WHISPER: Middleware for Confidential Communication in Large-Scale Networks

Valerio Schiavoni, Etienne Rivière and Pascal Felber
University of Neuchâtel, Switzerland
first.last@unine.ch

***Abstract*—A wide range of distributed applications requires some form of confidential communication between groups of users. In particular, the messages exchanged between the users and the identity of group members should not be visible to external observers. Classical approaches to confidential group communication rely upon centralized servers, which limit scalability and represent single points of failure. In this paper, we present WHISPER, a fully decentralized middleware that supports confidential communications within groups of nodes in large-scale systems. It builds upon a peer sampling service that takes into account network limitations such as NAT and firewalls. WHISPER implements confidentiality in two ways: it protects the content of messages exchanged between the members of a group, and it keeps the group memberships secret to external observers. Using multi-hops paths allows these guarantees to hold even if attackers can observe the link between two nodes, or be used as content relays for NAT bypassing. Evaluation in real-world settings indicates that the price of confidentiality remains reasonable in terms of network load and processing costs.**

I. INTRODUCTION

1) Context: The use of confidential communication between a group of distributed entities is at the core of many applications. Examples include private chat rooms in social networks, information sharing systems guaranteeing freedom of speech, control-flow and admission-control for pay-per-view live-streaming, distributed content indexes that should not be made public to prevent attacks, etc.

The confidentiality of the messages exchanged between the members of a *private group* is typically achieved by the use of encrypted channels that prevent malevolent parties from spying on the exchanged content. However, content encryption alone is insufficient for many of the applications mentioned above, that also require that the composition of the group additionally remain secret, i.e., one should not be able to determine whether or not a node belongs to a group. Furthermore, the existence of the group itself shall also be hidden to unauthorized parties.

Computer networks typically use centralized solutions for supporting private group communication, e.g., by relying on dedicated servers. VPNs allow nodes to create private communication channels with encrypted traffic. Group communication can leverage VPNs, for instance as part of a multi-site company infrastructure, with all communications between sites being routed through some VPN gateways.

In large-scale dynamic settings, where sizable populations of nodes are interconnected in self-organizing and often loosely structured overlays, the use of VPN-based solutions (or similar centralized mechanisms) is inadequate for implementing private group communications. The VPN gateways act as single points of failure, hereby forfeiting one of the major benefits of decentralized systems: their robustness to targeted attacks such as denial of service attacks. As soon as the attackers know the gateway, it is straightforward to disrupt communications within the private group by concentrating the attacks on the gateway. Further disadvantages of solutions based on VPNs or dedicated servers include their scalability to large amounts of users, their price, and their operating costs.

We should point out that these approaches do not keep the membership of private groups hidden from malevolent nodes; only the content of exchanged messages is protected. The process of hiding the communication partners, and hence the identity of the members of the group, must be provided by anonymizing systems such as TOR [1]. These systems rely on a set of dedicated servers that conceal the source of the message from the destination, typically using onion routing mechanisms [2], [3]. Here again, the cost of provisioning such a system with sufficiently many dedicated servers can be a hindrance in large-scale self-organizing networks.

2) Objectives: These observations make the case for a fully *decentralized, autonomous, and self-organizing* service to support *confidential communications* within groups of nodes in large-scale systems. Unlike existing approaches, this service should let private groups be created by ordinary nodes and emerge within the network, without relying on dedicated and trusted third parties servers. At the same time, it must hide communications between the members of a private group from the other nodes (*content privacy*), as well as keep the group memberships secret to external observers (*membership privacy*). This latter point is especially important, as it is impossible for an attacker to focus an attack on the members of a group without being able to determine their identity.

We target large-scale, Internet-wide networked systems in which a large majority of nodes [4] reside behind network address translation (NAT) devices or firewalls. Our algorithms exploit *peer sampling* as an underlying approach for organizing nodes in the network in a fully decentralized and autonomous manner. We consider *malevolent* nodes that spy upon other nodes in the system, but that follow the protocol specification and do not exhibit other byzantine behavior.

3) Contributions: This paper present WHISPER, a fully decentralized approach to confidential group communication in large-scale systems in which the traffic may have to take

multi-hops paths to circumvent network limitations such as NAT and firewalls. WHISPER supports the creation of confidential communication routes without the need for a trusted third party. It additionally provides membership management and overlay maintenance amongst private groups of nodes communicating in a confidential manner. Our comprehensive evaluation of WHISPER in real-world settings indicates that the price of confidentiality remains reasonable in terms of network load and processing costs.

4) Outline: The remainder of this paper is organized as follows. We first provide background information on the concepts underlying the WHISPER design in Section II: the peer sampling service, its gossip-based implementation, and the impact of NAT-aware implementations on the design of confidential group communication mechanisms. Sections III and IV describe the WHISPER components and algorithms. We present in Section V evaluation results of the WHISPER protocol stack implementation deployed on a cluster and on the PlanetLab testbed. We survey related work in Section VI, and conclude in Section VII.

II. Problem Definition and Overall Architecture

In this section, we first present our system model. We then give a short overview of the *peer sampling service* and discuss how practical aspects related to its implementation impact on the proposal of private group communications in large-scale systems.

A. System Model

We consider large-scale networked systems with no centralized entity or trusted third party. We target real-world networks with limited connectivity, rather than idealized settings where all nodes would be able to directly communicate with one another. In particular, we assume that, as observed on the Internet [4], a large majority of nodes reside behind NAT devices or firewalls and can only be reached by using dedicated NAT-traversal mechanisms such as hole-punching or relays. Finally, our system model takes into account churn: nodes are expected to join and leave the network continuously.

We consider the following threat model. Nodes always follow the protocol specification and do not forge, modify, replay, or drop messages. Nevertheless, nodes can be *malevolent* in the sense that they may spy upon other nodes that belong to groups they do not belong to themselves. Confidentiality can be broken in two ways: by observing the content of the messages exchanged between two members of a private group (content privacy), and by determining which nodes belong to a private group (membership privacy). Unless explicitly indicated otherwise, we assume that nodes are not colluding in trying to break confidentiality, i.e., each malevolent node acts on its own. In particular, we do not consider that an attacker is able to observe enough links and infer group communications based on multi-point traffic analysis.

We consider the links between nodes to be unsafe, i.e., an attacker may be able to observe the traffic sent over a given link. We do assume, however, that the attacker can only control a limited number of links. In particular, it is not able to spy on all the links of a multi-hops path (see Section III) between two nodes.

B. The Peer Sampling Service

Our algorithms exploit *peer sampling* as underlying approach for organizing nodes in the network in a fully decentralized and autonomous manner. Peer sampling transparently deals with the complexity of membership management in large dynamic networks, with nodes joining and leaving continuously, and hence simplifies the building of many complex protocol stacks and applications. One can mention application-level multicast [5], [6], [7], aggregation protocols [8], [9], [10], network size estimation [11], overlay construction [12], [13], [14], [15], failure detection [16], or network provisioning [17].

A peer sampling protocol, or *peer sampling service* (PSS), provides each node in the system with a continuous flow of other peers in the system that forms a sequence of partial views. Ideally, the graph built from the local views in a network operating a PSS highly resembles a random graph, with a small diameter, balanced node in- and out-degree, and strong resilience to disconnection. Membership management is implemented by ensuring that failed or departing nodes disappear from the views of live nodes after a bounded amount of time, and that newly arrived nodes are inserted, and remain, in a sufficient number of other views.

Typical implementations of a PSS for dynamic systems use gossip-based—or epidemic—protocols [18]. Such protocols are fully decentralized and rely on periodic pair-wise interactions between nodes. Each node has a small limited *view* of the network. For the PSS, this view consists of a set of c other nodes. Periodically, each node contacts another node from its view and they both exchange a subset of their view. Newly received entries are inserted in the local view, which is then truncated to the maximal size c. Such an exchange is typically denoted as a *shuffling* of views [19].

Several parameters impact on the operation of a gossip-based PSS [18]. In our context, we consider the following strategy. Each entry in the view of a node is tagged with an age that helps determine the *freshness* of the entry (how many exchanges since it has been inserted by the node to which it points to; entries get older at each cycle and nodes insert themselves with an age of 0). The exchange partner is always the entry with the highest age in the view, while after the exchange a new view is decided by each partner based on the lowest ages of the entries of their two view (this strategy is denoted as the `healer` in [18]).

The quality of the overlay created by the PSS is measured by its resemblance to a random graph with fixed out-degrees. A balanced distribution of the nodes' in-degrees ensures load-balancing. A low clustering factor indicates that the diversity of the peers in the views will be maximized: there is no presence of aggregates of nodes that are well-connected together but less linked to the rest of the network. It has been consistently observed by [8], [9], [10], [12], [13], [15], [17]

that low clustering factors lead to better convergence properties of protocols that use the PSS.

C. Practical PSS Implementation and NAT Resilience

Gossip-based PSS protocols [18] typically assume that direct communication is possible between any pair of nodes in the system. However, this assumption does not hold in a real system, where a majority of nodes lie behind network address translation (NAT) devices.[1] For instance, in a large study on 7M nodes, Casado and Freedman [4] observed that more than 60% of the nodes were behind NATs. We refer to the nodes behind a NAT device as N-nodes (*Natted*-nodes), while others are called P-nodes (*Public*-nodes).

Connecting to N-nodes requires implementing NAT-traversal techniques. A widely used approach is *hole punching*. It requires that communication is first established by N-nodes with a public *rendezvous* (RV) node, hence opening a connection through the NAT device. The RV node can then share with each N-node the address and port used by the other for this session so that they can communicate directly with each other. While this method is usable with a large number of NAT combinations [20], there are cases where hole punching is not applicable and the RV node has to act as a relay for all content sent to the N-node. Note that we consider firewalls as being similar to NAT devices w.r.t. to our problem, as their traversal is based on the same techniques.

Deploying a PSS onto a network without considering the impossibility of directly joining N-nodes results in a high imbalance of in-degrees and major negative impact on robustness and connectivity. We base our work on the $\mathcal{N}ylon$ [21] practical NAT-resilient PSS implementation, which maintains the following property: for any node B that lies in the view of a node A, there exists a possibility, known to $\mathcal{N}ylon$, to open a communication channel from A to B. This communication channel can be constructed with the help of a chain of P- and N-nodes acting as RVs. Note that the ability of A to communicate with B once the connection has been opened typically lasts longer than the time of presence of the node in the view, and depends on the lease time of association rules maintained by NAT devices. Typical lease times are in the order of minutes for UDP and hours for TCP. Cisco's lease times specifications [22] are of 5 minutes for UDP and 24 hours for TCP. Observations over a large variety of NAT devices [23] indicate that about 80% of the NAT do not support hole punching mechanisms for TCP, and about 65% for UDP. This results in an important aspect of $\mathcal{N}ylon$ in our context: in many cases, RV nodes will need to be used as relays for the actual messages exchanged between peers and not only for the connection establishment messages.

Building private groups directly on top of the PSS would contradict the objective of hiding the content of messages exchanged (irrespective of the use of $\mathcal{N}ylon$ mechanisms), as we consider that an external attacker may observe the link. Using a simple encryption mechanism would solve this issue, but the existence of secret communication between two nodes, and hence their participation to a common private group would be exposed and allow group membership mapping. When using the $\mathcal{N}ylon$ PSS implementation, the content must potentially go through relay nodes that can obtain the same information about the source and destination. The WHISPER protocol that we describe in the remaining of this paper goes beyond this simple but ineffective solutions by allowing the confidentiality of both communications and memberships.

[1] A NAT device allows sharing a single connection between multiple peers. It keeps track of existing connections with external peers and translates external addresses into private addresses assigned to the nodes behind the NAT device. All other traffic is typically dropped.

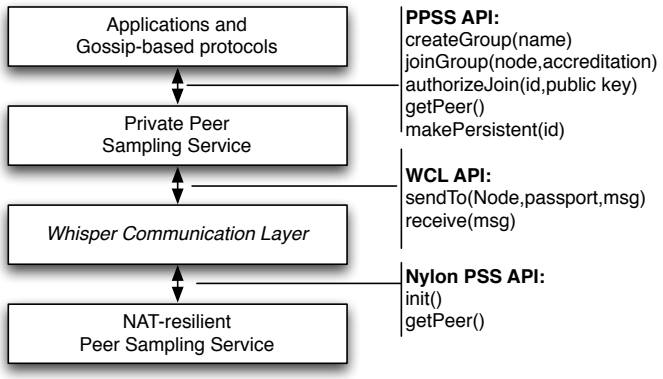

Fig. 1: WHISPER layers and interfaces.

D. The WHISPER Architecture

WHISPER is the combination of two layers.

- The WHISPER communication layer (WCL) operates on top of the $\mathcal{N}ylon$ NAT-aware PSS. It allows confidential communications between peers by protecting both exchanged content and relationship anonymity, even when relays have to be employed for bypassing NAT limitations or individual links be monitored by an attacker.
- The private peer sampling service (PPSS) operates on top of the WCL. It provides the services of a PSS: it acts as a provider of a *private view* of live peers for the applications operating in the private group. It leverages the WCL to guarantee that communications with any node in the private view will remain strictly confidential. The PPSS also deals with group management and membership authentication, and ensures that confidential connections between peers can be maintained even when the destination node does no longer belong to the view of the source node.

Figure 1 presents the relations between the various elements composing WHISPER. We describe the WCL and the required modifications to the PSS in Section III. The PPSS will be presented in Section IV.

III. WCL: THE WHISPER COMMUNICATION LAYER

The objective of the WCL is to allow two nodes, a source S and a destination D, to communicate in a confidential manner without the need for a trusted third party. Confidentiality here

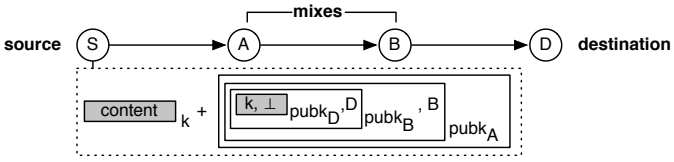

Fig. 2: A path using two mixes between S and D and the onion-encrypted message generated by S.

is twofold. (1) *Content confidentiality:* the content of the message exchanged should be visible only to S and D and no other nodes, including the relays that might be used by the $\mathcal{N}ylon$ PSS layer for bypassing NATs. (2) *Relationship anonymity:* the identity of S and D taken together must not be known by any other node, that is, it is not possible for a node to tell that S and D are engaged in an exchange. This latter guarantee is necessary to ensure that the membership of nodes to private groups can be kept secret to third-party nodes by the above PPSS layer.

A. WCL *Path Construction Algorithm*

The WCL provides a one-way multi-hop communication channel between two peers S and D. The content is first encoded by S using symmetric encryption with a random key k (we use AES in our prototype). This ensures content confidentiality.

To guarantee relationship anonymity, we communicate via a path of nodes following the principle of the real-time variant of Chaum's mixes [3], called onion routing [2]. An example is given by Figure 2. Onion routing is a technique for anonymizing communications in a distributed system using a path of relay nodes (called *mixes*). Each mix has a private and public key pair, and we consider that S, the node preparing the path, knows both the identity of nodes that can act as mixes and each of their public keys (we describe how we support public key management in Section III-B). S encrypts a pair $(k,\perp)$ with the public key of D in a variable O. Thereafter, S considers each mix M along the path, in reverse order starting from D, and encrypts O and the identity of the next hop using the public key of M, thus producing a new O. This implies that the decryption of the final O (the *onion path*) needs to be performed in sequence by each of the mixes, using their respective private keys, for determining at each step the identity of the next hop. The last hop (D) learns that it is the destination after decrypting the message, because the identity of the next hop is $\perp$, but this fact cannot be known by the previous mix.

Paths composed of exactly four nodes, including S and D, are sufficient to protect relationship anonymity. Consider a path $S \rightarrow A \rightarrow B \rightarrow D$ with A and B acting as mixes. As A and B have no mean to detect whether the next-to-next hop is $\perp$, they cannot determine whether they are forwarding the message to another mix or to the destination node. For the same reason, neither A nor B can determine whether the node that forwarded them the encrypted message is a mix or the source. Nodes are not colluding in our model so the identity of A's predecessor is not sent along with the message towards B [2]. This guarantees that no intermediate nodes are able to know the identity of A and B simultaneously, thus ensuring relationship anonymity. It also holds when the attacker can obtain the message exchanged on one hop (we consider that the attacker may be able to control one such hop but that it is not able to observe all three links on the path, as it would require a massive control of the network when hops are chosen randomly).

The path from S to D needs to be a valid path, that is, communication must be possible for the three hops $S \rightarrow A$, $A \rightarrow B$, and $B \rightarrow D$. In particular, we need to make sure that NAT-resilient routes are opened. The links $S \rightarrow A$ and $B \rightarrow D$ can leverage the existing NAT-resilient routes that have been opened by the $\mathcal{N}ylon$ PSS, and for which hole-punching or the setup of relays is still valid. $\mathcal{N}ylon$ opens such links *on demand* when a node in the view is contacted, either as part of the PSS operation or for the application. Note that thereafter the usability of these opened connections does not depend on the presence of the corresponding node in the $\mathcal{N}ylon$ PSS view. As detailed in Section II, NAT association rules lease times typically range from minutes for UDP to hours for TCP. This is much larger than the typical cycle time of the PSS operation, which is in the order of dozens of seconds [22]. Once a connection has been opened towards a node, the NAT traversal path remains usable as long as association rules remain active.

The creation of the routes and the associated state at each node are illustrated by Figure 3. Each node builds a *connection backlog* (CB) containing the connection information from $\mathcal{N}ylon$ about the valid hops towards nodes that have been recently contacted. We only consider for inclusion in the CB the nodes that are contacted (or that contact the local node) for a successful gossip exchange, and ignore connections created for the application. The rationale is that gossip exchanges are necessarily bidirectional while this may not be the case for application-level traffic. This means that if node X can contact another node Y through a NAT-resilient route, there also exists such a route from Y to X. The CB is a FIFO queue with maximal size of $2 \times c$ (twice the view size of the PSS). Gossip partners to which a path is opened are inserted at the beginning of the queue and supernumerary elements are removed at its tail. Every cycle, a node contacts a partner in its view and receives, on average, one gossip request from another node. This means that on average two valid new paths will be available during each PSS cycle. Considering for example a PSS cycle time of 10 seconds, $c = 10$ elements in the view, and a backlog of $2 \times c = 20$ elements in the CB, each node will remain at most for 100 seconds in the CB, i.e., much less than the minimal lease time for the NAT association rules.

The CB is used to decide on the $S \rightarrow A$ and $B \rightarrow D$ hops (we consider for now that the CB of D is known to S and

[2]Note that we can support colluding nodes by simply increasing the length of the path: using f mixes allow to support $f-1$ colluding nodes. We make the assumption that nodes are not colluding to simplify the presentation of the protocols and as it is unlikely to be the case in practice.

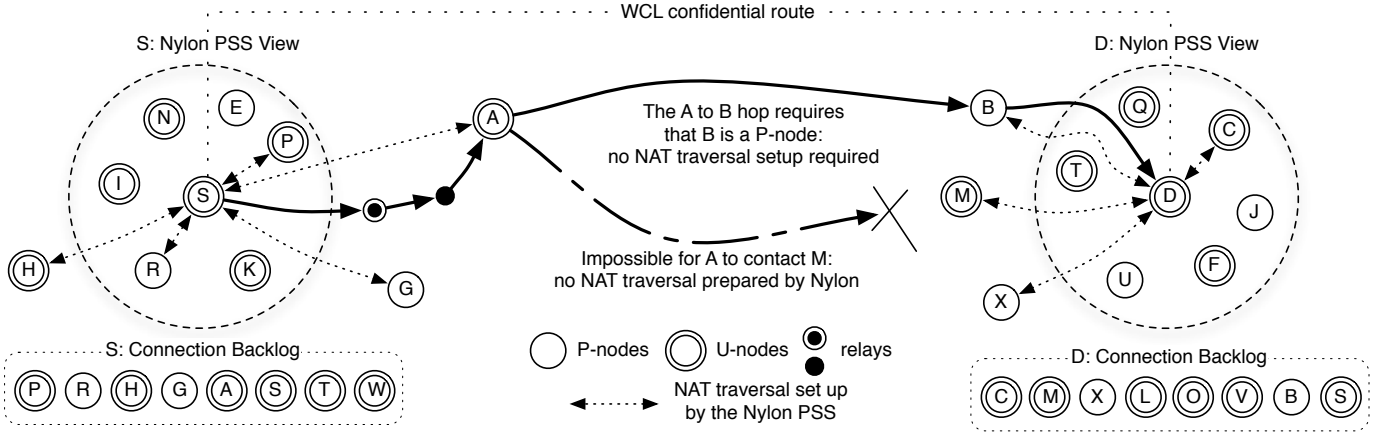

Fig. 3: Illustration of a WCL path constructed from the CBs of the source and destination nodes. Only nodes that are or have recently been in the CB can be used for constructing the path, and the next-to-last hop as the additional constraint that it must also link to a P-node.

will explain how this is achieved in Section IV). However the case of the $A \rightarrow B$ hop is different: even if the node S knew the CB of A and could make an informed choice for the next node B, there is no guarantee that there exists a valid node belonging to the CBs of A *and* D with which both nodes can communicate. Therefore, in order to ensure that A can contact B, B must be a P-node that can be reached without restriction by A. This is illustrated by Figure 3, where the connection from A to B is possible as B is a P-node, but where the choice of node M as the next-to-last hop would not work: there is no pre-opened path that traverses the potential NATs between A and M. Since the majority of nodes in the network lie behind NATs, we need to maintain a minimal number of P-nodes, denoted by Π, in the CB.

Upon inserting a new node in the CB, we verify that there at least Π P-nodes remain in the queue. If that is not the case, we take a P-node P from the PSS view that does not already belong to the CB, send it an empty message to ensure that a valid path exists from P to the current node, and finally insert P in the CB. This process is repeated as long as there are fewer than Π nodes in the CB. The worst case occurs when all the Π P-nodes of the CB lie at its tail. In this case, the new node inserted in the CB will result in the removal of the P-node at the tail, which will then be replaced by a new P-node at the head of the CB, flushing out another P-node that needs to be replaced too, and so on. It results that in the worst case, one will need to insert Π new P-nodes from the view. To support this, there must obviously be at least Π P-nodes in the view. To ensure this property, we introduce a modification to the PSS protocol operating with *Nylon*, as described next.

B. Peer Sampling and Public Key Sampling Services

In order to support the WCL, two modifications are required to the PSS protocol. First, for the connection backlog (CB) to contain Π P-nodes, we need to ensure that there are, at any point in time, at least Π P-nodes in the PSS view. Second, for creating a WCL path, the source node must know the public keys of each mix on the path. The first modification relates to the view truncation after an exchange, while the second complements the PSS with a decentralized public key management service.

1) P-nodes Availability Enforcement: Upon a view exchange with a partner node, the PSS protocol decides which subset of the view to keep based on one of the following policies (see Section II).

We bias these three selection policies so as to keep a number of at least Π P-nodes in the view. If the original selection policy breaks this condition, we enforce that the Π P-nodes with the smallest age from the view and the received entries are kept even if they would have been discarded by the unbiased policy. Biasing the selection towards P-nodes may lead to an imbalance of the in-degree of P-nodes compared to the N-nodes. A result of this imbalance is that P-nodes will be more loaded than N-nodes. This risk may arise if the percentage of P-nodes that must be kept in the view is higher than the percentage of P-nodes in the network (e.g., if a node requires $\Pi = 3$ P-nodes in a view of size $c = 10$ while only 10% of the nodes are P-nodes). In order to limit the effect of a high value of Π on the load of P-nodes we again bias the selection process and discard in priority the oldest P-nodes that are above the Π threshold.

We will present in Section V an evaluation of the effect on the quality of the overlay produced by the PSS with and without the P-nodes availability bias. Results show that when Π is set to reasonable values (i.e., a subset of the view size proportional to the ratio of P-nodes vs. N-nodes in the system) the bias has a very small effect on clustering and in-degree balancing.

2) Public Key Management: In order to create the onion path for the WCL, we need to use the public keys of all nodes on the path (A, B, and D in our example). Therefore, each node must know the public key of the nodes that are in its connection backlog (CB), as well as the public keys of the destination node D and at least one potential next-to-last hop towards a P-node B.

To that end, we build a simple decentralized public key

management service on top of the gossip interactions of the PSS. Each time two nodes perform a gossip view exchange, they insert each other in their respective CB. To ensure that nodes know the public key for all entries in their CB, they additionally piggyback their key on the gossip exchange messages. Note that keys are also exchanged with the P-nodes that are explicitly contacted and inserted in the CB when the count of P-nodes falls below Π. This simple mechanism is sufficient for constructing the first part of the onion path, until the next-to-last hop. S learns the public keys of the last two nodes (D and B) from the gossip interactions over WCL channels, as we explain next.

IV. PPSS: The Private Peer Sampling Service

The private peer sampling service (PPSS) layer interacts with the WHISPER communication layer (WCL) to implement group-based peer sampling in a strictly confidential manner. The PPSS constructs a *private view* and ensures that a confidential communication is possible with any of its node by leveraging a WCL route. The PPSS is also in charge of maintaining a persistent connection to nodes of the view if required by the application (e.g., as part of a gossip-based overlay construction protocol [12], [13], [14], [15]).

The membership of the nodes composing the private group can only be known by the other members of the same group, and the content of the communications between group members cannot be seen by third party nodes, including relays used to bypass NAT limitations. Third party nodes cannot even infer that two communicating nodes belong to the same group. Finally, a node part of several private groups will not disclose its memberships to the members the other groups it belongs to: each group is managed separately by a different instance of the PPSS.

A. Private Group Management

For joining a private group, a node must first receive an invitation with a temporary signed accreditation and the identity of one entry point in the group. This invitation is initiated by another application running on the system-wide PSS (e.g., in the case of a decentralized VPN emulation), or sent by an external channel such as a Web interface, instant messaging, email, etc.

A private group comprises at least one *group leader* and is associated with a public/private key pair. All nodes in the group know the public key while only one or several leader(s) know the private key. When a new node wants to join a group, it must send its accreditation to one of the leaders. If the accreditation is considered valid (e.g., it can be signed by the group's key or by an external invitation manager), then the contacted leader sends back to the new node its *passport* and the group's public key. A passport is formed by the node's identifier signed with the group's private key. Nodes that belong to a private group ship their passport together with all the communications performed inside the group. When a node receives a message with an invalid passport, the message is simply ignored. This effectively prevents nodes from revealing to non-members their participation to private groups.

The leader(s) selection mechanism is application-dependent. Nonetheless, it is possible to prevent the impossibility of integrating new nodes when the leaders are all offline, by implementing a leader election mechanism based on a gossip-based aggregation [8] of a maximum proposed value: each node in the group that detects that periodic heartbeats sent by leaders are not received anymore can trigger the proposal of a value based on the hash of its identifier. After the aggregation protocol converges (in a few cycles), each node knows the highest proposed value(s) and therefore the new leader(s). These new leaders then propagate a new public key, signed by their identity. This new key is then used along with the previous ones (from a history of public group keys) to verify and issue passports.

B. Overlay Management

The PPSS refreshes a node's private view for a given group along the same line as how the PSS refreshes the system-wide views. The exchange of views is based on the same mechanism, but the exchanged entries contain additional information (not just the contact information and age used by the PSS). Indeed, as a node must be able to contact any node in its private view by means of a WCL path in order to proceed to the exchange, it first requires the following information for each node: the identity of the private group partner and its public key. Additionally, for each N-node we need to include a set of Π P-nodes (identities and public keys) that can be used as the next-to-last hop for the WCL path (for P-node we can use any of the P-nodes known, e.g., from the Π P-nodes of the N-nodes in the CB, to use as the next-to-last hop in a WCL route). This additional information is added to the entries of the node's private view. Therefore, upon gossip interaction, the exchanged entries contain both the identities of some private group nodes and the necessary information to contact them in a confidential manner.

Figure 4 presents an example of the state kept at a node A participating to a private group. A gossip-based exchange of view is initiated by node A selecting node Z as the partner for the exchange. Both A and Z are N-nodes. Node I in the CB of A is selected for the first step of the onion route. This hop in the path may require the message to transit through one or several relay nodes, depending on the nature of the NAT devices used by nodes A and I. The private view of A for the group includes both the N-node Z and $\Pi = 2$ public nodes that can communicate directly with Z. Node A selects part of its private view (the P-node C and the N-node Q in the example), generates a new entry with its own identity, and sends both components along with its passport to node Z using a WCL path. Node Z, after checking the passport of A, inserts the new elements in its private view and performs the same operation to send its reply to node A.

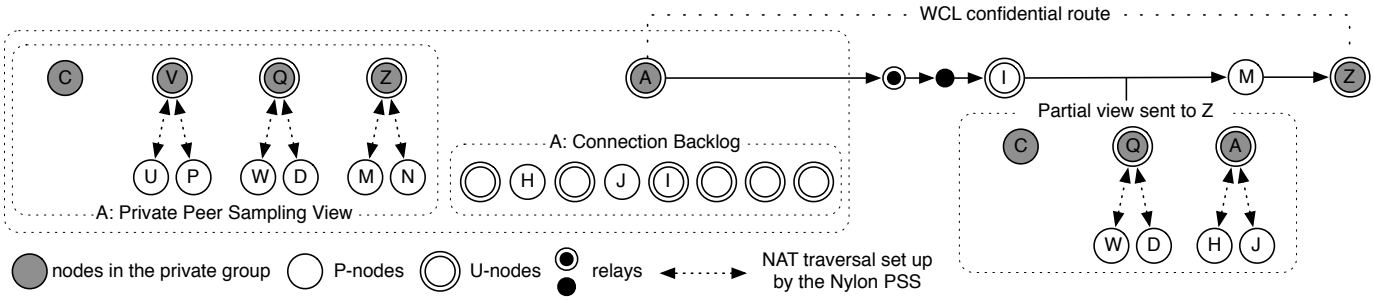

Fig. 4: WHISPER PPSS state at a node A participating to a private group and creating a WCL confidential path towards a gossip exchange partner node Z based on the information from its private view.

C. Persistent Paths

A node can contact the other nodes from a private group as long as they remain in its private view. The gossip-based peer sampling protocol operating on the views does not require connections between nodes to be persistent: only one-time exchanges are necessary to keep the network connected and provide nodes with fresh views of the private group. Nonetheless, applications typically require that some connections be made persistent. For instance, overlay creation protocol such as T-Man [12], GosSkip [13], Kelips [14] or T-Chord [15] maintain a separate view that contains nodes selected based on an application-dependent proximity criteria. These protocols are oblivious to the fact that the communication with gossip partners takes place using a confidentiality-enforcing mechanism like WHISPER. The only constraint is that they do not try to communicate directly with the other nodes (which would not be possible anyway in most cases due to NAT limitations). Instead, all communications must take place through the WHISPER layer.

A communication channel can be made persistent by means of a simple session mechanism. When a node in the private view is selected by the application as a persistently connected path, this node is kept in a private connection pool (PCP). Each node in the PCP is periodically contacted so as to refresh the Π P-nodes used by the WCL to communicate with it. This periodic refresh can occur at a lower frequency than gossip exchanges as it only depends on the duration of the NAT association rules. This mechanism ensures that communication remains possible and the path is kept persistent transparently to the application.

V. EVALUATION

We present in this section an evaluation of the WHISPER implementation over a cluster and on the PlanetLab testbed. We evaluate the system in a bottom-up approach, starting with the impact on the PSS of the enforcement of Π P-nodes in the view, the cost of public key management, the availability of WCL routes under varying levels of churn, and the bandwidth and computational costs of the the full WHISPER stack. We finally describe the performance of T-Chord [15], a gossip-constructed version of the Chord DHT [24], operating in a private group on top of the PPSS.

A. Experimental Settings

Our implementation leverages the SPLAY [25] framework, and uses a combination of Lua and C (for computationally intensive operations such as AES and RSA encryption and decryption). In order to support the experiments, we added a NAT emulation feature to SPLAY that was not described in the original paper [25]. We emulate the 4 major types of NAT devices, (`full_cone`, `restricted_cone`, `port_restricted_cone`, `sym`). Note that `sym` NAT devices require the use of relay nodes by the $\mathcal{N}ylon$ layer. To reflect real-world scenarios [4], we deployed 70% of the nodes behind NAT devices, evenly split between the four NAT types. Our emulation follows the RFC for TCP-friendly NAT (http://tools.ietf.org/html/rfc5382): filtering rules are registered to the emulated NAT devices on a per-connection (for TCP) or per-packet (UDP) basis.

We use the following two testbeds: (1) a local cluster of 22 computers equipped with a 2.2 GHz Core 2 Duo CPU and 2 GB of RAM, connected by a 1 Gbps switched network and supporting up to 1,000 WHISPER nodes; (2) a slice of 400 nodes on the global-scale PlanetLab testbed. For all experiments, we consider a PSS cycle time of 10 seconds and a PPSS cycle time of 1 minute.

B. Biased Peer Sampling Service

We start by evaluating the impact of the modifications to the PSS layer. As detailed in Section III, we modify the view truncation mechanism to ensure that Π P-nodes are kept at each node. We deploy 1,000 nodes on the cluster and experiment with values of $\Pi = 0$ (baseline, unmodified PSS) to $\Pi = 3$. The PSS view size is set to 10 nodes. Figure 5 presents the impact of our modification on the two metrics that characterize the resulting PSS graph: the clustering coefficient and the in-degree distribution. We observe that the impact on clustering (upper plot) is negligible. We also compared the distribution of clustering coefficients for N-nodes and P-nodes (not shown on the figure) and found them to be undistinguishable. As expected, the enforcement of Π P-nodes in the views results in a higher in-degree for the P-nodes, as illustrated by the two lower graphs. The choice of Π is indeed a compromise between load imbalance between P- and N-nodes and resilience to churn. In practice, a small value such

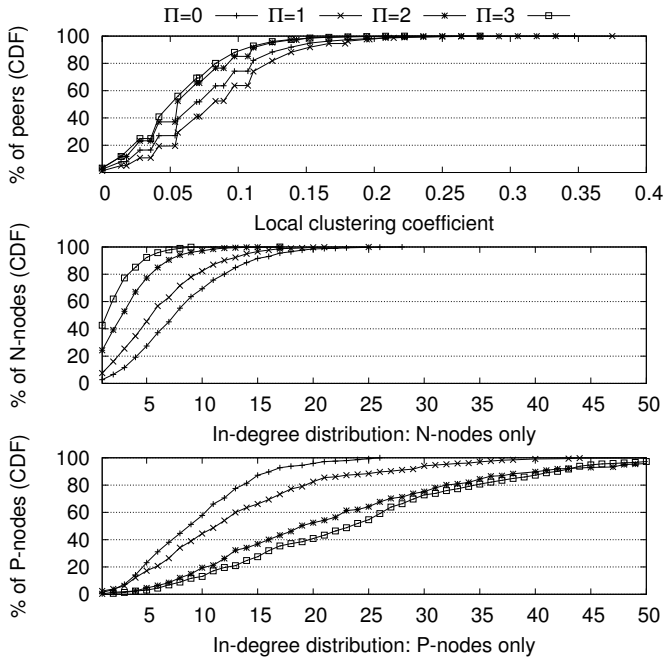

Fig. 5: Biased PSS: impact on clustering and in-degree distribution.

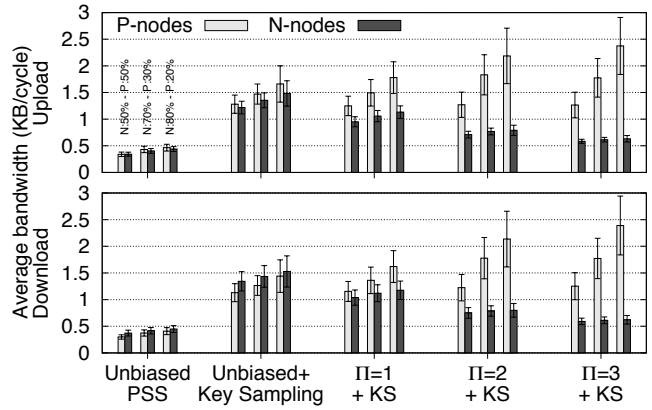

Fig. 6: Public Key Sampling Service: bandwidth costs (cumulative over 1,000 nodes).

as $\Pi = 2$ or $\Pi = 3$ is sufficient to allow the creation of WCL routes under high churn conditions, as we demonstrate later in this section.

C. Management Cost of Public Keys

We operate the public key management on top of the PSS exchanges. Each node initiates one exchange per cycle, henceforth sending and receiving one public key. In addition, each node receives on average one request from another node, with the probability of receiving a request depending on its in-degree. As a result of the uneven distribution of in-degrees, P-nodes are likely to have slightly more traffic related to public keys than N-nodes. Figure 6 presents the average bandwidth spent per cycle for various configurations: the unbiased PSS corresponds to $\Pi = 0$, with and without key exchanges. As expected, the bandwidth requirements for both types of nodes are balanced. As Π increases, the bandwidth requirement of P-node also increases, but they remain within very reasonable margins: a bandwidth of 2.5 KB per cycle corresponds to 256 bytes per second with our PSS cycle period of 10 seconds. We consider three configurations of P- to N-nodes ratios, 50%/50%, our default 70%/30% and 80%/20%. We observe that even in the most extreme case the load of P-nodes for public key management remains reasonable even for $\Pi = 3$.

Churn conditions	Success	Alt.	No alt.
No churn	100%	0%	0%
Churn rate: X=0.2% / minute *(30 leaves & 30 joins / 15 min.)*	98.3%	1.42%	0.28%
Churn rate: X=1% / minute *(150 leaves & 150 joins / 15 min.)*	96.7%	2.73%	0.47%
Churn rate: X=5% / minute *(750 leaves & 750 joins / 15 min.)*	96.5%	2.83%	0.77%
Churn rate: X=10% / minute *(1500 leaves & 1500 joins/15 min.)*	90.9%	7.86%	1.24%

```
1 from 0s to 30s join 1000
2 at 300s set replacement ratio to 100%
3 from 300s to 1200s const churn X% each 60s
4 at 1200s stop
```

TABLE I: Ratio of success on first attempt to construct a WCL route (**Success**), of failed attempts but where an *alternative* route is available (**Alt**) or when no such alternative route exists (**No Alt.**).

D. Availability of Anonymizing Routes under Churn

We now evaluate all WHISPER components together. Our first experiment studies the robustness of the WCL routes in presence of churn. We consider a set of 1,000 nodes (on average), each subscribing to one random group out of a set of 20 private groups, and we set $\Pi = 3$. We use the SPLAY's churn module [25] to inject node arrivals and departures in the system using the script presented at the bottom of Table I. We vary the amount of dynamism by changing the value X as detailed in the table, from no churn to a high level of churn: the leaving of 10% of the entire network per minute, and the joining of the same amount of new nodes (100% replacement ratio). Creating multiple hops paths in the presence of faults is the more critical aspect of WHISPER. Table I presents for increasingly-churned runs the following ratios. The ratio of success indicates how many of the WCL paths succeed first-hand, without the need to find an alternative path. We implement the automatic retry and distinguish between two ratios: when such an alternative path exists, and when it does not.[3] We observe that the success ratio without retry remains extremely high even with high level of churns, and that in a vast majority of cases it is possible to find such an alternative path. This is a result of the periodic shuffling of fresher entries both by the PSS and PPSS protocols. For a churn rate of X=1%, each of the 2.73% of paths that require an alternative,

[3]Note that we do not consider that the failure of the destination node is a WCL route failure. Failing to find a path to the destination after Π retries is actually considered by the PPSS layer as a failure of the destination, and the node removed from the private view.

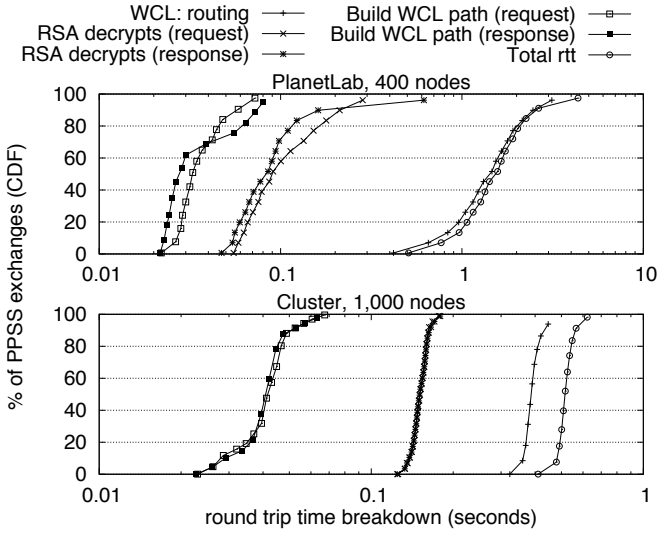

Fig. 7: Breakdown distribution of the round-trip-times of 1,500 private view exchanges by the PPSS over WCL channels. Exchanged views are at most 20 KB.

	AES	RSA	Total
N-node	625.9 μs (*< 0.001%*)	293.4 ms (*0.293%*)	294.0 ms (*0.294%*)
P-node	1,506 μs (*0.001%*)	626.1 ms (*0.626%*)	627.6 ms (*0.627%*)

TABLE II: Average CPU time per PPSS cycle used for encryptions and decryptions with AES and RSA. Percentages indicate the fraction of the 1 minute PPSS cycle a node spends on average for each operation type.

one tries on average 1.0 entry from the BL for the first mix and 1.12 entries from the Π P-nodes of the destination for the second mix to get a correct path (resp. 1.44 and 1.22 for X=5%). This accounts for a large number of successful first retries. In a minority of cases, no such alternative path can be found. This is vastly due to the case where no second mix can be found as the Π P-nodes known to reach the destination all failed. Higher level of alternative impossibility (such as the extreme 10% churn rate) can be mitigated with a larger value of Π (however increasing in-degree imbalance) or a smaller PPSS cycle (at the cost of increased bandwidth).

E. Delays of Anonymizing Routes

Our next experiment evaluates the costs of operating the WCL routes for view exchanges at the PPSS level. We present the delay breakdown in Figure 7 and the average computational loads in Table II.

We observe in Figure 7 that the communication latency is largely dominated by network delays. The time required to prepare the onion WCL path, including encryption for each mix, is nearly two orders of magnitudes less than network delays on a cluster as well as on PlanetLab. The decryption of the onion WCL path at each step similarly accounts for a small part of the overall delay. Even on heavily loaded PlanetLab machines with larger network delays and high message loss

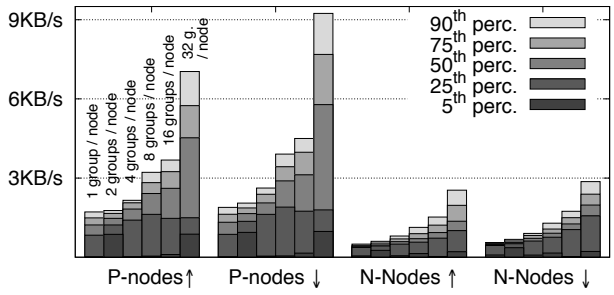

Fig. 8: Network bandwidth evolution as a function of the number of groups each node participates to (from a set of 120 groups, and for 400 nodes on PlanetLab).

rates, we manage to confidentially exchange private views within 2 seconds for more than 80% of the exchanges. On a cluster, all exchanges take less than 500 ms.

We detail the computational costs associated with encryption and decryption: AES is used to encode the content and RSA is used to encrypt the WCL path components. Table II presents the average processor time spent by N- and P-nodes during one cycle of the PPSS over a 1,000 nodes network in our cluster (that is, exactly 1,000 exchanges of PPSS views). The size of the exchanged data depends on the ratio of P-nodes in the PPSS view exchanged (as these nodes do not need to be complemented by Π P-nodes as connection means). Each node sends 5 entries of PPSS view. Each N-node entry comes with $\Pi = 3$ P-nodes. Since the size of public keys is 1KB, the maximum size of exchanged subsets of views is thus $\sim$20KB. The computational costs of other operations is completely negligible compared to the cost of encryptions and decryptions, hence we ignore it in the table. As expected, the cost of AES operations (which depends on the size of the message being sent over the WCL path) is very low compared to the cost of RSA (which is independent from that size). We observe that the load on P-node is about 2.13$\times$ higher than for N-nodes, but remains very low: less than 0.65% of one PPSS cycle is spent on the processor for encryption and decryption operations. The reason is that, due to the construction of WCL paths, P-nodes are more likely to act as mixes and hence spend about 4.12$\times$ more CPU time for RSA decryptions.

F. Bandwidth Consumption vs. Number of Private Groups

Our last benchmark of the WHISPER middleware framework evaluates the evolution of the network costs when the number of subscriptions of nodes to private groups increases. We consider 400 nodes on PlanetLab, operating a total of 120 private groups (each P-node creates, and acts as a leader for, one private group). We vary the number of subscribed group per peer from 1 to 32 in a logarithmic fashion. Figure 8 presents the evolution of the distribution of upload ($\uparrow$) and download ($\downarrow$) bandwidth requirements. We use stacked percentiles with shades of grey to represent a distribution. We observe that, consistently with the values in Table II, the bandwidth costs are more important for P-nodes due to their more important

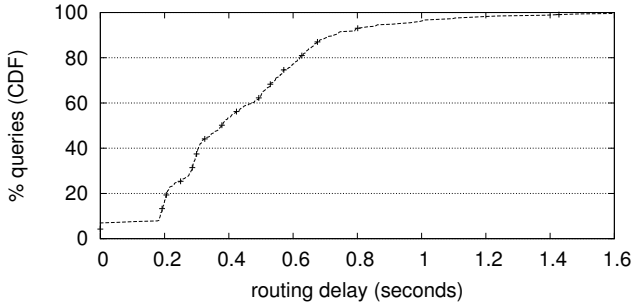

Fig. 9: Distribution of T-Chord routing delays for a 60-nodes private group amongst a 400 nodes cluster.

role, yet they remains within reasonable values. As expected, the bandwidth requirements increase linearly with the number of subscribed groups per peer, and the number of views to maintain.

G. Application of WHISPER: private T-Chord DHT

Our final experiment demonstrates the capacity of WHISPER to support the operation of applications and higher-level protocols amongst the members of a private group, so as to transparently benefit from confidentiality and secrecy.

We consider a system of 400 nodes on our cluster. Within this group, 60 nodes wish to operate a private index based on a DHT (e.g., to share the location of sensitive data). These nodes bootstrap, within their private group, a Chord DHT overlay [24]. We use the T-Chord [15] gossip-based construction of the Chord ring based on the T-Man overlay construction framework [12]. T-Chord constructs the Chord ring in a self-organizing manner, based on view exchanges with nodes obtained from a PSS (here, from the PPSS private view) and with nodes they are linked to in the Chord ring itself (here, the nodes with which a persistent connection has been opened by WHISPER). Upon view exchange the nodes select, for each type of link in the Chord ring, the entries that best match a proximity criteria (e.g., the closest node counterclockwise for the predecessor node) to implement view truncation. T-Chord converges in a few cycles to the perfect Chord ring.

Figure 9 presents the routing delays for 350 random queries in the Chord ring. The reply to the querying node must be made through the WCL layer to protect confidentiality. However, the destination node may not have the querying node in its view. Therefore the querying node ships its contact information with the query (its identity and public key, as well as those of Π P-nodes if necessary) so that it can be contacted back by the destination node with a single WCL path. Delays range from 190 ms (smaller delays are for keys held by the querying node) to about 1.5 seconds depending on the length of the routes. Although WHISPER performs extra processing compared to a plain Chord implementation, the lookup latency remains within acceptable limits considering the strong confidentiality guarantees that are provided to the application.

VI. RELATED WORK

Our contribution relates to work done in the context of anonymizing systems, security and practical implementation aspects of gossip-based protocols, and privacy-preserving gossiping.

Anonymity in networked systems has been first studied in the context of mail anonymization, with Chaum's proposal of using chains of mixes [3], later followed by its real-time variant called onion routing [2]. WHISPER's WCL leverages such onion routes to provide message confidentiality and relationship anonymity in large-scale networks. TOR [1] is a deployed set of anonymizing servers allowing users to browse the Internet anonymously. Cashmere [26] implements each mix using a group of servers for increased robustness. Other examples include AP3 [27], BiFrost [28], SPADS [29], and Tarzan [30]. All these systems consider a set of dedicated servers playing the role of mixes, while WHISPER instead leverages regular nodes in a self-organizing manner and does not require any dedicated infrastructure.

Alternative approaches to the use of mixes include multiple-path routing and network coding mechanisms such as *information slicing* [31], as well as systems based on the dining cryptographers approach such as Herbivore [32] and P5 [33].

Most of research on security aspects of gossip-based protocols have focused on tolerating byzantine faults. Examples include dissemination protocols, e.g., BAR gossip [7] or StarblabIT [34], and byzantine-resilient PSS protocols, e.g., the *secure peer sampling* [35], Brahms [36], Fireflies [37] and PuppetCast [38]. These mechanisms are complementary to WHISPER: we do not consider that nodes act maliciously in operating the protocol but only wish to spy upon other nodes. If we relax this assumption and consider malicious protocol modifications, WHISPER could be combined with one of these solutions to tolerate byzantine faults.

Practical implementations of the PSS for large-scale networks where a majority of nodes are behind NATs or firewalls include $\mathcal{N}ylon$ [21], upon which we base the design of WHISPER. In [39], Leitao *et al.* propose an alternative approach where nodes do not use NAT traversal to create communication paths but leverage P-nodes as rendez-vous nodes for forwarding gossip exchanges with other nodes.

The *Gossple anonymous social network* [40] proposes to build a decentralized social network, in which nodes with similar interests are linked in a dynamically evolving network, constructed using gossip techniques. It proposes to implement some level of anonymity by means of compact representations of the user profiles and by using gossip-on-behalf: nodes can delegate the management of their view and profile to a third-party node, making the mapping of communicating partners and groups a difficult (but not impossible) task.

VII. CONCLUSION

In this paper we have presented WHISPER, a system that supports confidential communications and private group membership through the use of relationship anonymity in large-scale networks, even when the underlying communication

layer is not safe and third party nodes need to be used for bypassing connection limitations such as NAT traversal restrictions. WHISPER is decentralized and self-organizing, and presents a standardized peer sampling interface to other protocols and applications. It does not require any trusted third party node. It allows nodes to create groups and invite other nodes to join, while ensuring that communications between the nodes of the group will not be seen by other nodes, and that the very existence of the group and its members is also kept confidential. We evaluated WHISPER using a deployed prototype and our experiments convey its ability to provide strong confidentiality guarantees with reasonable computational and communication costs.

REFERENCES

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Security'04: 13th UXENIX Security Symposium*, San Diego, CA, USA, aug 2004, pp. 303–320.

[2] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," in *First International Workshop on Information Hiding*, 1996, pp. 137–150.

[3] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[4] M. Casado and M. J. Freedman, "Peering through the shroud: The effect of edge opacity on IP-based client identification," in *Proc. 4th Symposium on Networked Systems Design and Implementation (NSDI 07)*, Cambridge, MA, Apr. 2007.

[5] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, "Lightweight probabilistic broadcast," *ACM Transactions on Computer Systems*, vol. 21, no. 4, pp. 341–374, 2003.

[6] W. Vogels, R. van Renesse, and K. Birman, "The power of epidemics: robust communication for large-scale distributed systems," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 131–135, 2003.

[7] H. Li, A. Clement, E. Wong, J. Napper, L. Alvisi, and M. Dahlin, "Bar gossip," in *Proc. of 7th Symposium on Operating System Design and Implementation (OSDI '06)*, 2006.

[8] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems*, vol. 23, no. 3, pp. 219–252, 2005.

[9] S. Kashyap, S. Deb, K. V. M. Naidu, R. Rastogi, and A. Srinivasan, "Efficient gossip-based aggregate computation," in *PODS '06: Proceedings of the twenty-fifth symposium on Principles of database systems*. New York, NY, USA: ACM Press, 2006, pp. 308–317.

[10] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 2003, p. 482.

[11] D. Kostoulas, D. Psaltoulis, I. Gupta, K. P. Birman, and A. J. Demers, "Active and passive techniques for group size estimation in large-scale and dynamic distributed systems," *Journal of Systems and Software*, vol. 80, no. 10, pp. 1639–1658, 2007.

[12] M. Jelasity, A. Montresor, and O. Babaoglu, "T-man: Gossip-based fast overlay topology construction," *Computer Networks*, vol. 53, no. 13, pp. 2321–2339, aug 2009.

[13] R. Guerraoui, S. B. Handurukande, K. Huguenin, A.-M. Kermarrec, F. L. Fessant, and E. Rivière, "GosSkip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles," in *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, Sep. 2006.

[14] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse, "Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead," in *Proceedings of IPTPS 2003*, 2003.

[15] A. Montresor, M. Jelasity, and O. Babaoglu, "Chord on demand," in *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, Washington, DC, USA, 2005.

[16] R. van Renesse, Y. Minsky, and M. Hayden, "A gossip-style failure detection service," in *Proc. of Middleware, the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, IFIP, Ed., The Lake District, UK, 1998, pp. 55–70.

[17] A. Fernandez, V. Gramoli, E. Jimenez, A.-M. Kermarrec, and M. Raynal, "Distributed slicing in dynamic systems," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS'07)*. Toronto, Ontario, Canada: IEEE Computer Society, jun 2007.

[18] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems*, vol. 25, no. 3, aug 2007.

[19] S. Voulgaris, D. Gavidia, and M. van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, Jun. 2005.

[20] R. Roverso, S. El-Ansary, and S. Haridi, "Natcracker: Nat combinations matter," in *Proc. of ICCN'09: 18th International Conference on Computer Communications and Networks*, vol. 0, 2009.

[21] A.-M. Kermarrec, A. Pace, V. Quema, and V. Schiavoni, "Nat-resilient gossip peer sampling," *Proceedings of the International Conference on Distributed Computing Systems (ICDCS'09)*, pp. 360–367, 2009.

[22] CISCO IP NAT translation timeouts specification: http://www.cisco.com/en/US/docs/ios/ipaddr/command/reference/iad_nat.html#wp1076124.

[23] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators," in *Proceedings of the annual USENIX Technical Conference*, Berkeley, CA, USA, 2005.

[24] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM '01*, Aug. 2001.

[25] L. Leonini, E. Rivière, and P. Felber, "SPLAY: Distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze)," in *NSDI'09: Proceedings of the 6th Symposium on Networked Systems Design and Implementation*. USENIX, apr 2009, pp. 185–198.

[26] L. Zhuang, F. Zhou, B. Y. Zhao, and A. Rowstron, "Cashmere: resilient anonymous routing," in *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, 2005.

[27] A. Mislove, G. Oberoi, A. Post, C. Reis, P. Druschel, and D. S. Wallach, "AP3: cooperative, decentralized anonymous communication," in *Proceedings of the 11th ACM SIGOPS European workshop*, 2004.

[28] M. Kondo, S. Saito, K. Ishiguro, H. Tanaka, and H. Matsuo, "Bifrost: A novel anonymous communication system with DHT," in *Proceddings of PDCAT '09*, 2009, pp. 324–329.

[29] P. Felber, M. Rajman, E. Rivière, V. Schiavoni, and J. Valerio, "SPADS: Publisher anonymization for DHT storage," in *IEEE P2P'10: 10th IEEE International Conference on Peer-to-Peer Computing*.

[30] M. Freedman, E. Sit, J. Cates, and R. Morris, "Introducing tarzan, a peer-to-peer anonymizing network layer," *Peer-to-Peer Systems*, 2002.

[31] S. Katti, J. Cohen, and D. Katabi, "Information slicing: Anonymity using unreliable overlays," in *NSDI'07: Proceedings of the 4th conference on Networked Systems Design & Implementation*, 2007.

[32] S. Goel, M. Robson, M. Polte, and E. G. Sirer, "Herbivore: A scalable and efficient protocol for anonymous communication," Cornell University Comp. and Inf. Science, Tech. Rep. TR2003-1890, feb 2003.

[33] R. Sherwood, B. Bhattacharjee, and A. Srinivasan, "P5: a protocol for scalable anonymous communication," *J. Comput. Secur.*, vol. 13, pp. 839–876, December 2005.

[34] K. P. Kihlstrom and R. S. Elliott, "Performance of an intrusion-tolerant gossip protocol," in *PDCS'09:21st IASTED International Conference on Parallel and Distributed Computing and Systems*.

[35] G. P. Jesi, A. Montresor, and M. van Steen, "Secure Peer Sampling," *Elsevier Computer Networks - Special Issue on Collaborative Peer-to-Peer Systems*, vol. 54, no. 12, pp. 2086–2098, 2010.

[36] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer, "Brahms: Byzantine resilient random membership sampling," *Computer Networks*, vol. 53, no. 13, pp. 2340–2359, aug 2009.

[37] H. Johansen, A. Allavena, and R. van Renesse, "Fireflies: scalable support for intrusion-tolerant network overlays," in *EuroSys'06: 1st ACM SIGOPS European Conference on Computer Systems*.

[38] A. Bakker and M. van Steen, "Puppetcast: A secure peer sampling protocol," in *Proc. of the European Conference on Computer Network Defense (EC2ND 2008)*, Dublin, Ireland, dec 2008, pp. 3–10.

[39] J. Leitão, R. van Renesse, and L. Rodrigues, "Balancing gossip exchanges in networks with firewalls," in *IPTPS'10: 9th international workshop on Peer-to-peer systems*.

[40] M. Bertier, D. Frey, R. Guerraoui, A.-M. Kermarrec, and V. Leroy, "The Gossple anonymous social network," in *Middleware'10: ACM/IFIP/USENIX 11th Middleware Conference*, Bangalore, India, dec 2010.