Efficient Unobservable Anonymous Reporting against Strong Adversaries

Nethanel Gelernter
Department of Computer Science
Bar Ilan University
Email: nethanel.gelernter@gmail.com

Amir Herzberg
Department of Computer Science
Bar Ilan University
Email: amir.herzberg@gmail.com

***Abstract*—We present DURP, a decentralized protocol for unobservable, anonymous reporting to an untrusted destination, with low latency and overhead. DURP provably ensures strong anonymity properties, as required for some applications (and not provided by existing systems and practical designs, e.g., Tor), specifically:**
– Provable unobservability against global eavesdropper and malicious participants.
– Provable source anonymity against a malicious destination.
– Probable-innocence against a malicious destination which is *also* a global eavesdropper.

DURP design is a modular combination of two modules: a *queuing module*, ensuring fixed rates for certain events, together with an *anonymization module*, which can use either Onion-Routing (DURP^{OR}) or Crowds (DURP^{Crowds}). We present analysis, backed by simulation results, of the network properties and performance of DURP, and show it has reasonable overhead. We also use the analysis results to create an optimized version of DURP.

I. Introduction

Anonymous communication is both important and challenging, motivating an abundance of research and systems. In fact, different applications and scenarios may require very different 'anonymity' requirements, and indeed many proposals and systems are incomparable since they satisfy different requirements or provide different trade-offs (often, between performance and anonymity properties). In fact, known solutions range from very weak anonymity properties (and low overhead), to stronger anonymity properties (but often unacceptably-high overhead).

In the recent years, most attention was given to weak anonymity (and high efficiency) decentralized designs, e.g., Tor [1], the most well known and widely used anonymous networking system. Tor is designed specifically to ensure 'low-latency', and its anonymity properties are modest, e.g., an eavesdropper listening on both sender and destination, can identify when they communicate. In particular, this holds for attacker that controls both the entry and the exit nodes.

The focus on low-latency and low-overhead anonymity networks, is mainly due to two reasons: *(1)* Prioritizing efficiency over anonymity, and *(2)* belief that strong attackers, in particular global eavesdroppers, are impractical.

The efficiency concerns are understandable; even surfing using Tor involves noticeable overhead. However, stronger attackers may often be a realistic concern, e.g., see reports of governmental eavesdropping, domestically and internationally. Furthermore, several works show that weaker attackers may also obtain 'MitM-like' abilities, e.g., off-path [2] and DoS attacks [3] [4] on Tor and by controlling few relays. It is also claimed that some Tor relays are controlled by 'snoops' [5]. These attacks demonstrate the tangibility of the threat from strong attackers (e.g., eavesdroppers, malicious participants and combinations of them), even from entities without many resources.

In this work, we show DURP, a decentralized protocol ensuring strong anonymity properties, specifically, unobservability against global eavesdroppers and malicious participants, together with anonymity against the destination. DURP is a practical and reasonably efficient, with a completely decentralized, peer-to-peer, scalable design.

DURP ensures strong anonymity properties, including *unobservability*, preventing eavesdroppers, or even other participants, from knowing that an entity communicates. This is required for some sensitive applications. However, in this work, we focus on a anonymous and unobservable communication to a single destination.

Specifically, we define and study the *unobservable reporting problem*. In this problem there are many *reporters*, all of them might want to send anonymous reports about confidential events to a known destination. The messages must be anonymous, such that the destination cannot know who is the reporter. Furthermore, due to the confidentiality of the events, and the known relationships between the reporters and the destination, regular anonymity is not enough: we want to hide the fact that a report was made, even from a global adversary which controls many of the reporters. Otherwise, such attacker might detect the existence of a report, and thereby learn the existence of a confidential event.

Reporting is an important application, which may require unobservable, anonymous communication, in particular, when the mere fact of a sender making a report may leak sensitive information. For example, consider detection of some critical scenario, e.g., a new cyber attack (where pattern of reports may expose vulnerabilities or operations), or reporting of large financial transactions in order to detect money-laundering operations (where the pattern of reports may leak business information such as number of large transactions). In these and other applications, an anonymous reporting mechanism may help detect critical issues (e.g., money laundering), by combining many indicators from different sources.

Specifically, we present Decentralized Unobservable Reporting Protocol (DURP), the first decentralized, reasonably-efficient protocol, which *provably* achieve both of the following crucial anonymity properties:
Unobservability against malicious peers: even an attacker which can eavesdrop on the communication, and also controls some of the participants (peers) in the network, cannot detect the existence or amount of communication.
Sender anonymity: hide the identity of the sender of a message from a malicious destination. Note that the destination can trivially detect the existence and amount of communication, i.e., unobservability against the destination is impossible.

Furthermore, DURP provides also some anonymity against a 'hybrid' attacker, who controls the *destination* as well as has the ability to eavesdrop on the communication. Our design includes the *On Load Send More (OLSM)* optimization, which further improves the anonymity against such 'hybrid' attacker.

A. Contributions

We present DURP, a *completely decentralized anonymous reporting* protocol. DURP is practical, with reasonable performance and overhead proven both analytically and using simulations; it is a scalable peer-to-peer protocol, i.e., all entities operate similarly, with fixed load to each peer, with minor differences between peers (and over time). Yet, DURP also ensures strong, *proven* anonymity properties, including *both* unobservability and source-anonymity (against malicious destination).

DURP is decentralized, without any single point of failure; this is important for reliability, security (e.g., against "legal attacks"), and for load balancing (required for efficient scalability). Unlike previous decentralized solutions (see below), DURP ensures unobservability against strong attackers and anonymity against the destination, while maintaining reasonable efficiency. DURP is also easy to implement and deploy, reusing existing anonymity protocols modularly.

B. Fully decentralized design

A basic threat to anonymity is corruption or coercion of the anonymizing servers, e.g., by using legal means to force a server to expose identities. Hence, for good, robust anonymity, decentralized designs are preferred, ensuring that such efforts to deanonymize require corruption or coercion of many servers. This is a basic design principle of Tor [1] and other mix-networks; in particular, Tor features thousands of relay nodes.

However, in Tor (and similar networks), complete decentralization is hard to achieve, since the network serves many end users, and each of them consumes part of the bandwidth of the servers. Therefore, it is highly desirable to use maximal bandwidth of each server. Indeed, in Tor, larger relays (with larger bandwidth) are chosen more often [6]; the differences in bandwidths are so extreme [7], that much of the traffic flows through a relatively modest number of (maybe few dozens) large relays. This makes some relays more lucrative targets, and motivates entities interested in deanonymization to operate such popular relays; the costs are not very high, certainly compared to the budget of some entities interested in deanonymization (e.g., security agencies).

In contrast, in DURP, our goal of unobservability implies that every node is both a producer of information and a providing of relaying services, and we do not need large nodes. Hence, DURP is completely decentralized, with equal load on each node, which remains fixed even as the network grows; this also ensures high reliability and good load balancing, and thereby efficient scalability.

C. DURP Overview

DURP combines two well-known and intuitive techniques for anonymity: *mixing* [8] and *constant sending rate*. Specifically, DURP maintains constant sending rate from the application level to the protocol level, to hide the communication from eavesdroppers and malicious peers (i.e., ensure unobservability). DURP also uses additional dummy traffic in the protocol level to increase the anonymity against global eavesdropper destination.

DURP uses mixing to ensure anonymity against the destination. Specifically, DURP uses, in a modular fashion, low-latency mixing protocols, such as Crowds [9] or Onion-Routing (OR) [10]. Indeed, DURP can be viewed as a framework that adds unobservability against malicious peers and global eavesdropper to anonymity protocols. This is done while maintaining reasonable overhead and load balancing.

DURP modular design is simple, and it can be deployed easily, even on a subset of existing Tor nodes (as an overlay network providing stronger anonymity and unobservability properties). The DURP framework dispatches the anonymity protocol in synchronous rounds, and maintain a constant rate sending from the application level to the protocol level by adding a message from the application each round with probability of $1/\rho$ for some $\rho > 1$.

D. DURP Properties

DURP has higher latency and communication (cf. to Crowds and Onion Routing), but this extra overhead is not excessive, and may be acceptable to many applications which require strong anonymity. To prevent load and keep the communication cost constant, the outgoing traffic and the average incoming traffic of every participant are constant (per round). The incoming traffic to each DURP relay, is bounded by $\frac{\log n}{\log \log n}(1+o(1))$ with probability $1-o(1)$ (see Section IV-E). In Section IV we present thorough analysis of other network properties, including the maximal reporting throughput as a function of the communication cost, the distribution of the buffers' sizes, the average latency and more. Using this analysis, it is possible to check whether it is worth to pay the communication cost for getting the strong anonymity properties, or to configure DURP by the communication restrictions. The known low overhead for the reporters makes DURP practical solution.

DURP ensures the following anonymity properties:

1) *Provable unobservability* against any probabilistic polynomial time ($\mathcal{PPT}$) global eavesdropper attacker that further controls any combination of malicious peers (formal theorem and proof in Section VI).
2) *Provable strong sender anonymity* against any probabilistic polynomial time ($\mathcal{PPT}$) attacker that controls the destination (Section V).

3) *Probable and Possible Innocence*. We rely on the definitions of Reiter and Rubin [9], and offered also quantification of the Probable and Possible innocence properties. DURP's (and in particular its optimized version) high degrees of Probable and Possible innocence against global eavesdropper destination are validated experimentally (Sections VII and VIII).

E. Related Works

The amount of research of anonymous communication, even only of decentralized designs, is too large for us to properly review here. Hence, we only discuss here briefly the most relevant and prominent works, and refer the reader to [11] for a survey covering many other works.

1) Protocols that ensures unobservability: Our design is reminiscent of PipeNet [12]. However, notice that [12] provides only very short, abstract description; there are details missing in the design itself, and there are neither formal proof for its anonymity properties nor performance evaluation - these are part of our main contributions. The analysis results helped us to detect some problems in DURP (Section VII) and to offer optimized version (Section VIII). Example to the abstraction of PipeNet is the fact that a user can launch a DoS attack (and cause a total stop of the system) by just stop sending.

Our security requirements are closest to these of Chaum's seminal DC-net design [13], which also ensures unobservability and source-anonymity. However, DC-net's overhead is too high for most applications; the minimal ratio between real messages bits and dummy ones is $\mathcal{O}(n)$, when n is the number of the peers. Furthermore, additional bandwidth is needed to deal with collisions [14]. In DURP, the minimal ratio is a constant (see Section IV).

Another approach, also initiated by Chaum, is using *mixes* [8]. However, mix-based systems that ensure anonymity and unobservability, like Mixmaster [15] and Miminion [16], have very high latency; it might take few hours for a mail to reach its destination.

Mixed-based solutions like [17] [18] [19] separate between the clients and the mixes, and therefore are less scalable and more vulnerable to legal attacks. Moreover, the first mix knows the source of the messages that it receives, and together with the destination, they can detect whether any specific message is real by forwarding only this message, and sending dummy messages instead of the other messages [20]. See further attacks and analysis in [21], [22].

Another difference between DURP and mixes based solutions [23], is the fact that the load on the participant is not the same, or is not uniform over time (i.e., mix might send at once a lot of messages or wait until the pool is get filled).

Other protocols may provide unobservability, but only against an eavesdropping attacker, and fail when the attacker also controls some of the nodes. Two examples are Tarzan [24] and Herbivore [25]. Both the protocols ensures unobservability against eavesdroppers, but many malicious peers can break the unobservability. Finally, *'buses-based'* approach of Beimel and Dolev [26], achieves unobservability - but *not anonymity*.

2) Protocols that ensures anonymity: Most of the recent research in the anonymity field is about low-latency protocols, mainly for web-surfing. Well-known protocols such as Crowds [9], Onion Routing [10], Tor [1], hordes [27], MorphMix [28], protocols for k-anonymity [29] [30] and others [31] [32] [33] [34] focus on anonymity but do not provide unobservability. All these protocols, provide some anonymity and are efficient. The cost is losing unobservability even against local eavesdropper (without requiring malicious peers) or weaker anonymity properties. Additionally, without adding delays, these low latency protocols leak information even against the destination [35]. Dummy traffic is known solution [20] [36] and is also necessary in order to hide whether one communicates.

Organization. In Section 2, we present the model and how we evaluate DURP's anonymity and unobservability. In Section 3 we introduce DURP. In Section 4 we prove DURP with OR or Crowds as a base protocol ensures sender anonymity against malicious destination. In Section 5, we formally prove DURP ensures unobservability against any combination of malicious peers and eavesdroppers. The proof is by reduction to the security of the public key encryption scheme used by DURP. In section 6 we empirically shows that DURP ensures also anonymity against global eavesdropper destination, and present an optimized version of the protocol. In the last section we conclude our work and discuss future wsork.

II. Model and Requirements

A. Communication and Setup Model

In this work we assume synchronous system, i.e. all peers operate in synchronous rounds, and all the messages sent in one round, are received in the next round. We assume private channels between each pair of participants, e.g., using symmetric cryptography. We also assume that all the participants have the public key of the destination. Another assumption we make is that all the messages that need to be sent are of the same length; obviously, message of different length might be divided or padded.

We consider a network of n participants $\{p_1, ..., p_n\}$. The first $n-1$ participants are called *peers*, and participant p_n, denoted also as d, is the (only) *destination*. Every pair of participants can communicate. The peers want to send *anonymous* reports to d, such that *only the destination* can observe whether a report was sent or not.

B. Anonymity and Unobservability Definitions

Precise definition of anonymity properties is challenging. We extend the elegant definitions of [37], which capture many variants of anonymity, to allow also active attackers. In the following subsection, we sketch our definition, for the two most relevant anonymity notions: *sender anonymity* and *unobservability*; additional notions and more details are available in [38].

*Comp-***N***-Anonymity:* To define Comp-**N**-Anonymity (for different anonymity notions **N** [37]) against different attackers, we define an experiment (see Alg 1). In the experiment, the attacker chooses and adaptively manages two scenarios, but only one is simulated. The different anonymity notions and the attacker's capability define different restrictions on the chosen scenarios.

The scenarios are modeled by messages matrices $M^{(0)}$, $M^{(1)}$. Messages have constant length, and are taken from $V = \{0,1\}^l$. $M_{i,j} \in V^*$ is the multiset of messages that p_i wants to send to p_j. The restrictions on the matrices, are modeled as relations on these matrices.

The experiment has four parameters in addition to the security parameter k: the protocol π, the number of the participants n, the $\mathcal{PPT}$ attacker $\mathcal{A}$ and the capability of the attacker *Cap*.

The protocol π has two $\mathcal{PPT}$ methods: *Setup*$(1^k, i, n)$ to get the initial state of the i-th participant and usually to generate and distribute identities and keys, and the *Simulate* method that simulates participant by the protocol π according to its state, the messages it receives, and its application level.

The capability is defined by a tuple *Cap*=$(\overline{H}, EV, MD) \in \mathcal{P}([n])^2 \times \{0,1\}$, such that *Cap*$[\overline{H}]$ is the malicious participant (controlled by the attacker) set, *Cap*$[EV]$ is the eavesdropped participants set, and *Cap*$[MD]$ defines whether the attacker is modeled also as malicious destination. By default *Cap*$[MD]$=0, such that the capability is defined also as an element in $\mathcal{P}([n])^2$.

When **N=UO** (unobservability), we do not restrict the communication between honest peers in both the scenarios, and forbid communication between honest and malicious peers. The attacker can send messages to the honest peers from the malicious peers, but in both the scenarios, honest peers do not want to send messages to malicious peers.

The relevant relation is denoted by $R^H_{\mathbf{UO}}$.

We prove that DURP is Comp-**UO**-anonymous against any combination of eavesdroppers and malicious peers. We consider attackers with capability *Cap*=$(S \subset [n-1], [n], 0)$.

The restriction on the scenarios, when **N=SA** (sender anonymity) against one malicious destination (modeled by capability *Cap*=$(\{n\}, \emptyset, 1)$), is that the traffic from the honest peers to destination in both the scenarios, is some constant permutation τ, i.e., if in the first scenario p_i wants to send the messages in $M^{(0)}_{i,n}$ to the destination, then in the second scenario, $p_{\tau(i)}$ must send the same messages.

In our case, where all the messages are for one destination, we can model the restriction on the scenarios matrices by the demand: $Rows(M^{(0)}) = \tau(Rows(M^{(1)}))$ (when the rows that represent the application level of malicious participants, must be empty).

The relevant relation is denoted by $R^{H,\tau}_{\mathbf{SA}}$.

We prove that DURP is Comp-**SA**-anonymous against malicious destination (the strongest anonymity notion against malicious destination).

We bring here shortened versions of the relevant definitions from [39]:

Definition 1: The **Comp-N-advantage** of an attacker $\mathcal{A}$ that runs with k as a parameter, is defined as:

$$Adv^{Comp-\mathbf{N}}_{\pi,n,\mathcal{A},Cap}(k) = |Pr[Expt^{Comp-\mathbf{N}-1}_{\pi,n,\mathcal{A},Cap}(k) = 1] - Pr[Expt^{Comp-\mathbf{N}-0}_{\pi,n,\mathcal{A},Cap}(k) = 1]|$$

Definition 2: Protocol π is Comp-**N**-anonymous, when $\mathbf{N} \in \{SA, UO\}$, against attackers with capability $Cap \in P([n])^2 \times \{0,1\}$, if for all $\mathcal{PPT}$ algorithms, $\mathcal{A}$, there exists a negligible function $negl$ such that,

$$Adv^{Comp-\mathbf{N}}_{\pi,n,\mathcal{A},Cap}(k) \leq negl(k)$$

Algorithm 1 $Expt^{\mathbf{Comp-N-b}}_{\pi,n,\mathcal{A},Cap}(k)$. From [39]

for $i = 1$ **to** n **do** $S_i \leftarrow \pi.Setup(1^k, i, n)$ **end for**
$EV = Cap[\overline{H}] \cup Cap[\text{EV}]$
$H = [n] - Cap[\overline{H}]$
$< S_{\mathcal{A}}, 1^{rounds} > \leftarrow \mathcal{A}.Initialize(1^k, \{S_i\}_{i \in Cap[\overline{H}]})$
for $t = 1$ to $rounds$ **do**
  $< S_{\mathcal{A}}, M^{(0)}, M^{(1)} > \leftarrow \mathcal{A}.InsertMessages(1^k, S_{\mathcal{A}})$
  if $(M^{(0)}, M^{(1)}) \notin R^H_{\mathbf{N}}$ **then**
    return 0
  end if
  for all $i \in H$ **do**
    $< S_i, \{C_{i,j,t}\}^n_{j=1} > \leftarrow \pi.Simulate(1^k, S_i, \{C_{j,i,t-1}\}^n_{j=1}, \{m^{(b)}_{i,j}\}^n_{j=1})$
  end for
  $< S_{\mathcal{A}}, \{C_{i,j,t}\}_{i \in Cap[\overline{H}]\ 1 \leq j \leq n} > \leftarrow \mathcal{A}.Simulate(1^k, S_{\mathcal{A}}, \{C_{i,j,t-1}\}_{i \vee j \in EV})$
  $< S_{\mathcal{A}}, b' > \leftarrow A.GuessB(1^k, S_{\mathcal{A}})$
  if $b' \neq NULL$ **return** b' **end if**
end for
return 0

III. Decentralized Unobservable Reporting Protocol (DURP$^{\Pi}$)

In this section we present the Decentralized Unobservable Reporting Protocol (DURP$^{\Pi}$). DURP is modular; the Π parameter of DURP$^{\Pi}$ refers to a 'base' anonymous-communication protocol, specifically, either Onion Routing (OR) [10] or Crowds [9]. In the following sections, we prove that, with both 'base protocols', DURP ensures two strong anonymity properties: (1) Comp-**UO**-anonymity against any combination of eavesdroppers and other (active) malicious participants, and (2) Comp-**SA**-anonymity against the destination.

A. Operation of DURP

A DURP's peer (potential sender) is initialized with four parameters: (1) the security parameter k, (2) the destination's public key pk_d, (3) the average number of rounds to add a message from the application ρ and (4) the expected average hops-counter for a message H.

Every peer running DURP maintains two data-structures: *Set* and *DSet*. *Set* contains messages to be sent to other peers, and *DSet* contains messages to be sent directly to the destination. The messages in *Set* are either received from other peers (messages that the peer has to relay), or inserted to *Set* from the application by the protocol (and their originator is the peer).

Every round, DURP$^{\Pi}$ sends exactly one message from *Set* (or a special *dummy* message, when *Set* is empty) to another peer; The creation and the process of the messages is done

by the base protocol Π (e.g., in Π=OR, when a message is created, the plaintext is wrapped as an onion [10]).

It may appear, that since the step above ensures constant sending rate by each peer, then unobservability is already satisfied. However, this is incorrect, as we explain in Section III-C.

Hence, DURP also maintains a fixed sending rate from the application, at each peer, to the destination. Namely, on every iteration, each peer adds a message from the application level to its *Set* with probability $\frac{1}{\rho}$. Notice that the application must provide a message; obviously, some signaling message can be sent to signal lack of real content. Such signal message should not be confused with the *dummy* message sent by DURP itself when *Set* is empty (see above). New messages from the application and dummy messages, are encrypted by DURP's public key encryption scheme, denoted by $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, using the destination's public key. DURP requires CPA secure encryption scheme [40].

A received message is added to *DSet* if its next (and last) destination is d; otherwise, it is added to *Set*. Every iteration, all the messages in $DSet$ are sent to the destination.

The pseudocode of DURP$^{\Pi}$ appears in Alg 2.

Algorithm 2 DURP$^{\Pi}$. The Initialize, CreateMessage and ReceiveMessage methods are done by Π. In line 12 (c, p) is chosen uniformly.

1: **Initialization(**k**,** pk_d**,** ρ**,** H**):**
2: $Set = DSet = \emptyset$
3: Π.**Initialize**(k, H)

4: **Every round do:**
5: **With** probability $\frac{1}{\rho}$ **do**
6: Get message m from the application
7: *Set*.add(Π.**CreateMessage**($\mathcal{E}_{pk_d}(m), d$))
8: **end do**
9: **if** *Set*.isEmpty() **then**
10: $(c, p) \leftarrow$ (Π.**CreateMessage**($\mathcal{E}_{pk_d}(dummy), d$))
11: **else**
12: $(c, p) \xleftarrow{r}$ *Set*
13: Send c to p
14: Send all the messages in *DSet* to d

15: **OnReceivingMessage(**c**):**
16: $(c, p) \leftarrow \Pi$.**ReceiveMessage**(c)
17: **if** $p == d$ **then**
18: *DSet*.add((c, p))
19: **else**
20: *Set*.add((c, p))

B. Onion Routing and Crowds as Base Protocols (Π)

We define the routing procedure of the protocols such that the *exit-peer*, the peer that forwards the message to the destination, will be chosen independently of the originator and other intermediate peers. Both protocols ensure sender anonymity against the destination, but not unobservability (even against very weak attacker like local eavesdropper), because they do not use cover traffic.

1) Onion-Routing (OR): The OR protocol [10] relies on public key encryption, to enforce routing of a message. The sender chooses a path of l intermediate peers. Then, the sender encrypts the message with the public keys of all the peers along the path, from the last to the first, creating an 'onion' around the message. When an intermediate peer receives the message, it removes the outer encryption layer, and forwards the plaintext (onion with one layer removed) to the next peer in the path. See pseudo code in Alg 3.

We now explain how to choose the routing path for ensuring Comp-**SA**-anonymity against malicious destination. The *ChoosePath(l)* method (Alg 3, line 6) works as follows: The first different $l-1$ peers are chosen uniformly, and then the exit-peer is chosen uniformly among all the peers (and might be one of the peers that are already in the path). The hops-counter of every message when it reaches the destination is $H = l + 1$.

Algorithm 3 The methods for the base protocol Π = OR. $l > 1$ is the number of the layers (intermediate peers in a path) in the protocol.

1: OR.**Initialize(**k**,**H**):**
2: $(pk, sk) \xleftarrow{\$} \mathcal{K}(k)$
3: Publish pk
4: $l \leftarrow H - 1$

5: OR.**CreateMessage(**m, d**):**
6: $(p_1, p_2, ..., p_l) \xleftarrow{r}$ ChoosePath(l)
7: $(c, p) \leftarrow (m, d)$
8: **for** $i = l$ **to** 1 **do**
9: $(c, p) \leftarrow (\mathcal{E}_{pk_i}(c, p), p_i)$
10: **end for**
11: **return** (c, p)

12: OR.**ReceiveMessage(**c**):**
13: $(c, p) \leftarrow \mathcal{D}_{sk}(c)$
14: **return** (c, p)

2) Crowds: In Crowds [9], the sender selects a random peer and sends the message to it. The intermediate peer forwards the message to another randomly chosen intermediate peer with probability P_f, or, with probability $1 - P_f$, sends the message to the destination.

We slightly change Crowds by adding one step before sending to the destination, i.e., referring to the (randomly choosen) peer that sends destination as the *exit-peer*, as follows (and see Alg 4):

1) The originator of the message sends the message to a random intermediate peer.
2) Every peer that receives a message as an intermediate peer, sends it to another intermediate peer with probability of P_f. With probability of $1 - P_f$, the peer chooses a random *exit-peer* among all the peers (including itself), and forwards the message with indication that the receiver is the exit-peer of the message.
3) The exit-peer sends the message to the destination.

The transmission between the peers is exactly like the complete transmission in original Crowds, hence the average number of hops (between peers) is $1 + \frac{1}{1-P_f}$. The exit-peer

always sends directly to the destination, hence the average number of hops in our modified-Crowds is simply $H = 2 + \frac{1}{1-P_f}$.

Algorithm 4 The methods for the base protocol Π = Crowds. P_f is the probability to forward the message. With probability of $1 - P_f$ the peer chooses an exit peer.

```
1: Crowds.Initialize(k,H):
2:     P_f ← (H−2)/(H−3)

3: Crowds.CreateMessage(m, d):
4:     p ←r ChooseNextPeer()
5:     c ← (m, d, False)
6:     return (c, p)

7: Crowds.ReceiveMessage(c = (m, d, isExitPeer)):
8:     if isExitPeer then
9:         (c, p) ← (m, d)
10:    else
11:        x ←r [0, 1]
12:        if x ≤ P_f then
13:            p ←r ChooseNextPeer()
14:            c ← (m, d, False)
15:        else
16:            p ←r ChooseExitPeer()
17:            c ← (m, d, True)
18:    return (c, p)
```

C. Constant Sending Rate $\not\Longrightarrow$ Unobservability

To conclude this section, we explain why unobservability is not necessarily guaranteed, merely by ensuring constant sending rate by the peers. For example, consider a synchronized version of Onion-Routing or Crowds, where, in addition, every peer sends one message per iteration. If a peer has no message to send at a round (from application or other peer), then it sends a dummy message.

Active attacker can learn whether a peer has messages to send, by making sure that the peer's queue is always not empty. In such a case, if the peer does not have messages from the application, it will always forward the attacker's messages, possibly to other collaborating participants. If the peer has some messages to send, there will be some delay in the delivery of the attacker's messages. Attacker can detect this and thereby know whether the peer is sending application messages to the destination.

DURP prevents such detection, by ensuring constant sending rate from the application in each peer (fixed probability of $\frac{1}{\rho}$ per round), and since the protocol handles all messages similarly, independently of their contents (dummy or real).

IV. DURP's Network Properties and Performance

A. Notations

We next present few notations used in the analysis.

Iterations-counter of a message is the counter of iterations from the first iteration of the message in the network. This counter increases by one every iteration and stops when the message reaches its destination.

IC is the average *iterations-counter* of messages that reached their destinations.

The *hops-counter* of a message m, is the number of 'send' (Alg 2, lines 13 and 14) events m passes.

We denote by $|\mathrm{C}reateMessage_{rnd}|$, the average number of 'CreateMessage' events during the round rnd. Similarly, we denote $|\mathrm{S}end_{rnd}|$ the average number of 'Send' events. The average hops-counter is denoted H, and defined as follows:

$$H \equiv \lim_{t\to\infty} \frac{\sum_{rnd=1}^{t} |\mathrm{S}end_{rnd}|}{\sum_{rnd=1}^{t} |\mathrm{C}reateMessage_{rnd}|}$$

ρ is the sending rate, i.e., every iteration, every peer has to add a message from the application with probability $\frac{1}{\rho}$.

N is the number of the peers in the network; N_i is the average number of peers whose *Set*s are of size i, when 'average' is over the entire execution. $N = \sum N_i$.

s_i is the average fraction of the *Set*s of size i, i.e., $s_i = \frac{N_i}{N}$, hence $\{s_i\}_{i=0}^{\infty}$ is the distribution of the *Set*s' sizes.

S is the average number of messages in the network's *Set*s. We denote by $|\mathrm{D}elivered_{rnd}|$ the number of messages that were added to *DSet* in round rnd: $S = \lim_{t\to\infty} \sum_{rnd=1}^{t} |\mathrm{C}reateMessage_{rnd}| - |\mathrm{D}elivered_{rnd}|$.

B. Network Properties

We now provide proof sketches for some network properties of DURP; the properties are affected by general attributes like H, and therefore the proofs are relevant for different base protocols.

A network running DURP$^{\Pi}$ is **stable**, if S is bounded and converging; the following lemma shows that the network is stable, provided that $\rho > H - 1$; we assume this holds in the entire analysis. The proof of this lemma appears only in the technical report [39], for lack of space.

Lemma 3: A network under DURP$^{\Pi}$ is *stable* when $\rho > H - 1$.

Lemma 4: The mean number of outgoing messages (messages that arrive to the destination) per iteration equals to the mean number of new messages per iteration, and both are $\frac{N}{H-1}$.

Proof: (sketch) The mean number of outgoing messages is the same as the mean of outgoing messages that move from *Set* to *DSet*.

If the mean of the new messages is greater, then S won't converge. On the other hand, the mean of the new messages cannot be less than the outgoing messages, because every outgoing message was created sometime.

The average number of outgoing messages is $\frac{N}{H-1}$, because the network is stable (Lemma 3), and it takes on average $H-1$ 'send' events (from *Set*) to reach some *DSet*, and each round exactly N 'send' events occur. ■

Lemma 5: The proportion between application messages and dummy messages is $\frac{H-1}{\rho-H+1}$.

Proof: Over t iterations, N peers send messages from *Set*s tN times. Each message is sent, on average, $H-1$ times until

it reaches some *DSet*. The average number of new application messages during t iterations is: $\frac{tN}{\rho}$. Because the network is stable, the total number of 'send' events of application messages from *Set*s is $\frac{(H-1)tN}{\rho}$. Hence, the dummy messages take $tN - \frac{(H-1)tN}{\rho}$ 'send' events (again, sending from *Set*s). Because the mean of dummy message's *hops-counter* is the same as application message, the average number of dummy messages is $(tN - \frac{(H-1)tN}{\rho})\frac{1}{H-1}$. Division between the values gives the above result. ■

Lemma 6: $s_0 = \frac{\rho-H+1}{(H-1)(\rho-1)}$.

Proof: According to the previous lemma, the average number of the dummy messages added every iteration to *Set*s is $\frac{N}{\rho} \cdot \frac{\rho-H+1}{H-1}$. By the protocol, dummy messages are created only in empty *Set*s. Before the creation of dummy message, there is a chance of $\frac{1}{\rho}$ that a message will be inserted to *Set* from the application instead. So, on average, only $1 - \frac{1}{\rho}$ of the peers with an empty *Set*, create a dummy message. This gives us the next equation: $\frac{N(\rho-H+1)}{\rho(H-1)} = s_0 N(1 - \frac{1}{\rho})$. And so, we get that $s_0 = \frac{\rho-H+1}{(H-1)(\rho-1)}$. ■

Lemma 7: $S = \sum_{i=1}^{\infty} N_i \cdot i = N \cdot \sum_{i=1}^{\infty} s_i \cdot i$.

Proof: Directly from the definitions of N_i and s_i. ■

Lemma 8: $IC = \frac{S(H-1)}{N} + 2$.

Proof: By Little law, the number of messages in the network's *Set*s, S, equals to the incoming messages rate, $\frac{N}{H-1}$, times the average number of iterations a message spends in the network's *Set*s. Because we also count the last two iteration (when the message reaches some *DSet* and the following iteration when it reaches its destination), we get the next equation: $S = \frac{N}{H-1}(IC - 2)$. ■

C. Calculating the distribution of Sets' sizes in the network

There are two methods to find $\{s_i\}_{i=0}^{\infty}$: simulations or analysis (queuing theory). Due to space limitations, we present here less efficient algorithm, but an algorithm that can be used also for the optimized version of DURP (Section VIII). The more efficient algorithm appears in [39].

The algorithm we present has excellent fit to the simulations (see Section IV-D). The idea of the algorithm is to look only on one *Set* and analyze the probabilities of changes in its size, and it is based on two simplifying assumptions.

1) While analyzing the number of messages that might arrive to *Set* we take the mean of the messages that were sent in some iteration and are will reach some *Set* in that iteration (messages that will not reach their exit-peer in that iteration). Of course we consider only messages that are sent from the other $(N-1)$ peers.
   We notate the mean of messages that might arrive to a peer's *Set* in an iteration by D, and from Lemma 4, we get that $D = (N-1)(1 - \frac{1}{H})$.
2) We calculate s_i only for $i < n$, for a parameter n.

The algorithm builds a matrix M in the following way: $M_{i,j}$ is the probability that *Set* of size j will be of size i in the next iteration. Ideally, M is an infinite matrix, for our algorithm we choose its size, n. The accuracy increases as n increases.

The function **ProbToGetK**(int k) (Alg 5) returns the probability for a peer to get k messages in an iteration.

Algorithm 5 The **ProbToGetK**(int k) function.

return $\binom{D}{k}\left(\frac{1}{N-1}\right)^k\left(1-\frac{1}{N-1}\right)^{D-k}$

Because D is a rational number, we should calculate the factorials in the binomial coefficient with Gamma function.

The function **ProbToBeChangedInK**(int k) (Alg 6) returns the probability for non zero *Set* size to be changed in k between two consecutive iterations. The minimal value for k that might return nonzero value is -1 (when both no message was sent to the peer and no message was inserted from the application); the corresponding maximal value, should be $N-1$, but due to our lenient assumption, we return 0 for every value that is greater than D.

Algorithm 6 The **ProbToBeChangedInK**(int k) function.

if $k < -1$ OR $k > D$ **then**
  return 0.
else if $k = -1$ **then**
  return $(1 - \frac{1}{\rho})$**ProbToGetK**(0).
else
  return $\frac{1}{\rho}$**ProbToGetK**$(k)+(1 - \frac{1}{\rho})$**ProbToGetK**$(k+1)$.
end if

Due to the range of the possible changes in *Set*'s size, M is almost lower triangular matrix; above the diagonal, only the values on the upper secondary diagonal are not zeros. Therefore, for calculating M, we should run on the non-zero cells of each column (starting from the first) and fill them as in the **BuildProbsMatrix** function (Alg 7).

Algorithm 7 The **BuildProbsMatrix**(int n) function. n is the accuracy parameter.

Initialize zeros matrix $M^{n \times n}$
for $i = 0$ to $n-1$ **do**
  $M_{i,0}$ =**ProbToGetK**(i)
  $M_{i,1}$ =**ProbToBeChangedInK**$(i-1)$
end for
for $j = 2$ to $n-1$ **do**
  for $i = j-1$ to $n-1$ **do**
    $M_{i,j} = M_{i-1,j-1}$
  end for
end for
return M

For $M_{i,j}$, $i \geq 1, j \geq 2$, **BuildProbsMatrix** avoids unnecessary calculations: while $j > 2$, $M_{i,j} = M_{i-1,j-1}$ because $i - j = (i-1) - (j-1)$.

Given such a matrix, we define the procedure **SetsSizesByMatrix** that receives a square matrix and outputs the *Set*s' sizes distribution applied by that matrix. Given that procedure, the *Set*s' sizes distribution of DURP is calculated with accuracy parameter n by **SetsSizesByMatrix**(**BuildProbsMatrix**(n)).

The input matrix M for **SetsSizesByMatrix**, defines the connections between all the s_i values. We need to solve the linear equations system: $M\overrightarrow{s} = \overrightarrow{s}$. We want to find the eigenvector for the eigenvalue $\lambda = 1$. If we look at the infinite transitions probabilities matrix, it is easy to see that it has eigenvalue $\lambda = 1$, because the sum of every column is 1. We use finite square matrix of size n, and if n is high enough, this matrix also has self value 1 or self value λ such that $|1 - \lambda|$ is very small. So, for getting the sizes distribution, the **SetsSizesByMatrix** (Alg 8) algorithm finds the eigenvector that corresponds to λ, and normalizes it such that the sum of its elements will be 1. The normalized vector is the wanted distribution.

Algorithm 8 SetsSizesByMatrix($M^{n\times n}$). Algorithm for getting the *Set*s' sizes distribution by a transitions matrix

$\lambda \leftarrow M$'s eigenvalue, l, that minimize $|l - 1|$.
$\overrightarrow{s} \leftarrow$ Eigenvector of λ
$sum \leftarrow$ Sum of $\overrightarrow{s}$ elements.
for $i = 0$ to $n - 1$ **do**
  $s_i = s_i/sum$
end for
return $\overrightarrow{s}$

D. Simulation Results

We ran simulations of DURP^{Crowds} and DURP^{OR}, with different H, ρ and N values. The very high correlation between the actual results and the above analysis, shows that the algorithm for getting the distribution of the Sets' sizes (Section IV-C) gives excellent results, and that using the algorithm and the lemmas, it is possible to predict network properties like IC, given DURP's configuration parameters (ρ, H).

We present the results of simulations for two different configurations. For every configuration, we compare the simulation results of both the base protocols that have identical expected results while they have the same *hops-counter* mean, H, to the expected results. We ran every simulation 8 times for 10,000,000 iterations, and after 1,000,000 iterations that started from network of peers with empty *Set*s.

The expected results are calculated as follows: The proportion between the application messages (reals) to the protocol dummies is calculated by Lemma 5. $N_0 = N \cdot s_0$ is calculated by Lemma 6. To calculate S by Lemma 7, we used **SetsSizesByMatrix**(**BuildProbsMatrix**(200)) (Figures 8 and 7) to get s_i for $i = 0, ..., 199$. In the end, we calculate IC by Lemma 8.

$MaxIC$ is the maximal value of *iterations-counter* for message in the simulation. Comparison of the simulation results and the expected results, appears in Tables I and II. The confidence interval is calculated by confidence level of 99.9%.

The *Set*s' sizes distribution by the algorithms, and the distribution in both the simulation for DURP^{OR} and DURP^{Crowds} is visually identical. We present the distributions of these simulations in Figure 3 in Section VIII when we compare DURP to its optimized version.

In the DURP^{Crowds} simulations, the confidence interval of the average *hops-counter* was less than 0.001 around the expected value. The average maximal *hops-counter* was 53.25 in the first case, and 72 in the second.

TABLE I. $N = 100, H = 5, \rho = 5$

	Expected results	DURP^{OR}	DURP^{Crowds}
$Real/Dummy$	4	4.00±0.00	4.00±0.00
N_0	6.25	6.25±0.00	6.25±0.00
S	929.92	934.05±0.57	933.74±0.67
IC	39.20	39.36±0.02	39.35±0.03
$MaxIC$	-	1129±62.68	1257.25±116.07

TABLE II. $N = 100, H = 6, \rho = 7$

	Expected results	DURP^{OR}	DURP^{Crowds}
$Real/Dummy$	2.5	2.50±0.00	2.50±0.00
N_0	6.67	6.67±0.00	6.67±0.00
S	832.93	835.71±0.47	835.51±0.41
IC	43.65	43.79±0.02	43.78±0.02
$MaxIC$	-	1063±137.87	1079.88±89.18

E. DURP's Performance

We first analyze the communication cost for DURP's participants. Then we analyze DURP's efficiency: communication overhead and latency. We also compare these properties with the DC-net protocol [13] which is fully decentralized protocol (all participants are equally lucrative targets, e.g., to legal attacks) that achieves the same anonymity goals.

1) DURP's communication cost: Every DURP's reporter, sends one message per round. Additionally, because every round there are $N - 1$ messages that might reach $N - 1$ other reporters (each. and the $N - 1$ potential recipients are different for each message), the probability that a reporter will receive more than $\frac{\log N}{\log \log N}(1 + o(1))$ is bounded from above with probability $1 - o(1)$ [41].

2) DURP's communication overhead: We compare the communication overhead with other protocols that use constant sending rate to ensure unobservability. Therefore, we test the communication by the maximal application messages that might reach the destination, per sending actions. Every DURP's iteration, N sending actions occur. From Lemma 4, and from the proportion between application messages and dummy ones, (Lemma 5), we get that by mean, every iteration $\frac{N}{\rho}$ application messages reach the destination. Therefore, For sending a message anonymously, instead of one sending action, we pay in ρ actions. Intuitively, we expected this result, because we took a message from the application with probability of $\frac{1}{\rho}$.

The ratio of ρ sending actions per real message is the minimal, and is gotten if the application always has something to send. If peers do not have something to send, the actual ratio increases.

In the DC-net protocol [13] the minimal ratio is $\mathcal{O}(n)$.

3) DURP's latency: We measure the latency of DURP by IC, the average *iterations-counter*. Although we have a formula for IC (Lemma 8), we cannot simply use it, as the calculation of S requires running an algorithm. However, the gotten IC value is a constant, and in the simulation results, we showed configurations where IC is less than $10 \cdot H$, when H is the expected value for IC in the base protocol (in base

protocols like OR and Crowds, a message is sent every iteration until it reaches its destination).

In the DC-net protocol, if we ignore the overhead of collisions, the latency is minimal.

4) Overhead vs. latency, and DURP's parameters: We showed that the communication overhead is determined by the ρ parameter. However, we can increase ρ and pay in more dummy messages (Lemma 5), and decrease the latency (IC). Another way to decrease the latency, is to decrease H (for example, using lower P_f in Crowds, or fewer encryption layers in OR). However, this might hurt the anonymity against coalition of malicious destination and peers.

In the DC-net protocol, we pay for the low latency in significant communication overhead. Herbivore [25], a DC-net based protocol, pays for the scalability and the improvement in the communication overhead in latency and in the loss of the unobservability against strong attackers.

V. Anonymity Proof

Theorem 9: DURP^{OR} and DURP^{Crowds} over $2 < n < l(k)$ participants ($l(\cdot)$ some polynomial) are Comp-**SA**-anonymous against malicious destination.

Proof: We prove that for $\Pi \in \{DURP^{OR}, DURP^{Crowds}\}$ and n as in the theorem, for all $\mathcal{PPT}$ $\mathcal{A}$ there exists a negligible function $negl$ such that $Adv^{Comp-\mathbf{SA}}_{\Pi,n,\mathcal{A},(\{n\},\emptyset,1)}(k) < negl(k)$.

We model the information $\mathcal{A}$ receives during the Comp-**SA**-b experiment as a sequence of tuples $(m,t,p) \in V \times \mathbb{N} \times [n-1]$, such that each tuple represents that a message m was received in iteration t from peer p.

We prove that for every two scenarios modeled by sequences of messages matrices: $\{M_i^{(0)}\}_{i=1}^s$ and $\{M_i^{(1)}\}_{i=1}^s$ such that for every $1 \le i \le s$, $(M_i^{(0)}, M_i^{(1)}) \in R^{H,\tau}_{\mathbf{SA}}$ (for some permutation τ), the probability to get the same information $\{m_i,t_i,p_i\}_{i=1}^t \in \{V \times \mathbb{N} \times [n-1]\}^*$ is identical in both the scenarios.

We first prove the following lemma:

Lemma 10: Given a scenario, modeled by a sequence of matrices $\{M_i^{(b)}\}_{i=1}^s$, for every two information sequences $\{m_i,t_i,p_i\}_{i=1}^t$ and $\{m_i,t_i,q_i\}_{i=1}^t$, the probability of the malicious destination to receive either of them, in $Expt^{Comp-\mathbf{SA}-b}_{\Pi,n,\mathcal{A},(\{n\},\emptyset,1)}(k)$, is identical.

Proof: Consider a run of the protocol for the scenario, where the destination receives the information $\{m_i,t_i,p_i\}_{i=1}^t$. Replacing only the *exit-peer* of every message m_i from p_i to q_i produces a run where the destination receives information $\{m_i,t_i,q_i\}_{i=1}^t$. The same holds also in the opposite direction. The probability of every such two runs (identical in all the random choices and different only in the exit peers) is identical. That is because the exit-peer of every message is chosen independently of any protocol or network parameter, and independently of the scenario (actually, the probability for any sequence $\{p_i\}_{i=1}^t$ is $(n-1)^{-t}$). The exit-peer choice does not affect the parameter t_i: from the iteration when the message is sent to its exit-peer, it takes only additional iteration until the message reaches the destination. ■

From the above lemma, it is enough to prove that the probability to get any information $\{m_i,t_i\}_{i=1}^t$ (due to the above lemma, we omit the p_i values) is identical in both the scenarios. But, because for every $1 \le i \le s$, $(M_i^{(0)}, M_i^{(1)}) \in R^{H,\tau}_{\mathbf{SA}}$, the application level of every peer p_i in the first scenario, is the same as the application level of peer $p_{\tau(i)}$ in the second scenario. The scenarios, are different only in the names of the peers. Therefore, the probability to get some information $\{m_i,t_i\}_{i=1}^t$, is identical in both the scenarios. ■

VI. Unobservability Proof

We prove that DURP^{OR} and DURP^{Crowds} ensure Comp-**UO**-anonymity against all the uninvolved parties: eavesdroppers, malicious peers (not the destination) and combinations of them. Because eavesdropping to more peers can only increase the attacker power, it is enough to prove Comp-**UO**-anonymity against any combination of the global eavesdropper and malicious peers. Namely, attackers with capability *Cap*=$(S \subset [n-1], [n])$.

Our proof is based on reducing DURP's Comp-**UO**-anonymity to the CPA-security of the public-key encryption scheme used by DURP. We recall the definition for indistinguishability under chosen-ciphertext attack (IND-CPA) based on right left oracle; we use the following definition, which is equivalent to the one in [40]:

Definition 11: Let $\psi = (\mathcal{K},\mathcal{E},\mathcal{D})$ be an asymmetric encryption scheme, let $\mathcal{A}$ be an algorithm that has access to an oracle and returns a bit. We consider the following experiment:

$Expt^{ind-cpa-b}_{\mathcal{A},\psi}(k)$	$Oracle\ \mathcal{E}_{pk}(\mathbf{LR}(m_0,m_1,b))$
(pk, sk) $\leftarrow \mathcal{K}(k)$	if $\lvert m_0\rvert \neq \lvert m_1\rvert$ then
$b' \leftarrow \mathcal{A}^{\mathcal{E}_{pk}(\mathbf{LR}(\cdot,\cdot,b))}(pk)$	return $\perp$
return b'	$c \leftarrow \mathcal{E}_{pk}(m_b)$
	return c

ψ is CPA secure if for all $\mathcal{PPT}$ algorithms, $\mathcal{A}$, there exists a negligible function, $negl$ such that,

$$|Pr[Expt^{ind-cpa-1}_{\mathcal{A},\psi}(k)=1] - Pr[Expt^{ind-cpa-0}_{\mathcal{A},\psi}(k)=1]| \le negl(k)$$

Before proving Theorem 12, we expand the Comp-**N**-b experiment (see Section II-B, Alg 1) to allow also $b=2$, i.e., $b \in \{0,1,2\}$, and define $\Pi^H_{\mathcal{E},pk}$ for $\Pi \in \{DURP^{OR}, DURP^{Crowds}\}$.

The Comp-N-2 experiment. The Comp-**N**-2 is almost identical to the Comp-**N**-b experiment ($b \in \{0,1\}$, (Alg 1), but ignores the messages between the honest peers and d in both the matrices, and replace them with a constant message μ for d, the (only) valid destination in the protocol. In the Comp-**N**-2 experiment, the honest peers always want to send copies of some constant message, μ.

Formally, the Comp-**N**-2 experiment is identical to the experiment in Alg 1, but with one addition: after verifying that the matrices pair, $(M^{(0)}, M^{(1)})$, is legal, the Comp-**N**-2 experiment creates a new matrix, $M^{(2)}$ by running a transformation f_μ, on $M^{(0)}$. This is done by adding between lines 9 and 10 of the experiment: $M^{(2)} = f_\mu(M^{(0)})$. Notice that because $(M^{(0)}, M^{(1)}) \in R^H_{\mathbf{N}}$, and because honest peers send only messages to d, we could use $M^{(1)}$ as well.

The transformation $f_\mu : \mathcal{M}_{n\times n}(V^*) \rightarrow \mathcal{M}_{n\times n}(V^*)$ is defined as follows:

Given a constant message $\mu \in V$, and the honest participants set, $H = [n] - \overline{H}$, f is defined such that:

$$f_\mu(M)_{i,j} = \begin{cases} \{\mu\} & \text{if } i \in H, j = n \\ M_{i,j} & \text{otherwise} \end{cases}$$

$\Pi^H_{\mathcal{E},pk}$. For $\Pi \in \{DURP^{OR}, DURP^{Crowds}\}$, we define $\Pi^H_{\mathcal{E},pk}$ as follows: $\Pi^H_{\mathcal{E},pk}$ is identical to Π, such that d's public key is pk, and with one change; when a message m is taken from the application of a peer with index in $H \subseteq [n]$ (a message that the peer sends to the destination, or a dummy in case that the application queue is empty), or when Set is empty (see lines 3 and 6 in Alg 2), instead of encrypting it using DURP's public key encryption scheme, the encryption of m is done by $\mathcal{E}(m)$ ($\mathcal{E}(m)$ does not necessarily depend on pk).

Theorem 12: Let ψ be the public-key encryption scheme used by DURP, and $l(\cdot)$ be some polynomial. If ψ is CPA-secure, then $\Pi \in \{DURP^{OR}, DURP^{Crowds}\}$ over $2 < n < l(k)$ participants is Comp-**UO**-anonymous against any combination of global eavesdropper and malicious peers.

Proof: We want to prove that for Π and n as in the theorem and $S \subset [n-1]$, for all $\mathcal{PPT}$ $\mathcal{A}$ there exists a negligible function $negl$ such that:

$$Adv^{Comp-\mathbf{UO}}_{\Pi,n,\mathcal{A},(S,[n])}(k) < negl(k)$$

We prove that given a $\mathcal{PPT}$ $\mathcal{A}$, that has non-negligible advantage for some $2 < n < l(k)$ and $S \subset [n-1]$, we can build $\mathcal{A}'$ that breaks the CPA-security (Definition 11) of ψ, the public-key encryption scheme used in DURP to encrypt the messages for the destination.

We prove that for $a \in \{0,1\}$ the attacker cannot efficiently distinguish whether it runs in the Comp-**UO**-a or in the Comp-**UO**-2 experiments while ψ is CPA-secure. Namely, for all $\mathcal{PPT}$ algorithms, $\mathcal{A}$, there exists a negligible function $negl$ such that,

$$|Pr[Expt^{Comp-\mathbf{UO}-2}_{\Pi,n,\mathcal{A},(S,[n])}(k) = 2] - Pr[Expt^{Comp-\mathbf{UO}-a}_{\Pi,n,\mathcal{A},(S,[n])}(k) = 2]| < negl(k)$$

By the hybrid argument technique (or simply, the triangle inequality), this is enough to prove that an attacker cannot efficiently distinguish whether it runs in the Comp-**UO**-0 or in the Comp-**UO**-1 experiments, while ψ is CPA-secure.

We now present a reduction from the indistinguishability between the Comp-**UO**-a and the Comp-**UO**-2 experiments to the CPA-security of ψ.

Given a $\mathcal{PPT}$ adversary, $\mathcal{A}$, and some $n > 2$ and $S \subset [n-1]$, such that for every negligible function, $negl$:

$$|Pr[Expt^{Comp-\mathbf{UO}-2}_{\Pi,n,\mathcal{A},(S,[n])}(k) = 2] - Pr[Expt^{Comp-\mathbf{UO}-a}_{\Pi,n,\mathcal{A},(S,[n])}(k) = 2]| > negl(k) \quad (1)$$

We build a $\mathcal{PPT}$, $\mathcal{A}'$, that breaks the CPA-security of ψ, the public-key encryption scheme used by DURP, according to Definition 11.

$\mathcal{A}'$ runs in one of the IND-CPA-b experiments without knowing whether b is 0 or 1, with an access to the encryption oracle $Oracle\ \mathcal{E}_{pk}(\mathbf{LR}(\cdot,\cdot,b))$, and it acts as follows:

1) Gets pk from the IND-CPA-b experiment.
2) $b' \leftarrow Expt^{Comp-\mathbf{UO}-a}_{\Pi^H_{\mathcal{E}_a,pk},n,\mathcal{A},(S,[n])}(k)$, such that for $a = 0$, $\mathcal{E}_a$ is defined as $Oracle\ \mathcal{E}_{pk}(\mathbf{LR}(m,\mu,b))$ and for $a = 1$ is defined as $Oracle\ \mathcal{E}_{pk}(\mathbf{LR}(\mu,m,b))$, and H is the honest participants set as defined in the third line of the experiment (see Alg 1).
3) If $b' = 2$ return 1. Otherwise return 0.

$\mathcal{A}'$ is polynomial in k as the Comp-**N**-b experiment takes $poly(k)$ time (see [38]).

We now prove the correctness of the reduction:

Because in both Π and $\Pi_{\mathcal{E}_a}$ messages are taken into Set independently of the application level of the participant (every round a message is moved from the application level to the protocol level with probability of $\frac{1}{\rho}$), the traffic pattern from the application level to the protocol level, has the same distribution for Π and $\Pi_{\mathcal{E}_a}$.

Additionally, we notice that if in the IND-CPA-b experiment $b = 1 - a$, in the experiment $Expt^{Comp-\mathbf{UO}-a}_{\Pi^H_{\mathcal{E}_a,pk},n,\mathcal{A},(S,[n])}(k)$, only encryptions of μ will be inserted into Sets of honest participants, exactly as though $Expt^{Comp-\mathbf{UO}-2}_{\Pi,n,\mathcal{A},(S,[n])}(k)$ was run. If $b = a$, then $\Pi^H_{\mathcal{E}_a,pk}$ is identical to Π. So $Expt^{Comp-\mathbf{UO}-a}_{\Pi,n,\mathcal{A},(S,[n])}(k)$ was run in the second step of $\mathcal{A}'$.

If $\mathcal{A}'$ runs in the IND-CPA-1 experiment, then its probability to return 1 is exactly the probability of $\mathcal{A}$ to return 2 in the Comp-**UO**-$(2-a)$ experiment:

$$Pr[Expt^{ind-cpa-1}_{\mathcal{A}',\psi}(k) = 1] = Pr[Expt^{Comp-\mathbf{UO}-(2-a)}_{\Pi,n,\mathcal{A},(S,[n])}(k) = 2] \quad (2)$$

In the IND-CPA-0 experiment, its probability to return 1 is exactly the probability of $\mathcal{A}$ to return 2 in the Comp-**UO**-$2a$ experiment:

$$Pr[Expt^{ind-cpa-0}_{\mathcal{A}',\psi}(k) = 1] = Pr[Expt^{Comp-\mathbf{UO}-2a}_{\Pi,n,\mathcal{A},(S,[n])}(k) = 2] \quad (3)$$

Substitution of Equations (2) and (3) in Equation (1), for $a \in \{0,1\}$, shows that ψ is not CPA-secure according to Definition 11.

Therefore, if ψ is CPA-secure, both $Expt^{Comp-\mathbf{UO}-a}_{\Pi,n,\mathcal{A},(S,[n])}(k)$ and $Expt^{Comp-\mathbf{UO}-2}_{\Pi,n,\mathcal{A},(S,[n])}(k)$ are indistinguishable for $a \in \{0,1\}$. By the hybrid argument technique, we get that if ψ is CPA-secure, $Expt^{Comp-\mathbf{UO}-0}_{\Pi,n,\mathcal{A},(S,[n])}(k)$ and $Expt^{Comp-\mathbf{UO}-1}_{\Pi,n,\mathcal{A},(S,[n])}(k)$ are also indistinguishable. By Definition 2, that means that Π over n participants is Comp-**UO**-anonymous against attackers with capability $(S,[n])$. ■

VII. The Effect of DURP$^{\Pi}$'s Networking Properties on DURP$^{\Pi}$'s Anonymity and Security

In this section we discuss possible problems and attacks against DURP$^{\Pi}$ that are effected by its networking properties.

A. Degradation of Service Attack

In DURP$^{\Pi}$, the destination server has an expected value for the number of incoming messages every iteration. Flooding the network does not necessarily affect the rate of the incoming messages to the destination because all the peers send one message per iteration, regardless their *Set*'s size. Additionally, due to the distribution of the *Set*s' sizes and the high variance (see Figure 3), it is hard for flooded peers to detect anomalies. We would like to have indications for such attacks from both the destination and the peers perspectives.

B. The Sets' Sizes

We separate this problem to two subproblems:

1) Too many empty Sets: The probability to have an empty *Set* is not low enough. There could be some implications for this fact. First, if an attacker blocks the incoming traffic to peer for low number of iterations, the peer cannot detect that (because it's normal to have an empty *Set*). Second, it is easy for a local eavesdropper to know when a peer is the source of messages it sends (even if the attacker does not know whether it is a real message or dummy).

2) The Sets' sizes are too low: As could be seen from the *Set*s' sizes distribution graphs (Figure 3), most of the *Set*s are of low size.

C. The Maximal Iterations-Counter Value is Too High

As could be seen in the Tables I and II, the maximal *iterations-counter* is high. We can fix this by changing the choosing algorithm. From simulations we ran, FIFO, or uniform choosing from the first X messages in *Set* ($X \in \{3, 4, 5\}$) improved dramatically the maximal *iterations-counter* value. Yet, we do not discuss these results in this paper, because it is obvious that we prefer to choose uniformly from *Set*, for maximizing the mix affect.

D. Anonymity against Eavesdropping Destination

We now show how the network properties and the problems they arise, affect the anonymity against even stronger attacker. Another example is brought in [39].

DURP does not ensures Comp-**SA**-anonymity against malicious destination that is also a global eavesdropper. Briefly, that is because with non-negligible (although low) probability, there could be some scenarios where the attacker can win in the Comp-**SA**-anonymity experiment [39].

In their paper about Crowds [9], Reiter and Rubin defined different degrees of anonymity. Among the degrees between *absolute privacy* to *provably exposed*, the *probable innocence* and the (weaker) *possible innocence* are described as follows:

Definition 13: (From [9]) A sender (receiver) is *probably innocent* if from the attacker's perspective, the sender (receiver) appears no more likely to be the originator than to not be the originator.

A sender (receiver) is *possibly innocent* if from the attacker's perspective, there is nontrivial probability that the real sender is someone else.

Based on Definition 13, we want to measure the innocence degree of a sender. We created an experiment that works as follows: After initializing the protocol, one of N possible senders is chosen, and it sends one signed message to the eavesdropping destination. The attacker wins in the experiment if it guesses correctly the sender.

Based on this experiment, we define *probable innocence degree* against an attacker as the probability of the attacker to lose in the experiment. Similarly, we define *possible innocence degree* as the probability that from the attacker's perspective, the sender is not the only suspected (the probability that the attacker does not detect the real sender for sure).

E. DURP's Innocence Degree

To find DURP's innocence degree against malicious destination that is also a global eavesdropper, we conservatively modeled the global eavesdropper destination attacker, by giving it more than the power a real global eavesdropper has.

We let the attacker know the exact sizes of the peers' *Set*s at any time. Moreover, we let the attacker know how many messages of each state are in the *Set*s. For DURP^{OR} we model the state of each message by the hops it already done. In DURP^{Crowds} each message might have one of three states: a message with zeroed hops-counter, a message with non-zero hops-counter in *Set* and a message with non-zero hops-counter in *DSet*.

The attacker knows the state of every message that is sent, but it cannot distinguish between messages in the same *Set* when the messages have the same state.

For this strengthened attacker, we can easily calculate the probability of each participant to be the sender in the attacker's perspective, by building a simple probability tree based on the information that the attacker has.

As the eavesdropping destination is a passive attacker, the most it can do is to wait for the test message to arrive, and then to calculate a probability vector, that contains the probability of every participant to be the originator of the message. Given such a vector, the attacker should only choose the participant with the highest probability to be the originator. In a case where many participants, including the real sender, are most suspect, we assume the attacker detects the real sender (as though the real sender was the most suspect).

We simulated the experiment for both DURP^{OR} and DURP^{Crowds}, against the strengthened global eavesdropper destination attacker described above. We counted both the failures of the attacker (the times when the real sender was not the most suspect), and the times when the real sender was not totally exposed (the attacker did not know for sure who is the sender). As a fraction of the total experiments number, we respectively denote these measurements by the *probable innocence degree* (shortly *Prob* degree) and the *possible innocence degree* (*Poss* degree).

The results presented in this section, represent stronger attacker than the real global eavesdropper destination attacker.

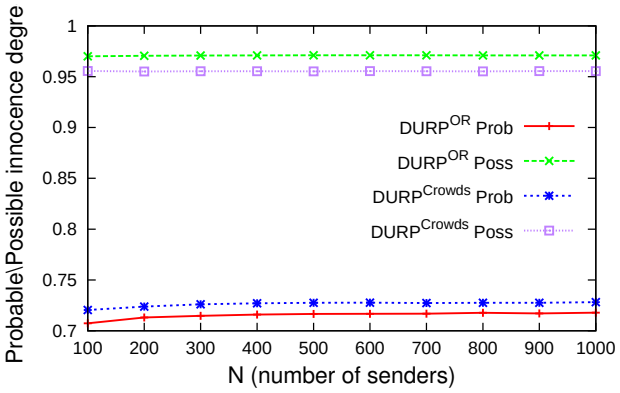

Fig. 1. The Prob and Poss degrees of DURP^{OR} and DURP^{Crowds} against the strengthened eavesdropping destination attacker, as a function of N, the number of potential senders in the experiment, when $\rho = H = 5$.

Hence, we refer the results as lower bounds for the *Prob* and *Poss* degrees. The results are calculated by averaging millions simulations of the described experiment.

We present the results of the experiments for $\rho = 5$ and H = 5 (4 encryption layers in OR, and $P_f = \frac{2}{3}$ in Crowds), and as a function of the number of senders in Figure 1.

For comparison, we ran the same analysis for the Onion-Routing and Crowds [1] protocols in synchronized model (the peers acts in rounds and every sent message reaches its destination until the next round).

To give Crowds and OR a real chance to achieve good results, we defined a traffic pattern functions: $f_1, f_2 : [N] \to \mathbb{N}$, from the participant's index, to the number of messages the participant produces and sends. The traffic pattern function is called by every participant every experiment round.

In the first time, we assumed traffic pattern of one message per iteration for every sender: $f_1 : i \mapsto 1$. In the second case we assumed traffic pattern of one message per iteration with probability of $\frac{1}{2}$: $f_2 : i \overset{r}{\mapsto} \{0, 1\}$. Although the traffic volume in the first case is four times higher then in DURP simulations, and two times in the second, the results are worse than DURP. See Figure 2.

Although achieving better Prob and Poss degrees, the results of DURP are disappointing, due to $Sets$' sizes distribution applied by DURP, and mainly due to the extremely high variance. Example for the Sets' sizes for two configurations appears in Figure 3 in the next section.

VIII. The OLSM Optimization

We now introduce the On Load Send More (OLSM) optimization. OLSM fixes the problems mentioned in the previous section, and particularly ensures much higher Prob and Poss degrees. The optimization has three parameters:

1) $MinSize$. Minimal *Set* size for a peer to run the optimization.
2) T. The number of tries to send another message.

[1]We used the version of Crowds we presented in Section III-B2, as it gives better results than the original Crowds protocol with $P_f = \frac{3}{4}$ that gives $H = 5$

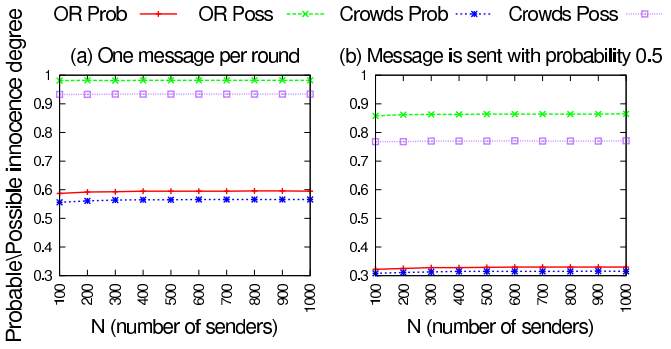

Fig. 2. The Prob and Poss degrees against the strengthened attacker, for OR and Crowds configured to have average hops-counter $H = 5$. (a) represents traffic pattern of one message per round for every participant, and (b) represents traffic pattern of message with probability 0.5 per round.

3) P. The probability that single try succeeds.

The optimization works as follows: if after running the round procedure (see Alg 2), a peer still has *Set* with size $\geq MinSize$, then he gets T chances to send additional messages. Every chance, a random number, $0 \leq r \leq 1$, is chosen, and if $r \leq P$, then another message is sent from *Set* (see Alg 9). We denote DURP$^{\Pi}$ with the OLSM optimization by DURP$^{\Pi}_{OLSM}$.

Algorithm 9 The OLSM optimization is called right after the round procedure (Alg 2).

```
if Set.size ≥ MinSize then
  for i = 1 to T do
    r ←r [0,1]
    if r ≤ P then
      (c, p) ←r Set
      Send c to p
    end if
  end for
end if
```

For using DURP$^{\Pi}_{OLSM}$ and getting its properties, some configurations of DURP$^{\Pi}$ should be changed. In DURP$^{\Pi}$ we demanded that $\rho > H-1$ (Lemma 3); DURP$^{\Pi}_{OLSM}$ demands the opposite: $\rho < H - 1$; otherwise, if $MinSize$ is high enough, DURP$^{\Pi}_{OLSM}$ acts almost the same as DURP$^{\Pi}$.

A. DURP$^{\Pi}_{OLSM}$ Network Properties

We now present some lemmas without their proofs; some of the proofs are similar to those of DURP$^{\Pi}$'s similar properties, and some proofs are trivial. Anyway, we present them such that no lemma depends on its follows:

Lemma 14: The mean of the sending actions during one OLSM call is $T \cdot P$.

Lemma 15: A network under DURP$^{\Pi}_{OLSM}$ is *stable* when $\frac{1+T \cdot P}{H-1} - \frac{1}{\rho} > 0$.

All the below lemmas, assume Lemma 15's condition.

Lemma 16: The average number of the outgoing messages (messages that arrive to the destination) and the average number of new messages every iteration are $\frac{N}{\rho}$.

Lemma 17: The average *iterations-counter* $IC = \frac{\rho \cdot S}{N} + 2$.

Lemma 18: The number of 'send' events in the round procedure every iteration is N (exactly as in DURP$^{\Pi}$).

Lemma 19: The average number of 'send' events due to the OLSM optimization every iteration is $(H-1)\frac{N}{\rho} - N$. We denote this value as $OLSM_{hops}$.

Lemma 20: The average number of OLSM calls every iteration, denoted by $OLSM_{calls}$, is $\frac{1}{T \cdot P} OLSM_{hops}$.

Lemma 21: $\frac{1}{\rho} s_{MinSize} + \sum_{i=MinSize+1}^{\infty} s_i = \frac{OLSM_{calls}}{N}$.

The last lemma stems from the fact that the OLSM optimization is called only if the *Set*'s size is greater than $MinSize$ or if *Set*'s size equals $MinSize$ and a message was added to *Set* from the application (probability of $\frac{1}{\rho}$).

B. Algorithm for getting the distribution of the sizes of the Sets

In Section IV-C we showed how to calculate the *Sets*' sizes distribution by a matrix that defines the probabilities for *Set* of size j to be changed to size i. We now show how to build such a matrix for DURP$^{\Pi}_{OLSM}$.

As in DURP$^{\Pi}$, we notate the mean of messages that might arrive to some peer (not the destination) as D. D is calculated by substraction of the outgoing messages' mean from the mean of the total sending actions in iteration when we count only messages of other peers (we consider only messages that might arrive to the peer). $D = \frac{N-1}{\rho}(H-1) - \frac{N-1}{\rho} = \frac{(N-1)(H-2)}{\rho}$.

The function **OlsmProbToSendK**(int k) (Alg 10) returns the probability that during a call to the OLSM optimization, k messages were sent from the caller's *Set*.

Algorithm 10 The **OlsmProbToSendK**(int k) function.

return $\binom{T}{k} P^k (1-P)^{T-k}$

Let's denote the difference between the incoming messages (calculated by **ProbToGetK**, Alg 5, but with the new D value) and the outgoing messages due to OLSM call (calculated by **OlsmProbToSendK**, Alg 10) by Δ. The function **OlsmΔProb**(int k) (Alg 11) returns the probability that during a call to the OLSM optimization $\Delta = k$.

Algorithm 11 The **OlsmΔProb**(int k) function.

```
output = 0
for i = 0 to T do
    output += OlsmProbToSendK(i)·ProbToGetK(i + k)
end for
return output
```

Similarly to DURP$^{\Pi}$, we build the probabilities transitions matrix, M, using the **OlsmBuildProbsMatrix** function (Alg 12). As in DURP$^{\Pi}$, with a bit longer code, we could avoid recalculation of the same values.

Given such a matrix, we can use the procedure **SetsSizesByMatrix** (Alg 8) to calculate the *Sets*' sizes distribution applied by that matrix. The calculation of the *Sets*' sizes distribution of DURP$^{\Pi}_{OLSM}$ with accuracy parameter n is done by **SetsSizesByMatrix**(**BuildProbsMatrix**(n)).

Algorithm 12 The **OlsmBuildProbsMatrix**(int n) function.

```
Initialize zeros matrix M^{n×n}
for j=0 to n-1 do
  for i=j-1 to n-1 do
    if j = 0 then
      M_{i,j}=ProbToGetK(i)
    else if j < MinSize then
      M_{i,j}=ProbToBeChangedInK(i-j)
    else if j = MinSize then
      M_{i,j}=(1/ρ)Olsm∆Prob(i-j)+(1-1/ρ)ProbToGetK(i-j+1)
    else if j > MinSize then
      M_{i,j}=(1/ρ)Olsm∆Prob(i-j)+(1-1/ρ)Olsm∆Prob(i-j+1)
    end if
  end for
end for
return M
```

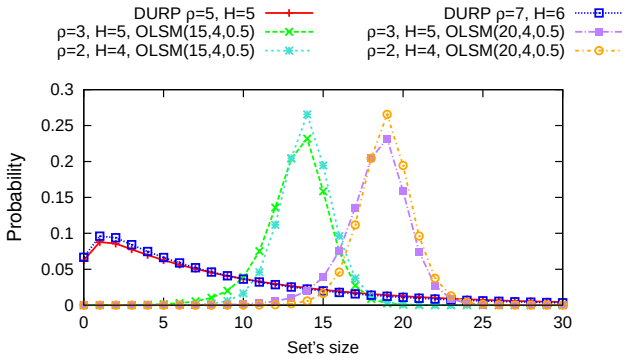

Fig. 3. The distribution of *Sets*' sizes for different configurations of DURP$^{\Pi}$ and DURP$^{\Pi}_{OLSM}$. The results are for $\Pi \in \{\text{OR, Crowds}\}$.

C. OLSM Simulation Results

We ran the simulations for DURP$^{\Pi}_{OLSM}$ as we did for DURP$^{\Pi}$ (Section IV-D). The expected results are calculated as follows: **SetsSizesByMatrix**(**OLSMBuildProbsMatrix**(100)) (Alg 8 and 12) outputs s_i for $i = 0, ..., 99$. S is calculated by Lemma 7, and IC by Lemma 17. $MaxIC$ is the maximal *iterations-counter* value for a message in the simulation. $OLSM_{hops}$ and $OLSM_{calls}$ are calculated by Lemmas 20 and 21.

Figure 3 contains examples for the $Sets$' sizes distribution for four different configurations of DURP$^{\Pi}_{OLSM}$. The *Sets*' sizes distributions by the algorithm and by the simulations for both DURP^{OR} and DURP^{Crowds} are visually identical.

Tables III and IV contain comparison of the simulation results to the expected results according to the analysis. The *Sets*' sizes distribution appears in Figure 3. In both the cases, for $\Pi = Crowds$, the average hops-counter was tightly around 5, and the maximal hops-counter was around 51 ± 2.5.

D. The Advantages of DURP$^{\Pi}_{OLSM}$

1) Better network properties: The problems of DURP$^{\Pi}$ we mentioned in Section VII, are mitigated by DURP$^{\Pi}_{OLSM}$. As a direct result of the negligible number of empty $Sets$ (see Figure 3), the number of dummy messages (line 10 in Alg 2) is negligible.

TABLE III. $N = 100, H = 5, \rho = 3, MinSize = 20, T = 4, P = 0.5$

	Expected results	DURP^{OR}_{OLSM}	$\text{DURP}^{Crowds}_{OLSM}$
S	1833.34	1832.85±0.06	1832.75±0.05
IC	57.00	56.99±0.00	56.98±0.00
$MaxIC$	-	405.5±35.83	865.75±70.27
$OLSM_{hops}$	33.33	33.33±0.01	33.33±0.01
$OLSM_{calls}$	16.67	16.67±0.00	16.67±0.00

TABLE IV. $N = 100, H = 5, \rho = 3, MinSize = 15, T = 4, P = 0.5$

	Expected results	DURP^{OR}_{OLSM}	$\text{DURP}^{Crowds}_{OLSM}$
S	1333.45	1333.03±0.06	1332.89±0.07
IC	42.00	40.99±0.00	40.99±0.00
$MaxIC$	-	293.75±12.50	615.62±49.36
$OLSM_{hops}$	33.33	33.34±0.01	33.34±0.01
$OLSM_{calls}$	16.67	16.67±0.00	16.67±0.01

The low variance and the new traffic pattern, enable detection of anomalies, e.g., flooding attacks, from the perspective of both the reporters and the destination. Reporters know that with high probability, the number of the messages in their *Set* is in some range around the $MinSize$ parameter. In case of blocking or flooding a reporter, it is possible to detect this anomaly. From the destination's perspective, if the network is flooded, the number of the messages that reach the destination per iteration will exceed the value appears in Lemma 16.

From the simulation results, we noticed that the maximal *iterations-counter* is also lower than DURP^{Π}.

2) DURP^{Π}_{OLSM}'s anonymity properties: Theorems 9 and 12 hold also for DURP^{Π}_{OLSM}; the proofs are almost identical. Additionally, as we believed, DURP^{Π}_{OLSM} gave significantly higher Prob and Pros degrees against eavesdropping destination, as could be seen in Figure 4.

Examining the results, the fact that in $\text{DURP}^{Crowds}_{OLSM}$ the attacker succeeded with a double probability than DURP^{OR}_{OLSM}, and as opposed to the results in Figure 1 and relatively more than in Figure 2, was a bit surprising. The messages of Crowds are mixed better in Sets because the path length is not constant (there are more possible paths to the destination).

The explanation for this phenomenon, is that originators of messages that reached the destination through shorter paths, have higher probability to be suspected or exposed. In Crowds, the length of the path is chosen by geometric distribution with

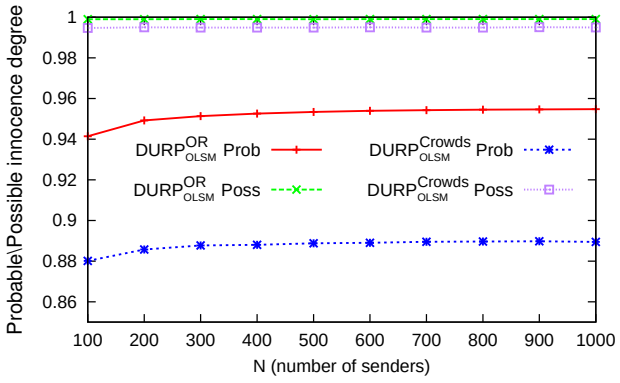

Fig. 4. The Prob and Poss degrees against the strengthened attacker, for DURP^{Π}_{OLSM} with $\rho = 3$, $H = 5$, and the OLSM parameters: $MinSize = 15$, $T = 4$, $P = 0.5$.

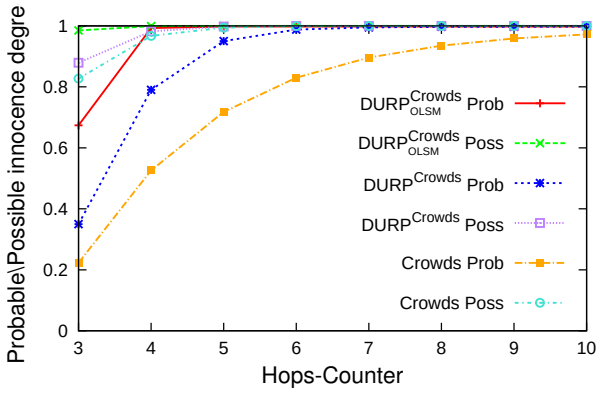

Fig. 5. The Prob and Poss degrees as a function of the hops-counter of the test message in the experiment. The $\text{DURP}^{Crowds}_{OLSM}$ was configured as in Figure 4, DURP^{Crowds} was configured as in Figure 1, and Crowds was configured as in Figure 2 (a). All the simulations were done over network of 500 senders.

the parameter $1 - P_f$; therefore, $\frac{1}{1-P_f}$ of the messages has the shortest possible path. We noticed that in all Crowds (with some traffic pattern), DURP^{OR} and $\text{DURP}^{Crowds}_{OLSM}$, most of the attacker's successes are with messages that reached the destination via short paths. Figure 5 depicts this point.

IX. Conclusions

Anonymous and unobservable communication is an interesting research challenge, and significant for some sensitive applications and scenarios. We show DURP, a practical, efficient, and fully decentralized (peer to peer) protocol, ensuring anonymous and unobservable communication to a single destination ('reporting'). DURP is modular, and can be instantiated with known protocols like Onion-Routing or Crowds.

DURP may be appropriate for sensitive applications, which require anonymity and unobservability requirements beyond these provided by existing practical designs, e.g., Tor. In particular, Tor does not provide either anonymity or unobservability against a global eavesdropping adversary.

We proved that DURP ensures source anonymity, as well as unobservability against malicious peers and global eavesdroppers. We also showed that DURP provides anonymity against global eavesdropper destination. We also analyzed its network properties and performance, and showed it has reasonable overhead.

More work is required toward practical decentralized protocols for strongly-anonymous communication. One obvious goal is to support *multiple destinations*, with destination-anonymity and unlinkability - while maintaining reasonable overhead. Another challenge is to investigate the feasibility of a provably-anonymous protocol, for an eavesdropping adversary which controls the destination; a further challenge is to allow the adversary also to control some of the nodes (peers). Another challenge is support of (semi) asynchronous networks.

References

[1] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The Second-Generation Onion Router," in *USENIX Security Symposium*. USENIX, 2004, pp. 303–320. [Online]. Available: http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html

[2] Y. Gilad and A. Herzberg, "Spying in the Dark: TCP and Tor Traffic Analysis," in *Privacy Enhancing Technologies Symposium*, ser. Lecture Notes in Computer Science, S. Fischer-Hübner and M. Wright, Eds., vol. 7384. Springer, 2012, pp. 100–119. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31680-7

[3] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against tor," in *Proceedings of the 2007 ACM workshop on Privacy in electronic society*. ACM, 2007, pp. 11–20.

[4] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz, "Denial of service or denial of security?" in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 92–102.

[5] S. Chakravarty, G. Portokalidis, M. Polychronakis, and A. D. Keromytis, "Detecting traffic snooping in tor using decoys," in *Recent Advances in Intrusion Detection*. Springer, 2011, pp. 222–241.

[6] R. Dingledine and N. Mathewson, "Tor path specification." [Online]. Available: https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=path-spec.txt

[7] "Tor network status." [Online]. Available: http://torstatus.blutmagie.de/index.php?SR=Bandwidth&SO=Desc

[8] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[9] M. Reiter and A. Rubin, "Crowds: Anonymity for web transactions," *ACM Transactions on Information and System Security (TISSEC)*, vol. 1, no. 1, pp. 66–92, 1998.

[10] M. Reed, P. Syverson, and D. Goldschlag, "Anonymous connections and onion routing," *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 4, pp. 482–494, 1998.

[11] J. Ren and J. Wu, "Survey on anonymous communications in computer networks," *Computer Communications*, vol. 33, no. 4, pp. 420–431, 2010.

[12] W. Dai, "Pipenet 1.1," 1996. [Online]. Available: http://www.weidai.com/pipenet.txt

[13] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of cryptology*, vol. 1, no. 1, 1988.

[14] M. Waidner, B. Pfitzmann *et al.*, "The dining cryptographers in the disco: unconditional sender and recipient untraceability with computationally secure serviceability," *J.-J. Quisquater and J. Vandewalle, editors, Advances in CryptologyEUROCRYPT*, vol. 89, p. 690, 1989.

[15] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman, "Mixmaster Protocol — Version 2," IETF Internet Draft, 2003.

[16] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a Type III Anonymous Remailer Protocol," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003, pp. 2–15.

[17] A. Pfitzmann, B. Pfitzmann, and M. Waidner, "Isdn-mixes: Untraceable communication with very small bandwidth overhead," in *GI/ITG Conference on Communication in Distributed Systems*, vol. 267, 1991, pp. 451–463.

[18] A. Jerichow, J. Muller, A. Pfitzmann, B. Pfitzmann, and M. Waidner, "Real-time mixes: A bandwidth-efficient anonymity protocol," *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 4, pp. 495–509, 1998.

[19] O. Berthold, H. Federrath, and S. Köpsell, "Web mixes: A system for anonymous and unobservable internet access," in *Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 115–129.

[20] A. Serjantov, R. Dingledine, and P. Syverson, "From a trickle to a flood: Active attacks on several mix types," in *Information Hiding*. Springer, 2003, pp. 36–52.

[21] M. Wright, M. Adler, B. N. Levine, and C. Shields, "An analysis of the degradation of anonymous protocols." in *NDSS*, vol. 2, 2002, pp. 39–50.

[22] M. K. Wright, M. Adler, B. N. Levine, and C. Shields, "The predecessor attack: An analysis of a threat to anonymous communications systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 4, pp. 489–522, 2004.

[23] K. Sampigethaya and R. Poovendran, "A Survey on Mix Networks and Their Secure Applications," *Proceedings of the IEEE*, vol. 94, no. 12, pp. 2142–2181, 2006.

[24] M. Freedman and R. Morris, "Tarzan: A peer-to-peer anonymizing network layer," in *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 2002, pp. 193–206.

[25] S. Goel, M. Robson, M. Polte, and E. Sirer, "Herbivore: A scalable and efficient protocol for anonymous communication," 2003.

[26] A. Beimel and S. Dolev, "Buses for anonymous message delivery," *Journal of Cryptology*, vol. 16, no. 1, pp. 25–39, 2003.

[27] C. Shields and B. Levine, "A protocol for anonymous communication over the internet," in *Proceedings of the 7th ACM conference on Computer and communications security*. ACM, 2000, pp. 33–42.

[28] M. Rennhard and B. Plattner, "Introducing morphmix: peer-to-peer based anonymous internet usage with collusion detection," in *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*. ACM, 2002, pp. 91–102.

[29] L. von Ahn, A. Bortz, and N. J. Hopper, "K-anonymous message transmission," in *Proceedings of the 10th ACM conference on Computer and Communications Security*. ACM, 2003, pp. 122–130.

[30] G. Yao and D. Feng, "A new k-anonymous message transmission protocol," in *Information Security Applications*. Springer, 2005, pp. 388–399.

[31] A. Mislove, G. Oberoi, A. Post, C. Reis, P. Druschel, and D. Wallach, "Ap3: Cooperative, decentralized anonymous communication," in *Proceedings of the 11th workshop on ACM SIGOPS European workshop*. ACM, 2004, p. 30.

[32] A. Nambiar and M. Wright, "Salsa: A structured approach to large-scale anonymity," in *Proceedings of CCS 2006*, November 2006.

[33] P. Mittal and N. Borisov, "Shadowwalker: peer-to-peer anonymous communication using redundant structured topologies," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 161–172.

[34] H.-C. Hsiao, T. H.-J. Kim, A. Perrig, A. Yamada, S. Nelson, M. Gruteser, and W. Ming, "LAP: Lightweight anonymity and privacy," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, May 2012.

[35] N. Hopper, E. Vasserman, and E. Chan-Tin, "How much anonymity does network latency leak?" *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 2, p. 13, 2010.

[36] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright, "Timing attacks in low-latency mix systems," in *Financial Cryptography*. Springer, 2004, pp. 251–265.

[37] A. Hevia and D. Micciancio, "An indistinguishability-based characterization of anonymous channels," in *Privacy Enhancing Technologies*. Springer, 2008, pp. 24–43.

[38] A. nonymous, "On the Limits of Provable Anonymity," Tech. Rep., 2013, available from the authors upon request via the program chair.

[39] ——, "Efficient Unobservable Anonymous Reporting against Strong Adversaries," 2013, draft of full paper. Available from the authors upon request via the program chair.

[40] M. Bellare and P. Rogaway, "Asymmetric Encryption," http://cseweb.ucsd.edu/ mihir/cse207/w-asym.pdf.

[41] M. Raab and A. Steger, "balls into bins - a simple and tight analysis," in *Randomization and Approximation Techniques in Computer Science*. Springer, 1998, pp. 159–170.