Improve the Efficiency of MANET against Blackhole Attack Using AODV Protocol

Sakthivel. K¹, Sumathi. G², Dhamothiran. V³, Vijayabalan⁴
Assistant Professor¹, Student²,³,⁴
Department of CSE
The Kavery Engineering College, India

Abstract:
The project Improve the efficiency of MANET against black hole attack using AODV protocol is developed by using NS2. In the network, there will be a transmission of packets from source to destination. For the particular transmission, we have to cross many nodes. The nodes are already registered. The registered IP address is there for every node. There will be a request is send by source protocol for transmission of packets. In the black hole attack, a node can send duplicate response to source node. Because of this, packets are get lased. In our project, we overcome this attack before it get happens in order to improve the efficiency of the mobile ad-hoc network. There will be an identification of black hole node done by source node. The source node will find the proper destination node if the registration IP address and acknowledgement IP address of the node both are same. In this we can avoid delivery of packets to other nodes. This will makes to eliminate the black hole node. So the proper transmission takes place.

I. INTRODUCTION:

A With the growth in the use of MANETs, as a standalone networking tool and as the basis for other emerging technologies such as IoT and VANETs the demand or security on this underlying technology is increasing as well. Ubiquitous MANET protocols however, were developed with the focus on efficient routing and data transfer performance, not security issues. This, in turn, led to the current situation where these protocols are vulnerable to a multitude of attacks, including spoofing attacks, flooding attacks, wormhole attacks; replay attacks black-hole attacks, colluding mis-relay attacks, and many others. Black-hole attack, and the more general black-hole attack, on MANETs, are manifested when a malicious Node is able to silently discard some (black-hole) of the messages passing through it. Thus, mitigation of the black-hole attack will also solve the more famous black-hole attack as well. Denial Contradictions with Fictitious Node Mechanism (DCFM), is an algorithm devised to specifically address a denial of service (DoS) attack variant called node isolation in OLSR based networks. As both node isolation and black-hole attacks require similar preliminary steps for attack execution, namely coaxing a victim into appointing the attacker as sole multi-point relay (MPR) node, which is responsible for broadcasting a node’s existence to the network, we found DCFM to be a good basis for mitigating the black-hole attacks as well. Black hole attacks to commence the information provided by DCFM can be used to minimize it as well. These techniques, dubbed IMP.

Ad-hoc On-Demand Distance Vector Routing:
An ad-hoc network is of mobile nodes without the required inter venation of any centralized access point or existing in Restructure. In this paper we present Ad-hoc On Demand Distance Vector Routing AODV a novel algorithm for the operation of such the cooperative engagement of a collection ad-hoc Networks. Each Mobile Host operates as a specialized router and routes are obtained as needed i.e, on demand with little or no reliance on periodic advertisements. Our new routing algorithm is quite suitable for a dynamic self starting network as required by users wishing to utilize ad-hoc networks. AODV provides loop free routes even while repairing broken links. Because the protocol does not require global periodic routing advertisements the demand on the overall bandwidth available to the mobile nodes is substantially less than in those protocols that do necessitate such advertisements.

Attacks against OLSR: Distributed Key Management for Security
In Mobile Ad Hoc Networks (MANETs), mobile nodes use wireless devices to create spontaneously a larger network, larger than radio range, in which communication with each other is made possible by the means of routing. One routing protocol for such MANET networks is OLSR, on which this article focuses. We examine the security issues, and describe an architecture including multiple securing mechanisms. The attacks prevented by this architecture, along with details about protocols, algorithms, mechanisms and implementation details are give

Existing System:
Existing on sensing the wireless channel, approach assigns a max trust value to all its neighboring system based nodes. A node will not do any further communication with a neighbor whose trust value is less than min trust value When a source node receives a RREP message, it updates its routing table, starts transmitting the data packets and inserts a unique sequence number with each transmitted data packet. When the timer expires without hearing the retransmission of this packet, the node reduces the trust value for its next hop node. Trust value information is updated and disseminated to other neighboring nodes. If the trust value of a node decreases below min trust value, it will be isolated by all the nodes in the network.
Disadvantages of existing system:

- High energy consumption due to expensive cryptography and authentication schemes.
- Increases system complexity.
- Increases communication overhead.
- Detection of malicious node consumes more time.

Proposed system:

We incorporate our proposed mechanism into AODV as an example of its use with on-demand routing protocols. The mechanism introduces a new concept of Self-Protocol Trustiness (SPT) which clarifies that the detection of a We propose a new Black hole Resisting Mechanism (BRM) that can be used for all on-demand routing protocols. This paper demonstrates a significant improvement in performance when using our mechanism.

ADVANTAGES OF PROPOSED SYSTEM:

- It does not require expensive cryptography or authentication mechanisms.
- No modifications to the packet formats needed, so the overhead is in small amount.
- Decreases communication overhead.
- Accuracy in mitigation of malicious node.

SYSTEM CONFIGURATION (Minimum Configuration)

The hardware used for the development of the project is:

HARDWARE REQUIREMENTS

- Processor: Intel Pentium 2.10GHz
- Ram: 1 GB
- Hard disk: 160 GB

SOFTWARE REQUIREMENTS

The software used for the development of the project is:

Operating system: Red Hat Linux Mint
Simulator: ns2
Package: ns-allinone-2.27, sensorsim-2.27
Languages: tcl, c++

System Design:

II. NETWORK SIMULATOR -2

NS (version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkeley written in C++ and OTcl. NS is primarily useful for simulating local and wide area networks. Although NS is fairly easy to use once you get to know the simulator, it is quite difficult for a first-time user, because there are few user-friendly manuals. Even though there is a lot of documentation written by the developers which has in depth explanation of the simulator, it is written with the depth of a skilled NS user. The purpose of this project is to give a new user some basic idea of how the simulator works, how to setup simulation networks, where to look for further information about network components in simulator codes, how to create new network components, etc., mainly by giving simple examples and brief explanations based on our experiences. Although all the usage of the simulator or possible network simulation setups may not be covered in this project, the project should help a new user to get started quickly.

CONCEPT OVERVIEW

Ns uses two languages because simulator has two different kinds things it needs to do. On one hand, a detailed simulation of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets.

- Ns meets both of these needs with two languages, C++ and OTcl. C++ is fast to run but slower to change, making it suitable for detailed protocol implementation. OTcl runs much slower but can be changed very quickly (and interactively), making it ideal for simulation configuration. ns (via tclcl) provides glue to make objects and variables appear on both languages.

THE CLASS SIMULATOR

The overall simulator is described by a Tcl class Simulator. It provides a set of interfaces for configuring a simulation and for choosing the type of event scheduler used to drive the simulation. A simulation script generally begins by creating an instance of this class and calling various methods to create nodes, topologies, and configure other aspects of the simulation. A subclass of Simulator called Old Sim is used to support ns v1 backward compatibility.

NODE BASICS

The basic primitive for creating a node is Set ns [new Simulator] $ns node the instance procedure node constructs a node out of simpler classifier objects. The Node itself is a standalone class in OTcl. However, most of the components of the node are themselves Tcl Objects. The typical structure of a (unicast) node is as shown in Figure 2.8. This simple structure consists of two Tcl Objects: The function of these classifiers is to distribute incoming packets to the correct agent or outgoing link.

TRACE AND MONITORING SUPPORT

There are a number of ways of collecting output or trace data on a simulation. Generally, trace data is either displayed directly during execution of the simulation, or (more commonly) stored in a file to be post-processed and analyzed. There are two primary but distinct types of monitoring capabilities currently supported by the simulator. Trace objects are configured into a simulation as nodes in the network topology, usually with a Tcl “Channel” object hooked to them, representing the destination of collected data (typically a trace file in the current directory). The other types of objects, called monitors, record counts of various interesting quantities such as packet and byte arrivals, departures, etc.

TRACE FILE FORMAT

The trace support in OTCl consists of a number of specialized classes visible in OTCl but implemented in C++, combined with a set of Tcl helper procedures and classes defined in the ns library. An example of a trace file might appear as follows:
Default NS2 Structure and Description of the Functionality

NS2 is an object oriented simulator, written in C++, with a Tcl interpreter as a front-end. The simulator supports a class hierarchy in C++ (also called the compiled hierarchy), and a similar class hierarchy within the Tcl interpreter (also called the interpreted hierarchy).

Data Design

Tcl/ C++ variable binding:
Class InstVar defines the methods and mechanisms to bind a C++ member variable in the compiled shadow object to a specified Tcl instance variable in the equivalent interpreted object. The binding is set up such that the value of the variable can be set or accessed either from within the interpreter, or from within the compiled code at all times. Whenever the variable is read through the interpreter, the trap routine is invoked just prior to the occurrence of the read. The routine invokes the appropriate get function that returns the current value of the variable.

Simulator

The simulator is an event-driven simulator. The scheduler runs by selecting the next earliest event, executing it to completion, and returning to execute the next event. Unit of time used by scheduler is seconds. Scheduled to execute at the same time, their execution is performed on the FIFO manner (first scheduled – first dispatched). No partial execution of events or pre-emption is supported. An event generally comprises an event time, event id and a handler function. Two types of objects are derived from the base class Event - packets events and “at-events”. Packets events will be discussed later in detail. An "at-event" is a Tcl procedure execution scheduled to occur at a particular time. This Tcl code first creates a simulation object, then changes the default scheduler implementation to be heap-based, and finally schedules the function "finish" to be executed at time 300.5 (in seconds). "At-events" are implemented as events where the handler is effectively an execution of the Tcl interpreter.

Schedulers

There are presently four schedulers available in the simulator, each of which is implemented using a different data structure: a simple linked-list, heap, calendar queue (default scheduler type), and “real-time”:

- Heap scheduler: Implements the scheduler using a heap structure. This structure is superior to the list structure for a large number of events, as insertion and deletion times are in $O(\log N)$ for $N$ events.
- Real-Time Scheduler: Attempts to synchronize the execution of events with real-time. It is currently implemented as a subclass of the list scheduler. The real-time capability in ns is still under development, but is used to introduce an NS2 simulated network into a real-world topology to experiment with easily-configured network topologies, cross-traffic, etc. This only works for relatively slow network traffic Nodes and Packet Forwarding:

Recall that each simulation requires a single instance of the class Simulator to control and operate that simulation. The class provides instance procedures to create and manage the topology, and

The procedures and functions described in this chapter can be found in the following files:

```
~ns/tcl/lib/ns-tcl, ~ns/tcl/lib/ns-node.tcl, ~ns/tcl/lib/ns-rtmodule.tcl, ~ns/rtmodule.cc, ~ns/rtmodule.h, ~ns/classifier.cc, ~ns/classifier.h, ~ns/classifier-addr.cc.
```

The basic primitive for creating a node is:

```
set ns [new Simulator]
Sns node
```

However, most of the components of the node are themselves Tcl Objects. The typical structure of a node is as shown in figure 4.
This simple structure consists of two Tcl Objects: an address classifier (`classifier_`) and a port classifier (`dmux_`). The function of these classifiers is to distribute incoming packets to the correct agent or to correct outgoing link.

All nodes contain the following components:

- An address (`id_`) – monotonically increasing by 1 (from initial value 0), across the simulation namespace, as nodes are created.
- A list of neighbors (`neighbor_`)
- A list of agents (`agent_`)
- A node type identifier (`node type_`)
- A routing module

- Control functions are:
  - `$node entry` Returns the entry point for a node. The instance variable, `classifier_` contains the reference to this classifier.
  - `$node reset` Resets all agents at the node.

- Address and Port number management function:
  - `$node id` Returns the node number of the node. This number is automatically incremented and assigned to each node at creation by the class Simulator method, `$ns node`

The Classifier

In NS2, this task is performed by a simple classifier object. Multiple classifier objects, each looking at a specific portion of the packet, forward the packet through the node. A node in NS2 uses many different types of classifiers for different purposes. Each classifier contains a table of simulation objects indexed by slot number, which identifies the next simulation object. The C++ class Classifier (see `~ns/classifier.h`) provides a base class from which other classifiers are derived. The classify() method is pure virtual, indicating the class Classifier is to be used only as a base class. If the index is valid, and points to a valid Tcl Object, the classifier will hand the packet to that object using that object’s recv() method. If the index is not valid, the classifier will print and error message and terminate execution. The command() is a method which provides instproc-like to the Tcl interpreter.

Human Interface Design

Simple Links:

Links are created to connect the nodes and complete the topology. We restrict ourselves to describe the simple "point to point" links (NS2 supports a variety of other media). NS2 provides the instance procedure simple-x-link [] to form a unidirectional link from one node to another.

```
$ns simplex-link <node0> node1> <bandwidth> <delay> <queue type>
```

The above command creates a link from `<node0>` to `<node1>`, with specified `<bandwidth>` and `<delay>` characteristics. The link uses a queue of type `<queue type>`.

The procedure also adds a TTL checker to the link. Five instance variables define the link:

- `head_` - Entry point to the link, it points to the first object in the link.
- `queue_` - Reference to the main queue element of the link.
  - Simple links usually have one queue per link. Other more complex types of links may have multiple queue elements in the link.
  - `ttl_` - Reference to the element that manipulates the TTL in every packet.

Delay and Packet Scheduling:

Packet scheduling refers to the decision process used to choose which packets should be serviced or dropped.

Delays represent the time required for a packet to traverse a link. The amount of time required for a packet to traverse a link is defined to be \( \frac{s}{b} + d \) where \( s \) is the packet size (as recorded in its IP header), \( b \) is the speed of the link in \( \text{bits/sec} \), and \( d \) is the link delay in seconds.

Delays are defined in Link Delay class in `~ns/delay.cc` and the implementation of the delay is in the recv() method. This method operates by receiving a packet, p, and scheduling two events.

Agents:

Agents represent endpoints where network-layer packets are constructed or consumed, and are used in the implementation of protocols at various layers. The class Agent has an implementation partly in Tcl and partly in C++ and is contained in `~ns/agent.cc`, `~ns/agent.h`, and `~ns/tcl/lib/ns-agent.tcl`. The C++ class Agent includes internal state members to assign various fields to a simulated packet before it is sent.

This state includes the following:

- `addr_` – source address in packet
- `size_` packet – size in bytes (placed into the common packet header)
Agents are used in the implementation of protocols at various layers.

**TIMERR_IDLE, TIMER_PENDING, or (TIMER_HANDLING)**.

The abstract base class Timer Handler contains the following protected members:

- `virtual void expire (Event* e) = 0` – The pure virtual function `expire()` must be defined by the timer classes deriving from this abstract base class.
- `virtual void handle (Event* e)` – consumes an event

**Module**

1. **Processing Node ID:**
   The author proposes to add a new field to the RREQ control packet named Processing Node ID (PN_ID) field. This field will be used to store the value of IP of the intermediate node that processes the RREQ. The source node will keep this field empty when it initiates the route request and forward the RREQ packet. It will keep a copy of its own RREQ.

2. **Broadcast received RREQ:**
   If the node is an intermediate node, it is required to send the RREQ back to the originator node by adding the value of its IP address to the field named PN_ID. This process is to be done by all the source node and intermediate nodes.

3. **Storage in cache memory:**
   The source node and the intermediate nodes will store the values of the received RREQ viz, Destination sequence Number, Destination IP address, PN_ID its own cache memory if it is a new RREQ. If the RREQ has already been processed, it will discard it.

4. **Compare with received RREP:**
   Once the RREQ reaches destination node, it will send the RREP back by unicasting. The nodes which get back the RREP will compare the values of the already stored RREQ fields in the cache i.e. Originator IP address, Originator Sequence Number, Destination Sequence number, Destination IP address and Timestamp with the values in the RREP.

5. **Detection of Black hole:**
   If a Black hole node is detected, because of its behavior, the data packets are not forwarded to it and either a new route Discovery is initiated or next available node having optimal path is used to forward the data.

**III. CONCLUSION:**

Efficiency is one of the important feature while deployment of MANET. In this paper, we have studied the routing security issue in the MANETs described the black hole attack and proposed a feasible solution by using AODV protocol in order to avoid the black hole attack and also securing the network from malicious node. Proposed system is based on comparing IP address of the destination during registration time and response time. This can be used to find the proper destination and secured route.

**IV. REFERENCE:**


