Anonymous DTN routing

7/24/2013

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 receiver is not revealed to any other nodes whom the sender or receiver 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 receiver, other untrusted nodes cannot learn the permanent IDs of sender and receiver 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 and receiver identity anonymity
   - Passive attackers cannot learn the permanent ID of the sender or receiver by eavesdropping packets.
   - Active attackers can learn nothing more than ephemeral IDs of sender and receiver by compromising a subset of untrusted nodes.
2. Unlinkability

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

3. Location anonymity
   - Permanent ID of a node should not be revealed to attackers.
   - Passive or active attackers cannot link observed ephemeral address to the corresponding permanent address.
4. 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 different group of trusted nodes and knows 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.

1. A node has symmetric keys of other nodes.
2. A node knows random seeds and ephemeral IDs of its trusted nodes.
3. Start time and end time of each epoch is identical in all nodes.
4. Mutual trust: If Alice trusts Bob, Bob also trusts Alice.

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, receiver identity anonymity through the use of ephemeral ID. Each node n in DTN generates and updates its own ephemeral ID, and other nodes that n trusts should be able to generate the current ephemeral ID of n at any time. Since the nodes that n trusts are given the permanent ID of n, they can map the ephemeral ID of n to its permanent ID at any time. However, attackers or nodes that n does not trust cannot learn the permanent ID of n. Also the attackers may receive the current ephemeral ID of n from n or other nodes, but they cannot generate ephemeral ID of n on their own. Therefore, even though the attackers observe or receive packets sent by n, they cannot determine the permanent ID of n. Moreover, the attackers recognize a stream of packets from a single node 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 receiver is completed within a single epoch, that is, ephemeral IDs of the sender and receiver are not changed during the delivery. Even in this case, attackers are only able to learn ephemeral IDs of the sender and receiver 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 receiver 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.
- $\{m\}_k$, $\{c\}_{k^{-1}}$: Public key encryption/decryption of message m.
- n_i: A node in DTN. $n_i \in N$.
- id_i, eid_i: Permanent ID and ephemeral ID of node n_i.
- r_i: A random seed generated by n_i.
- R_i: A set of random seeds of nodes in G_i.
- G_i: A set of IDs of trusted nodes of n_i.
- E_i: A set of ephemeral IDs of nodes in G_i. It may include ephemeral IDs in used last m-previous epochs.
- H_i: A set of ephemeral IDs of contact history (neighbor nodes) of n_i. It may include ephemeral IDs in used last m-previous epochs.
- BF_{E_i}: Bloom filter encoding of ephemeral IDs of nodes in G_i.

2.2 Network initialization

Inputs: A node n_i has G_i and R_i.

Outputs: A node n_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 n_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:    H_i ← NULL
12:    return
13: end procedure
```

When a node n_i enters a DTN network, it is assumed that n_i is given id_i, r_i, G_i and R_i. Then n_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 n_i executes `Initialize()` to generate new parameters eid_i and E_i. eid_i, an ephemeral ID of n_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 n_i updates E_i by generating new ephemeral IDs of nodes in G_i. (Lines 8-10) At the end of each epoch, a node n_i resets H_i since it cannot generate new ephemeral IDs of untrusted nodes. (Line 11)

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.

2.3 Beacon construction

Inputs: A node n_i has E_i, a set of ephemeral IDs of the nodes in G_i, and H_i, a neighbor node list of n_i.
Outputs: A node n_i outputs a beacon message $beacon_i$.

```
1: procedure GenBeacon(id_i, E_i, H_i)
2:     beacon_i ← {eid_i ‖ BF_{E_i} ‖ BF_{H_i}}
3:     return beacon_i
4: end procedure
```

After network initialization, n_i advertises its presence to other nodes by generating a beacon message. The beacon message contains ephemeral ID of n_i, bloom filter encoding of E_i, and bloom

filter encoding of H_i. BF_{E_i} contains current ephemeral IDs of other nodes n_i trusts, and BF_{H_i} contains current ephemeral IDs of n_i's neighbor nodes that n_i does not trust. Ephemeral IDs of other nodes would be obtained through beacon messages from other nodes.

Beacon message of n_i should not reveal more than a current ephemeral ID of n_i. An attacker may try to learn ephemeral IDs contained in BF_{E_i} or BF_{H_i}, possibly through brute force attack. However, an attacker cannot assure whether a certain ephemeral ID is in BF_{E_i} because of the possibility of false positive match. Moreover, ephemeral IDs, BF_{E_i} and BF_{H_i} of nodes are changed at the end of an epoch that occurs to all nodes almost at the same time, an attacker cannot track a specific node on expiry of the current epoch.

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 should put certain offset to vary its beacon period.

2.4 Packet construction

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

Outputs: n_s outputs a packet *packet*.

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

A sender node n_s constructs a packet by first encrypting the message using a symmetric key k_{sd} shared between the sender and receiver of the packet. Then the sender generate *packet* which contains the encrypted message along with the ephemeral ID of the sender and receiver.

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 source 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 k_{sd} cannot learn the message m. The only things that a relay node can learn are ephemeral IDs of the source and destination nodes. 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 n_i has a packet *packet* to forward. n_i and n_j are on contact. n_i has $beacon_j$, a beacon message of n_j, and a symmetric key k_{ij}.

Outputs: n_i either transmits an encrypted packet $packet'$ to n_j or remove $packet$ from the queue.

procedure ForwardPacket($packet, beacon_j$)
  if $(trusted\_tx\_count[packet] + untrusted\_tx\_count[packet]) \geq TH_{total}$ **then**
    RemoveFromQueue($packet$)
    return
  end if

  if $epoch\_expired$ **then**
    if $packet.eid_s \in E_i$ **then** ▷ n_i is trusted by n_s ($n_s \in G_i \leftrightarrow n_i \in G_s$).
      $id_s \leftarrow$ MatchID($packet.eid_s$)
      $packet.eid_s \leftarrow$ GenEphemeralID(id_s, r_s, t)
    end if
    if $packet.eid_d \in E_i$ **then** ▷ n_i is trusted by n_d ($n_d \in G_i \leftrightarrow n_i \in G_d$).
      $id_d \leftarrow$ MatchID($packet.eid_d$)
      $packet.eid_d \leftarrow$ GenEphemeralID(id_d, r_d, t)
    end if
  end if

  if $packet.eid_d = beacon_j.eid_j$ **then** ▷ n_j is the destination of $packet$.
    $packet' \leftarrow \{packet\}_{k_{ij}}$
    SendPacket($packet, beacon_j.eid_j$)
    RemoveFromQueue($packet$)
    return
  else if $packet.eid_s \in beacon_j.BF_{E_j}$ **then** ▷ n_j is trusted by n_s($n_s \in G_j \leftrightarrow n_j \in G_s$).
    $packet' \leftarrow \{packet\}_{k_{ij}}$
    $trusted\_tx\_count[packet] + +$
    SendPacket($packet, beacon_j.eid_j$)
    return
  else if $untrusted\_tx\_count[packet] < TH_{untrusted}$ **then**
    if $packet.eid_d \in beacon_j.BF_{H_j}$ **then** ▷ n_j is not trusted by n_s, but n_d is in the neighbor list of n_j
      $packet' \leftarrow \{packet\}_{k_{ij}}$
      $untrusted\_tx\_count[packet] + +$
      SendPacket($packet, beacon_j.eid_j$)
      return
    end if
  end if
  return
end procedure

procedure ValidateBeacon($beacon_j$)
  if $eid_i \in beacon_j.BF_{E_j}$ AND $beacon_j.eid_j \notin E_i$ **then** ▷ n_i does not trust n_j but eid_i is

```
    included in BF_{E_j}.
41:         return(False)
42:     else
43:         return(True)
44:     end if
45: end procedure
```

A node n_i has a packet to relay, or has generated a packet and wants to send to destination node. Then whenever n_i contacts n_j, that is, n_i receives a beacon message from n_j, n_i performs a procedure `ForwardPacket()` to forward the packet.

We put a certain threshold for a packet replication to avoid pure epidemic routing. (Lines 2-5) Then n_i checks if the current epoch has been expired and if it has, n_i tries to updates ephemeral IDs of sender and destination contained in the packet. n_i is only able to update eid_s (or eid_d) only if n_i is trusted by n_s (or n_d). (Lines 7-16) If n_j is the destination node of the packet, n_i forwards the packet and removes the packet from its queue. However, n_j may not remove the packet from the queue for receiver anonymity from eavesdroppers. (Lines 18-22) If n_j is not the destination but trusted by n_s, then n_j forwards the packet to n_j and keeps the packet for further replication. (Lines 23-27) If n_j is not the destination nor a node trusted by n_s, n_j checks if n_j has an ephemeral ID of the destination eid_d as its neighbor node in BF_{H_j}. n_j forwards the packet to n_j only if eid_d is in BF_{H_j} to prevent possible bandwidth waste. (Lines 28-35)

The forwarding protocol should work properly even if a few untrusted nodes are compromised and try to receive packets from packets from a relay node, possibly by using a fabricated beacon message. A compromised node n_j may try to fabricate its E_j so that a relay node n_i considers n_j as a node trusted by the source node n_s. An example of the attack would be setting E_j to all 1s so that the compromised node n_j can be considered to be trusted by any other nodes. This attack would waste replication threshold TH_{total} of n_i and may deter packet forwarding to a proper route. To prevent this attack, the relay node n_i may confirm if its own ephemeral ID eid_i is included in E_j. If n_j is not a trusted node of the relay node but eid_i is included in E_j, n_i may refuse to forward the packet to n_j. (Lines 39-45)

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