International Journal of Engineering & Technology, 7 (3.4) (2018) 17-23

SPC

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper



Shell Script to Clone AODV Routing Protocol in Network Simulator-2

Roshan Ulla Khan, Vijayalakshmi A. PhD

Research Scholar, Department of Computer Science.
CHRIST (Deemed to be University), Bengaluru, India
Assistan\t Professor, Department of Computer Science,
CHRIST (Deemed to be University), Bengaluru, India,
**Corresponding author E-mail: vijayalakshmi.nair@christuniversity.in*

Abstract

Background and Objective: Most of the research that are carried in ad hoc routing protocol is through simulation. While working with a simulator, the codes are enclosed in a component that is accessible to all the developers. The difficulty arises as there is no enough documentation and users find it difficult to modify different C++ and TCL files. Even if one component is modified then the entire Network Simulator-2 (NS-2) suite must be reconfigured. Cloning the protocol manually takes a lot of time and prone to error. Our objective is to ease the work of developers and researchers by showing the procedure to clone the AODV protocol automatically using a script. **Methodology:** In this study, a shell script is developed that will clone the AODV protocol by modifying 18 C++ and TCL files of the protocol and NS-2 suite by automatically inserting the code in exact files at exact position. It also configures the NS-2 and installs the entire NS-2 suite along with setting the path in .bash files. **Results:** In this research work, a comparison of cloned protocol with AODV protocol is done based on throughput time and packet loss metrics and the results generated are exactly same for both the protocols. The results of the study reveal that the proposed script clones the AODV protocol successfully. **Conclusion:** This work proves that the proposed script can clone the AODV protocol faster with just one execution of shell script. This methodology will save the time and help the developers or research to focus more on their study on the protocol.

Keywords: AODV Routing Protocol, Network Simulator-2, Ad-hoc Protocols, Access Protocol, Wireless Networks, Clone AODV

1. Introduction

The need for routing protocol in networking is to find a path for transferring the data packets from source to destination. AODV is an On-Demand routing protocol which supports unicast and multicast packet transmission even in dynamic node movement [1].
AODV protocol locates new routes only when it is necessary. The nodes information is stored in a routing table for both unicast routes and multicast routes separately. These nodes information are preserved only if needed. AODV protocol doesn't support the unidirectional links and it expects that nodes in the channel detects each other's relay. And the fixed interval beaconing leads to unnecessary bandwidth consumption [2]. AODV protocol is also vulnerable to various attacks as it is based on premise that all nodes should work together and if nodes are not working together then establishing a route is not possible. These vulnerability of AODV protocol can be reduced, by making necessary changes in its code.

Making changes to AODV protocol is a difficult task. Even if single change is made then related code files of simulator must be modified if necessary. This comes as a hardship for developers or researchers who wants to work on the protocol improvisation. To reflect the changes in protocol execution the NS-2 suite should be reconfigured. After reinstalling if the protocol does not generate results as expected, then the NS-2 suite must be reinstalled (as a new package) from the beginning. This is a tedious work. To avoid these issues a protocol clone can be made and make the necessary changes to cloned protocol. In this case even if cloned protocol does not work as expected, still the original protocol will be working good. But cloning protocol manually requires a person to know all the modification to be made to 18 C++ and TCL files. To ease the work of cloning, a shell script is developed in this work, which will automatically clone the protocol from creating folders, making modification to files (C++, TCL), configuring NS-2 and making changes to path files.

Network Simulator-2 is an individual event simulator [3]. Which can be used to improvise the performance of the protocol by making modification and simulating the scenarios [4]. It also provides implementation of networking nodes, establishing link between transport protocols, routing protocols and traffic generators. NS-2 provides the facilities to design a networking scenario to test the networking cases, by modifying the protocols to improvise performance. But modifying the protocol in NS-2 is prone to errors, if any of the changes are made to any protocol files then the network simulator package should be configured to create the object files through Makefile. To avoid these errors, existing protocol can be cloned to make the changes in it. Creating a clone of the protocol will ease work of the developers or researchers. So that they can experiment their work on cloned protocol and even if errors are occurred still they will have the original protocol and reinstalling of network simulator packages can be avoided.

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The process of modifying AODV protocol includes modification to 18 C++ and TCL files, these individual files modification have been discussed in later section. If the cloning is done manually then one should have the knowledge of what code to be included in those files. In the manual process of cloning it is important to completely understand every file code, to include the modifications and it is time consuming. To ease the work, a shell script is written which will automatically clone the AODV routing protocol just by running it in command prompt. This script will clone the AODV protocol and the cloned protocol works exactly as AODV protocol with no performance alteration or any changes in it. Thus, allowing the researchers or developers to experiment their ideas of improvement in this protocol.

2. Background

A thorough literature review was conducted to understand the need of cloning the AODV routing protocol. There are areas in which the AODV routing protocol need to be improvised. For which many papers are reviewed which represents the improvisation and showcases the areas of improvisation in AODV routing protocol

Black Hole attacks in AODV protocol: AODV routing protocol is a reactive protocol means whenever a node wants to send data packet to another node, it searches for different route to find out the shortest path. In AODV black hole attack, a malicious node sends the wrong information showing the shortest path to the destination [5]. When a node transfers data through this node either it drops the packet or it further transfers it to other malicious nodes. When a source node starts route discovery it uses route request packet (RREQ), this RREG packet is sent to all its neighboring nodes which consist of following attributes <source address, source sequence number, broadcast_id, destination address, destination sequence number, hop count>. The source address is an originator address, source sequence number works as a time stamp, broadcast_id is an identification number for RREG packet, destination address gives the information to the mediator nodes that source is looking for this destination address, destination sequence number is kept empty initially until it finds out the destination sequence number, hop count records the distance from source to destination. In Fig 1 node A initially consist of following information <A,1,1,E, ,0> then it broadcasts RREG packet to its neighbor nodes C and M to find node E.

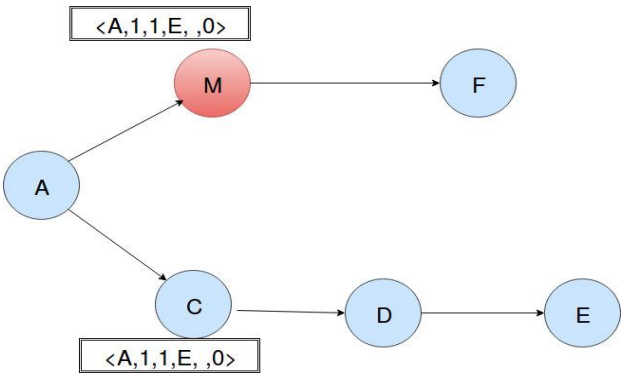


Fig. 1: RREQ from node A to node C and M

If C and M nodes having any route to reach node E they will reply to A or else they will broadcast the request further. But in this network, node M is considered as malicious node. Even though node M is not connected to destination node it will create a fake node reply packet and this route reply packet will indicate that to send any data packet to node E follow node M and same fake route packet is sent to source node A as shown in Fig 2.

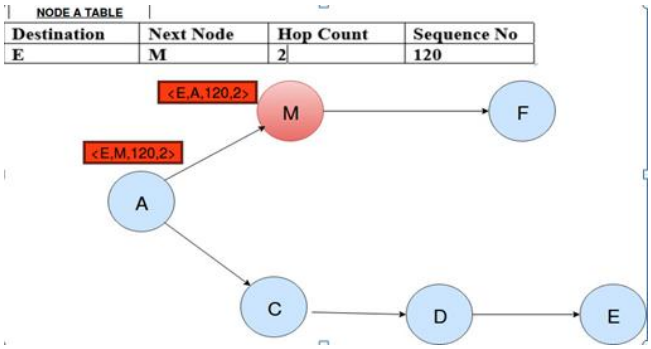


Fig. 2: node A table entry of path to reach node E

Rahul Singh et.al [5] have discussed about a solution to avoid black hole problem. They have discussed that by removing the intermediate node to send the reply let only the destination address send the reply to the source. Still it leads to two problems one is that it will increase the waiting time for source to establish the route to pass the data packet and the other problem is what if the malicious node acts as a destination node and replies.

They have also discussed about the solution in which the source node waits for RREG from more than two intermediate nodes if the attributes of routing are matching then data packets are sent. The problem with this method is that it creates a delay and it also takes more time to examine the reply sent by the destination nodes.

Survey on AODV and DSR (Dynamic Source Routing) routing protocols: Charu Aharma et.al [6] have done a survey by comparing the on-demand AODV and DSR routing protocol. The survey results show that the AODV protocol has more routing overhead than the DSR routing protocol. Reason is DSR does not maintain any recurrent internal route maintenance messages but in AODV route discovery process is chronic and it also consumes more battery power. The AODV routing protocol has normalized media access control (MAC) overhead and it is based on the combination of DSDV and DSR routing protocols. Its performance is superior in high mobility, it is more controlled and always selects the fresher route. By comparing these features, they have concluded that both the protocols are good based on the type of parameters provided and scenario in which they are implemented [6].

Energy Efficiency: Reena Singh et.al [7] have designed an Energy Efficient AODV routing algorithm by improvising RREQ (Routing Request) and RREP (Routing Reply) handling processing order to save energy in wireless mobile devices. They have considered a minimum threshold energy for the intermediate nodes, which will be used in data packet transferring. If the energy of the intermediate nodes is less than minimum energy level then another route is opted in which intermediate nodes have more energy than minimum threshold energy level, until or unless no other path is available.

Protocol implementation and Network Simulation using NS-2: Keijo Harju et.al [8] have discussed about NS-2 architecture as shown in Fig 3 and the usage of NS-2 to test or improvise the protocols. Network Simulator 2's architecture is designed using C++/Otcl which is based on object oriented concept. Which provides detailed protocol functionality through separation of control between background code and the TCL script which attain scalability and extensibility in its operation. Otcl is used for controlling of networking simulations in which probability of modifying sim-

ulation is faster. For the users to work along with these concepts

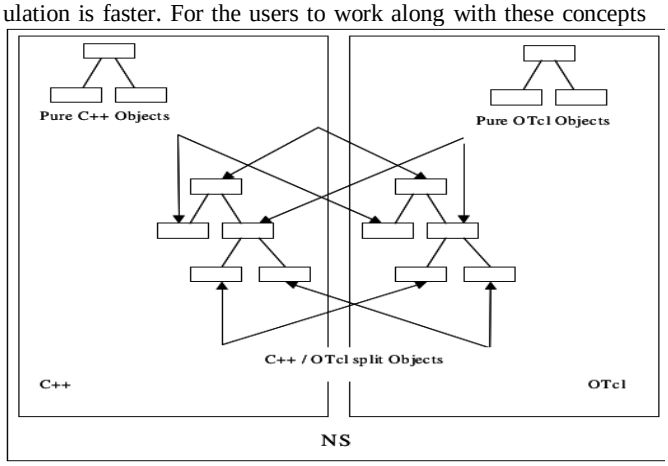


Fig. 3: General architecture [8]

they should be knowing C++ and Otcl**.**
The C++ and Otcl objects are present in the separate files and these will be linked together using binding function.

The modification of C++ files can be done through Otcl directly and the operations to perform are carried out through functions. This procedure of binding is shown in Fig 4.

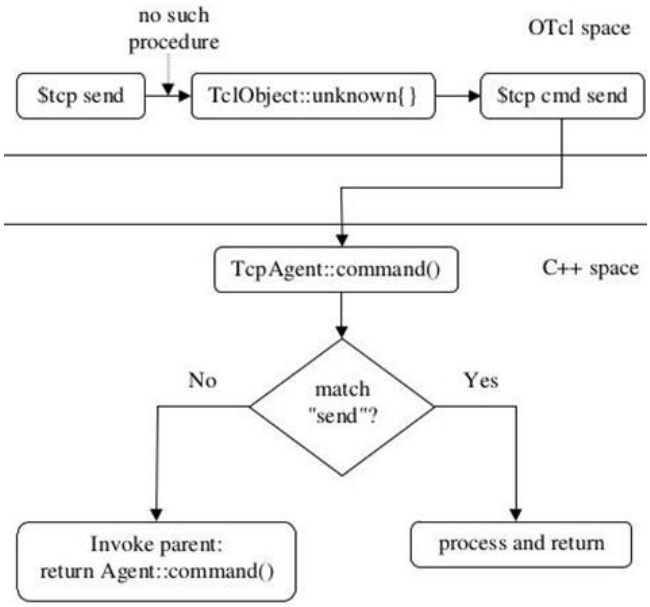


Fig. 4: Procedure for interpretation of command from OTcl to C++ [8]

3. Network Simulator 2

Network Simulator 2 is an open-source event driven simulator which can be used in academic research for simulating Computer networks, MANETs (Mobile Ad Hoc Network), WSNs (Wireless Sensor Networks). Network simulator is a software program emulates the working of computer networks by performing the calculation between network resources like routers, packets, hosts and servers using formulas [9]. The generated results using simulation are analyzed and based on the analysis, users can customize the simulation to obtain their results. It supports protocols like IPv4, IPv6, UDP and TCP [9].

NS-2 is two language architectures that is C++ and Otcl. To execute C++ code, it should have gcc compiler in Linux system, it is fast to run on a machine codes and the changes to make is difficult [10]. Otcl script can be executed using the interpreter and it takes more time to execute, compared to that of C++ but the modification can be done easily. So, C++ is a compiled hierarchy and Otcl is an interpreted hierarchy [10]. If both the languages have its own features then why two programming languages are needed? That is because even if any minor changes are made in C++ it needs to be recompiled and the files to run a protocol are many and it's not an easy task. NS-2 has its own coding style as shown in Fig 5. It keeps the input parameters in a configuration file. Then there is no need to change C++ code. This configuration file is called TCL simulation script.

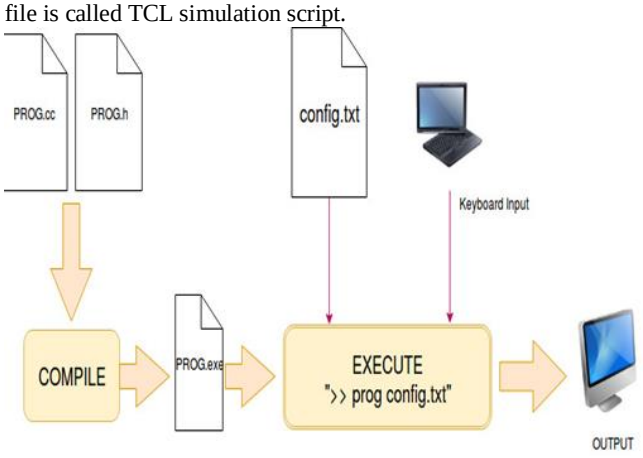


Fig. 5: C++ coding with configuration file

Tool Command Language: Tool Command Language (TCL) is a dynamic open source programming language for developing web and desktop applications, on the operating systems like Linux, Windows, Mac OS x [11]. Reading the configuration from C++ files is difficult so TCL script is used, C++ contains classes where has TCL is not an object-oriented programming. To implement the components in C++ is complicated, so it can be defined in Otcl file. In general C++ acts as an internal machine and TCL as the user interface.

4.Implementation

AODV protocol is an important routing protocol and the improvisation of this protocol can be done in fields like Energy Efficiency, Black Hole Problem, Reducing the complexity in finding the shortest path and many more. These improvisations can be done using the Network Simulator tools. The NS-2 architecture is based on two programming languages C++ and Otcl. C++ is a background language, even if a small change is made then the code must be compiled to incorporate the changes in object file and it is time consuming and prone to errors. To avoid these issues the protocol can be cloned and make the necessary changes to. But manually cloning AODV protocol is a difficult process as it required to modify 18 C++ and TCL files. If even one small code is missing then the cloning will not work. So, in this research work a shell script is developed which will automatically clone the AODV protocol. Using this script, any number of AODV protocol clones can be created to make the necessary changes and analyze or improvise the protocol.

Shell Script Procedure to Clone AODV protocol: In this script the cloned protocol is named as CAODV (Cloned Ad Hoc On-Demand Distance Vector) and ns-allinone-2.35 version is used to clone the protocol.

STEP 1: Copying and renaming the aodv files to caodv

Move control inside ns-2.35 folder and copy aodv folder to caodv

$cp -r aodv caodv
Rename all files inside caodv by prefixing "c". For instance

$mv aodv.cc caodv

STEP 2: Modify /common/packet.h file (using stream editor command)

-Insert a new packet type

$sed -i '/PT_MDART=72;/a static const packet_t PT_CAODV=73; packet.h

-Declare cloned protocol type with static constant inside class p_info { }

$sed -i '/type == PT_AODV/a type == PT_CAODV ||' packet.h

-Initialize the packet type inside static void initName() function.

$sed -i '/name_\[PT_AODV\]= "AODV";/a \ name_[PT_CAODV]= "CAODV";' packet.h

STEP 3: Modification to calculate trace values in cmu-trace.h file and cmu-trace.cc file s

- Declare cloned protocol function to receive packets and offset values in class CMUTrace: public Trace {} class inside cmu-trace.h file.

$sed -i '/format_aodv/a \ void format_caodv(Packet *p, int offset);' cmu-trace.h

- Declare cloned protocol header to access the format_caodv() function in cmu-trace.cc file

$sed -i '/diff_header.h/a #include <caodv/caodv_packet.h>' cmu-trace.cc

-Define CMUTrace::format_caodv() function.

The definition of format_coadv() function is stored inside a text file (cmu_trace.txt) , through script it is inserted inside the cmu-trace.cc file

$sed -i '/double CMUTrace::duration_scaling_factor_ = 0.0;/r cmu_trace.txt' cmu-trace.cc

- Define case condition for cloned protocol inside CMUTrace::fomat().

$sed -i '/case PT_TORA:/i case PT_CAODV: \n\ format_caodv(p, offset); \n break;' cmu-trace.cc

STEP 4: Declare the Clone routing protocol, set up the routing agent for cloned AODV.

-Declare cloned protocol network type in set protolist{} class inside ns-packet.tcl file

$sed -i '/AODV \# routing/a CAODV ' ns-packet.tcl

-Setting up the routing agent inside ns-lib.tcl file

$sed -i '/AOMDV {/i CAODV { \n set ragent [$self create caodv-agent $node] \n }' ns-lib.tcl

-Setting up cloned protocol agents inside \tcl\lib\ns-agent.tcl
The code to set up the agent is stored inside a text file named ns-agent.txt

$sed -i '/Agent\/AODV set dport_ 0/r ns-agent.txt' ns-agent.tcl

-Setting up the special processing for cloned protocol inside \tcl\lib\ns-mobilenode.tcl file

$sed -i '/$agent set sport_ $port/r ns-mobilenode.txt' ns-mobilenode.tcl

STEP 5: Modify \queue\priqueue.cc file to store cloned protocol packet type in a queue under PriQueue::recv() funtion.

$sed -i '/case PT_AOMDV:/i case PT_CAODV:' priqueue.cc

STEP 6: Define a statement inside \ns-2.35\makefile, \ns-2.35\Makefile.in, \ns-2.35\makefile.vc to create object files for the cloned protocol, routing table, queuing and for logs.

-The code to modify these make files is stored inside a text file named makefile.txt

$sed -i '/aodv\/aodv_rqueue.o/ r makefile.txt' Makefile

STEP 7: Declare Friend class in routing table file

-Declare Freind class for CAODV and CAODVLocalRepairTimer class in class rt_entry{} class of \routing\rttable.h

$sed -i '/friend class rttable;/a friend class CAODV; \n friend class CAODVLocalRepairTimer;' rttable.h

STEP 8: Define strcmp() function for cloned protocol same as AODV protocol in \wpan\p802_15_4nam.cc

The code is stored in a text file named wpan.txt to modify the p802_15_4nam.cc file.

$sed -i '/PT_AODV:/r wpan.txt' p802_15_4nam.cc

STEP 9: Define packet header management in \ dei80211mr-1.1.4\src\InitTCL.cc file.

$sed -i '/\/AODV/r inittcl.txt' InitTcl.cc

STEP 10: Rename all aodv files to caodv and AODV to CAODV inside caodv folder.

$sed -i -e 's/aodv/caodv/g' ~/ns-allinone-2.35/ns-2.35/caodv/*

$sed -i -e 's/AODV/CAODV/g' ~/ns-allinone-2.35/ns-2.35/caodv/*

STEP 11: Rename the timer classes in caodv.cc, caodv.h and caodv_rtable.h files

-Code to change BroadcastTimer

$sed -i -e 's/BroadcastTimer/CAODVBroadcastTimer/g' /caodv/caodv.h

Make the same modification to HelloTimer, NeighboirTimer, RouteCacheTimer and LocalRepairTimer classes.
STEP 12: Recompilation

Recompile packet.cc since packet.h is modified
$touch ~/ns-allinone-2.35/ns-2.35/common/packet.cc

STEP 13: ConFig the ns-2.35 package

$ ./config
if configuration fails then follow step 14 or else stop

STEP 14: Install ns2.35 package and run the make commands

$ ./install
$cd ns-2.35
$sudo make install

STEP 15: Save the script inside ns-allinone2.35 folder and RUN the script in command prompt.

If the output is same as Fig 6 then the protocol is successfully cloned. It will show a message "Ns-allinone package has been installed successfully".

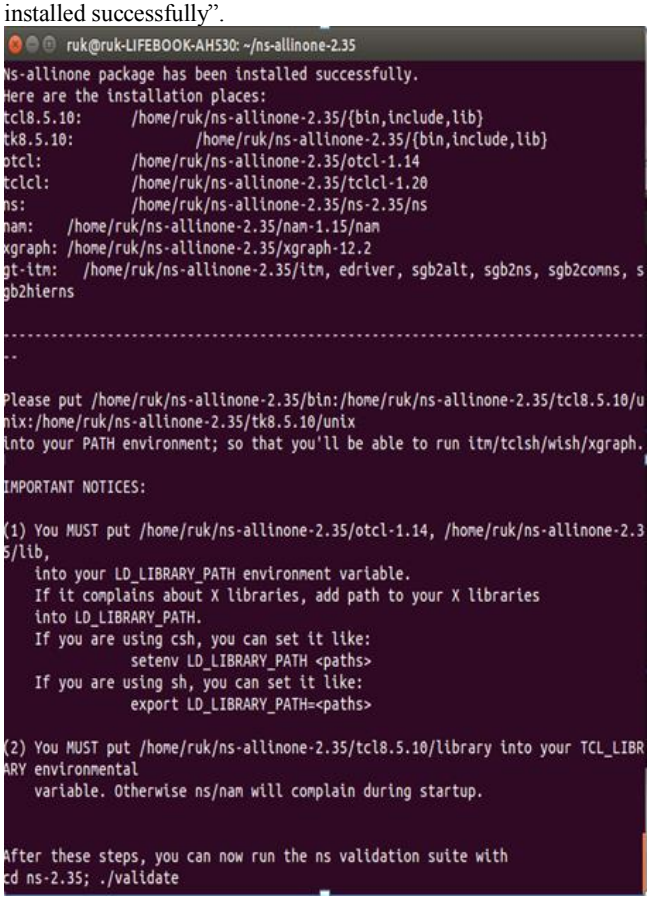


Fig 6. Cloned AODV protocol installation success prompt

5.Results and Discussion

To test the working of CAODV protocol same as AODV, a TCL script is written and NS-2 latest version 2.35 is used to test the results.

Scenario

- The total topology area (M*M) of 500*500 is taken to test.
- Omni antenna model is used for broadcasting.
- Nodes' are specified manually at the initial state itself.
- Nodes' position is set manually.
- The nodes' movement is random and speed is constant.
- Two-Ray ground radio propagation model is used.

Simulation Parameters: The simulation parameter is set up in TCL file. These parameters depict the type of networking resources that will be used in this simulation. Table 1 shows all the parameters and values.

Table 1: Simulation Parameters

Parameter	Value
Simulator	NS-2 (version 2.35)
Channel Type	Channel/WirelessChannel
Radio-propagation model	Propagation/TwoRayGround
Network interface type	Phy/WirelessPhy
MAC type	Mac/802_11
Interface queue Type	Queue/DropTail/PriQueue
Link Layer Type	LL
Antenna	Antenna/OmniAntenna
Maximum packet in ifq	50
Area (M*M)	500*500
Number of mobile nodes	8
Source Type	UDP
Simulation Time	80.01 Seconds
Routing Protocols	AODV, CAODV

The objective of this work is to clone the AODV routing protocol using shell script, which will work same as AODV protocol in NS-2. In this paper the complete procedure is described which is followed to clone the protocol using shell script. Which is the faster, easy and time saving method of cloning the routing protocol. After cloning the protocol, a TCL script is written with parameters mentioned in Table 1. In the TCL script for routing protocol parameter first AODV protocol is declared and the trace file results are represented in excel file and the graphs for the same trace file data are generated. Then it is tested with the CAODV protocol as routing protocol parameter which is cloned using the shell script. The results of CAODV protocol is same as AODV protocol resulting that the cloning is successful. The performance of CAODV protocol is tested and compared with AODV protocol in two areas.

Four trace files are created for throughput time and packet loss for total 80.00 seconds simulation time. The results obtained for AODV and Cloned AODV protocol are shown in Fig 7 and Fig 8 for throughput time, Fig 9 and Fig 10 for Packet loss.

Comparison of Performance Metrics between AODV and Cloned AODV Protocol: The comparison scenario consists of 8 mobile nodes: consisting of 4 source nodes and 4 destination nodes. The size of a transmitting packet is 512 bytes. Transmission rate of a node is 600 kbps for calculating throughput, 200 kbps for packet loss. It is assumed that the nodes are in transmission range at a constant distance of 195m and the simulation time lasted for 80.00 seconds.

The Fig 7 and Fig 8 shows the comparison of throughput time between AODV and CAODV (Cloned AODV) protocols. Throughput is calculated based on bits per second using a formula represented as Throughput = (Total number of packets delivered * Each packet size) / Simulation duration. In this work 8 nodes are used and the simulation is running for total of 80 seconds.

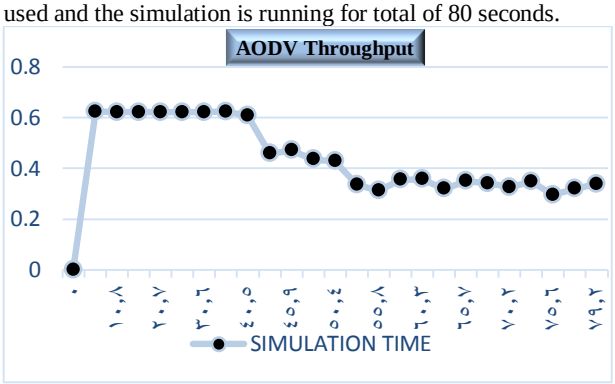


Fig. 7: Throughput time results of AODV

As shown in the Fig 8 it has generated the same results as the AODV protocol results are shown in Fig 7. Hence depicting that the Cloned AODV protocol works same as AODV protocol.

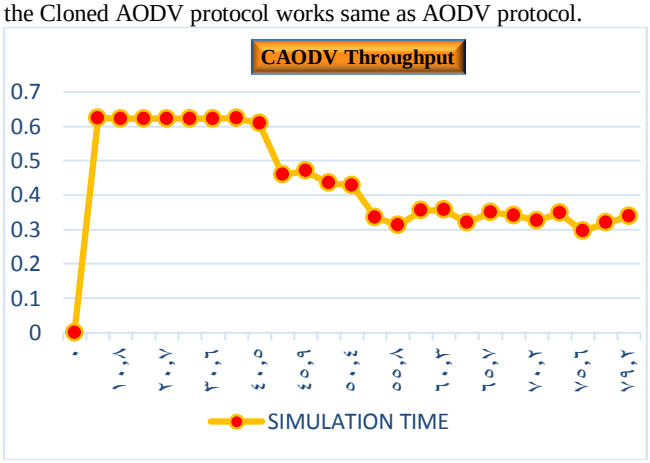


Fig. 8: Throughput time results of CAODV

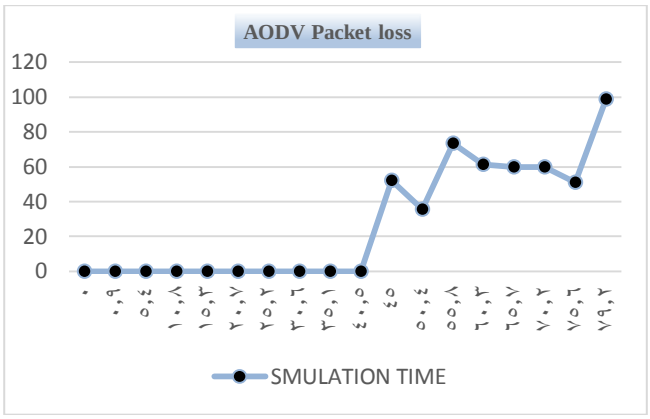


Fig. 9: Packet loss results of AODV

The packet loss metrics is tested between the AODV and Cloned AODV protocol. Fig 9 and Fig 10 indicates throughput relationship with respect to same queue length calculated for same node for 802.11b MAC type. As shown in Fig 9 and Fig 10 graphs, that the packet loss is exactly same for both the protocols. Indicating that Cloned AODV protocol works exactly same as AODV protocol without any changes.

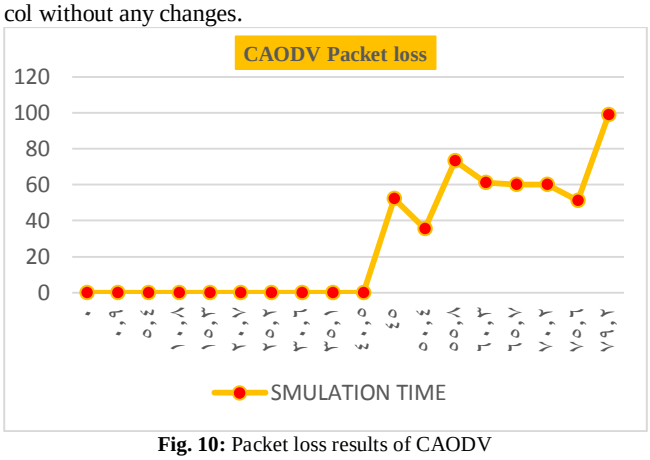


Fig. 10: Packet loss results of CAODV

The results obtained depicts that Cloned AODV protocol works exactly as AODV protocol. Thus, the shell script to clone AODV protocol can be used by researchers or developers to improvise the AODV protocol in NS-2.

6.Conclusion

AODV protocol has many areas of improvisation in it. Improvising the AODV protocol performance is difficult in NS-2. Even if a single C++ code is modified then entire network simulator package must be reconfigured. If the modification or configuration fails then the NS-2 package must be reinstalled. This can be avoided using a cloned AODV protocol. To Manually clone AODV protocol total 18 C++ and TCL files should be modified and remembering the code and the files to be modified is difficult. In this paper a procedure is designed to clone the AODV protocol using shell script. Hence making it easier and faster to clone the AODV protocol.

The cloning of the AODV protocol has given an idea to clone all other protocols. In our future work same concepts of cloning can be used to clone all other reactive routing protocols.

Acknowledgement

First and foremost, praises and thanks to the God, the Almighty, for His showers of blessings throughout my research work to complete the research successfully.

I would like to thank my research guide, Prof. Vijayalakshmi A, for the patient guidance, encouragement and advice she has provided throughout my time as her student. I have been extremely lucky to have a guide who cared so much about my work, and who responded to my questions and queries so promptly.

I thank the management of Computer Science Department, Christ University, India for their support to do this work. I thank Head of our department Prof. Joy Paulose for his genuine support to complete this research work successfully.

Study Conducted Period

This research study is conducted as part of curriculum of MCA programme at Christ University from May 2016 to January 2018.

Authors Contribution

Mr. Roshan Ulla Khan contributed to the design and implementation of the research. Prof Vijayalakshmi A and Roshan Ulla Khan carried out the experiment and wrote the manuscript.

Conflict of Interest

I Roshan Ulla Khan the corresponding author and Prof, Vijayalakshmi A the Co-author of this research paper don't have any conflicts regards to this research manuscript.

Significance Statement

This study proposes a technique which will clone AODV routing protocol Automatically through a shell script. It will help the researchers and developers to clone the AODV protocol with just executing one shell script rather than modifying 18 plus files manually. Thus, it will save the time and help researchers and developers to focus more on their protocol improvements by creating the clone automatically in NS-2.

References

[1] Anon, (2017). [online] Available at: http://shodhganga.inflibnet.ac.in/bitstream/10603/30965/7/07_chapter%202.pdf [Accessed 27 Oct. 2017].

[2] Selvarani, A. and Selvam, P. (2011). Study of Routing Protocols in Mobile Ad Hoc Networks. International Journal of Computer Ap-

plications. [online] Available at: http://www.ijcaonline.org/icvci/number4/icvci1234.pdf [Accessed 4 Jun. 2017].
[3] Anon, (2017). [online] Available at: http://www.linuxjournal.com/article/5929 [Accessed 3 Apr. 2017].
[4] Engpaper.com. (2017). wireless networking using ns2. [online] Available at: http://www.engpaper.com/wireless-networking-using-ns2.htm [Accessed 30 Nov. 2016].
[5] singh, R. and Uphdaya, A. (2016). A Literature Review on Black Hole Attacks on AODV Protocol in MANET. International Journal of Engineering Trends and Technology, 38(1), pp.44-49.
[6] Sharma, C. and kaur, H. (2015). Literature Survey of AODV and DSR Reactive Routing Protocols. International Journal of Computer Applications, (0975 – 8887).
[7] Singh, R. and Gupta, S. (2014). EE-AODV: Energy Efficient AODV routing protocol by Optimizing route selection process. International Journal of Research in Computer and ISSN (Online) 2278- 5841 Communication Technology, 3(1).
[8] Harju, K. and Korventausta, S. (2001). Network Simulation and Protocol Implementation Using Network Simulator 2. [online] Available at: http://www.cs.tut.fi/~kucherya/ns2/ns2over.pdf [Accessed 3 Apr. 2017].
[9] Network Simulator - ns2. (2017). Network Simulator - ns2. [online] Available at: https://ns2tutor.weebly.com/ [Accessed 14 Sep. 2016].
[10] Hossain, E. and Issariyakul, T. (2008). Introduction to Network Simulator NS2. US: Springer, p.512.
[11] [Whatis.techtarget.com. (2017). What is TCL? What Opens a TCL? File Format List from WhatIs.com. [online] Available at: http://whatis.techtarget.com/fileformat/TCL-Script-in-the-TCL-TK-Language [Accessed 2 Mar. 2017].