ODAR: On-Demand Anonymous Routing in Ad Hoc Networks

Denh Sy, Rex Chen and Lichun Bao

Bren School of Information and Computer Sciences and Calit2

University of California, Irvine, CA 92697

Emails: {dsy, rex, and lbao}@ics.uci.edu

Abstract— Routing in wireless ad hoc networks are vulnerable to traffic analysis, spoofing and denial of service attacks due to open wireless medium communications. Anonymity mechanisms in ad hoc networks are critical security measures used to mitigate these problems by concealing identification information, such as those of nodes, links, traffic flows, paths and network topology information from harmful attackers. We propose ODAR, an On-Demand Anonymous Routing protocol for wireless ad hoc networks to enable complete anonymity of nodes, links and source-routing paths/trees using Bloom filters. We simulate ODAR using J-Sim, and compare its performance with AODV in certain ad hoc network scenarios.

I. INTRODUCTION

Privacy concerns are increasing in the Internet and wireless networks due to the mounting intrusions and attacks. Anonymity that hides the identities of various components or parts of a network communication is one of the countermeasures. Strong anonymity in network communication prevents address spoofing, route forgery, and certain denial of service (DoS) attacks by concealing the true identity of the traffic.

In the Internet, several network-based anonymity approaches provide anonymous communication between end-nodes, including DC-nets [8], Crowds [21], MIX networks [7], and Onion Routing [13]. Crowds network is a randomized packet forwarding network that obscures the packet senders by randomly throwing each packet between Crowd network nodes for a variable number of times before delivering to the final destination. However, Crowds provides no recipient anonymity, and no anonymity against a global attacker or a local eavesdropper [21].

MIX networks and *Onion Routing* use cryptographic algorithms to establish anonymous paths as well as encrypting the data packets. In Onion routing, to construct an anonymous path, a packet origin must store and maintain information about the topology of the Onion router network, out of which a number of Onion routers are selected by the packet origin to construct the path. In addition, each node on the path is supplied with a symmetric key for encrypting and decrypting data packets, and getting instructions for the next hop router in the path. When a data packet is ready to send, the source recursively encrypts the packet and the next hop information with the symmetric keys along the path. This creates a layered structure, in which it is necessary to decrypt all outer layers of the onion in order to retrieve an inner layer.

In ad hoc networks, efficient anonymity is still an elusive topic. Similar approaches to Onion routing have been used. However, keeping up-to-date information about the topology of the network is complex in the presence of dynamic ad hoc topologies. Therefore, a *trapdoor* mechanism is used to collect the source route between communicating pairs. A trapdoor is a common concept in cryptographic functions that defines a one-way function between two sets [24]. A *global trapdoor* is an information collection mechanism in which intermediate nodes may add information elements, such as node IDs, into the trapdoor. Only certain nodes, such as the source and destination nodes can unlock and retrieve the elements using pre-established secret keys. The design of a global trapdoor requires end-to-end key agreement between the source and destination nodes. AnonDSR provides a security parameter establishment (SPE) component to manage shared secrets between end-systems [23].

Global trapdoors are used in several proposals, such as SDAR (Secure Distributed Anonymous Routing) [3], AnonDSR (Anonymous DSR) [23] and SDDR (Secure Dynamic Distributed Routing) [12]. Both SDAR and AnonDSR implement onion routing scheme by collecting the symmetric keys created by intermediate nodes using a global trapdoor managed by the source and

destination nodes, thus providing both route anonymity and end-to-end data privacy. However, the identities of intermediate nodes on the source route are still visible to the source and destination nodes in such schemes.

ANODR [15] used onion routing, but is different from above approaches in that each forwarding node adds an encrypted layer to the route request message like an onion. The source and destination nodes do not necessary know the forwarding node IDs. The destination node ID in the route request is encrypted in a trapdoor that only the destination can decrypt. Kong *et al.* further proved that ASR [27] is a variant of ANODR [16].

Because the global trapdoor depends on public key encryption, AnonDSR and SDAR are better for high-end nodes that can run public key cryptography efficiently, while ANODR and ASR are more suitable for low-end nodes and medium mobility [18].

ASRP (Anonymous Secure Routing Protocol) provides authenticated anonymous on-demand routing by setting up virtual circuits between neighbors in ad hoc networks [9]. The virtual circuit IDs and data packets are encrypted with shared secrets between neighbors. MASK is another anonymous routing protocol that establishes virtual circuits between source and destination, and uses pseudo-link IDs for path presentations [26].

Unfortunately, concealing the node, link or path identities in data packets may not be sufficient for traffic anonymity due to timing analysis [20]. A technique called mixing can thwart this attack, such as sending reordered messages, inserting dummy messages, or introducing random delays [7].

We propose ODAR, an On-Demand Anonymous Routing protocol, which provides node, link and path anonymities in ad hoc networks based on Bloom filters. The use of Bloom filters additionally gives ODAR the storage-, processing- and communication-efficiencies, making it suitable in the ad hoc network environments. Castelluccia *et al.* were the first to use Bloom filters to compress source route information after the source route is discovered using DSR [6], [14]. However, no research was found so far that uses Bloom filters for anonymous source routing purposes.

The rest of the paper is organized as follows. Section II describes the essential idea of anonymous routing using Bloom filters. Section III presents the anonymity mechanisms to various types of nodes on the source route, and a key distribution mechanism for secret establishment in ad hoc networks. Section IV specifies ODAR route maintenance mechanism. Section V evaluates ODAR using simulations. Section VI concludes the paper.

II. ODAR Data Forwarding

A. Network Assumptions

We assume that a wireless ad hoc network consists of energy-, storage-, processing- and communication-constrained nodes that are deployed over a wide range of areas. The resources available at each node in the ad hoc network could be heterogeneous in that they possess different memory-, processing- and communication-capabilities, and take different roles to maintain the network communication functionalities. Wireless neighbors can communicate with one another via the wireless channel with high probability of success. The network topology is relatively static for the duration of manageable communication sessions.

Many information can work as the identification sources, such as IP, URI, URL or MAC addresses or special attributes provided during the ad hoc network deployment. For communication initialization and anonymity purposes, we assume that only the source node knows the identity of its intended destination when it wants to establish the connection. The source does not have to know the whole network topology for communication purposes, and no node has to obtain the global view of the network.

B. Anonymity Goals

We intend to provide the following anonymities in ODAR:

- *Identity anonymity*: A node receiving or sending data packets cannot be identified by its neighbors. It is computationally difficult for adversaries to search and determine the node's true identity.
- *Route/path anonymity*: A node forwarding packets must not be able to infer the identities of other nodes that also participate in the data forwarding.
- *Topology/location anonymity*: Routing information maintenance does not reveal the distance, neighbor link information of a node, nor the true routing path or tree information. Neither can they be deduced from routing information in the packets.

These goals are very different from other related security problems in routing such as resistance to route disruption or "denial-of-service" attacks. In fact, the attackers may avoid such aggressive schemes, in an attempt to be as "invisible" as possible, until it traces, locates and/or physically destroys the assets.

C. Bloom Filter

A Bloom filter is a space-efficient probabilistic bit-vector data structure for storing the elements of a set, and

testing whether or not any given element is a member of the set [1]. A Bloom filter is defined by N – the *size* or the *number of bits* in the filter N, k – the *number of hash functions*, and n – the highest *number of elements* n that it can hold under a *false positive threshold* P. Traditional hash table is a special case of Bloom filter where $k = 1$. Usually, elements are only added to the set in Bloom filters, but not removed.

There is a well-known result about the relations between these parameters, as shown in Eq. (1) [1].

$$n = -N/\ln P, \text{ where } k \approx -\log_2 P. \tag{1}$$

Bose *et al.* provides a slightly more accurate analysis of the relationship between N, n, k and P [2].

Bloom filters are widely used in resource routing in peer-to-peer networks, packet tracing, packet counting, and packet forwarding problems [4].

For clarity in the following discussions, Bloom filter elements are often represented by their identifiers directly in figure illustrations, instead of being transformed into the k bits derived from the hash functions.

D. Source Routing Using Bloom Filters

For the anonymous routing purpose, we use source routing for packet forwarding, wherein the packet is attached with the path information.
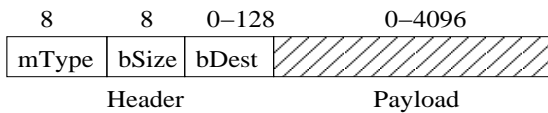

Fig. 1.   Packet Header Format and Payload for Source Routing.

Fig. 1 illustrates the packet header format in ODAR. The message type `mType` indicates the types of message contained in the `Payload` field. The destination `bDest` is a Bloom filter that contains the node IDs on the source route from the packet origin to the destination, and its size is specified by field `bSize`. For clarity in our presentations, nodes are denoted in their plaintext forms in the source route. In Section III, we describe how source, destination and intermediate forwarding nodes are securely anonymized. Although variable in applications, the exemplary field sizes in the ODAR packet header are denoted atop each field.

Fig. 2 illustrates how a packet is forwarded in the ad hoc network. When a packet is sent out to reach its destinations or predefined intermediate nodes, the complete path information is encoded in the `bDest` address field of the packet using Bloom filters. When an
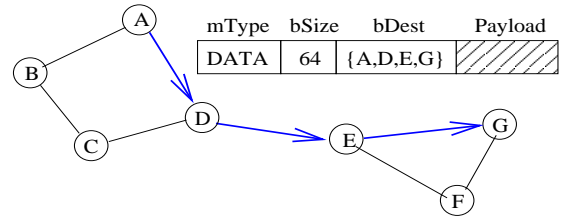

Fig. 2.   Source Routing Process in ODAR.

intermediate node receives the packet, the node inspects the Bloom filter `bDest` to see if its ID is in the Bloom filter. If true, the node simply rebroadcasts the packet.

In order to avoid routing loops, each node maintains a history of forwarded packets using Bloom filters by extracting and hashing the packet identification information, such as the header fields and portions of the payload. If a packet has been sent before, a node simply discards the packet.

The application of Bloom filters in the `bDest` field of data packets provides one of the anonymities in ODAR that conceals the network topology and path information, as well as the origin and destination information.

TABLE I

NOTATION

A, B etc.	Node identifiers.
Y_A	The long-term public key of node A.
Y_a	The session public key of node A.
X_A	The long-term private key of node A.
X_a	The session private key of node A.
K_{aB}	The shared secret between the source A and destination B.
$\text{sh}(x)_k$	Secure hash function on input bit-string x with key k, such as SHA-1 [11] or HMAC [17].
KSProp	Message to propagate key server path.
KREQ	Message to request the public key of a node.
KREP	Message for key server to send a public key to a node.
RREQ	Message to search for a source route.
RREP	Message to answer a source route query.
DATA	Data packet.

For clarity, Table I provides the symbol notation and the corresponding meanings for key establishment, anonymity and path maintenance in ODAR. Especially, six types of ODAR packets are indicated in Table I, and will be used when we describe anonymous data forwarding, key establishment and path finding operations in the rest of the paper.

III. Anonymity

We differentiate two types of identification for anonymous routing — the end-host names, and the forwarding node names. In anonymous routing, the forwarding nodes' identities are known only by the nodes themselves, and no pseudonyms are necessary. However, the destination's identity has to be known beforehand for the source to search the corresponding path. Therefore, the destination ID requires pseudonym generation methods. We provide anonymity mechanisms for the intermediate forwarding nodes and the end-hosts, respectively.

A. Intermediate Node Anonymity

As shown in the source routing process in Fig. 2, the IDs of the forwarding nodes are hashed into the `bDest` Bloom filter. Data packets are routed correctly as long as the forwarding nodes can find out that they are members of the `bDest` Bloom filter. Therefore, a simple anonymous mechanism can be used, where a node generates a secret random number for itself, and uses it as a key in secure hash functions to derive the index keys of the Bloom filter.

In addition, several private random keys can be iterated in the secure hash functions so that index key generations are further obfuscated in the `bDest` Bloom filter at a specific intermediate node. This way, attackers cannot recognize the nodes that are actively forwarding traffic. However, the node needs to carry out several rounds of secure hash operations when looking up the `bDest` Bloom filter in packet forwarding.

B. End-Host Anonymities

In network communication, names are involved at different layers of the networking stack, and reflects information such as topological or geographic locations, application-specific semantics, or data-object attributes. For anonymity purposes, these identities are replaced with pseudonyms. Interpretation of pseudonyms can only be done by nodes with knowledge of the true names, such as the node itself or the source nodes that have knowledge of the names. In addition, pseudonyms allow nodes to use longer and more complicated addressing architecture in ad hoc networks, such as 128-bit IPv6 or URL addresses.

Diffie-Hellman algorithm has been widely used in key distribution and management protocols for establishing a shared secret over an unprotected communications channel [10], [19]. In Diffie-Hellman algorithm, a prime number q and its primitive root g is chosen and published in the network. When two nodes A and B need to share

a key, node A generates a private random value $X_A < q$, and computes a public value according to Eq. (2).

$$Y_A = g^{X_A} \bmod q. \tag{2}$$

Similarly, node B generates a private value $X_B < q$ and a public value $Y_B = g^{X_B} \bmod q$. Each side keeps the private random numbers secret, and makes the public Y values available to the other side. Then the shared symmetric key is derived at both nodes A and B according to Eq. (3).

$$K_{AB} = Y_A^{X_B} = Y_B^{X_A} = g^{X_A X_B} \bmod q. \tag{3}$$

We refer to the public and private values as **keys** in this paper. In ODAR, each node possess two types of private keys. The first type is used to generate a long-term public value Y for accepting new incoming connections, and the other is session-based to create short-term public keys for connecting with destination nodes. We differentiate the long- and short-term keys with upper and lower cases, respectively. That is, we denote the long-term public key of a node A by Y_A, and the short-term session key by Y_a. Similarly, the long- and short-term private keys are denoted by X_A and X_a, respectively, as shown in Table I.

1) Key Server: In ad hoc networks, it is necessary to implement a public key distribution mechanism for session-key generations. A comprehensive survey on key distribution mechanisms can be found in [5].

For simplicity, we assume that the ad hoc network has only one key server to hold the public keys of all other nodes for key storage and distribution purposes. The space requirement on the key server is linear with regard to the number of nodes in the network.

Because ad hoc network topology changes frequently, and new nodes may join and leave the network frequently, the key server needs to maintain its presence by periodically propagating its public key with a key server message `KSProp` to all interested nodes. Additionally, the `KSProp` message also carries the traversed source route from the key server so as to establish the route from the key server to every other node in the network, thus providing the path for a node to request the public keys of other nodes later.
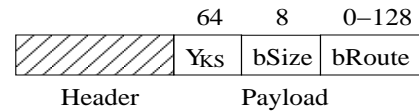
	64	8	0–128
////////	Y_{KS}	bSize	bRoute
Header		Payload	

Fig. 3. Key Server Propagation Packet Format.

The `KSProp` message is placed in the payload of an ODAR packet, and its format is illustrated in Fig. 3,

in which Y_{KS} is the public key of the key server, `bRoute` is the source route, and `bSize` is the bit-size of `bRoute`.

2) Key Distribution: A shared secret between the source and destination node is necessary to hide the source and destination identities during the source route request and response phases. The shared key derivation is achieved through the Diffie-Hellman algorithm after retrieving the long-term public key of the destination through the key server.
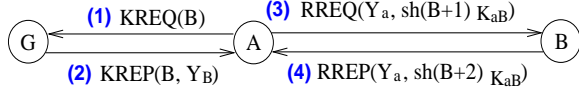

Fig. 4.   Key Distribution during Source Route Construction.

The key distribution mechanism for anonymous end-to-end communication is illustrated by four steps (1)-(4) in Fig. 4 using four types of messages. Fig. 4 illustrates that the source node A retrieves the public key Y_B of the destination node B from the key server G, and finds the route to node B anonymously. We explain the meanings of the messages as follows.
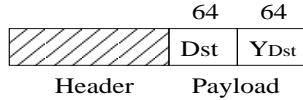

Fig. 5.   Public Key Request and Reply Packet Format.

Fig. 5 shows the format of both `KREQ` and `KREP` messages. `Dst` is the publicly known pseudonym of the destination, and Y_{Dst} is the field for placing the public key of the destination.
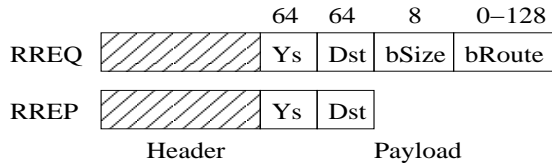

Fig. 6.   Source Route Request and Reply Packet Format.

Fig. 6 presents the route request and reply message formats, respectively. Field Y_S contains the session key of a source node S. `Dst` is the securely hashed destination pseudonym, which equals to $\text{sh}(B+1)_{K_{aB}}$ in step (3) of Fig. 4, or $\text{sh}(B+2)_{K_{aB}}$ in step (4) of Fig. 4. Fields `bRoute` and `bSize` define the Bloom filter for storing the source route. In step (4) route reply phase, the source route is provided in the ODAR header field. `B+1` and `B+2` are for authentication purposes.

3) Anonymity in the Route Discovery: The anonymous key distribution and mutually authenticated source route discovery are described as follows:

Step (1): Node A sends out a key request message `KREQ` to the key server G for the public key of node B. The key request message does not reveal the identity of node A except for indicating that some node wants to contact node B of the network.

Step (2): Upon receiving the key request message, key server G looks through its public key table and finds the public key Y_B associated with identifier B, then responds with a key reply message `KREP` to node A using the same source route contained in the key request message.

Step (3): Once node A gets the public key of node B, it generates a session key Y_a based on a private key X_a according to Eq. (2), and computes a session secret K_{aB} using Y_B, X_a according to Eq. 3. Afterward, node A uses key K_{aB} to securely hash $B+1$ by secure hash function $\text{sh}(B+1)_{K_{aB}}$, the result of which is used to identify the destination. Then, node A sends the route request message `RREQ` to find the source route to node B with two pieces of information — session key Y_a as node A's pseudonym, and destination $\text{sh}(B+1)_{K_{aB}}$ as node B's pseudonym. These pseudonyms are only recognizable by node A and node B, respectively.

Step (4): When the source route request message `RREQ` arrives at node B, node B is able to use its long-term private key X_B, generated initially, and the session key Y_a to derive the shared secret K_{aB}, and recompute $\text{sh}(B+1)_{K_{aB}}$ to see if it is the intended destination of the message. Since this is true, node B generates the route reply message `RREP` along with the session key Y_a and another secure pseudonym $\text{sh}(B+2)_{K_{aB}}$ using key K_{aB}. Pseudonym $\text{sh}(B+2)_{K_{aB}}$ authenticates node B to node A so that node A can assert session key Y_a maps to the true source route to node B in the reply message.

Steps (3) and (4) do not reveal any source and destination information with the key acquisition process. Therefore, the end-host anonymity is guaranteed. Furthermore, node B does not know who it is communicating with. Stronger *Mutual* authentication of nodes A and B can be added to the existing mechanism, but is outside the scope of this paper on anonymity.

In step (3), because session key Y_a can change per destination, the traffic characteristics per node are concealed as well.

```
/* Routing Protocol Initialization */
   Init (I)
1 {
2    if (I ≡ KEY_SERVER) {
3       new pktKeyServer (KSProp, Y_I);
4       pktKeyServer.bRoute += sh(I + 1)_{K_I};
5       Broadcast (pktKeyServer);
6    }
7 }
```

Fig. 7.   ODAR Specification: Initialization.

```
/* Send a packet to destination D with session key Y_i */
   SendPkt (I, Y_i, D, pkt)
1 {
2    if (Y_i ∈ mySessionInfo and
          mySessionInfo[Y_i].bDest≠0) {
       // Route to D exists.
3       pkt.bDest = mySessionInfo[Y_i].bDest;
4       Broadcast (pkt);
5    }
6    else { // Find the route to node D.
7       mySessionInfo += (Y_i, D);
8       if (mySessionInfo[Y_i].Y_D ≠0) {
          // If have the public key of D, then send RREQ.
9          K_{iD} = Diffie-Hellman(Y_i, Y_D);
10         new pktPath (RREQ, Y_i, sh(D + 1)_{K_{iD}});
11         pktPath.bRoute += sh(I + 1)_{K_I}; /* Secure hash */
12         Broadcast (pktPath);
13         mySessionInfo[Y_i].Queue += pkt;
14      }
15      else if (myKeyKS≠0) {
          // If have no key of D, then request it from key server
16         new pktKey (KREQ, myKeyKS, D);
17         pktKey.bDest = myPathKS;
18         Broadcast (pktKey);
19         mySessionInfo[Y_i].Queue += pkt;
20      }
21      else { // No key server, no communication.
22         Drop (pkt);
23      }
24   }
25 }
```

Fig. 8.   ODAR Specification: Outgoing Packet Transmission.

IV. ODAR ROUTING CONTROL

Routing control protocols are generally categorized into two types — proactive and reactive. ODAR is a reactive on-demand routing control algorithm, and follows the usual route request and reply mechanism as used by other similar protocols, such as DSR [14] and AODV [22]. The packet formats used by route request and reply mechanisms are defined in Fig. 6 when we describe the anonymity of ODAR.

The path discovery process is activated when an out-going packet coming from the application layer cannot find a route to the destination, and starts by the node sending out a RREQ message to the network. If the destination receives the RREQ message, it responds with a route reply message RREP to return the complete source route to the source.

Fig. 7-9 specify ODAR related functions, such as ODAR initialization **Init**(), packet transmission **SendPkt**() and packet processing **Process**(), respectively, using C-style pseudo-codes. The notation scheme is that data structure names and their members are written in `true-type` font, function names and reserved words are in **bold-face** font, and comments are in *italic* font. The local node's ID is represented by symbol `I`.

The variable and function names are mostly self-explanatory. Specifically, local data structures of node `I` include:

- `mySessionInfo` is an array of all end-to-end session information, such as the session identifier Y_i, destination ID, anonymous source route `bDest`, and the packet queue manager `Queue` of the session.
- Bloom filter `myPktSent` records the packets that node `I` has sent out. The packet identification information include ODAR packet header plus fields Y_S and `Dst` of the payload field if the message is route request RREQ.
- `myKeyKS` and `myPathKS` store the public key and path to the key server.
- The key server node also maintains a key list `PublicKeyList`, storing all node IDs and their public keys in the network.

In function **Init**() of Fig. 7, the node checks whether it is the key server KEY_SERVER. If so, it sends out the `KSProp` message to initialize the key distribution infrastructure (**Init** lines 2-6).

In **SendPkt**() of Fig. 8, the node checks whether it has the source route to the destination. If yes, the packet is directly sent out (**SendPkt** lines 2-5). Otherwise, the packet is for a new session, and the node looks up the session information list to find the public key of the destination. If the node has contacted the destination before, it has the public key, and a new route request is sent to the network(**SendPkt** lines 8-14). Otherwise, the public key of the destination has to be requested from the key server first(**SendPkt** lines 15-20). If the key server is unknown so far, the packet has to be dropped (**SendPkt** lines 21-24).

In **Process**() of Fig. 9, the node asserts that it has not sent the packet before (**Process** lines 3-5). Then,

```
   /* Process packet pkt received from network interface. */
   Process (I, pkt)
1  {
2    bool forward = false;
3    if (pkt ∈ myPktSent) { // Loop prevention.
4      return;
5    }
6    switch (pkt.mType) {
7      case KSProp:
8        if (myPathKS ≡ 0) {
           // Save key server's public key and path.
9          myKeyKS = pkt.Y_KS;
10         myPathKS = pkt.bRoute;
11         myPathKS += sh(I + 2)_{X_I}; // Remember the path.
12         pkt.bRoute += sh(I)_{X_I}; // Keep growing the path.
13         forward = true;
14       }
15     case KREQ or KREP:
16       if (pkt.mType ≡ KREQ and I ≡ KEY_SERVER) {
17         pkt.mType = KREP; // Retrieve the key.
18         pkt.Y_Dst = PublicKeyList (pkt.Dst);
19         forward = true;
20       }
21       else if (pkt.mType ≡ KREP and
             sh(I + 2)_{X_I} ∈ pkt.bDest) {
22         mySessionInfo.Process (pkt.Dst, pkt.Y_Dst);
23       }
24       else if (sh(I)_{X_I} ∈ pkt.bDest) {
25         forward = true;
26       }
27     case RREQ:
28       K_{sI} = Diffie-Hellman(pkt.Y_s, Y_I);
29       if (sh(I + 1)_{K_{sI}} ≡ pkt.Payload.Dst) {
30         pkt.mType = RREP;
31         pkt.bSize = pkt.Payload.bSize;
32         pkt.bDest = pkt.Payload.bRoute + sh(I + 2)_{K_I};
33         pkt.Payload.Dst = sh(I + 2)_{K_{sI}};
34         forward = true;
35       } // Reached destination, send RREP.
36       else {
37         pkt.bRoute += sh(I)_{X_I};
38         forward = true;
39       } // Keep searching for destination.
40     case RREP:
41       if (sh(I + 1)_{X_I} ∈ pkt.bDest and
             pkt.Payload.Y_s ∈ mySessionInfo) {
           // Arrived at source of the RREQ.
42         mySessionInfo += (Y_s, pkt.bDest);
43         mySessionInfo[Y_s].Queue.Process ();
44       }
45       else if (sh(I)_{X_I} ∈ pkt.bDest) {
46         forward = true;
47       }
48     case DATA:
49       if (sh(I)_{X_I} ∈ pkt.bDest) {
50         forward = true;
51       }
52       else if (sh(I + 1)_{X_I} ∈ pkt.bDest) {
           // Process payload at source.
53         ProcSource (pkt.Payload);
54       }
55       else if (sh(I + 2)_{X_I} ∈ pkt.bDest) {
           // Process payload at destination.
56         ProcDest (pkt.Payload);
57       }
58   }
59   if (forward ≡ true) {
60     Broadcast (pkt);
61     myPktSent += pkt; // Hash into Bloom filter.
62   }
63 }
```

Fig. 9. ODAR Specification: Packet Processing.

the packet is processed according to its type. If the packet is key server propagation message KSProp, the node records the source route (**Process** lines 10-11), and keeps propagating the packet (**Process** lines 12-13). If the packet is key request KREQ, and the node is the key server, a key reply is sent back (**Process** lines 16-20). Otherwise, if the packet is key reply KREP, the node checks if it is the destination of this packet, and processes the key information if so (**Process** lines 21-23). The node may also forward the packet if it is on the source route (**Process** lines 24-26).

In case the packet is routing control message for route request RREQ, the node first checks if it is the destination. If so, a route reply message RREP is composed and sent (**Process** lines 28-35). Otherwise, the node keeps propagating it (**Process** lines 36-39). In case the packet is a route reply message RREP, the node first checks if it is the origin of the source route in the packet. If

so, the node records it, and processes queued messages accordingly (**Process** lines 41-44). Otherwise, the node forwards the packet if it is on the source route (**Process** lines 45-47).

In case the packet is a data packet DATA, the node either forwards it (**Process** lines 49-51), or processes it (**Process** lines 52-57).

Note that in source route, the source and destination nodes stores themselves differently. The source I always identify itself with ID I+1, while the destination with I+2. This is for the direction of the data flow, which effects how the application layer processes the packet.

Several functions are **not** specified further in Fig. 7-9, such as APIs provided either by the lower layer (*e.g.* **Broadcast** which sends a packet through the wireless medium), or by the upper layer (*e.g.* **ProcSource** and **ProcDest** which process the packet payload information at the source and destination nodes, respectively). **Diffie-**

Hellman is the shared secret computation function according to the public and private keys using Eq. (3).

V. EVALUATIONS

A. Security Analysis

ODAR provides a novel use of Bloom filters for efficient and anonymous routing in ad hoc networks. The `bDest` field in data packets contains the whole source route information in a single Bloom filter.

ODAR can provide different levels of anonymity as implemented in Section III. First, the true node identities are hidden by using pseudonyms. Secondly, forwarding node identities are hidden by securely hashing the node identity into Bloom filters of the source routes. Third, the source route is completely hidden due to the node ID encryption and Bloom filter storage (Section II-D).

The "man-in-the-middle" attack commonly known to the Diffie-Hellman method poses limited threat to the source route anonymity because the public key of the key server can be easily certified and distributed beforehand for authentication. ODAR inherently thwarts address spoofing attacks because the source route are anonymous and network topology information is hidden from the perspective of any node. Therefore, the success rate of a spoofed packet getting through the network is exponentially reduced.

Nonetheless, ODAR does present some shortcomings. The routing information aggregation process, high saturation ratio of the source route Bloom filter can all raise the false positive rates in the source routes, thus causing unnecessary packet forwarding.

Furthermore, ODAR cannot prevent nodes on the source path from injecting invalid packets and staging denial of service attacks in ad hoc networks, which consume the energy of the network nodes.

Due to space limitations, we have not discussed the scalability problem in ODAR, which can be solved by three approaches: increasing the size of the `bDest` Bloom filter, providing multiple `bDest` fields, or applying hierarchical routing architecture.

B. Simulations

ODAR was simulated using J-Sim [25], a component based network simulator written in Java. In the simulations, a hundred homogeneous nodes are arranged such that nodes are approximately 80 meters (m) apart from each other in a $400 \times 1600m^2$ rectangular area. The Free Space propagation model was used with channel bandwidth 2 Mbps. A CBR traffic flow is created in each simulation between two nodes that are two, four,

and eight hops aways in three scenarios, respectively. The rate of the CBR traffic is 10 packets/second, and each packet size is 512 bytes. The simulations ran for 100 seconds.

For comparison purposes, AODV routing algorithm was also simulated in the same environments as ODAR, therefore a total of six different test scenarios are generated, three for each of ODAR and AODV. AODV uses IEEE 802.11 DCF using DATA/ACK procedure with no RTS/CTS, while ODAR uses IEEE 802.11 DCF broadcast mechanism without ACK.

In the simulations, the numbers of control and data packets in the whole network are collected as well as the numbers of data packets injected and received at the source and destination nodes, which are used to compute the delivery ratios.
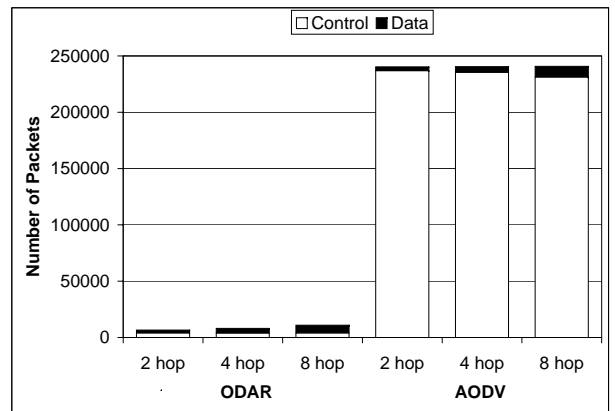

Fig. 10.   Total Number of Packets Generated.

Fig. 10 presents the number of control and data packets for ODAR and AODV, respectively. In AODV, control packets consist of neighbor HELLO packets, route request, route reply, and route errors. Control packets in ODAR are attributed to packets for initial flooding of key server path and route discovery packets in the routing control algorithm. In all scenarios, the number of packets in ODAR was about 4,000, while that of AODV was much higher at 230,000 packets due to continuous HELLO packets. This is more clearly illustrated in the following Fig. 11.

Fig. 11 presents a comparison on the number of control packets in ODAR and AODV. For ODAR, initially there are large number of packets, which represents flooding of the key server propagation followed by the routing control algorithm. For AODV, the initial spike is a result of route request and route reply. Thereafter, the number of control packets are HELLO messages
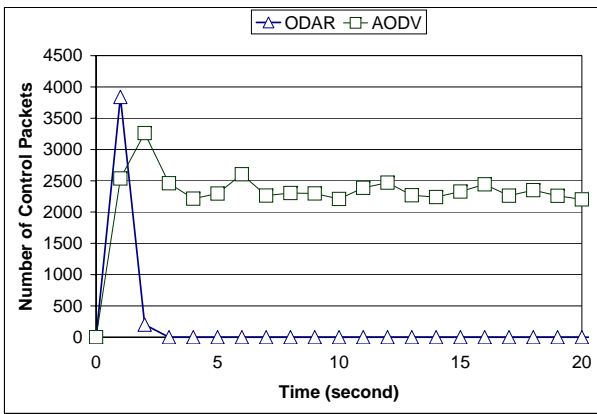
Fig. 11. The Numbers of Control Packets in ODAR and AODV.

Fig. 13. Delivery Ratios for AODV and ODAR.

for neighborhood maintenance, and remains relatively constant over the time interval in AODV. In contrast, no more control packets are sent after the path finding algorithm is completed at about three seconds into the simulation in ODAR.
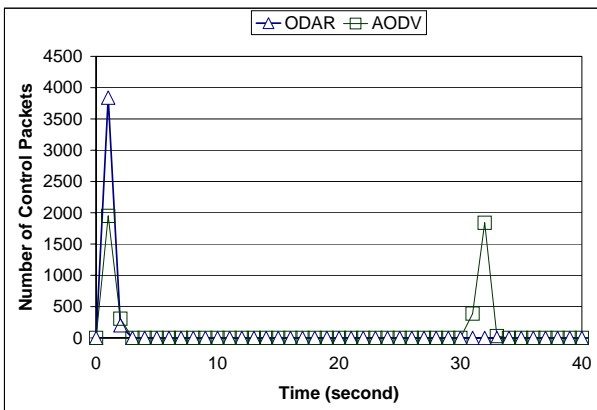

Fig. 12. Control Packets in ODAR and AODV in Similar Settings.

For a fair comparison, we virtually turned off the neighborhood maintenance messaging by setting the HELLO message intervals at 30 seconds in AODV. Fig. 12 compares the number of control packets in ODAR and the modified AODV, in which a spike appears at approximately 30 second. As we can see that ODAR performs close to AODV in terms of control overhead under similar settings.

Fig. 13 illustrates the delivery ratios of ODAR and AODV, respectively. AODV achieves higher delivery ratios than ODAR when the number of hops to traverse increases. The reason lies not in the routing protocol but in the MAC layer acknowledgments and the backoff mechanisms. AODV benefits from MAC layer acknowl-
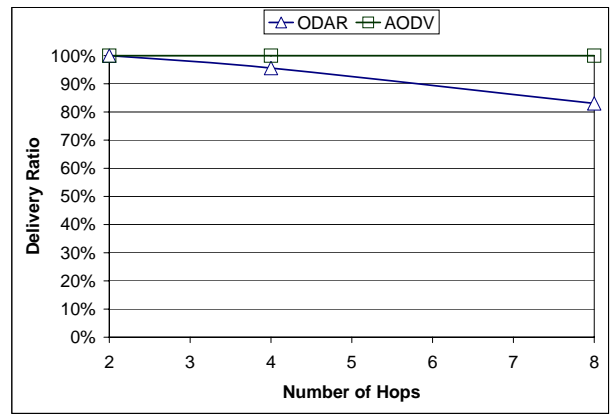
edgments with retransmission, while ODAR does not. In ODAR, reliability based on retransmissions needs to be handled by the higher level protocols.

VI. Conclusion

We have presented ODAR, a novel On-Demand Anonymous Routing protocol in ad hoc networks using Bloom filters. We have described the mechanisms to efficiently store source routes anonymously, and to forward data packets anonymously. A key management mechanism is described in order provide strong anonymity for end-to-end communications. Performance comparison between ODAR and AODV indicates ODAR has comparable control overhead to AODV.

Acknowledgment

References

[1] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, 13(7):422–426, Jul. 1970.

[2] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang. On the False-Positive Rate of Bloom Filters. *Submitted Under Review*, 2004.

[3] A. Boukerche, K. El-Khatib, L. Xu, and L. Korba. A novel solution for achieving anonymity in wireless ad hoc networks. In *Proc. of the 1st ACM PE-WASUN*, 2004.

[4] A. Broder and M. Mitzenmacher. Network applications of Bloom filters: a survey. In *Proc. of the 40th Annual Allerton Conference on Communication, Control, and Computing*, 2002.

[5] S.A. Camtepe and B. Yener. Key Distribution Mechanisms for Wireless Sensor Networks: a Survey. Technical report, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, Mar. 23 2005. TR-05-07.

[6] C. Castelluccia and P. Mutaf. Hash-Based Dynamic Source Routing. In *IFIP Networking, LNCS 3042*, pages 1012–23, 2004.

[7] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 4(2), Feb. 1982.

[8] D. Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptography*, 1(1):65–75, 1988.

[9] Yi Cheng and Dharma P. Agrawal. Distributed Anonymous Secure Routing Protocol in Wireless Mobile Ad Hoc Networks. In *OPNETWORK 2005*, Aug. 2005.

[10] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transaction on Information Theory*, 22:644–654, Nov. 1976.

[11] E. Eastlake and P. Jones. RFC 3174 - US Secure Hash Algorithm 1 (SHA1). Technical report, IETF, Sep. 2001.

[12] K. El-Khatib, L. Korba, R. Song, and G. Yee. Secure dynamic distributed routing algorithm for ad hoc wireless networks. In *International Conference on Parallel Processing Workshops (ICPPW)*, 2003.

[13] D. Goldschlag, M. Reed, and P. Syverson. Onion Routing for anonymous and private internet connections. *Communications of the ACM*, 42(2):39C4, 1999.

[14] D.B. Johnson and D.A. Maltz. *Mobile Computing*, chapter Dynamic Source Routing in Ad Hoc Wireless Networks, pages 153–181. Kluwer Academic Publishers, 1996.

[15] J. Kong and X. Hong. ANODR: ANonymous On Demand Routing with Untraceable Routes for Mobile Ad-hoc Networks. In *MOBIHOC*, 2003.

[16] Jiejun Kong, Xiaoyan Hong, Mario Gerla, and M.Y. Sanadidi. Comparison: ASR is a Variant of ANODR. Technical report, UCLA, 2005.

[17] H. Krawczyk, M. Bellare, and R. Canetti. RFC 2104 - HMAC: Keyed-Hashing for Message Authentication. Technical report, Network Working Group, 1997.

[18] Jun Liu, Jiejun Kong, Xiaoyan Hong, and Mario Gerla. Performance Evaluation of Anonymous Routing Protocols in Mobile Ad-hoc Networks. In *IEEE Wireless Communications and Networking Conference (WCNC)*, Las Vegas, Nevada, USA, Apr. 3-6 2006.

[19] R. C. Merkle. Secure Communication over an Insecure Channel. *Communication of ACM*, 21:294C99, Apr. 1978.

[20] J.-F. Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. *DIAU Lecture Notes in Computer Science*, page 10C29, 2000.

[21] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, Nov. 1998.

[22] E. M. Royer and C. E. Perkins. Multicast Operation of the Ad hoc On-Demand Distance Vector Routing Protocol. In *Proc. of MOBICOM*, pages 207–218, Seattle, WA, Aug. 1999.

[23] R. Song, L. Korba, and G. Yee. AnonDSR: Efficient Anonymous Dynamic Source Routing for Mobile Ad-Hoc Networks. In *ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, 2005.

[24] W. Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall, Upper Saddle River, NJ, Jul. 15 1998.

[25] H. Tyan. J-Sim. http://www.j-sim.org/.

[26] Y. Zhang, W. Liu, and W. Lou. Anonymous communications in mobile ad hoc networks. In *Proc. of the 24th International Conference of the IEEE Communications Society (INFOCOM)*, 2005.

[27] B. Zhu, Z. Wan, M. Kankanhalli, F. Bao, and R. Deng. Anonymous secure routing in mobile ad-hoc networks. In *Proc. of the 29th Annual IEEE International Conference on Local Computer Networks (LCN)*, pages 102–8, 2004.