figure 5: Topology structure (with mobility)

4.1 Results of AODV, DSDV

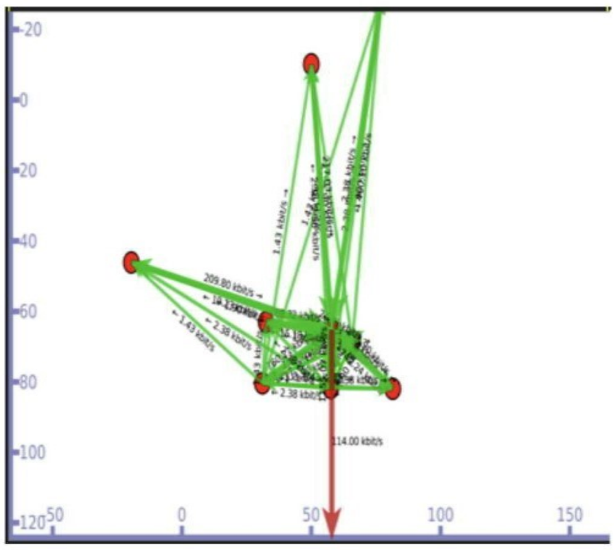
4.1.1AODV

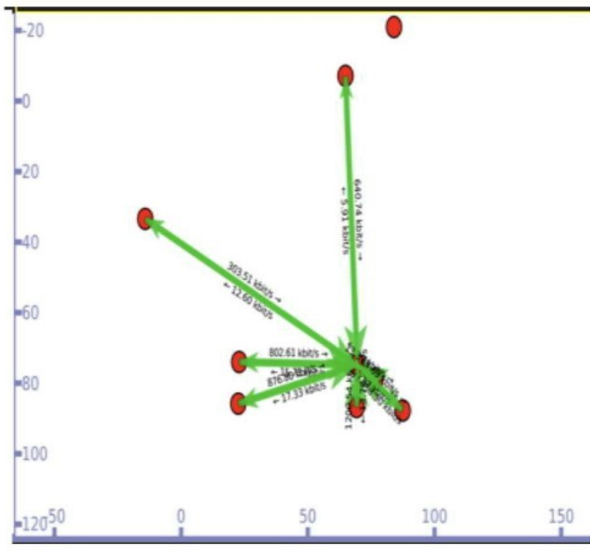
- Total Packets Sent: 66307
- Total Packets Received: 41500
- Total Packets Lost: 16300
- Throughput: 409.16kbps
- Packet Delivery Ratio: 75%

4.1.2DSDV

- Total Packets Sent: 65487
- Total Packets Received: 45098
- Total Packets Lost: 12337
- Throughput: 848.73kbps
- Packet Delivery Ratio: 81%

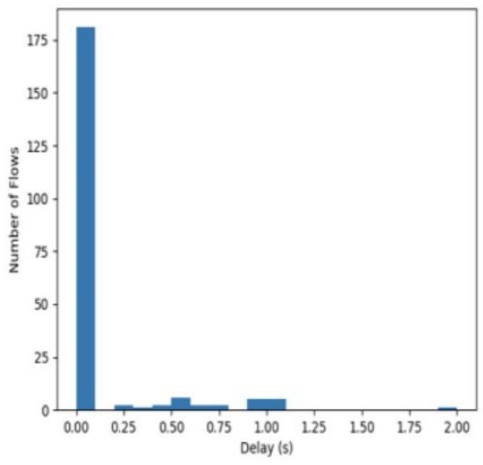
4.2 Simulation of AODV, DSDV

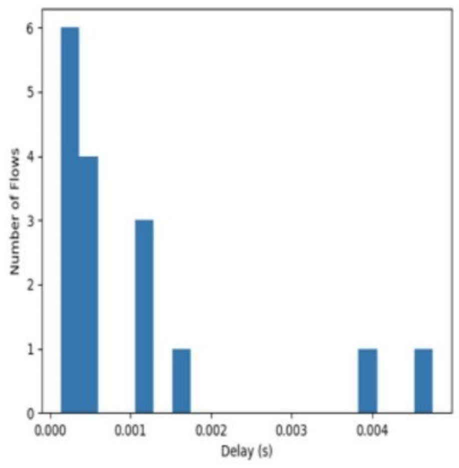




4.2.1 Figure 6: Simulation of AODV, DSDV(snapshots are different because of different time signatures)

4.3 Delay Stats of AODV, DSDV





4.3.1 Figure 7: Delay Stats of AODV, DSDV

Above delay stats indicates that, AODV performs fast as like as the previous topology. It maintains almost all the flows with minimum delay, whereas the other protocols like DSDV performance almost identical.

5. Analysis of the results

From above experiments, we can see that DSDV perform well in terms throughput in both scenarios in NS-3 and AODV does not perform well. AODV increases performance when the node mobility.
Network Simulator NS-3 development is still in progress and so, there might be some error in the results regarding AODV performance. Cause, conceptually AODV should have performed better because it removes the overhead of periodic broadcasting of DSDV. This might the case of code similarity of AODV and DSDV, cause they inherit the same base of code. Hence, there might be some overlap in methods of both protocols.

6.Summary

- NS-3 is highly customizable and programming friendly. You can write internal protocol code and simulation code in same languages (C++ or Python), which increases the robustness and stability of the code.

- Generally, as we can see from above comparisons, the routing protocol performance depends on some certain variables i.e. Nodes Count, Data-Rate, delay, nodes mobility model, distance etc.

- If there are too many nodes in range with motion, more collision and packet loss will happen, and data validity will be lower.

7.References:

- NS-3: https://www.nsnam.org/
- NS-3 Docs: https://www.nsnam.org/documentation/
- Introduction to NS-3: "*Konstantinos Katsaros k.katsaros @ surrey.ac.uk*"
- *NS3 Simulator,Installation & Overview By Adil Alsuhaim*

- Protocol Refs:
- "*Mobile Ad Hoc Networks Tutorial- Sridhar Iyer, IIT Bombay*"
- "*Routing Protocols in Ad-hoc Networks, Self-configuring systems (SCS) TTM3 – Host 2004, Jorn Andre Berntzen*"