Noname manuscript No.
(will be inserted by the editor)

Message Anonymity on Predictable Opportunistic Networks

D. Chen · G. Navarro-Arribas · C. Pérez-Solà · J. Borrell

Received: date / Accepted: date

Abstract A Predictable Opportunistic Network (POppNet) is a network where end-to-end connectivity is not guaranteed, and node communication happens in an opportunistic manner, but the behavior of the network can be predicted in advance. The predictability of such networks can be exploited to simplify some mechanisms of more generic OppNets where there is no prior knowledge on the network behavior. In this paper, we propose some solutions to provide anonymity for messages on POppNets by using simple onion routing, and thus to increase the privacy of the nodes in communication.

Keywords Opportunistic networking, Predictable opportunistic networks, Onion routing, Message anonymity.

1 Introduction

Opportunistic Networks (OppNets) are networks where communication happens opportunistically between nodes, end-to-end connectivity is not guaranteed and disruptions and delays are to be expected (Mota et al., 2014). They are also denoted as Delay and Disruption Tolerant Networks (DTNs), although the term DTNs usually describes OppNets which use the specific Bundle protocol (Scott and Burleigh, 2007).

D. Chen, G. Navarro-Arribas, J. Borrell
Dep. of Information and Communications Engineering,
Universitat Autonoma de Barcelona
CYBERCAT-Center for Cybersecurity Research of Catalonia
E-mail: {depeng.chen, guillermo.navarro, joan.borrell}@uab.cat

C. Pérez-Solà
Internet Interdisciplinary Institute (IN3),
Universitat Oberta de Catalunya (UOC),
CYBERCAT-Center for Cybersecurity Research of Catalonia
E-mail: cperezsola@uoc.edu

Among OppNets there are those where the contacts between the nodes follow an specific pattern (Jain et al., 2004). In such cases the behavior of the network nodes can be predicted to some extend, and thus the communication and interactions between them can be known in advance. Such networks are usually denoted as Predictable Opportunistic Networks (POppNets). This predictability can be exploited to improve routing in the network for instance. Predictability here, refers to the fact that the connectivity, the topology of the network, and its evolution over time, can be predicted ahead of time.

We focus our work in the specific POppNet that raises from a network build on public transportation systems. Consider that all public buses in a city carry a simple network node allowing them to opportunistically exchange messages. Given the routes and timetables of the buses one can predict when interactions between nodes will occur during the day, and thus these networks can be used as a low cost urban networks. Routing can be more efficiently solved in POppNets than in generic OppNets due to their predictability, and we believe that some security services can also be improved. More precisely, anonymous routing is a difficult and complex problem in OppNets. Current solutions to provide anonymous routing in OppNets require complex cryptographic solutions, and complex setups. This is due to the fact that traditional solutions such as onion routing (Goldschlag et al., 1996) cannot be directly applied in such networks. We will show however that the predictability in these networks can be exploited to actually use a simplified onion routing approach to provide anonymous routing for messages in POppNets. This idea was first sketched in (Antunez-Veas and Navarro-Arribas, 2016; Chen et al., 2017).

The aim of our proposal is to support applications where one end needs to send anonymous short messages in one direction. We assume that we can directly use the nodes public keys to perform a simple onion routing since establishing a session key incurs in more penalty than gain. Although these might seem strong assumptions, they are commonly used in OppNet environments and protocols such as Bundle protocol (Scott and Burleigh, 2007). The expected delivery time is also relatively large, which allows for more randomness in the path selection. Note that in traditional onion routing, nodes on the onion path are randomly selected (or selected based on some random parameter). Such randomness is important because it ensures that an attacker will not be able to easily guess the path of a given message.

The contribution of this paper is precisely the study of the application of onion routing in POppNets to achieve message anonymity. We focus our experimentation in a specific network of public transportation from the city of Seattle, and conduct a thorough analysis based on different configurations of this network with assorted network density. As we will show, the density of the network is a key issue for our proposal. We provide methods to determine the onion routing path, and introduce measures to study and quantify the anonymity that can be achieved when using such paths, with the final goal of increasing the privacy of the nodes in communication.

The paper is organized as follows. Section 2 introduces notations and the model for OppNets. In Sect. 3 we introduce the algorithms to determine the

path and Sect. 4 defines the anonymity measures. Section 5 shows the experiments conducted in the Seattle bus POppNet network both in terms of performance and anonymity. Finally, Sect. 6 describes the related work and Sect. 7 concludes the paper.

2 A model for Opportunistic Networks

An OppNet can be modeled as a dynamic graph. That is, a graph with dynamic components, which in our case are the presence of edges and nodes. Vertices represent network nodes and edges represent the fact that there is a connection between two nodes. These graphs are also denoted in the literature as temporal networks (Holme and Saramki, 2012; Pan and Saramki, 2011), time-varying graphs (Casteigts et al., 2012), temporal graphs (Kostakos, 2009), or evolving graphs (Xuan et al., 2003). In our case, each edge has a temporal presence based on the time that a connection can be established between the two nodes, usually based on the coverage of the network nodes.

For our scenario we consider the dynamic graph to be undirected because a connection between two nodes represents the fact that these two nodes can communicate in both directions. We denote such an undirected dynamic graph as $G(V, E)$, where V is the set of nodes, and E is the set of edges. Each edge, denoted as $e = (u, v, t, \lambda)$, is a *temporal edge* between nodes u and v, starting at time t, with a duration of λ. Note that $(u, v, t, \lambda) = (v, u, t, \lambda)$. For simplicity we will use $\lambda(e)$ as the duration of the link in edge e, and $t(e)$ as the starting time for edge e, or simply λ and t if the edge can be easily inferred from the context. We will not consider the timed presence of nodes. Note that a node presence can be modeled by eliminating all its edges during the time when the node disappears.

We also consider a unique transmission time τ for all messages to be sent in the network. This can easily be extended to a variable transmission time specific for each message, or edge. The transmission time includes the time required to establish the connection, send, and receive the message. We also assume that for all edge $e \in E$ for a given graph $G(V, E)$, $\lambda(e) \geq \tau$. That is, all edges in the graph can be used to send a message. Edges that do not address this constraint can be removed from the graph in a pre-processing step.

The graph $G^*(V^*, E^*)$ is the static undirected graph obtained from G by considering all its edges without time constraints (all edges are in the graph independently of time). We use $N(v)$ to denote all neighbors of node v in G^*. That is, $N(v) = \{u \mid (v, u, t_i, \lambda_i) \in E^*, \forall i\}$.

We provide some definitions here that will be used in the paper.

Definition 1 The *time forward neighbors* or future neighbors of a given node $u \in V$ at time t_c for the dynamic graph $G(V, E)$ are defined as: $N^{t+}(u, t_c) = \{v \mid (u, v, t, \lambda) \in E, t_c \leq t + \lambda - \tau\}$.

Similarly,

Definition 2 The *time backward neighbors* or past neighbors of $v \in V$ at time t_c are defined as: $N^{t-}(v, t_c) = \{u \mid (u, v, t, \lambda) \in E, t + \tau \leq t_c\}$.

In general, $N^{t+}(u, t_i) \cup N^{t-}(u, t_i) = N(u)$. From a practical point of view, we usually want to consider time forward and backward neighbors up to a maximum delay or carry time. If a node wants to deliver a message to another node, the first one can carry the message up to a maximum time. In such case, for simplicity we define a unique maximum carry time for all nodes denoted as δ_M. The extension to a variable carry time depending on the node is straightforward.

Definition 3 The *time forward neighbors with carry* of node $u \in V$ at time t_c considering a maximum delay or carry time δ_M for the dynamic graph $G(V, E)$ are the nodes: $N_{\mathcal{C}}^{t+}(u, t_c, \delta_M) = \{v \mid (u, v, t, \lambda) \in E, t - \delta_M \leq t_c \leq t + \lambda - \tau\}$.

Definition 4 The *time backward neighbors with carry* of node $u \in V$ at time t_c considering a maximum delay or carry time δ_M for the dynamic graph $G(V, E)$ are the nodes: $N_{\mathcal{C}}^{t-}(u, t_c, \delta_M) = \{v \mid (u, v, t, \lambda) \in E, t + \tau \leq t_c \leq t + \lambda + \tau + \delta_M\}$.

We also consider the degree of a node at a given time. That is, the degree of a node will be time dependent. In our case we are interested in looking for the degree associated to time forward neighbors with carry, and time backward neighbors with carry. We thus define the time forward degree and time backward degree of a node as follows.

Definition 5 The *time forward degree* of a node u at time t_c with a maximum delay δ_M is defined as: $deg_{\mathcal{C}}^{t+}(u, t_c, \delta_M) = \mid N_{\mathcal{C}}^{t+}(u, t_c, \delta_M) \mid$, and the *time backward degree* is defined as $deg_{\mathcal{C}}^{t-}(u, t_c, \delta_M) = \mid N_{\mathcal{C}}^{t-}(u, t_c, \delta_M) \mid$

$P = \langle (v_1, t_1), (v_2, t_2), \ldots, (v_l, t_l) \rangle$ denotes a *path* in the dynamic graph $G(V, E)$, where (v_i, t_i) represents the node in the path and the time that the message arrives to such node. Given the path P the *length* of the path is the number of nodes included in path. The *duration* of a path P is the time taken by the message to arrive to destination. That is, for $P = \langle (v_1, t_1), \ldots, (v_l, t_l) \rangle$, $duration(P) = t_l - t_1$.

3 Paths for onion routing in POppNets

Routing in OppNets is a difficult problem due to the unforeseeable nature of contacts between nodes, which leads most of the times to the adoption of (limited) broadcast approaches for routing (Borrego et al., 2019; Sharma et al., 2019; Borah et al., 2018). However, the fact that we are dealing with predictable opportunistic networks (POppNets) greatly simplifies such routing. Routing can be seen as establishing paths in the time-based dynamic graph that represents the network (Jain et al., 2004). Solutions exists to determine shortest paths in such networks. This is however not a good solution from a

privacy perspective if we consider an onion routing strategy. In this case, the predictability of the chosen path is a key issue that needs to be protected. In this section we introduce two stochastic strategies to determine paths in dynamic graphs, and thus, opportunistic networks. The main objective of these approaches is to determine a path to be used for onion routing in such networks. The stochastic nature of both algorithms has the goal of keeping the predictability of the path as low as possible.

3.1 Random Path Finder

The first strategy is to use a time-based random walk on the graph ensuring that the predictability of the path is very low. This will be somehow similar to randomly choosing the onion routing nodes in Tor.

Algorithm 1: Random path finder.

Data: $G(V, E)$ is a dynamic graph; $s \in V$ is the source node; $d \in V$ is the destination node; t_s is the starting time; t_c is the current time; L_{min} is the minimum path length; L_{max} is the maximum path length; τ is the transmission time;
Result: Path P from s to d, such that $L_{min} \leq length(P) \leq L_{max}$.

```
1  begin
2      P = ⟨(s, t_s)⟩;
3      t_c = t_s;
4      continue = True;
5      while continue do
6          u = getLastNode(P);
7          select a random edge e = (u, v, t, λ) such that e ∈ N^{t+}(u, t_c);
8          if t_c > t(e) then
9              t_c = t_c + τ;
10         else
11             t_c = t(e) + τ;
12         P = P + ⟨(v, t_c)⟩;
13         if v is repeated in P then
14             P = mergeNodePath(P, v);
15         if v = d and L_min < length(P) < L_max then
16             continue = False;
17         else if N^{t+}(u, t_c) = ∅ or length(P) > L_max then
18             P = ⟨(s, t_s)⟩;
19             t_c = t_s;
20     return P;
```

This method, shown in Algorithm 1, allows inclusion of repeated nodes but performs a shrink or merge of the path if a repeated node is found (`mergeNodePath(`P, v`)`), eliminating the intermediate nodes. This intermediate nodes do not provide additional security in the path given that the repeated node could correlate traffic received twice.

For example, in path $P = \langle (v_0, t_0), (v_1, t_1), (v_2, t_2), (v_3, t_3), (v_4, t_4), (v_5, t_5) \rangle$, if we know that node v_2 is the same as node v_4, the resulting merged path will be: $P = \langle (v_0, t_0), (v_1, t_1), (v_2, t_2), (v_5, t_5) \rangle$. That is, node v_2 carries the message until time t_4 to send it then to node v_5. We have found this approach to be faster than just backtracking or re-starting the path search if a repeated node appears.

For practical reasons we bound the search to prevent very long paths (both in time and number of nodes). This bound can be set based on different parameters: maximum path length, duration, computation time, ... In our case, we have set the bound to a maximum path length (L_{max}) and a maximum number of executions of the main loop (set to $10M$ in the experiments of the evaluation). As we will see in Section 5 the algorithm produces paths with reasonable length and duration.

3.2 Forward-Backward Path Finder

In order to improve the random walk algorithm, we have designed another stochastic approach using a meet-in-the-middle strategy. The idea is to use information from both source and destination nodes to perform a stochastic path search from both nodes at the same time. As we will see, this approach is more likely to find the path in case it exists and can produce shorter paths. This algorithm is denoted as the Forward-Backward search (FB), and shown in Algorithm 2.

The algorithm performs a partial path search starting from the source node, where a path will be randomly selected from potential forward neighbors. At the same time the analogous procedure is performed from the destination node selecting backward neighbors. Actually, to increase uncertainty, the algorithm keeps track of all potential forward and backward partial paths to finally randomly select an intersection, yielding the final path. In order to speed up the process, both forward and backward searches are done at the same time, so each step of the algorithm increases the path length by two new nodes.

In Algorithm 2, the partial paths are kept in an array-like structure, $P_f[i]$ and $P_b[i]$, for forward and backward partial paths respectively, where i is the number of nodes included in the partial path. This is shown to ease the understanding of the algorithm, but from an implementation perspective, these partial paths have to be stored in a more efficient data structure such as a tree. A partial forward path tree keeps the source node for the forward paths an all possible paths as successive branches. Each node stores the earliest time the node is visited. The same is analogously done for the partial backward nodes.

The intersection between forward and backward partial paths can be done looking for an intersecting node (yielding an odd path length) or edge (yielding an even path length).

Given a partial path $P' = \langle (v_1, t_1), \ldots, (v_l, t_l) \rangle$ of length l, we use $last(P') = (v_l, t_l)$ to denote the last node and associated time, and $first(P') = (v_0, t_0)$ to denote the first node and its associated time of the partial path.

Algorithm 2: Forward-backward path finder.

Data: A dynamic graph $G(V, E)$; the source node $s \in V$; the destination node $d \in V$; the starting time t_s; the maximum arriving time t_d; the transmission time τ; the minimum forward time t_f; the maximum backward time t_b; the forward-backward count c_{fb}; the set of partial forward paths P_f ; the set of partial backward paths P_b.

Result: A path P from s to d.

```
1  begin
2      c_fb = 0;
3      t_c = t_s;
4      P_f[c_fb] = {⟨(s, t_s)⟩};
5      P_b[c_fb] = {⟨(d, t_b)⟩};
6      while True do
7          if c_fb ≥ 2 then
8              L_n = commonNodes(P_f[c_fb], P_b[c_fb], G, τ);
9              if L_n ≠ ∅ then
10                 P = getPathByNodes(P_f[c_fb], P_b[c_fb], L_n);
11                 return P;
12             L_e = commonEdges(P_f[c_fb], P_b[c_fb], G, τ);
13             if L_e ≠ ∅ then
14                 P = getPathByEdges(P_f[c_fb], P_b[c_fb], L_e);
15                 return P;
16         tmpForwardSet = ∅;
17         foreach tmpForwardPath ∈ P_f[c_fb] do
18             tmpForwardSet =
                 tmpForwardSet ∪ extendForwardPath(G, tmpForwardPath, τ);
19         P_f[c_fb + 1] = tmpForwardSet;
20         get the minimum forward time t_f;
21         tmpBackwardSet = ∅;
22         foreach tmpBackwardPath ∈ P_b[c_fb] do
23             tmpBackwardSet =
                 tmpBackwardSet ∪ extendBackwardPath(G, tmpBackwardPath, τ);
24         P_b[c_fb + 1] = tmpBackwardSet;
25         get the maximum backwardTime t_b;
26         if t_f > t_b or c_fb ≥ ⌊(L_max − 2)/2⌋ then
27             there is no path between s and d;
28             return NULL;
29         P_f[C_fb + 1] = removeRedunantFpaths(P_f[C_fb + 1], t_b);
30         P_b[C_fb + 1] = removeRedunantBpaths(P_b[C_fb + 1], t_f);
31         c_fb = c_fb + 1;
32     return P;
```

The `CommonNodes` function, gets all common nodes that can be used to intersect forward and backward paths. The function will use P_f and P_b and try to compare the last node of partial paths in P_f, and the first node of partials paths in P_b to check if there are common nodes. That is, if $slast(P_f) = \{v_i|(v_i, t_i) = last(P'), \forall P' \in P_f\}$ denotes the set of last nodes from the set of partial paths P_f, and $sfirst(P_b) = \{v_j|(v_j, t_j) = first(P''), \forall P'' \in P_b\}$ the

set of first nodes of partials paths in P_b, then $CommonNodes = slast(P_f) \cap sfirst(P_b)$.

Similarly, the function `CommonEdges` will attempt to find the path based on common edges of the last nodes from P_f and first nodes from P_b. More specifically, in this function it will check if there are some valid edges which satisfy the time requirement of the path. $CommonEdges = \{e = (s, d, t, \lambda) \mid e_i \in E, t + \lambda \geq timeP_f(s) + \tau, t \leq timeP_b(d) - \tau, \forall s \in slast(P_f), d \in sfirst(P_b)\}$. Where $timeP_f(v)$ returns the time associated to v in the corresponding partial path from the set P_f, and $timeP_b(v)$ the time associate to v in P_b.

Then, function `getPathByNodes` and `getPathByEdges` will select a random node or edge from `CommonNodes` (L_n) or `CommonEdges` (L_e) respectively to construct the final path.

When the algorithm cannot get the path at the current iteration, it will use `extendForwardPath` and `extendBackwardPath` functions to increase the length of the partial paths both in P_f and P_b with all possible time-valid neighbors. The presence of loops is avoided by avoiding repeating a node when extending each path.

In order to reduce memory requirements the algorithm prunes partial paths from the forward and backward set. Paths which will not be possible to use for node or edge intersection are removed in each loop by the functions `removeRedundantFpaths` and `removeRedundantBpaths`. Forward partial paths, where the time of the last node is greater than t_b, and backward partial paths, where the time of the first node is lower than t_f are removed. As described in the algorithm, t_f is the minimum time of the last node among all paths in P_f, and t_b is the maximum time of the first node from all paths in P_b.

4 Measuring anonymity

Determining the anonymity achieved by using onion routing with a given path will depend on several factors. Even with our stochastic approaches from the previous section, the topology and behavior of the network are key issues to take into considerations. In this section we will describe measures that can help in determining the anonymity achieved based on several aspects of the chosen path and the network topology and temporal behavior.

4.1 Anonymity set and anonymity degree

In order to estimate the anonymity in onion routing schemes it is quite common to rely on k-anonymity models (Samarati, 2001) and entropy based metrics (Diaz et al., 2002; Serjantov and Danezis, 2002; Castillo-Perez and Garcia-Alfaro, 2013). In our case we will assume two different and analogous adversary models depending on the knowledge of the attacker. In general we assume that the attacker knows the dynamic behavior of the network (which is usually

public knowledge). Then, we consider two different cases, where the attacker knows:

1. The destination node and the time the message arrives to such node. In this case the goal of the attacker is to guess the source node of the communication.
2. The source node and the time when the message departed from such node. Now the attacker will attempt to guess the destination node.

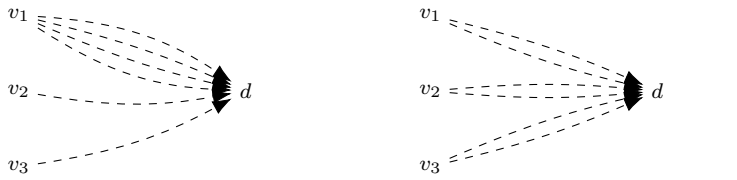
The first model is the most commonly considered in generic onion routing schemes. There, the common scenario is for a given client to use TOR to connect to a given server. The client wants to hide its identity (IP address) from the server. In our case, we have considered both cases in terms of completion. The second case will be such that the attacker is observing (monitoring) the source node and will try to guess the destination of the message.

In this section we will present the definition of *anonymity set* and *anonymity degree* for the first model. The second one is analogous.

The *anonymity set*, $S(v,t)$, of a node $v \in V$ that has received a message at time t, is the set of all possible source nodes (possible origins of the message). In the general case, $S(v,t) \subseteq V$ for any v and t. The size of $S(v,t)$ can be used as an estimation of privacy or anonymity as it is done in typical k-anonymity models.

To complement this measure, we determine the entropy of the anonymity set, by considering the probability associated to each possible source node, which might not be uniform. We introduce the measure of such probability based on the number of existing simple paths that can reach the destination node from all possible source nodes. A *simple* path is a path that does not repeat nodes.

As an example, in Figure 1, we can see two different scenarios with an anonymity set of size 3 for the destination node d at time t_s, $S(d,t_s) = \{v_1, v_2, v_3\}$, the dashed arrows represent paths and we consider that all edges are present all the time to simplify the example.



(a) $|S| = 3$,$H(S) = 1.25$ (b) $|S| = 3$, $H(S) = 1.58$

Fig. 1: Example of the same anonymity set with different entropy.

We denote the set of simple paths from v_1 to d arriving at node d at time t_d as $P_{t_d}(v_1, d)$, so $|P_{t_d}(v_1, d)|$ is the number of different simple paths from v_1 to d. With this in mind, it is clear that the anonymity degree should be different in each case of Figure 1.

We define the probability of a given node v of being the source node of a path arriving at node d at time t_d as:

$$\mathcal{P}_{d,t_d}(v,d) = \frac{|P_{t_d}(v,d)|}{\sum_{u_i \in S} |P_{t_d}(u_i,d)|} \tag{1}$$

We can then compute the entropy of the anonymity set S for the destination node d at time t_d as follows.

$$H(S(d,t_d)) = -\sum_{v_i \in S} \mathcal{P}_{d,t_d}(v_i,d) \log \mathcal{P}_{d,t_d}(v_i,d) \tag{2}$$

Following with the example, we have that $H(S(d,t_d))$ for Figure 1a is 1.25 while for Figure 1b it is 1.58. Intuitively it is reasonable to have greater entropy for the second case, where the uncertainty about the origin is greater based on the possible paths to the destination node.

The maximum entropy of the anonymity set is achieved when $S = V$ and the probability for all $v_i \in S$ is uniform. Thus, we define the maximum entropy for the anonymity set as:

$$H_M = log(|V|) \tag{3}$$

Now we can introduce the anonymity degree $\mathcal{A}$ for a given destination node $d \in V$ and time t_d as:

$$\mathcal{A}(d,t_d) = \frac{H(S(d,t_d))}{H_M} \tag{4}$$

We are assuming here that the path is constructed following an stochastic approach and all paths from source to destination are equally probable.

Thus, the minimum possible value of the anonymity degree is 0, and will be achieved when there exists just one possible path ending at the destination node v at time t. In this case, the entropy of the anonymity set is 0, because there is no uncertainty about the source node. In turn, the maximum possible value of the anonymity degree is $1 - \epsilon$ (for a small ϵ), and it is reached when the entropy of the anonymity set is maximal, that is, when all nodes in the graph are valid sources and the probability of a path starting in each of them is exactly the same. Note that the entropy will never reach $\log(V)$, since the destination node v itself will never be a valid source node (we are requiring paths to be *simple* paths). As a consequence, the maximum anonymity degree tends to 1, but can not be exactly 1.

The anonymity degree informs about the privacy of a destination node at a given time in the context of a network of a certain order. Therefore, it can be used to compare the anonymity of nodes inside the same network or with nodes of other networks of the same order.

Although networks with high density are intuitively better from a privacy perspective, and will be so in the general case, we can not ensure that the anonymity degree in higher density networks will always be better. Think, for instance, in a very dense network where most of the paths end up at the same node.

On the contrary, if we only consider the size of the anonymity set, higher density networks will always present bigger (or equal) anonymity sets.

An important drawback of this anonymity measure is that its computation is not feasible for high density networks. Finding all simple paths has exponential memory requirements making it unfeasible for common desktop computers. We have used a time constrained modification of the Rubin algorithm (Rubin, 1978) to compute all simple paths in relatively low density networks. For higher density networks we had to develop an approximated method.

The approximated method computes the approximated anonymity set, entropy of the anonymity set, and anonymity degree of a given destination node d which receives a message at time t_d by repeatedly searching backwards for simple paths ending at d. That is similarly as doing a time constraining reversed random walk repeatedly.

At each iteration of the algorithm, a new path ending at node d at t_d is searched by randomly selecting the previous hop of the path between the time backward neighbors (see Section 2) of the current node that have not been visited previously (i.e., that are not already part of the path).

At each hop of the path, the current time is updated and each of the iterations stops whenever one of the following conditions is reached: *a)* the target number of hops is reached, *b)* the maximum duration time is exceeded. When the target number of hops of the path is reached, the iteration finishes and the last node is considered the source node of the path. On the contrary, when the maximum duration time is exceeded, the path is discarded. The approximated anonymity set of d at starting time t_d, $S'(d, t_d)$, is then the set of source nodes found during all the iterations of the algorithm.

Given an execution of the algorithm with n iterations, $p(v)$ is the number of times one of the iterations ends up in each source node v. Then, the estimated probability of node v being a source for a path ending at d at time t_d, ${\mathcal{P}'}_{d,t_d}(v, d)$, is $p(v)/n$.

Finally, the approximated entropy, H', and the approximated anonymity degree, $\mathcal{A}'$, can be computed with equations 2 and 4, using $\mathcal{P}'$ instead of $\mathcal{P}$.

On the one hand, notice that $|S'|$ will always be less than or equal to $|S|$, but this relation does not necessarily hold for $\mathcal{A}$ and $\mathcal{A}'$. On the other hand, the greater the number of iterations n of the algorithm, the more closer the results of the approximated algorithm should be with the exact ones.

We have seen how to compute S, $\mathcal{A}$ (and S', $\mathcal{A}'$) for the first adversary model, where the attacker knows the destination node and the time that the message arrives to such node. From now on we will denote these measures as $\overleftarrow{S}$, $\overleftarrow{\mathcal{A}}$ (and $\overleftarrow{S'}$, $\overleftarrow{\mathcal{A}'}$). Analogous measures of the anonymity set and the anonymity degree can be computed for the second adversary model. In this case the attacker knows the source node and the starting time and we can find all possible paths starting at such node in such given time. The definition of these measures is analogous to the ones we have seen and will be denoted from now on as $\overrightarrow{S}$, $\overrightarrow{\mathcal{A}}$ (and $\overrightarrow{S'}$, $\overrightarrow{\mathcal{A}'}$)

4.2 Path-degree measure

We have considered another metric based on the degree of all the nodes of a given path. In some sense these degrees give an estimation of the path uncertainty in each node. If an attacker knows a partial path, or can identify a given node in the path, the degree of the node and following unknown nodes can be seen as the difficulty in guessing the rest of the path. Moreover, the degree of a node in the path can be seen as the probability that an attacker has in guessing the next node of the path knowing only the current node and time.

Given a path $P = \langle (v_1, t_1) \dots (v_l, t_l) \rangle$, of length l, its *path-degree* measure $\mathcal{D}$ is defined as:

$$\mathcal{D}(P) = \frac{1}{2} \left(\prod_{i=1}^{l} \left(deg_{\mathcal{C}}^{t+}(v_i, t_i, \delta_M) \right)^{-1} + \prod_{i=1}^{l} \left(deg_{\mathcal{C}}^{t-}(v_i, t_i, \delta_M) \right)^{-1} \right) \quad (5)$$

The path-degree measure combines the time forward and backward degrees of the nodes in the path in order to give a generic measure. The measure is defined in the interval $[0, 1]$. The value 1 is given by the worst case, where the degree of each node in the path, both forward and backward times, is 1. On the contrary, values close to 0 are better from a privacy perspective since they denote higher degrees in the path.

The measure is not normalized with respect to the path length because we believe that such length should be taken into consideration. Higher lengths are better for privacy and will yield lower values for the path-degree measure.

4.3 Discussion about the privacy measures

When considering privacy in our proposal, the anonymity set size measure can be comparable to the well known k-anonymity model. The related anonymity degree helps in better understand the actual distribution of probabilities within such anonymity set. It is important to note that we need to provide both measures together, given that the anonymity degree is given as a ratio of the maximum possible entropy. As an example, we can have two different anonymity sets with the same anonymity degree but with very different sizes. In such case, clearly the case with the bigger set is better for privacy. When the size of the anonymity set is the same, the cases with higher entropy are better for privacy. In some sense, the anonymity degree can be seen as a measure of diversity related to the anonymity set, similarly as l-diversity (Machanavajjhala et al., 2007) is related to k-anonymity. It gives information about the distribution of probabilities within the anonymity set.

On the other hand, the path-degree measure, gives information about how easy or difficult it might be to obtain a given path by a random walk (forward and backwards). It gives information about the diversity of alternatives to follow in the path. They main idea is to summarize in a very broad and general

manner, the possible difficulties that an attacker observing a partial path can have to complete the whole path.

Both types of measures are not directly related since they measure different things. The anonymity set and anonymity degree are possible the most interesting ones, and the ones that have more relation to how anonymity is traditionally measured. We think however that the path-degree measure is an interesting complement to the other ones.

As we will observe in the evaluation (see Section 5) in general all privacy measures are highly related to the density of the network. Higher density will yield better privacy measures.

5 Evaluation

In this section we present the evaluation of our proposal and the results we obtain. First, we introduce the dataset and setup of the experiments, and then the results regarding performance and anonymity.

5.1 Dataset

In our experiments we have used the CRAWDAD *rice/ad_hoc_city* dataset (Jetcheva et al., 2003). This is a wireless ad hoc network based on the Seattle public bus transportation. Each network node is located in a bus and contacts between them are established based on the coverage of the two nodes (ability to send a message between them). It can be considered as a POppNet due to the high predictability in bus timetables and itineraries. The network contains round 1200 buses covering a 5100 square kilometer area. It is clear that there will be errors in the predictability of the network but we assume these errors to be tolerable in the application domain (e.g. the use of our proposal in live critical communication systems is completely discouraged). This is something usually assumed and accepted in opportunistic networking. Transmission time is assumed to be 1 second (this includes establishing the connection and sending a short message).

Network	$\|V\|$	$\|E\|$	avg. degree	graph density
N1	1179	543692	922.293	0.783
N2	1179	271813	461.091	0.391
N4	1177	135976	231.055	0.196
N8	1178	67896	115.273	0.098
N16	1170	34090	58.274	0.050
N32	1157	17064	29.497	0.026

Table 1: Datasets

The network that we consider spans through 24 hours, and has a high density with more than 500,000 dynamic edges during this time interval. Network density is a key issue in our proposal. Higher density means it is easier to construct paths and more difficult for an attacker to predict possible potential paths. In order to test our results with lower density networks we have produced different versions of the same Seattle network with lower number of edges. More precisely, we have obtained networks $N2$, $N4$, $N8$, $N16$, $N32$, by randomly selecting 1/2, 1/4, 1/8, 1/16, 1/32 number of edges of the original network denoted as $N1$. Table 1 shows the number of edges, average degree, and graph density for each network taken from their corresponding static graph G^*. Given that they correspond to the same network, with the same number of nodes and the same overall behavior it makes the comparison between them to be focused exclusively on the network density.

We also note that the distribution of the edges overtime is not uniform, as expected in a real network based on a public transportation system. There are hours with higher density than others, presumably corresponding to rush hours as shown in Figure 2.

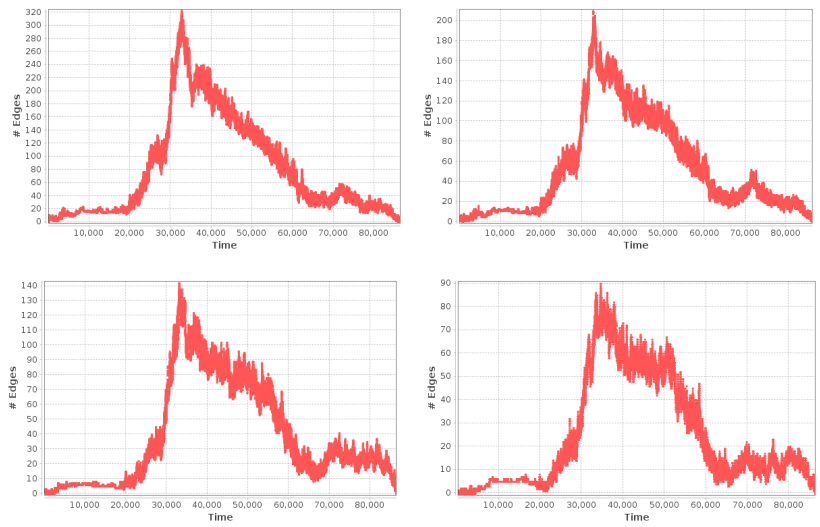


Fig. 2: Distribution of edges for networks $N1$, $N4$, $N16$, and $N32$ over 24 hours (time given in thousands of seconds).

We have performed our experiments in different time frames. We consider two types of starting times for our experiments. The experiments denoted as *zero* are paths that start at a random time from the interval $[0, 10000]$, while experiments denoted as *rush* are starting in the interval $[20000, 35000]$.

The performance experiments from Section 5.2.1 have been executed in a desktop computer, Intel i7 CPU at 3.4GHz, 16GB memory. For the exhaustive

computation of the anonymity degree (cf. Table 4) we used a computer with an Intel Xeon E3-1230 V2 at 3.30GHz, and 30 GB of memory.

Given the stochastic nature of our proposed algorithms, and the different scenarios that can happen in real networks, each experiment is composed of 100 cases for *zero* and *rush* starting times, giving a total of 200 executions. For each case a different pair of nodes (source and target), and starting time are randomly selected, then the average is usually considered. We also separate *zero* and *rush* experiments in order to be able to appreciate different situations.

We will denote our random algorithm as R (Section 3.1), and the forward-backward algorithm as FB (Section 3.2). In order to properly evaluate our proposal, we need to consider the penalty introduced by using our algorithms. That is, what do we have to sacrifice in terms of performance as compared to a non-anonymous routing. For such purpose we have considered a shortest path algorithm for dynamic graphs. We have used an adaptation of the Dijkstra algorithm to take into consideration time constraints, similarly to the one from (Xuan et al., 2003). This algorithm is deterministic and thus, not desirable for anonymous communications. We will denote it as X.

We have conducted experiments both in terms of performance and privacy. Performance experiments show how good are the stochastic algorithms in terms of their efficiency, while privacy experiments attempt to evaluate their security.

5.2 Performance

We show here the performance evaluation of finding paths using the algorithms from Section 3 by evaluating several parameters or characteristics and usually comparing them to the results obtained with the shortest path algorithm X. The X algorithm can be considered in most cases the best performance achievable.

5.2.1 Applicability related to network density

To be applicable, our algorithms need to be able to find a number of paths from source to destination. An algorithm can fail in finding a path from source to destination either due to the nonexistence of this path or because it is just not able to find an existent path due to its stochastic nature. Not only we need to find a path but a minim number of paths big enough to provide good anonymity. We compare the behavior of R, and FB algorithms to that of X, and related to the density of the network in Figure 3. The failure rate indicates the average percentage of paths not found on each network. Clearly, if a path exists between two nodes X will provide such path, while R, and FB might fail.

We can see that any algorithm can be successfully applied more than 95% of the times on networks with high density ($N1$ to $N4$), while, when density decreases, the behavior of FB is very close to that of X and thus is preferable

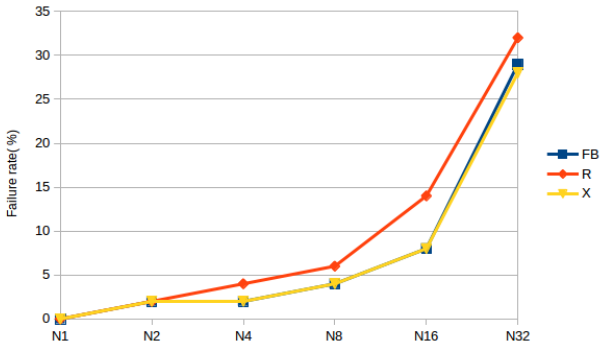
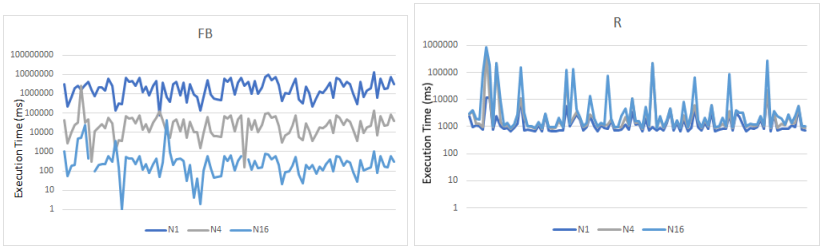
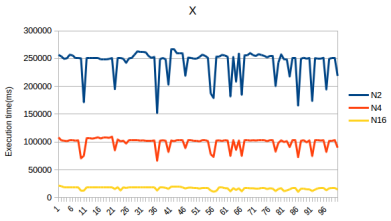


Fig. 3: Applicability of the algorithms on networks with different density.



(a) Execution time for FB. (b) Execution time for R.



(c) Execution time for X.

Fig. 4: Average execution time

over R. FB can still be successfully applied more than 90% of the times on $N8$ and $N16$. However, on $N32$, which has a very low density, the applicability of any algorithm sharply decreases.

5.2.2 Execution time

Figure 4 shows the execution time of X, R, and FB on networks with different density. In general our algorithms R and FB are faster than X, notably R. As expected, execution time is proportional to the network density, although the difference is less appreciable in R.

5.2.3 Path length

The minimum path length should be always 5, since it is the minimum path length required for onion routing to be secure. Even so, larger paths can be found by R and FB given their stochastic behavior. We compare the results of R and FB with the actual shortest path obtained from X.

Net	FB	R	X
N1	5.0 (0.0)	6.8 (1.4)	4.0 (1.0)
N2	5.0 (0.0)	6.9 (1.5)	4.0 (1.0)
N4	5.0 (0.0)	7.1 (1.6)	4.0 (1.0)
N8	5.1 (0.6)	7.2 (1.7)	4.2 (1.7)
N16	5.2 (0.9)	6.8 (1.5)	5.0 (1.0)
N32	5.5 (1.3)	7.1 (1.3)	5.8 (2.5)

(a) Zero time

Net	FB	R	X
N1	5.0 (0.0)	6.8 (1.0)	3.5 (1.1)
N2	5.0 (0.0)	6.7 (1.2)	3.7 (1.0)
N4	5.1 (0.4)	7.1 (1.7)	3.7 (1.1)
N8	5.1 (0.6)	7.3 (1.7)	4.4 (1.5)
N16	5.2 (0.9)	7.1 (1.2)	4.5 (1.5)
N32	5.5 (1.3)	7.2 (1.6)	5.0 (1.8)

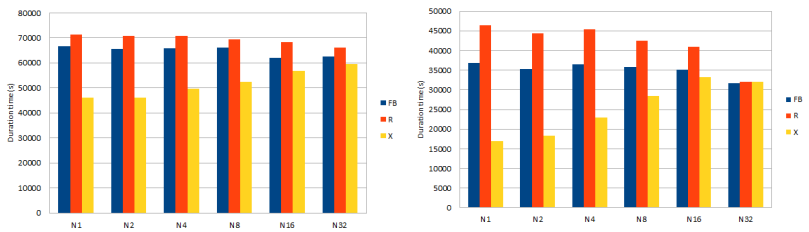
(b) Rush time.

Table 2: Average path length. Standard deviation is shown in parenthesis.

Table 2 shows the average path length for each execution of FB and R and the shortest path, X. FB obtains paths closest to the required minimum due to its meet-in-the-middle strategy. Paths from R are larger but within a reasonable range.

5.2.4 Duration time

Figure 5 shows the duration time (c.f. Section 2) for all the algorithms and networks in zero and rush times.



(a) Zero time (b) Rush time

Fig. 5: Duration time

Shortest duration times are obtained in rush time experiments, and in general FB paths have a shorter duration than R. In any case we consider the penalty in duration time, as compared to X, to be quite acceptable for our scenarios.

5.3 Anonymity

As mentioned in Section 4.1, the anonymity set size and the anonymity degree can be difficult to compute in high density networks. We can however use the approximate method also introduced in Section 4.1. We have computed the anonymity set size and anonymity degree for both adversarial scenarios from Section 4.1, that is $|\overleftarrow{S}|$, $\overleftarrow{\mathcal{A}}$, and $|\overrightarrow{S}|$, $\overrightarrow{\mathcal{A}}$, and their respective approximated values.

Table 4 shows the exact anonymity degree and anonymity set size for $N32$, and $N16$. Those are the only networks where we could compute the exact values with our equipment (cf Section 5.1). On the other hand, Table 6 shows their approximation using the approach described in Section 4.1. The results where obtained using a fixed path length of five nodes and fixing the maximum duration of the path to the average of the cases obtained from algorithm R as detailed in the previous section. One million iterations of the algorithm were performed for each analyzed source node.

Net	$\lvert\overleftarrow{S}\rvert$	$\overleftarrow{\mathcal{A}}$	Net	$\lvert\overleftarrow{S}\rvert$	$\overleftarrow{\mathcal{A}}$
N32	834.47 (228.5)	0.7969 (0.16)	N32	712.36 (316.8)	0.7632 (0.21))
N16	1045.61 (120.5)	0.8703 (0.10)	N16	1022.83 (141.6)	0.8717 (0.11)

(a) Zero hours (b) Rush hours

Table 3: Exact anonymity set size ($\lvert\overleftarrow{S}\rvert$) and degree ($\overleftarrow{\mathcal{A}}$) for N32, N16, N8. Standard deviation is shown in parentheses.

Net	$\lvert\overrightarrow{S}\rvert$	$\overrightarrow{\mathcal{A}}$	Net	$\lvert\overrightarrow{S}\rvert$	$\overrightarrow{\mathcal{A}}$
N32	923.66 (272.4)	0.8229 (0.14)	N32	822.03 (344.2)	0.7855 (0.22)
N16	1096.6 (154.85)	0.8703 (0.10)	N16	1038.69 (226.6)	0.8551 (0.12)

(a) Zero hours (b) Rush hours

Table 4: Exact anonymity set size ($\lvert\overrightarrow{S}\rvert$) and degree ($\overrightarrow{\mathcal{A}}$) for N32, N16, N8. Standard deviation is shown in parentheses.

We consider the approximated approach to be very accurate for the anonymity degree. The estimation error is given in Table 7 for both adversary models The estimation error on the anonymity set size is a bit bigger, but we consider it also to be acceptable (see the relative error from Table 7). This is due to the fact that finding all possible paths with the approximated approach is difficult. This approach will more likely find destination nodes with higher number of paths, which makes the anonymity degree more accurate that the anonymity set approximations.

Net	$\lvert\overleftarrow{S'}\rvert$	$\overleftarrow{\mathcal{A}'}$
N32	801.19 (222.01)	0.854596 (0.12)
N16	951.89 (97.27)	0.904594 (0.03)
N8	1004.04 (81.24)	0.912801 (0.03)
N4	1025.11 (65.5)	0.916637 (0.02)
N2	1036.66 (46.6)	0.918900 (0.02)
N1	1043 (46.93)	0.919998 (0.02)

(a) Zero hours

Net	$\lvert\overleftarrow{S'}\rvert$	$\overleftarrow{\mathcal{A}'}$
N32	566.5 (271.46)	0.746111 (0.23)
N16	849.6 (127.59)	0.875215 (0.05)
N8	930.81 (77.11)	0.895185 (0.03)
N4	964.9 (68.14)	0.901639 (0.03)
N2	969.28 (62.61)	0.902211 (0.03)
N1	989.09 (63.57)	0.905207 (0.03)

(b) Rush hours

Table 5: Approximated anonymity set size ($\lvert\overleftarrow{S'}\rvert$) and degree ($\overleftarrow{\mathcal{A}'}$). Standard deviation is shown in parentheses.

Net	$\lvert\overrightarrow{S'}\rvert$	$\overrightarrow{\mathcal{A}'}$
N32	792.88 (297.7)	0.825724 (0.2)
N16	970.88 (253.8)	0.889494 (0.12)
N8	1027.2 (236.74)	0.908159 (0.07)
N4	1061.1 (203.28)	0.920988 (0.05)
N2	1067.69 (196.53)	0.922615 (0.04)
N1	1074.93 (195.42)	0.924665 (0.04)

(a) Zero hours

Net	$\lvert\overrightarrow{S'}\rvert$	$\overrightarrow{\mathcal{A}'}$
N32	565.77 (304.9)	0.736670 (0.26)
N16	852.38 (292.27)	0.842796 (0.18)
N8	966.36 (241.81)	0.891140 (0.09)
N4	1020.53 (221.87)	0.909172 (0.05)
N2	1027.16 (218.55)	0.909134 (0.06)
N1	1042.02 (204.87)	0.913349 (0.05)

(b) Rush hours

Table 6: Approximated anonymity set size ($\lvert\overrightarrow{S'}\rvert$) and degree ($\overrightarrow{\mathcal{A}'}$). Standard deviation is shown in parentheses.

Note that the results of both privacy measures (namely, the anonymity set size and the anonymity degree) increase with the density of the network. Moreover, in this scenario, utility can be understood as the ability to successfully and efficiently send messages. Therefore, utility also increases with the density of the network (see Section 5.2: when more edges are kept in the network, it is more probable to find a valid path that can be used to send a message. As a consequence, in this scenario privacy and utility go hand by hand.

We can also compute the path degree measure from Section 4.2 using the paths found in Section 5.2. As an example, the averages of all cases for each network and time interval for paths found with the FB algorithm are given in Table 8.

We can see that, although the path-degree measure increases as the network density decreases, the values for the lower density network are still very low. This measure gives an idea of the degree of each node in the path, or the difficulty for an attacker of guessing the next node (given a concrete node on the path) in both directions, forward and backward.

Net	Time	$\lvert\overleftarrow{S}\rvert$ error		$\overleftarrow{\mathcal{A}}$ error	
		absolute	relative	absolute	relative
N32	Zero	33.28	0.0399	0.0577	0.0724
	Rush	145.86	0.2048	0.0171	0.0224
N16	Zero	93.72	0.0896	0.0343	0.0394
	Rush	173.23	0.1694	0.0035	0.0041
Net	Time	$\lvert\overrightarrow{S}\rvert$ error		$\overrightarrow{\mathcal{A}}$ error	
		absolute	relative	absolute	relative
N32	Zero	130.78	0.1416	0.0028	0.0034
	Rush	256.26	0.3117	0.0488	0.0621
N16	Zero	125.72	0.1146	0.0192	0.0220
	Rush	186.31	0.1794	0.0123	0.0143

Table 7: Estimation error in the approximated calculations.

	$\mathcal{D}(P)$	
Network	Zero	Rush
N1	$1.26E-20$	$1.22E-18$
N2	$4.65E-19$	$9.60E-18$
N4	$2.96E-16$	$1.78E-15$
N8	$2.89E-15$	$2.82E-14$
N16	$4.94E-11$	$3.26E-09$
N32	$6.10E-09$	$5.00E-14$

Table 8: Path-degree measure

6 Related Work

The study of POppNets in the literature is usually focused on improving routing as compared to generic OppNets. POppNets appear in specific scenarios, such as satellite networks, public bus networks, or even human mobility (Song et al., 2010). The use of such predictability to improve or design security related services or mechanisms in the context of OppNets has not be exploited yet to our knowledge. Examples of the first case are (Huang et al., 2015), (Fan et al., 2015), or (Fraire and Finochietto, February 2015).

Anonymous communications have been studied in the context of Opportunistic networks and Delay-tolerant networks. In general, due to the lack of end-to-end connectivity on these kind of networks, security solutions for them are difficult and complex. Furthermore, one cannot relay on common solutions from more conventional connected networks due to the need to support disruptions and delays.

In this regard, (Kate et al., 2007) proposes a security architecture for DTNs using Identity-Based Encryption (IBE) with support for anonymous authentication with pseudonyms. Anonymous routing is somehow achieved by using gateways that hide the sender/receiver.

A common approach in several works is to apply an onion routing strategy in DTNs, but creating onion groups. That is, nodes are grouped and onion layers are based on those groups. Any node of the group can forward the message of the corresponding layer. This usually implies that all nodes of the group can decrypt the layer. ARDEN (Shi et al., 2012) uses groups of nodes to perform the layering process and broadcast messages between these groups. The message route between groups is chosen randomly. To perform the cryptographic layers attribute based encryption (ABE) is used. Similarly, (Sakai et al., 2016) also uses onion groups and allows several copies of the message. In (Vakde et al., 2011), authors allow such groups to be dynamic, but use straightforward public key cryptography, having all members of the group sharing the same private key.

Another interesting work is (Lu et al., 2010). Here authors want to hide the physical location of the sender. They propose to fragment the message and send the fragments to different receivers, with the aim of creating confusion for the attacker.

In this line, another approach to achieve anonymous communications in dynamic networks in general, is precisely to introduce noise in the communications. This approach is very common in traditional data privacy (including Statistical Disclosure Control, or Privacy-preserving Data Mining) (Torra, 2017). As an example (van den Hooff et al., 2015) attempts to ensure differential privacy for several observable characteristics or metadata. These approaches are quite different from the ones presented in this paper, and can be seen as alternate methods. We believe that knowing the network behavior in advance makes our approach more suitable for fast transmission of single short messages.

Most of the existing works on anonymous communications deal with general OppNets, where predictability of network behavior is not considered. We are not aware at the time of writing about publications on anonymous communication in POppNets. Solutions from non predictable networks can obviously be applied to POppNets, but taking into account its predictability allows to simplify and improve the anonymous routing in such networks.

Our proposal could be applied to other types of dynamic networks like more general mobile ad-hoc networks (MANETs). In such networks however, our proposal lacks interest since common onion routing can be applied quite similarly as it is done in static networks such a the TOR network in the Internet. We believe that the fact that we are dealing with opportunistic networks is what makes our proposal valuable and interesting.

7 Conclusions

We have analyzed the use of simple onion routing in POppNets to achieve message anonymity. We conducted our experiments on a concrete network from the Seattle public bus transportation system. We evaluated path establishment and anonymity degree in this network with variable network density. We have

also provided tools to asses the anonymity for the paths. These are concrete anonymity measures that can help in evaluating several characteristics related to path anonymity. It is clear that our approach will be very dependent on the actual network topology and behavior and thus, we provide tools to evaluate the convenience of using onion routing in these types of networks. Our proposal is especially suitable in high density networks, allowing anonymous routing with relatively simple mechanisms as compared to other OppNet solutions.

We have used a very simplistic approach for onion routing using only the public keys of the nodes to build the onion layers. We assume that we can directly use each node public key instead of establishing a session key, as this establishment incurs in more penalty than gain. This is convenient in our scenario and in general in OppNets but more advanced solutions could be exploited for other applications. We have also not considered the performance needed by nodes to perform the cryptographic operations since they are negligible if compared to the network delays. As future work we also consider the combination of this approach with different anonymity techniques. One of such techniques is the introduction of noise or confusion in the network through synthetic network messages. The combination of these two approaches can be useful in for example lower density networks where path predictability could be more difficult to ensure.

Acknowledgments

This paper has been partially supported by Spanish Government under grants TIN2017-87211-R, RTI2018-095094-B-C22 "CONSENT", and TIN2014-57364-C2-2-R "SMARTGLACIS". Depeng Chen acknowledges the support from the China Scholarship Council, Grant No. 201606140138. C. Pérez-Solà was working at Universitat Rovira i Virgili when some of this work was done. Authors acknowledge Cristina Fernandez-Cordoba and Jordi Herrera-Joancomarti for insightful comments on some aspects related to the work described in this paper.

References

Antunez-Veas A, Navarro-Arribas G (2016) Onion routing in deterministic delay tolerant networks. In: Foundations and Practice of Security, Springer International Publishing, no. 9482 in Lecture Notes in Computer Science, pp 303–310

Borah SJ, Dhurandher SK, Woungang I, Kumar V, Barolli L (2018) A multi-objectives based technique for optimized routing in opportunistic networks. Journal of Ambient Intelligence and Humanized Computing 9(3):655–666

Borrego C, Borrell J, Robles S (2019) Efficient broadcast in opportunistic networks using optimal stopping theory. Ad Hoc Networks 88:5 – 17, DOI https://doi.org/10.1016/j.adhoc.2019.01.001

Casteigts A, Flocchini P, Quattrociocchi W, Santoro N (2012) Time-varying graphs and dynamic networks. International Journal of Parallel, Emergent and Distributed Systems 27(5):387–408

Castillo-Perez S, Garcia-Alfaro J (2013) Onion routing circuit construction via latency graphs. Computers & Security 37:197214, DOI 10.1016/j.cose.2013.03.003

Chen D, Navarro-Arribas G, Borrell J (2017) On the applicability of onion routing on predictable delay-tolerant networks. In: 2017 IEEE 42nd Conference on Local Computer Networks (LCN), pp 575–578, DOI 10.1109/LCN.2017.17

Diaz C, Seys S, Claessens J, Preneel B (2002) Towards measuring anonymity. In: Privacy Enhancing Technologies, Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, p 5468

Fan L, Minsu H, Zhiyuan y, Chao Z, Wang Y (2015) Reliable topology design in time-evolving delay-tolerant networks with unreliable links. IEEE Transactions on Mobile Computing 14(6):1301–1314

Fraire J, Finochietto JM (February 2015) Routing-aware fair contact plan design for predictable delay tolerant networks. Ad Hoc Networks 25:303–313

Goldschlag DM, Reed MG, Syverson PF (1996) Hiding routing information. In: Proc. of Information Hiding, p 137150

Holme P, Saramki J (2012) Temporal networks. Physics Reports 519(3):97–125

van den Hooff J, Lazar D, Zaharia M, Zeldovich N (2015) Vuvuzela: Scalable private messaging resistant to traffic analysis. In: Proceedings of the 25th Symposium on Operating Systems Principles, ACM, SOSP 15, pp 137–152

Huang M, Chen S, Fan L, Yu W (2015) Topology design in time-evolving delay-tolerant networks with unreliable links. In: Global Communications Conference

Jain S, Fall K, Patra R (2004) Routing in a delay tolerant network. In: Proc. of SIGCOMM '04, p 145158

Jetcheva JG, Hu YC, PalChaudhuri S, Saha AK, Johnson DB (2003) CRAWDAD dataset rice/ad_hoc_city (v. 2003-09-11). Downloaded from `https://crawdad.org/rice/ad_hoc_city/20030911`, DOI 10.15783/C73K5B

Kate A, Zaverucha GM, Hengartner U (2007) Anonymity and security in delay tolerant networks. In: Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on, IEEE, pp 504–513

Kostakos V (2009) Temporal graphs. Physica A: Statistical Mechanics and its Applications 388(6):1007–1023

Lu X, Hui P, Towsley D, Pu J, Xiong Z (2010) Anti-localization anonymous routing for delay tolerant network. Computer Networks 54(11):1099–1910

Machanavajjhala A, Kifer D, Gehrke J, Venkitasubramaniam M (2007) L-diversity: Privacy beyond k-anonymity. ACM Trans Knowl Discov Data 1(1)

Mota VF, Cunha FD, Macedo DF, Nogueira JM, Loureiro AA (2014) Protocols, mobility models and tools in opportunistic networks: A survey. Computer Communications 48:5–19, DOI 10.1016/j.comcom.2014.03.019

Pan RK, Saramki J (2011) Path lengths, correlations, and centrality in temporal networks. Physical Review E 84(1):016105(10)

Rubin F (1978) Enumerating all simple paths in a graph. IEEE Transactions on Circuits and Systems 25(8):641–642, DOI 10.1109/TCS.1978.1084515

Sakai K, Sun MT, Ku WS, Wu J, Alanazi FS (2016) An analysis of onion-based anonymous routing for delay tolerant networks. In: Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on, IEEE, pp 609–618

Samarati P (2001) Protecting respondents identities in microdata release. IEEE Transactions on Knowledge and Data Engineering 13(6):1010–1027, DOI 10.1109/69.971193

Scott K, Burleigh S (2007) Bundle protocol specification. RFC 5050, IETF

Serjantov A, Danezis G (2002) Towards an information theoretic metric for anonymity. In: Privacy Enhancing Technologies, Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, pp 41–53

Sharma DK, Dhurandher SK, Agarwal D, Arora K (2019) krop: k-means clustering based routing protocol for opportunistic networks. Journal of Ambient Intelligence and Humanized Computing 10(4):1289–1306

Shi C, Luo X, Traynor P, Ammar MH, Zegura EW (2012) Arden: Anonymous networking in delay tolerant networks. Ad Hoc Networks 10(6):918–930

Song C, Qu Z, Blumm N (2010) Limits of predictability in human mobility. Science 327(5968):1018–1021

Torra V (2017) Data Privacy: Foundations, New Developments and the Big Data Challenge. Studies in Big Data, Springer

Vakde G, Bibikar R, Le Z, Wright M (2011) Enpassant: Anonymous routing for disruption-tolerant networks with applications in assistive environments. Security and Communication Networks 4(11):1243–1256

Xuan BB, Ferreira A, Jarry A (2003) Computing shortest, fastest, and foremost journeys in dynamic networks. International Journal of Foundations of Computer Science 14(02):267–285