Vulnerability Analysis and Security Framework (BeeSec) for Nature Inspired MANET Routing Protocols

Nauman Mazhar, Muddassar Farooq
Center for Advanced Studies in Engineering (CASE)
19 Ataturk Avenue, G-5/1 Islamabad
Pakistan

ABSTRACT

Design, development and evaluation of adaptive, scalable, and power aware Bio/Nature inspired routing protocols has received a significant amount of attention in the recent past. However, to the best of our knowledge no attempts have been made to systematically analyze their security vulnerabilities. In this paper, we investigate the security vulnerabilities of *BeeAdHoc*, a well known power aware, Bio/Nature inspired routing protocol. Our analysis clearly demonstrates that the malicious nodes in an untrusted MANET, where *BeeAdHoc* is used for routing, can significantly disrupt the normal routing behavior. We then propose a security framework, *BeeSec*, for *BeeAdHoc* that enables it to counter the different types of threats. We also designed an empirical framework, embedded into a well known simulator, ns-2, to systematically validate the operational security of *BeeSec*. An interesting outcome of the research is that *BeeSec*, even with significant overhead of the security framework, achieves better performance as compared to state-of-the-art, non-secure, classical routing protocols *AODV* and *DSR*.

Categories and Subject Descriptors

C.2.0 [**General**]: [Security and protection]; C.2.1 [**Network Architecture and Design**]: [Distributed networks, Wireless communication]; C.2.2 [**Network Protocols**]: [Protocol architecture, Routing protocols]

General Terms

Algorithms, Design, Security

Keywords

Swarm Intelligence, Mobile Ad Hoc Networks, Self-Organization, Cryptographic Security, Misbehavior Detection

1. INTRODUCTION

Mobile Ad Hoc Networks (MANETs) is becoming an active area of research [10]. All nodes in the transmission range of one another communicate directly using their radios and relay messages to their neighbors for further transmission towards the destination if it is not in their transmission range. Nodes are mobile, which results in a continuously changing topology. Moreover, route discovery and data forwarding phases must efficiently consume the limited battery capacity of the nodes. *DSR* (Dynamic Source Routing) [5] and *AODV* (Ad-Hoc On-demand Distance Vector Routing) [8] are well known classical reactive routing protocols in MANETs.
Biological systems show a number of properties, such as self-organization, adaptivity, scalability, robustness, autonomy, locality of interactions and distribution. These are highly desirable in dealing with the complex issues in MANETs. *AntHocNet* [3], *BeeAdHoc* [14] and *Termite* [7] are considered to be state-of-the-art Bio/Nature inspired routing protocols. Their common features are: multipath discovery and probabilistic distribution of data traffic on these multi paths to achieve better performance. This feature also results in depletion of the batteries of nodes at the same rate.
MANETs provide an ideal environment to a malicious node to fabricate and launch attacks. It can either disrupt the normal routing behavior or significantly degrade the performance of a network [6]. A malicious node can join a MANET and easily launch different types of attacks:

- *Fabrication attacks* are launched by a malicious node by installing fake routes instead of valid ones. This is made possible by creating and launching forged or bogus agents in the networks.
- In *Tampering attacks* a malicious node illegally modifies different information fields of an agent, i.e. its source address, id, delay, remaining battery capacity, etc. The malicious node can also relay old agents.
- In *Dropping attacks* a malicious node drops route discovery packets to either divide the network into several partitions or launch a Denial of Service (DoS) attack.

To the best of our knowledge no serious and in depth research has been undertaken in the Bio/Nature inspired community to analyze the security risks of the above-mentioned Bio/Nature inspired MANET routing protocols on similar lines as the researchers in classical network community have done for *DSR* [4] and *AODV* [17]. However, some valuable attempts have been made by the authors of [16] [15] to study the vulnerabilities of *BeeHive*, a Bio/Nature inspired routing protocol for fixed networks, and they have proposed two security frameworks for *BeeHive* to counter the attacks.
The stochastic routing in Bio/Nature inspired routing protocols makes it difficult to identify that the goodness of a

certain path is degraded because of an attack or its quality has degraded due to real network conditions. The major contributions of the work proposed in this paper are:

1. Design, development and implementation of a security framework, *BeeSec*, for *BeeAdHoc* protocol.
2. Realization of a comprehensive performance evaluation framework, in ns-2, to systematically analyze the impact of different types of attacks on a routing protocol. The framework is also used as a validation tool for *BeeSec* to demonstrate that it has successfully encountered most types of attacks.
3. The results obtained through extensive experiments clearly demonstrate that *BeeSec*, despite its higher processing overhead and large size of agents, achieves better performance as compared to even the non-secure MANET protocols, *AODV* and *DSR*. We believe that the most important reason for the superior results is that *BeeAdHoc*, the base protocol of *BeeSec*, was designed from scratch to achieve a higher performance but at low energy consumption [14].

The rest of the paper is organized as follows. In Section 2 we provide a brief introduction to the *BeeAdHoc* protocol and in Section 3 we carry out a systematic analysis of its security vulnerabilities. Then we introduce relevant features of our *BeeSec* security framework in Section 4. Section 5 describes our validation and evaluation framework, now embedded in ns-2, and we demonstrate the significant impact that a malicious node can have on the routing behavior of *BeeAdHoc*. However, *BeeSec* is shown to successfully counter these attacks. In order to verify that our security enhancements did not degrade the performance of the original *BeeAdHoc* algorithm, we extensively compared *BeeSec* with *BeeAdHoc*, *AODV* and *DSR*, in Section 6. The results clearly demonstrate that *BeeSec* achieves better performance as compared to *AODV* and *DSR*. Finally, we conclude the paper with an outlook to our future research.

2. BEEADHOC PROTOCOL

BeeAdHoc is a nature inspired, source routing protocol for MANETs, with design based on the foraging principles of honey bees [14]. It mainly utilizes two types of *agents*; *scouts* to discover routes and *foragers* to transport data. Each node maintains a hive with an *Entrance*, *Packing Floor* and a *Dance Floor*. Entrance provides an interface to the Media Access Control (MAC) layer of the network stack. Packing floor is an interface to the transport layer and receives data from it. Data packets are stored at the packing floor, if route discovery to their destination is in progress. Routes are stored at the dance floor.

When route to a destination is needed, forward scout is broadcast in the network, in an expanding ring search. Receiving nodes append their addresses to the source route of the scout till it arrives at the destination node. The destination node reverses the route and unicasts the scout back to the source node. Once the scout returns to the hive of the source node, it advertises the route to foragers. Foragers are recruited for this path if its quality is above a threshold. Consequently, recruited foragers transport data to the destination node. On their journey, they also collect relevant routing information that is used to evaluate the dance number, which represents the quality of the path traversed. We provide only that much description of the *BeeAdHoc* protocol as is necessary to make this paper self-contained. Interested readers can find its complete description in [14].

3. BEEADHOC VULNERABILITIES

The purpose of *Byzantine* [9] attacks is to disrupt the normal routing behavior of a protocol. In these attacks, an attack is considered successful if a malicious node achieves its objectives; divert traffic towards itself when it does not lie on a path towards destination or remove itself from a valid path. In both cases, the performance of a network is severely degraded even if the malicious node does not drop the data packets.

3.1 Security Threat Analysis of BeeAdHoc

We systematically analyzed different shortcomings in *BeeAdHoc* that could enable a malicious node to launch a number of attacks. We outline a few of them here.

- **Scout Related Attacks.** When a scout is on its forward journey trying to find a route to the destination, it is propagated using the broadcast technique. As a consequence, a large number of nodes receive its replicas even if they do not lie on a direct path leading towards the destination. A malicious node can partially modify the source route in a scout and retransmit it. Otherwise, it can insert a completely new source route and send the scout back as a unicast packet towards the source. Similarly, a malicious node can also alter the source header of a unicast backward scout. Alternatively, it can forge a scout by spoofing the source address or inserting fake scout ID, or both. Once these scouts return to the source node, the forged route is established between the source and destination nodes.
- **Forager Related Attacks.** Foragers carry data packets in their payload and are transmitted as unicast packets. A malicious node can, again, modify the forager's source route during its journey from source to destination. Similarly a malicious node can launch a forged forager with spoofed source address and fake source route towards destination. Once this forager returns to the source node, a fake route is established. Alternatively, a malicious node can launch a fake forager towards the source node with a forged route. In this scenario the forged route will be installed more quickly as compared to the previous case.

One fundamental assumption, in order to launch the above-mentioned attacks successfully, is for the malicious node to ensure that the route from source to destination is complete. If hop-by-hop connectivity does not exist, the forged scout or forager will be dropped during its forward or backward journey. This, instead, in a worst case scenario could degenerate into a DoS attack if no foragers or scouts ever return at the source node. A malicious node can launch a relatively *softer attack* by modifying the different routing parameters in order to artificially enhance the quality of a path. This will increase the *dance number* value and as a consequence, more replicas of this route will be stored. Therefore, the probability to forward a higher number of data packets on this low quality route increases that results in wasting precious network resources.

4. BEESEC: SECURITY FRAMEWORK FOR BEEADHOC

We can easily conclude from Section 3 that *BeeAdHoc* can be secured against *Byzantine* attacks if we secure the header fields of scouts and foragers against unauthorized modification. In networks, secure systems research employs either public key encryption (*Digital Signatures*) or symmetric key systems (*Message Authentication Codes*) for authentication.

Symmetric key systems have less processing overhead [12] but the communicating parties need to know and store a large number of keys. Moreover, the keys have to be regularly changed and distributed securely, making key management a serious challenge. This shortcoming is even more serious in an untrusted MANET environment where nodes frequently enter and leave the network. This calls for an on-line, real time, secure key distribution system which is scalable to large networks. In contrast, a node in public key cryptography requires just one pair of public/private keys for itself and one public key for each node with which it wants to communicate, making key distribution relatively simple. Therefore, we find it more appropriate to use public key cryptography in a MANET environment.

BeeSec, a secure version of *BeeAdHoc* utilizes digital signatures. The scouts and foragers are modified to carry digital signatures that are computed on source address, packet ID, routing information and source route, etc. In addition, the integrity of the source route is maintained to ensure that no node on the route can be removed.

Symbol	Description
s, d, i	source, destination & the ith node
IP_s, IP_d, R, R_i	source Internet Protocol (IP) address, destination IP address, complete source route & source route upto $node_i$
FS_{sd}, BS_{sd}, F_{sd}	forward scout, backward scout and forager going from $node_s$ to $node_d$
Hdr_{BeeSec}	header of *BeeSec* protocol
$S_{sct}, ID_{sct}, D_{sct}$	source, ID and destination of a scout
$Auth_{FS}$	digital signature of scout header fields IP_s, ID_{sct}, D_{sct}
$Chk_{RtInteg}$	digital signature of current source route of a scout including all hops
$Auth_{BS}$	digital signature of backward scout header fields IP_s, ID_{sct}, D_{sct} and R_i
$Auth_F$	digital signature of forager header fields IP_s, IP_d, R_i
$Auth_{RtInfo}$	digital signature of route info field in the header of a forager
$H(M)$	hash of message M
KU_s, KR_s	public & private keys of $node_s$
$Sign(.)$, $Verify(.)$	functions to create & verify digital signatures; detailed description in [12]

Table 1: List of symbols used in the paper

4.1 Scout Authentication

Authentication of a scout is done by computing digital signature (authenticator) and inserting into the header of a scout. A receiving node can, with the help of the authenticator, verify that the scout was not tampered by a malicious node. Certain fields in the header of a scout remain unchanged (fixed) during its forward or backward propagation i.e IP_s, ID_{sct}, D_{sct}. In comparison, the source route is changed by each node as it inserts its address into it.

Algorithm-1: Security Extensions for Forward Scouts

```
for all ( $FS_{sd}$ launched from $S_{sct}$ to $D_{sct}$ ) do
  if ( $FS_{sd}$ broadcast from $node_i$ ) then
    compute $Auth_{FS}$ for $node_i$
    store $Auth_{FS}$ for $node_i$ in $Hdr_{BeeSec}$
    compute $Chk_{RtInteg}$ for $node_i$
    store $Chk_{RtInteg}$ for $node_i$ in $Hdr_{BeeSec}$
    if ( $node_i \neq S_{sct}$ for $FS_{sd}$ ) then
      store list of previous $Auth_{FS}$ in $Hdr_{BeeSec}$
      store $Chk_{RtInteg}$ for $node_{i-1}$ in $Hdr_{BeeSec}$
    end if
    broadcast $FS_{sd}$

  else if ( $FS_{sd}$ received at $node_i$ ) then
    if ( $FS_{sd}$ revisiting a node in source route ) then
      drop $FS_{sd}$ and exit
    end if
    for all ( $Auth_{FS}$ values in $Hdr_{BeeSec}$ ) do
      verify $Auth_{FS}$ value
      if ( $Auth_{FS}$ fails ) then drop $FS_{sd}$ and exit
      end if
    end for
    verify $Chk_{RtInteg}$ value for $node_{i-1}$
    if ( $Chk_{RtInteg}$ for $node_{i-1}$ fails ) then
      drop $FS_{sd}$ and exit
    else
      store $Chk_{RtInteg}$ for $node_{i-1}$ in $Hdr_{BeeSec}$
    end if
    if ( $node_i \neq$ 1st hop after launch of $SS_{sd}$ ) then
      verify $Chk_{RtInteg}$ value for $node_{i-2}$
      if ( $Chk_{RtInteg}$ for $node_{i-2}$ fails ) then
        drop $FS_{sd}$ and exit
      end if
    end if
    if ( $node_i \neq D_{sct}$ for $FS_{sd}$ ) then
      store $Auth_{FS}$ values in $Hdr_{BeeSec}$
      pass $FS_{sd}$ to entrance for re-broadcasting
    else
      pass $FS_{sd}$ to entrance to convert to $BS_{ds}$
    end if
  end if
end for
```

Algorithm-2: Security Extensions for Backward Scouts

```
for all ( $BS_{ds}$ returning from $D_{sct}$ to $S_{sct}$ ) do
  if ( $BS_{ds}$ unicast from $node_i$ to $node_{i+1}$ ) then
    if ( $node_i == D_{sct}$ for $BS_{ds}$ ) then
      compute $Auth_{BS}$
      store $Auth_{BS}$ in $Hdr_{BeeSec}$
    end if
  else if ( $BS_{ds}$ received at $node_i$ ) then
    if ( $node_i == S_{sct}$ for $BS_{ds}$ ) then
      verify $Auth_{BS}$ for $D_{sct}$
      if ( $Auth_{BS}$ for $D_{sct}$ fails ) then
        drop $BS_{ds}$ and exit
      else
        pass $BS_{ds}$ to entrance for inclusion in dancefloor
      end if
    end if
  end if
end for
```

Each $node_i$ broadcasting a scout computes the digital signature, $Auth_{FS}$, on fixed scout header fields using the digital signature function and its private key, as:

$$Auth_{FS_i} = Sign(H(IP_s, ID_{sct}, D_{sct}), KR_i) \qquad (1)$$

It then inserts it into the header and broadcasts the scout. An intermediate node can authenticate the $Auth_{FS}$ values stored in the scout header for each node in the source route using the digital signature verification function, as:

$$Verify(Auth_{FS_i}, H(IP_s, ID_{sct}, D_{sct}), KU_i) \qquad (2)$$

A broadcasting $node_i$ also computes another authenticator, $Chk_{RtInteg}$, on the modifiable source route field, as:

$$Chk_{RtInteg} = Sign(H(R_i), KR_i) \tag{3}$$

An intermediate $node_i$ uses the verification function to ensure that the source route was indeed sent by $node_{i-1}$, as:

$$Verify(Chk_{RtInteg_{i-1}}, H(R_{i-1}), KU_{i-1}) \tag{4}$$

However, if the previous node illegally modified the source route, the above process cannot provide protection against it. Therefore, two authenticators are stored in the scout header after the first hop. In this way, $node_i$ can use $Chk_{RtInteg}$ of $node_{i-1}$ and $Chk_{RtInteg}$ of $node_{i-2}$ to detect any illegal modification of the source route by $node_{i-1}$.

$$Verify(Chk_{RtInteg_{i-2}}, H(R_{i-2}), KU_{i-2}) \tag{5}$$

Finally, $node_i$ appends its IP address and computes a new $Chk_{RtInteg}$ value on the new source route and inserts it into the header of the scout. In this way, the predecessor of predecessor node acts as a protector against illegal source modification by the previous node. However, this approach makes a reasonable assumption; no two immediate successor nodes on a path are malicious nodes.
When the scout reaches the destination, D_{sct}, it is unicast back towards the source of the scout, S_{sct}. But before transmitting, D_{sct} computes the $Auth_{BS}$, as:

$$Auth_{BS} = Sign(H(IP_s, ID_{sct}, D_{sct}, R), KR_s) \tag{6}$$

Once the scout is received by S_{sct}, it can verify the integrity of scout header fields by using the signature verification function. A scout is dropped if the verification fails.

4.2 Forager Authentication

A forager has fixed (non-modifiable) fields like IP_s, IP_d and R. To protect these fixed fields, once a forager is launched by $node_s$, it computes an authenticator, $Auth_F$, and places it in the forager header.

$$Auth_F = Sign(H(IP_s, IP_d, R), KR_s) \tag{7}$$

At each hop from source to destination, a forager collects different parameters to model the quality of a route, i.e, delay and remaining battery capacity. To protect this routing information, a sending $node_i$ computes an authenticator and stores it in the header of the forager.

$$Auth_{RtInfo_i} = Sign(H(RouteInformation_i), KR_i) \tag{8}$$

In addition, the authenticator for the previous $node_{i-1}$ is also placed in the forager header. This authenticator allows the next hop $node_{i+1}$ to verify that the routing information of $node_{i-1}$ was not tampered by $node_i$. $Auth_{RtInfo}$ of $node_{i-1}$ is more relevant for those performance metrics whose value accumulates at each hop. In this way, a node can only falsify its own value and not the accumulated value of the complete path. Based on the results of authenticator verifications (Algorithm-4), the destination node uses the routing information from $node_{i-1}$ or $node_{i-2}$ to compute the dance number for the route. When the route's information field is found tampered by $node_{i-1}$, it is restored to the actual value of $node_{i-2}$. In the case of accumulated route information, the restored value is quite close to the actual. Again the assumption must hold that two immediate successor nodes on a path are not malicious nodes.

Algorithm-3: Security Extensions for Sending Foragers

```
for all ( $F_{sd}$ going from $node_s$ to $node_d$ ) do
  if ( $F_{sd}$ unicast from $node_i$ to $node_{i+1}$ ) then
    if ( $node_i$ == $node_s$ for $F_{sd}$ ) then
      compute $Auth_F$ for $node_s$
      store $Auth_F$ for $node_s$ in $Hdr_{BeeSec}$
    end if
    compute $Auth_{RtInfo}$ for $node_i$
    store $Auth_{RtInfo}$ for $node_i$ in $Hdr_{BeeSec}$
    if ( $node_i \neq node_s$ for $F_{sd}$ ) then
      store route information for $node_{i-1}$ in $Hdr_{BeeSec}$
      store $Auth_{RtInfo}$ for $node_{i-1}$ in $Hdr_{BeeSec}$
    end if
    send $F_{sd}$ to next hop $node_{i+1}$
  end if
end for
```

Algorithm-4: Security Extensions for Receiving Foragers

```
for all ( $F_{sd}$ going from $node_s$ to $node_d$ ) do
  if ( $F_{sd}$ received at $node_i$ ) then
    if ( $node_i$ == 1st hop after launch of $F_{sd}$ ) then
      verify $Auth_{RtInfo}$ for $node_{i-1}$
      if ($Auth_{RtInfo}$ fails) then drop $F_{sd}$ and exit
      end if
      pass $F_{sd}$ to entrance and send to next hop $node_{i+1}$

    else
      if ( $node_i \neq node_d$ for $F_{sd}$ ) then
        verify $Auth_{RtInfo}$ for $node_{i-1}$
        if ($Auth_{RtInfo}$ fails) then drop $F_{sd}$ and exit
        end if
        verify $Auth_{RtInfo}$ for $node_{i-2}$
        if ($Auth_{RtInfo}$ fails) then drop $F_{sd}$ and exit
        end if
        compare route info values for $node_{i-1}$ & $node_{i-2}$
        if (values not valid) then drop $F_{sd}$ and exit
        end if
        pass $F_{sd}$ to entrance and send to next hop $node_{i+1}$
      end if

      if ( $node_i$ == $node_d$ for $F_{sd}$ ) then
        verify $Auth_F$
        if ( $Auth_F$ fails ) then drop $F_{sd}$ and exit
        end if
        verify $Auth_{RtInfo}$ for $node_{i-1}$ & $node_{i-2}$
        if ( both $Auth_{RtInfo}$ values verified )
          compare route info values for $node_{i-1}$ & $node_{i-2}$
          if (route info values valid) then
            use route info for $node_{i-1}$ for dance number
          else
            use route info for $node_{i-2}$ for dance number
          end if
        else if (only $Auth_{RtInfo}$ for $node_{i-2}$ verified )
          use route info for $node_{i-2}$ for dance number
        else drop $F_{sd}$ and exit
        end if
        send $F_{sd}$ to entrance for inclusion in dancefloor
      end if
    end if
  end if
end for
```

5. SIMULATIONS FOR ATTACKS

We designed a test scenario in ns-2 to systematically evaluate the impact of attacks launched by malicious nodes in a MANET running *BeeAdHoc* and *BeeSec* protocols. We realized *BeeSec* in ns-2 while the authors of [14] provided the source code for *BeeAdHoc*.
The security framework of *BeeSec* used *Secure Hash Algorithm* (SHA1) for hashing and *Digital Signature Algorithm* (DSA) for digital signatures. To implement these security algorithms, we integrated a crypto library, OpenSSl [13] into

ns-2. We selected the recommended key length of 1024 bits. In comparison, the hash function produced a 20 byte value while the size of digital signature varied between 45 to 49 bytes. A packet can be authenticated with the keys of a node using digital signatures stored in the header of a packet. We assume all keys to be pre-distributed for the purpose of our simulations.

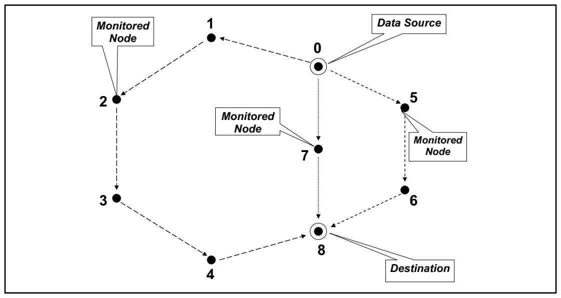


Figure 1: Node topology selected for attacks

5.1 Network Topology

We selected a rectangular area of operation ($1000 \times 500 m^2$) in which nine nodes were placed. A constant bit rate *(CBR)* source was started at *Node 0* and all data packets were destined for *Node 8*. We decided to have immobile nodes as it makes it easier to analyze the impact of an attack. However, the analysis can easily be applied, without loss of generality, to a scenario of mobile nodes.
It is evident from Figure 1 that three distinct paths exist between *Node 0* and *Node 8*; *0-7-8*, *0-5-6-8*, *0-1-2-3-4-8*. We observed that the path *0-7-8* was discovered earlier in most of the simulations. We equipped each node with a traffic scope to monitor the traffic patterns. Figure 2(a) and 2(d) show for *BeeAdHoc* and *BeeSec* respectively, the distribution of packets on different paths under normal network state. As expected, the number of packets following *0-7-8* path is significantly higher than *0-5-6-8* path. It is interesting to note that virtually no packets take *0-1-2-3-4-8* path.

5.2 Attacks on BeeAdHoc and BeeSec

We designed and implemented an *attacker framework* in ns-2 that has the ability to launch different types of attacks. The framework was instrumental in verifying the robustness of *BeeSec* to a number of attacks, mentioned in Section 3.

Attk-1: Forging Forward Scout. We gave 100 seconds initialization time with the objective that routes are initially discovered. *Node 4* launched forged forward scouts after 100 seconds of injection of data into the network. These forward scouts specified *Node 0* as the source and *Node 8* as the destination. In *BeeAdHoc*, the scouts were returned and as a result, path *0-1-2-3-4-8* got established. One can see in Figure 2(b) that malicious *Node 4* was successfully able to divert the subsequent data packets towards itself on the longest path *0-1-2-3-4-8*. In contrast, forged scouts in *BeeSec* could not be authenticated because $Auth_{FS}$ for *Nodes 0,1,2,3* were not available for verification (Algorithm 1). Consequently, the forged scouts were dropped (Figure 2(e)) that could disrupt the normal routing behavior.

Attk-2: Forging Backward Scout. In this attack, *Node 2* launched forged backward scouts at time t=100 seconds with D_{sct} as *Node 8*. As a result, the forged backward scouts also established the path *0-1-2-3-4-8*. Figure 2(c) demonstrates the success of the attack where *Node 2* was able to divert subsequent data packets towards itself. The rising point of the curve for the higher packet rate was much closer to the start time of the attack as compared to the 10 packets/sec attack packet rate shown in the figure. Again, the forged backward scouts were dropped in *BeeSec* due to failure to verify $Auth_{BS}$ (Algorithm 2). Consequently, as seen in Figure 2(f), the path *0-1-2-3-4-8* was not established.

Attk-3: Returning Scouts with Modified Route. For each received scout, malicious *Node 5* changed the source route to *8-4-3-2-1-0* and instead of broadcasting it further, sent it back as a unicast message. As a result the longer path *0-1-2-3-4-8* got established instead of the desired path *0-5-6-8*. This path was set earlier, as a result, more foragers were returned for this path. Figure 3(a) shows the impact of the attack on *BeeAdHoc*, but in *BeeSec* all such backward scouts were dropped. One can see in Figure 3(d) that the distribution of packets was limited to just two paths (*0-7-8*, *0-5-6-8*) and no packet followed *0-1-2-3-4-8*.

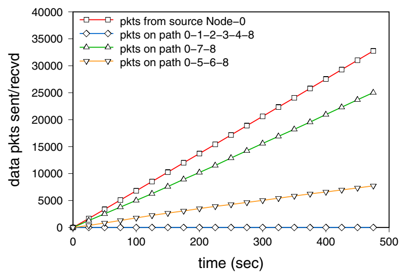
Attk-4: Forging Spoofed Forager. In this attack, at time t=150 seconds, malicious *Node 1* started sending forged foragers to *Node 0* and artificially reduced the delay value of route *0-1-2-3-4-8* to minimum. Once these forged foragers, carrying manipulated quality values, arrived at *Node 0*, due to a better quality metric more replicas of them were created. Figure 3(b) for *BeeAdHoc* shows that *Node 1* was able to divert significant number of packets towards itself. Since the forged foragers were launched at regular intervals, gradually the already established routes were abandoned in favour of this non-optimal route. But in *BeeSec*, the forged foragers were dropped because $Auth_F$ and $Auth_{RtInfo}$ could not be verified (Algorithm 4). Figure 3(e) demonstrates that the attack was successfully countered in *BeeSec*.

Attk-5: Modifying Forager Route Information. In this attack, *Node 7* artificially increased the delay value in a forager returning from *Node 8*, to make the path *0-7-8* undesirable. Figure 3(c) shows that the attack was successful on *BeeAdHoc* because the high delay value caused a significant reduction in the number of replicas for foragers of path *0-7-8*. But in *BeeSec*, *Node 0* discarded these foragers because $Auth_{RtInfo}$ for *Node 7* was not verified. In this case, the actual value of delay was restored from the delay value of *Node 8*. It can be seen in Figure 3(f) that *BeeSec* was able to counter the attack.

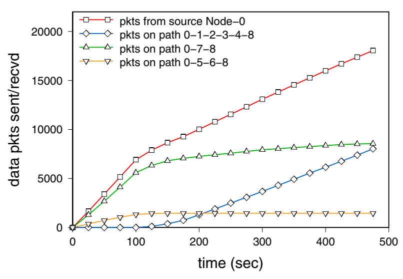
We recorded the average delay values during the experiments to see the impact of attacks on the performance of the network. We have tabulated these values in Table 2 for normal and attack scenarios. One can see that the values of delay in *BeeAdHoc* significantly increased under attacks (in

Attack	*BeeAdHoc*			*BeeSec*	
	Normal	*Attack*	*Increase*	*Normal*	*Attack*
Attk-1	63.78	178.48	179.9%	78.91	78.91
Attk-2	14.74	143.65	874.6%	21.07	21.07
Attk-3	14.74	25.30	71.6%	21.07	21.07
Attk-4	14.89	29.13	95.6%	16.51	16.51
Attk-5	14.89	18.32	23%	16.51	16.51

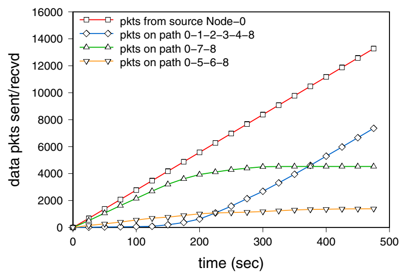
Table 2: Packet delays (ms) under attack



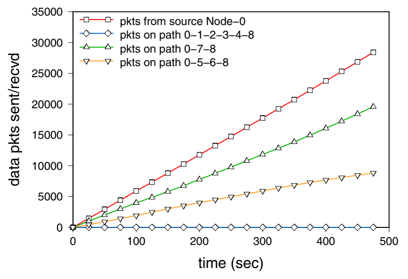
(a) BeeAdHoc: normal behavior



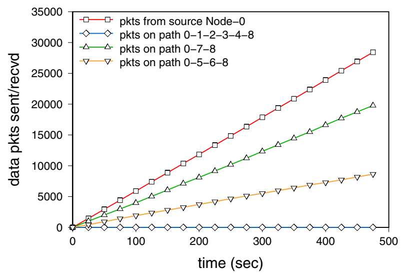
(b) BeeAdHoc: forging forward scout



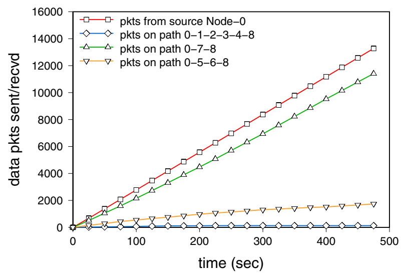
(c) BeeAdHoc: forging backward scout



(d) BeeSec: normal behavior



(e) BeeSec: forging forward scout



(f) BeeSec: forging backward scout

Figure 2: Attacks on BeeAdHoc and BeeSec (Attk-1, Attk-2), showing packet (pkt) route diversions

the range from 23% to 874%) because the packets started following the longer paths. But *BeeSec* did not suffer from this problem because it successfully countered these attacks. Note that the delay values in *BeeSec* remained unchanged.

6. SIMULATIONS FOR PERFORMANCE

The purpose of the experiments listed in current section is to empirically validate that the security framework employed in *BeeSec* did not result in significant degradation of its performance as compared to existing state-of-the-art algorithms: *BeeAdHoc*, *DSR* and *AODV*. We used the *BeeAdHoc* implementation by the authors of [14] and *DSR*/*AODV* implementations distributed with ns-2. Our tests are based on the scenario in [1]. We studied the performance of algorithms in a rectangular area of operation $2400 \times 480m^2$ by increasing the number of nodes from 10 to 60. The nodes move according to the "random waypoint" model [5]; each node selects the destination and speed randomly, and stops for a certain pause time. Speed is selected from a uniform distribution, between 1 m/s (walking speed) and 20 m/s (car speed within cities). All nodes generate constant bit rate (CBR) peer-to-peer data traffic, at a rate of 30 packets/second. Reported results are average values over five independent runs, to factor out any stochastic elements in the environment or the algorithms. Simulation time for the algorithms is set to 1000 seconds.

6.1 Metrics

We used the following metrics for a comprehensive performance evaluation of the algorithms. Here, we just provide brief definitions for a basic understanding. An interested reader should consult [14] for better insight and motivation for these parameters.

Throughput. *The number of data bits delivered to the application layer at destination in a unit interval of time.* If y bits are delivered within t time, the throughput is $\frac{y}{t}$.

Packet Delivery Ratio. *The ratio of data packets successfully delivered to the destination nodes and total number of packets generated for those destinations.*

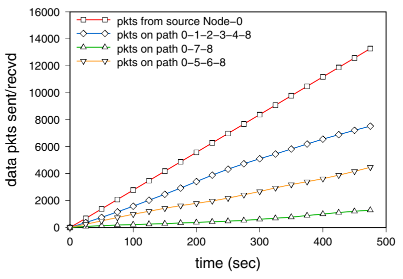
End-to-End Delay. *Time interval once a data packet is generated by the application of a node and when it got delivered to the application layer of a destination node.* It includes the *reactive wait time* that a data packet has to wait at the source node because a route to the destination node needs to be discovered.

Energy per user data. *Energy consumed in transporting one kilobyte of data to its destination.* It includes the energy consumed for both data and control traffic. We used the model presented in [2] to estimate send/receive energy for both broadcast and point-to-point mode.

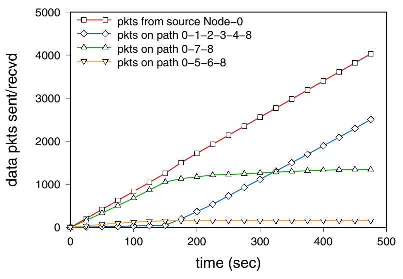
Transmission efficiency. *The number of data bytes delivered to the application at destination nodes, at the cost of a unit control byte.* Control bytes include the bytes of control packets, and additional bytes of control information in the header of each data packet for source routing algorithms.

6.2 Simulation Results

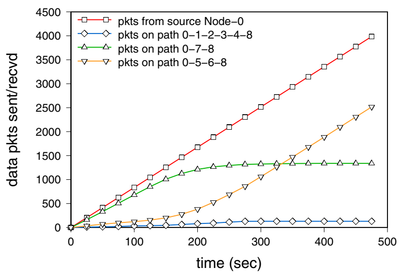
One can see in Figure 4(a) that *BeeSec*, despite complex processing and large signature carrying packets, is able to deliver the same number of packets as compared to *BeeAdHoc*, and *DSR*. But its packet delivery ratio in large MANETs is about 3% higher than *AODV*. Remember that *DSR* and *AODV* protocols are without the overhead of security extensions. Even then the end-to-end delay of *BeeSec* is significantly less than *AODV* (see Figure 4(b)) and much smaller as compared to *DSR*, especially for large MANETs. We believe that this is due to the multipath discovery and mainte-



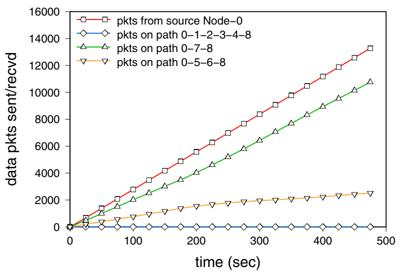
(a) BeeAdHoc: returning scouts with modified route



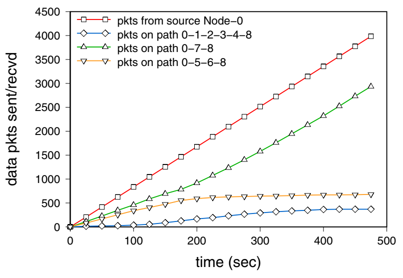
(b) BeeAdHoc: forging spoofed forager



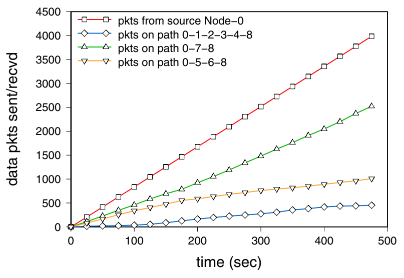
(c) BeeAdHoc: modifying route info



(d) BeeSec: returning scouts with modified route



(e) BeeSec: forging spoofed forager



(f) BeeSec: modifying route info

Figure 3: Attacks on BeeAdHoc and BeeSec (Attk-3, Attk-4, Attk-5), showing packet (pkt) route diversions

nance feature of the base protocol, *BeeAdHoc*, which ensures that the routes to destinations are almost always available. Consequently the reactive route discovery time is factored out in the end-to-end delay. As a result, the net throughput of *BeeSec*, shown in Figure 4(c), is much higher than *AODV* and *DSR*. A similar trend for energy per user data is also seen in Figure 4(d): *BeeSec* has less energy consumption as compared to *AODV* and *DSR* despite its higher communication cost of sending digital signatures with the agents. Similarly, transmission efficiency of *BeeSec*, shown in Figure 4(e), is approximately the same as compared to other algorithms for higher node MANETs. This gives us an insight that in large MANETs one needs to transmit a large number of control packets for successful delivery of data packets.

We further investigated the control overhead of the protocols by analyzing in detail their different components. Table 3 shows the measured control bytes in control packets (CB_C), in data packets (CB_D), and the aggregate (CB_T). Control bytes in control packets were further categorised as being peer-to-peer (*p2p*) or broadcast (*bc*). Table 3, along with Figure 4(f), gives us valuable insight into designing a security framework for source routing algorithms. In *BeeSec*, 84.7% of control overhead is attributable to control information inside foragers. Each forager carries authentication data for various fields including the source route. Nevertheless this overhead is still 25.5% smaller compared to a non-secure version of *DSR*.

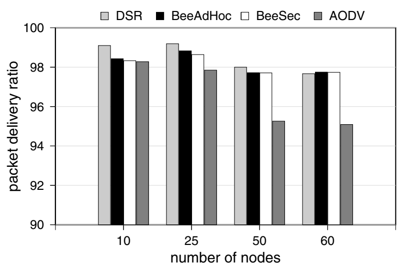
Protocol	*In Control Packets*			CB_D	CB_T
	p2p	*bc*	CB_C		
BeeAdHoc	124	362	486	1376	1862
DSR	19329	3333	22662	1348	24010
AODV	402	3495	3897	0	3897
BeeSec	239	2497	2736	15152	17888

Table 3: Control bytes for protocols (kb)

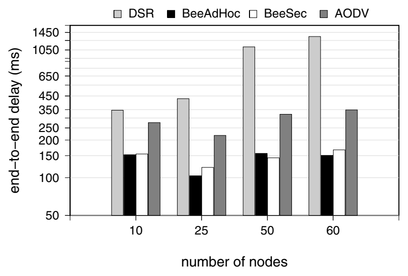
7. CONCLUSION

In this paper we have taken the first cardinal step towards providing a secure routing framework for Bio/Nature inspired MANET routing protocols. We took a well known Bio/Nature inspired routing protocol, *BeeAdHoc*, as a case study. We systematically analyzed the protocol for vulnerabilities that could enable a malicious node to launch a number of *byzantine attacks*, against *BeeAdHoc* routing. We designed and implemented a security framework, in ns-2, that allowed us to study the impact of these attacks. The results of our extensive experiments demonstrate that a malicious node can seriously disrupt the routing behavior of a protocol, which can result in significant degradation of the network performance.

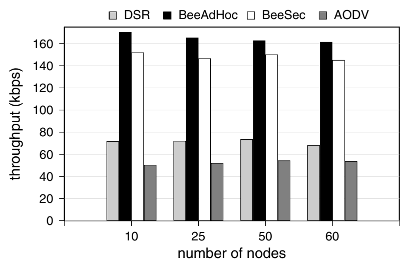
Finally, we proposed a public key (digital signature) based security framework, *BeeSec*, that can counter the attacks launched by a malicious node against *BeeAdHoc*. We believe that the framework can also be adapted easily for *AntHocNet* and *Termite*, to make these protocols secure. The results of our experiments with *BeeSec* verify that the framework was able to successfully counter the attacks launched against the routing protocol. Moreover, we have also shown that the performance metrics of *BeeSec*, including the cryptographic overhead, are either much better or close to the state-of-the-art MANET routing protocols: *AODV* and *DSR*. This gives us a good motivation to compare *BeeSec* with Ariadne [4] and SAODV [17], which are the secure versions of *DSR* and *AODV*. Concurrently, we are also developing an *Artificial Immune System (AIS)* model for securing *BeeAdHoc* in an



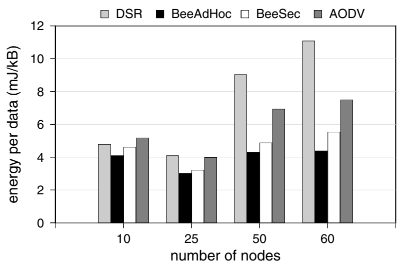
(a) Packet delivery ratio



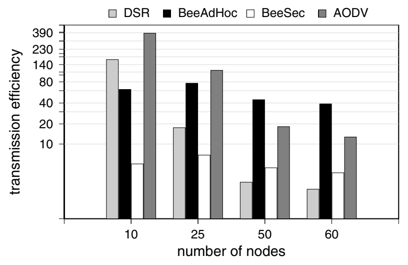
(b) End-to-End delay



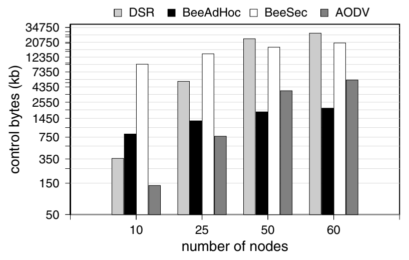
(c) Throughput



(d) Energy per user data



(e) Transmission efficiency



(f) Control byte overhead

Figure 4: Simulation results comparing BeeSec with BeeAdHoc, DSR and AODV

energy efficient manner. The research published in [15] and [11] is intriguing enough to make this effort. Our experiences and results in these efforts will be the subject of our forthcoming publications.

Contact information. The email addresses of the authors are (naumaz, farooq)@case.edu.pk.

8. REFERENCES

[1] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of Fourth ACM/IEEE Conference on Mobile Computing and Networking (MobiCom)*, pages 85–97, 1998.

[2] Laura M. Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings of IEEE INFOCOM*, 2001.

[3] L.M. Gambardella G. Di Caro, F. Ducatelle. Anthocnet: An adaptive natureinspired algorithm for routing in mobile ad hoc networks. *European Transactions on Telecommunications*, 16(2):443–455, 2005.

[4] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. *Wireless Networks*, 11(1-2):21–38, 2005.

[5] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[6] Z.J. Haas L. Zhou. Securing ad hoc networks. *IEEE Network Magazine*, 13(6), Dec 1999.

[7] S.Wicker M. Roth. Termite: Ad-hoc networking with stigmergy. In *Proceedings of IEEE GLOBE-COM*, Dec 2003.

[8] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[9] R. Perlman. Network layer protocols with byzantine robustness. PhD Thesis, Deptt of Elec. Engg. and Computer Science, MIT, 1998.

[10] E. Royer and C. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, 1999.

[11] S. Sarafijanovic and J.Y. Le Boudec. An artificial immune system approach with secondary response for misbehavior detection in mobile ad-hoc networks. *IEEE Transactions on Neural Networks*, 16(5), Sep 2005.

[12] W. Stallings. *Cryptography and Network Security - Principles and Practices*. Pearson Educ., Inc., 2003.

[13] J. Viega, Matt Massier, and Pravir Chandra. *Network Security with OpenSSL*. O'Reilly & Assoc., Inc, 2002.

[14] H.F. Wedde, M. Farooq, T. Pannenbaecker, B. Vogel, C. Mueller, J. Meth, and R. Jeruschkat. Beeadhoc: an energy efficient routing algorithm for mobile ad hoc networks inspired by bee behavior. In *GECCO*, pages 153–160, 2005.

[15] H.F. Wedde, C. Timm, and M. Farooq. Beehiveais: A simple, efficient, scalable and secure routing framework inspired by artificial immune systems. In *PPSN*, pages 623–632, 2006.

[16] H.F. Wedde, C. Timm, and M. Farooq. Beehiveguard: A step towards secure nature inspired routing algorithms. In *EvoWorkshops*, pages 243–254, 2006.

[17] Manel Guerrero Zapata. Secure ad hoc on-demand distance vector (saodv) routing. Internet-Draft, draft-guerrero-manet-saodv-05.txt, February, 2005.