Assignment No. B_6

Title: To understand network simulator NS2 .

OBJECTIVES:

1.To understand network simulator NS2 .

Problem Statement: Use network simulator NS2 to implement:

a. Monitoring traffic for the given topology

b. Analysis of CSMA and Ethernet protocols

c. Network Routing: Shortest path routing, AODV.

d. Analysis of congestion control (TCP and UDP).

Theory:

Network Simulator:

"ns-2" redirects here. For the fictional robot from the works of Isaac Asimov, see NS-2 (literary character). For the hepatitis C virus protein, see NS2 (HCV). For the video game, see Natural Selection 2.

"ns-3" redirects here. For the hepatitis C virus protein, see NS3 (HCV).For other uses of "NS", see NS (disambiguation).

ns (from **network simulator**) is a name for a series of discrete event network simulators, specifically **ns-1**, **ns-2** and **ns-3**. All of them are discrete-event computer network simulators, primarily used in research[4] and teaching. ns-3 is free software, publicly available under the GNU GPLv2 license for research, development, and use.

The goal of the ns-3 project is to create an open simulation environment for computer networking research that will be preferred inside the research community:[*citation needed*]

- It should be aligned with the simulation needs of modern networking research.
- It should encourage community contribution, peer review, and validation of the software.

Since the process of creation of a network simulator that contains a sufficient number of high-quality validated, tested and maintained models requires a lot of work, ns-3 project spreads this workload over a large community of users and developers.

History

ns-1

The first version of ns, known as ns-1, was developed at Lawrence Berkeley National Laboratory (LBNL) in the 1995-97 timeframe by Steve McCanne, Sally Floyd, Kevin Fall, and other contributors. This was known as the LBNL Network Simulator, and derived in 1989 from an earlier simulator known as REAL by S. Keshav.

ns-2

Ns-2 began as a revision of ns-1. In 1995 ns development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. In 2000, ns-2 development was support through DARPA with SAMAN and through NSF with CONSER, both at USC/ISI, in collaboration with other researchers including ACIRI.

Ns-2 incorporates substantial contributions from third parties, including wireless code from the UCB Daedelus and CMU Monarch projects and Sun Microsystems. For documentation on recent changes, see the version 2 change log.

ns-3

In 2006, a team led by Tom Henderson, George Riley, Sally Floyd, and Sumit Roy, applied for and received funding from the U.S. National Science Foundation (NSF) to build a replacement for ns-2, called ns-3. This team collaborated with the Planete project of INRIA at Sophia Antipolis, with Mathieu Lacage as the software lead, and formed a new open source project.

In the process of developing ns-3, it was decided to completely abandon backward-compatibility with ns-2. The new simulator would be written from scratch, using the C++ programming language. Development of ns-3 began in July 2006.

The first release, ns-3.1 was made in June 2008, and afterwards the project continued making quarterly software releases, and more recently has moved to three releases per year. ns-3 made its twenty first release (ns-3.21) in September 2014.

Current status of the three versions is:

- ns-1 development stopped around 2001. It is no longer developed nor maintained.
- ns-2 development stopped around 2010. It is no longer developed, and the last maintenance release was in 2013.
- ns-3 is still developed (but not compatible for work done on ns-2)

Simulation workflow:

The general process of creating a simulation can be divided into several steps:

1. ***Topology definition*:** To ease the creation of basic facilities and define their interrelationships, ns-3 has a system of containers and helpers that facilitates this

process.

2. ***Model development*:** Models are added to simulation (for example, UDP, IPv4, point-to-point devices and links, applications); most of the time this is done using helpers.
3. ***Node and link configuration*:** models set their default values (for example, the size of packets sent by an application or MTU of a point-to-point link); most of the time this is done using the attribute system.
4. ***Execution*:** Simulation facilities generate events, data requested by the user is logged.
5. ***Performance analysis*:** After the simulation is finished and data is available as a time-stamped event trace. This data can then be statistically analysed with tools like R to draw conclusions.
6. ***Graphical Visualization*:** Raw or processed data collected in a simulation can be graphed using tools like Gnuplot, matplotlib or XGRAPH.

The Ad Hoc On-Demand Distance Vector (AODV):

The Ad Hoc On-Demand Distance Vector (AODV) routing protocol enables multi-hop routing between participating mobile nodes wishing to establish and maintain an ad-hoc network. AODV is based upon the distance vector algorithm. The difference is that AODV is reactive, as opposed to proactive protocols like DV, i.e. AODV only requests a route when needed and does not require nodes to maintain routes to destinations that are not actively used in communications. As long as the endpoints of a communication connection have valid routes to each other, AODV does not play any role.

Features of this protocol include loop freedom and that link breakages cause immediate notifications to be sent to the affected set of nodes, but only that set. Additionally, AODV has support for multicast routing and avoids the Bellman Ford "counting to infinity" problem. The use of destination sequence numbers guarantees that a route is "fresh".

The algorithm uses different messages to discover and maintain links. Whenever a node wants to try and find a route to another node, it broadcasts a Route Request (RREQ) to all its neighbors. The RREQ propagates through the network until it reaches the destination or a node with a fresh enough route to the destination.

Then the route is made available by unicasting a RREP back to the source. The algorithm uses hello messages (a special RREP) that are broadcasted periodically to the immediate neighbors. These hello messages are local advertisements for the continued presence of the node and neighbors using routes through the broadcasting node will continue to mark the routes as valid.

If hello messages stop coming from a particular node, the neighbor can assume that the node has moved away and mark that link to the node as broken and notify the affected set of nodes by sending a link failure notification(a special RREP) to that set of nodes.

AODV also has a multicast route invalidation message, but because we do not cover multicast in this report we will not discuss this any further ODV needs to keep track of the following information for each route table entry:

Route table management:

Destination IP Address: IP address for the destination node.

Destination Sequence Number: Sequence number for this destination.

Hop Count: Number of hops to the destination.

Next Hop: The neighbor, which has been designated to forward packets to the destination for this route entry.

Lifetime: The time for which the route is considered valid.

Active neighbor list: Neighbor nodes that are actively using this route entry.

Request buffer: Makes sure that a request is only processed once.

Route discovery

A node broadcasts a RREQ when it needs a route to a destination and does not have one available. This can happen if the route to the destination is unknown, or if a previously valid route expires. After broadcasting a RREQ, the node waits for a RREP. If the reply is not received within a certain time, the node may rebroadcast the RREQ or assume that there is no route to the destination.

Forwarding of RREQs is done when the node receiving a RREQ does not have a route to the destination. It then rebroadcast the RREQ. The node also creates a temporary reverse route to the Source IP Address in its routing table with next hop equal to the IP address field of the neighboring node that sent the broadcast RREQ. This is done to keep track of a route back to the original node making the request, and might be used for an eventual RREP to find its way back to the requesting node.

The route is temporary in the sense that it is valid for a much shorter time, than an actual route entry. When the RREQ reaches a node that either is the destination node or a node with a valid route to the destination, a RREP is generated and unicasted back to the requesting node. While this RREP is forwarded, a route is created to the destination and when the RREP reaches the source node, there exists a route from the source to the destination.

Route maintenance:

When a node detects that a route to a neighbor no longer is valid, it will remove the routing entry and send a link failure message, a triggered route reply message to the neighbors that are actively using the route, informing them that this route no longer is valid. For this purpose AODV uses a active neighbor list to keep track of the neighbors that are using a particular route. The nodes that receive this message will repeat this procedure. The message will eventually be received by the affected sources that can chose to either stop
sending data or requesting a new route by sending out a new RREQ.

Conclusion: Thus we have studied NS2 and AODV.