Anonymous DTN routing

August 17, 2013

1 Introduction

In this project, we propose a routing protocol which provides *identity anonymity* and *location anonymity* to the participants in DTN. Identity anonymity means that the identity of packet sender or destination is not revealed to any other nodes whom the sender or destination don't trust. Location anonymity implies that the geographic location of a node cannot be tracked by other nodes whom the node doesn't trust.

To achieve identity anonymity and location anonymity, we use the concept of *ephemeral ID*. A node generates an ephemeral ID periodically and the other nodes whom it trusts only can map the ephemeral ID to the permanent ID. Then the node uses an ephemeral ID instead of its permanent ID to communicate with other nodes. Since a packet delivered through the proposed routing protocol does not contain permanent IDs of sender and destination, other untrusted nodes cannot learn the permanent IDs of sender and destination from the packet. Also, it is not possible to track a specific node since the ephemeral IDs of nodes are changed periodically.

1.1 Attack model

1. Passive global eavesdropper.

2. Active attacker who is able to compromise a subset of untrusted nodes

3. Location tracking by learning permanent node ID through eavesdropping (passive attacker) or message handshake (active attacker)

4. No analog finger printing.

1.2 Problem Definition

1. Sender identity anonymity

   - Only packet sender and destination learn the ephemeral ID and permanent ID of the packet sender.
   - Passive and active attackers learn nothing about the packet sender.
   - Nodes that the sender trusts cannot learn anything about the packet sender.

2. Receiver identity anonymity against untrusted nodes

- Only packet sender, destination and trusted nodes of the packet destination learn the ephemeral ID and permanent ID of the packet destination.

- Passive attackers learn nothing about the packet destination.

- Active attackers learn the ephemeral ID of the destination, but cannot map the ephemeral ID to the actual node/device.

3. Unlinkability

- Passive or active attackers are not able to link any observed packets sent from a node to packets received by another node.

4. Location anonymity

- Passive or active attackers cannot link an observed ephemeral address to the actual node/device.

5. Reasonable efficiency (delivery rate, bandwidth, delivery latency)

- Compared to underlying routing protocol (TBD)
- Compared to other anonymous routing protocols (ARDEN, TOR)

1.3 Assumption

1. Each node has a group of trusted nodes and knows permanent IDs of the trusted nodes.

2. Loosely synchronized time between nodes.

To make the problems simpler, we set a few more assumptions which may be removed or changed later.

3. A node knows random seeds and ephemeral IDs of its trusted nodes.

4. Every node follow the locally identical epoch schedule. Since the nodes are loosely synchronized, epoch schedule of each node is also loosely synchronized.

5. Mutual trust: If Alice trusts Bob, Bob also trusts Alice.

6. Anonymous routing is possible only when a packet sender trusts a packet destination.

1.4 Project Goals

In this project, we try to solve the problems stated in 1.2 by proposing a routing protocol which provides anonymity to nodes in DTN through the use of ephemeral ID. In addition, the proposed protocol should show reasonable efficiency in terms of packet delivery rate, bandwidth and packet delivery latency.

Our protocol achieves sender identity anonymity and weak receiver identity anonymity against untrusted nodes through the use of ephemeral ID. Each node i in DTN generates and updates its own ephemeral ID, and other nodes that i trusts are able to generate the current ephemeral ID of i at any time. Given the permanent ID and the random seed number of i, the nodes that i trusts

can generate the ephemeral ID of i at any time. However, attackers or nodes that i does not trust cannot learn the permanent ID of i. Also the attackers may receive the current ephemeral ID of i from i or other nodes, but they cannot generate ephemeral ID of i on their own. Therefore, even though the attackers observe or receive packets sent by i, they cannot determine the permanent ID of i. Moreover, the attackers recognize packets from a single node over different epochs as several streams of packets from several nodes, since the ephemeral ID of the sender is changed periodically.

Unlinkability is also achieved by the use of ephemeral ID. Unlinkability may be broken if packet delivery from a sender to a destination is completed within a single epoch, that is, ephemeral IDs of the sender and destination are not changed during the delivery. Even in this case, attackers are only able to learn ephemeral IDs of the sender and destination and cannot map the ephemeral IDs to the permanent IDs. Moreover, the length of epoch is set to be reasonably short so that ephemeral IDs of sender and destination contained in a packet would be changed at least once during the delivery.

Location anonymity is achieved by the use of ephemeral ID which is changed periodically. Since any kind of analog finger printing is not considered in this project, attackers cannot track a node without knowing a permanent ID and a random seed of a victim node that are used for generating ephemeral ID.

Efficiency of the protocol is not fully considered in the current document. We try to avoid pure epidemic routing for bandwidth efficiency but for now, there's not much to say about packet delivery rate and packet delivery latency.

2 Protocol

2.1 Notation

- m: Message to be sent. All messages are of the same size.

- $c \leftarrow \{m\}_k$: Public (symmetric) key encryption of message m.

- $m \leftarrow \{c\}_{k^{-1}}$: Public (symmetric) key decryption of c.

- i: A node in DTN. $i \in N$.

- id_i, eid_i: Permanent ID and ephemeral ID of node i.

- r_i: A random seed number of i.

- G_i: A set of permanent IDs of trusted nodes of i.

- R_i: A set of random seeds of nodes in G_i.

- E_i: A set of ephemeral IDs of nodes in G_i.

- H_i: A set of ephemeral IDs of contact history (neighbor nodes) of i. H_i contains ephemeral IDs that are not in E_i.

- BF_{E_i}: Bloom filter encoding of ephemeral IDs of nodes in G_i.

2.2 Network initialization

Inputs: A node i has G_i and R_i.

Outputs: A node i outputs eid_i and E_i, a set of ephemeral IDs of the nodes in G_i. H_i, the neighbor node list of i, is initialized.

```
 1: procedure GENEPHEMERALID(id, r, t)
 2:     eid ← H(id, r, t)
 3:     return eid
 4: end procedure
 5:
 6: procedure INITIALIZE( )
 7:     eid_i ← GenEphemeralID(id_i, r_i, t)
 8:     for all eid_j ∈ E_i do
 9:         E_i.eid_j ← GenEphemeralID(G_i.id_j, R_i.r_j, t)
10:     end for
11:     for all eid_d ∈ P_i do
12:         if P_i.eid_d ∈ E_i then                              ▷ i trusts d ↔ d trusts i.
13:             P_i.eid_d ← GenEphemeralID(G_i.id_d, R_i.r_d, t)
14:         end if
15:     end for
16:     H_i ← NULL
17:     return
18: end procedure
```

When a node i enters a DTN network, it is assumed that i is given id_i, r_i, G_i and R_i. Then i is able to build E_i, a set of ephemeral IDs of nodes in G_i by executing `Initialize()` procedure. On entering a DTN network or at the end of each epoch, a node i executes `Initialize()` to generate new parameters eid_i and E_i. eid_i, an ephemeral ID of i, is generated by `GenEphemeralID()` procedure where eid_i is generated using a hash function $H()$ which takes the permanent id id_i, a random seed r_i and time t as inputs (Line 7). Then i updates E_i and P_i by generating new ephemeral IDs of nodes in G_i (Lines 8-15). At the end of each epoch, a node i resets H_i since it cannot generate new ephemeral IDs of untrusted nodes (Line 16).

Ephemeral ID or changes of ephemeral ID should not reveal anything about a node. An attacker should not be able to learn a permanent ID from an ephemeral ID. Also, an attacker should not be able to track a specific node by monitoring an ephemeral ID of the target node. In order to prevent such attacks, we apply the same epoch schedule to all nodes in DTN. Since nodes in DTN are assumed to be loosely synchronized to each other, an identical epoch schedule can be applied to all the nodes. That is, every node in DTN changes its own ephemeral ID almost at the same time. Therefore, it would not be possible for an attacker to identify a specific node by monitoring its unique timing of ephemeral ID changes.

4

2.3 Connection setup between two nodes

Inputs: A node i has E_i, a list of ephemeral IDs of the nodes in G_i, and P_i, a list of ephemeral IDs of destinations of packets stored in i.
Outputs: Each node outputs beacon. A relay node i outputs a hello message *hello*. Next hop node j outputs a pulling message *pull*.

```
 1: procedure GENBEACON( )
 2:     beacon_j ← {eid_j}
 3:     return beacon_j
 4: end procedure
 5:
 6: procedure GENHELLO( )
 7:     hello_ij ← {eid_i ‖ BF_{P_i}}
 8:     return hello_ij
 9: end procedure
10:
11: procedure PULLPACKETS(BF_{P_i} )
12:     pull_ji ← {BF_{P_i} ∩ BF_{(E_j ∪ H_j)}}
13:     return pull_ji
14: end procedure
```

After network initialization, a node j advertises its presence to other nodes by generating a beacon message which contains eid_j, an ephemeral ID of j (Lines 1-4). Upon receiving a beacon message from j, i may send j a hello message which contains a packet digest of i, that is, a bloom filter encoding of P_i (Lines 6-9). Since BF_{P_i} contains ephemeral IDs of destinations of packets stored in i, j can calculate an intersection of P_i and $(E_j \cup H_j)$ and send i a pulling message with the calculated intersection.

Beacon message of j should not reveal more than a current ephemeral ID of j. Since eid_j is changed at each epoch, an attacker would not be able to track j over different epochs. If beacon messages are transmitted exactly periodically, an attacker would be able to track a specific node by monitoring the timing of beacon message transmission. Therefore each node may put certain time offset to vary its beacon period.

An attacker may track a node i based on the uniqueness of BF_{P_i} contained in hello message $hello_i$. Assuming that an attacker knows ephemeral IDs of other nodes, BF_{P_i} reveals that ephemeral IDs of destination nodes packets stored in i. However, the leakage of ephemeral IDs of packet destinations would not explicitly reveal G_i or E_i. Moreover, i would come across with other nodes in DTN and exchange packets with them. Packets stored in i are replicated to many other nodes and it would diminish the uniqueness of BF_{P_i} over time. Therefore, it won't be feasible for an attacker to track a node i by its packet digest BF_{P_i}.

2.4 Packet construction

Inputs: A sender s knows the ephemeral ID of the destination n_d and shares a symmetric key k_{sd} with d.

Outputs: s outputs a packet p.

1: **procedure** GENPACKET(m, k_{sd}, eid_d)
2:     $c \leftarrow \{id_s \| m\}_{k_{sd}}$
3:     $p \leftarrow \{eid_d \| c\}$
4:     **return** p
5: **end procedure**

A packet sender s constructs a packet by first encrypting the message along with its ephemeral ID id_s using a symmetric key k_{sd} shared between the sender and destination of the packet. Then the sender generate p which contains the ephemeral ID of the destination eid_d and the encrypted message c.

The packet can be accessed by all relay nodes on the route from the source to the destination, regardless of whether the relay nodes are trusted by the destination node or not. Therefore, we need to assure that any untrusted relay node cannot learn anything from the packet, and any trusted relay node cannot learn anything about the message m included in the packet. Since the message m is encrypted using a symmetric shared key k_{sd}, any relay node without the key ksd cannot learn the message m. The only things that a relay node can learn is an ephemeral ID of a source node. However, untrusted relay node cannot learn the permanent IDs of the source or destination nodes from their ephemeral IDs.

2.5 Packet forwarding

Inputs: An intermediate node i has a packet p to forward. Nodes i and j are on contact. i has $pull_{ji}$, a pulling message of j, and a symmetric key k_{ij}.

Outputs: i outputs an encrypted packet p.

1: **procedure** FORWARDPACKET($p, eid_j, pull_{ji}$)
2:     **if** $p.eid_d = eid_j$ **then**   ▷ j is the destination of p. p is assigned the highest priority (0).
3:         $p' \leftarrow \{p\}_{k_{ij}}$
4:         InsertTxQueue($p', eid_j, 0$)
5:         RemoveRxQueue(p)
6:         **return**
7:     **else if** $p.eid_d \in pull_j$ **then**   ▷ p is pulled by j. p is assigned the second higest pririty (1).
8:         $p' \leftarrow \{p\}_{k_{ij}}$
9:         InsertTxQueue($p', eid_j, 1$)

```
10:    else                                    ▷ p is assigned the lowest pririty (2).
11:        p' ← {p}_{k_{ij}}
12:        InsertTxQueue(p', eid_j, 2)
13:    end if
14:    return
15: end procedure
```

A node i has a packet to relay, or has generated a packet and wants to send it to destination node. Then whenever i contacts j and receives a pulling message from j, i performs a procedure `ForwardPacket()` to forward the packet.

If j is the destination node of the packet, i insert the packet into its TX queue and set the highest priority (0) for the packet. Then i removes the packet from its RX queue (Lines 2-6). If j is not the destination but trusted by the destination d, then i inserts the packet into the TX queue with the priority (1) which is lower than (0). i keeps the packet for further replication (Lines 7-9). If j is neither the destination nor a node trusted by d, i insert the packet into the TX queue with the lowest priority (2). Those packets with the lowest priority are sent to j only after all the other enqueued packets with higher priority are forwarded to j (Lines 11-13).

The forwarding protocol should work properly even if a few untrusted nodes are compromised and try to receive packets from a relay node, possibly by using a fabricated beacon message. A compromised node j may try to fabricate its pulling message so that a relay node i considers j as a proper next hop node. An example of the attack would be pulling all packets stored in i. This attack would waste bandwidth resource and may deter packet forwarding to a proper route. However, this kind of attack would not prevent i from forwarding its packets to other legitimate nodes, and those nodes would be able to forward the packets through proper routes to the destination.

3 Simulator

3.1 DTN Code for ns-2 and ns-3

- `https://wiki.aalto.fi/display/~jlakkako@aalto.fi/DTN+Code+for+ns-2+and+ns-3}`

- The latest version has been released recently in 2012.

- Supported DTN routing protocols

    - Epidemic routing
    - Binary spray and wait

- Supported network model

    - Random walk
    - SF cab trace or Helsinki city center trace (from ONE simulation)

- Other functionality

    - Return receipts / antipackets
    - A simple congestion control algorithm

3.2 DTNSim2

- `http://watwire.uwaterloo.ca/DTN/sim/`

- The latest version has been released in 2006.

- Used in ARDEN paper. Seems like an official simulator introduced in Delay Tolerant Networking Research

- NO documentation at all. Group.

- Supported DTN routing protocols

  - Epidemic routing (MEED)
  - Unicast routing (ED)

- Supported network model

  - ARDEN uses Haggle and RollerNet dataset

3.3 The ONE simulator

- `http://www.netlab.tkk.fi/tutkimus/dtn/theone/`

- The latest version has been released in 2011.

- Supported DTN routing protocols

  - Several epidemic routing protocols
  - Spray and Focus (variant of spray and wait)
  - Prophet

- Supported network model

  - Random walk
  - Helsinki city center trace
  - Working Day Movement model