Onion Routing in Predictable Delay Tolerant Networks

Adrian Antunez-Veas and Guillermo Navarro-Arribas

Department of Information and Communications Engineering,
Universitat Autònoma de Barcelona (UAB), 08193 Cerdanyola, Spain
{aantunez, gnavarro}@deic.uab.cat

Abstract Abstract goes here

Keywords: Onion Routing, Delay-tolerant Network, Anonymous networking, Privacy, Security

1 Introduction

To be done

2 Related work

Aim of this section: Show the existing research about this topic. Identify his flaws and explain briefly what we do to solve (or improve) them.

3 Proposal

In this section, we review some relevant concepts like predictable DTNs, onion routing and key management. Later, we define a method to put together all contacts information. Finally, we propose an algorithm for onion routing path choosing process using the prior knowledge of the network.

3.1 Relevant concepts and assumptions

We define as predictable DTNs at such networks where the behaviour is known in advance or where a repetitive action occurs over the time.

These networks fits perfectly with the concept of oracle schemes. Oracle schemes are a subset of DTN routing protocols. In this schemes there is a set of nodes that has nearly full knowledge of the network and its planned evolution [1]. Oracle schemes are used in deterministic scenarios where the oracle node will have full details of occurred contacts as well as the future ones, among other things.

For instance, we could consider public transport networks as predictable (deterministic) networks because every node performs the same route periodically so this route can be known in advance. This information could be shared among all nodes in the network in order to let them be "oracle", i.e: let them have a full knowledge of the behaviour of the whole network. Is important to note that despite these networks are nearly deterministic, an error needs to be assumed for exceptional situations.

As we explained previously, all network information will be known even by active adversaries, therefore a security analysis needs to be carried out. This analysis is discussed in section 4.

3.2 Onion routing overview

First, we provide a high-level description of how onion routing works as is the essence of this proposal. The source node, wishing to send an anonymous message to another node, chooses a path and ciphers the message several times making "layers". Each layer of encryption uses a different pre-shared key between the source and a node on the path. When the message reaches a node, the layer is removed with the private key of the node and the packet gets modified. At the end of the process the destination will receive the fully decrypted message without knowing who sent it.

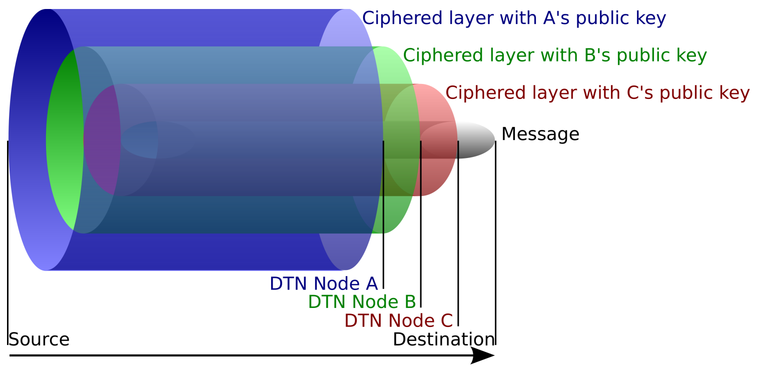

Ciphered layer with A's public key
Ciphered layer with B's public key
Ciphered layer with C's public key
Message
DTN Node A
DTN Node B
DTN Node C
Source
Destination

Figure 1: Onion routing graphical example of how layering works.

An example can be seen in figure 1 where the source node wants to send an anonymous message to a destination, using a previously chosen path composed by the nodes A, B and C. The source ciphers the message with the public key of C, B and A respectively. When the message reaches every hop, each node removes the external layer using the node private key.

In order to use onion routing a prior key exchange process needs to be performed. We consider that the loading of the network information behaviour as well as the keys sharing process will be done simultaneously at a previous stage, i.e: before starting the periodical routing. For example, in public transport networks the data as well as the needed keys will be shared before the scheduled route starts.

3.3 Contact data representation

Is it clear that we need to use a structure to represent the network activity. One method that seems to work is to use a graph structure as a way to represent network activity [?] [?] [?]. Where each node of the network is represented as graph vertices and each contact between nodes as graph edges.

A graph $G = (V, E)$ consists of two sets V and E [2]. The elements of V are called *vertices* or *nodes* while the elements of E are called *edges*. Each edge is composed by two vertices. Edges and vertices can have *attributes*. An attribute is a value associated either to edges and vertices that permits to store information.

There is a subset of graphs in the graph theory called *dynamic graphs*. Dynamic graphs permit to represent the evolution of the network during the time easily, i.e: nodes and edges may change positions and appear and/or disappear through time. To do so, we can set a pair of attributes that point out when a node appears and disappears.

Dynamic graph provides a good way to store information about the network. In our case this information will be related to contacts like the instant of time when the connection opportunity occurs and how long this contact has been.

In figure 2 the evolution of a given graph representing different contacts between buses in a public transport network is shown. In this example we can see that at the beginning, no contacts have been produced yet. As the time passes, the contacts (*edges*) appear and disappear between buses (*nodes*).

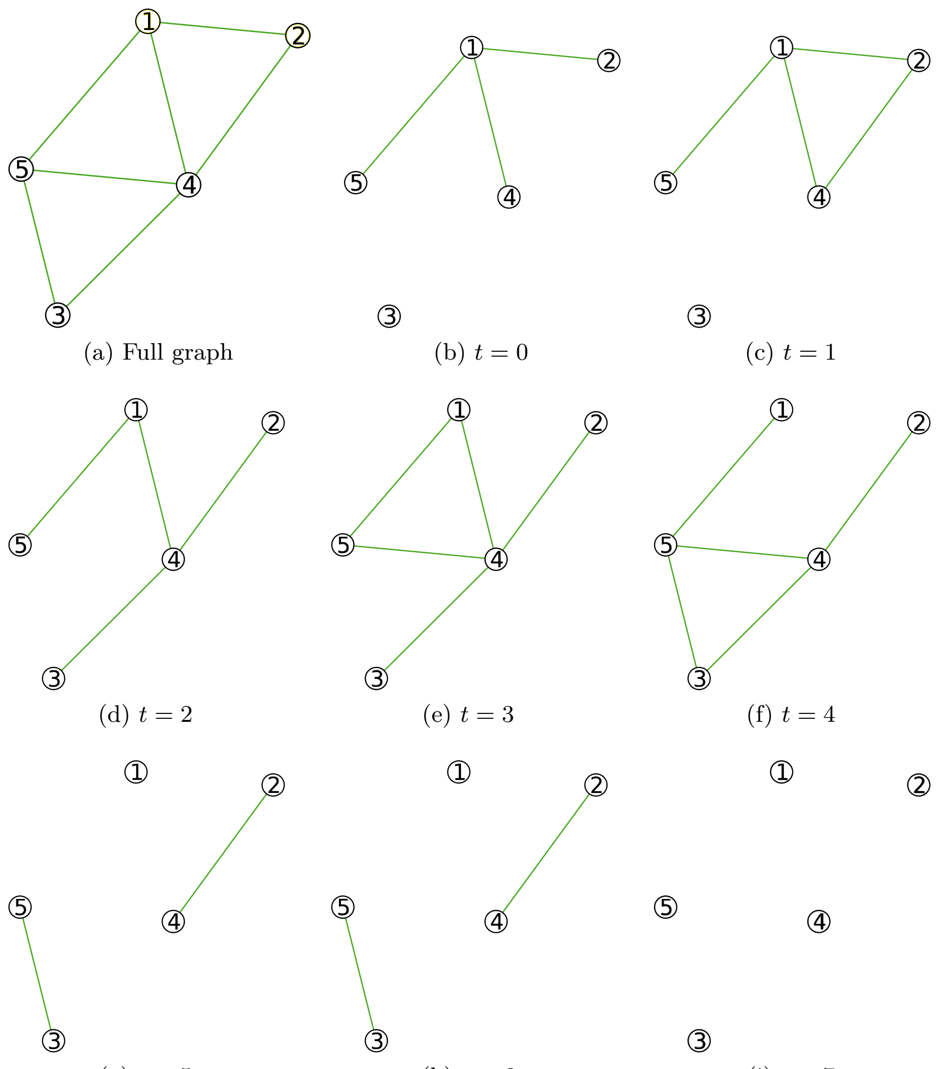
3.4 Path choosing

Imagine we have a source node s willing to send a message to a destination d at time t using onion routing. In onion routing you need to choose the number of nodes n where the message will pass through. Node s obtains a path to perform the layering process and send the message. The time required to exchange the message between nodes is defined as tt.

To make even harder the path guessing for thirds, as the network knowledge is shared among all nodes, the node s obtains a set of paths k in order to choose one randomly. To sum up, we need a method *f(s, d, t, n, k, tt)* that will retrieve a set of paths to finally choose one following the t, n, k and tt requirements.

The procedures shown in Algorithm 1 and Algorithm 2 are an example of deterministic choosing for onion routing. Both of them inherit values to filter out unneeded paths. Specifically, in Algorithm 1 we return from a given node *host* at time *currentTime* a set of neighbors that are available or will be available to forward the message. In Algorithm 2 we performed a recursive search to get up to k paths of length n from *source* to *destination*.

In figure 2 we showed an example of contact data representation using dynamic graphs. Applying our path choosing method with: source node *s=1*, destination



(a) Full graph (b) $t = 0$ (c) $t = 1$

(d) $t = 2$ (e) $t = 3$ (f) $t = 4$

(g) $t = 5$ (h) $t = 6$ (i) $t = 7$

Figure 2: In (a) the full (static) graph can be seen. In $(b)-(h)$ different snapshots in given instants of time showing the evolution of the dynamic graph are depicted.

node *d=5*, starting time *t=0*, number of paths *k=4*, number of nodes of each path of *n=5* and transmission time *tt=1*, we get the following paths:

Path: 0
(1:0) → (2:0) → (4:1) → (1:2) → (5:3). Arrival time: 4
Path: 1
(1:0) → (2:0) → (4:1) → (3:2) → (5:4). Arrival time: 5
Path: 2
(1:0) → (4:0) → (1:1) → (4:2) → (5:3). Arrival time: 4
Path: 3
(1:0) → (4:0) → (2:1) → (4:2) → (5:3). Arrival time: 4

(n: t): Means message sent to node n at time t.

So we got 4 different paths composed by 5 nodes each (3 layers will be done). From the given set of paths, the source node s will need to choose one in order to perform the onion routing itself.

Algorithm 1 Procedure to get valid neighbors of a given node

INHERIT: (1) *source*, source node. (2) *destination*, destination node. (3) t, when the message will be sent. (4) k, maximum number of paths. (5) n, number of nodes for each path (including source and destination). (6) tt, transmission time required to forward the message.
INPUT: (1) *host*, host node. (2) *currentTime*, current time.
OUTPUT: (1) *validNeighbors*, a set of valid neighbors to whom forward the message.

```
1: procedure GetAvailableNeighbors (host, currentTime)
2:     for each $nbr \in host.neighbors()$ do
3:         if $(nbr.activationTime + nbr.contactDuration - tt) \geq currentTime$ then
4:             $validNeighbors.add(nbr)$
5:     return $validNeighbors$
```

4 Security analysis

Aim of this section: Brief introduction of what are we going to talk about in this section.

4.1 Threat model

Aim of this section: Define against who we try to defend our data. What the attacker may or may not do. Define some considerations such as sufficient strong cryptographic algorithms with sufficiently long keys to prevent practical cryptanalysis attacks.

4.2 Passive adversaries

Aim of this section: Needs to be defined. I am not even sure if this kind of adversaries will be in our proposed model.

Algorithm 2 Procedure to get paths that follows the inherited requirements

```
INHERIT: (1) source, source node. (2) destination, destination node. (3) t, when the
    message will be sent. (4) k, maximum number of paths. (5) n, number of nodes for
    each path (including source and destination). (6) tt, transmission time required to
    forward the message.
INPUT: (1) currentPath, current path. (2) currentTime, current time.
OUTPUT: (1) paths, a set of paths that follows the inherited requirements.
 1: procedure GetPaths(currentPath, currentTime)
 2:     if $currentPath.size() \neq n$ and $paths.size() < k$ then
 3:         $node \leftarrow currentPath.last()$
 4:         $validNeighbors \leftarrow GetAvailableNeighbors(node,\ currentTime)$
 5:         for each $nbr \in validNeighbors$ do
 6:             $oldPath \leftarrow currentPath$
 7:             $nbr.sendingTime \leftarrow \max(nbr.activationTime, currentTime) + tt$
 8:             $currentPath \leftarrow nbr$
 9:             if $nbr = destination$ and $currentPath.size() = n$ and $currentPath \notin paths$
    then
10:                 $validNeighbors.add(nbr)$
11:             else
12:                 GetPaths($currentPath, nbr.sendingTime$)          ▷ Recursive part
13:             $currentPath \leftarrow oldPath$
14:     return $paths$
```

4.3 Active adversaries

Aim of this section: Needs to be defined. I am not even sure if this kind of adversaries will be in our proposed model.

5 Evaluation

In this section the security of our proposal is evaluated using a realistic scenario. We decided to recreate the small public transport network of the UAB campus using the Network Simulator 3 (NS-3). NS-3 is a well-known discrete-event simulator targeted primarily for the research usage [4]. We explain as well how the mobility model was obtained as well as what parameters were used during the simulation. Finally we will test our proposed method in the simulation to evaluate how it performs.

5.1 Scenario: Campus buses

In order to test our proposal we considered a very small public transportation network that works inside the Autonomous University of Barcelona (UAB) composed by 5 buses that makes different routes around the UAB campus. Is important to note that every single bus makes the same route daily. By this way, this example can be seen as a good example of deterministic networks.

Each bus has a DTN node that allows to achieve secret communications as well as source anonymity using onion routing protocol. There are several applications that can take profit of such networks like anonymous reporting systems.

5.2 Mobility Model

We obtained the mobility model going through different stages. First, we exported the UAB Campus map from OpenStreetMaps into SUMO software [3], filtering some unnecessary items like buildings and railways with the Java OpenStreetMap editor tool [5].

Once the campus roads were imported in SUMO, we recreated the bus movements of each bus taking into consideration the official bus schedule of the UAB public transportation network [7]. In addition, we tuned some bus characteristics like acceleration and deceleration parameters in order to get coherent travel times.

Finally, we exported the model to a NS-2 mobility trace as is explained in [6]. The NS-2 mobility trace can be used in NS-3. We used the simulator to obtain important contact related data of the campus network, i.e: information about the duration of the contacts as well as the instant of time when they occurred.

5.3 Simulation setup

The DTN modules are under current development in NS-3 [8]. For this reason we decided to implement a neighbour discovery in the application layer. This application broadcasts beacons messages periodically looking for new contact opportunities. The interval time is the time to wait between beacons while the expiration time is the time a neighbour is considered valid, these parameters can be set up manually.

Parameter	Value
Number of nodes	5
Wi-Fi range	100 meters
Interval time	1 second
Expiration time	2 seconds
Simulation time	15 hours
DSS Rate	1 Mbps
IEEE 802.11 specification	802.11b

Table 1: Simulation setup.

As can be seen in table 1, we also tuned different wireless parameters to be suitable with resource-constrained computers like those that could be used in each bus.

5.4 Simulation results

In figure 3 we have an overall view of the network activity. There is a group that have really short communications, these communications can be suitable for those applications that does not require a huge amount of data to send. There is another group that has long communications (nearly 7 minutes), these group is able to perform complex communications sending higher amount of data. The average contact time is near 1 minute indeed is suitable for several applications like anonymous reporting systems. A summary of contacts related information during the simulation is shown in table 2.

Metric	Value
Number of contacts	1161
Average contact time	72.72 seconds
Maximum contact time	412 seconds
Minimum contact time	1 second

Table 2: Contact information.

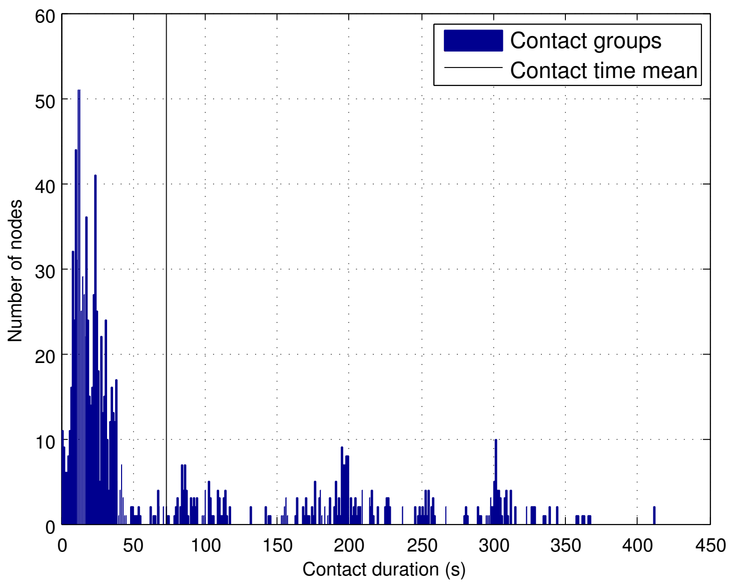


Figure 3: Number of contacts with the same duration.

Create a ratio between time and probability to guess the source. Normalize the value between 0 and 1. Do a 3-D graphic modifying k and n. Normalize like https://docs.tibco.com/pub/spotfire/5.5.0-march-2013/UsersGuide/norm/norm_scale_between_0_and_1.htm

6 Conclusions and future work

Aim of this section: There are two parts:
The first one the conclusions of this project. What we can extract from our research. Could be useful? Important things that needs to be remarked, etc.
The second one the future work. What needs to be done in the future to solve some undesired behaviours, explore unresearched lines of this work, etc.

[Conclusions part]

[Future work]

Search and analyse efficient ways of path choosing, being able to use the algorithm in resource-constrained computers.

In the network could exist black holes, i.e: nodes that drop all the traffic. In order to mitigate the impact of such nodes in the whole network a reputation system could be used. By this way the reputation value will be shared among all the nodes in the network. The path choosing algorithm should be modified too to take this value into consideration in the choosing process. Finally this will lead into another security analysis due to the more reputation a node has, most probable is to be one of the chosen.

The simulation model could be adapted to consider traffic modifications, generating the enough information to decrease the number of failed contacts due to a bad contact prediction.

Acknowledgements Acknowledgements go here.

References

1. Farrell, S., Cahill, V.: Delay- and Disruption-tolerant Networking. Artech House telecommunications library, Artech House (2006)
2. Gross, J., Yellen, J., Zhang, P.: Handbook of Graph Theory, Second Edition. Discrete Mathematics and Its Applications, Taylor & Francis (2013)
3. Institute of Transportation Systems: SUMO - Simulation of Urban MObility, http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000
4. Nsnam: NS-3 webpage, https://www.nsnam.org
5. Open Street Map: JOSM, https://josm.openstreetmap.de
6. Raj, C., Upadhayaya, U., Makwana, T., Mahida, P.: Simulation of vanet using ns-3 and sumo. International Journal of Advanced Research in Computer Science and Software Engineering 4, 563–569 (2014)
7. UAB: Horaris d'autobus 2015, http://www.uab.cat/doc/horaris_busUAB_2015
8. Zhou, D.: DTN Bundle Protocol, https://www.nsnam.org/wiki/Current_Development

All links were last followed on June 19, 2015.