Capacity Management Protocol for a Structured P2P-based Online Social Network

Newton Masinde, Sebastian Bischoff and Kalman Graffi
Technology of Social Networks
Heinrich Heine University,
Universitätsstrasse 1, 40225 Düsseldorf, Germany
email: {firstname.lastname}@hhu.de
web: http://tsn.hhu.de/

***Abstract*—Peer-to-peer (P2P) networks were first made popular by file-sharing applications such as KaZaA, Napster, LimeWire and BitTorrent. These early applications on P2P networks relied exclusively on always connected users to ensure that the services were available by utilizing some form of centralized indexing services. With the rapid technological developments, the computing devices have become increasingly smaller and are connected via mobile networks run by teleecommunication companies. Additionally, more sophisticated applications that are even more resource intensive such as online social networks (OSNs) on P2P platforms are possible. The smaller devices that are connected to the P2P OSNs may not have adequate resource capacities, such as bandwidth, memory or storage space, to share with the P2P network. In this paper, we present an algorithm that handles capacity management as he new users join the network. We implement and test the algorithm on our P2P framework for OSNs called LibreSocial, and show that network functions well even when there is a significantly high number of limited capacity devices join the network.**

***Index Terms*—Distributed networks, peer-to-peer network, online social network, heterogeneous networks**

I. Introduction

Peer-to-peer (P2P) networks are composed of highly heterogeneous systems, which differ based on the resource capacities such as processing speed, storage availability and bandwidth, participant dynamics in terms of resources they are willing to contribute to the network or software artifacts such as system default configurations [1]. During the early onset of research on P2P overlays, much emphasis was placed on file-sharing applications. Recently, other more sophisticated web-based applications have emerged. Most recent applications are the P2P-based online social networks (OSNs), such as LibreSocial [2] (previously LifeSocial.KOM [3]–[5]), DECENT [6], LotusNet [7] among others. Recently, it has been seen that users are now more concerned with data privacy and online anonymity. In addition, the ever-increasing system maintenance costs consequent to the nearly exponential increase of system users have lead to scalability concerns for OSNs on the centralized platform [8]. The combination of these two issues have contributed extensively to the emergence of P2P OSNs. Generally, OSN users connect to the network using mostly heterogeneous devices, with an increasing number of users now preferring the use of mobile devices in accessing their social profiles. At face-value it may appear that P2P networks as a platform for offering OSN services may directly support heterogeneous devices, but this may not be the case because to fully realize the benefits due to heterogeneity, the P2P-based OSN must incorporate accurate models that capture the peer heterogeneity well [9]. The possible advantages due to the presence of reliable heterogeneity management in the overall network include load balancing [10], flexible replica placement [11] and better throughput [12]. However, heterogeneity management features must be explicitly accounted for in the design of the system to ascertain practicality and scalability [13].

The growth of the Internet has been paralleled by the proliferation of computing devices with varying computing capabilities (such as storage, processing speed and bandwidth), from the traditional desktop computers to much smaller, more mobile devices such as laptops, palmtops and mobile phone devices. In many instances, these devices connect to the Internet via a mobile telecommunication network. As a consequence, the Internet has evolved to support this heterogeneity. However, the heterogeneity exhibited in terms of computing capabilities of the interconnected devices presents a challenge in P2P network, especially if the number of strong computing devices are below a certain threshold. We differentiate between *"strong"* and *"weak"* computing devices based on the resource capacity, with strong devices having sufficient resources and weak devices lacking in one or more resource aspect, similar to KaZaA. In most instances, weak devices tend to be mobile and are a large contributor to churn in the P2P network. Thus, the presence of more weak devices than may be tolerable in the network presents a considerable challenge in guaranteeing efficient routing, data availability and ensuring a stable network. In any case, since the average user connects to their profile via weak devices at some point, the design should anticipate the presence of a large number of such devices in the network and facilitate them.

In this paper, we present a capacity management algorithm to help in handling a heterogeneous network to identify the different devices and their capacities. The rest of the paper proceeds as follows. In Section II, we begin by discussing the problem of resource-based (or capacity-based) management in structured P2P network, looking at some of the proposals that have been presented in studied literature to solve this challenge. In Section III, we briefly present our P2P-based OSN solution LibreSocial [2] (previously LifeSocial.KOM [3]–[5]) focusing on the overlay design. Thereafter we discuss our protocol to handle capacity management in LibreSocial and present our experimental setup in Section IV.

II. Related Works

One of the obvious strengths of P2P networks is that the peers can potentially cooperate with each other and share resources, information and services. This ability is only met if the network can efficiently discover the peer resources available and the capacities they can handle as well as offer to the network. The P2P overlay network designs must take into consideration resource optimality for possible large-scale deployment in environments such as the Internet, else it may necessitate complete redesign in future iterations [14]. Therefore, the challenge of achieving capacity-aware heterogeneity support in P2P network has been, and continues to be, a subject of research, as it is a function of the overlay. The overlay design should include: *1*) a proper selection of an identifier (ID) space, *2*) a mechanism to map resources and peers to the ID space, *3*) a mechanism to manage

the ID space by the peers, *4*) a logical structure of the network *5*) a routing strategy, and *6*) a maintenance strategy [15].

Many of the proposed P2P OSNs are hosted on structured overlays such as Chord [16], Pastry [17] or Kademlia [18], which utilize Distributed Hash Tables (DHTs). DHTs rely on uniform hash functions, such as SHA-1, and therefore there is an assumption of uniform distribution of objects stored in the DHTs to guarantee that no peer is overloaded. Also, there is the aspect of uneven file requests, both in terms of number and file sizes, hence varying loads based on storage and bandwidth consumption. These application design complexities must be carefully studied and systematically addressed to develop a system that is capacity-aware. The use of DHTs enable efficient locating of files for quick insertions, querying and deletions based on key-based routing which promises high reliability, high scalability and low deployment costs [19]. However, there are two challenges that are evident from using the DHTs in the design of the applications:varying object load and varying node capacities [20].

A. Varying object loads

This focuses on handling of the *varying loads* within the network. DHT algorithms usually incorporate load balancing techniques, but the interpretations of what a load-balanced network refers to vary in literature such as assigning equal number of files to peers, ensuring a file count proportional to peer's capacity or peers' consumption [20]. But this does not address the case of varying file sizes or presence of popular files leading to high loads at a particular peer. This therefore necessitates more robust techniques. Load balancing techniques have been classified into three groups object placement, routing and underlay [21].

1) Object placement: This plays a key role in ensuring that the identifier space is balanced, while preventing loading imbalances due to requests. The use of *namespace balancing* techniques such as consistent hashing and its derivatives in Chord, Pastry and Kademlia, *virtual servers* first introduced in Chord, or *multiple hash functions* such as in CAN [22] helps to uniformly distribute identifiers and keys over the identifier space. *Caching* and *replication* techniques are useful in reducing load imbalance as multiple copies of the data items are distributed among nodes. By including additional mechanisms such as discussed in [23] to handle selection of objects to replicate, selection of replica peers and balancing of load requests among peers, this can be achieved.

2) Routing: The manner in which the lookup queries are serviced, that is, the routing algorithm that the P2P network implements is the point of focus. Generally, each node will maintain its own *routing table* filled with a mapping of identifiers and corresponding IP addresses of neighboring peers. Most DHT-based routing algorithms are greedy in nature and therefore may lead to path convergence resulting in overloading of a few nodes with forwarded requests [24], which necessitates routing load balancing. This is realized through *link reorganization* where the routing strategy is unchanged but the link organization is altered as in Pastry [17], or *path redundancy* wherein the link organization is unchanged but the routing strategy is changed as in HyperCup [25].

3) Underlay load balancing: Load balancing requires traffic optimization in the underlying physical network, as there is no guarantee that the nodes and links in the overlay and underlay are perfectly matched. Hence, nearby nodes may receive distant overlay identifiers. This therefore requires the exploitation of network proximity, achieved via three methods. The first is use of *topology-based IDs* which includes information on the physical location such as in CAN [22], but this may destroy uniform distribution of identifiers in the address space. The second is the use of *proximity neighbor selection* which permits multiple eligible nodes such as in Pastry [17], with the algorithm selecting the closest node based on other aspects such as geographical distance or hop counts for the routing entry. The last method is *proximity routing* which may be used when overlay construction does not consider the underlying topology, so that the forwarding node may forward a packet to the physically closest eligible node to the destination.

Having covered the important aspects of consideration for handling object loads in a P2P environment, the next important point of analysis is how to handle node capacities. This is discussed next.

B. Varying node capacities

Handling node capacities is addressed by paying attention to two important factors, that is, *capability* and *availability* of the network nodes. Capability lays an emphasis on the heterogeneous nature of the nodes considering the differences exhibited in the computing abilities (processing power, bandwidth, memory and storage space). The network should handle and make use of such differences when they arise via suitable algorithms that monitor these features. These algorithms should allow participating nodes with resources capacities that meet certain predefined threshold values to publish themselves while ensuring that other nodes are able to utilize these resources [26]. The second factor, availability, emphasizes the length of time the nodes are in the network. The more capable and available nodes are the more stable is the network, and scalability is increased. As a possible solution for keeping track of node availability, [27] suggests the use of a dynamic `NodeID` assignment in which a nodes assigned load-balanced ID (`LBID`) if it tends to be more available and hence reliable, and a load-free ID (`LFID`) otherwise. Therefore, in general, if node capability and availability are considered carefully and incorporated into the system design, it should be expected that the maintenance overheads will undoubtedly increase but the upside is that the overall performance of the network will be far much better.

III. Overlay Design of LibreSocial

LibreSocial [2] is a P2P-based OSN designed on a structured P2P network. It utilizes a heavily modified FreePastry[1], an open source implementation of the Pastry done by Rice University. The architecture of LibreSocial has been presented in detail in previous publications [2]–[5]. We primarily focus on how LibreSocial achieves heterogeneity support, in particular, load and capacity management.

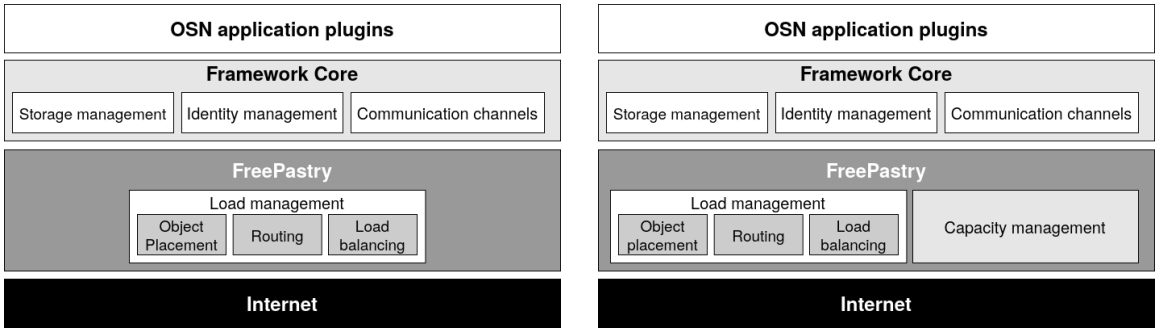
A. Modified Freepastry

As previously mentioned, LibreSocial is designed atop a heavily modified Freepastry which is a DHT-based. The following are the core services that the overlay offers in LibreSocial

Identity space: The DHT offers support for routing based on an identity space of size 2^{160} that is cyclic, that is, the peer with the lowest ID succeeds the peer with the highest ID. Thus peer IDs are 160-bit values. The peer `nodeID`s is the public key from the generated asymmetric key using elliptic curve cryptography (ECC) during registration. Thus all communication to the node is encrypted using the recipient's `nodeID`, which is the public key, and signatures are easily verifiable using the same.

Profile and `userID` management: During the registration, when the peer is able to join the network, a profile item is created. This may be fully or partially encrypted using the AES encryption algorithm and stored with the `nodeID`. As the `nodeID` is mutable, an immutable `userID` is also generated during profile creation and mapped to the `nodeID` using a mapping-item which is replicated in

[1]https://www.freepastry.org/



(a) LibreSocial's architecture

(b) Proposed capacity management feature added

Fig. 1. Monitoring setup

the network for redundancy. Since the `nodeID` (which is the public key) is stored as a signed data item under the `userID`, other peers are able to get the location of their contact in the network by simply knowing the `userID` rather than the `nodeID`.

Routing: Message routing is done by performing prefix matching until the recipient is reached, or the peer having the longest matching prefix is reached which possibly manages that identity space. The routing algorithm guarantees message routing in $\lceil O(log_2 N) \rceil$ hops, and utilizes two other structures, the routing table and the leafset. Entries in the routing table are made in such a manner that the distances from the peer are in exponentially increasing distances. The routing table is organized into $\lceil O(log_{2^b} N) \rceil$ rows, b representing a configuration parameter, with a typical value of 4, and the total $2^b - 1$ entries per row. The routing table is extended so that each position is a bucket of multiple peer addresses similar to the Kademlia DHT. With this, a requesting node sends parallel requests to k different peers, waits for the k responses and then iterates the requests to the new k parallel peers. Thus it implements parallel and iterative routing. The leafset contains L entries that are numerically close, such that $|L/2|$ entries are numerically smaller (predecessors) and $|L/2|$ entries are numerically larger (successors).

B. Load management in LibreSocial

Fig. 1a is the overview of LibreSocial's current architecture. As LibreSocial is designed using Freepastry as the overlay, hence a DHT, it offers load management, which includes the following strategies:

- *Object placement*: Implements consistent key mapping [28] for namespace balancing which guarantees object load balancing, and caching and replication to ensure request load balancing.
- *Routing*: Implements link reorganization, as there are many possible routes between any two given nodes. Thus a redundant link is simply replaced with another existing one without altering the average path length of subsequent lookup queries. This guarantees that the overlay is balanced
- *Underlay*: Implements proximity neighbor selection to allow multiple eligible nodes. Therefore, the closest node based on some metric such as of hop count, delay, geographical distance and so on, is chosen to fill in a particular routing entry. This ensures a balance in the network underlay

IV. Approach for Capacity Awareness

The introduction of capacity management into LibreSocial as modeled in Fig. 1b requires a delicate balance between monitoring the hardware specifications of the user's device (capacity awareness) and making of adjustments to the load management. We make note of the following limitations that impact the implementation of a primitive but effective capacity awareness algorithm into LibreSocial so as to take into account load management as it is currently implemented.

- *Replica placement*: Only strong nodes are to be utilized for replica placement. In case a weak node cannot find enough strong nodes to perform replication on, then this node temporarily is identified as a strong node.
- *Routing*: Weak nodes do not participate in routing and therefore do not have a routing table. In the case that a weak node becomes a strong node, it will temporarily create a routing table, which will be erased once it is downgraded to a weak node again. However, to ensure that the communication is possible, the leafset is no interfered with.
- *Underlay*: The proximity neighbor selection implemented in FreePastry is not altered thus the closest node is always selected to maintain a balance in the underlay.

A. A capacity awareness algorithm

Based on the limitations described, we describe a capacity awareness algorithm for LibreSocial. Let $T\{T_s, T_w\}$ be the set containing hardware parameters that identifying if the current device should be considered strong (T_s) or weak (T_w). A device may only satisfy T_s or T_w but not both. Also Let $C\{C_{bw}, C_{bc}, C_m, C_s\}$ be a set containing the threshold capacity values for bandwidth (C_{bw}), battery capacity (C_{bc}), memory (C_m) and storage (C_s). The values of T and C are then useful in determining the state of the current user's device as illustrated in Algorithm 1. Algorithm 2 describes the capacity handling process which works as follows. Generally, each node in the network periodically checks and saves its own status by running Algorithm 1. When two nodes are communicating, the status of each node is passed on as part of the identity payload overhead, which the other node can now use to verify its neighbor's status and store it in a hashmap, with the `neighborID` as the key and the state (`true`/`false`) as the value. A value of `true` indicates a weak node and `false` indicates a strong node. The status hashmap contains all the known `neighborID`'s with their currently state which is updated with every subsequent communication. If the node that is processing this identity information is a weak node, then it does not build a routing table (or routing set), otherwise it is built. In case the communicating node is identified as a weak node, then the node is not added to the replica set or is removed if it is present. In this way, it is guaranteed that only strong nodes receive routing requests and also store replicas hence reinforcing heterogeneity awareness.

B. Experimental setup

The capacity handling algorithm is evaluated using a benchmarking process. A benchmark is a tuple of *quality attributes* (or *properties*)

```
Input  : T{T_s, T_w}, C{C_bw, C_bc, C_m, C_s}
Output: weakStatus
hwConfigs ← hardwareCheck ()
if hwConfigs ⇒ T_s then
    if C{} not satisfied then
        weakStatus ← true
    endif
    else
        weakStatus ← false
    endif
endif
else if hwConfigs ⇒ T_w then
    weakStatus ← false
endif
else
    weakStatus ← false
endif
return weakStatus
```

Algorithm 1: isWeak

```
Input  : neighborID, myID
Output: nodeStatusHashmap, routingSet, replicaSet
nodeIsWeak ← getWeakStatus (neighborID)
iAmWeak ← isWeak ()
if iAmWeak ⇒ true then
    routingSet ← ∅
    if nodeIsWeak ⇒ false then
        replicaSet.add(neighborID)
    endif
    else if nodeIsWeak ⇒ true then
        nodeStatusHashmap.add(neighborID,true)
        if replicaSet.contains(neighborID) ⇒ true then
            replicaSet.remove(neighborID)
        endif
    endif
endif
else if iAmWeak ⇒ false then
    nodeStatusHashmap.add(neighborID,false)
    routingSet.add(neighborID)
    replicaSet.add(neighborID)
endif
```

Algorithm 2: capacityHandling

Q, *metrics M*, and test scenarios *S* [29]. We discuss each of the aspects of the benchmark process for the capacity management protocol in the following.

1) Quality properties and metrics: The main considerations for the quality is the performance of the system due to the capacity management introduced. We consider the netwok and storage performance with the following metrics identified.

- Network performance - Node count, maximum and average routing table size; average number of lookup requests and lookup time; average number of hop counts; maximum network data and message rates per node.
- Storage performance - Average local objects and maximum replicas stored per node; maximum callback data stored and maximum callback storage data rate; average storage and retrieval time.

2) Test scenario: LibreSocial's includes two plugins, that is, test and monitoring plugins, which allow us to conduct benchmarking tests. A detailed discussion on how they work is described in [30]. Our test bed is the high performance computing (HPC) cluster provided by Heinrich Heine University[2]. To test the capaciy handling algorithm in the HPC cluster, Algorithm 1 was modified to Algorithm 3. This change allows manipulation of the ratio between weak and strong nodes in the network. Algorithm 3 works by taking the value of the port assigned for the node and performs a modulo operation to determine if the status will be a weak or strong node. The modulo operation is based on a value provided called the `weakRatio`. For testing, the values of `weakRatio` were 2, 3, 4 and 5 to cause the total number of weak nodes in the network to be approximately 50%, 67%, 75% and 80% respectively. If the modulo operation returns a 0, the node was classified as a strong node, else a weak node. The exception to this rule was when the port number corresponded to the main bootstrap node also called the master node, or when the number of strong nodes stored by a node is less than the replication factor. The use of the replication factor to check the initial status when booting the node helps to ensure that the nodes do not fail due to absence of required possible replication nodes. The test networks had 200 nodes in total. The test workload is shown in Table

[2]https://www.zim.hhu.de

```
Input  : myPortNo, masterPortNo, replicationFactor, weakRatio
Output: weakStatus
strongNodes ← nodeStatusHashmap.count(value ⇒ false)
if myPortNo ⇒ masterPortNo or replicationFactor > strongNodes then
    weakStatus ← false
endif
else if myPortNo mod weakRatio ⇒ 0 then
    weakStatus ← false
endif
else
    weakStatus ← false
endif
return weakStatus
```

Algorithm 3: isWeak (for HPC test)

TABLE I
TEST WORKLOAD.

Plugin	Action	Repetitions *(Count)*	Duration * *(mins)*
Messaging	Send message	10	5
	View inbox message	10	5
	View outbox message	10	5
Live chat	Send multichat invitation	10	5
	Send multichat message	10	5
Group	Create group	10	5
	Invite friend to group	10	5
	View group	10	5
	View my group list	10	5
File storage	Create folder	10	5
	Upload file in folder	10	5
	View folder	10	5
Forum	Create forum thread	10	5
	Comment forum thread	10	5
	Viewt forum	10	5
Photos	Create photo album	10	5
	Upload photo	10	5
	View friend's albums	10	5
	View own albums	10	5
Calendar	Create calendar event	10	5
	Edit calendar event	10	5
	View calendar event	10	5
Voting	Create vote	10	5
	Add public vote	10	5
	Voting invite user	10	5
	Vote	10	5
	Get my votings	10	5
	Get voting results	10	5

* Two minute pause after completion of the repetitions for each test case.

V. EXPERIMENTAL RESULTS

The following is an analytical discussion of the results gathered from the benchmarking process for the capacity management protocol for LibreSocial.

A. Network analysis

The results are shown in Figs. 2a-2l. We present an interpretation of the meaning and impact of each set of results, beginning with the network analysis followed by the storage analysis.

a) Node count: Figs. 2a and 2b portray the nature of the state of the weak and strong nodes in the network throughout the duration. In a general sense, the strong nodes appear to be more stable throughout, while in the case of the weak nodes, it is noteworthy that as the ratio of weak nodes to strong nodes increases, the stability of the weak nodes is evidently affected, as evidenced by the significant drop of about 30 nodes for 80% test and 10 nodes for 75% test.

b) Network data rate: This is depicted in Fig. 2c. For 50%, 67% and 75% tests, there is a slight increase in the data rate when they are compared. However, for 80% tests, we note a significant increase from 120 minutes to about 150 minutes and thereafter, the network data rate is less than the other 3 experiments. This sudden surge in data transfer correlates with the drop in the number of nodes, in specific the weak nodes, and may actually be caused by network maintenance data as the network adjusts to the failing nodes back to a stable state.

c) Network message rate: The results are shown in Fig. 2d. Significant in the results shown is an increase in he message rate as the ratio of weak nodes increases. From the results of the network data rate in Fig. 2c, it is not immediately evident that there is high generation of maintenance messages for the 75% test. However, this is clearly visible when looking at the messaging rate for 75% test which occurs between minutes 160 and 200. For the 80% test, the high messaging that occurs between minutes 120 and 150 corresponds with the high data rate. In both cases, this is attributed to the maintenance messages. The other are seen to have regular and stable message characteristics.

d) Maximum and average routing table size: The maximum routing tables for the strong and weak nodes are shown in Figs. 2e and 2g. These show that the largest route set for any individual strong node did no exceed about 28 nodes. In the case of the weak nodes, we observe that there were some weak nodes that were designated as strong nodes at the joining phase but after a few moments in the network, they changed to weak nodes hence the routing sets were removed as expected. Figs. 2f and 2h on the other hand show the average sizes of the routing tables for the strong and weak nodes. In the case of the strong nodes, which generate a routing set and therefore have routing tables, we observe a rise and drop in the average routing table size that corresponds wo the reduction in the network size due to failing weak nodes. Taking note of the fact that FreePastry utilizes link reorganization to maintain routing, and this clearly impacted the routing tables for the strong nodes. The weak nodes did not have any routing tables as required.

e) Average lookup requests answered: The results are portrayed in Figs. 2i and 2j for the strong and weak nodes. The results for the strong nodes are as would be expected, that is, as the ratio of strong to weak nodes get smaller, the responses to requests by the strong nodes should increase significantly. This trend is also expected for the weak nodes, and even though the weak nodes are not expected to hold replicas, they can respond for objects that they own. This expectation is clearly matched by the results with an anomaly observed, where in, there are more responses with the 75% test than the 80% test. It is possible that the failing of the weak nodes had quite a significant impact on responses to lookup requests directed to the weak nodes. However, this may require further investigation to understand the cause.

f) Average lookup time: Figs. 2k and 2l give the overview of the average lookup time characteristics of the network for the strong and weak nodes respectively. The lookup times are all in the order of milliseconds, which we consider acceptable. We note the significant surge in lookup time for both the strong and weak nodes from minute 100 to 150 for the 80% test. This correlates with the time period when there was failure of some nodes. Also the observed increased rerieval time at the beginning of the experiment may be due to lookup of bootstrap nodes.

B. Storage analysis

The results are given in Figs. 3a-3l and discussed in the following.

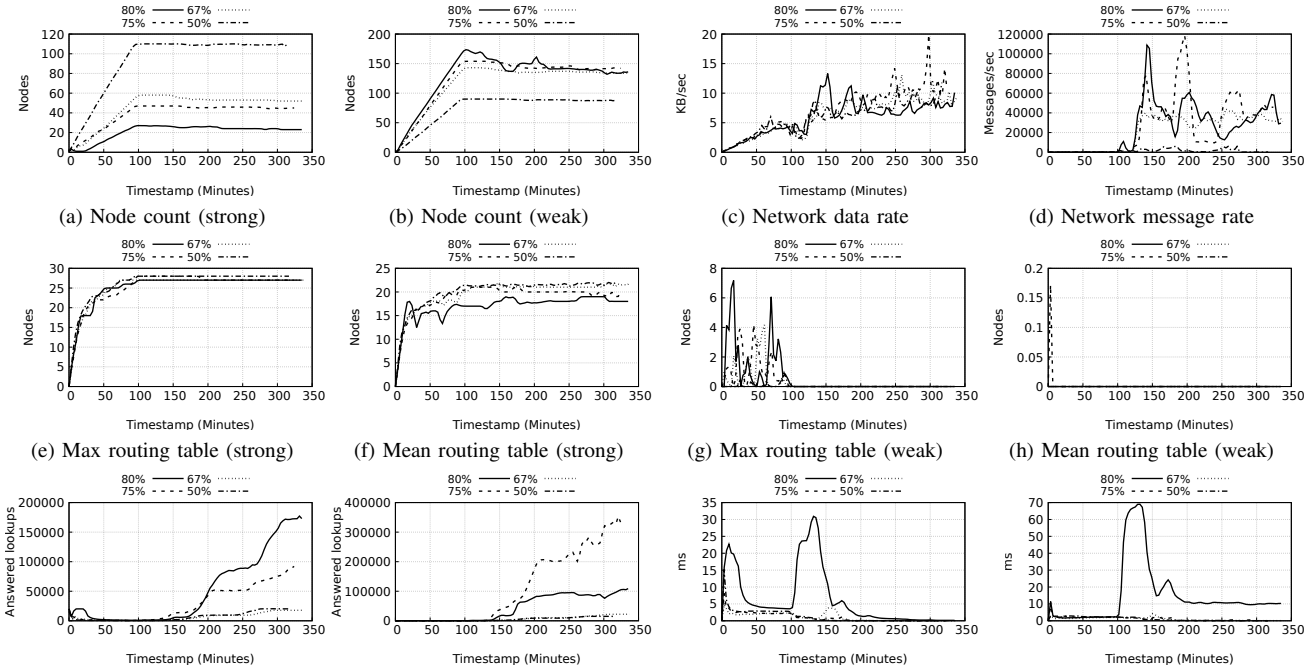
a) Average local objects stored: Figs. 3a and 3b for the strong and weak nodes respectively. Both the strong and weak nodes show a decrease in the objects stored as the ratio between the weak and strong nodes gets larger. Generally, it would be expected that the average number of stored local objects should be quite similar in both the strong and weak nodes.

b) Average replica objects: For the strong and weak nodes, the results are shown in Figs. 3c and 3d respectively. The strong nodes have four times as many replica objects as the weak nodes in all tests. These results are evidence that the capacity management works to ensure that weak nodes do not get replicas, hence overloading the scarce resources that they may have.

c) Maximum retrieval time: These are given in Figs 3e and 3f for strong and weak nodes respectively. For the strong nodes, there are a few instances that abnormally long retrieval times occur with the maximum recorded time being about 150 seconds at 290th minute for the 75% test and about 25 seconds at 205th minute for the 67% test and 15 seconds at 110th minute for the 80% test. For the weak nodes, the 80% test showed abnormally long retrieval times reaching over 550 seconds twice, a the 10th minute mark and 310th minute. This leads us to conclude that data retrieval is greatly affected by the presence of too many weak nodes in the network.

d) Maximum storage time: The results are shown in Figs. 3g and 3h for the strong and weak nodes respectively. For the strong nodes, the maximum time taken to store an item becomes unusually long as the number of weak nodes in the network increases. This is evident with the 75% test with a noted maximum store time above 200 seconds occurring at the 260th minute, and the 80% test showing a maximum storage time of about 1600 seconds at the 160th minute and another spike of about 900 seconds at the 325th minute. For the weak nodes, the store time values are even larger. with the values reaching 3300 seconds at 310 minutes, 1000 at 270 minutes and 600 seconds at 170 minutes for the 80%, 75% and 67% tests respectively. This dismal results for the store time are due to the obvious reduction in the number of strong nodes in the network resulting in delays in storing data.

e) Callback data: Figs 3i and 3j shows the callback data stored and the callback storage data rates. LibreSocial employs push-based message passing approach that utilizes callback functions. One of the callback functions utilized is called the continuation function which is used for asynchronous operations. In this, the callback is called to further process a task that was put on hold, for example, a storing operation where some portion of the execution code is register as a continuation which continues execution after the store operation is completed. The unusually high store times are thus directly attributed to the continuation call back function, which can be seen by comparing the callback storage data rate in Fig. 3j and the maximum store time for the weak nodes in 3h. As the number of strong nodes decrease, meaning fewer nodes to store the replicated data, the available nodes take more time to process the storage requests resulting in increased callback data and consequently increased storage time.

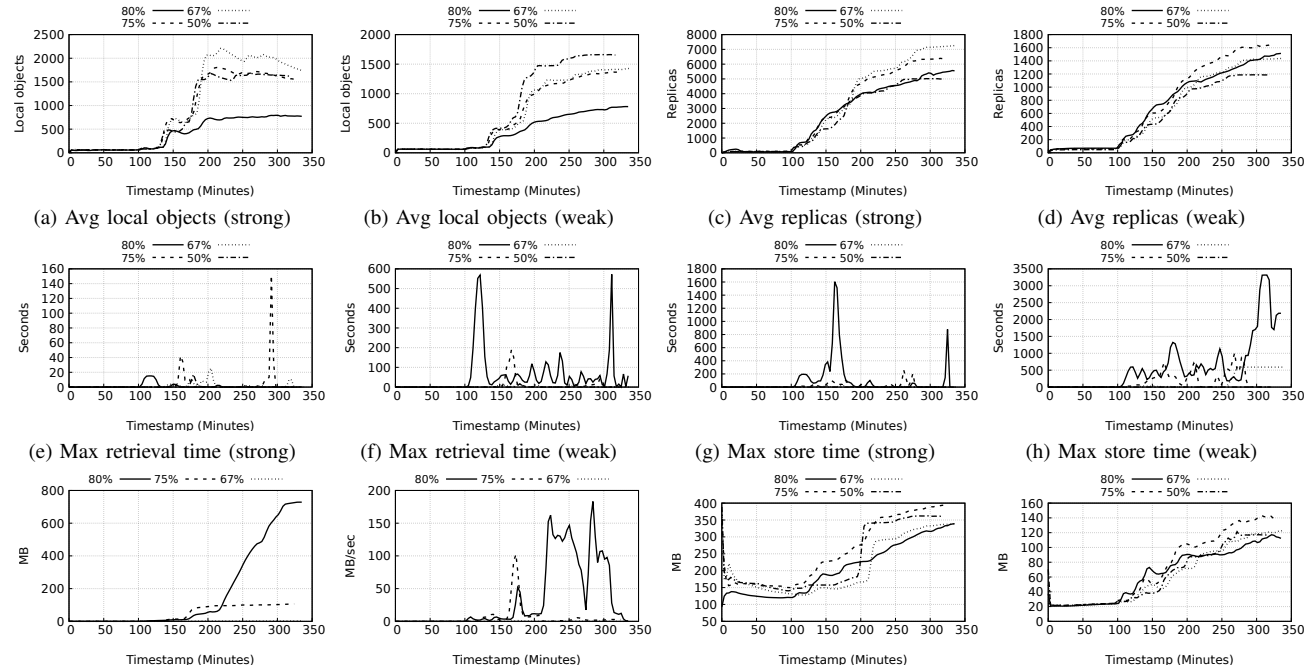


(a) Node count (strong) (b) Node count (weak) (c) Network data rate (d) Network message rate

(e) Max routing table (strong) (f) Mean routing table (strong) (g) Max routing table (weak) (h) Mean routing table (weak)

(i) Avg lookup requests (strong) (j) Avg lookup requests (weak) (k) Avg lookup time (Strong) (l) Avg lookup time (weak)

Fig. 2. Network impact analysis



(a) Avg local objects (strong) (b) Avg local objects (weak) (c) Avg replicas (strong) (d) Avg replicas (weak)

(e) Max retrieval time (strong) (f) Max retrieval time (weak) (g) Max store time (strong) (h) Max store time (weak)

(i) Callback data stored (j) Callback storage data rate (k) Avg memory contributed (l) Avg memory utilized

Fig. 3. Storage statistics analysis

f) Average memory contributed and utilized: The results are shown in Figs. 3k and 3l for contributed and utilized averages respectively. Each participating node in the network shares a portion of its memory space for the network. There is a slight decrease in the memory contributed as well as memory utilized on average as the number of weak nodes increase. However, the exception to this trend is seen with the 67% test which shows slight increase rather than decrease. It is possible that the weak nodes contribute less memory as they do not participate as network stores for replication, and also are not actively engaged in the routing process. The largest disparity between the average maximum memory contributed and utilized between the tests is about 60 and 40 MB at the end of the experiment.

Summary: The network is seen to be generally very stable despite having a large number of weak nodes as in the case of the 80% test, meaning that the capacity management is functioning as intended. During the network startup phase, all 200 nodes join. However, as the load is introduced, the effect of the weak nodes in the network immediate especially with the 80% test, for which about 35 nodes fail. This means that the introduction of capacity management in the system, a limitation is immediately realized, that is, the number of strong network nodes have to be sufficiently high to ensure that the network does not collapse due to overloading of the strong nodes with storage requests. Retrieval requests are not adversely affected as weak nodes can also service such requests. If there is high network churn, particularly weak nodes, then the lookup times are affected due to link reorganization at the strong nodes.

VI. Conclusion

In this paper, we have presented a capacity management algorithm that works in conjuction with the load management structure of our P2P framework for OSN called LibreSocial. The capacity management algorithm ensures that nodes joining the network are assigned the correct node status based on the type of device they are using and the capacities of the devices, with limited capacity devices referred to as weak nodes, otherwise they are referred to as strong nodes. We have shown that the LibreSocial network is able to render itself stable and remain active even after introduction of a high number of weak node (80%). In addition, we see that the weak nodes are spared from participating in the storage of replicated data effectively. However, such a high number of weak nodes had a significant effect on the network function with increased messaging and lookup times, as well as the storage functions causing abnormally high storage and retrieval times.

Acknowledgement

Computational infrastructure and support were provided by the Centre for Information and Media Technology at Heinrich Heine University Düsseldorf

References

[1] D. Stutzbach, R. Rejaie, and S. Sen, "Characterizing unstructured overlay topologies in modern p2p file-sharing systems," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, pp. 267–280, Apr. 2008.

[2] K. Graffi and N. Masinde, "Libresocial: A peer-to-peer framework for online social networks," *arXiv preprint arXiv:2001.02962*, 2020.

[3] K. Graffi, P. Mukherjee, B. Menges, D. Hartung, A. Kovacevic, and R. Steinmetz, "Practical security in P2P-based social networks," in *Proceedings of the IEEE 34th Conference on Local Computer Networks, 2009. LCN 2009.* IEEE, October 2009, pp. 269–272.

[4] K. Graffi, C. Gross, P. Mukherjee, A. Kovacevic, and R. Steinmetz, "LifeSocial.KOM: A P2P-based Platform for Secure Online Social Networks," in *Proceedings of the IEEE International Conference on Peer-to-Peer Computing (P2P'10)*. IEEE, August 2010, pp. 1–2.

[5] K. Graffi, C. Gross, D. Stingl, D. Hartung, A. Kovacevic, and R. Steinmetz, "LifeSocial. KOM: A secure and P2P-based solution for online social networks," in *Proceedings of the 2011 IEEE Consumer Communications and Networking Conference (CCNC)*. IEEE, January 2011, pp. 554–558.

[6] S. Jahid, S. Nilizadeh, P. Mittal, N. Borisov, and A. Kapadia, "DECENT: A decentralized architecture for enforcing privacy in online social networks," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*. IEEE, Mar. 2012, pp. 326–332.

[7] L. M. Aiello and G. Ruffo, "Lotusnet: Tunable privacy for distributed online social network services," *Computer Communications*, vol. 35, no. 1, pp. 75–88, January 2012.

[8] T. Maqsood, O. Khalid, R. Irfan, S. A. Madani, and S. U. Khan, "Scalability Issues in Online Social Networks," *ACM Computing Surveys*, vol. 49, no. 2, pp. 40:1–40:42, sep 2016.

[9] Z. Yang, Y. Xing, F. Xiao, Z. Qu, X. Li, and Y. Dai, "Exploring peer heterogeneity: Towards understanding and application," in *2011 IEEE International Conference on Peer-to-Peer Computing*. IEEE, Aug. 2011, pp. 20–29.

[10] P. B. Godfrey and I. Stoica, "Heterogeneity and load balance in distributed hash tables," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 1. IEEE, March 2005, pp. 596–606.

[11] P. Skowron and K. Rzadca, "Flexible replica placement for optimized p2p backup on heterogeneous, unreliable machines," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 7, pp. 2166–2186, Mar. 2016.

[12] G. Tian, Y. Xu, Y. Liu, and K. Ross, "Mechanism design for dynamic p2p streaming," in *IEEE P2P 2013 Proceedings*. IEEE, Sep. 2013, pp. 1–10.

[13] Š. Girdzijauskas, A. Datta, and K. Aberer, "Structured overlay for heterogeneous environments: Design and evaluation of oscar," *ACM Trans. Auton. Adapt. Syst.*, vol. 5, no. 1, Feb. 2010.

[14] V. Darlagiannis, A. Mauthe, and R. Steinmetz, "Overlay design mechanisms for heterogeneous, large-scale, dynamic p2p systems," *Journal of Network and Systems Management*, vol. 12, no. 3, pp. 371–395, 2004.

[15] Š. Girdzijauskas, "Designing peer-to-peer overlays a small-world perspective," Ph.D. dissertation, Lausanne, 2009.

[16] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 27-31, 2001, San Diego, CA, USA*. ACM, 2001, pp. 149–160.

[17] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001, Proceedings*, ser. Lecture Notes in Computer Science, vol. 2218. Springer, 2001, pp. 329–350.

[18] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, ser. Lecture Notes in Computer Science, P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, Eds., vol. 2429. Springer, 2002, pp. 53–65.

[19] L. Li, X. Xu, J. Wang, and W. Wang, "Ds2: A dht-based substrate for distributed services," *Peer-to-Peer Networking and Applications*, vol. 6, no. 4, pp. 380–396, 2013.

[20] P. Wette and K. Graffi, "Adding capacity-aware storage indirection to homogeneous distributed hash tables," in *2013 Conference on Networked Systems*. IEEE, Mar. 2013, pp. 35–42.

[21] P. Felber, P. Kropf, E. Schiller, and S. Serbu, "Survey on load balancing in peer-to-peer distributed hash tables," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 473–492, Jan. 2014.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '01. New York, NY, USA: ACM, 2001, pp. 161–172.

[23] E. Spaho, L. Barolli, and F. Xhafa, "Data Replication Strategies in P2P Systems: A Survey," in *2014 17th International Conference on Network-Based Information Systems*. IEEE, Sep. 2014, pp. 302–309.

[24] S. Serbu, P. Kropf, and P. Felber, "Fault-tolerant p2p networks: how dependable is greedy routing," in *Workshop on Dependable Application Support in Self-Organising Networks (DASSON'07)*, 2007, pp. 1–5.

[25] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl, "HyperCuP - Hypercubes, Ontologies, and Efficient Search on Peer-to-Peer Networks," in *Agents and Peer-to-Peer Computing*, G. Moro and M. Koubarakis, Eds. Berlin, Heidelberg: Springer, 2003, pp. 112–124.

[26] N. J. Navimipour and F. S. Milani, "A comprehensive study of the resource discovery techniques in peer-to-peer networks," *Peer-to-Peer Networking and Applications*, vol. 8, no. 3, pp. 474–492, 2015.

[27] K. Kim, "Dynamic nodeid based heterogeneity aware p2p system," *Computer Communications*, vol. 32, no. 6, pp. 1062 – 1071, 2009.

[28] A. Haeberlen, J. Hoye, A. Mislove, and P. Druschel, "Consistent key mapping in structured overlays," CS Department, Rice University, Technical Report TR05-456, 2005.

[29] A. Kovacevic, K. Graffi, S. Kaune, C. Leng, and R. Steinmetz, "Towards Benchmarking of Structured Peer-to-Peer Overlays for Network Virtual Environments," in *Proceedings of the IEEE International Conference on Parallel and Distributed Systems*. IEEE, 2008, pp. 799–804.

[30] N. Masinde, L. Khitman, I. Dlikman, and K. Graffi, "Systematic Evaluation of LibreSocial–A Peer-to-Peer Framework for Online Social Networks," *Future Internet*, vol. 12, no. 9, p. 140, 2020.