2012

Performance comparison between Ad Hoc On Demand Distance Vector and Dynamic Source Routing Protocols with security encryption using OPNET

Jaseem Jafferi

Performance comparison between Ad Hoc On Demand Distance Vector and Dynamic Source Routing Protocols with security encryption using OPNET

By

Jaseem Jafferi

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in
Networking and System Administration

Rochester Institute of Technology

B. Thomas Golisano College

of

Computing and Information Sciences

Rochester Institute of Technology

B. Thomas Golisano College

of

Computing and Information Sciences

Master of Science in
Networking and System Administration

Project Approval Form

Student Name: Jaseem Jafferi

Thesis Title: Performance comparison between Ad Hoc On Demand Distance Vector and Dynamic Source Routing Protocols with security encryption using OPNET

Project Committee

Name	Signature	Date
Prof. Tae Oh		01/06/2012
Chair		
Prof. Charles Border		01/06/2012
Committee Member		
Prof. Bill Stackpole		01/06/2012
Committee Member		

Abstract

Application for wireless networking has been evolving rapidly and is becoming an integral part in our everyday life. Also with the recent performance advancement in wireless communication technologies, mobile wireless ad-hoc networks has been used in many areas such as military, health and commercial applications. Mobile ad hoc networks utilize radio waves and microwaves to maintain communication channel between computers. 802.11 (Wi-Fi) is the pre-eminent technology for building general purpose wireless networks. Mobile ad-hoc networking (MANET) utilize the Internet Protocol (IP) suite and aims at supporting robust and efficient operation by incorporating routing functionality into the mobile nodes. MANET is among one of the wireless networks that uses 802.11 to transmit data from the source to the destination. Since MANET is used in applications like defense, security is of vital importance due to its wireless nature. Wireless networks are vulnerable to attacks like eavesdropping, Man-In-The-Middle-Attack (MITM), hijacking, and so are MANETs. A malicious node can get within the wireless range of the nodes in the MANET and can disrupt the communication process. Various routing protocols have been proposed using encryption techniques to protect routing in MANETs. In this thesis, I implemented security encryption techniques (SHA-1 and RSA)in two reactive routing protocols which are Ad Hoc On Demand Distance Vector (AODV) routing protocol and Dynamic Source Routing (DSR) routing protocol and compared their network performance using performance evaluation parameters: Average end-to-end-delay, routing load, packet delivery fraction. Encryption techniques like SHA-1 and RSA were used to maintain the confidentiality and the integrity of the messages send by the nodes in the network. There have been several researches so for but no one has ever compared the performance of secured MANET protocols. I am going one step further by comparing the secured routing protocols which would be helpful in determining which protocol performs better that can be used in scenario where security is of utmost importance.

Table of Contents

1. Introduction

MANETs are infrastructureless wireless networks. MANETs are basically self-configuring network of mobile devices connected by wireless links and can be rapidly deployed offering mobility and connectivity without constraints anywhere and anytime. A MANET is a multi-hop network where each node is responsible for sending data packets to the other node until the data packet reaches the destination. Nodes in MANET are constantly moving making the multi-hop network topology dynamic. Due to the mobile nature of the nodes some links break while new links between nodes are created. Nodes may be deployed in an unpredictable environment like unknown terrains, hazardous conditions, hostile environments and there is a high possibility of the nodes could get tampered or destroyed completely. MANET is also affected by the unreliability of the wireless network. Wireless networks are unreliable and subject to errors due to the method of transmission of data. Unpredictability is also due to high levels of electro-magnetic interference (EMI) or severe changes in weather conditions. MANET is characterized as a resource constrained network. The nodes in the network operate on batteries and have limited memory and processing capabilities. The nodes may be situated in areas where it is not possible to recharge or replace the batteries and are considered to have a limited life time [1]. Mobility makes mobile wireless networks more prone to attacks than the wired network. An attacker can secretly join the network and perform attacks like eavesdropping, spoofing and Denial-of-Service attack [2]. Since the MANET network is constantly mobile each node must act as a router and should be able to find a route to send the packets to the node that would eventually forward it to the destination node. The routers in the network communicate with each other by following a set of rules defined in the standard called a *protocol* or a *routing protocol .*

Routing Protocols: A routing protocol specifies how nodes in the network communicate with each other and delivers packet to its intended destination. Routing is basically a two step process: a. to

determine the optimal routing paths and b. transfer the packets through the network. The packet is transmitted hop-by-hop from one node to another node depending on the routing algorithm a routing protocol runs to calculate the route. Initially each node only knows about its one hop neighbor and then about the entire network. A MANET routing protocol should have the following characteristics:

1. *Distributed nature*: In a MANET network there is no centralized control. Every node is responsible for transferring the packets and the protocol should be able to adapt to the distributed nature of the MANET network

2. *On demand Operation*: A MANET routing protocol find routes only on a demand or need basis in order to make use of the limited resources in an efficient manner. This type of protocol is also called a *Reactive Routing Protocol*.

3. *Proactive operation*: In certain situations latency caused due to demand based route calculation may be unacceptable. Therefore the routing protocol should also be able to find routes ahead of time if resources are available. This type of protocol is also called a Pro*active Routing Protocol*.

4. *Loop Free*: A routing protocol must be loop free to ensure proper message delivery and efficient network operation.

5. *Scalable*: A routing protocol should be scalable. As new nodes join and existing nodes gets disconnected from the network the routing protocol should be able to find new routes to forward the packets without having to restructure the entire network.

6. *Security*: Since MANETs are more vulnerable to attacks provisions should be made to implement security using IP security techniques or other encryption techniques.

7. *Bidirectional/Unidirectional links*: Because of the dynamic nature of MANET networks, a routing protocol should be able to execute on both bidirectional and unidirectional links [3].

Security in Mobile ad hoc networks has become a major concern and has emerged as an important research area. MANETs are wireless networks, more susceptible to intrusion by malicious agents than any wired network. Wired networks can be protected by restricting the physical access to the network infrastructure which reduces the risk of intrusions. However, physical security measures are less effective in limiting access to the wireless media. Authentication mechanism is needed to prevent unauthorized access to the MANETs. Also, we need to ensure the integrity of the messages transferred from the source to the destination. There are chances that the malicious node could change the message and pass it on in the network which may either result into loss of data and in some cases no data is been transferred. Therefore there is a need to implement security parameters in order to protect MANETs from malicious users.

In this thesis I have implemented encryption techniques in two reactive routing protocols AODV and DSR to protect the routing messages that are forwarded by the nodes to deliver the data to the intended destination. SHA-1 was used to maintain the integrity of the routing messages to make sure the hop count field in AODV was not changed during transmit. RSA was used to authorize the nodes in the network running AODV and DSR routing protocols.

Types of Routing Protocols: There are three types of routing protocols. They are as follows:

1. *Proactive routing protocols*: Proactive routing protocols are also called as Table-Driven routing protocols. They are called proactive because each and every node in the network maintain routing information even before it is needed. The nodes are constantly looking for changes in the network topology and update their routing table such that each node in the network knows the route to every

other node in the network. They maintain multiple routing tables. One is considered to be active currently in use and other as a backup when the topology changes. The protocols are not suited for large network as they need t maintain route information to each and every node in the network which may cause more overhead leading to consumption of more bandwidth. Examples of proactive routing protocols are OLSR, DSDV and IARP.

2. *Reactive routing protocols*: Reactive routing protocols are also called as On-Demand routing protocols. It get its name because of the fact that nodes are active only when there is a need to set up communication to transfer data. They do not maintain any routing information or routing activity in the network if there is no communication. Only when there is a need to transfer data communication is established by the source, looking for a route to reach to the destination. The route discovery is done by flooding the network with route request packets. Due to its on-demand nature these protocols have less overhead, are bandwidth efficient and suitable for large networks. Since they do not have any back up route in case of the change in the topology the source node needs to initiate another route discovery process which makes route maintenance a little expensive. Examples of such protocols are AODV and DSR.

3. *Hierarchical routing protocols*: An hierarchical routing protocol makes use of the advantage of both proactive and reactive routing protocol. Depending on the type of application they are for, the nodes can be active only until they the path to reach to every other node in the network and then operate in an on-demand manner. Examples of hierarchical routing protocols are CBRP and FSR.

Statement of Purpose: The aim of this thesis was to compare the performance of two reactive routing protocols (AODV and DSR) such that security encryption techniques are implemented to measure their performance against each other.

Through this thesis I went one step further as compared to other researches where either two traditional routing protocols were compared to get to a conclusion which one outperforms the other or security was implemented in a particular protocol and its performance was evaluated.

2. Reactive Routing Protocol

A reactive routing protocol is an on-demand routing protocol for MANETs. A source node using reactive routing protocol will initiate a route lookup process only when it has to send data to another node in the network. Hence need for a route triggers the route look up process and therefore it is named as Reactive Routing Protocol. A reactive routing protocol is considered to be a bandwidth efficient routing protocol because of its on demand basis of route calculation. A reactive routing protocol performs route discovery and route maintenance function. Need for a route triggers a route discovery process. A source node “S” needs to find a route to the destination node “D”. Node “S” sends a Route Request (RREQ) packet which is heard by its immediate neighbors. Node “S” maintains a list of its immediate neighbors that have direct communication link with node “S”. The neighbor list is maintained by periodically sending Hello packets to the neighbors and in return the neighbors reply back immediately to the initiator of the Hello packet. Node “S” also uses a sequence number that serves as an Identification number to make sure that same RREQ is not broadcasted in the network. It also serves to determine the freshness of the route. Figure 1a. below shows the use of sequence number. If node “S” has a valid route to node “D” it can proceed with the data transfer. If “S” does not have any route to “D” and route discovery is not in process it sends an RREQ to its neighbors with Sequence Number incremented by 1. Node “S” maintains a list of nodes to which it has sent RREQ packet for destination “D”. Each RREQ is uniquely identified by the source address, destination address, a sequence number. Suppose the RREQ from “S” reaches node “X”. Node “X” looks for a valid route to “D” in its routing table. If it has a route to “D” it sends the RREQ through that route to “D” and waits for a reply to make sure the route to “D” in its

routing table is still valid. If it gets a Route Response (RREP) from “D” it sends back the RREP to “S” else “X” sends a RREQ to its neighbors until it reaches “D”. Now as the RREQ is been propagated each node in the network also maintain a list of node from whom they received RREQ creating a reverse path to the initiator of the RREQ. Once RREQ reaches “D” it uses the reserve path created during RREQ and sends a RREP to the source node “S”. This is how route discovery takes place using a reactive routing protocol. Node “S” starts sending packets after route discovery is completed. There might be a case where there are multiple routes from one source to the destination. In such a case the most optimal route is kept in the active routing table while the other routes are kept in the passive routing table which can be used as a backup route in case a link breaks. Optimal route is based on the metrics like hop count from source to the destination or route cost. In case of the tie with one metric, other metric is used to determine the optimal path. Figure 1 below describes the route discovery process of a reactive routing protocol.

Once route discovery process is completed the nodes in the network needs to maintain the routes in the route table. This is done by periodically sending Hello messages to each node. The nodes receiving Hello message should reply to it immediately otherwise the link is considered to be broken which is notified to the source via the Route Error (RERR) message. Figure 2b below shows the propagation of RERR message to the source node S. The source then initiates a new route discovery process. Figure 2b below shows the new route discovered by the source node S. Each route is associated with a time out field which gets updated every time the route is used. If the route is not used for a long time, the timer on it expires and the route is deleted from the active route table and transferred into passive route table. The sequence number allows the nodes to determine how fresh their information is on the other node. Whenever a node sends a new message it increases its sequence by one. All the nodes keeps track of the sequence number of other nodes it talks to. The higher the sequence number the fresher the route is. In the figure 1a below node 1 is forwarding a RREP to node 4 and notices that

the route in the RREP has a better sequence number than the route in its RREP. So it replaces the its route with a fresh enough route.

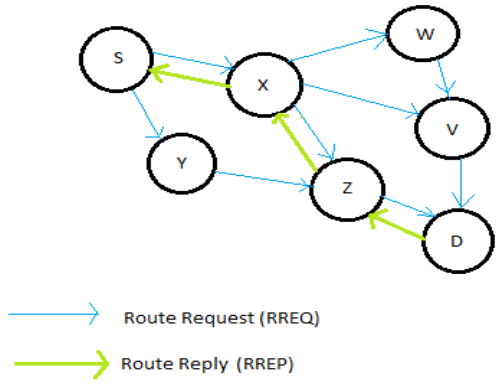


Figure 1: Route Discovery Process

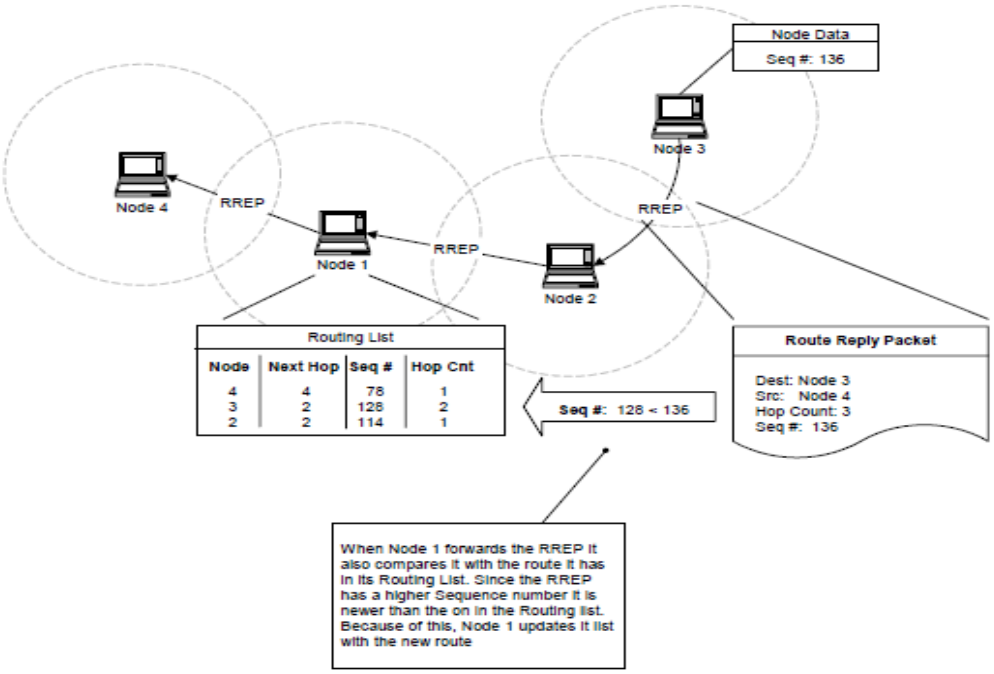


Figure 1a. Sequence numbers: http://www.antd.nist.gov/wctg/aodv_kernel/aodv_guide.pdf

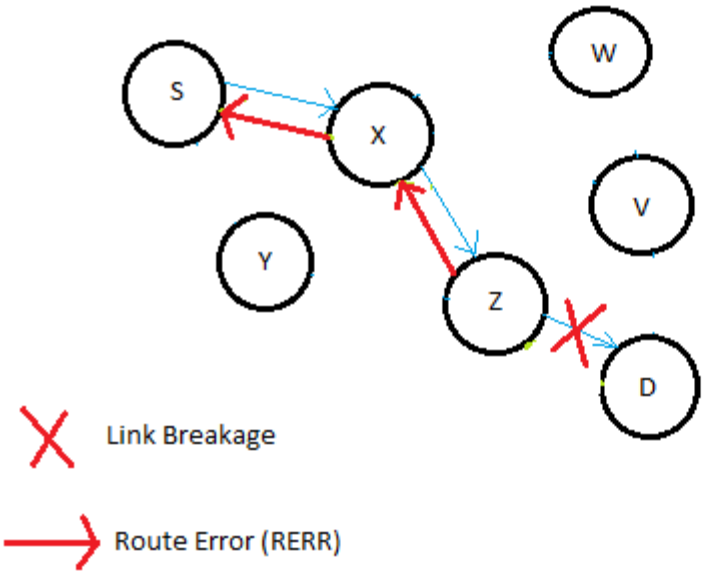


Figure 2a: Link from node Z to node D breaks. Node Z sends the RERR to the source node S

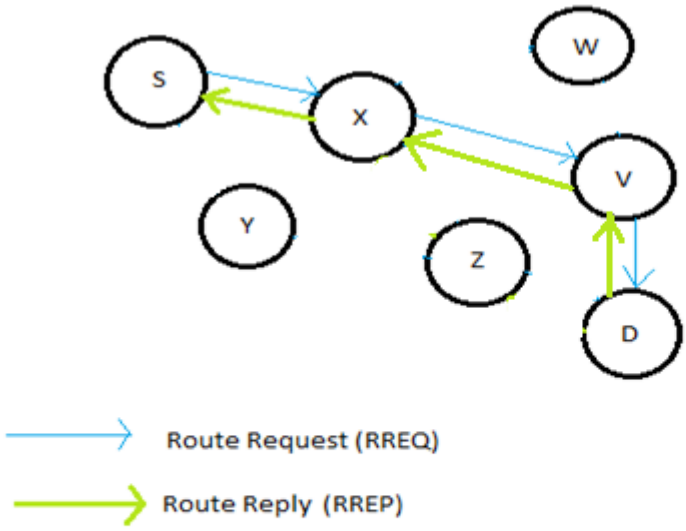


Figure 2b: Source node S finds a new route to reach the destination node D

Ad Hoc On Demand Distance Vector (AODV) and Dynamic Source Routing (DSR) routing protocol: AODV and DSR are reactive routing protocol. They initiate a route request process only when there is a need to transfer data from one source to the destination. Both uses RREQ and RREP packets during a route discovery process, Hello messages to maintain the routes and RERR packets to notify the sender in case of a link failure. Both AODV and DSR work the same when it comes to route discovery process and router maintenance process but the way they maintain routing information.

AODV uses traditional routing table one entry per destination. It relies on the routing table to send RREP packets back to the source and then send data packets to the destination. It uses sequence numbers to determine the freshness of routing and to avoid routing loops. These sequence numbers are carried in the packet header of the data packet. It sets a timer on each route in the routing table and once the timer expires the route is considered to be a stale route. Stale routes are deleted and if in the future data is to be transferred to the same destination, route request process is initiated. So in case of AODV nodes maintain information about only its one hop neighbor in its routing table [4].

DSR makes use of source routing. The source node knows the complete hop by hop route to the destination. The data packet contains the complete hop by hop route to the destination. The nodes in the network just follow that route. RREQ and RREP packets are also source routed. Each route request also contains the address of the intermediate node through which the RREQ has been forwarded. DSR also makes use of source caching. The forwarding nodes also cache the source route in their routing table for future use. Since the route is source forwarded DSR does not need any sequence number to avoid routing loops. In case of a link failure the intermediate node uses an alternate route in its route cache to send the data packets and notifies the source about the change in the route. The source removes any route using this link from the route cache. The source then piggybacks the RERR in the following RREQ to make sure the intermediate nodes clear the route that has a failed link. The nodes

using DSR makes use of the Promiscuous Listening. The nodes check whether it has a better route to the destination. If so, it sends a gratuitous RREP to the source of the route with the new route [5].

Therefore the nodes using DSR are exposed to more routing information than AODV. In a single RREQ and RREP cycle the nodes are aware of the entire network topology. Nodes using AODV has a limited routing information one hop per node. DSR makes use of route caching and replies to all the RREQ reaching a destination. So there may be a possibility of having multiple routes to reach the destination which can be used in case of a link failure. AODV replies to one RREQ the one that reaches first to the destination i.e. the route with the least number of hops. So in case of a route failure the source initiates a route discovery process. In DSR, there is no such mechanism of stale routes. In case of a multiple routes, the nodes may use stale routes which can start polluting other caches. In contrast AODV has a much better approach to take care of the stale routes. It uses sequence numbers to determine the freshness of the route in case of multiple routes. The route with a higher sequence number is considered to be fresh.

AODV and DSR are the most widely used MANET routing protocols. This can be proved by the number of researches going on with these protocols. There are also academic papers that clearly mention the popularity of these protocols.

Over the years researchers have been showing interest in VOIP operated over MANET. The reason behind this is MANET being infrastructureless VOIP can be created over network without any infrastructure support. Applications of such a VOIP network could be used in emergency search, battle field and disaster rescue operations. This VOIP application is been implemented using AODV routing protocol [14]. AODV guarantees loop free routing. There was another research in the area of AODV where an algorithm was developed to improve the performance of the routing protocol [15]. Since researches have been done to improve the protocol performance it is evident that it is due to its most

common use in the real world environment.

DSR is popular because of it being simple and efficient MANET routing protocol. It easily guarantees loop free routing through source routing, can be operated in a network with unidirectional links, and performs fast network recovery when there is a route change in the network. Researchers have showed much interest to improve the performance the DSR since its invention. The way to efficiently make use of the energy resources was introduced in a paper "*Modified Energy-Aware DSR Routing for Ad Hoc Network*" by Yang Tao and Wei Luo. In this paper the protocol was modified in such a way that it would select a route which would require the least amount of energy and have the maximum energy at that point in time [16]. Another research by Asokan R. and Chandrashekhar S, "*Performance Enhancement of Dynamic Source routing using Simulated Annealing*" was proposed with QoS as the main component. This QoS aware routing protocol had better performance than the traditional DSR in terms of throughout, end-to-end delay and packet delivery ratio [17].

All these researches in the area of AODV and DSR directly or indirectly were aimed at improving the performance of the traditional protocols. Thus it can be concluded that AODV and DSR are amongst the most popular and widely used Reactive Routing Protocols in MANET.

3. Literature Review

There have been several research endeavors in comparing reactive routing protocols. One such endeavor was the *Performance Comparison of two On-Demand Routing Protocols for Ad Hoc networks*. This research compares AODV and DSR using a simulation model with MAC and physical layer models to study inter-layer interactions and their performance implications. NS-2 (Network Simulator) was used to evaluate the performance parameters. NS-2 can simulate multi-hop wireless networks with physical, data link and MAC layer models. The distributed coordination function (DCF) of IEEE 802.11 is used as

the MAC layer. DCF uses Request to send (RTS) and clear to send (CTS) to send packets to neighboring nodes. Every data packet sent is followed by an ACK response. Broadcast data packets and RTS control messages are sent using physical carrier sensing. The simulation used CSMA/CA to transmit the data packets and a commercial radio interface. WLAN was a shared media radio with a nominal bit rate of 2Mb/sec and a nominal bit range of 250 meters. The routing protocol model sees all the packets forwarded and responds by performing appropriate activities. RREQ was considered as a broadcast packet in the MAC. RREP, RERR and data packets were all unicast packets, the destination being the neighboring node. Link breakage was detected using feedback from the MAC layer and a signal is sent to the routing layer when the MAC layer fails to deliver a unicast packet to the next hop. The simulation never uses Hello messages instead it uses CTS and ACK. A failure to receive a CTS after an RTS and an ACK following data transmission indicates a data transfer failure. In this simulation the protocols maintained a send buffer of 64 packets. Packets waiting for a route reply were buffered and queued at the interface queue until the MAC layer can transmit them. Traffic sources used in this simulation are continuous bit rate (CBR). Only 512 bytes packets were used and the number of source-destination pair and the number of packets transferred was varied to change the offered load in the network. The simulation used the random way point model in the rectangular field with 50 and 100 nodes. The nodes moved at a uniformly distributed speed between 0-20m/sec. Simulations were run for 900 simulated seconds for 50 nodes and 500 simulated seconds for 100 nodes. Each data point represents an average of atleast 5 runs with identical scenarios but different randomly generated mobility scenario. Identical mobility and traffic scenarios were used for both protocols performance evaluation. The evaluation performance characteristics were (i) Packet delivery fraction-It is the ratio of the data packets delivered to the destination to those generated by the CBR sources, (ii) Average end-to-end delay of the data packet – this include all possible delays caused by buffering during route discovery latency, queuing at

the interface queue, retransmission delays at the MAC, propagation and transfer times, (iii) Normalized routing load-the number of routing packets transmitted per data packet delivered at the destination [6].

Another research work in the area of MANET routing protocols was the implementation of *AODV-SEC*. AODV-SEC aims at securing AODV routing protocol. It is a protocol extension to the AODV protocol. To implement security in AODV the research relies on PKI. Every node in the network owned a certified keypair, current certificate of the certificate authority (CA) to verify unknown certificates from other nodes and every node owned a certificate in order to participate in the network. AODV-SEC used Digital Signatures to guarantee the origin and integrity of the data. Hence digital signatures were used to protect the content of messages from been modification and to verify the originator of the request or reply. Digital signature was included in the route setup packets. RSA algorithm and SHA-1 hashing was used to implement AODV-SEC. Hashing can be used to secure data and in this research it was used to secure the minimal length of the route. SHA-1 hashing implemented in the protocol protected the number of hops from being modified by the nodes as the current hop count value was secured using hash chain values.

AODV protocol was extended to implement AODV-SEC functionality. The 8-bit Length field in AODV was changed to 16-bit to guarantee AODV-SEC extension. Message formats of RREP, RREQ, RERR and Hello messages for AODV was extended to include Hash function, Top hash, CrypteOR, CryptLH, Signature, and Certificate.

Libcrypto library was used in order to implement real cryptographic mechanism like RSA and SHA-1 for simulation. NS-2 simulator was used to simulate the performance of AODV-SEC. A two-way ground model was used as the physical model. In the simulator the parameters of 2.4GHz Lucent Orinoco WaveLAN DSSS Radio interface was applied. The data rate was set to 11Mb/sec and the transmission range was 170m. Distributed Coordinated Function (DCF) was used as the Media Access

Model. A random way point mobility model was used. Two scenarios sizes were used in the simulation. Small scenario had a size of 900 x 200m and 20 simulated nodes. Large scenario had a size of 1500 x 300m and 50 simulated nodes. The node pause time varied between 0s (high mobility) and 600s (low mobility). Constant Bit Rte model was used to model data traffic. The simulation was done on different number of data sources distributed evenly. In a small scenario either 4 or 16 sources were used and in the large scenario either 10 or 20 sources were used. The packet size had a size of 512 bytes. The simulation was done to obtain the following results: (i) packet delivery fraction, (ii) average end-to-end delay, (iii) normalized routing load, (iv) normalized MAC load, (v) route acquisition time, (vi) number of RREQs per node [7].

A Secure Routing Protocol for Ad Hoc Network (ARAN) is a protocol that was designed to secure routing process in MANETs. In this research wok the authors had described the security threats against ad hoc routing protocols specifically AODV and DSR. They also described the security requirements to secure MANETs routing protocol and implemented ARAN. ARAN is based on certificates and introduces authentication, message integrity and non-repudiation in a MANET network as a part of minimal security policy. Like AODV-SEC, ARAN makes use of cryptographic certificates to secure routing in MANETs. A certification process takes place before the route instantaneous process that guarantees end to end authentication. It relies on a trusted certificate server "T" whose public key is known to all valid nodes in the network. Nodes requests for a certificate from the server "T" before entering the network and each nodes gets only one certificate after authenticating its identity to the server. The certificate contains the IP address of node, public key of the node, a timestamp t of when the certificate was created, and a time e at which the certificate expires. All nodes using ARAN maintains fresh certificates and uses these certificates to authenticate themselves to another node during the exchange of routing messages. A source node "S" begins a route initiation process to destination "D" by broadcasting a *route discovery packet* (RDP) to its neighbors. The RDP includes a packet type identifier (which is RDP in this

case as it is a route discovery process), IP address of the destination (IP_D), certificate of node "S", a nonce and the current time t, all signed by S's private key.

$S \rightarrow$brdcast: [RDP , IP_S, $cert_S$, N_S, t] K_S

When a node receives a RDP message is sets up a reverse path to the source by recording the IP address of the node it received the RDP. Suppose that RDP send by "S" RDP is received by node "P". "P" uses S's public key from S's certificate to validate the signature and verify that S's certificate has not expired. Node "P" also checks the IP address of the source to make sure it has not already processed the RDP packet. If the RDP is not processed the nodes sign the contents of the message, removes the signature of the previous nodes and append its own signature and rebroadcasts the RDP. Once the RDP reaches the destination a REP packet is send by the destination to the source which follows the reverse path. The REP packet also contains the fields as the RDP packet.

Protocol evaluation of ARAN was done using Global Mobile Information Systems Simulation Library (GloMoSim). Traffic type in the simulation used was CBR over UDP and 802.11 MAC layer. Two types of field configurations were used. One with 20 nodes distributed over a 670m x 670m terrain and 50 nodes over a 1000m x 1000m terrain. Here also a random way point mobility model was used. Each node raveled to a randomly selected location at a configured speed and stayed there for a specified pause time before changing its location again. The transmission range was 250m. Simulations were ran for constant node speeds of 0, 1, 5 and 10 m/sec, with pause time fixed at 30sec. Five CBR sessions were simulated in each run, with random source and destination pairs. Each session generated 100 packets 512 bytes each at the rate of 4 packets per second. ARAN was simulated using 512 bit key and 16 byte signature. To determine the performance of the ARAN protocol the following parameters were obtained through the simulation: packet delivery fraction, routing load (bytes), routing load (packets), average path length, average route acquisition latency, average end-to end delay of data packets [8].

Another research in the area of MANET was the development of *Energy Aware* routing protocol. The protocol was an extension to the AODV and was named as AODV_EA. The basic idea was to find neighboring nodes with an active route to the destination, having energy above the threshold value. The nodes that had energy level more than the threshold level were eligible to participate in the route discovery process. The distance from the node to the destination was also taken into consideration. The route discovery process sped up as more and more active routes satisfying the energy and distance criteria were discovered. Local repair of the active routes were automatically built in the scheme as it had both distance and energy metrics. The source node "S" broadcasts an *ACTIVE_REQ* message containing threshold value "Tth", and the current estimated distance Ds from "S" to destination "D". At the neighbor node "N" energy of the node "En" is greater than the pre-defined threshold "Tth" and the condition T-T(N) < Tth is satisfied where T and T(N) are the current time and the time when the last packet had been forwarded D through N respectively then a reply message *ACTIVE_REP* message containing the route length is sent. If En is greater than Eth and the distance from the node Ds is greater than the current estimated distance of the node from the destination Dn then an ACTIVE_NEG message is sent. If En is less than Eth than no reply is sent. At the source node S all the ACTIVE_REP messages are received and scanned. Only the neighbor with the shortest route is selected for forwarding data and other nodes are stored as alternate nodes in the event of a link failure. ROUTE_REQ message is sent to the selected node. If only ACTIVE_NEG messages are received by the S then neighbor with the minimum distance Dn is selected for forwarding data. Other nodes are kept for future use in case of a link failure. ROUTE_REQ2 message is sent to the selected node. When the node receives ROUTE_REQ message it forwards it to the available active route and the destination node D sends back the ROUTE_REP on the reverse path. When the node receives ROUTE_REQ2 it repeats the process with its neighboring nodes from the beginning of the route discovery. When S receives a ROUTE_REP, a route is considered to be

established and data is forwarded aver the established route. Figure 3a-3e below explains the routing process.

AODV_EA was also simulated using NS-2 and its performance was compared with AODV and DSR. Network scenarios in the area of 700 x 700m for 10 and 20 nodes and 1000 x 1000m for 50 nodes were used for simulation. Value for packed delivery ratio was observed by varying pause times from 0 to 500 seconds in different scenarios from small to large networks. The performance of the proposed protocol was performed on the basis of packet delivery fraction (PDF), end-to-end delay and was compared against AODV and DSR. The results obtained at the end of the simulation showed that AODV_EA performed better than the traditional AODV and DSR routing protocols [9].

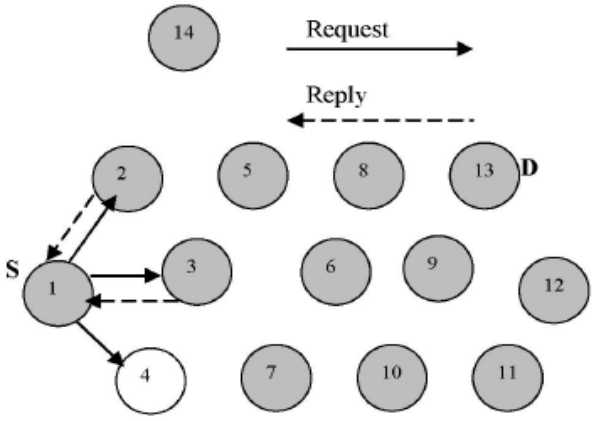


Figure 3a: Source S initiates active route request. Nodes which satisfy threshold criteria reply

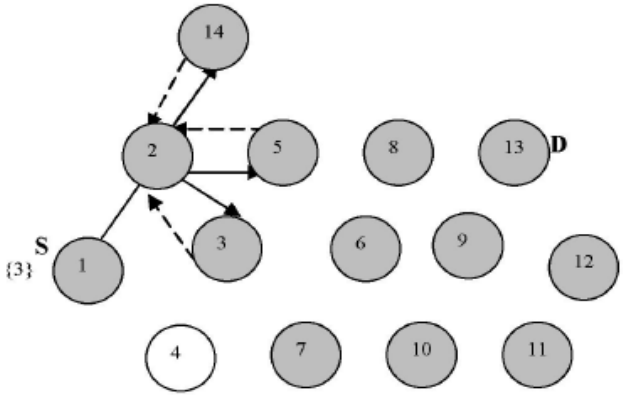


Figure 3b: Node 2 is selected for route request and node 3 is kept as backbone node

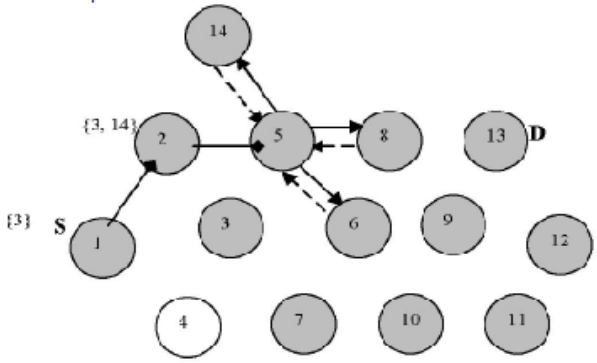


Figure 3c: Node 5 is selected for route request and node 3, 14 are kept as backbone nodes

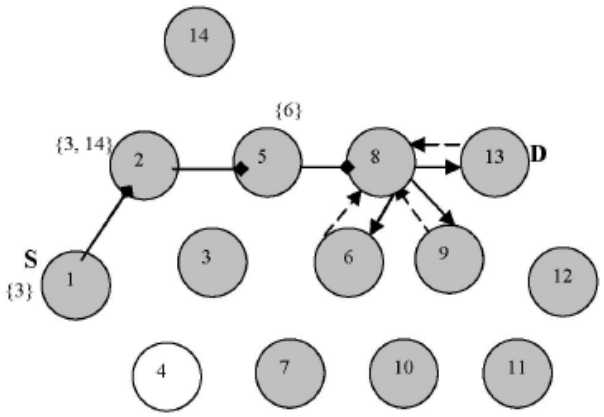


Figure 3d: Node 8 is selected for route request and node 6 is kept as backbone node

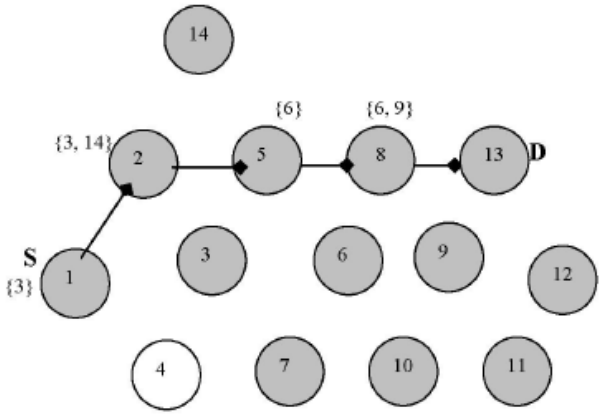


Figure 3e: Node 13 is selected, which is the ultimate destination and node 6, 9 are kept as backbone nodes

http://dl.acm.org/citation.cfm?id=664514

Note: All the gray colored nodes in the figure have energy above the threshold level.

A research work conducted by the team of University of California, *Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks* resulted into an efficient and effective distribution of certificates among the nodes in the network. In this research the certification authority functions were distributed through a threshold secret sharing mechanism. Each entity holds a secret share and multiple entities in the neighborhood jointly provided complete services. The certification scheme was based on PKI to maintain security of data via valid certificates issued by a globally trusted certification server. The researchers came up with a new scheme to provide certification services by distributing the certification authority (CA) functionality to each local neighborhood. A coalition of K neighbors served as a CA and jointly provided certification services to mobile nodes in the network. They came up up with a self initialized protocol to handle the dynamic nature (joins and leaves) of the nodes and secret share updates. Each node can be reinitialized by K neighbors and once initialized it is qualified to be a coalition member to serve its neighborhood. RSA encryption was used as a PKI to generate public and private keys. The system CA's RSA key pair was denoted by {SK,PK} where SK was the system/private

key and PK was the Public key. The certificates were signed using the private key which was verified by the public PK. Threshold secret sharing is such that SK was shared among the network entity. Each entity v_i holds a secret share P_{vi} and any K of such secret share holders could collectively function as a CA. Each entity also had their own RSA public and private keys $\{sk_i, pk_i\}$ to maintain end-to-end security. The nodes obtain its secret share during the bootstrapping process or through self initialization service. So in order to compromise the security of the network the attacked had to compromise the K nodes in the network to get the secret of the keys, making it hard for the attacker. An assumption is made that at the bootstrapping phase an entity can obtain certificates form a trusted centralized management. Later if an entity joins a network or wishes to recover its certificate from a system crash, then a well known policy must be predetermined before the ad hoc network is formed. The initial certificate issued was renewed on a timely basis within T_{renew} time. Certificates were also renewed once a node updates its personal key pair by presenting its current valid certificate and a future expiration time T<(current time + T_{renew}) for the new certificate. Simulation of the proposed work was done using NS-2 and metrics like success ratio (the ratio of number of the successful certification services over the number of attempts during simulation time), average delay (average latency for each node to perform certification services) and average number of failures (number of times an entity fails on average before successfully accomplishing its certification) were obtained [10].

An extension to the above research *Certification and Authentication Services for Securing MANET Protocols* was basically done to improve the certificate distribution and renewal process that had some assumptions. A set of configurable policies options was proposed for certificate issuing and renewal, and private-key share issuing and updating. The policy was used to enforce the new certificate and he private key of the system SK_{CA} issuing in the DCA follow strictly the same identity verification that a centralized CA would do to avoid multiple identity tampering attacks. This research also specifies how actually the certificates were acquired by the nodes. Localized maintenance of a certificate cache and

CRL was proposed. Also this research extended the model to support multiple DCA and trust path constructions. The policies for certificate issuing and renewal were: (i) based on identity check (ii) The user was prompted to decide if a certificate request could be server (iii) deny i.e a certificate request is denied and an error message was returned to the requester (iv) reject i.e the certificate request was discarded silently (v) other (user defined). Local certificate database was used depending on the type of type of routing protocol used. Certificate management was based on the choice for proactive or reactive construction of local certificate cache and Certificate Revocation List (CRL), time for keeping unused certificates in the cache and maximum size of the certificate. Proactive certificate distribution requires the sender to attach its certificate with the message transmitted. This may lead to a network overhead as certificate need to be evaluated for each message. So in such a case recently validated certificates were cached and sender had to attach certificate only in some of the messages it originates. In a reactive certificate distribution sender does not attach its certificate in the message but only provides when explicitly required. Also in this approach a concept of multiple CAs was introduced. A trust relationship between two nodes trusting in two different CAs was established if there were atleast K nodes trusting both CAs. This was done by generating a certificate for CA1 and CA2 that was signed by k private-key-shares from CA1 and CA2 respectively. The authentication service proposed in the research was called as the MANET Authentication Extension (MAE) and was appended to each routing protocol packet. Hash functions and digital signatures were used to create MAE. MAE was mainly applied to the mutable field whose value changes as it gets forwarded to another node. In AODV, MAE was appended to RREQ and RREP as RREQ and RREP have mutable field (hop count) whose values changes as the message is forwarded from one nod to another. RERR was signed only by the node forwarding them. In DSR, MAE was appended only to RREQ and RREP was simply signed by the originator of the message [11].

The above researches were mainly focused on either implementing security in a MANET on-demand routing protocol or comparing AODV and DSR to get to a conclusion of which protocol is better in which environment. In my thesis, the first thing I did was implemented some security parameters using encryption techniques in AODV and DSR routing protocol. Then compared the secured protocols against each other to evaluate their performance and get to a conclusion which protocol performs better in which environment. I used RSA and SHA-1 cryptographic encryption schemes to maintain confidentiality, and integrity of the routing messages.

RSA is an algorithm for public-key cryptography. It is based on the generation of two large prime numbers “p” and “q” such that their product is “n”. A secret exponent is computed by doing mathematical calculation on “n”, “p” and “q” through which the public and private keys are generated. Routing messages will be encrypted and decrypted using these keys by the sender and the receiver respectively. I am using RSA because it is considered to be one of the strongest encryption techniques. RSA problem is based upon factorizing a number “n” in order to get the two prime numbers “p” and “q”. In a real world scenario “n” is considered to be very large such that factorizing “n” becomes impractical. The task of RSA is described as finding the e^{th} roots of an arbitrary number modulo “n”. The ability for computers to factor large numbers and attack RSA encryption is improving rapidly and there are systems that can find factors of prime numbers with more than 200 digits. Also there is no known factorization algorithm to factor large numbers derived by multiplying two large prime numbers. A test to factor a 200 digit number took 1.5 years and over 50 years of computer time [12].

Also in my thesis I have used SHA-1 for creating a message digest. Message digest ensures the integrity of the message. Routing messages will be run through SHA-1 algorithm to create a message digest. The receiver also runs the message through the SHA-1 algorithm and compares the message digest it has received. The message digest can then be given as an input for digital signature algorithm

(RSA in this thesis). SHA-1 is called secure because it is infeasible to find a message that corresponds to a particular message digest or two messages with the same message digest. Any changes to the message in transmit, would cause the message digest to change and the signature would fail to verify indicating the message has been modified [13].

As far as hashing and signatures are concerned, the process is slightly different in the AODV and DSR the way these cryptographic functions are implemented. In an ad hoc network when a node needs to send some data to a distinct node it needs to get the routing information in order to reach to the destination node. Therefore there are two kinds of messages: Routing messages and Data messages. Data messages can be protected by any point-to-point security system like IPSec. While routing messages gets processed, modified and resent by the intermediate nodes. The processing of these routing messages also causes the intermediate nodes to update their routing table in some cases. A need arises for the intermediate node to authenticate the information contained in the routing messages before they update their routing table. Also in some ad hoc networking protocols there are cases where some of the routing information is constantly changed on a hop-by-hop basis like the hop count in Distance-Vector routing protocols. The hop count is either incremented or decremented on a hop-by-op basis. Therefore we can conclude that the routing messages carry two types of information: Mutable; information that changes during transmission by the intermediate nodes and Non-Mutable; information that is never altered during transmission by the intermediate nodes. So there is a need to protect the mutable fields in order to get rid of the assumption “trust between intermediate nodes”.

AODV is a distance vector routing protocol. The “hop count field” in the AODV’s RREQ and RREP packet gets changed on every hop by the intermediate node. There could be a possibility that any malicious node in the network may decrement the hop count by 1 then the routing process may get disrupted and the original data will never get to the destination. Therefore hop count is protected by

creating hash chains using SHA-1. Other fields are non-mutable and can be protected by encryption and creating signatures using RSA for the intermediate node to authenticate itself to other nodes.

Hash chain was created by applying a one-way hash function repeatedly to a seed. A seed can be any random number generated, when there is a need to transmit new RREQ/RREP routing messages. Whenever a node needs to generate a RREQ or RREP message it does the following to create hash chain: 1. Generate a random number (seed). 2. Set the “Max Hop Count” field to the Time To Live value from the IP header i.e. Max Hop Count = Time To Live. 3. Set the “hash” field to the seed value i.e hash = seed. 4. Sets the “Hash Function” field to the identifier of the hash function i.e. Hash Function = h. 5. Calculates “Top hash” by hashing seed Max Hop Count times i.e. Top Hash = $h^{\text{Max Hop Count}}$(seed) where h is the hash function (SHA-1 in my case), $h^i(x)$ is the result of applying the function h to x "i" times. When any intermediate node receives a RREQ or RREP message, it performs the following operation to verify the hop count: 1. Applies the hash function h Maximum Hop Count minus Hop Count times to the value in the hash field, verifies if the resultant value is equal to the value contained in the Top Hash Field i.e. Top Hash = $h^{\text{Maz HopCount} - \text{Hop Count}}$(hash). 2. Before rebroadcasting a RREQ or forwarding the RREP, the node applies the hash function to the value in the Hash filed to account for the next hop. The “hash function” field in the signature extension indicates which hash function with be used to create hash chains.

Digital Signatures are used to protect the integrity of the non-mutable fields in the RREQ and RREP messages. All the fields are signed except for the hop count of the AODV message and the hash from the signature extension field. Signing the messages makes sure that the message was from a particular node and that node cannot deny on sending that message. Whenever an intermediate node receives an RREQ or RREP it first verifies the signature. If the signature is verified than only the node updates its routing table and creates a reverse route to that node. Also the messages are encrypted

before they are sent to the intermediate node. At each node the message is decrypted to process the message and then it is encrypted again before sending it to the next node. When there is a need to transfer data, the source generates an RREQ. To protect hop count, hash chain is created (top hash). Hop count is only in the RREQ and RREP packets. For other routing messages like RERR there is no need to create hash chains. The packet is encrypted, signed and send to the intermediate nodes. The next node grabs the packet and verifies the signature, verifies the hop count, decrypts the packet and before forwarding the RREQ it again calculates top hash, encrypts and signs the packet. Thus the packet is secured from malicious nodes.

DSR is based on Source Routing. Before sending the data the source node gets the route to the destination. Once it has the route to the destination it adds that route in its routing packets (RREQ, RREP, etc) and sends the data along with it. This route is the full route the packet takes from the source to the destination to get data delivered to its destination. Thus in DSR there are no mutable fields which makes life so much easy. DSR routing messages can be protected by encryption and signing the packets by each intermediate node for authentication. Signature and Encryption process is same as in AODV. The RERR message in both AODV and DSR does not have any mutable fields. They are signed and encrypted for protection from malicious nodes. The operation of DSR is the same as AODV except for the hash chains. In case of DSR there is no need to create hash chains, encryption and digital signing does the job of securing the packets. Figures 12a, 12b and 13a, 13b in the appendix section show the signature extension for AODV and DSR respectively.

An assumption in this thesis would be that each node in the network already has a certificate. The fields in the certificate would be certificate type, hash function type, CA identification, certificate serial number, IP address of the node, expiration date, exponent e (public key), modulus n (public key) and CA signature. Extensions will be made to the original AODV and DSR routing header packets to

include signature and hash. The signature and the hash will be appended to the AODV and DSR message. Additions to RREQ and RREP would be hash function, top hash, hash and last hop signature. RERR and HELLO message does not contain any mutable fields and so only signature field would be added to these packets.

4. Network Simulators

The nodes in the MANET network are aimed to accomplish mission critical tasks, detecting fire in a forest, collecting the rise in the level of water, in the battle field by a group of soldiers to communicate and many more. Also routing is a very challenging task in MANET networks due to the limited functionality and the mobility of the node. These nodes are battery powered and constrained by limited energy, are exposed to hazardous conditions and may get damaged. Therefore it is very important to carefully test, analyze and evaluate the routing protocol before implementing it in a real world scenario. This can be achieved by simulations. The most popular network simulators commercially available are, OPENT Modeler, NS-2, GloMoSim and the not so popular because of the advancement in technology are REAL, INSANE, NETSim and Meisie.

NS-2 is a discrete event simulator based on two languages C++ and OTcl. OTcl, is basically an object oriented version of Tcl. C++ is used to model the behavior of the protocol and OTcl is used by the user for configuration and simulating scripts. To implement a protocol in NS-2 first the protocol is implemented by creating bindings between the C++ and OTcl in such a way that each C++ class has a corresponding OTcl class accompanied to it. Then the scenario of the simulation is described in the OTcl script. Then we can simply run the simulation and analyze the generated trace files. Working with NS-2 is simply not easy because of the OTcl scripts. One has to learn many concepts related to Octl as it also used for configuring the scenario. Another disadvantage of NS-2 is that its code is not consistent. This is due to the fact it being an open source application. Also NS-2 doesn’t have enough features and

functions to describe, analyze or visualize the simulation scenario. The tools it has are written using scripting languages by different people and since there is a lack of general analysis tool we may get different values for the same metric. Also in NS-2 debugging is difficult due to the combination of C++ and OTcl scripts. There are issues related to the large amount of memory it uses. With a few hundred nodes simulations done with NS-2 has often lacked scalability.

GlomoSim was designed using parallel discrete-event simulation provided by C-based parallel simulation language, Parsec which provides a scalable simulation environment for wired and wireless networks. GloMoSim is built using a layered approach and standard APIs are also available which can be used between the different layers allowing rapid integration of models developed at different layers by the users. Configuration files like app.conf and Config.in are used to specify the network characteristics. The app.conf describes the application type, bit rate etc which relates to the generation of traffic. The Config.in file was used to describe the remainder of the parameters like simulation range, power range, mobility etc. This simple approach of GloMoSim is not as powerful as the OTcl of NS-2 but still it has proved that it is suitable for wireless simulation and reduces configuration complexity and confusion as compared to NS-2's OTcl. In GloMoSim a limited set of Ad Hoc routing protocols are been implemented. GloMoSim has also been reported of having programming bugs which leads to simulation crash. Researches sometimes had to do 1000 simulations in order to get a successful simulation and sometimes would never get a successful simulation. Another disadvantage of GloMoSim is that it measures values per node and not per packet which limits the statistics that can be collected from the simulation.

OPENT Modeller is a powerful network simulator which has the capability to simulate wired and wireless networks. It was originally designed for the use in companies to diagnose and reorganize the company network but we can also implement our own algorithm in it using the powerful set of tools and

functionality it provides. OPNET can perform discrete event simulation to analyze the system performance and behavior. With OPNET most of the deployment can be done with the Graphical User Interface. OPNET supports hierarchical model building. Network editor is used to create the network which defines the position of the interconnection and communicating entities; node and link. Each node in the network is represented by a node model. The node model basically is a collection of interconnected modules. The modules are either connected by a packet stream or statistics wires. There are two types of modules. The first type has a predefined characteristics and a set of built in parameters. Point-to Point transmitter, radio receivers and packet generators are examples of this type of modules. The second type contains highly programmable modules called processors or queues. Each node is described by a block structured diagram and the functionality of each programmable node is defined by a process model. Process Models are created using process editor which describes the logic flow and the behavior of queue and processor modules. The process model makes use of interrupts to communicate between processes. Proto-C language is use to build process models which consists of state transition diagrams, a library of kernel procedures and the standard C programming language. States and transitions define the progression of the process in response to events. Events are generated which causes a state change in the process model if certain conditions are either true or false. Within each state, logic can be created using the functions provided by the OPNET kernel library or even pure C programming language. Within a process we can create child processes to perform a sub-task which makes the former a parent process.

Once all the models are defined the performance and the behavior of the network model can be studied by exercising the models in a dynamic simulation. Before running the simulation the user should decide what type of information should be extracted from the simulation like application specific statistics and behavioral characteristics. The simulations can be run either from the OPNET GUI or by using the op_runsim utility program. Simulations can be used to generate different forms of output like

numerical data, animation and traces provided by the OPNET debugger. OPNET simulations support open interfaces to the C language and the host computer's operating system it can also generate output in the console window by printing the results required, can even generate binary or ASCII files and even interaction with other programs. We can also display graphs using the OPNET Analysis Tool. It represents the graph in the form of abscissa axis (horizontal) and ordinate axis (vertical). The plotting area can have one or more graphs describing the relationship between the variables mapped to the two axis [18] [19] [20] [21].

5. SHA-1

SHA-1 is a hashing algorithm used to compute condensed representation of a message or a test or a data file. It takes a message of length < 2^{64} bits as an input a produces a 160 bit output called a message digest. Message digest ensures the integrity of the message. This message digest can then be used for creating signatures. The message digest is given as an input to the signing algorithm like Digital Signature Algorithm. DSA signs the message digest instead of the original message which is way better as the message digest is usually smaller than the original message. The original message will be run through SHA-1 algorithm to create a message digest. The receiver also runs the message through the SHA-1 algorithm and compares the message digest it has received. SHA-1 is called secure because it is infeasible to find a message that corresponds to a particular message digest or two messages with the same message digest. Any changes to the message in transmit, would cause the message digest to change and the signature would fail to verify indicating the message has been modified. SHA-1 is also irreversible i.e. given a message digest it is computationally infeasible to find the original message. Being said that it means that the hash does not contain enough information that would find the original text message, and it would take vast amount of resources and time to find a message from its corresponding hash value.

SHA-1 is widely used in a number of security applications electronic fund transfer, electronic mail, software distribution, data storage and other application which require data integrity assurance and data origin authentication. It is also used in protocols, including TSL, SSL, PGP, SSH, S/MIME and IPSec. Apart from these applications it can also be used whenever there is a requirement to generate a condensed form of the original message. SHA-1 was considered to be so much secure that it was used in US Government applications and in cryptographic algorithms and protocols for the protection of sensitive unclassified information Private and commercial organizations were also encouraged to adopt and use SHA-1. Though SHA-1 is secure enough, the US National Institute of Standards and Technology said that Federal agencies should stop using SHA-1 for applications that require collision resistance and must use SHA-2. This because in February 2005, a team of Chinese Cryptographers Xiaoyun Wang, Yiqun Lisa Yin, Bayarjargal, and Hongbo Yu announced that they found a collision in the full version of SHA-1 that requires 2^{69} operations which is fewer than the brute-force search. Collision in a hash function basically means two inputs with the same hash. The brute-force search for a hash with "L" being the number of bits in the message digest, to find a collision $2^{L/2}$ operations would be required. So for a SHA-1 message digest which is 160 bits the brute-force search would take 2^{80} operations. The Chinese team imposed an attack on SHA-1 and claims to break it in fewer than 2^{69} operations [22]. The team further added that "their analysis was built upon the original differential attack on SHA-0 [*sic*], the near collision attack on SHA-0, the multi-block collision techniques, as well as the message modification techniques used in the collision search attack on MD5. Breaking SHA-1 would not be possible without these powerful analytical techniques." On 17th Aug 2005 an improvement was found by another team of cryptographers Andrew Yao and Frances Yao at the CRYPTO 2005 rump session, lowering the complexity required for finding a collision in SHA-1 to 2^{63} [23]. Since then researchers have been trying to improve the attack on SHA-1 and another attempt was made in 2008 by Stephane Manuel who could produce hash collisions with estimated theoretical complexity of 2^{51} to 2^{57} operations [24]. Even though

researchers claim that there is a weakness in SHA-1 by improving the complexity requirement to 2^{51} operations, it still a big number to calculate and would require a lot of resources and power to compute such a value. Therefore it is infeasible to calculate such a large value making SHA-1 still secure. Also the researchers do not talk anything about the irreversibility feature f SHA-1. The irreversibility of SHA-1 still stands where it is. As of today there are still many applications that rely on SHA-1 making it popular and widely used hash algorithm.

6. RSA

RSA is an algorithm for public-key cryptography. It is based on the generation of two large prime numbers “p” and “q” such that their product is “n”. A secret exponent is computed by doing mathematical calculation on “n”, “p” and “q” through which the public and private keys are generated. An integer “e” (also known as the public exponent or the encryption exponent) is selected such that, (1<e<phi) where phi = (p-1)*(n-1) and also the gcd(e,phi) = 1. Then compute the value of “d” (also known as the private exponent or the decryption exponent) 1 < d < phi such that ed = 1(mod phi) or d=(1/e) (mod phi). Thus we get the public and the private keys. Public keys (n,e) and private keys (n,d). The sender computes the cipher text by the following equation; $c = m^e \bmod n$ where “m” is the hash of the original message. For decryption the receiver computes the value of d runs the cipher text through the following equation $m = c^d \bmod n$. This equation extracts the hash value of the corresponding message. Digital signing of the message is done by the following equation; $s = m^d \bmod n$ where “m” is the message digest of the original message such that integer “m” is between 1 and n-1. For signature verification the message digest is extracted by the following equation; $v = s^e \bmod n$. The receiver then independently computes the message digest of the information that has been and signed and compares message digest. The signature is considered to be valid if both the message digest are identical.

I am using RSA because it is considered to be one of the strongest encryption techniques. RSA problem is based upon factorizing a number “n” in order to get the two prime numbers “p” and “q”. In a real world scenario “n” is considered to be very large such that factorizing “n” becomes impractical. The task of RSA is described as finding the e^{th} roots of an arbitrary number modulo “n”. The ability for computers to factor large numbers and attack RSA encryption is improving rapidly and there are systems that can find factors of prime numbers with more than 200 digits. Also there is no known factorization algorithm to factor large numbers derived by multiplying two large prime numbers. A test to factor a 200 digit number took 1.5 years and over 50 years of computer time [25]. The biggest advantage of RSA is that it uses public key encryption. The text message will be encrypted with someone’s public key which is known by everyone. Only the person intended to receive that text message will be able to decrypt it by using its private key which only that person is aware of. This text message cannot be decrypted by the public key. Also the signing of messages by RSA helps the recipient verify that it was sent by the person it was supposed to get from.

The fields for the signature and hash are also called as the extension fields of the signature extension. The signature extension fields were added to the traditional AODV and DSR packets as an extension to the original packet. The signature extension fields for AODV and DSR are shown in the Appendix section.

7. Methodology

The comparison of the two protocols would be done using OPNET modeler. A rectangular field would be used for simulation. The main interest of this thesis would be to get an idea about which protocol performs better in the presence of security encryption techniques. The simulation will be done under different types of environment like varying number of nodes, packet sizes or traffic load and effect of mobility to get the following performance parameters: average end-to-end delay, routing load

and packet delivery fraction. RSA and SHA algorithms would be implemented and hash, and signatures generated by these algorithm will be appended to the AODV and DSR message. Each node will possess a certificate to maintain its identity in the network. As the routing messages are transmitted on a hop by hop basis the nodes check for the changes in the hash value by comparing it with the original hash value. The signature in the routing message helps other nodes to identify the sender of the message. Also nodes in the network cannot deny on sending a particular message. The routing messages will contain the signature of the originator of the message (i.e. RREQ/RREP) and the signature of the last hop.

I started with reading scholarly articles and academic research papers. Google was of great help in finding such articles and I browsed through the ACM and IEEE websites. The research area of my interest was any work done related to AODV and DSR. As I was going through the websites I saw a huge number of papers dedicated only to AODV and DSR amongst all the MANET routing protocols. This was evident that AODV and DSR were amongst the most widely used MANET routing protocol in research as well as in other applications. Before I started the actual research I made sure I had a solid understanding of the operations involved in these protocols. There were papers about improving the performance of these protocols. Some compared them, some created a fancy protocol out of the traditional protocol that would best operate for a particular type of application, and some even implemented security parameters in AODV. People had presented the idea of implementing security parameters in DSR but there was no research that actually secured the DSR protocol. I read a decent amount of papers and got a good understanding about the current researches going on with AODV and DSR.

Since I had decided to use OPNET simulator to compare the performance of the protocols I had to learn about how OPNET works. I browsed through the example networks section and opened a scenario for AODV. At the top level there is a network topology. The network topology is comprised of subnets. The basically comprises of devices that has wireless communication link between each other.

The devices are represented by a node model. The node model is comprised of different types of modules like; Processor Modules, Queue Modules, Esys Modules, Transmitter Modules and Receiver Modules. Each module in the node model has a process model which is nothing but an event driven state transition diagram. The states may have logic implemented in them for it to perform a particular task in case an event triggers that state. The process model may also have child processes associated to them. So since this was a routing protocol it has to be somewhere in the IP module. I opened the process model for "IP" and opened child process model which took me to the MANET process model. Within the MANET process module there were several other process modules which had the routing protocols process model. I opened the AODV process model and opened the function block which had all the logic related to the operation of the AODV protocol. I went through the program code and got a decent understanding about the flow of the code that would help me implement my security code in it. OPNET's tutorial was of great help. It gave me a good understanding about the rich set of in-built functions OPNET had. I then started developing an algorithm to implement SHA-1. Then I implemented that algorithm in a C programming code. I tested the SHA code in a simple environment outside of the OPNET. Then I added my code to the OPNET code within the function block. I also added the required function prototype to the project header files. I compiled the code in the OPENT environment. Fixed the issues and ran the simulation. The simulation run was successful. The hash values were printed to the OPNET console. Also the simulation time went up when I added SHA-1 code to the original code. Then I adjusted the packet fields for AODV to add the hash field in the packet. This was done in a C file associated with the project. Added additional parameters to the function call for the routing messages. SHA-1 test run was successful and it was working fine. Then I started working on the algorithm for RSA. I implemented the algorithm in a C code. The code was compiled and run in a simple environment. Errors were fixed and test run was Ok. Made sure the decryption and the signature verification worked. Then I added the RSA code to the OPNET code. Fixed the errors and made sure it worked as a whole with the

project. Added the AODV extension fields for signature to the appropriate C file associated with the project. Added additional parameters to the function call for the routing messages. Again the simulation time went up after adding the RSA code. Results of encryption/decryption and signature/signature verification were displayed on the OPNET console. Performed a test run, both encryption/decryption and signature/signature verification worked flawless. Thus one model for AODV was successfully completed with encryption parameters implemented. For DSR it was pretty simple. I just had to figure out where to add the encryption code. Add it and do test runs. The test runs were successful. Encryption/decryption and signature/signature verification were working fine. The results were displayed on the OPNET console. Thus the two models were ready for comparison.

The comparison was done by running the protocol over an application. OPNET already has in built applications that we can select and have the application run on the protocol. This can be done by the "application config" object in the object palette. To edit the attributes for the application config object by selecting the "Application Definition" field and then selected the number of rows. The number of rows corresponds to the number applications to be added. If two rows are added two fields for the applications will pop up. I am using FTP as an application. We can also specify the type of application load we want to run like low, medium and heavy load. Then we need to create a profile for the application. The profile defines the run time parameters for the application. This can be done using the "Profile Config" object from the Object Palette. Right click on it to set the attributes. Click on the "Applications" field and select the type of application which in my case I selected FTP. I named the application as FTP. Then here we can also have multiple FTP sessions or different type of applications by selecting the number of rows. The fields that determine the run time parameter for the defined applications are as follows:

1. *Start time offset*: The value set in this field depends on the type of "Operation Mode" specified. If the operation mode is set to "Simultaneous" the offset refers to the first instance of each application from the start of the profile. If the operation mode is set to "Serial (Order)" or "Serial (Random)" then this offset refers to the time from the start of the profile to the start of the first application.
2. *Duration*: It defines the amount of time for the application session runs. If it is set to "End of Profile" it lasts until the simulation time expires. If it is set to "End of Last task" it lasts until the last task of the application has completed.
3. *Repeatability*: This attribute specifies the method for repeating the profile. Inter-Repetition defines when the next session of the profile starts. Number of repetitions determines the distribution name and arguments to be used for generating a random number of times this profile is repeated. Repetition Pattern defines when the next session of the profile will start.
4. *Operation Mode*: This field determines how the application starts. If it is set to Serial (Ordered) then applications starts one after another in an ordered form. If it is set to Serial (Random) then the applications starts one after another in a random order. If it is set to simultaneous then all the applications will start at the same time.
5. *Start Time*: This attribute determines when during the simulation the application session will begin.

The defined application runs depending on the values set in these fields. This is how an application was assigned to a profile that determines its run time parameters during the simulation. The two protocol models were configured with the same application type and the parameters to run the application. Results were collected with different application parameter set to different value to

compare the two protocols in different scenarios. Also the results were collected considering the number of nodes in the network. The simulation was run with different number of nodes to get to a conclusion about how the application run on the protocols affect the overall performance of the network. Depending on the results a conclusion was made about which protocol performs best in which environment.

8. Simulation Results

Simulation results were collected by running FTP application over the two routing protocols. Before running the simulation I had to specify the desired output values. This can be done by selecting DES -> Choose Individual Statistics from the menu bar of the project editor. I selected AODV, FTP and Wireless by expanding the Global Statistics tab. Global statistics accounts for the performance of the network as a whole.

Scenario1: 50 nodes network, medium FTP traffic: The simulation run time was 5 minutes. For AODV it took 5 minutes 35 seconds to complete the simulation. For DSR it took 2 minutes 20 seconds to complete the simulation. Figure 3a and 3b represents the output from AODV and DSR respectively. The graph shows the results for the FTP Upload Response Time and Download Response Time. Looking at the graphs for both protocols I get to a conclusion that DSR performs way better than the AODV protocol in this scenario. DSR performs more than 50% better than AODV. In both cases DSR is faster than AODV. Figure 4a and 4b represents delay for AODV and DSR. The graph shows that delay is better in DSR than AODV. Figure 4c and 4d represents the throughput for AODV and DSR. The graphs show that AODV has a better throughput. Figure 5a and 5b represents the routing traffic in AODV and DSR respectively. Routing load is nothing but the number of routing packets generated to be sent to deliver a data packet. In AODV more routing traffic is generated than DSR for the same piece of data to send to the destination. Therefore DSR is better than AODV in terms of routing traffic being generated. Packet

delivery fraction is defined as the ratio of packets delivered to that generated by the traffic generator. Packet delivery fraction was better in AODV as compared to DSR. The reason for this is DSR being very aggressive in caching routes.

Scenario 2: 32 nodes network, low FTP traffic: The Simulation run time was set to 5 minutes and it took 4 minutes 27 seconds for AODV and 1 minute and 6 seconds for DSR to complete the simulation. Figure 6a and 6b represents the output for FTP upload and download response times for AODV and DSR respectively. DSR takes much less time to upload and download FTP traffic as compared to AODV. Figure 7a and 7b represents delay for AODV and DSR respectively. The graph shows that AODV has more delay than DSR. Figure 7c and 7d shows the output for throughout in AODV and DSR and again the throughput in AODV is a little better than DSR. Figure 8a and 8b shows the output for the routing traffic for AODV and DSR respectively which says that in case of AODV more routing traffic is generated. With the nodes reduced to 32, the packet delivery fraction for AODV was just a little better than DSR. Since the nodes got reduced to 32, it had to do less caching as compared to earlier with 50 nodes in the network.

Scenario3: 20 nodes, 5 mobile nodes, high FTP traffic: The simulation run time was 5 minutes and it took 4 minutes 48 seconds for AODV and 2 miniutes 34 seconds for DSR to complete the simulation. Figure 9a and 9b shows the FTP upload and download response time for OADV and DSR respectively. Again here DSR is better than AODV. Figure 10a and 10b shows the delay for AODV and DSR respectively. Here again with DSR the delay is less compared to AODV. Figure 10c and 10d represents the throughput for AODV and DSR, which shows that the throughput is little better in AODV than DSR. Figure 11a and 11b shows the routing traffic which is nothing but the routing load for AODV and DSR respectively. Again DSR generates less routing traffic as compared to AODV to send the same amount of data. With 20 nodes in the network the packet delivery fraction in DSR was better as compared to AODV. Since there

were only 20 nodes there was less caching and the number of stale routes in the routing table was also reduced, resulting into less number of packet drops.

9. Conclusion

The aim of this thesis was to compare the two reactive routing protocols such that security is implemented in them to protect the network from being compromised. I created security encryption techniques SHA-1 and RSA in AODV and DSR. OPNET simulator was used for comparing the performance of the routing protocols. The protocols were compared under different scenarios like number of nodes, mobility, and type of traffic. Results for all the scenarios are shown in the graphs which are in the Appendix section.

Looking at all graphs of these different scenarios I get to a conclusion that DSR performs than AODV because of the fact that in AODV there is a mutable field (hop count) which needs to be protected. Protecting the hop count is costly in terms of processing because the first node had to create a hash chain which results into hashing the seed (a random number) max hop count time. So the bigger the max hop count value the more it hashes and the more costly AODV becomes to protect when performance is concerned. Though in AODV we get better throughput than DSR because DSR makes aggressive use of caching to facilitate multiple routing path to the destination . Also the packet delivery fraction in AODV is better than DSR in larger networks. DSR is better though with smaller networks. Aggressive route caching of DSR populates the routing table with stale routes. There is no mechanism in DSR to determine the freshness of route. The metric use to determine the best route is the hop count. So if a stale route with a less number of hop count gets selected, it results into packet drop and decreases the packet delivery fraction ratio.

10. Deliverables

- Comparison results for AODV and DSR
- C++ programming code and the pseudo code
- Graphs from the simulation
- Thesis report that document the whole process of encryption scheme and performance of the two protocols

11. Research Timeline

Proposed Timeline of Thesis process: 2010-2012

Nov 12 to Nov 25 – Research for Thesis Idea begins
Read published research papers

Dec 1 – First Preliminary meeting with Prof. Tom Oh

Dec 2 – Dec 09 – Study OPNET and MANET

Dec 10 – Turn in the Thesis Proposal to NSSA
Situation Analysis
Problem Statement

Dec 11 – Tentative first committee meeting

Jan 2011 (Week1-Week4) – Experimental Research and Analysis

Feb (Week1 – Week3) – Situation Analysis
Problem Statement
Mission Statement
Goals

Objectives
Timeline
Evaluation Plan
Bibliography
Glossary
Appendix

March (Week 3) – Review Progress with Committee

Apr (Week 1) – Gather extended information

May (Week 1 - 4) – Develop code for SHA-1

June (Week 1 - 4) – Develop code for RSA

July (Week 1 – Week 4) – Add code to OPNET

Aug (Week 1 – Week 4) – Add code to OPNET

Sept (Week 1 - Week 4) – Add code to second model of OPNET

Oct (Week 1 – Week 4) – Work on getting results

Nov (Week 1 – Week 4) – Gather data for report writing.

Dec (Week 1 – Week 4) – Report Writing
– Submit first draft to committee members
– Review the comments and as per the committee
– Submit Second draft to committee members
– Prepare for Presentation

Jan 06 2012 – Thesis Defense

12. Appendix

Results of the Simulation

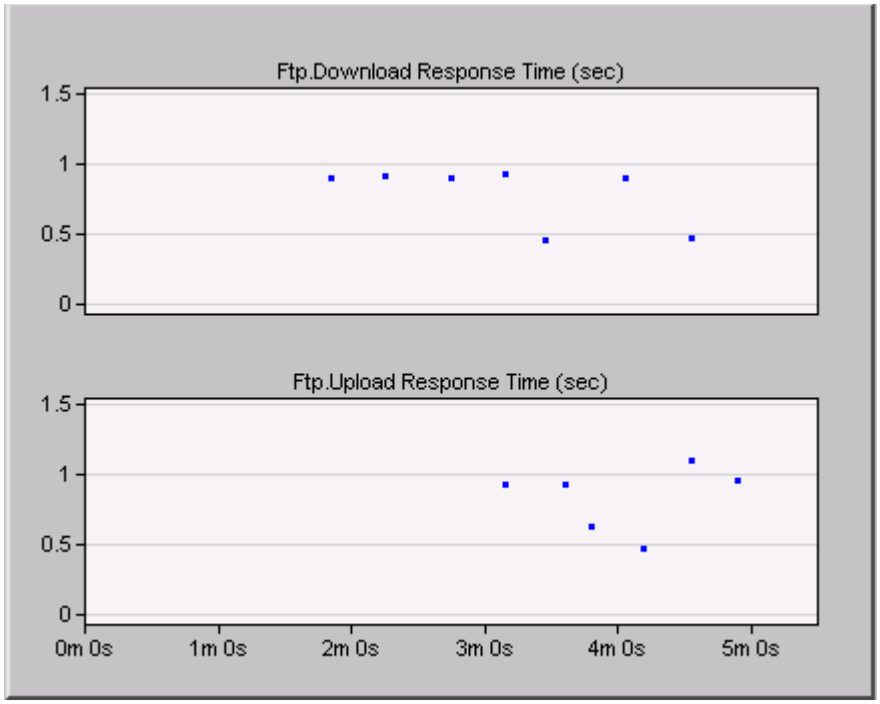


Figure3a. AODV FTP traffic

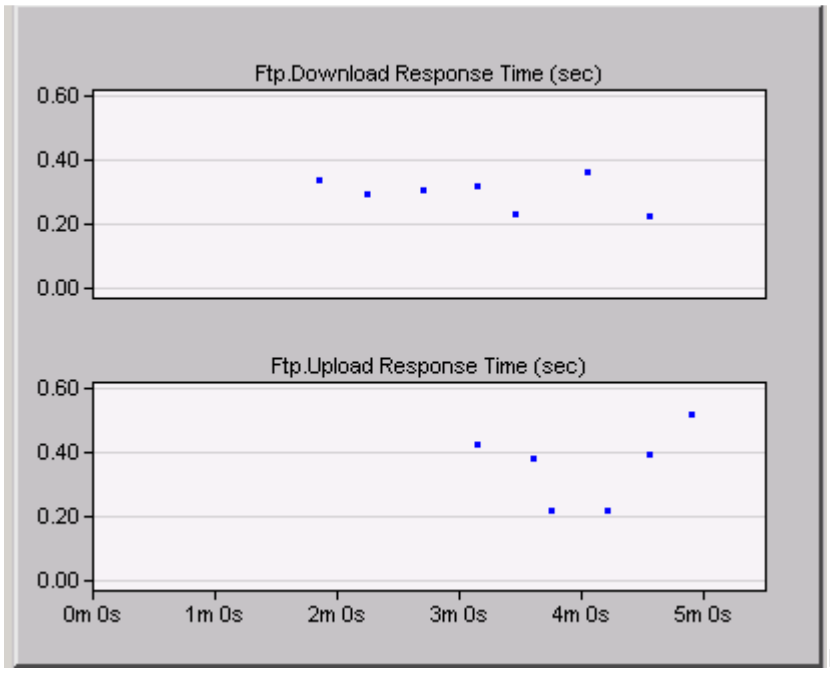


Figure 3b: DSR FTP traffic

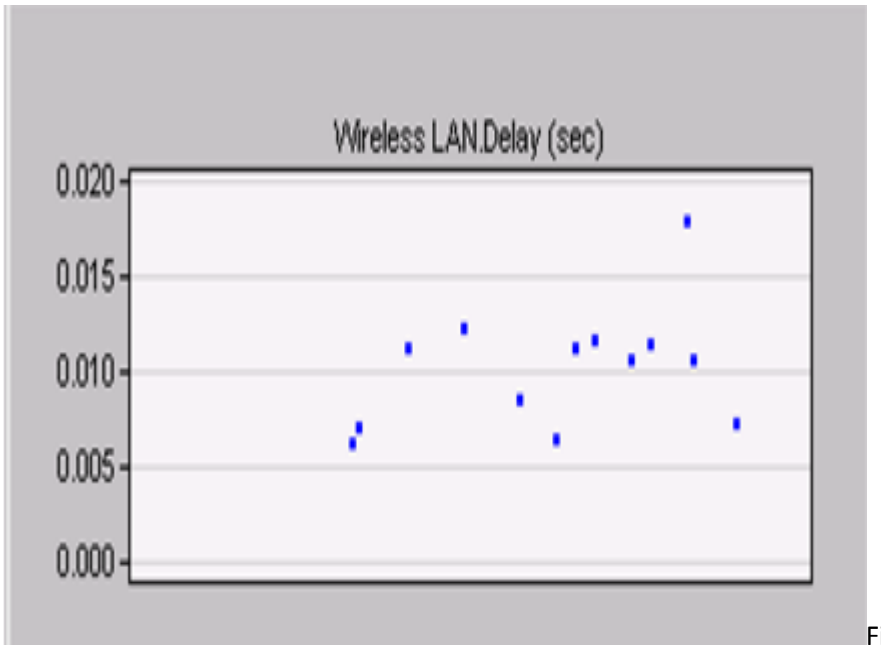


Figure 4a: AODV delay

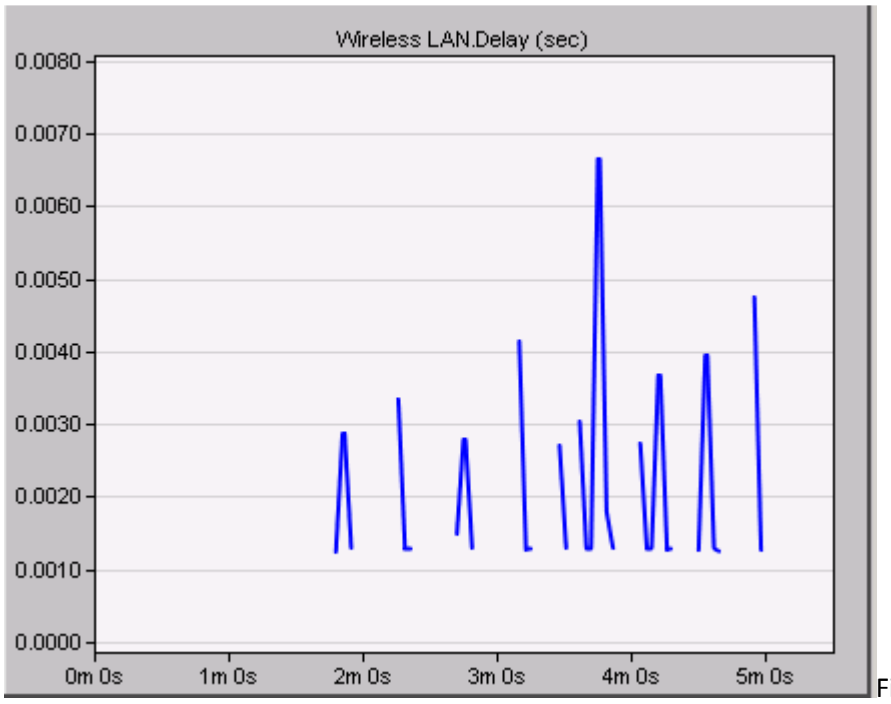


Figure 4b: DSR delay

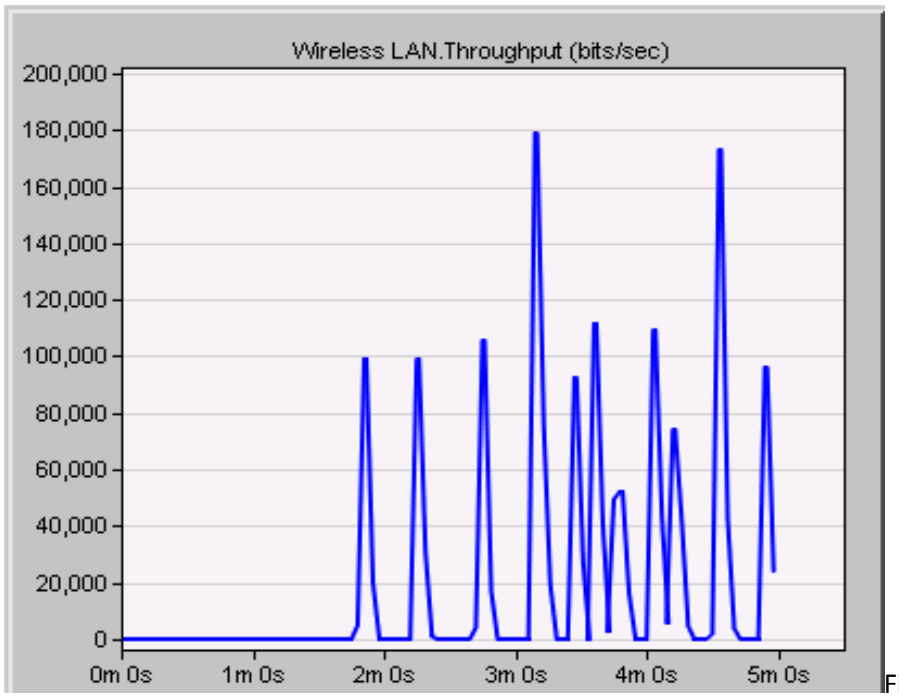


Figure 4c: AODV throughput

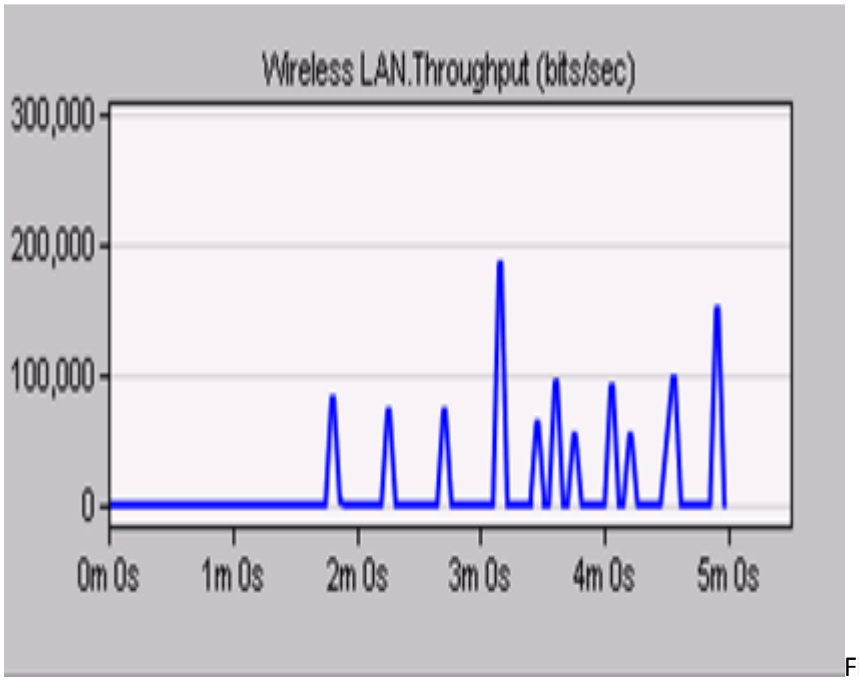


Figure 4d: DSR throughput

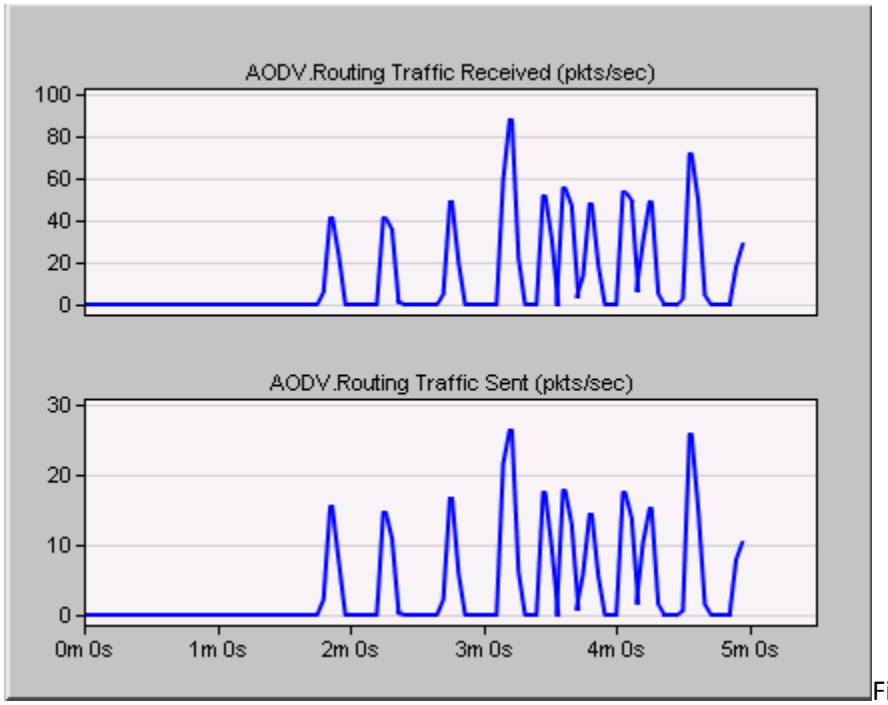


Figure 5a: AODV routing traffic

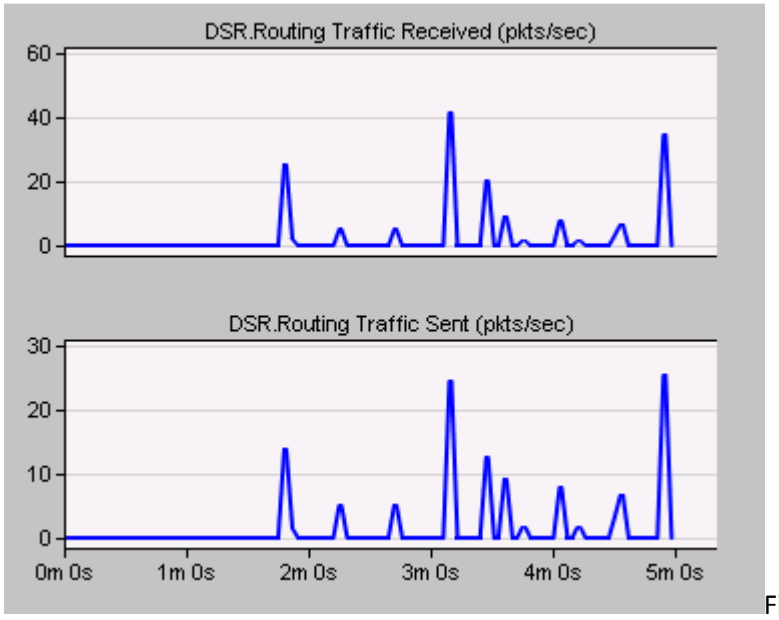


Figure 5b: DSR routing traffic

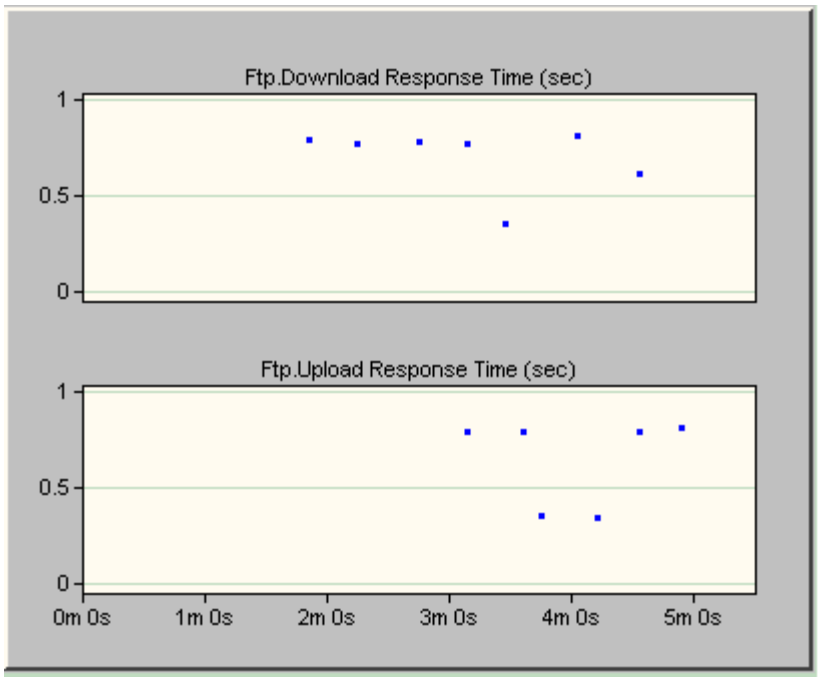


Figure 6a: AODV FTP traffic

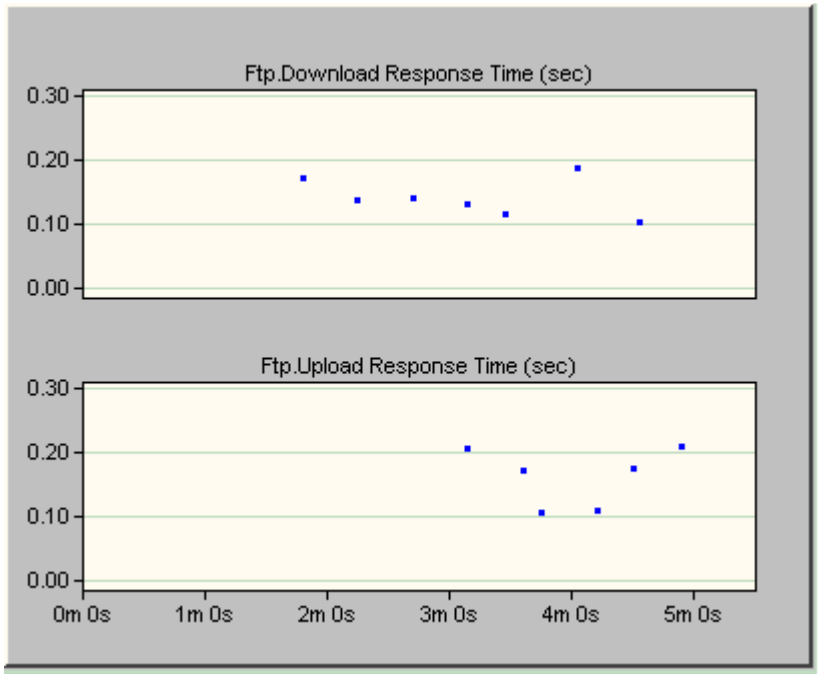


figure 6b: DSR FTP traffic

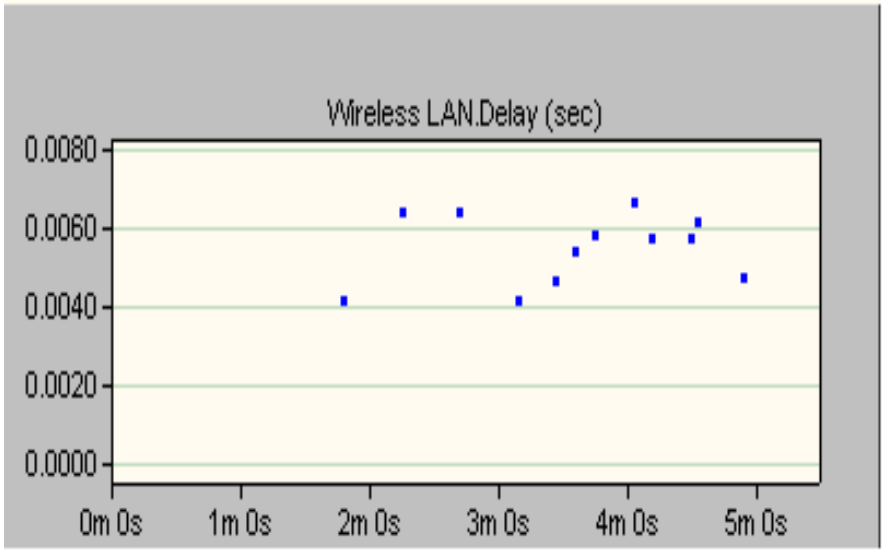


Figure 7a: AODV delay

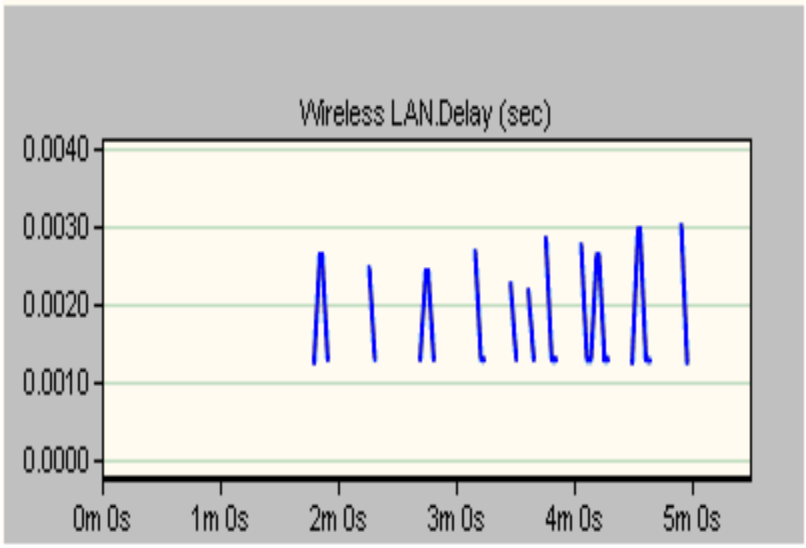


Figure 7b: DSR delay

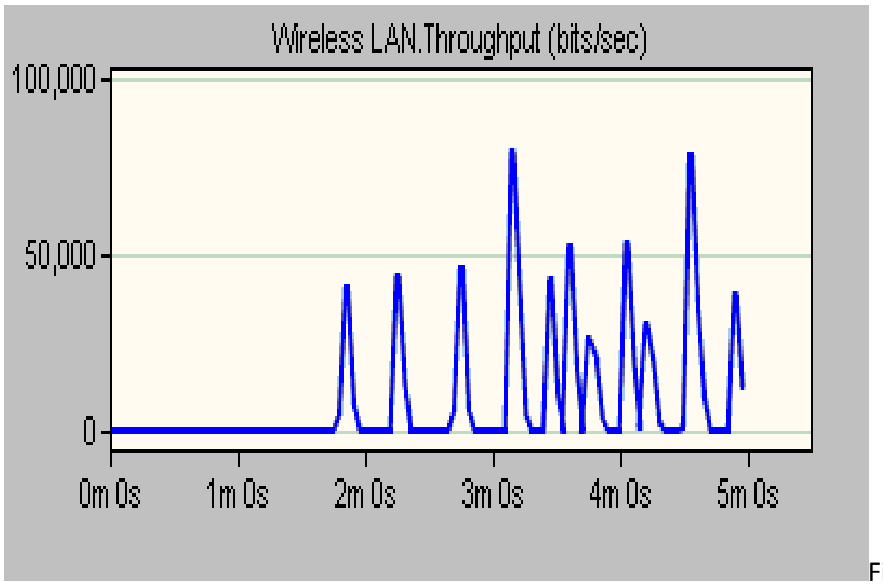


Figure 7c: AODV throughput

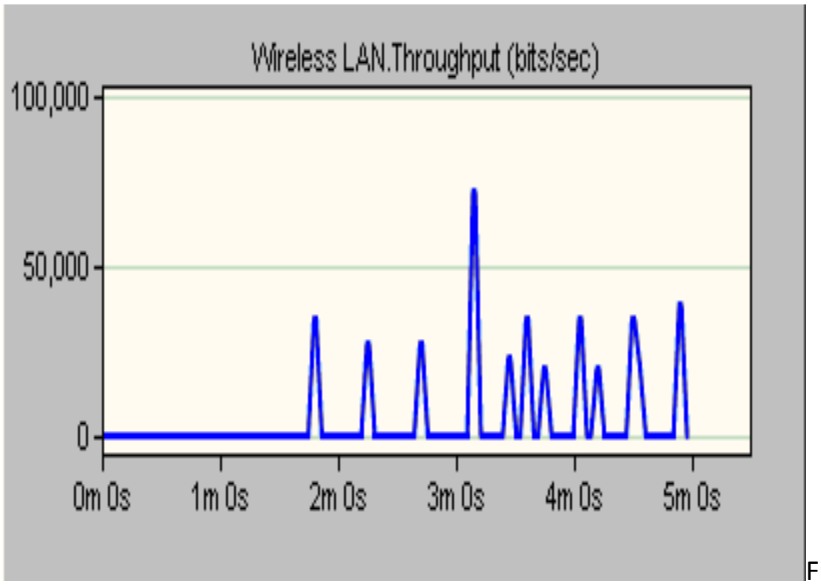
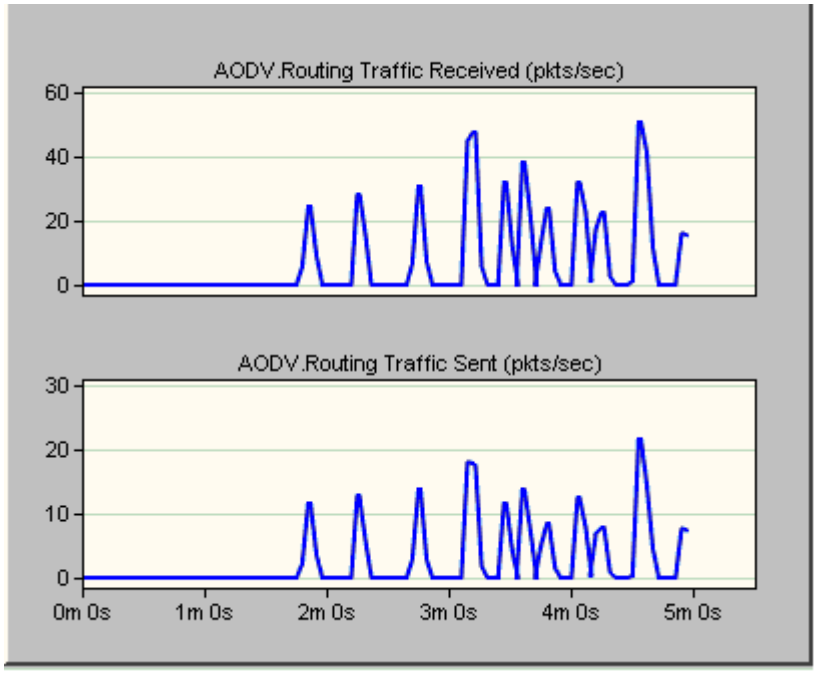


Figure 7d: DSR throughput



Figue 8a: AODV Routing traffic

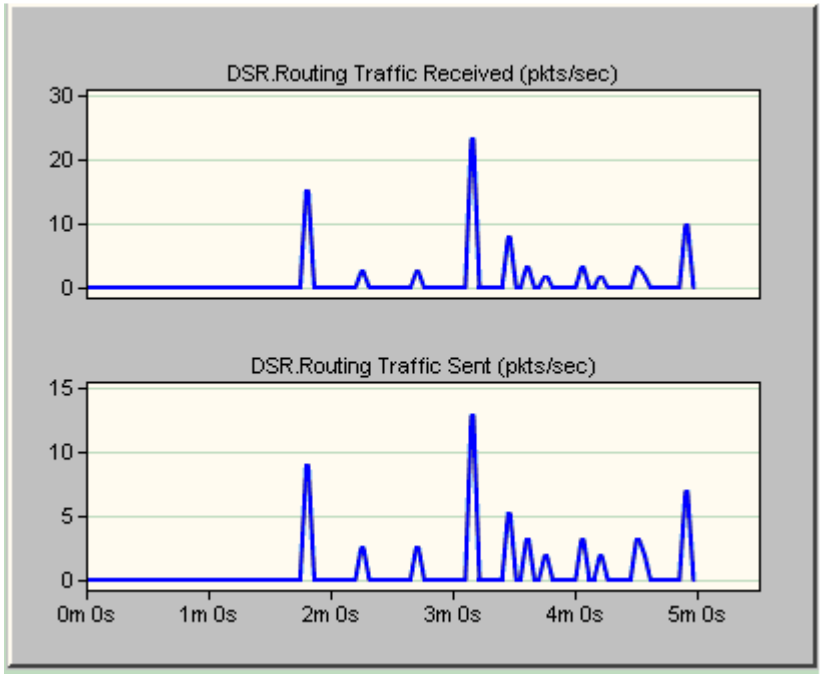


Figure 8b: DSR routing traffic

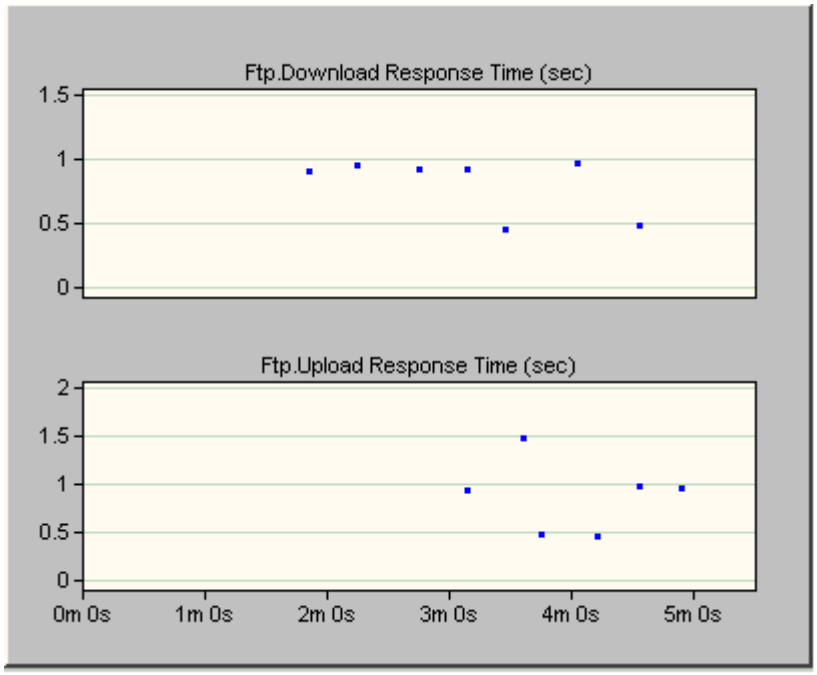


Figure 9a: AODV FTP traffic

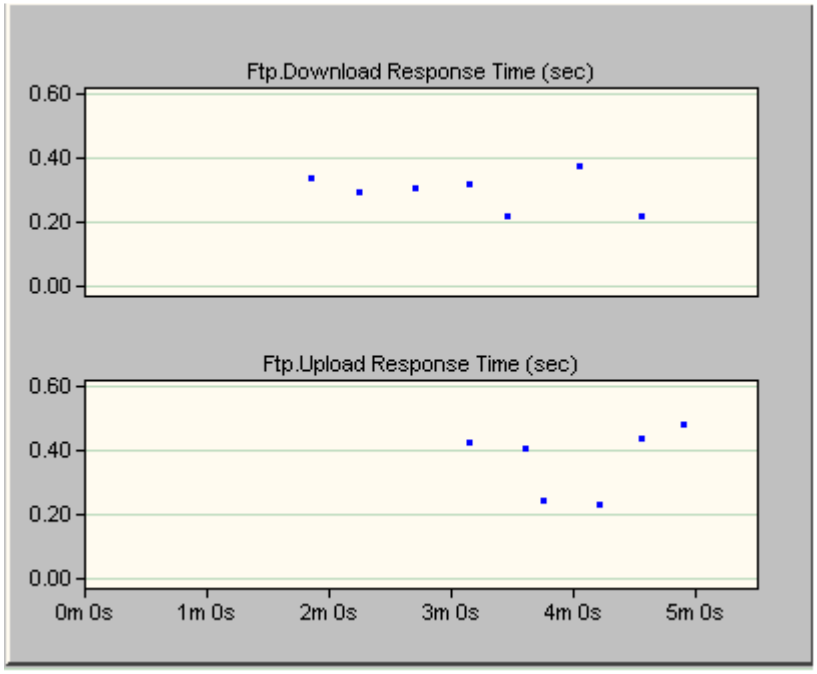


Figure 9b: DSR FTP traffic

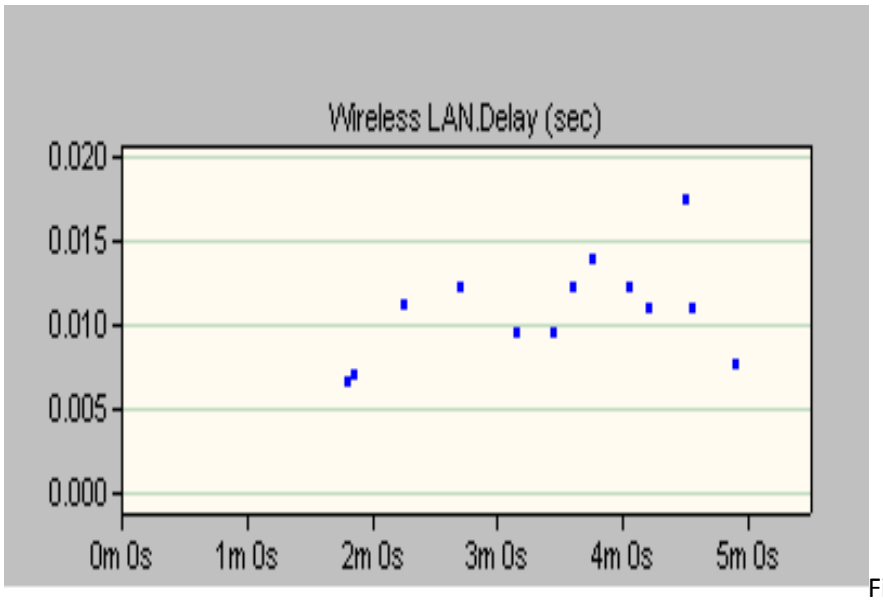


Figure10a:AODV delay

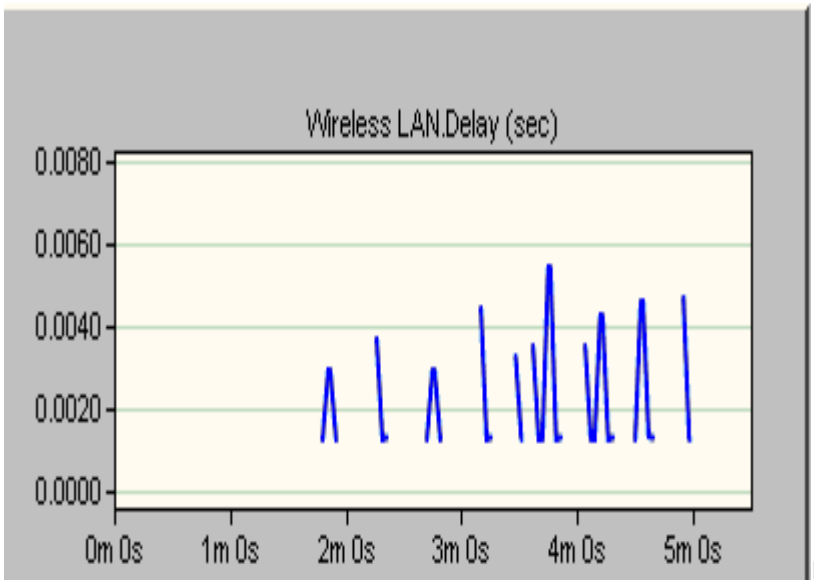


Figure10b:DSR delay

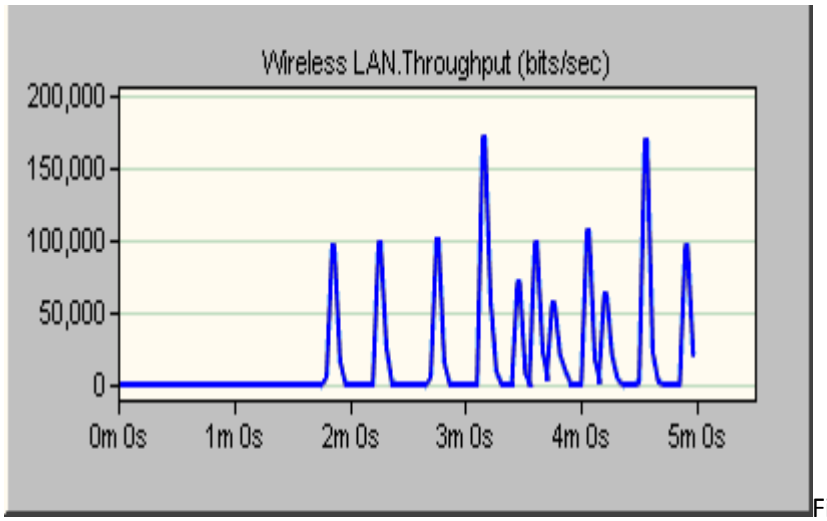


Figure 10c: AODV throughput

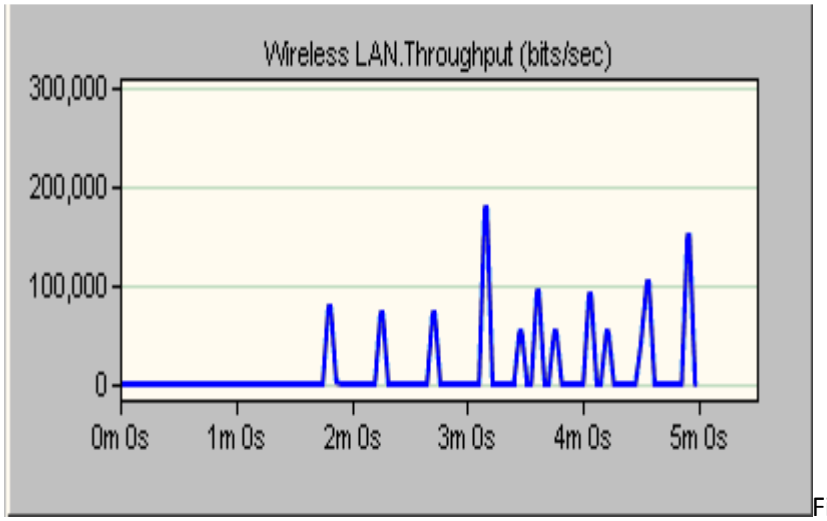


Figure 10d: DSR throughput

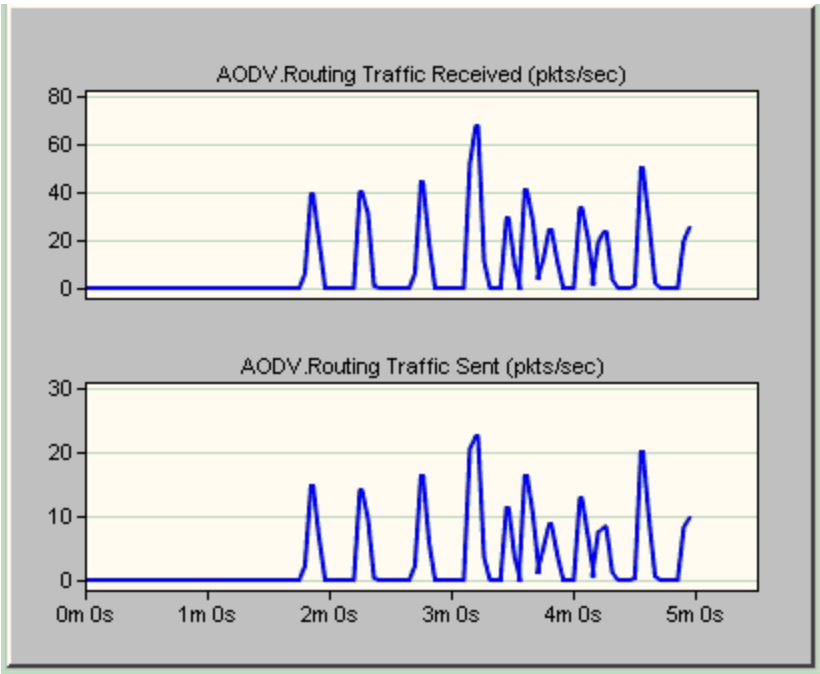


Figure11a: AODV Routing traffic

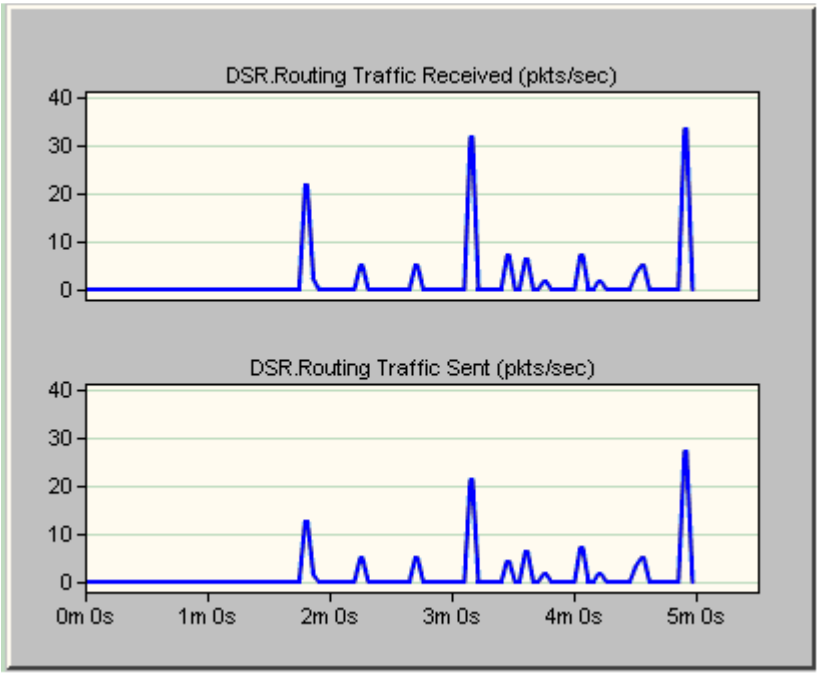


figure11b: DSR Routing traffic

Signature Extension Packet Formats

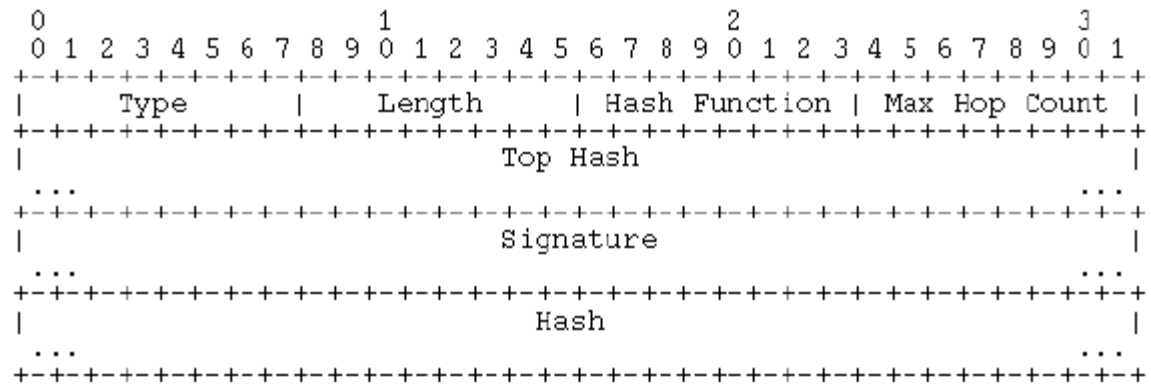


Figure 12a: AODV RREQ Signature Extension

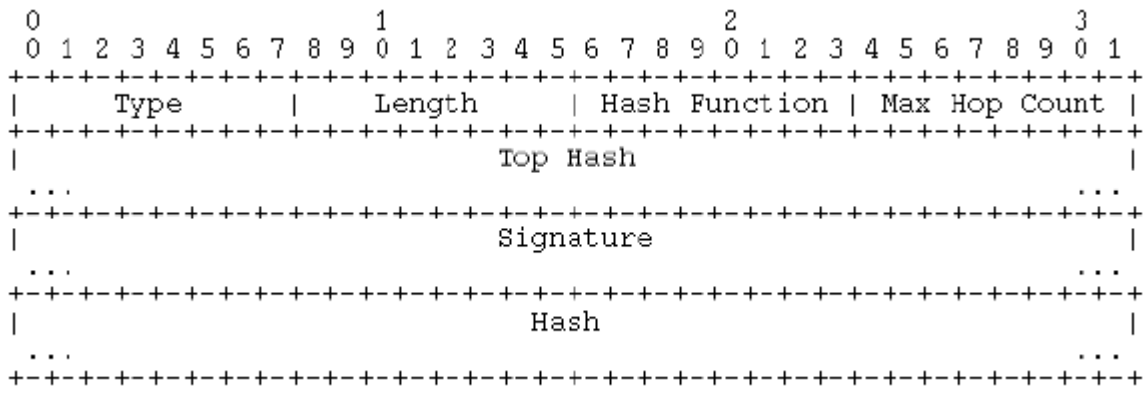


Figure 12b: AODV RREP Signature Extension

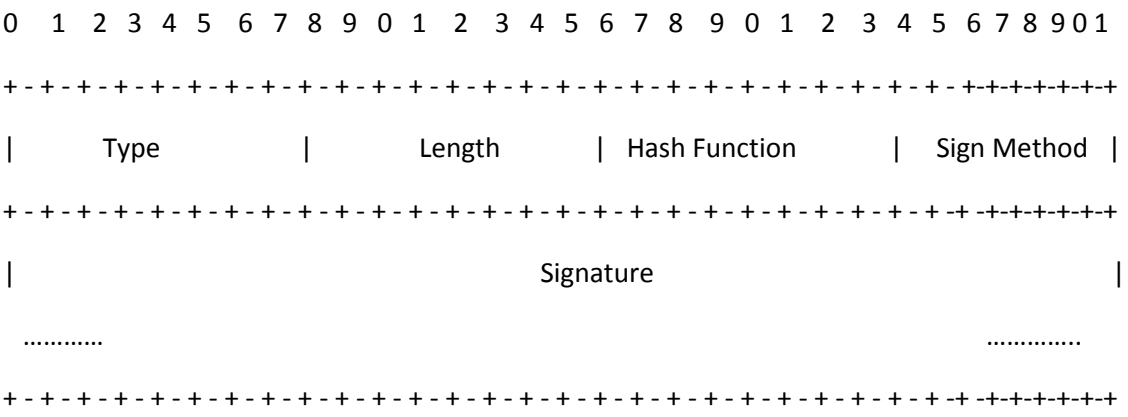


Figure 13a: DSR RREQ Signature Extension

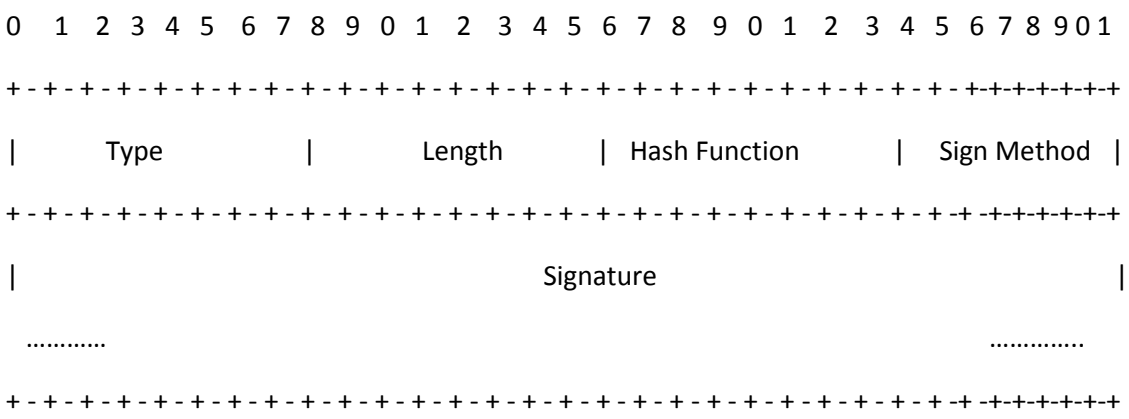
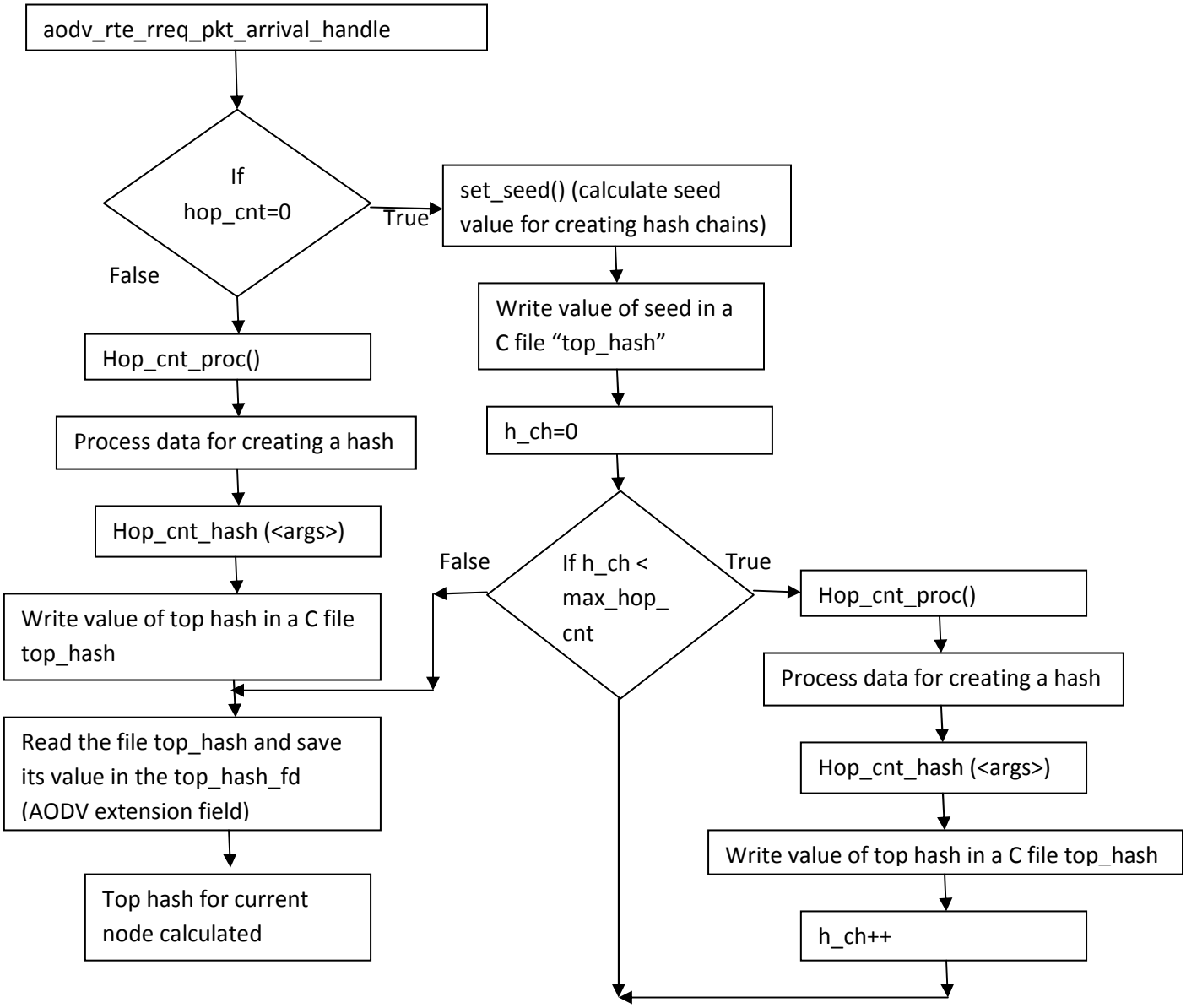


Figure 13b: DSR RREP Signature Extension

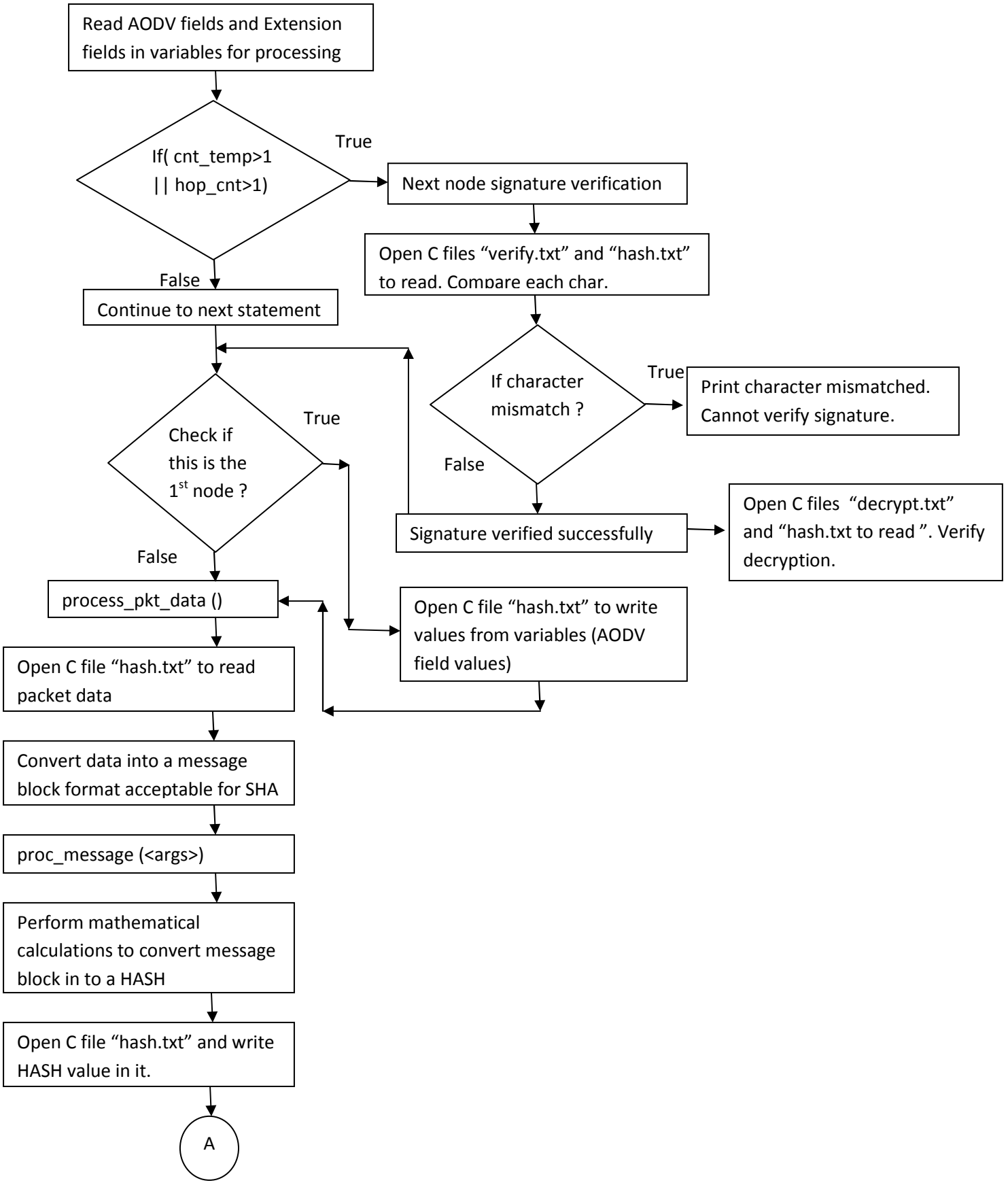
Signature Extension Fields:

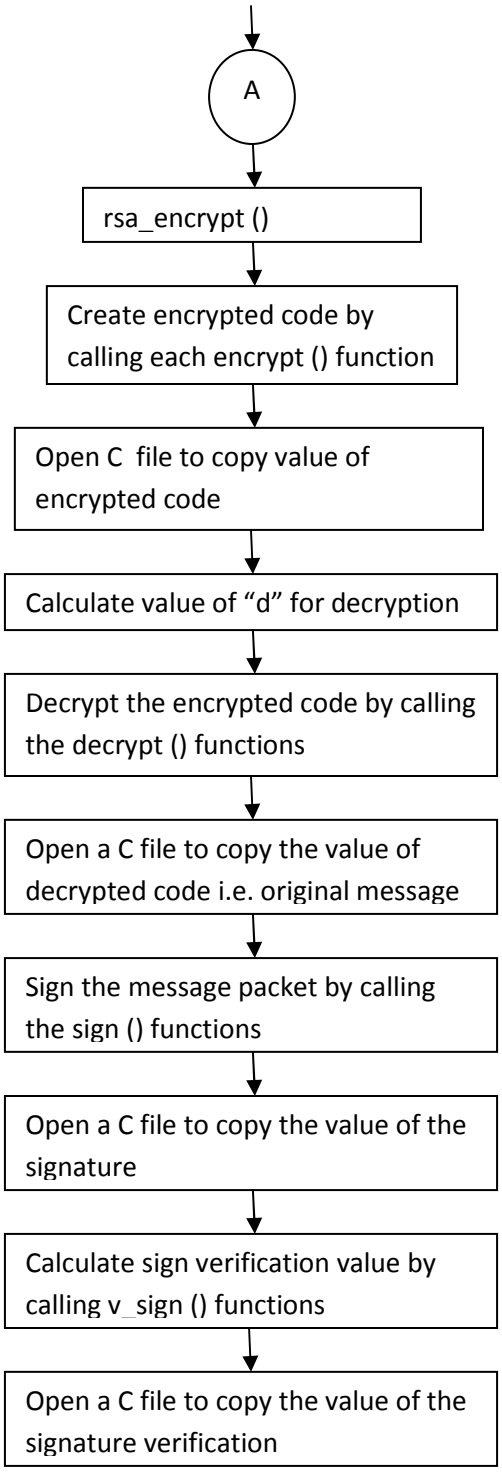
1. Type: Type represents if it is a RREQ or RREP extension. It is set to 64 for RREQ and 65 for RREP.
2. Length: Length represents the length of the packet excluding the type and Length fields in the packet extension.
3. Hash Function: Specifies the type of hash function to use. If it is set to 1, it usesMD5 as the hash function. If it is set to 2, it uses SHA-1 as the hash function.
4. Max Hop Count: The maximum hop count supported by the hop count authentication.
5. Top hash: It is the top hash for the hop count authentication in AODV.
6. Signature: This field contains the signature of the node. The node signs all the fields but the hop count and the hash.
7. Hash: The hash for the actual hop count.

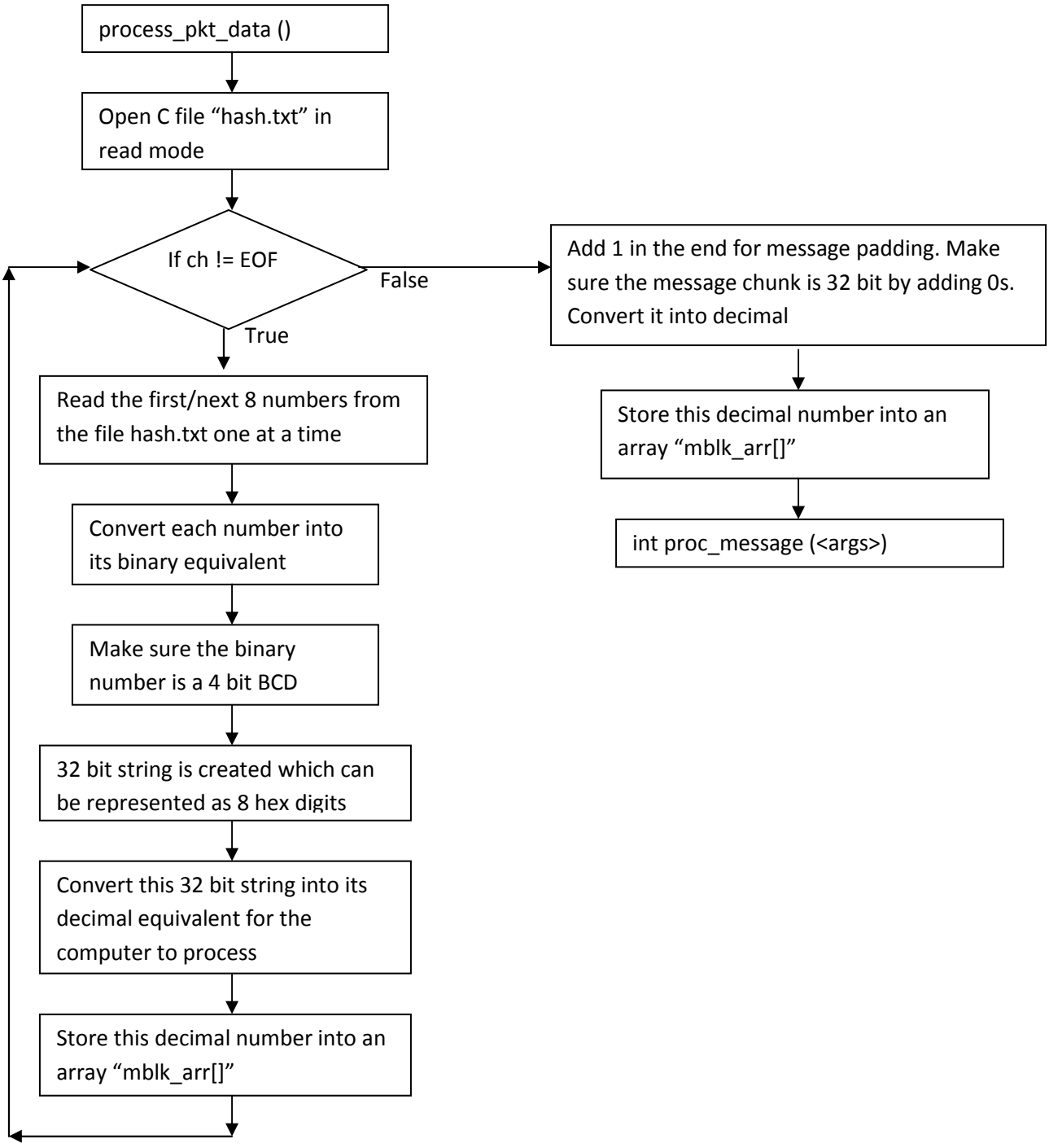
Flowcharts

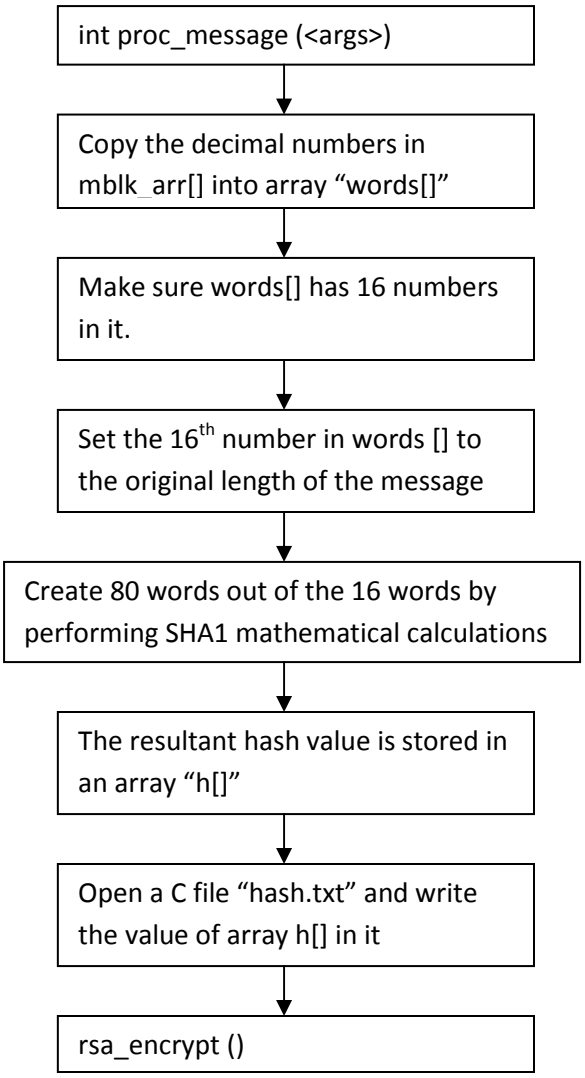

aodv_rte_rreq_pkt_arrival_handle
If hop_cnt=0
True
False
set_seed() (calculate seed value for creating hash chains)
Write value of seed in a C file "top_hash"
h_ch=0
If h_ch < max_hop_ cnt
True
False
Hop_cnt_proc()
Process data for creating a hash
Hop_cnt_hash (<args>)
Write value of top hash in a C file top_hash
h_ch++
Hop_cnt_proc()
Process data for creating a hash
Hop_cnt_hash (<args>)
Write value of top hash in a C file top_hash
Read the file top_hash and save its value in the top_hash_fd (AODV extension field)
Top hash for current node calculated

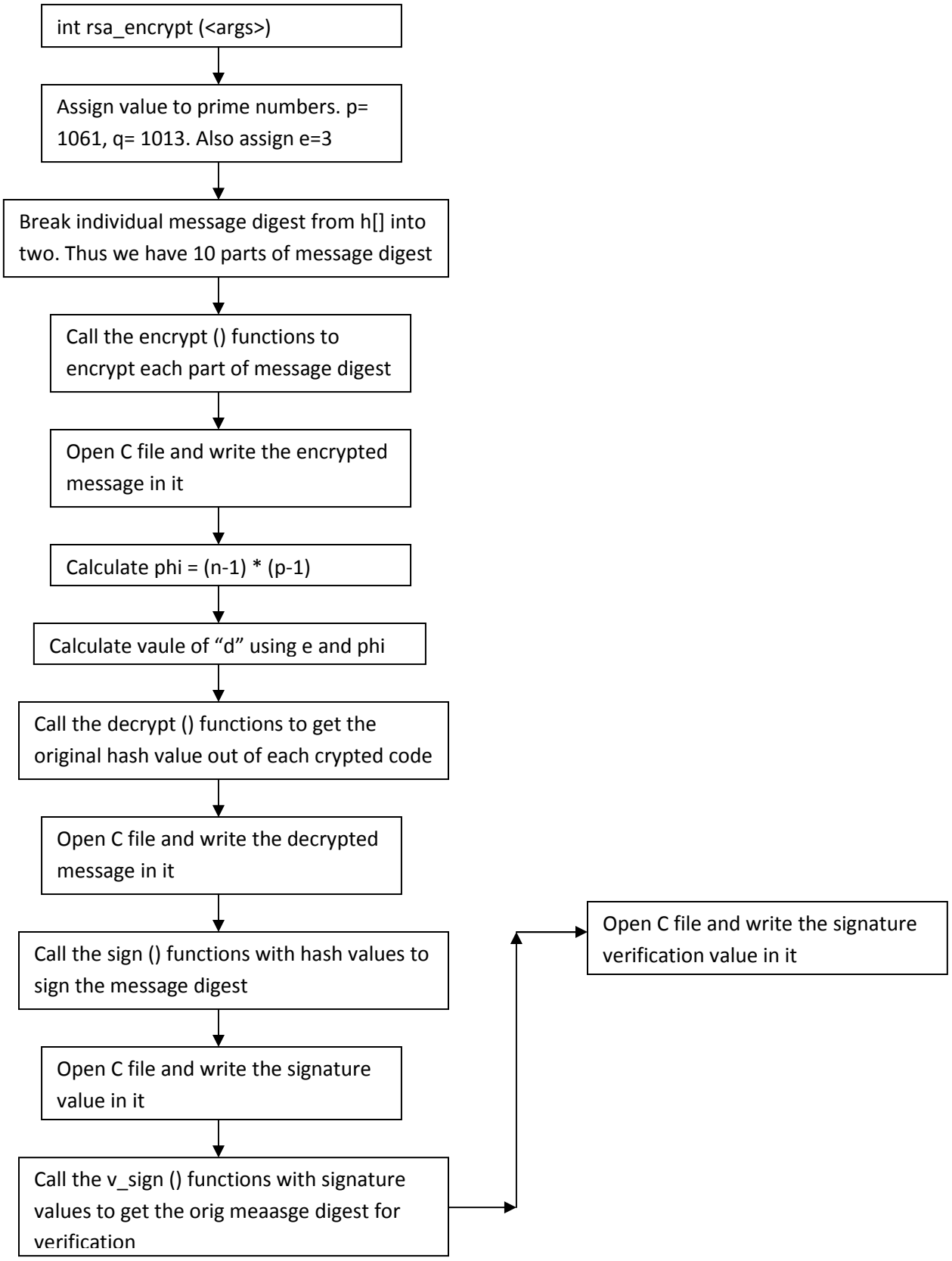
Create Hash and Signature for the packet




A
rsa_encrypt ()
Create encrypted code by calling each encrypt () function
Open C file to copy value of encrypted code
Calculate value of “d” for decryption
Decrypt the encrypted code by calling the decrypt () functions
Open a C file to copy the value of decrypted code i.e. original message
Sign the message packet by calling the sign () functions
Open a C file to copy the value of the signature
Calculate sign verification value by calling v_sign () functions
Open a C file to copy the value of the signature verification


process_pkt_data ()
Open C file “hash.txt” in read mode
If ch != EOF
False
True
Add 1 in the end for message padding. Make sure the message chunk is 32 bit by adding 0s. Convert it into decimal
Store this decimal number into an array “mblk_arr[]”
int proc_message (<args>)
Read the first/next 8 numbers from the file hash.txt one at a time
Convert each number into its binary equivalent
Make sure the binary number is a 4 bit BCD
32 bit string is created which can be represented as 8 hex digits
Convert this 32 bit string into its decimal equivalent for the computer to process
Store this decimal number into an array “mblk_arr[]”


int proc_message (<args>)
Copy the decimal numbers in mblk_arr[] into array “words[]”
Make sure words[] has 16 numbers in it.
Set the 16th number in words [] to the original length of the message
Create 80 words out of the 16 words by performing SHA1 mathematical calculations
The resultant hash value is stored in an array “h[]”
Open a C file “hash.txt” and write the value of array h[] in it
rsa_encrypt ()


int rsa_encrypt (<args>)
Assign value to prime numbers. p= 1061, q= 1013. Also assign e=3
Break individual message digest from h[] into two. Thus we have 10 parts of message digest
Call the encrypt () functions to encrypt each part of message digest
Open C file and write the encrypted message in it
Calculate phi = (n-1) * (p-1)
Calculate vaule of “d” using e and phi
Call the decrypt () functions to get the original hash value out of each crypted code
Open C file and write the decrypted message in it
Call the sign () functions with hash values to sign the message digest
Open C file and write the signature value in it
Call the v_sign () functions with signature values to get the orig meaasge digest for verification
Open C file and write the signature verification value in it

Algorithm

For AODV hop count:

1. Check if the hop count is zero.
2. If hop count is zero then it is at the first node. Calculate seed for top hash by calling set_seed() function.
3. Write value of seed in a C file “top_hash”.
4. Calculate hash chain by hashing seed max_hop_cnt times by calling the hop_cnt_proc() function to create a message block and then call the hop_cnt_hash() function(<args>) to calculate the top hash.
5. Write value of top hash in a C file top_hash. Then goto Step 8.
6. Else if the hop count is not zero then it is not at the first node. The node will verify and calculate its top hash by calling the hop_cnt_proc()function to create a message block and then call the hop_cnt_hash() function(<args>) to calculate the top hash.
7. Write value of top hash in a C file top_hash.
8. Increment the hop count by 1.
9. Continue to the next statement.

For Packet hash and Signature:

1. Read the AODV fields in the variables for packet processing. Also write the values of the fields in a C file hash.txt.
2. Check if the hop count > 1.
3. If hop count is greater than 1 then it is not at the first node. So the current node first verifies the signature.
4. Opens the verify.txt and hash.txt files for signature verification.
5. If there is any character mismatch during verification the signature is not verified.
6. Then decryption of the message takes place. Decryption can be verified by reading the value from hash.txt. Since the hash value was encrypted, decryption should result into hash value.
7. If the hop count is not greater than 1 than it is at the first node. The packet needs to be processed to create a message block for hashing. Call process_pkt_data function.
8. Open the file hash.txt in read mode.
9. If hash.txt has not reached EOF.
10. Read the first/next 8 numbers from the file hash.txt one at a time.
11. Convert each number into its binary equivalent.
12. Make sure the binary number is a 4 bit BCD.
13. 32 bit string is created which can be represented as 8 hex digits.
14. Convert this 32 bit string into its decimal equivalent for the computer to process.

15. Store this decimal number into an array “mblk_arr[]”. Then go to Step 10.

16. If the file hash.txt has reached EOF.

17. Add 1 in the end for message padding.

18. Make sure the message chunk is 32 bits long by adding zeros to the end.

19. Store this decimal number into an array “mblk_arr[]”.

20. Then call the proc_message(<args>) function to calculate the hash value.

21. Copy the decimal numbers in mblk_arr[] into array “words[]”.

22. Make sure words[] has 16 numbers in it.

23. Set the 16^{th} number in words [] to the original length of the message.

24. Create 80 words out of the 16 words by performing SHA1 mathematical calculations .

25. The resultant hash value is stored in an array “h[]”.

26. Open a C file “hash.txt” and write the value of array h[] in it.

27. Then call the rsa_encrypt(<args>) function to encrypt and sign the message digest.

28. Assign value to prime numbers. p= 1061, q= 1013. Also assign e=3.

29. Break individual message digest from h[] into two. Thus we have 10 parts of message digest.

30. Call the encrypt () functions to encrypt each part of message digest.

31. Open C file encrypt.txt and write the encrypted message in it.

32. Calculate phi = (n-1) * (p-1).

33. Calculate vaule of “d” using e and phi.

34. Call the decrypt () functions to get the original hash value out of each crypted code.

35. Open C file decrypt.txt and write the decrypted message in it.

36. Call the sign () functions with hash values to sign the message digest.

37. Open C file signature.txt and write the signature value in it

38. Call the v_sign () functions with signature values to get the original message digest for verification

39. Open C file verify.txt and write the signature verification value in it.

C Programming code

```
int proc_message(unsigned long int mblk_arr[], int orig_len, int msg_chnk)
{
  FILE *fp;
  int t;
  unsigned long int words[80], tempr, h[4], a, b, c, d, e, f, k;
  unsigned long int *ptr_mblk_arr;

  FIN (proc_message (unsigned long int mblk_arr[], int orig_len, int msg_chnk));

  h[0] = 0x67452301, h[1] = 0xEFCDAB89, h[2] = 0x98BADCFE, h[3] = 0x10325476, h[4] = 0xC3D2E1F0;

  ptr_mblk_arr = mblk_arr;
  for(t=0;t<msg_chnk;t++)
  {
     words[t] = *ptr_mblk_arr;        //Get each word by incrementing the memory location
     ptr_mblk_arr++;
  }
```

```
 for(t=msg_chnk; t<=14; t++)
 {
     words[t] = 0xFFFFFFFF;
 }

words[t] = orig_len;

for(t = 16; t < 80; t++)
  {
    words[t] = words[t-3] ^ words[t-8] ^ words[t-14] ^ words[t-16];
    words[t] = words[t] << 1;
  }

      a = h[0];
      b = h[1];
      c = h[2];
      d = h[3];
      e = h[4];

for(t = 0; t < 80; t++)
      {
         if(t<=19)
         {
            f = (b & c) | ((~b) & d);
            k = 0x5A827999;
         }
         else if(t<=39)
         {
            f = b ^ c ^ d;
            k = 0x6ED9EBA1;
         }
         else if(t<=59)
         {
            f = (b & c) | (b & d) | (c & d);
            k = 0x8F1BBCDC;
         }
         else
         {
            f = b ^ c ^ d;
            k = 0xCA62C1D6;
         }

         tempr = a << 5;
         tempr = (tempr + f + e + k + words[t]) & 0xFFFFFFFF;
         e = d;
         d = c;
         c = b << 30;
         b = a;
```

```
            a = tempr;
        }

        h[0] = h[0] + a;
        h[1] = h[1] + b;
        h[2] = h[2] + c;
        h[3] = h[3] + d;
        h[4] = h[4] + e;

    printf("\n\n");
    printf("Hash Field (Packet Hash): %u %u %u %u %u",h[0], h[1], h[2], h[3], h[4]);
    printf("\n\n");

            fp = fopen("c:\\test\\hash.txt","w");
            fprintf(fp,"%u%u%u%u%u",h[0],h[1],h[2],h[3],h[4]);
            fclose(fp);


            rsa_encrypt(h);


            FRET (0);
   }


int rsa_encrypt(unsigned long int h[])
{

  FILE *fp;
  unsigned long int *h_ptr;
  unsigned int p,q,n,phi,d;

  unsigned int h1, h2, h3, h4, h5;
  unsigned int h11, h12, h21, h22, h31, h32, h41, h42, h51, h52;

  unsigned int mess1,mess2,mess3,mess4,mess5,mess6,mess7,mess8,mess9,mess10;
  unsigned int c1,c2,c3,c4,c5,c6,c7,c8,c9,c10;

  unsigned int s11,s12,s21,s22,s31,s32,s41,s42,s51,s52;
  unsigned int v11,v12,v21,v22,v31,v32,v41,v42,v51,v52;
  int e;

  FIN (rsa_encrypt(unsigned long int h));

  h_ptr = h;
  h1 = *h_ptr; h_ptr++; h2 = *h_ptr; h_ptr++; h3 = *h_ptr; h_ptr++; h4 = *h_ptr; h_ptr++; h5 = *h_ptr;
h_ptr++;
  p = 1061; q = 1013;
```

```
n = p*q;
e = 3;

//Break individual message digest(H) into two
h11 = h1/100000;
h12 = h1%100000;

h21 = h2/100000;
h22 = h2%100000;

h31 = h3/100000;
h32 = h3%100000;

h41 = h4/100000;
h42 = h4%100000;

h51 = h5/100000;
h52 = h5%100000;

c1 = encrypt1(n, h11, e);
c2 = encrypt2(n, h12, e);
c3 = encrypt3(n, h21, e);
c4 = encrypt4(n, h22, e);
c5 = encrypt5(n, h31, e);
c6 = encrypt6(n, h32, e);
c7 = encrypt7(n, h41, e);
c8 = encrypt8(n, h42, e);
c9 = encrypt9(n, h51, e);
c10 = encrypt10(n, h52, e);

printf("\nEncrypted code of message is %u %u %u %u %u %u %u %u %u %u", c1, c2, c3,c4, c5, c6, c7,
c8, c9, c10);

fp = fopen("c:\\test\\enc_code1.txt", "w");
fprintf(fp,"%u", c1);
fclose(fp);

fp = fopen("c:\\test\\enc_code2.txt", "w");
fprintf(fp,"%u", c2);
fclose(fp);

fp = fopen("c:\\test\\enc_code3.txt", "w");
fprintf(fp,"%u", c3);
fclose(fp);

fp = fopen("c:\\test\\enc_code4.txt", "w");
fprintf(fp,"%u", c4);
fclose(fp);
```

```
  fp = fopen("c:\\test\\enc_code5.txt", "w");
  fprintf(fp,"%u", c5);
  fclose(fp);

  fp = fopen("c:\\test\\enc_code6.txt", "w");
  fprintf(fp,"%u", c6);
  fclose(fp);

  fp = fopen("c:\\test\\enc_code7.txt", "w");
  fprintf(fp,"%u", c7);
  fclose(fp);

  fp = fopen("c:\\test\\enc_code8.txt", "w");
  fprintf(fp,"%u", c8);
  fclose(fp);

  fp = fopen("c:\\test\\enc_code9.txt", "w");
  fprintf(fp,"%u", c9);
  fclose(fp);

  fp = fopen("c:\\test\\enc_code10.txt", "w");
  fprintf(fp,"%u", c10);
  fclose(fp);

  phi = (p-1)*(q-1);

  d = calc_d(e, phi);

  mess1 = decrypt1(c1, n, d);
  mess2 = decrypt2(c2, n, d);
  mess3 = decrypt3(c3, n, d);
  mess4 = decrypt4(c4, n, d);
  mess5 = decrypt5(c5, n, d);
  mess6 = decrypt6(c6, n, d);
  mess7 = decrypt7(c7, n, d);
  mess8 = decrypt8(c8, n, d);
  mess9 = decrypt9(c9, n, d);
  mess10 = decrypt10(c10, n, d);

  printf("\n**********Decrypting the encrypted code*********");
  printf("\nOriginal message after decryption is %u %u %u %u %u %u %u %u %u %u",mess1, mess2,
mess3, mess4, mess5, mess6, mess7, mess8, mess9, mess10);

  s11 = sign11(h11, d, n);
  s12 = sign12(h12, d, n);
  s21 = sign21(h21, d, n);
  s22 = sign22(h22, d, n);
```

```
  s31 = sign31(h31, d, n);
  s32 = sign32(h32, d, n);
  s41 = sign41(h41, d, n);
  s42 = sign42(h42, d, n);
  s51 = sign51(h51, d, n);
  s52 = sign52(h52, d, n);

  printf("\nThe signature is %u %u %u %u %u %u %u %u %u %u",s11,s12,s21,s22,s31,s32,s41,s42,s51,s52);

  fp = fopen("c:\\test\\signature.txt", "w");
  fprintf(fp,"%u%u%u%u%u%u%u%u%u%u",s11,s12,s21,s22,s31,s32,s41,s42,s51,s52);
  fclose(fp);

  fp = fopen("c:\\test\\signature11.txt", "w");
  fprintf(fp,"%u",s11);
  fclose(fp);

  fp = fopen("c:\\test\\signature12.txt", "w");
  fprintf(fp,"%u",s12);
  fclose(fp);

  fp = fopen("c:\\test\\signature21.txt", "w");
  fprintf(fp,"%u",s21);
  fclose(fp);

  fp = fopen("c:\\test\\signature22.txt", "w");
  fprintf(fp,"%u",s22);
  fclose(fp);

  fp = fopen("c:\\test\\signature31.txt", "w");
  fprintf(fp,"%u",s31);
  fclose(fp);

  fp = fopen("c:\\test\\signature32.txt", "w");
  fprintf(fp,"%u",s32);
  fclose(fp);

  fp = fopen("c:\\test\\signature41.txt", "w");
  fprintf(fp,"%u",s41);
  fclose(fp);

  fp = fopen("c:\\test\\signature42.txt", "w");
  fprintf(fp,"%u",s42);
  fclose(fp);

  fp = fopen("c:\\test\\signature51.txt", "w");
  fprintf(fp,"%u",s51);
```

```
 fclose(fp);

 fp = fopen("c:\\test\\signature52.txt", "w");
 fprintf(fp,"%u",s52);
 fclose(fp);

 v11 = v_signn11(s11, e, n);
 v12 = v_signn12(s12, e, n);
 v21 = v_signn21(s21, e, n);
 v22 = v_signn22(s22, e, n);
 v31 = v_signn31(s31, e, n);
 v32 = v_signn32(s32, e, n);
 v41 = v_signn41(s41, e, n);
 v42 = v_signn42(s42, e, n);
 v51 = v_signn51(s51, e, n);
 v52 = v_signn52(s52, e, n);

printf("\nThe Signature Verification is %u%u%u%u%u%u%u%u%u%u",
v11,v12,v21,v22,v31,v32,v41,v42,v51,v52);

 fp = fopen("c:\\test\\verify.txt", "w");
 fprintf(fp,"%u%u%u%u%u%u%u%u%u%u", v11,v12,v21,v22,v31,v32,v41,v42,v51,v52);
 fclose(fp);

 FRET(0);
}

unsigned int sign(unsigned int h, unsigned int d, unsigned int n)
{
 int i;
 unsigned long long int mod1, mod2;

 FIN (sign(unsigned int h, unsigned int d, unsigned int n));

 mod2=1;

 for(i=0;i<d;i++)
 {
  mod1 = h % n;
  mod2 = (mod2*mod1)%n;
 }
FRET (mod2);
}
```

```
unsigned int v_signn(unsigned int s, int e, unsigned int n)
{
 int i;
 unsigned long long int mod1, mod2;

 FIN (v_signn(unsigned int s, int e, unsigned int n));

 mod2 = 1;

 for(i=0;i<e;i++)
 {
  mod1 = s % n;
  mod2 = (mod2*mod1)%n;
 }
 FRET (mod2);
}

unsigned int decrypt(unsigned int c, unsigned int n, unsigned int d)
{
 long long unsigned int mod1, mod2 = 1;
 int i;

 FIN (decrypt(unsigned int c, unsigned int n, unsigned int d));

 for(i=0; i<d; i++)
 {
  mod1 = c % n;
  mod2 = (mod2 * mod1)%n;
 }
 FRET (mod2);
}

unsigned int calc_d(int e, unsigned int phi)
{
 unsigned int i;

 FIN (calc_d(int e, unsigned int phi));

 for(i = 1; i < phi; i++)
 {
  if(((e*i) - 1) % phi == 0 )    // (ed-1) mod phi
     FRET (i);
 }
 FRET(0);
}
```

```
unsigned int encrypt(unsigned int n, unsigned int h, int e)
{
 unsigned int m, enc1;
 unsigned long long int mod1 = 1;
 int i;

 FIN (encrypt(unsigned int n, unsigned int h, int e));

 m = h;
 printf("\n\n The value of h = %u", h);
 for(i = 0; i < e; i++)
 {
  mod1 = mod1 * (m % n);                //c = m^e mod n
 }
 enc1 = mod1 % n;
 FRET (enc1);
}
```

Issues Encountered throughout the thesis work

My thesis was mostly based on writing a C code to secure the routing protocols. Since I was not from a programming background it was a little challenging for me to write a C code. But once I got a hang of the C programming language it was like a piece of cake. In my thesis I had to grab the contents of the packets and create a hash of it. It was infeasible to get the packet field values in one variable because of the limitation of the C language variable size. I could have used an array variable and have stored each field value in a location but then later for the next node to do the hash of hash value, storing the hash value in the same array variable at each location would have become cumbersome. So I decided to use FILE variables. I copied the contents of the fields in a C FILE and then read them one at a time during the hashing process. The resultant value of the hash was then copied into the C same file to ensure the next node can read the hash from the same FILE. Thus I had to create only one function block for all the nodes.

RSA and SHA-1 computations results into big integer numbers. I had to also make sure that my code works with the OPNET code and was developed according to the 32-bit architecture of OPNET simulation. In the beginning I decided to use seven digit integer numbers for “p” and “q” (the prime numbers) in RSA. Multiplication of these integers resulted in to a fairly big number which C would not support. The maximum variable size C can support is 4 bytes. So therefore I used the “unsigned long long int” data type. The range of this data type is from 0 to 4294967295. So I had to use five digits integer for p and q that would be within the range of the “unsigned long long int” data type. I was still having some issues with the decryption and signature verification. I tested the code in a simple environment. I debugged the code and was watching the variables and the values inside the variables at each run. Then I realized that the value of “d” which is the secret component used for decryption and digital signing, its value was going outside the range of “unsigned long long int”. The formula for calculating d = (1/e) * mod(phi). So value of phi (phi = (p-1)*(q-1)) was affecting value on d. So then I had to again go back to the prime numbers p and q and set a smaller value possibly a 4 digit number. I tested the code and it worked like a charm. So decryption and signature verification issues were solved by selecting the appropriate numbers for p and q that would fall within the range of “unsigned long long int” data type in C. Also there is a requirement that in order to successfully encrypt and decrypt the messages a condition m < n needs to be satisfied where m is the plain text message and n is the public key. The message blocks in the hash output was bigger than n. So I had to break one hash block into two and then send each part in the blocks for encryption. Decryption was also done for each part separate and then the message was concatenated.

Compilation of the code resulted into a few error messages like “L value required”. This was because I was trying to assign the value of a string variable to another string variable by using the “=” operator i.e. a=b where “a” and “b” are character variables. So in that case I had to convert the variable on the right of the operator to integer then copy it to the integer variable and again convert the later

variable to character variable. I also had issues working on pointer variables. Pointer variables were used for pointing to the location where the first block of hash out was stored. Error messages were given for pointer variables not defined correctly and inappropriate use of pointers. I fixed those errors by properly referencing the pointer variable to point to the location where the hash value resides. Other small errors related to data type, syntax errors, function prototype, assignment errors were resolved by doing a little research on the internet.

Adding my cryptographic code to OPNET's code was a challenging task. I opened AODV project in OPNET and was browsing though the code written for the IP module. I was thinking that since AODV does routing the code for AODV should be within the IP module. I opened the code written in each state of the process model to see if I could get to know where AODV packets are generated. I was adding printf statements where I could see op_pkt_create function. This function is used for creating a packet. I didn't find anything related to AODV in the IP module's process model. I then checked the "Function Block" in the process block. The code in the function clock was really big so I had to use "find" to look for the packet create function. I use to add printf statements and then run the simulation and see the packet fields in the console section. But I didn't find any packets related to AODV all I found was IP packets. I then started browsing the tabs in the menu bar on the project editor. It was really hard to find it that way because there were so many tabs options to select in the menu bar. I then contacted OPNET support asking for help explaining then the situation. I also talked to Hasan Tuncer a PhD. Student at RIT who also works on OPNET. He asked me to use printf statements and debugger and gave me a few tips and tricks that he had acquired working on OPNET. I also asked for help from Dr. Oh. He asked me to use visual studio debugger to debug the code and find where the AODV packets were generated. I then read an article on debugging using Visual studio but that article was not written in detail so it didn't helped me lot. I read another article about using OPNET debugger. To use that Windows Debugger should be installed on the computer. Then in the Preferences section of OPNET I had to add the path to the

debugger to let OPNET know where to find the debugger. During Simulation run I checked the “Add ODB” option in the simulation editor that would enable debugging. I ran the simulation with the debugger but didn’t get anything related to the packet fields. I again went to Hasan asking for help but this time I took my laptop and showed him the actual simulation. He then pointed me the child process. Within the IP module there was a child process for MANET. In the IP Process module child process can be opened by clicking the Menu Tab -> File -> Open Child process model. I browsed through the code for MANET within the process module and the function block. I tried using the printf statements but couldn’t find anything related to AODV. Also OPNET support replied back to me and they directed me to tutorials to help me find where AODV packets were generated. Within the MANET process there were other child processes for MANET related protocols. I was happy to see AODV and child process as that is what I was looking for. So AODV and DSR code was within the child processes of the IP module. I opened the function block for AODV that was within the process module. I then browsed through the code and saw a function block “aodv_rte_rreq_pkt_arrival_handle”. It was this function block where RREQ packets were created. I went through the function block and figured out where the fields were the packet was created. So once the packet was created I used printf statements to verify the values in the packet fields. For AODV created hash chain using the hop count value. Other fields were sent for hashing, encryption/decryption and digital signature/verification. I compiled the simulation and got errors. Errors were related to function proto types, variable declaration, and invalid memory access. For the function proto types I had to add the variable for the signature extension to the “aodv_pkt_support_rreq_option_create” function which was in the aodv_pkt_support.c file. I also had to add the signature extension variables to the same function but in aodv_ptypes.h. The prototypes for the functions I added to the OPNET code was also missing so I added them to the aodv_rte.h file which is the header block for the AODV process model. There I also added the signature extension fields to the routing messages. All of the errors were fixed except for the invalid memory access. I tried

troubleshooting the code to find out what the issue could be but had no luck. So I went to Yoshi who is doing his PhD at RIT. Yoshi has been working on OPNET for a long time and had good understanding of how the OPNET code works. He saw the function block I had created and pointed out in no time about the error. The syntax “FIN” function was not correct. For the functions created by OPNET the FIN function defined as: FIN (aodv_pkt_support_rreq_option_create(<args>)). I did the same when defining the FIN function for the code I had developed, FIN (sign22(<args>)) but that didn’t work. I had to actually add the arguments to the function instead of the “<args>” used by the OPNET. Also when a function returns a value instead of the traditional “RET” statement in C, OPNET uses “FRET”. So I had to replace the RET with the FRET. I changed the FIN statement to “FIN (sign22(unsigned int h22, unsigned int d, unsigned int n))” and used FRET in the code. I compiled the code and everything worked fine. The simulation run was successful. By now I had a good understanding of where and how to add the encryption code working on DSR was much easier. I followed the same procedure as I did for AODV and got encryption added to DSR. To get the results I talked to Dr. Oh about the generation of traffic in the network. He discussed with me about the “Application Config” and “Profile Config” and suggested me to read the tutorial for the same. Application Config and Profile Config were pretty easy to configure and the results were collected for comparison.

13. References

[1] Multipath routing in Mobile Ad-Hoc Networks: Issues and Challenges by Dipak Ghosal, Stephen Mueller, Rose P Tsang

[2] http://www.ietf.org/rfc/rfc2501.txt

[3] http://w3.antd.nist.gov/wctg/manet/docs/perf_routing_protocols.pdf

[4] http://moment.cs.ucsb.edu/pub/wwan_chakeres_i.pdf

[5] http://www.ietf.org/rfc/rfc4728.txt

[6] Performance Comparison of Two On-Demand Routing Protocols for Ad Hoc Network

http://www.cs.jhu.edu/~cs647/class-papers/Routing%20algorithms/das.pdf

[7] Challenges of Secure Routing in MANETs: A Simulative Approach using AODV-SEC

http://ieeexplore.ieee.org.ezproxy.rit.edu/stamp/stamp.jsp?tp=&arnumber=4053935

[8] A secure routing protocol for ad Hoc networks

http://ieeexplore.ieee.org.ezproxy.rit.edu/stamp/stamp.jsp?tp=&arnumber=1181388

[9] Energy aware On Demand Routing for MANETs

[10] Providing Robust and Ubiquitous Security Support for Mobile Ad Hoc Networks

http://irl.cs.ucla.edu/papers/ICNP01-haiyun.pdf

[11] Certification and Authentication Services for Securing MANET Routing Protocols

[12] http://www.garykessler.net/library/crypto.html#pkc

[13] http://www.itl.nist.gov/fipspubs/fip180-1.htm

[14] An Extended AODV Protocol for VOIP Application in MANET

http://www.ecti-thailand.org/assets/papers/803_pub_19.pdf

[15] Performance analysis of Mobile Ad Hoc Network using AODV protocol

http://www.cscjournals.org/csc/manuscript/Journals/IJCSS/volume3/Issue5/IJCSS-137.pdf

[16] Modified Energy-Aware DSR Routing for Ad Hoc Network

http://ieeexplore.ieee.org.ezproxy.rit.edu/stamp/stamp.jsp?tp=&arnumber=4340178

[17] Performance Enhancement of Dynamic Source routing using Simulated Annealing

http://ieeexplore.ieee.org.ezproxy.rit.edu/stamp/stamp.jsp?tp=&arnumber=4787687

[18] Performance of Wireless Network Simulators

http://delivery.acm.org.ezproxy.rit.edu/10.1145/1460000/1454639/p59-orfanus.pdf?ip=129.21.35.191&acc=ACTIVE%20SERVICE&CFID=54528533&CFTOKEN=57595356&__acm__=1321813834_a746dc843328cfe8c005ec02347d964a

[19] On the Accuracy of MANET Simulators

http://delivery.acm.org.ezproxy.rit.edu/10.1145/590000/584499/p38-cavin.pdf?ip=129.21.35.191&acc=ACTIVE%20SERVICE&CFID=54528533&CFTOKEN=57595356&__acm__=1321813808_41f93ccc7c986c57fbac6835dcf45f12

[20] Network Simulations with OPNET

http://delivery.acm.org.ezproxy.rit.edu/10.1145/330000/324232/p307-chang.pdf?ip=129.21.35.191&acc=ACTIVE%20SERVICE&CFID=54528533&CFTOKEN=57595356&__acm__=1321813796_9d7741b2a8f2d474389eb1c41e65bd71

[21] On the Evaluation and Classification of Routing Protocols for Mobile Ad Hoc Netwrok by *Daniel Lang*

[22] http://www.schneier.com/blog/archives/2005/02/sha1_broken.html

[23] http://eprint.iacr.org/2007/474

[24] http://eprint.iacr.org/2008/469.pdf

[25] http://www.geometer.org/mathcircles/RSA.pdf