How Far Removed Are You? Scalable Privacy-Preserving Estimation of Social Path Length with Social PaL

Marcin Nagy
Aalto University
marcin.nagy@aalto.fi

Thanh Bui
Aalto University
thanh.bui@aalto.fi

Emiliano De Cristofaro
University College London
e.decristofaro@ucl.ac.uk

N. Asokan
Aalto University and
University of Helsinki
asokan@acm.org

Jörg Ott
Aalto University
jorg.ott@aalto.fi

Ahmad-Reza Sadeghi
TU Darmstadt/CASED
ahmad.sadeghi@trust.cased.de

ABSTRACT

Social relationships are a natural basis on which humans make trust decisions. Online Social Networks (OSNs) are increasingly often used to let users base trust decisions on the existence and the strength of social relationships. While most OSNs allow users to discover the length of the social path to other users, they do so in a centralized way, thus requiring them to rely on the service provider and reveal their interest in each other.

This paper presents Social PaL, a system supporting the privacy-preserving discovery of arbitrary-length social paths between any two social network users. We overcome the bootstrapping problem encountered in all related prior work, demonstrating that Social PaL allows its users to find all paths of length two and to discover a significant fraction of longer paths, even when only a small fraction of OSN users is in the Social PaL system – e.g., discovering 70% of all paths with only 40% of the users. We implement Social PaL using a scalable server-side architecture and a modular Android client library, allowing developers to seamlessly integrate it into their apps.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Network**]: Distributed Systems—*Distributed applications*

Keywords

Privacy, Mobile Social Networks, Proximity

1. INTRODUCTION

The ability to learn the social path length to other social network users can often help individuals make informed trust and access control decisions. For instance, if attendees at a large convention could easily find other attendees with whom they share social links (e.g., a LinkedIn connection), this could help them decide who to chat or meet up with. Similarly, travelers and commuters could more consciously decide with whom to interact, share rides, etc. In general, discovering the social path length between users is beneficial in many interesting scenarios, such as estimating the *familiarity* to a location (which can in turn be used for context-based security [25]), as well as for routing in delay-tolerant ad-hoc mobile networks [6] and anonymous communications [17, 27].

Problem Statement. The widespread adoption of Online Social Networks (OSNs) makes it appealing to measure the length of a path between two nodes, e.g., to use this information as a signal of reciprocal trust and/or social interest. Today, a Facebook user can see the number of common friends with another user, while LinkedIn also displays the social path length. However, as popular OSNs are centralized systems, so are the features they offer to discover social paths. As such, they do not particularly adapt well to mobile environments where social interactions are tied to physical proximity, thus severely limiting the feasibility of many applications scenarios—users may not always be able to connect to the Internet or willing to reveal their location and/or interests to the provider. Relying on centralized systems to learn social path lengths implies that, whenever Alice queries a server for the social path length to Bob, the server learns that Alice is interested in Bob, their frequency of interactions, and their locations.

This prompts the need for decentralized and privacy-preserving techniques for social path length estimation. Users should only learn if they have any common friends, without having to reciprocally reveal the identities of friends that they do not share, and discover the length of the social path between them (for paths longer than two), without learning which users are in the path.

Technical Roadmap. Our work builds on Common Friends [30], a system supporting privacy-preserving common friend discovery on mobile devices: building on a cryptographic primitive called Private Set Intersection (PSI) [10], it allows mutual friends to be discovered by securely computing the intersection of friendship *capabilities*, which are periodically distributed from a Common Friends user to all the friends who are also using it. However, besides being limited to social paths of length two, Common Friends only discovers the subset of the mutual friends who are *already* in the system, thus suffering from an inherent bootstrapping problem.

This paper introduces Social PaL, the first system that supports the privacy-preserving estimation of the social path length between any two social network users. We introduce the notion of *ersatz nodes*, created for users that are direct friends of one or more users of Social PaL but are not part of the system. We guarantee that two users of Social PaL will be able to discover *all* their common

WISEC'15, June 21–26, 2015, New York, NY, USA

http://dx.doi.org/10.1145/2766498.2766501.

friends in the OSN (i.e., all paths of length two). We then present a hash chain-based protocol that supports the (private) discovery of social paths longer than two, and demonstrate its effectiveness by means of extensive simulations.

Our work is not limited to designing protocols: we also fully implement `Social PaL` and deploy a scalable server architecture and an Android client library enabling developers to seamlessly integrate it into their applications.

Contributions. In summary, we make the following contributions:

1. We present an efficient privacy-preserving estimation of social paths of arbitrary length (Section 3).
2. We state and prove several properties of `Social PaL` including the fact that, for two users A and B: (i) `Social PaL` will find all common friends of A and B, including those who are not using it, and (ii) for each discovered path between A and B, `Social PaL` allows each party to compute the *exact* length of the path (Section 4).
3. Using samples of the Facebook graph, we empirically show that even when only 40% of users use the system, `Social PaL` will discover more than 70% of all paths, and over 40% with just 20% of the users (Section 5).
4. We support Facebook and LinkedIn integration and release the implementation of a scalable server-side architecture and a modular Android client library, allowing developers to easily integrate `Social PaL` in their applications (Section 6).
5. We build two Android apps: a friend radar displaying the social path length to nearby users, and a tethering app enabling users to securely share their Internet connection with people with whom they share mutual friends (see Section 7).

2. BACKGROUND

2.1 Private Discovery of Common Friends

We start by discussing the problem of privately discovering common friends, i.e., social paths of length two. We argue that minimal security properties for this problem include both *privacy* and *authenticity*, as users should neither learn the identity of non-shared friends nor claim non-existent friendships.

Private Set Intersection (PSI) [10]. A straightforward approach for privately discovering common friends is to rely on PSI, a primitive allowing two parties to learn the intersection of their respective sets (and nothing else). If friend lists are encoded as sets, then PSI could be used to privately find common friends as the set intersection. One could also limit disclosure to the *number*, but not the identity, of shared friends, using Private Set Intersection Cardinality (PSI-CA) [7], which only reveals the size of the intersection. However, using PSI (or PSI-CA) guarantees privacy but not authenticity, as one cannot prevent users from inserting arbitrary users in their sets and claim non-existent relationships.

Bearer Capabilities. In order to guarantee authenticity, Nagy et al. [30] combine bearer capabilities [36] (aka bearer tokens) with PSI, proposing the `Common Friends` service, whereby users generate (and periodically refresh) a random number – the "capability" – and distribute it to their friends via an authentic and confidential channel. As possession of a capability serves as a proof of friendship, users can input it into the PSI/PSI-CA protocol, thereby only learning the identity/number of common and authentic friends.

Since capabilities are large random values, a simpler variant of PSI for private common friend discovery that only relies on cryptographic hash functions and does not involve public-key operations

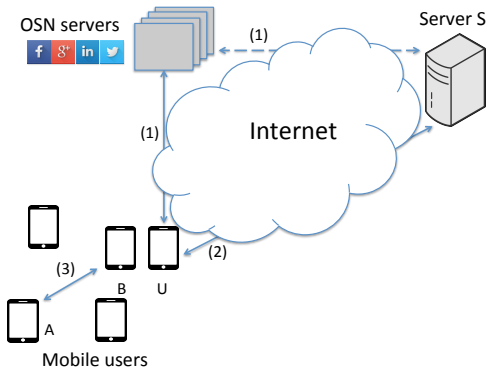


Figure 1: Overview of the `Common Friends` architecture, involving three protocols: (1) OSN user authentication protocol, (2) `Common Friends` capability distribution protocol, (3) `Common Friends` discovery protocol.

can be used.[1] Parties can hash each item in their set and transfer the hash outputs: since the hash is one-way, parties cannot invert it, thus they only learn the set intersection upon finding matching hashes.[2] This can be further optimized using the ***Bloom Filter based PSI (BFPSI)*** primitive (for high-entropy items) outlined in [30].

Bloom Filters. A Bloom Filter (BF) is a compact data structure for probabilistic set membership testing [4]. Specifically, a BF is an array of β bits that can be used to represent a set of α elements in a space-efficient way. BFs introduce no false negatives but can have false positives, even though the probability of a false positive can be estimated (and bounded) as a function of α and β.

Formally, let $X = \{x_1, \ldots, x_\alpha\}$ be a set of α elements, and BF be an array of β bits initialized to 0. BF(j) denotes the j-th item in BF. Then, let $\{h_i : \{0,1\}^* \rightarrow [1,\beta]\}_{i=1}^{\gamma}$ be γ independent cryptographic hash functions, salted with random periodically refreshed nonces. For each element $x \in X$, set $BF(h_i(x)) = 1$ for $1 \leq i \leq \gamma$. To test if $y \in X$, check if BF$(h_i(y)) = 1\ \forall i$. An item appears to be in a set even though it was never inserted in the BF (i.e., a false positive occurs) with probability $p = (1 - (1 - 1/\beta)^{\gamma \cdot \alpha})^{\gamma}$. The optimal size of the filter, for a desired probability p, can be estimated as: $\beta = \lceil (-\log_2 p)/(\ln 2) \rceil \times \alpha$.

2.2 `Common Friends`

In Figure 1, we illustrate the `Common Friends` [30] system: it involves a server S, a set of OSN servers (such as Facebook or LinkedIn), and a set of mobile users, members of one or more of these OSNs. S is implemented as a social network app (i.e., a third-party server), which stores the bearer capabilities uploaded by the `Common Friends` application running on users' devices. It also allows a user's `Common Friends` application to download bearer capabilities uploaded by that user's friends in the OSNs. `Common Friends` consists of three protocols: (1) user authentication, (2) capability distribution, and (3) common friend discovery.

OSN User Authentication enables the OSN server to authenticate a user U, provide U's OSN identifier (ID_U) to S, and U to authorize S to access information about U's friends in the OSN, which can be done using standard mechanisms, such as OAuth [14].

[1] However, it is not clear whether it is possible to do so for PSI-CA, i.e., to only count the number of common friends.

[2] On the other hand, if sets contained low-entropy items, a malicious party could (passively) check whether any item item is in counterpart's set, independently of whether or not it lies in the intersection.

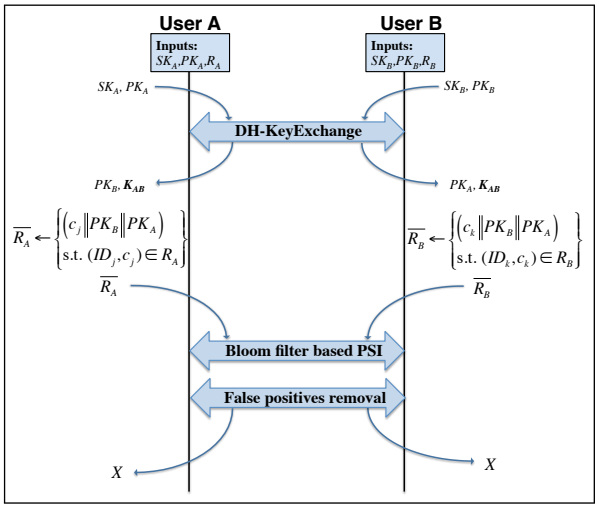
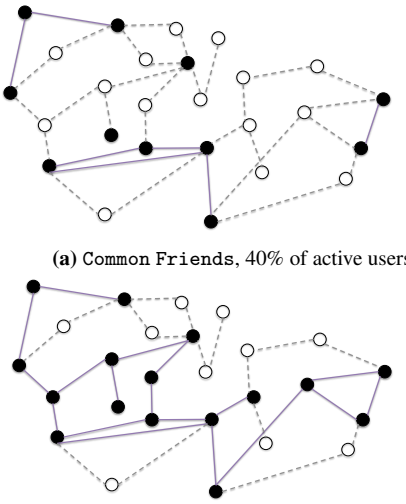


Figure 2: Common friend discovery protocol based on BFPSI and bearer capabilities from [30].



(a) `Common Friends`, 40% of active users

(b) `Common Friends`, 60% of active users

Figure 3: Coverage of `Common Friends` with 40% and 60% of users using the system. Black (resp., white) nodes denote users (resp., non-users) of the system. Purple/solid edges denote a direct friendship discoverable by `Common Friends`.

Capability Distribution involves S and U, communicating over a secure channel with server authentication provided by certificate $Cert_S$, and client authentication based on the previous OSN user authentication process. User U generates a random capability c_U (taken from a large space) and uploads it to S over the established channel. S stores c_U along with the social network user identifier ID_U, and sends back $R_U = \{(ID_j, c_j) : ID_j \in friends(ID_U)\}$, which contains pairs of identifiers and corresponding capabilities of each friend that has already uploaded his capability. The protocol needs to be run periodically to keep R_U up-to-date, as capabilities are periodically refreshed.

Common Friend Discovery is a protocol run between two users, A and B, illustrated it in Figure 2, allowing A and B to privately discover their common (authentic) friends, based on BFPSI. First, A and B exchange their public keys (PK_A and PK_B, respectively) and generate a shared key (K_{AB}) used to encrypt messages exchanged as part of the protocol. To prevent man-in-the-middle attacks, A (resp., B) cryptographically binds the DH channel to the protocol instance: A (resp., B) extends each item in the capability set R_A (resp., R_B) by appending DH public keys PK_A, PK_B, building effectively a new set $\overline{R_A}$ (resp., $\overline{R_B}$). A inserts every element of the $\overline{R_A}$ set into a Bloom Filter BF_A and sends it to B. B discovers the set of friends (X') in common with A by verifying whether each item of his input set $\overline{R_B}$ is in BF_A. Since Bloom Filters introduce false positives, the set X' may contain non-common friends. Thus a simple challenge-response protocol is run, where B requires A to prove knowledge of items available in X'. At the end of the protocol, A and B output the set of their common friends X.

3. SOCIAL PAL

We now present the design and the instantiation of the `Social PaL`, the system to compute the social path length between two OSN users in a decentralized and privacy-preserving way.

3.1 System Design

Limitations of [30]. Before introducing `Social PaL`'s requirements, we discuss two main limitations of `Common Friends` [30], as addressing them constitute our starting point:

1. *Bootstrapping*: Users A and B can discover a mutual friend (say C), only if C has joined the `Common Friends` system and uploaded his capability to S. That is, `Common Friends` will only discover a subset of the mutual friends between A and B until all of them start using it.
2. *Longer social paths*: `Common Friends` only allows its users to learn whether they are friends or have mutual friends. If two users have a longer social path between them, `Common Friends` cannot detect it.

To illustrate `Common Friends`'s bootstrapping problem, we plot, in Figure 3, a simple social network with 27 nodes (i.e., users) and 34 edges (i.e., friendship relationships). Black circles represent users who are using `Common Friends`, and white circles – those who are not. Purple/solid edges represent direct friend relationships (i.e., social paths of length 1) that are discoverable by `Common Friends`. When the user base is only 40% of all OSN users (Figure 3(a)), only 7 out of the 34 direct friend relationships are discoverable (i.e., coverage is approximately 20%). When it increases to 60%, coverage increases to about 50% (Figure 3(b)).

System model. `Social PaL`'s system model is the same as that of `Common Friends` (presented in Section 2.2). It involves a server S (which we design as a social network app), a set of OSN servers (such as Facebook or LinkedIn), and a set of mobile users members of one or more of these OSNs (cf. Figure 1).

Functional requirements. Ideally, `Social PaL` should allow any two users to always compute the exact length of the social path between them, even when `Social PaL` is being used only by a fraction of OSN users. In order to characterize how well `Social PaL` meets this requirement, we use a measure of the likelihood that any two users would discover, using `Social PaL`, an existing social path of a given length between them. We denote this measure as `Social PaL`'s *coverage*.

We define `Social PaL`'s functional requirements as follows:

- *(Correctness).* Users A and B can determine the exact length of a social path between them (if any).
- *(Coverage Maximization).* `Social PaL` should maximize coverage, in other words, the ratio between the number of social paths (of length n) between A and B *discovered* by `Social PaL` and the number of *all* social paths (of length n) between A and B.

Symbol	Description
Entities	
S	Server
A, B	User A, B, resp.
U	Generic User (can be either A or B)
E	User E (ersatz node)
Social graph data	
ID_{U}	Social identifier of U
$F(ID_{\mathsf{U}})$	Set of direct friends of U
$F^k(ID_{\mathsf{U}})$	Set of social contacts k hops from U
Keys	
PK_A, PK_B	DH public key of A, B, resp.
K_{AB}	DH session key between A and B
Cryptographic functions	
$h^i(x)$	Hash chain of item x of length i
`Social PaL` *protocol data*	
c_j	Capability uploaded by user with identifier ID_j
c_j^k	Capability of degree k uploaded by user with identifier ID_j
R_U	Set of capabilities downloaded by U from S
R_U^h	Set of higher order capabilities downloaded by U from S
R_U^d	Set of derived higher order capabilities downloaded by U from S
I	Union of capabilities' sets R_U, R_U^h, R_U^d
$\overline{I}$	Input sets to `Social PaL` discovery protocol
BF_A	Bloom filter sent by A

Table 1: Notation.

Privacy requirements. From a privacy point of view, `Social PaL` should satisfy the following requirements. Let A and B be two `Social PaL` users willing to discover the length of the social path existing between them:

1. A and B discover the set of their common friends but learn nothing about their non-mutual friends;
2. A and B do not learn any more information other than what it is already available from standard OSN interfaces.

In other words, `Social PaL` should allow two users to learn the social path length between them (if any), but not the nodes on the path, without reciprocally revealing their social link. If a path between the users exists that is of length two (i.e., users have some common friends), then they learn the identity of the common friends (and nothing else). This only pertains to interacting users, as ensuring that no eavesdropping party learn any information about users' friends can be achieved by letting users communicate via confidential and authentic channels.

Threat model. We assume that the participants in `Social PaL` are honest-but-curious. The OSN server is trusted to correctly authenticate OSN users and not to attempt posing as any OSN user. The `Social PaL` server S is trusted to distribute `Social PaL` capabilities only to those `Social PaL` users authorized to receive them. `Social PaL` users use the legitimate `Social PaL` client,[3] but they might attempt to learn as much information as possible about friends of other `Social PaL` users with whom they interact. We aim to guarantee the privacy requirements discussed above in this setting, and prevent the OSN server or the server S to learn any information about interactions between `Social PaL` users.

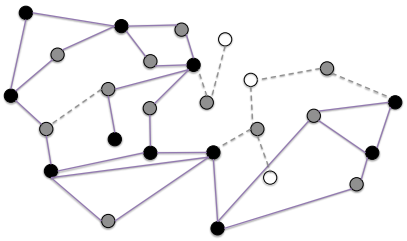
3.2 Bootstrapping `Social Pal`

Before presenting the details of the system, in Table 1, we introduce some notation used throughout the rest of the paper.

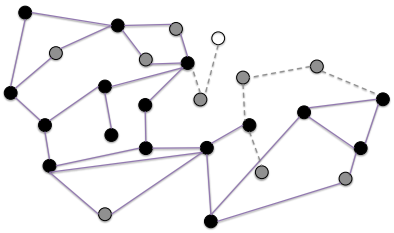
Ersatz nodes. One fundamental building block of `Social PaL` are ersatz nodes[4], which we introduce to overcome the bootstrapping problem faced by services like `Common Friends`. Recall from Section 2.2 that, in the original `Common Friends` design, the server

[3]This is enforced by the OSN app interfaces which ensure that only designated client apps are allowed to talk to a particular OSN app server, i.e., the `Social PaL` server S.

[4]The word *ersatz*, originally from German, means "substitute."



(a) `Social PaL`, 40% of active users with ersatz nodes



(b) `Social PaL`, 60% of active users with ersatz nodes

Figure 4: Coverage of `Social PaL` with 40% and 60% of users using the system and with the addition of ersatz nodes (in grey). Purple/solid edges here denote a direct friendship discoverable by `Social PaL`.

S stores bearer capability c_U, uploaded by a user U, together with his social network identifier ID_{U}. The pair (ID_{U}, c_U) constitutes U's *user node* in the social graph maintained by S. The set of U's friends $F(ID_{\mathsf{U}})$ is the set of edges incident in the user node.

In `Social PaL`, we let S create an *ersatz node* for all users who have not joined the system but who are friends with a user who has. An ersatz node is identical to a standard user node, but its capability is generated by S, instead of the user. Figures 4(a) and 4(b) show how coverage improves when ersatz nodes are added, e.g., with only 40% joining the system, coverage reaches 75%.

Ersatz node creation. Adding ersatz nodes requires a few changes in the capability distribution protocol, compared to that from Section 2.2. We highlight these changes in Figure 5, specifically, in the blue-shaded box. Before returning R_U to U, S first computes the set $M_U = \{ID_{\mathsf{E}} : \neg\exists(ID_{\mathsf{E}}, c_{\mathsf{E}})\}$ which contains the social network identifier of each "missing user" E.

Then, $\forall ID_{\mathsf{E}} \in M_U$, it creates E's an ersatz node as follows:

1. Create an *ersatz capability* $c_{\mathsf{E}} \in_R \{0,1\}^l$ (where l is the length of a capability) for E and store $\{(ID_{\mathsf{E}}, c_{\mathsf{E}})\}$.
2. Create an initial friend set $F(ID_{\mathsf{E}})$, which at this stage contains only ID_{U}.

After the successful creation of all needed ersatz nodes, S returns R_U, which includes the capabilities from the nodes of all of U's friends, including the ersatz nodes.

Active social graph updates. Users of `Social PaL` explicitly authorize the server S to retrieve their friend lists from the OSNs. Since an ersatz node E is not a user of `Social PaL`, S cannot learn the full set of E's friends $F(ID_{\mathsf{E}})$. Instead, it maintains an estimate of $F(ID_{\mathsf{E}})$ based on the events it can observe from users of `Social PaL`. For example, when a user U adds E as a friend, S learns that ID_{E} is added to $F(ID_{\mathsf{U}})$ and can infer that ID_{U} should be added to $F(ID_{\mathsf{E}})$. Each $ID_{\mathsf{U}} \in F(ID_{\mathsf{E}})$ corresponds to a real user U who has explicitly authorized S to learn about the edge U-E in the social graph.

Turning ersatz nodes into "standard" nodes. If a user E for whom S has created an ersatz node later joins `Social PaL`, he

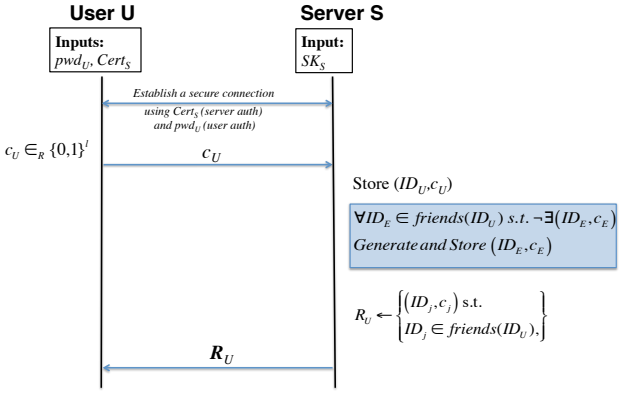


Figure 5: Adding ersatz nodes to the capability distribution.

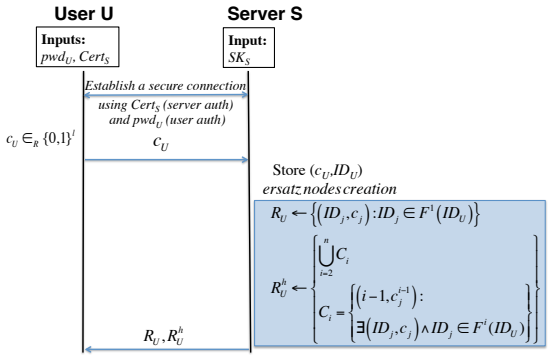


Figure 6: `Social PaL`'s capability distribution protocol.

can simply upload his capability c_E to S, who (1) overwrites the old ersatz capability with c_E, (2) queries the OSN for E's friend list $F(ID_{\mathsf{E}})$, and (3) updates the existing, possibly incomplete, list of E's friends with $F(ID_{\mathsf{E}})$, turning an ersatz node into a standard node. Note that this operation is transparent to all users.

3.3 Discovering Longer Social Paths

We now present the full details of our `Social PaL` instantiation: besides addressing the bootstrapping problem (using ersatz nodes), it also allows two arbitrary users to calculate the social path length between them. We denote with $Dist(\mathsf{A}, \mathsf{B})$ the social path length between two users A and B, i.e., the minimum number of hops in the social network graph that separates A and B.

Intuition. We set to allow `Social PaL` to discover the social path length between users in the OSN by extending the capability distribution to include further relationships beyond friendship (e.g., friend-of-a-friend) and rely on capability matches for estimating the social path length. By using cryptographic hash functions, we can generate and distribute capabilities of higher order that serve as a proof of a social path between users.

Notation. In the rest of the paper, we use the following notation:

- The hash chain $h^i(x)$ of item x (of length i) corresponds to the evaluation of a cryptographic hash function $h(\cdot)$ performed i times on x. When $i = 0$, $h(x) = x$. Specifically:

$$h^i(x) = \begin{cases} \underbrace{h(h(\cdots(h}_{i\ times}(x))\cdots)) & i \geq 1 \\ x & i = 0 \end{cases}$$

- c_j^k is a k-degree capability and is defined as $c_j^k = h^k(c_j)$
- $F^k(ID_{\mathsf{U}}), k \geq 1$ denotes the set of social contacts that are k-hops from user U.

Capability Distribution: In Figure 6, we detail `Social PaL`'s protocol for capability distribution. Interaction between U and S is identical to the capability distribution protocol from Figure 5, up until the creation of missing ersatz nodes is completed. In the updated protocol S returns two sets, namely R_U and R_U^h, where R_U^h denotes the set of higher order capabilities provided to U by other OSN members that are at least 2-hops from U. It is composed of a number of subsets C_i, $i = 2, \ldots, n$ with each subset C_i containing $i-1$ order capabilities of users in $F^i(ID_{\mathsf{U}})$. Formally,

$$R_U^h = \textstyle\bigcup_{i=2}^{n} C_i, \text{and}$$
$$C_i = \{(i-1, c_j^{i-1}) : \exists(ID_j, c_j) \wedge ID_j \in F^i(ID_{\mathsf{U}})\}$$

Consequently, the total cardinality of R_U^h and R_U is:

$$|R_U| + |R_U^h| = \sum_{i=1}^{n} |F^i(ID_{\mathsf{U}})|$$

Finally, U generates missing higher order capabilities. For every received capability c_j^i of degree i, U hashes it $n-i$ times to generate a sequence of higher order capabilities of the form:

$$((i+1, c_j^{i+1}), \ldots, (n, c_j^n))$$

All elements of such sequences are combined into one set of derived higher order capabilities R_U^d. Finally all capability sets are combined to form I:

$$I = R_U \cup R_U^h \cup R_U^d$$

The resulting set I will be used to derive the input sets for PSI during the social path length discovery protocol as explained below. The cardinality of the input set to PSI is therefore:

$$|I| = \sum_{i=1}^{n} |F^i(ID_{\mathsf{U}})| \times (n - i + 1)$$

To construct $F^n(ID_{\mathsf{U}})$ of user U, S tracks changes in friend lists of users by using the following logical implication: if j represents a friend of i and i is $k-1$ hops from U and j was not previously identified at less than k hops from U, then j is k hops from U. Formally:

$$(ID_i \in F^{k-1}(ID_{\mathsf{U}}) \wedge ID_j \in F(ID_i) \wedge$$
$$ID_j \notin F^m(ID_{\mathsf{U}}), m < k) \implies ID_j \in F^k(ID_{\mathsf{U}})$$

Finally, as capabilities are meant to be short-lived (i.e., they should expire within a couple of days), the protocol needs to be run periodically in order to keep I up-to-date.

Social Path Discovery: In Figure 7, we illustrate the `Social PaL` discovery protocol. The protocol involves two users A and B, who are members of the same OSN. It begins with establishing a secure channel (via Diffie-Hellman key exchange), followed by cryptographic binding of the Diffie-Hellman channel to the protocol instance, which is needed to avoid man-in-the-middle attacks. A (resp., B) appends both public keys to each capability c_j in I_A (resp., I_B) set to form $\overline{I_A}$ ($\overline{I_B}$). The resulting sets are:

$$\overline{I_A} = \{(c_j || PK_A || PK_B) : (*, c_j) \in I_A\}$$
$$\overline{I_B} = \{(c_j || PK_A || PK_B) : (*, c_j) \in I_B\}$$

Note that the $*$ symbol in the above equations indicate that, while constructing $\overline{I_A}$ and $\overline{I_B}$, the first element of each pair contained in I_A and I_B is ignored.

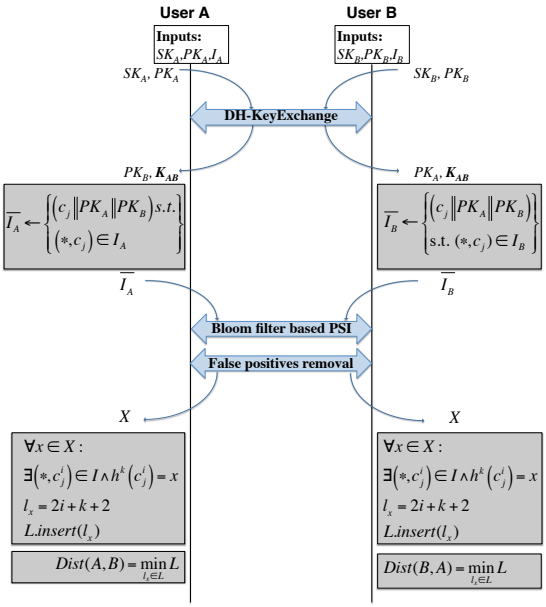


Figure 7: Illustration of `Social PaL` discovery protocol. The updated part is marked in grey background.

Next, both users execute the steps of `Common Friends`'s discovery protocol, on the above input sets. Specifically, they interact in a Bloom-filter based PSI execution and run the challenge-response part of the protocol needed to remove potential false positives (as discussed in Section 2.2). The interactive protocol ends with parties outputting the intersection of the sets. From this point on, both users perform identical actions to calculate the social path length between them. All operations are done locally, i.e., with no need to exchange data. This process consists of two phases: (1) calculating the social path length input set L (i.e., the set containing lengths for all discovered paths between A and B), and (2) selecting the shortest length among all lengths contained in L. To this end, A (B) builds set L by performing following actions on every item $x \in X$:

1. Finding a capability c_j^i such that $\exists(*, c_j^i) \in I \wedge h^k(c_j^i) = x$
2. Calculating path length l_x via matching capability x (which was obtained from some user, say C) and inserting it into L:

$$l_x = \underbrace{(i+1)}_{Dist(A,C)} + \underbrace{(i+k+1)}_{Dist(C,B)} = 2i + k + 2$$
$$L.insert(l_x)$$

At the end, A and B learn the final path length $Dist$ between them by finding the lowest value of items included in L:

$$Dist(\mathsf{A}, \mathsf{B}) = Dist(\mathsf{B}, \mathsf{A}) = \min_{l_x \in L} L$$

If $Dist(\mathsf{A}, \mathsf{B}) \leq 2$, then A and B have common friends between them, thus `Social PaL` returns identifiers of all these common friends as in the original `Common Friends` service. While we could use `Social PaL` to reveal the first hop identifiers for $Dist > 2$, we do not due to the privacy requirements outlined in Section 4.2.

4. ANALYSIS

This section presents the analysis of `Social PaL`, showing that it fulfills functional and privacy requirements from Section 3.1.

4.1 Correctness

LEMMA 1. *If $ID_{\mathsf{P}} \in F^k(ID_{\mathsf{A}})$, then:*

1. *There exists a path $X = \{x_i\}$, for $i \in \{0, \ldots, k-1\}$, between A and P, in the social graph.*
2. *A receives the i^{th} order capability $c_{x_i}^i$ from every x_i in X.*

PROOF. When $ID_{\mathsf{P}} \in F^k(ID_{\mathsf{A}})$, by using the logical implication for the social graph building (see Section 3.3), it must hold that, in order to include $ID_{x_{i+1}}$ in $F^{i+1}(ID_{\mathsf{A}})$, ID_{x_i} must be included in $F^i(ID_{\mathsf{A}})$. Therefore, we can recursively argue:

$$\exists x_{k-1} : ID_{x_{k-1}} \in F^{k-1}(ID_{\mathsf{A}}) \wedge ID_{x_{k-1}} \in F(ID_{\mathsf{P}})$$
$$\ldots$$
$$\exists x_{i+1} : ID_{x_{i+1}} \in F^{i+1}(ID_{\mathsf{A}}) \wedge ID_{x_{i+1}} \in F^{k-i}(ID_{\mathsf{P}})$$
$$\exists x_i : ID_{x_i} \in F^i(ID_{\mathsf{A}}) \wedge ID_{x_i} \in F^{k-1-i}(ID_{\mathsf{P}})$$
$$\ldots$$
$$\exists x_0 : ID_{x_0} \in F(ID_{\mathsf{A}}) \wedge ID_{x_0} \in F^{k-1}(ID_{\mathsf{P}})$$

Note that, for every $i \in \{0, \ldots, k-1\}$, there exists a connection to x_{i-1} and x_{i+1}, thus, $\{x_0, x_1, \ldots, x_{k-1}\}$ form a path X between A and P. Considering $x_i \in X, 0 \leq i < k-1$, since $x_i \in F^i(ID_{\mathsf{A}})$, then A receives $c_{x_i}^i$. □

THEOREM 1. *Let there be a path $X = \{x_i\}, i \in \{0, ..., d\}, d \geq 0$ between A and B in the social graph. If path X is discovered by the `Social PaL` discovery protocol, then both A and B can estimate the exact length $d + 2$ of path X.*

PROOF. Let n denote the highest degree of capabilities.

If $d < n$:

- The set of capabilities of A and B are $\{c_{x_0}, c_{x_1}^1, \ldots, c_{x_i}^i, \ldots, c_{x_d}^d\}$, and $\{c_{x_d}, c_{x_{d-1}}^1, \ldots, c_{x_i}^{d-i}, \ldots, c_{x_0}^d\}$, respectively. (See Figure 8(a) for a graphic illustration of the distribution of capabilities for A and B.)
- If A gets a matching capability for $c_{x_i}^i$, then it must corresponds to $c_{x_i}^{d-i}$ for B.
- A substitutes $k = d - i$ in $Dist(\mathsf{A}, \mathsf{B})$ and receives:
$$l_x = 2i + (d - i) + 2 = i + d + 2$$
- A gets multiple capability matches, and sets $Dist(\mathsf{A}, \mathsf{B})$ to be the minimum l_x, which is for $i = 0$, and $Dist(\mathsf{A}, \mathsf{B}) = d + 2$. (Similar argument holds for B.)

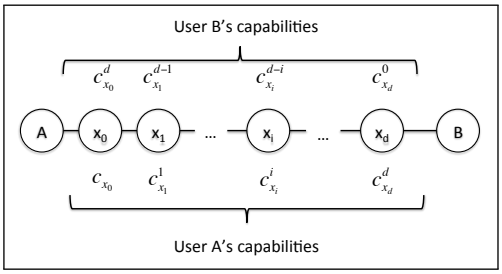
If $d \geq n$:

- The set of capabilities of A and B are $\{c_{x_0}, c_{x_1}^1, \ldots, c_{x_i}^i, \ldots, c_{x_n}^n\}$ and $\{c_{x_d}, c_{x_{d-1}}^1, \ldots, c_{x_n}^{d-i}, \ldots, c_{x_i}^{d-n}\}$, respectively – see Figure 8(b). (Capabilities for which A and B obtains matches are marked in green.)
- If A gets a capability match for: $\{c_{x_i}^i, \ldots, c_{x_n}^n\}$, A substitutes $k = n - i$ in $Dist(\mathsf{A}, \mathsf{B})$ and receives:
$$l_x = 2i + (n - i) + 2 = i + n + 2$$
- A gets multiple capability matches, and sets $Dist(\mathsf{A}, \mathsf{B})$ to be the minimum l_x, which is for $i = d - n$, and $Dist(\mathsf{A}, \mathsf{B}) = d + 2$. (Similar argument holds for B.) □

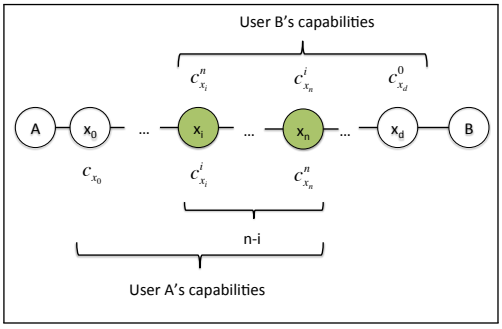
4.2 Privacy

As discussed in Section 3.1, our `Social PaL` instantiation needs to provide users with strong privacy guarantees, i.e., interaction between two users A and B does not reveal any information about their non-mutual friends or any other information than they could discover by gathering information from the standard OSN interface.

Capability Intersection. First, we review the security of the common friends discovery protocol from `Common Friends` [30], since



(a) Theorem 1 capability distribution for $d < n$ case.



(b) Theorem 1 capability distribution for $d \geq n$ case.

Figure 8: Illustration of theorem 1 capability distribution.

it constitutes the basis of our work. Its security, in the honest-but-curious model, reduces to the privacy-preserving computation of set intersection. That is, privacy stems from the security of the underlying Private Set Intersection (PSI) protocol that `Common Friends` instantiates to privately intersect capabilities and discover common friends. This is proven by means of indistinguishability between a real-world execution and an ideal-world execution where a trusted third party receives the inputs of both parties and outputs the set intersection. [30] uses Bloom Filter based PSI (BF-PSI) and, as discussed in Section 2.1, this does not impact security since sets to be intersected are random capabilities, thus, high-entropy, non-enumerable items.

Discovery. Now observe that the interactive part of the `Social PaL`'s (social path) discovery protocol – i.e., the part where information leakage might occur – mirrors that of `Common Friends`'s discovery protocol. During the protocol execution, users A and B engage in a BFPSI interaction, on input, respectively, $\overline{I_A}$ and $\overline{I_B}$, i.e., the sets of their capabilities, and obtain X, which is used to reconstruct the social path between A and B.

If A and B are friends with each other (or have mutual friends), they can find out the identity of the user(s) corresponding to matching capabilities, thus, learning that there exists a social path of length 1 (or 2) *and* the identity of their mutual friends, but nothing else. In fact, if an adversary could learn the identity of non-mutual friends, then, we could build an adversary breaking the common friends discovery protocol from [30] based on BFPSI. Similarly, if there exists no social path between A and B, then the BFPSI interaction does not reveal any information to each other.

On the other hand, if there exists a social path between A and B of length $Dist(A, B) > 2$, then the matching capabilities are for user nodes for which S has removed identifiers ID_j. Therefore, A and B do not learn the identity of the users yielding a social path between them, but only how many.

Trust in Server S. Each user U explicitly authorizes `Social PaL` to retrieve the set $F(ID_{\mathsf{U}})$ of U's friends. Requesting users to disclose their friend lists is a common practice in social network and smartphone applications. `Social PaL` uses this information to have the server S maintain, distribute, and, in the case of ersatz nodes, create capabilities attesting to the authenticity of friendships. This implies that S gradually learns the social graph from `Social PaL` users, however, what S learns is a small subset of what the OSN already knows. Neither S nor the OSN learn any additional information, e.g., as opposed to centralized solutions, user locations or interactions between users.

Authenticity of capabilities. In Section 3.1, we assumed the use of legitimate `Social PaL` client applications: as all mobile platforms provide application-private storage, it is reasonable to assume that an adversary on a client device cannot steal the capabilities downloaded on that device by the legitimate client application or otherwise manipulate the input to the protocol. Alternatively, the integrity of the Bloom Filter could be ensured by letting S sign the Bloom Filter along with the public key of the corresponding `Social PaL` client. The BFPSI protocol would then need to be modified accordingly so that each party checks the signature on the other party's Bloom Filter is valid and that the same keypair is used to establish the secure channel.

5. COVERAGE EVALUATION

We now present an empirical evaluation of `Social PaL`'s coverage, using three publicly available Facebook sample datasets. Specifically, we analyze how *coverage* attained by the social path discovery depends on the fraction of OSN users who join the system, i.e., the probability that two users discover an existing path between them in the social network.

5.1 Datasets Description

We use three datasets derived from a single dataset, created by Gjoka et al. [12, 13], using three different sampling techniques:

(1) The **Social Filter dataset** [33] is our primary dataset. It contains $500,000$ users, a connected component derived using the "forest fire" sampling method [20] from the original dataset [33]. As forest fire sampling does not preserve node degree, each node in this dataset has an average node degree of 30, which is significantly less than in the original dataset. To investigate the effect of the reduced node degree on coverage, we also use the two more datasets.

(2) The **MHRW dataset** [12] is built using the *Metropolis-Hastings Random Walk* (MHRW) method with 28 independent random walks. It contains the friend lists of $957,359$ users. We call this the set of *sampled users*. Each of them has an average of 175 friends, including both other sampled users and those who were part of the original dataset but that were not sampled – we call them *outside users*. The MHRW dataset contains a total of 72.2 million *outside users* (who are friends of one or more *sampled users*). Because of the nature of the MHRW sampling, the average number of connections between two *sampled users* in this set is only 3, thus it is used to evaluate `Social PaL`'s coverage among poorly connected users.

(3) The **BFS dataset** [12] is built using *Breadth First Search* (BFS) from 28 independent BFS traversals. It consists of 2.2 million *sampled users*, with an average of 310 friends. The number of *outside users* is 93.8 million. BFS sampling results in highly connected subgraphs, and the average number of connections among *sampled users* is 53. Thus, we use the BFS dataset to measure `Social PaL`'s coverage among well connected users.

5.2 Simulation

Procedure. To evaluate `Social PaL`'s coverage on each of the three datasets, we used the following simulation procedure. First,

Fraction of OSN with Social PaL	Path length	with ersatz [%]		w/o ersatz [%]	
		avg	std	avg	std
20%	2	100	0.0	25.12	0.27
	3	47.59	0.11	8.17	0.09
	4	44.17	0.21	3.85	0.09
40%	2	100	0.0	46.05	0.30
	3	73.05	0.09	25.27	0.11
	4	68.93	0.09	18.64	0.14
60%	2	100	0.0	65.02	0.17
	3	88.69	0.07	46.87	0.14
	4	85.86	0.11	40.94	0.19
80%	2	100	0.0	83.02	0.10
	3	97.32	0.04	72.10	0.18
	4	96.37	0.05	68.71	0.21

Table 2: Coverage results for the Social Filter dataset.

Fraction of OSN with Social PaL	Path length	with ersatz [%]		w/o ersatz [%]	
		avg	std	avg	std
20%	2	100	0.0	2.52	0.1
	3	22.77	0.23	0.15	0.02
	4	27.86	0.26	0.003	0.002
40%	2	100	0.0	5.26	0.15
	3	43.71	0.22	0.55	0.02
	4	48.46	0.25	0.03	0.004
60%	2	100	0.0	7.85	0.14
	3	63.50	0.13	1.19	0.03
	4	67.09	0.24	0.09	0.005
80%	2	100	0.0	10.34	0.14
	3	82.18	0.19	2.04	0.03
	4	83.90	0.29	0.19	0.01

Table 3: Coverage results for the MHRW dataset.

we chose, at random, a subset of *sampled users*, which we call the *test set*. For the Social Filter dataset, we used the whole set as the *sampled users* set. We chose four different sizes for the *test set*: $\{20, 40, 60, 80\}\%$ of the *sampled users*. Note that the *test set* represents the fraction of the users of an OSN who use Social PaL.

Then, for a given social path length n ($n \in \{2, 3, 4\}$), we randomly selected $50,000$ pairs of users from the *test set* in such a way that at least one path of length n exists. Finally, we computed the fraction of user pairs for which Social PaL discovers an existing path between them. We did this for two cases: Social PaL with support for ersatz nodes, and without it. Each simulation was repeated 10 times. In total, we conducted 720 different simulations.

Results. We now present the results of our simulations for each of the three datasets. Social PaL's discovery coverage is presented in Table 2 for the the Social Filter dataset, Table 3 for the MHRW dataset, and Table 4 for the BFS dataset. Additional graphs on coverage results are available from the full version of the paper [29].

Without ersatz nodes, coverage increases linearly as more users start using Social PaL. The rate of growth is highest for the Social Filter dataset and lowest for the MHRW dataset. In general, the coverage figures are low. For instance, even if 80% of OSN users have Social PaL, the coverage for paths of length 4 ranges between 0.19% (the MHRW dataset) and 68.71% (the Social Filter dataset). The introduction of ersatz nodes results in a remarkable improvement across the board in all datasets. As expected, the coverage for paths of length 2 is 100%. When 80% of OSN users are in the Social PaL system, the coverage is well above 80% in all cases. Even when only 20% of users have Social PaL, coverage is still above 40% in all cases, except for the MHRW dataset.

5.3 Take-aways

Ersatz nodes dramatically improve coverage. With ersatz nodes, Social PaL discovers 100% of social paths of length 2, thus addressing one of the major limitations of the Common Friends system [30]. The coverage for paths of length 3 and 4 always increases, between 10% and 80%, depending on the fraction of OSN users in Social PaL and the dataset used for the simulations.

Variation of coverage across different datasets. We observe better coverage results with the BFS dataset than with the MHRW dataset. As the BFS dataset represents coverage among well connected users, the density of ersatz nodes between random users is higher than in the MHRW dataset, thus yielding better overall coverage. The BFS dataset models societies, such as most of the western societies, where the penetration of OSNs is high. The high coverage results with the BFS dataset suggests that Social PaL will do well in this context. On the other hand, the MHRW dataset models societies where OSN connectivity is poor and, although Social PaL is not as effective here, it may still perform reasonably well, detecting the majority of social paths even before the number of users joining Social PaL reaches 50%.

Fraction of OSN with Social PaL	Path length	with ersatz [%]		w/o ersatz [%]	
		avg	std	avg	std
20%	2	100	0.0	13.69	0.34
	3	42.42	0.30	4.31	0.11
	4	54.11	0.21	1.51	0.04
40%	2	100	0.0	23.85	0.26
	3	63.46	0.23	10.98	0.17
	4	72.39	0.32	6.71	0.11
60%	2	100	0.0	33.19	0.26
	3	78.62	0.17	18.50	0.09
	4	84.28	0.17	14.30	0.13
80%	2	100	0.0	41.46	0.28
	3	90.39	0.12	26.24	0.19
	4	93.03	0.10	22.71	0.11

Table 4: Coverage results for the BFS dataset.

6. IMPLEMENTING SOCIAL PAL

In this section, we present our full-blown implementation of the Social PaL system. We aim to support scalability for increasing number of users (in terms of CPU performance and memory) and to enable developers to easily integrate it into their applications.

6.1 Server Architecture

Server components. On the server side, the Social PaL system extends the PeerShare server [28], which allows two or more users to share sensitive data among social contacts, e.g., friends in a social network. We use the following basic functions of PeerShare: (1) OSN interfaces to retrieve social graph information, (2) the OAuth [14] component for user authentication, and (3) the data distribution mechanism. On top of these components, we develop a new server architecture that supports the addition of server-based applications via an extension mechanism. This design choice allows us to implement the Social PaL functionality in such a way that the system can efficiently scale (in terms of memory and CPU performance) to support an increasingly large number of users.

As illustrated in Figure 9, the server architecture consists of the following components: the *Common Apps Server* , a group of applications (e.g., the Social PaL App), the *OSN Communicator Module*, and the *Bindings Database* (Bindings DB). The Common Apps Server provides the basic functionality that is common for all applications: (1) storage of data uploaded by users in the Bindings DB, (2) distribution of users' data to other authorized users, and (3) retrieval of basic social graph information, which is needed for enforcing the appropriate data distribution policy. The OSN Communicator Module is a plugin-based service responsible for querying OSNs for social graph information. Its plugin-based structure allows us to easily add support for new OSNs. The Bindings DB stores data uploaded by users and information on how to distribute them among social contacts. More details about the server components are available in [29].

Social PaL server implementation. The Social PaL server application is notified via a App Event callback interface about new capabilities uploaded by users. It uses a App-DB Updates interface to create any required ersatz nodes and properly update the recipient sets of capabilities during the social graph building process.

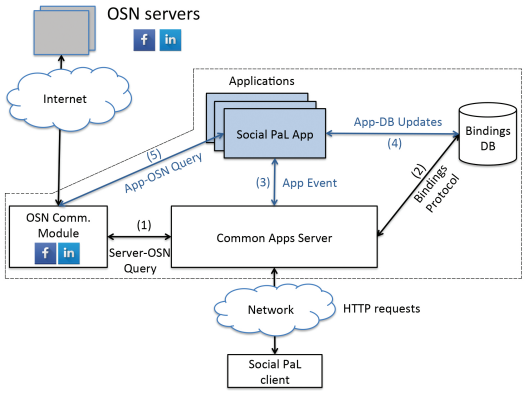


Figure 9: Single instance of `Social PaL` server architecture. Common Apps Server component are marked in black, while application specific elements are marked in blue.

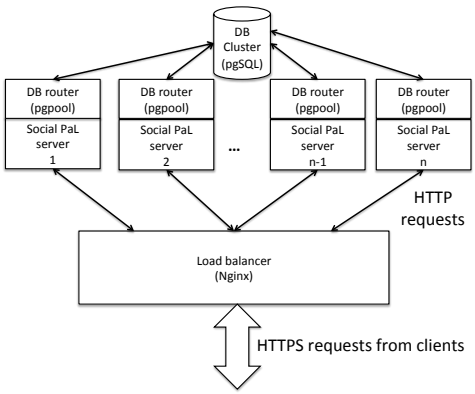


Figure 10: Scalable `Social PaL` server architecture.

The `Social PaL` application also uses the App Event interface to handle capability download requests. The Common Apps Server, instead of immediately returning data it has read from the Bindings DB, passes them to the `Social PaL` application that generates any missing higher order capabilities. Finally, the `Social PaL` application returns the complete set of capabilities back to the Common Apps Server, which completes the request handling. Note that our implementation of the OSN Communicator Module supports interactions with both LinkedIn and Facebook.

Implementation details. Our core `Social PaL` server is written in PHP. We support capabilities of 0^{th} and 1^{st} order, allowing to discover social paths between users that are up to 4 hops from one another. Based on relevant prior work [3, 26, 37], a 4-hop distance is enough for most practical use cases. As the Bindings DB needs to store the capabilities of users and information about how to share those, the necessary amount of persistent storage will substantially grow if `Social PaL` becomes widely used. Thus, to limit the data storage overhead, `Social PaL` server does not store any higher order capabilities in the Bindings DB, but generates them when requested by the requesting client. Tests on our server show that generating higher order capabilities has a negligible impact on the `Social PaL` *capability distribution protocol* performance (i.e., the server generates 1 million higher order capabilities in about 500ms using the hardware described in Section 6.2). Finally, in order to implement the LinkedIn OAuth module for the OSN Communicator Module, we use the OAuth Pecl extension for PHP, while, for the Bindings DB, PostgreSQL database server.

System scaling. Since `Social PaL` may generate a large number of server requests if used by a large number of users, we can take following steps to ensure that the system can scale. Our proposed scaling architecture is illustrated in Figure 10. It includes a powerful HTTP front-end server (such as Nginx) acting as load balancer, which terminates incoming secure HTTPS connections and forwards server requests upstream to n instances of `Social PaL` servers acting as request handlers. Each `Social PaL` server instance will run the *HipHop Virtual Machine* (HHVM) daemon that handles HTTP requests. HHVM usage can massively improve server performance, as it uses just-in-time compilation to take advantage of the native code execution instead of the interpreted one [34].

Each instance of `Social PaL` server runs, locally, a database query router (pgpool) providing access to the actual database cluster including multiple PostgreSQL servers. The query router enhances the overall database access performance by keeping open connections to the database cluster, load-balancing the stored data among multiple instances of the database servers, and temporarily queuing requests for database access in case of cluster overload. Note that there are no cross-dependencies between the `Social PaL` server instances for the database read access, thus, no complex control mechanism is needed to support this parallelism.

Server code. The source code of the server implementation is available from `https://github.com/SecureS-Aalto/SoPaL`.

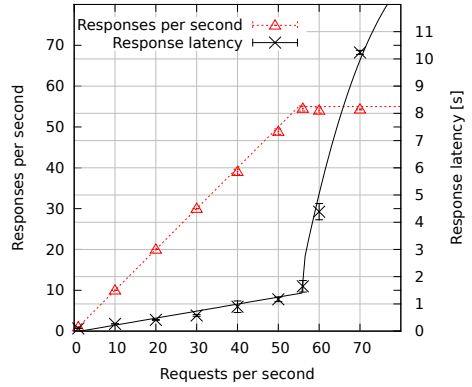
6.2 Server Performance Evaluation

Performance testbed. We evaluated our server implementation in a testbed consisting of two machines: the first played the role of a single `Social PaL` server instance (cf. Figure 9), while the second simulated a group of client devices. The server ran on a 4-core machine with a 2.93GHz CPU on each core and 128GB of RAM. It hosted Nginx (version 1.1.19), PostgreSQL server (version 9.1), and php5-fpm for the PHP 5.6 engine. Inter-process communication was implemented via UNIX sockets. The machine running the clients had 8 CPU cores (at 2.93 GHz) and 64GB of RAM.

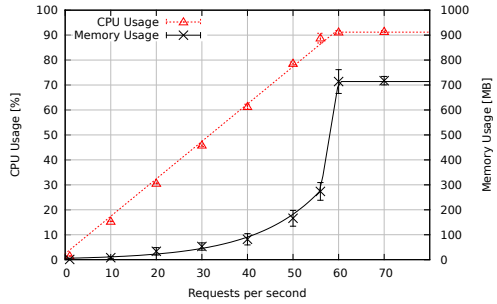
To eliminate the unpredictability of network latency, we modified the server implementation by replacing the OSN Communicator Module with the local service that provided social graph information based on the MHRW dataset. We populated the Bindings DB with capabilities generated for the 120,000 *sampled users* from the MHRW dataset. The capabilities of *sampled users* together with capabilities generated for ersatz nodes constituted about 10 million data items that were stored in the Bindings DB. Finally, to minimize impact of the client-server transmission delay, we kept the server and client machines in the same network and connected them using a 1 Gbit/s Ethernet link via the single switch.

Experiments. We evaluated server performance by sending bursts of n requests per second, for $n \in \{1, 10, 20, 30, 40, 50, 60, 70\}$, from the client machine to fetch capabilities from the server (i.e., download of R_U and R_U^d in Figure 6) for 60 seconds. Fetching capabilities involves many read operations on the Bindings DB, thus yielding the highest load on the server among all operations of the `Social PaL` capability distribution protocol. In each experiment, which we repeated ten times, we measured the number of received responses per second together with the latency of each response on the client machine, and CPU usage together with memory consumption on the server machine.

Results. Figure 11 illustrates the results of our experiments. We observe that 56 requests per second yields a saturation point for the server. Below 56 requests/second, the number of responses per second and the response latency grow linearly. Whereas, as depicted in Figure 11(a), above 56, we observe an exponential growth of the response latency and the constant number of received responses per second. Figure 11(b) also shows that CPU usage reaches more than 90% above 56 requests/second. Peak memory consumption is



(a) Responses per second and response latency vs # requests per second.



(b) CPU usage and memory consumption vs # requests per second.

Figure 11: `Social PaL` server performance.

about 700MB, which also shoots up significantly when the number of requests crosses the saturation point.

The 56 requests/second saturation point shows that the performance of our server implementation is in line with that emerging from studies of systems equipped with similar hardware [34, 35]. We also looked at the server performance when only handling 1 request per second and observed that the average response latency is about 50ms, and the average Bindings DB interaction time is around 5ms. Since client-server network latency is negligible, the vast majority of request handling takes place in the PHP interpreter, which highlights that PHP is the server's bottleneck. Therefore, in order to improve the server performance, php5-fpm should be replaced with HHVM, which is reported to be significantly more performant [34, 35]. Further gains could also be obtained by migrating the PostgreSQL server to a separate machine connected over a fast link (cf. Fig. 10). We leave these as part of future work.

Assuming that the server handles 56 requests per second, a total of 4.84 million requests can be processed daily by a single instance of `Social PaL` server with comparable hardware capabilities. Assuming that each user executes the `Social PaL` capability distribution protocol around 4 times a day, about 1.21 million `Social PaL` users can be handled by one `Social PaL` server instance. Since user requests are independent of each other, and because the scalable architecture of the `Social PaL` server allows adding further instances easily (as described in Section 6.1), the total capacity of `Social PaL` system amounts to the cumulative number of users that can be handled across all `Social PaL` server instances. Finally in order to avoid making PostgreSQL become the bottleneck of the system (which may be caused if many `Social PaL` server instances are added), the Bindings DB should be turned into a database cluster with data sharding and replication enabled. This guarantees that the data kept in the Bindings DB is synchronized and accessible with high enough availability.

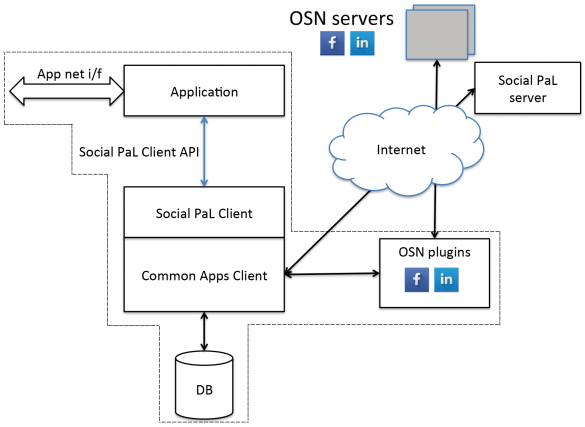


Figure 12: `Social PaL` client architecture.

6.3 Client Implementation

Client architecture. The client-side architecture of `Social PaL` is depicted in Figure 12. It consists of the *Common Apps Client*, the set of OSN plugins, and the `Social PaL` *Client*. The first two components are responsible for the communication with the `Social PaL` server, while the last one provides the interface for the applications. Together, these components form a mobile platform library that can be easily imported by developers into their applications. To facilitate support for multiple OSNs, similar to our server design, we have decoupled OSN-specific functionality from the Common Apps Client and made it a plugin-based solution.

We have considered two possible design choices for the client architecture: (1) designing it as a stand-alone service with applications connecting to it, using available inter-process communication mechanisms, or (2) as a service integrated into the application. We choose the latter as it supports application-private storage for capabilities (i.e., not accessible by other applications) and enables each application to have its own `Social PaL` server. This choice provides additional protection against capability leakage to a malicious application and removes the requirement to deploy the global `Social PaL` server. On the other hand, if multiple instances of `Social PaL` application runs on the same device, we would incur increased network traffic and require more storage space in comparison to the stand-alone service approach. We argue that this tradeoff is acceptable, as the `Social PaL` capability distribution is run no more than a few times a day. Also, the amount of data to store is likely to be limited in the order of tens of megabytes, which is justifiable given the clear usability and deployability advantages.

Implementation details and performance. We have implemented the client library on Android, operating as an Android lightweight service. The core operation of the client functionality only involves the `Social PaL` discovery protocol: since this does not add any more expensive cryptographic operations than those in `Common Friends`, we refer to the performance evaluation presented in the full version of the paper [29], which attests to the limited computation and communication overheads.

`Social PaL` **Client API.** The `Social PaL` application interface is used by applications to run the `Social PaL` discovery protocol. It has been designed to be readily usable by application developers that are not cryptography experts, but are nonetheless interested in implementing privacy-preserving discovery of social paths. This allows developers to delegate the responsibility of this process to the `Social PaL` Client, and requires them to integrate only four basic methods into the application code, which we present below.

Applications running the Social PaL discovery protocol act as Social PaL message forwarders between the two Social PaL Client instances. The application starting the Social PaL discovery session calls the startSoPaLSession method. This returns an opaque Social PaL object, which is forwarded to the remote party. From this point onward, both parties invoke handleSoPaLMessage for every message received. This method processes the received message, and if needed, creates a response. It also returns a flag indicating if the protocol execution is completed. If so, the application uses the getResult method to get the social path length it has to a remote party. Finally, the application must call endSoPaLSession to let the Social PaL Client release all resources from the session.

Besides these four basic methods, the Social PaL Client also provides three advanced methods: (1) rejectSoPaLSession creates a Social PaL message that can be sent by the application to the remote party if it does not want to run the discovery protocol; (2) updateCapabilities and (3) renewCapability can be used by the application to force fetching the most recent capabilities from the server, and to generate and upload a new capability to the server, respectively. (More details about the Client methods are available from the full version of the paper [29]).

7. SAMPLE APPLICATIONS

To illustrate Social PaL's relevance and practicality, we integrate it into two Android apps, SpotShare and nearbyPeople, supporting both Facebook and LinkedIn.

SpotShare[5] is an extension of TetheringApp, presented in [30]. It allows a user to provide tethered Internet access to other SpotShare users (where access to the tethering hotspot is protected by a password) so that access control policies can be based on social relationships. For instance, the user can decide to allow tethered access only to friends of friends: to this end, SpotShare uses Social PaL in order to determine, in a privacy-preserving way, if the specified social relationship holds. If so, the password is securely and automatically sent to the requesting device. In the current version of SpotShare, we do not enable discovery of social paths beyond two hops, as we assume that most users would not want to allow people with whom they have no common friends to tether off their smartphone, but removing this constraint is trivial.

nearbyPeople[6] is a "friend radar" app allowing users to interact with people around them and discovering common friends shared with users of nearby devices, as well as social path lengths, without having to broadcast their social profiles or rely on a central server. It relies on the privacy guarantees of Social PaL and the SCAMPI opportunistic router [18] for device-to-device communication.

8. RELATED WORK

Privately discovering social relationships. Nagy et al. [30] introduce Common Friends, reviewed in Section 2.2, combining bearer capabilities with BFPSI/PSI-CA to allow OSN users to discover, respectively, the identity or the number of their common friends in a private, authentic, and decentralized way. While we build on the concept of capabilities and rely on BFPSI, recall that Common Friends suffers from an inherent bootstrapping problem and is limited to the discovery of social paths to OSN users that are two hops away. Our work does not only address Common Friends's limitations via a novel methodology, but also presents the full-blown implementation of a scalable server architecture and a modular Android client library enabling developers to easily integrate Social PaL into their applications.

Mezzour et al. [24] also describe techniques for decentralized path discovery in social networks. They use a notion similar to capabilities to represent friendships and hashing to derive higher-order capabilities, however, their scheme distributes a *different* capability on behalf of a given user to every other user, while Social PaL distributes the same capability to all users at a given distance. [24]'s computational/communication overhead is significantly higher than that of Social PaL: the former requires two PSI runs, with sets of size equal to the total number of paths from a node up to the maximum supported path length, whereas, the latter only requires a single BFPSI run, with input sets as big as the number of paths that have length equal to half the maximum supported path length.[7] Furthermore, [24] incurs the same bootstrapping problem as [30]: if a friend A of user U does not participate in the system, U cannot detect paths to some other user B that go through A. Finally, [24] aims to build a decentralized social network, while we aim to bootstrap the system based on existing centralized social networks.

Liao et al. [23] present a privacy-preserving social matching protocol based on property-preserving encryption (PPE), which however relies on a centralized approach. Li et al. [21] then propose a set of protocols for privacy-preserving matching of attribute sets of different OSN users. Private friend discovery has also been investigated in [15] and [38], which do not provide authenticity as they are vulnerable to malicious users claiming non-existent friendships. While [8] addresses the authenticity problem, it unfortunately comes at the cost of relying on relatively expensive cryptographic techniques (specifically, a number of modular exponentiations linear in the size of friend lists and a quadratic number of modular multiplications).

Lentz et al. introduce SDDR [19], which allows a device to establish a *secure encounter* – i.e., a secret key – with every device in short radio range, and can be used to recognize previously encountered users, while providing strong unlinkability guarantees. The EnCore platform [1] builds on SDDR to provide privacy-preserving interaction between nearby devices, as well as event-based communication for mobile social applications.

Building on Social Relationships. Prior work has also focused on building services on top of existing social relationships. Cici et al. [5] use OSNs to assess the potential of ride-sharing services, showing that these would be very successful if users shared rides with friends of their friends. Sirivianos et al. [33] propose a collaborative spam mitigation system leveraging social networks of administrators, while [31] and [32] use OSNs to verify the veracity of online assertions. Freedman and Nicolosi [9] describe a system using social network for trust establishment in the context of email white-listing, by verifying the existence of common friends. Besides not discovering paths longer than two, [9] also does not address the issue of friendships' authenticity – unlike Social PaL.

Daly et al. [6] present a routing protocol (called SimBet) for DTN networks based on social network data. Their protocol attempts to identify a routing bridge node based on the concept of centrality and transitivity of social networks. Li et al. [22] design another DTN routing protocol (called Social Selfishness Aware Routing) which takes into account user's social selfishness and willingness to forward data only to nodes with sufficiently strong social ties. Other work [16, 39, 40] also propose adjusting message forwarding based on some social metrics.

[5] https://play.google.com/store/apps/details?id=org.sesy.tetheringapp

[6] https://se-sy.org/projects/pet/nearbypeople.html

[7] In the full version of the paper [29], we report expected input set sizes based on the datasets introduced in Section 5.

OSN Properties. Another line of work has studied properties of OSNs. Ugander et al. [37] and Backstrom et al. [3] study the structure of Facebook social graph, revealing that the average social path length suggested by the "small world experiment" [26] (i.e., six) does not apply for Facebook, as the majority of people are separated by a 4-hop path.

9. CONCLUSION

This paper presented `Social PaL` – a system geared to privately estimate the social path length between two social network users. We demonstrated its effectiveness both analytically and empirically, showing that, for any two OSN users, `Social PaL` discovers all social paths of length two and a significant portion of longer paths. Using different samples of the Facebook graph, we showed that even when only 20% of the OSN users use the system, we discover more than 40% of all paths between any two users, and 70% with 40% of users.

We also implemented a scalable server-side architecture and a modular client library bringing `Social PaL` to the real world. Our deployment supports Facebook and LinkedIn integration and allows developers to easily incorporate it in their projects. `Social PaL` can be used in a number of applications where, by relying on the (privacy-preserving) estimation of social path length, users can make informed trust and access control decisions.

In future work, we will augment `Social PaL` with information about the *tie strength* [2, 11] between users, and present a usability study of some sample applications built on top of the system.

Acknowledgments. We thank Minas Gjoka and Michael Sirivianos for sharing the Facebook datasets, Swapnil Udar for helping with the `SpotShare` implementation, and Jussi Kangasharju, Pasi Sarolahti, Cecilia Mascolo Panos Papadimitratos and Narges Yousefnezhad for providing feedback on the paper. Simon Eberz suggested the idea of signed Bloom Filters discussed in Section 4.2. This work was partially supported by the Academy of Finland's "Contextual Security" project (274951), the EC FP7 PRECIOUS project (611366), and the EIT ICT Labs.

References

[1] P. Aditya, V. Erdelyi, M. Lentz, E. Shi, B. Bhattacharjee, and P. Druschel. EnCore: Private, Context-based Communication for Mobile Social Apps. In *MobiSys*, 2014.

[2] V. Arnaboldi, A. Guazzini, and A. Passarella. Egocentric Online Social Networks: Analysis of Key Features and Prediction of Tie Strength in Facebook. *Elsevier Computer Communications*, 36(10), 2013.

[3] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna. Four Degrees of Separation. *CoRR*, abs/1111.4570, 2011.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7), 1970.

[5] B. Cici, A. Markopoulou, E. Frias-Martinez, and N. Laoutaris. Assessing the Potential of Ride-sharing Using Mobile and Social Data: A Tale of Four Cities. In *Ubicomp*, 2014.

[6] E. M. Daly and M. Haahr. Social Network Analysis for Routing in Disconnected Delay-tolerant MANETs. In *MobiHoc*, 2007.

[7] E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and Private Computation of Cardinality of Set Intersection and Union. In *CANS*, 2012.

[8] E. De Cristofaro, M. Manulis, and B. Poettering. Private Discovery of Common Social Contacts. In *ACNS*, 2011.

[9] M. J. Freedman and A. Nicolosi. Efficient Private Techniques for Verifying Social Proximity. In *IPTPS*, 2007.

[10] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT*, 2004.

[11] E. Gilbert. Predicting Tie Strength in a New Medium. In *CSCW*, 2012.

[12] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in Facebook: A Case Study of Unbiased Sampling of OSNs. In *INFOCOM*, 2010.

[13] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Practical Recommendations on Crawling Online Social Networks. *IEEE JSAC on Measurement of Internet Topologies*, 2011.

[14] D. Hardt. The OAuth 2.0 authorization framework. RFC 6749, RFC Editor, 2012.

[15] Y. Huang, E. Chapman, and D. Evans. Privacy-preserving applications on smartphones. In *HotSec*, 2011.

[16] P. Hui, J. Crowcroft, and E. Yoneki. Bubble Rap: Social-based Forwarding in Delay Tolerant Networks. In *MobiHoc*, 2008.

[17] A. Johnson, P. Syverson, R. Dingledine, and N. Mathewson. Trust-based anonymous communication: Adversary models and routing algorithms. In *CCS*, 2011.

[18] T. Kärkkäinen, M. Pitkänen, P. Houghton, and J. Ott. SCAMPI Application Platform. In *CHANTS*, 2012.

[19] M. Lentz, V. Erdélyi, P. Aditya, E. Shi, P. Druschel, and B. Bhattacharjee. SDDR: Light-Weight, Secure Mobile Encounters. In *USENIX Security Symposium*, 2014.

[20] J. Leskovec and C. Faloutsos. Sampling from Large Graphs. In *KDSS*, 2006.

[21] M. Li, N. Cao, S. Yu, and W. Lou. FindU: Privacy-preserving personal profile matching in mobile social networks. In *INFOCOM*, 2011.

[22] Q. Li, S. Zhu, and G. Cao. Routing in Socially Selfish Delay Tolerant Networks. In *INFOCOM*, 2010.

[23] X. Liao, S. Uluagac, and R. A. Beyah. S-MATCH: Verifiable Privacy-preserving Profile Matching for Mobile Social Services. In *DSN*, June 2014.

[24] G. Mezzour, A. Perrig, V. D. Gligor, and P. Papadimitratos. Privacy-Preserving Relationship Path Discovery in Social Networks. In *CANS*, 2009.

[25] M. Miettinen, S. Heuser, W. Kronz, A.-R. Sadeghi, and N. Asokan. ConXsense: Automated Context Classification for Context-aware Access Control. In *ASIACCS*, 2014.

[26] S. Milgram. The small world problem. *Psychology Today*, 67(1):61–67, 1967.

[27] P. Mittal, M. Wright, and N. Borisov. Pisces: Anonymous Communication Using Social Networks. In *NDSS*, 2013.

[28] M. Nagy, N. Asokan, and J. Ott. `PeerShare`: A System Secure Distribution of Sensitive Data Among Social Contacts. In *NordSec*, 2013.

[29] M. Nagy, T. Bui, E. De Cristofaro, N. Asokan, J. Ott, and A.-R. Sadeghi. How Far Removed Are You? Scalable Privacy-Preserving Estimation of Social Path Length with Social PaL (Full Version). `http://arxiv.org/abs/1412.2433`, 2015.

[30] M. Nagy, E. D. Cristofaro, A. Dmitrienko, N. Asokan, and A. Sadeghi. Do I know you? Efficient and Privacy-preserving Common Friend-Finder Protocols and Applications. In *ACSAC*, 2013.

[31] G. Norcie, E. De Cristofaro, and V. Bellotti. Bootstrapping Trust in Online Dating: Social Verification of Online Dating Profiles. In *USEC*, 2013.

[32] M. Sirivianos, K. Kim, J. W. Gan, and X. Yang. Assessing the veracity of identity assertions via OSNs. In *COMSNETS*, 2012.

[33] M. Sirivianos, K. Kim, and X. Yang. SocialFilter: Introducing Social Trust to Collaborative Spam Mitigation. In *CollSec*, 2010.

[34] C. Stocker. HHVM with Symfony 2 looks amazing. `http://blog.liip.ch/archive/2013/10/29/hhvm-and-symfony2.html`.

[35] A. Tagliapietra. Symfony benchmark using HHVM. `http://www.alexfu.it/2013/10/22/symfony-benchmark-on-hhvm.html`.

[36] A. S. Tanenbaum et al. Using Sparse Capabilities in a Distributed Operating System. In *ICDCS*, 1986.

[37] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The Anatomy of the Facebook Social Graph. *CoRR*, abs/1111.4503, 2011.

[38] M. Von Arb, M. Bader, M. Kuhn, and R. Wattenhofer. VENETA: Serverless friend-of-friend detection in mobile social networking. In *WiMob*, 2008.

[39] Y. Zhang, J. Zhao, and G. Cao. Roadcast: A Popularity Aware Content Sharing Scheme in VANETs. *SigMobile Mob. Comput. Commun. Rev.*, 13(4), Mar. 2010.

[40] J. Zhao and G. Cao. VADD: vehicle-assisted data delivery in vehicular ad hoc networks. In *INFOCOM*, 2006.