Lilliput: A Storage Service for Lightweight Peer-to-Peer Online Social Networks

Thomas Paul* Niklas Lochschmidt* Hani Salah† Anwitaman Datta‡ Thorsten Strufe§

*TU Darmstadt, Germany, thomas.paul@cs.tu-darmstadt.de, nlochschmidt@gmail.com
†Palestine Polytechnic University, Palestine, hani@ppu.edu
‡Nanyang Technological University, Singapore, anwitaman@ntu.edu.sg
§TU Dresden, Germany, thorsten.strufe@tu-dresden.de

***Abstract*—P2P-based social networking services are severely challenged by churn and the lack of reliable service providers, especially considering the high frequency of posts and profile updates of their users. Improved consistency and data availability shall facilitate better acceptance, which in turn will enhance privacy, an inherent benefit of this class of systems.**

We present Lilliput, a P2P storage primitive designed with the characteristics of Online Social Network workloads in mind. Lilliput separates the storage of static bulk data (videos and photo albums) from the *essential social glue* (e.g. basic profile information, frequent updates, notifications, and personal messages): it provides the latter through agile, lightweight replica groups. Extensive simulations show that Lilliput ensures high data availability (99.07% to 99.64%) and consistency, with a small bandwidth usage under realistic usage and load models.

***Index Terms*—Online Social Networks; P2P; Decentralized; Storage; Overlays**

I. Introduction

Online social networks (OSNs) have radically changed the way people interact and share personal data. However, centralized OSNs like Facebook raise many concerns regarding privacy, ownership, and censorship. The end-user in these systems is at the mercy of (i) the goodwill of service providers to not abuse their pivotal role and (ii) ability of providers to prevent abuse by malicious outsiders and insiders.

Several solutions have been proposed in the last years to improve privacy and censorship resistance in OSNs. Among them, realizing OSN features over a peer-to-peer (P2P) infrastructure has been widely proposed and explored in the last years, and it is vastly looked at as a promising approach (e.g. see [1]–[9] and the references therein).

Existing P2P-based OSNs (both academic and open-source community initiatives) emulate the feature set of popular social networking services like Facebook and Twitter. We argue that these systems are not mature enough to provide easy-to-use and reliable service. This is mainly because they rely on very strong assumptions in their design and evaluations, especially with respect to churn behaviour of peers. In particular, the churn rates measured in popular OSNs (e.g. in Facebook [10]) are too high to reliably replicate the *entire* profiles in a P2P manner. That is to say, too many peers are needed, under a realistic churn, to guarantee 24/7 availability of up-to-date profiles.

Instead, our goal in this paper is to realize a P2P-based OSN that is lightweight and at the same time provides high data availability and consistency. To that end, we propose to dedicate the P2P storage primarily to the *essential social glue* (e.g. basic profile information, recent updates, notifications, and messages). In contrast, the old and rarely accessed [10] bulk data (e.g. photo albums and videos) can be encrypted and stored by a third party (e.g. a cloud provider), without disclosing comprehensive communication patterns.

The rationale behind this design choice is threefold: First, the essential social glue contains the most critical data from privacy's point of view, like communication patterns and the social graph. It cannot be hidden from centralized storage providers by means of encryption, since the exchange of ciphertext unveils this information. Therefore, centralized storage of such data should be avoided. Second, the design mitigates the negative impact of storing and serving stale bulk data on quality-of-service. Third, decreasing the size of replicated data mitigates both the impact of free-riding peers as well as the incentives to free ride.

The proposed design is also supported by recent research on user behaviour in OSNs. For instance, Paul et al. [10] recently showed that 84.79% of displayed news-feeds in Facebook is not older than 24 hours. They also showed that 41.76% of shared content consists of external links which only need a few bytes to be stored and disseminated. These results imply that the total volume of the essential social glue is small, thus can be easily maintained by a P2P infrastructure.

The main contribution that we present in this paper is Lilliput, a set of protocols to realize a storage service for lightweight P2P-based OSNs. Lilliput lies in the middle of the solution space between the two extremes: Distributed Hash Tables (DHTs) and static group-based replication [2]. By this, we attempt to identify a sweet-spot in the trade-off between high profile availability and low resource consumption.

Lilliput consists of small data overlays (i.e. replica groups) each of which is created by the data owner. Other peers are invited to join the overlay till the number of participants reaches a preset (small) threshold. From that moment onwards, the data owner can disconnect, and the participating peers will manage the churn in the overlay on their own. Lilliput motivates cooperation among peers by providing reciprocity

in serving and replicating data. It also benefits from the high likelihood to serve the same content by incrementally updating data rather than transmitting entire copies of the data.

We show via extensive simulations that Lilliput, even with small replica groups and under high churn rates, can keep the essential social glue online and up-to-date. We show also that this is achieved using a small amount of network resources.

The remainder of this paper is structured as follows: We review the related work in Section II where we point out the gap this paper fills. Next, we describe Lilliput in details in Section III and evaluate it in Section IV. After that, we discuss several aspects in Section V. Finally, we conclude the paper in Section VI.

II. Related Work

P2P-based storage systems have been studied widely in the last years. General-purpose P2P storage approaches like Oceanstore [11] and UniStore [12] are designed to provide a persistent storage for predominantly static data. That is to say, they are not optimized to store frequently updated social data. In the remainder of this section, we focus only on prior studies that are highly related to ours, i.e. P2P-based OSNs and storage approaches designed specifically for OSNs.

Most existing works on P2P-based OSNs [1], [5]–[7], [13], [14] take a holistic view on the system. Availability, consistency, and dissemination of profiles received peripheral attention in these studies. Instead, they focus on the system design, communication protocols, and encryption schemes. Majority of these studies relies on either social friends or an underlying storage service (e.g. a DHT) to replicate profiles.

Notable, more related, studies include [15], SuperNova [3], My3 [16], GemStone [17], and S-DATA [18]. The focus of [15] is mainly on dissemination of profile data (using friends), while the focus of SuperNova is on bootstrapping and incentives for participation.

The main idea of My3 is to select a subset of trusted friends to replicate profile data as well as to perform access control. My3 requires each user to grant these friends a *complete* access to her profile data. It also assumes that these friends will enforce the access control benevolently, and will not abuse the data they have access on. We argue that these are very strong assumptions! Furthermore, My3 relies on the assumption that each user has a small subset of friends whose online time patterns cover the online times of all other friends. This is a patently wrong assumption, and can lead to extremely poor data availability [19].

In Gemstone, each profile owner selects a set of peers to replicate her profile data based on criteria like online experience and social relationship. One important issue in such a design is that profile data very likely will be replicated on peers with churn patterns similar to that of the profile owner. This obviously translates to low profile availability.

S-DATA addresses the aforementioned issue of Gemstone by introducing an external centralized service with global knowledge to select peers with *complementary* churn patterns to that of the profile owner to replicate her data. Based on this knowledge, each member of a replica group hosts profiles of all members. We highlight two issues in S-Data: First, changes in the churn behaviour of any participating peer necessitates the creation of a new group. Second, S-DATA assumes users to trust the centralized service (to not misuse the knowledge of churn patterns).

In addition to the issues discussed above, the aforementioned approaches store the *entire* data in the OSN. In a popular OSN with realistic churn rates [10], this translates into massive volumes of data requiring too many nodes to guarantee 24/7 data availability and consistency.

Lilliput addresses the aforementioned issues, thus contributes to the progress of P2P-based OSNs as follows: First, Lilliput dedicates the P2P storage primarily to the essential social glue. Amount of this data has been shown [10] to be small enough to be maintained by a P2P infrastructure. Second, replica groups are created in Lilliput based on the *current* status of the network, i.e. without any assumptions about churn patterns. This way, Lilliput does not prefer long-lived nodes (over short-lived ones) for data replication, thus does not overload them. In addition, this also implies that nodes do not have to explicitly disclose or exchange churn patterns. Third, Lilliput does not prefer friends for data replication. The result is rather an agile storage service which requires low network resources. Lilliput provides highly available, yet consistent, storage even in the presence of frequent updates.

III. Introducing Lilliput

A. System Overview

Each user who joins the social networking service the first time creates a profile and establishes an overlay to replicate her data. Created overlays are small, connected, and identified by IDs derived from the handles of the users who created them. In particular, overlay sizes can be between three and nine nodes. Such small sizes help to achieve scalability, as they allow for flooding a profile and its updates at affordable cost. We chose a minimum of three nodes because with only two nodes, and one of them dies, the last node may not live long enough till it finds new overlay members and replicate content to them. As for the maximum, we chose nine nodes because our experiments (Section IV-B3) show that the overlay maintenance traffic grows too fast with a higher number of nodes. The exact size can be adapted according to system settings and environment.

Each overlay is registered and can subsequently be found using a discovery service. The discovery service itself is out of the scope of this paper. It can be implemented using any of the well-known approaches (e.g. DHTs or central registries).

B. System Environment Assumptions

We make the following assumptions to make sure that Lilliput is suitable under challenging conditions:

- *Heterogeneity of Devices:* We assume P2P-based OSN applications to run on a variety of devices, including PCs, laptops, tablets, and smart phones. This implies that

the potential to contribute storage, computational, and bandwidth resources to the OSN is heterogeneous.
We argue that requiring nodes to contribute according to their resources is not the best design choice. This is because free-riding peers, a well-known problem in P2P systems (e.g. see [20], [21]), will lie about their resources so that they keep their contributions low without losing benefit. We handle this problem by requiring low, yet equal, resource contributions from all nodes.

- *Churn:* The devices that contribute to the OSN will not be accessible all the time. That is to say, we assume users to connect their devices to the OSN only during the times when they gain a benefit from the OSN.

C. Security Goals and Threat Models

The main goal for building distributed OSNs (like P2P-based ones) is to protect the user data both from centralized OSN providers as well as from external attackers. In the following, we show that the design of Lilliput fulfils this goal.

We use the confidentiality, integrity, availability (CIA) model to guide policies for information security within Lilliput. We describe how availability is achieved in Lilliput in Subsection III-E. As for confidentiality and integrity of user data, they can be achieved by applying existing encryption schemes. In particular, we use the Profile Management Scheme (PMS) [22] which can ensure confidentiality and integrity while minimizing key distribution and storage overheads. Without loss of generality, we adopt the adversary model of [22], i.e. the PPT attacker who can, among other features, access and modify messages.

The PMS scheme, however, does not provide mechanisms for secure messaging. We fill this gap in Lilliput by allowing the message sender to include an encrypted link into the public section of the receiver's profile. This link points to the message (or a set of messages) that is stored at the sender's profile to mitigate the chances to overwhelm the receiver's abilities to receive messages. Such a link is signed both by the message sender as well as by the nodes replicating the profile.

D. Definition of Data Structures

In the following we describe the two main data structures that are maintained for the operation of Lilliput:

1) User Profile: A user profile is a container enclosing all data items owned by a user. It consists of three sections: (i) a header for metadata (size, IDs, etc.), (ii) a payload section containing data to be downloaded upon interest, and (iii) an inbox for user messages. Integrating all data items of one user into one container reduces the overhead of checking integrity, and also simplifies replica maintenance.

In Lilliput, we use the PMS-SK scheme [22] to protect the confidentiality of user data, applying its key handling mechanism as well as the provided user profile operations. This choice is motivated by the demonstrated capabilities of PMS-SK, particularly in protecting user data from unintended access as well as in achieving perfect unlinkability.

As it is proposed in [22], we define all data items to be stored in key-value pairs: "A profile P is modeled as a set of pairs $(a, \bar{d}) \in \mathscr{I} \times \{0,1\}^*$ where $\mathscr{I} \subseteq \{0,1\}^*$ is the set of possible attribute indices a and d corresponding values stored in P. We assume that within a profile P, attribute indices are unique. Furthermore, we assume that each profile P is publicly accessible but is distributed in an authentic manner by its owner $U_P \in U$. Also, every user U owns at most one profile and the profile owned by U is denoted P_U."

2) Candidate List: Each node maintains a candidate list of nodes that can be invited to join the profile's replica group. The list can include both nodes obtained by a discovery service and those encountered during normal operation (e.g. in other replica groups, or during profile requests). The candidate list is shared among all replica group members.

To reduce the number of stale entries in candidate lists, each node removes from its candidate list the candidates that deny invitation requests as well as those that timeout.

E. Bootstrapping and Maintenance Protocols

We describe in this subsection the protocols that are necessary to establish and maintain data overlays (i.e. replica groups) in Lilliput. We assume the existence of P2P overlays with basic services like DHT lookup and peer sampling.

1) Bootstrapping: Each node acquires a unique ID the first time it joins the OSN. This ID is used to identify both the node in overlay operations as well as the profile of the corresponding user within the social networking service. Node IDs in Lilliput are generated either at random or by hashing a unique string (e.g. e-mail address). After acquiring an ID, the node joins and registers in the discovery service. Next, it creates a profile and the corresponding data overlay, establishes an initial candidate list, and starts normal operation.

2) Creation of Data Overlays: To maintain profiles in Lilliput, each profile owner has to establish her own data overlay. For this, the profile owner creates an initial data structure containing the overlay ID (the same as the owner ID). The profile owner then leverage the lookup service to randomly select the allowed minimum number of replica nodes (r_min = 3), and invite the selected nodes to join the overlay. This procedure is repeated till r_min nodes accept the invitation. Next, the profile owner establishes a TCP connection to send the initial data structure to the r_min nodes.

The profile owner will act as leader for her own data overlay as long as her node is online. In the absence of the profile owner, the node whose ID is closest to the overlay ID is elected as a leader using a form of the Bully algorithm [23].[1] More precisely, the first node that detects the absence of the leader will calculate the distances between the overlay ID and the IDs of the overlay members (including its own ID). The closest online node is then elected as the new leader (heartbeat messages are used to check whether a node is online or not).

[1]This is just a design decision to elect a unique leader quickly. However, any other leader election algorithm can be used.

3) The Invitation Procedure: To achieve 24/7 availability, at least one node in each data overlay has to be available at every point in time. To that end, the nodes in the data overlay will select and invite other nodes whenever the number of participating nodes drops below r_min.

As illustrated in Fig. 1, the invitation procedure is led by the overlay leader. More precisely, the overlay leader selects a node from the candidate list (selection strategies are discussed in Subsection III-G), and sends to the selected node an invitation message. This message contains the overlay ID, the leader ID, and optionally the ID of another node in the data overlay with which to perform the initial data transfer.

The invited node can either reject the invitation (e.g. because its storage is overloaded) or accept it and respond with an invitation acknowledgement. In case of rejection, the overlay leader will remove the invited node from its candidate list, and invite the next node in the candidate list. In contrast, in case of accepting an invitation, the invited node will open a TCP connection either with the overlay leader or with the designated download node, if available, and request the data.

After a successful initial transfer, the invited node sends a data acknowledgement to the overlay leader as well as heartbeat messages to the other overlay members. After that, this node is considered an established overlay member.

4) Monitoring the Overlay Status: Since nodes are invited to join data overlays upon demand, the number of available profile replicas (i.e. the current overlay size) should be monitored continuously. To this end, overlay nodes periodically exchange heartbeat messages. More precisely, these messages are used to detect failed or disconnected nodes, thus know when the overlay size drops below r_min. The same messages are also used to measure and disseminate round-trip-times among nodes as well as utilization of node resources.

5) Reconnecting Nodes to Data Overlays: To reduce bandwidth consumption, nodes that come back online after a downtime will first try to reconnect to the data overlays to which they were previously participating. In case the attempt fails, the node removes the corresponding data (so that the freed storage can be used to store data of other data overlays). However, in case the failed attempt was made by the node that originally created the overlay (i.e. the data owner), the node will re-instantiate its data overlay.

F. Application Protocols

Application protocols are the protocols used to read from and write to data overlays. Read and write operations are initiated by a client application and handled by a member of the corresponding data overlay.

In write operations, the contacted node disseminates the written data to the other overlay members. Once all overlay members sent the write request, the first contacted node sends an acknowledgement back to the client application.

G. Node Selection Strategies

As mentioned above, when the size of a data overlay drops below r_min, the overlay leader will select a node from its candidate list and invite it to join the overlay. The node selection strategy might affect the system performance. We examined Lilliput with the following three selection strategies:

1) *Random:* Nodes are selected from the candidate list at random.

2) *equalizeConnections:* With this strategy, the overlay leader attempts to equalize the number of data overlays shared with other nodes, for load balancing purposes. To do so, the node selection algorithm calculates, for each candidate node, a score inversely proportional to the number of overlays (s) the node is sharing with the leader. The node with the highest score is then invited to join the overlay.
   Note that the case $s = 0$ implies assigning the same (highest) score to all candidate nodes that are not sharing any overlay yet with the leader, thus inviting all of them. To avoid this, such nodes are assigned random scores between 0 and the maximum score (s_{max}):

$$score = \begin{cases} s_{max} \cdot rand([0,1]) & \text{if } s = 0 \\ (s_{max} - s) + rand([-0.01, 0.01]) & \text{if } s > 0 \end{cases}$$

   The small random factor ($rand([-0.01, 0.01])$) provokes nodes with the same score to be chosen randomly.

3) *filterShortTimeThenEqualizeConnections:* Here, candidate nodes are scored similar to the *equalizeConnections* strategy. The only difference is that new nodes (e.g. those that have been online for less than two minutes) will be scored zero, thus will not be invited. The rationale behind this strategy is twofold: First, new nodes cannot contribute much to profile availability and simultaneously they cause high synchronization traffic. Second, such nodes may prefer to dedicate their network links in the first few minutes foremost to bootstrapping-related messages.

IV. Evaluation

To assess both the efficiency of Lilliput as well as the availability it provides, we performed an extensive packet-level simulation study. We describe our evaluation assumptions, setup, and parameters in Subsection IV-A. After that, we discuss the results in Subsection IV-B.

A. Assumptions and Setup

1) Churn Assumptions: Churn describes how peers arrive and depart a P2P system over time. It reflects the unreliability of nodes in the system. In addition, measuring the performance of P2P-based systems is always related to the churn model.

We used the KAD trace of Steiner et al. [24], [25], a realistic and widely accepted trace, to generate churn. More precisely, we built a churn generator that generates trace files from the KAD trace with the desired properties.[2] Our churn generator

[2]Our churn generator is available for download at https://www.p2p.tu-darmstadt.de/research/p2p-churn-generator/

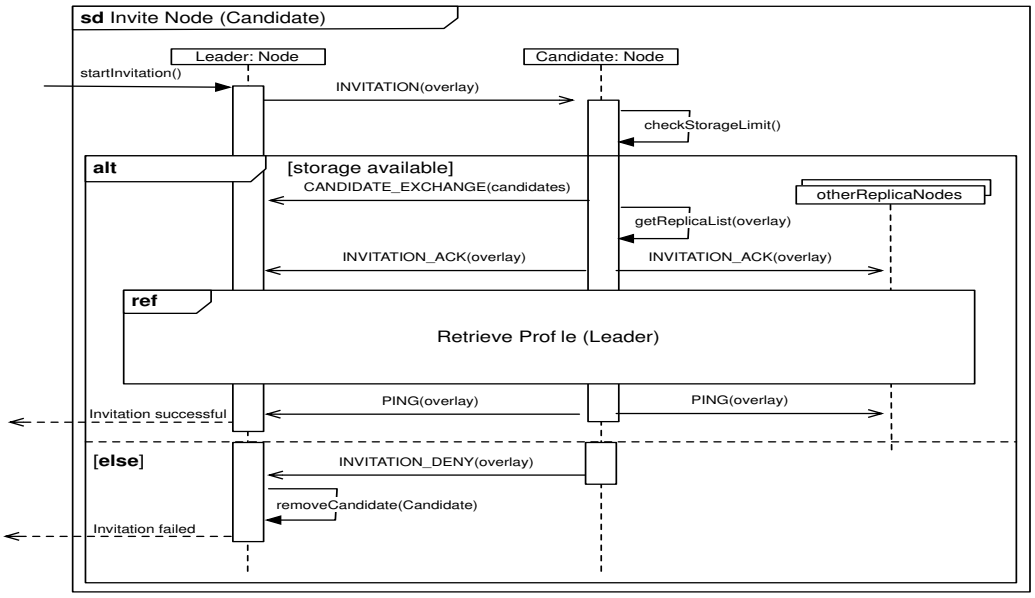

Fig. 1: Sequence diagram of the invitation procedure

generates synthetic trace files for arbitrary numbers of nodes, simulation durations, and time-zone distributions. We derived arrival rates for single time-zones that exhibit a clear diurnal pattern (Fig. 2). Furthermore, in all traces we assumed in average about 11% of nodes to be online and had situations with only 7.9% of nodes in the system (Fig. 3). This makes our churn model more challenging than the churn models used in previous evaluations of P2P-based OSNs.

2) Simulation Environment: We used OMNeT++ [26] (general purpose network simulator) in conjunction with OverSim [27] (overlay network simulation framework). OverSim supports simulations driven by trace files. However, we modified OverSim such that nodes leave the system and return later with their states preserved. Furthermore, we modified OverSim's SimpleUnderlay network topology such that latencies among nodes are assigned based on their time-zones (for this, nodes were placed in an euclidean coordinate system according to their time-zones).

3) Setup and Parameters: We simulated with several OSN sizes of 1000, 5000, 10000, and 15000 nodes, distributed over four different time-zones. Simulating with larger OSN sizes was computationally infeasible (we simulated using a 24-core machine with a 160-GB RAM). We argue that higher simulation abstraction levels which allow much larger network sizes do not reflect realistic timing patterns like packet-level simulations do.

Each node was equipped with a reliable Internet connection (1-Mbit uplink, 10-Mbit downlink) as well as a 500-MB storage space. Latency between two nodes was calculated by summing three components: (i) a fixed value of 20 ms (according to OverSim's SimpleUnderlay topology), (ii) a fraction of the euclidean distance between the positions of the two nodes, and (iii) a random jitter. The maximum latency was set to 600 ms. With such resources, nodes can provide the required contribution to the P2P system.

In the following, we summarize the other parameters:

- *r_min:* The minimum overlay (i.e. replica group) size. We started experiments with $r\_min = 2$, and then increased it. The results showed that r_min values higher than 4 did not help to significantly improve profile availability but caused high resource consumption.
- *r_max:* The maximum overlay size. This value can range between r_min+1 and ∞. Our experiments showed that r_max values greater than 9 result in a very high heartbeat messaging overhead.
- *profileSize:* We simulated with a profile size of 10 MB. According to the results of [10], this size is enough to store what we call the *essential social glue* (recall that Lilliput is not designed for long term storage of bulk data).
- *maxStorageSize:* The device owner may want to limit the storage which is used by Lilliput to host other users' profiles. We evaluated the effect of this limit on profile availability and network load with three values: 500 MB, 1000 MB, and ∞.

4) Measurements: Each experiment lasted for 14 simulated days. The first five days were used as a warm-up period. The results reported below are based on the observations from day 6 till day 14. In particular, we measured the consumed storage space, the bandwidth utilization, the number of connections as well as the number of overlays each node participated in. For the data overlays, we reported the overall availability over

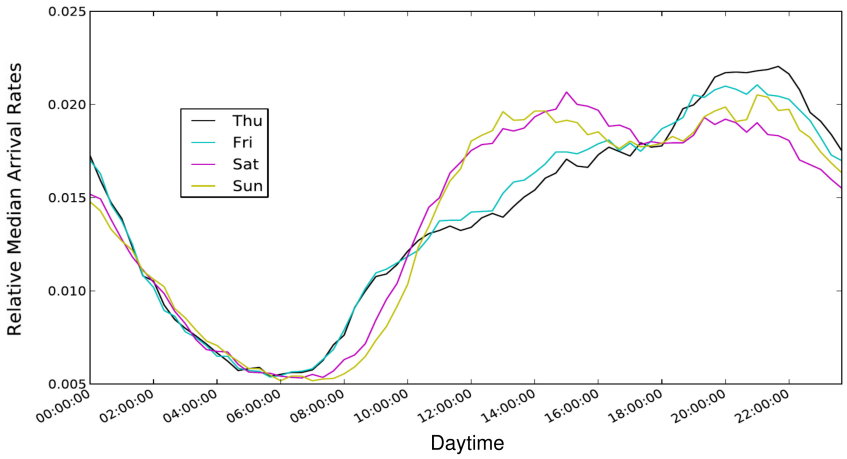

Fig. 2: Diurnal churn patterns during weekdays (Monday till Wednesday are not included since they are similar to Thursday)

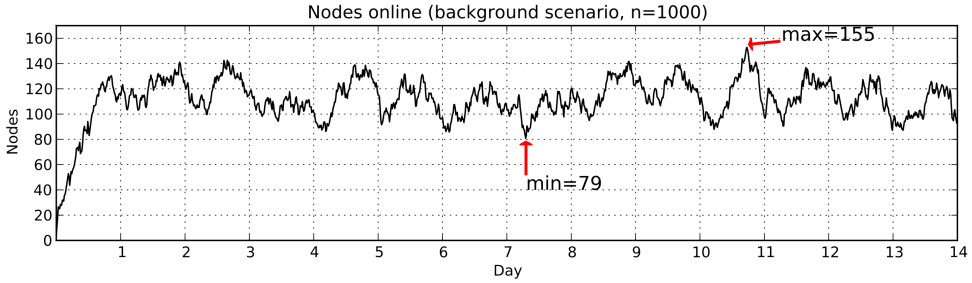

Fig. 3: Number of online nodes: diurnal patterns are cumulated and weighted (according to [24], [25]) across all time-zones

time, the number of concurrent active nodes and the total number of invited nodes. Finally, we measured the ratio of data overlays available with respect to the total number of overlays created over time.

Overlay size	Overlays available 24/7 (%)	Total time availability (%)
1k	97.6	99.64
10k	96.4	99.18
15k	96.37	99.07

TABLE I: Availability of data overlays

B. Simulation Results

The results we discuss below represent simulations applied the *filterShortTimeThenEqualizeConnections* node selection strategy, motivated by its features (see Subsection III-G).

As a first step we justify the size of the experiment and show that increasing the number of nodes by factor 15 does not change the results in general (Fig. 4). In the following, we present and discuss the most important numerical results, mainly focusing on: profile availability, communication overhead, and storage utilization.

1) Profile Availability: Excluding the warm-up period, we discovered parameter combinations that led to an availability above 99.07% (Table I). With r_min = 3 and r_max = 8, 96.37% of the profiles have never been offline during the whole simulation time. The profiles that have been offline at least once had a median availability of 84.28%.

2) Communication Overhead: The communication overhead is an important issue, especially in mobile environments. Users may reject to use an OSN application that utilizes a large part of the available bandwidth. We measured the communication overhead by the amount of data that is sent and received by each node (Fig. 5).

The bandwidth utilization is driven by the assumed churn, the replication parameters (r_min, r_max), the maximum storage space, and sizes of the stored objects. The churn influence on bandwidth utilization can translate into the probability of nodes to rejoin overlays in relation to those needing an entire new copy of the data.

Fig. 6 shows the amount of traffic needed to keep data online for 14 days (i.e. experiment time), under different combinations of replication parameters (r_min, r_max) as well as two different maximum storage spaces (500 MB and 1 GB). We can see that significantly increasing r_min would increase the bandwidth utilization while increasing r_max would decrease it. These results can be explained as follow: On

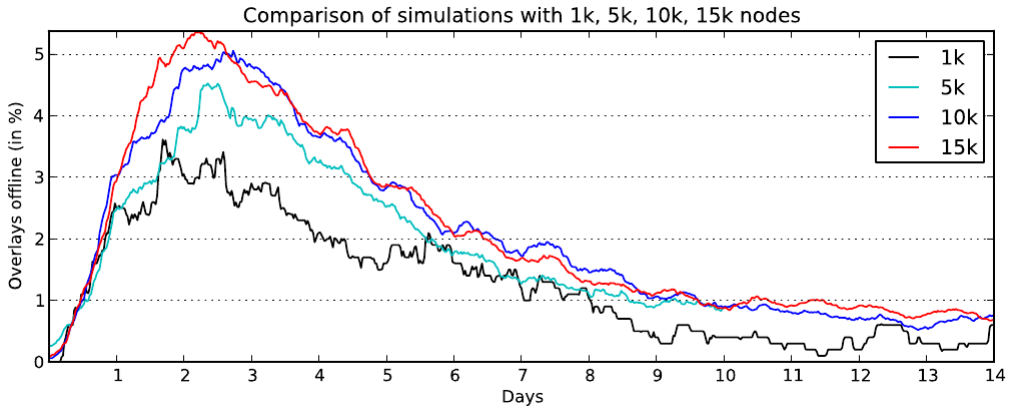

Fig. 4: Influence of the network size on the availability of profiles (settings: r_min = 3, r_max = 7)

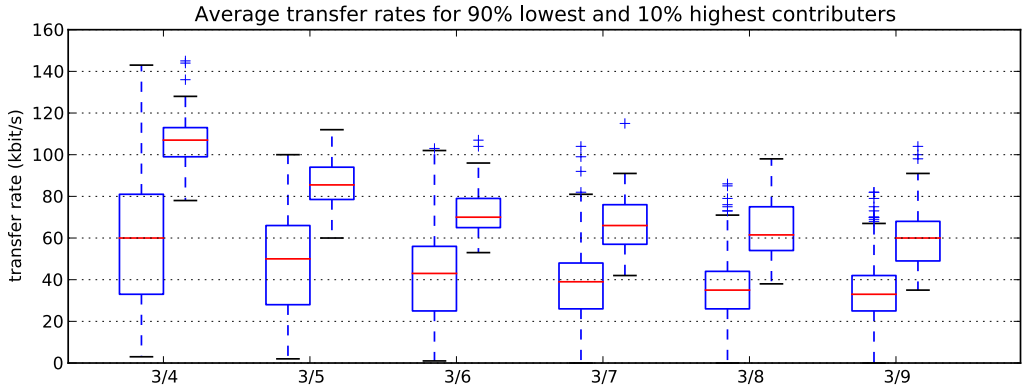

Fig. 5: Average transfer rates with different combinations of r_min, r_max and a maximum storage of 500 MB; in each two adjacent bars, the left bar represents the 90% quantile of the nodes contributing the lowest amount of bits while the right one represents the 10% quantile of highest contributing nodes

the one hand, increasing r_min means increasing sizes of data overlays, thus increases number of flooding messages as well as invitations. On the other hand, increasing r_max means increasing the probability that nodes rejoin data overlays, thus the probability to update a stale copy of a profile (rather than transferring a new copy of the data). As for the influence of the maximum storage space, we can see that increasing it from 500 MB to 1 GB would result in reducing the traffic. This is because a higher storage capacity translates into a higher probability that nodes rejoin data overlays.

Fig. 7 compares the amounts of sent data to the amounts of received data per node, with respect to r_min (changing r_max did not impact the results). The differences between the two (sent and received) are attributed to nodes leaving the system during data transfer (i.e. caused by heavy churn).

3) Optimized Parameter Selection Strategy: The r_min parameter needs to be at least 3 to achieve a reasonably high availability (Fig. 8). In spite of slowly increasing payload data transmission effort per node, the total traffic in the system quickly explodes when setting the value of r_min to 4 or higher (Fig. 7). This is attributed to the increasing amount of overlay maintenance traffic. We thus argue that the value 3 is the best choice for r_min.

The parameter r_max should be set to 8 to achieve the highest profile availability while minimizing the total traffic in the system (Fig. 6, Fig. 8). Increasing r_max above 8 does not improve availability. However, in case of limited storage (500 MB), $r\_max > 8$ even reduces the availability. This is because it becomes harder to find nodes with spare resources during the invitation process.

The storage utilization in Lilliput strongly depends on the chosen values of r_min and r_max. To determine the effect of a limited storage on the system performance, we experimented (as mentioned in Subsection IV-A3) with three different storage sizes: 500 MB, 1000 MB, and ∞. We found that limiting the storage usage to 500 MB does not cause any performance disruption. With such a limited storage, as can be seen in Fig. 9, each node exhausts its storage capacity when $r\_min > 2$.

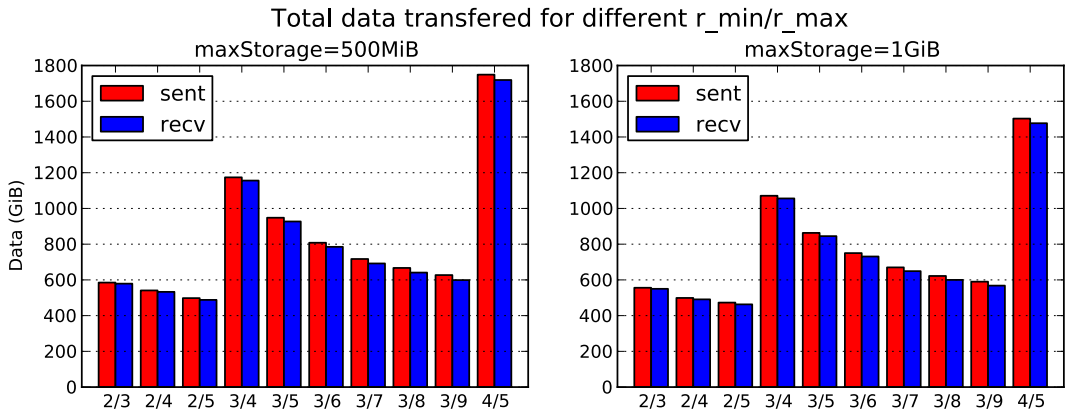

Fig. 6: Influence of different combinations of r_min, r_max on the amount of transferred data; network size = 1000

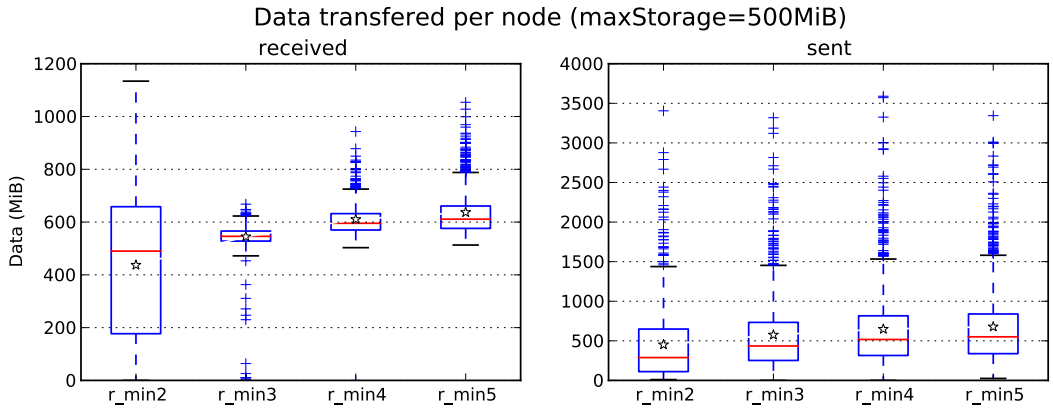

Fig. 7: Amounts of sent data and received data per node; the star markers indicate the average values; outliers indicate early or late adopters during the bootstrap phase

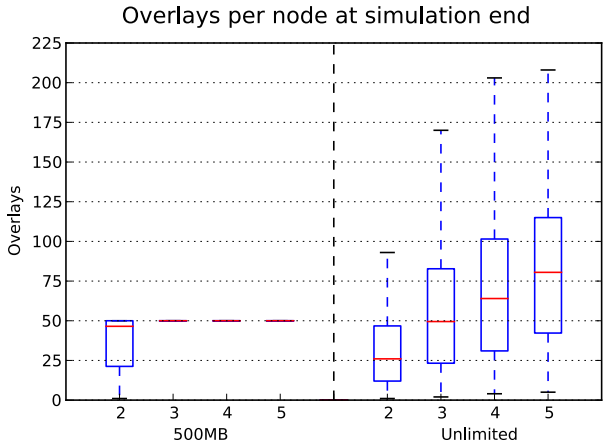

Fig. 9: Number of overlays each node is participating in at the end of the simulation time with respect to r_min

V. Discussion

A. Comparison to Related Work

In contrast to Lilliput, the related works discussed in Section II use static replication schemes to ensure data availability in distributed OSNs. Static replication, however, requires to find a minimal set of nodes that are expected to provide 24/7 availability with their accumulated online periods. This works well only when assuming either long online durations of at least a minor fraction of stable nodes or stable diurnal online time patterns of a majority of nodes in the replication set.

We consider S-Data [18], based on its features and properties, the closest related work to Lilliput. It has been evaluated under a much more conservative churn model. Even though, it achieves a notably lower profile availability than Lilliput. In particular, the authors state that: “For a mean peer uptime of 8 hours it is possible to have more than 93% of the groups online even under 50% failure rate”.

B. Data Consistency

Content freshness is not an issue for Lilliput. This is because the content is replicated in a fully connected overlay in which participating nodes share identical copies of the content. More precisely, the profile owner has to be online to update her profile. When the profile owner is online, she serves access requests to her profile data from the original copy. The profile

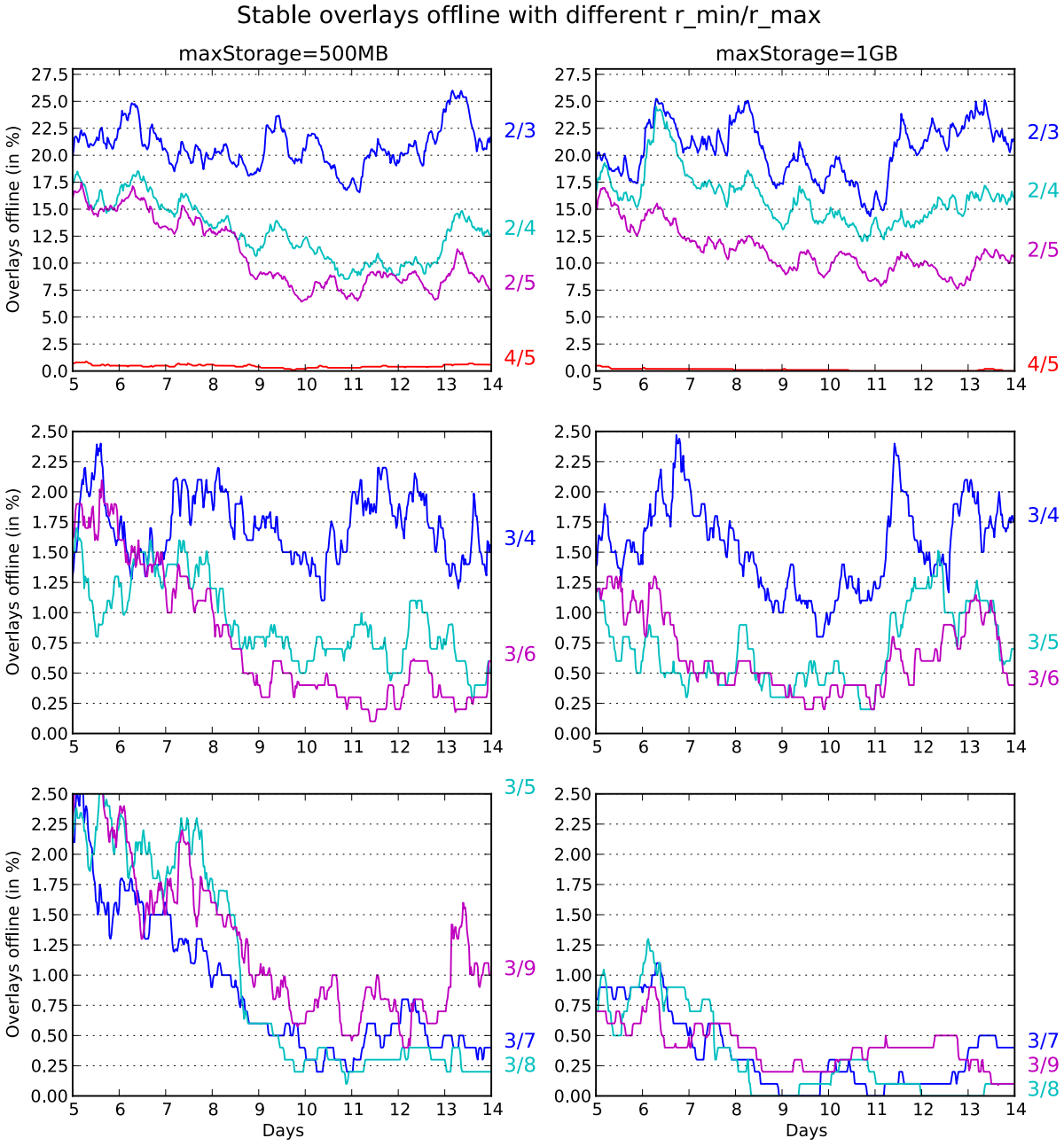

Fig. 8: Fraction of overlays that have been offline at least once, with different combinations of r_min, r_max values

owner, being part of her own overlay, makes sure that the replicas are up-to-date. This is done by uploading her profile to the replicating nodes. The replicating nodes can subsequently serve access requests (that arrive while the profile owner is offline) with the latest copy of the requested data.

Two cases in which the consistency is challenged do remain: (i) if a breakdown occurs during a profile update (from the profile owner to replicating nodes, or among replicating nodes), and (ii) if a breakdown occurs after a message has been sent, but before it has been delivered to the profile owner. We do not see any chance to tackle the first case. Regarding the second one, the only chance for the message sender to make sure that the message arrived is to check back later.

C. Robustness Against Selfish Nodes

Selfish nodes are the ones that do not offer any storage resources but burden the system by replicating their data. This is possible either if the incentive scheme is gamed or in case a sybill attack creates multiple identities and successfully establishes multiple replica overlays.

The robustness of Lilliput against selfish nodes depends on the spare resources that exist in the system. During our experiments, we found that the total amount of storage per node, on average, needs to be at least 50 times larger than the profile size to achieve high data availability while minimizing bandwidth utilization. That is to say, with a maximum profile size of 10 MB, every node should provide only 500 MB of local storage on average. However, when half of the nodes provide 1 GB of storage, the other half could potentially act selfishly without affecting the functionality.

VI. Conclusion

We introduced Lilliput, a storage primitive for P2P-based OSNs. Lilliput is placed in the middle of the solution space between the two extremes: DHTs and static replication groups, aiming to combine their advantages. The novel idea of Lilliput is to disentangle static bulk data from the essential social data, and to devote the P2P storage foremost to the latter. This way, Lilliput significantly reduces the size of replicated data, thus reduces the bandwidth utilization as well as the storage overhead. Lilliput provides reciprocity of nodes in serving and replicating profiles to motivate cooperation. It also benefits from the high likelihood to serve the same content by only updating profiles instead of transmitting entire profiles.

By extensive simulations, we showed that Lilliput can be deployed under heavy churn and still maintains data redundancy to achieve an outstanding profile availability (above 99%).

Acknowledgement

This work was supported by the German Research Foundation (DFG), the Collaborative Research Center (SFB) under the MAKI (1053) project, the German Society for International Cooperation (GIZ) – project no. 15.2011.3-003.32, and by a Singapore-German Research Collaboration Project (1N Program) on reliable and secure data storage in decentralized social networks.

References

[1] S. Buchegger, D. Schioberg, L. Vu, and A. Datta, "PeerSoN: P2P Social Networking – Early Experiences and Insights," *SNS*, 2009.
[2] L. A. Cutillo, R. Molva, and T. Strufe, "Safebook: Feasibility of transitive cooperation for privacy on a decentralized social network," in *WoWMoM*, 2009.
[3] R. Sharma and A. Datta, "Supernova: Super-peers based architecture for decentralized online social networks," in *COMSNETS*. IEEE, 2012.
[4] M. Durr, M. Maier, and F. Dorfmeister, "Vegas – a secure and privacy-preserving peer-to-peer online social network," in *PASSAT*, 2012.
[5] S. Nilizadeh, S. Jahid, P. Mittal, N. Borisov, and A. Kapadia, "Cachet: a decentralized architecture for privacy preserving social networking with caching," in *CoNEXT*. ACM, 2012.
[6] L. M. Aiello and G. Ruffo, "Lotusnet: Tunable privacy for distributed online social network services," *Computer Communications*, vol. 35, no. 1, pp. 75–88, 2012.
[7] S. Jahid, S. Nilizadeh, P. Mittal, N. Borisov, and A. Kapadia, "Decent: A decentralized architecture for enforcing privacy in online social networks," *SESOC*, 2012.
[8] T. Maqsood, O. Khalid, R. Irfan, S. A. Madani, and S. U. Khan, "Scalability issues in online social networks," *ACM Computing Surveys (CSUR)*, vol. 49, no. 2, p. 40, 2016.
[9] X. Zuo and A. Iamnitchi, "A survey of socially aware peer-to-peer systems," *ACM Computing Surveys (CSUR)*, vol. 49, no. 1, p. 9, 2016.
[10] T. Paul, D. Puscher, and T. Strufe, "The user behavior in facebook and its development from 2009 until 2014," *CoRR*, vol. abs/1505.04943, 2015.
[11] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer *et al.*, "Oceanstore: An architecture for global-scale persistent storage," *ACM Sigplan Notices*, vol. 35, no. 11, pp. 190–201, 2000.
[12] M. Karnstedt, K.-U. Sattler, M. Richtarsky, J. Muller, M. Hauswirth, R. Schmidt, and R. John, "Unistore: querying a dht-based universal storage," in *ICDE*, 2007.
[13] A. Cutillo, R. Molva, and T. Strufe, "Safebook: a privacy preserving online social network leveraging on real-life trust," *IEEE Communication Magazine*, 2009.
[14] K. Graffi, S. Podrajanski, P. Mukherjee, A. Kovacevic, and R. Steinmetz, "A distributed platform for multimedia communities," in *International Symposium on Multimedia*, 2008.
[15] G. Mega, A. Montresor, and G. P. Picco, "Efficient dissemination in decentralized social networks," in *IEEE P2P*, 2011.
[16] R. Narendula, T. G. Papaioannou, and K. Aberer, "A decentralized online social network with efficient user-driven replication," in *PASSAT*, 2012.
[17] F. Tegeler, D. Koll, and X. Fu, "Gemstone: Empowering decentralized social networking with high data availability," in *GLOBECOM*, 2011.
[18] N. Shahriar, S. R. Chowdhury, M. Sharmin, R. Ahmed, R. Boutaba, and B. Mathieu, "Ensuring β-availability in p2p social networks," in *(ICDCS), Workshop*, 2013.
[19] R. Sharma, A. Datta, M. del Amico, and P. Michiardi, "An empirical study of availability in friend-to-friend storage systems," in *IEEE P2P*, 2011.
[20] S. Jun and M. Ahamad, "Incentives in bittorrent induce free riding," in *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*. ACM, 2005, pp. 116–121.
[21] D. Carra, P. Michiardi, H. Salah, and T. Strufe, "On the impact of incentives in emule: Analysis and measurements of a popular file-sharing application," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 9, pp. 94–104, 2013.
[22] F. Günther, M. Manulis, and T. Strufe, "Cryptographic treatment of private user profiles," in *FC*. Springer, 2012.
[23] Q. E. K. Mamun, S. M. Masum, and M. A. R. Mustafa, "Modified bully algorithm for electing coordinator in distributed systems." *WSEAS Transactions on Computers*, 2004.
[24] M. Steiner, T. En-Najjary, and E. W. Biersack, "Long term study of peer behavior in the kad dht," *IEEE/ACM TON*, 2009.
[25] ——, "A Global View of KAD," in *ACM SIGCOMM*, San Diego, California, USA, 2007.
[26] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in *SIMUTools*, 2008.
[27] I. Baumgart, B. Heep, and S. Krause, "OverSim: A Flexible Overlay Network Simulation Framework," *IEEE GIS*, 2007.