A Distributed Platform for Multimedia Communities

Kalman Graffi, Sergey Podrajanski, Patrick Mukherjee, Aleksandra Kovacevic, and Ralf Steinmetz
Multimedia Communications Lab[1] and Real-Time Systems Lab[2], Technische Universität Darmstadt
Email: {graffi[1],sandra[1],steinmetz}@kom.tu-darmstadt.de, mukherjee[2]@es.tu-darmstadt.de

***Abstract*— Online community platforms and multimedia content delivery are merging in recent years. Current platforms like Facebook and YouTube are client-server based which result in high administration costs for the provider. In contrast to that peer-to-peer systems offer scalability and low costs, but are limited in their functionality. In this paper we present a framework for peer-to-peer based multimedia online communities. We identified the key challenges for this new application of the peer-to-peer paradigm and built a plugin based, easily extendible and multifunctional framework. Further, we identified distributed linked lists as valuable data structure to implement the user profiles, friend lists, groups, photo albums and more. Our framework aims at providing the functionality of common online community platforms combined with the multimedia delivery capabilities of modern peer-to-peer systems, e.g. direct multimedia delivery and access to a distributed multimedia pool.**

I. Introduction

Nowadays, social interaction over the Internet is an ongoing trend. Since the arise of blogs and the possibility to publish user generated content, the number of active users increased significantly. Once the self-presentation with blogs was established, platforms like YouTube [2] and Last.FM [3] arose, in which users can share their videos and music taste. It was a short step from self-presentation to interactivity, finding discussion partners on the same topic of interest.

Currently, online community platforms are the next step towards a seamless interaction with friends and people one know. Millions of users worldwide are attracted by e.g. MySpace[3] (110M), Facebook[3](98M), StudiVZ[3] (5M) and Xing[3] (5M), to which we refer in the following.

On these platforms, users can present themselves and interact in different ways with their friends. According to Alexa.com [4] the sites are ranked in the top ten most clicked sites (MySpace and Facebook worldwide and StudiVZ in Germany), which shows the importance of web-based interaction of Internet users. These online community platforms are currently limited to the exchange of text messages and pictures, more challenging applications offering personalized video blogging, multimedia content sharing and support for (haptical) interactions are still to come.

However, current architectures for multimedia community platforms are client-server based, which result in both a scalability problem (which can only be avoided with investing a significant amount of money) and a limited innovation progress (as there is only one provider). Architectures based on the peer-to-peer (p2p) paradigm has proved to be self-scalable (new users contribute their resources) in various applications for file sharing, video streaming and telephone conferencing. Building a p2p based framework for multimedia online communities promises to resolve the scalability problem and to provide further interaction possibilities like live (video supported) chatting, group-based shared folders for collaboration purposes and more interactive tools. Severe challenges arise while building such an architecture.

Contribution: In this paper we present an extendible framework for multimedia online communities. Focus of this work is to identify the challenges for building an distributed online community platform based on p2p technology. We implemented the framework and present how to organize the data structures in the distributed online community platform and how to keep the architecture modular and still easy to extend for the upcoming trends in multimedia research.

First, we introduce in Sec. II the characteristics of multimedia community platforms and identify the costs, quality requirements and challenges when applying the p2p paradigm. In Sec. III we discuss related work and show what knowledge can be reused and where current solutions are limited. We present our p2p based framework for multimedia online communities in Sec. IV, discuss design decisions in Sec. V and draw a conclusion in Sec. VI.

II. Problem Statement and Requirements

Building a distributed solution for multimedia online communities states several challenges. Regarding the features it has to provide, it should allow common functionality of today's online community sites (like Facebook). We identified following services as common: 1) Registration of new users. 2) Presenting the user's profile and provide individual settings customizable by the corresponding user. 3) Grouping of users in friends list, user-generated groups and organizational networks. 4) Search functionality for users and groups. 5) Direct communication and presenting of multimedia content.

These functionality is typically provided by a centralized platform. This comes with high costs for the platform provider. For a distributed framework providing the same functionality we also identified following non-functional requirements.

The load for maintaining the infrastructure and providing the above mentioned services should be distributed on the users of the infrastructure. By distributing the load, peers are expected to participate in the network and to provide a reasonable amount of storage space.

[1,2]Authors supported by the German Research Foundation, Research Group 733, "QuaP2P: Improvement of the Quality of Peer-to-Peer Systems" [1].

[3]www.myspace.com, www.facebook.com, www.studivz.de, www.xing.de

Security is a challenging issue in a distributed multimedia online community. Following security features are needed most. First, authenticity of (maintenance) messages and objects in the system. Further, confidential, integer and authenticated direct communication and access control on the objects. Only owners should be able to change their data and to define who is allowed to access it. We sketch a security solution but keep the focus on the framework.

A distributed framework needs to be extendible in order to be able to fulfill the requirements of upcoming multimedia trends. Extendability of the framework enables the vendor or even the users to add new functionality by writing small plugins that can easily be integrated into the existing application.

In the next section we discuss approaches for the above mentioned requirements. Client-server based solutions may fulfill the requirements, they come with high costs for the vendor. P2P based applications address partially some functions of a multimedia online community, e.g. communication, and object sharing, but they do not address all at once. In Sec. IV we present our solution addressing all requirements on a distributed framework for multimedia online communities.

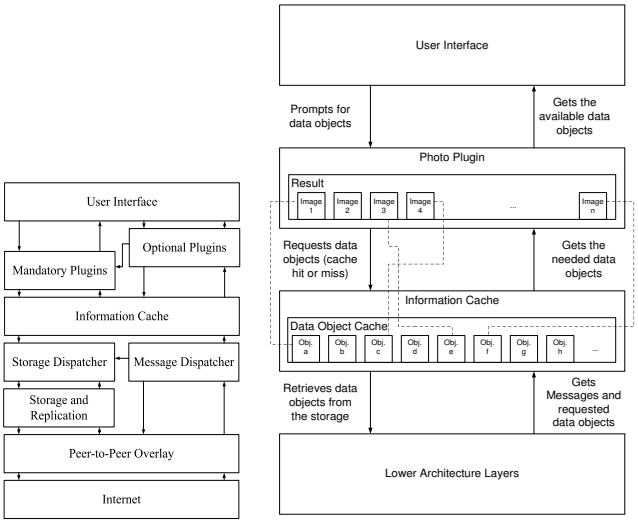
III. Related Work

Current platforms for multimedia online communities follow the client-server paradigm, one vendor provides the servers which provide the service to the users. While it is quite convenient for the users, the vendor carries all the costs.

According to [5] annual administration expenses for the Facebook vendor are estimated to be 1.05$ per single platform user, while having 98 Million users in 2007 [6]. It is straight forward to assume that administration costs which the Facebook vendor has to bear sum to about 98 Million of dollars per year. The most of these expenses are server administration costs. For YouTube, LastFM and MySpace the principle is the same. There as well, the most crucial resources, storage space and bandwidth, are provided by servers and the capacities of the clients are unused. A second limitation of client-server based solutions is the limited innovation progress. Typically only the provider is able to add new features and functions to the system, which limits the extendability of the platform.

The p2p paradigm provides an alternative for building a service oriented multimedia platform. Basically, in a p2p system the consumers and users of a service are building the infrastructure that provides the service. In a file sharing scenario, the users' devices form an overlay in which desired content can be found. The p2p paradigm can help to drastically lower the costs for a provider [7], as administration costs and the service load are shared among the users of the system.

Several applications demonstrate the advantages of the paradigm. For file sharing applications several overlays have been proposed. Unstructured overlays like FastTrack [8] or BubbleStorm [9] allow to search for the desired content using keywords. Structured overlays like Pastry [10] and Kademlia [11] map objects in the system to peers using a specific scheme and allow to lookup objects according their ID. Whereas file sharing does not state strict requirements on the quality of service (QoS), Internet telephony tools like Skype [12] or video streaming applications like Zattoo[4] or Joost[4] state QoS requirements. For telephony, low delay and jitter is crucial, for video streaming high throughput as well. However, these popular p2p applications do not address multimedia online communities, no large-scale group interaction is supported.



(a) Architecture Blocks (b) Usage Scenario of a Plugin

Fig. 1. Layer-based Architecture for Multimedia Online Communities

Several distributed storage applications like OceanStore [13] address the reliable handling of large data objects, but do not offer user interaction capabilities. Wuala[4] enhances file-sharing with communication functionality, but does not offer common collaboration functions of multimedia online communities. Groove[4] is a p2p based collaborative tool providing some limited support for group interaction, like message boards, chatting and shared folders. However, it only scales to a dozens of users in a group and does not support millions like in common multimedia online communities.

All presented applications provide a valuable and optimized function, but to our best knowledge, there is no p2p based multimedia online community platform. We present a distributed extendible framework combining the described functions and fulfilling the requirements stated in Sec. II.

IV. A Framework for Multimedia Online Communities

The architecture for multimedia online communities has to provide many functionality. Routing, distributed data storage and community specific functions are some among them. In order to combine these functions to a framework we use a layered model. Each layer provides specific functionality to upper layers by using the services provided by lower layers.

A. Main Building Blocks of the Framework

For a multimedia online community we have several functional requirements, which are presented in Fig. 1(a). A layer for routing, for storage and message handling, a cache and

[4]www.zattoo.com, www.joost.com, www.wua.la, www.groove.net

several plugins implementing base features of the multimedia online community are needed and discussed next.

P2P Overlay: A structured p2p overlay, i.e. a Distributed Hash Table (DHT), maps objects to peers according their object ID. Further, it allows to route messages to peers responsible for specific IDs, i.e. *routeMessage(nodeID, message)*. In our implementation we used FreePastry [14] which implements Pastry [10]. It is well maintained and widely used.

Storage and Replication: A storage and replication layer uses the DHT and allows to store data objects in a distributed and reliable way. This is essential in our platform as user information like profiles, friendship states, photos and videos have to be stored reliably even if the corresponding user goes offline. We assume the DHT functionality *put(key, object)*, *get(key)* and *delete(key)*. In our implementation we used PAST [15], an extension to FreePastry, which we extended with *delete(key)* by replacing with an empty object.

Storage Dispatcher: Whereas the Storage and Replication layer offers the reliable storage of data objects, the Storage Dispatcher processes the application specific data to make it storable. Further, it provides additional functionality like removing or modifying data and is in charge that the storage operations are performed. As we operate in a distributed and unreliable network, peers may go offline. The Storage Dispatcher detects failed storage operations and triggers new storage jobs. The list of offered functions is *storeItem(object)*, *getItem(key)* and *deleteItem(key)*.

Message Dispatcher: The Message Dispatcher offers direct communication for the higher layers of the platform. Higher layers, especially the multimedia online community related functionality, are encapsulated in so-called plugins. These plugins can either communicate over shared storage objects (indirect communication) or via direct plugin to plugin communication from peer to peer. Using the Message Dispatcher, messages are delivered in a reliable way, once the destination peer/plugin is online. The Message Dispatcher offers the functions *sendMessage(nodeIdentifier, pluginIdentifier, message)* and *receiveMessage(pluginIdentifier)*.

Information Cache: As many plugins operate on stored data objects (e.g. friends lists, photo albums), it is necessary to have a local cache for the objects that has been retrieved once. The Information Cache is a local representation of the DHT and a contact point for higher layers, which may ask the Information Cache for specific objects using *getItem(key)* and *getMessage(pluginIdentifier)*. The Information Cache maintains for each requested data object ID a timestamp and whether it is in the Cache, it has been requested or it is not available in the network. Using these three possible states for a data object, a proper answer can be replied to the querying higher layer, which can then react instantly. By periodically requesting the data object until either the data object is found or marked as not available (based on a timeout), the higher layers can proceed the snapshot of the results any time. An example is depicted in Fig. 1(b), where the higher layer (here Photo Plugin) is prompted to provide the contents of an album. The plugin queries the Information Cache periodically and uses the

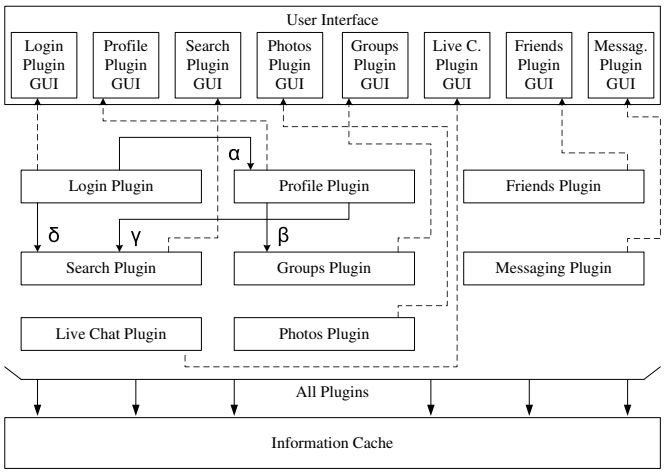


Fig. 2. Plugins Interacting with the UI and the Information Cache

current snapshot of the results. Once the object is requested again and a timeout depending on the timestamp and the current object state (available, pending or missing) is reached, the Information Cache identifies next steps. Upon a timeout of an available object it is requested again in order to find a new version, a pending object is set to missing, and a missing object is looked up again using the Storage Dispatcher. We use the least recently used cache replacement strategy. Please note, that current p2p applications do not operate on distributed data structures and do not use an Information Cache. During our research we identified this component being a highly valuable component for operations on the data, as discussed in Sec. V.

Plugin Layer: The underlying layers provide the functionality of reliable storage, quick data access and plugin to plugin and peer-to-peer communication using the Storage Dispatcher and Message Dispatcher. Based on these functionality we establish a layer for plugins that provide the desired functionality for online community platforms. Plugins are small building blocks providing a specific functionality, having a plugin identifier and an interface for accessing this functions and the results. We differentiate between mandatory plugins that are expected to be implemented in the framework of any participating peer and optional plugins, which cannot be assumed to be on every participating peer. This differentiation helps to make the framework extendible, as new (optional) plugins can be built and installed individually on any peer. Optional plugins can hereby reuse both the functions provided by the lower layers as well the functions provided by the mandatory plugins. We designed plugins to be shareable code packets that can be loaded dynamically into the system using *init()* and *terminate()* functions. For that we built a class loader, that parses at the beginning a specific folder and loads plugin compatible classes.

User Interface: The User Interface is well decoupled from the underlying framework. Plugins offer an interface for accessing their functions and retrieving the results. It is expected that every plugin has its own user interface, the set of all plugin-specific user interfaces are combined in a graphical user interface framework, like it is done in Eclipse [16]. We discuss this design decision in Sec. V. The interactions

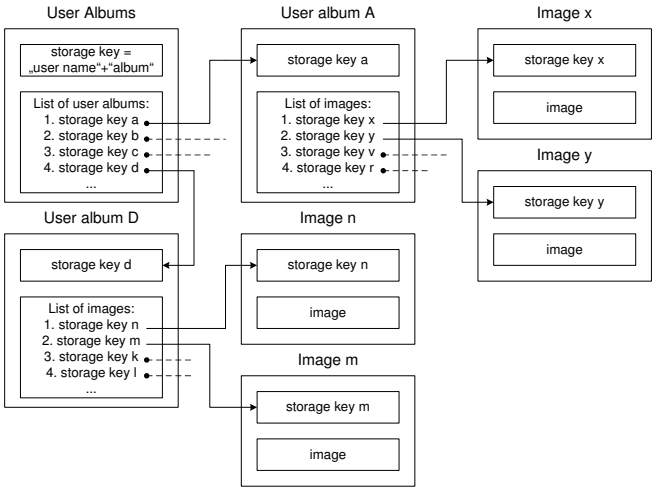


Fig. 3. Example of Distributed Linked Lists used for Albums and Photos

between all plugins and user interfaces is depicted in Fig. 2. The figure shows that all plugins use the underlying layer and for all plugins a plugin specific GUI exists which is combined in a GUI framework. Plugins may use the different functionality (marked with α, β, γ and δ) of each other. The figure also shows the list of plugins we have implemented in the framework. Before presenting the plugins in detail, we describe the distributed data structures used and give a small example, how the layers interact.

B. Distributed Data Structures

Multimedia online communities require to store various kinds of information in the network persistently. Typical examples are user profiles, friends lists, photo albums, video sets with comments and information on interest groups. Casual get and put functionality of DHTs are not sufficient to build such complex applications. For that reasons we created a distributed data structure that operates with distributed linked lists. These are built up of several storable objects, identified by a key, that store besides usage specific content (e.g. meta data to images) pointers on other data objects. With these pointers a graph of interdependent objects is established, a distributed linked list.

In Fig. 3 we present the concept of distributed linked lists on the example of the Photo Plugin. One main idea behind the concept is, that every data object can be uniquely identified by its key. Further the key of the object is used to store and find it in the DHT. Plugins give their data objects specific keys starting with the plugin ID. The list of all albums corresponding to an user Alice for example is built by concatenating and hashing the terms *photoplugin: albumsof* and *user: Alice*, which results in a storage key. The corresponding object contains as payload some meta data and a list of pointers, pointing at data objects that contain the specific albums. One album is a data object containing some meta data and a list of pointers, pointing at image objects. The image objects contain as well some meta data and the picture itself. These data objects are placed using the Storage and Replication layer in the network.

Please note, that the Photo Plugin is just a representative of storable multimedia files. Similarly video files like in YouTube [2] or audio files like in Last.FM [3] can be stored as well. Once the user retrieved the (small) multimedia file, it is presented to him by the GUI. Every album and every image is stored under an individual key in the network on a different peer, thus the load is distributed.

C. Example of Layer Interaction

Plugins use the functions of the framework and implement the specific functionality for multimedia online communities. Fig. 1(b) shows an example of the Photo Plugin interacting with the GUI and the Information Cache. Please note that the photo represents all storable multimedia content. The Photo Plugin offers to get a set of images belonging to a specific album. It can be triggered by its GUI (or other plugins) to retrieve the photos of an album with a given name. As a result it provides a set of images, this set may be incomplete, some images may be marked as pending or missing. It is left to the GUI how to interpret and visualize this information. However, various GUIs may exist for the Photo Plugin, optimized for various devices and purposes.

We assume that the Photo Plugin is triggered periodically and asked for a current snapshot of the results. The main task of the Photo Plugin is to identify which data has to be loaded and to combine data that needs to be presented. First, the Photo Plugin retrieves based on the album name a list of the images corresponding to this album. The list of the images is then stored in the Information Cache. The next time the Photo Plugin is asked it checks whether the list of image pointers corresponding to the album is available in the Cache and if it is so, iterates the list, checks whether the corresponding images are there and requests the pending/missing images. The plugin replies with a list of available images and the state of the missing images. As we assume that the plugin is triggered frequently by its GUI, more and more images arrive and the set of images provided to the GUI grows. The Information Cache is in charge to get the requested objects from the distributed storage and to mark missing objects. Note that other plugins can access the same data from the cache as well.

The plugin itself does not contain any timers or complex mechanisms, it is always triggered (here from the GUI), it checks the availability of the data to combine, combines the data to a reply and passes the reply to the triggering instance. This stateless design eases the development of new plugins and is discussed in Sec. V.

D. Plugins for the Multimedia Online Community Platform

We implemented several plugins named in the requirements analysis using the framework, i.e. for registration and profiles, grouping, searching and communication, were addressed. An overview is presented in Fig. 2.

Login Plugin: The Login Plugin enables an user to join the network by providing information relevant for the authentication. We assume that we have pseudonym identities, i.e. identities that can be checked, but not who the corresponding person is. We use asymmetrical cryptographic public keys as node IDs in the p2p network. Any communication to a

node can be encrypted directly with the node ID. The Login Plugin contains a self-signed document containing an user name (which may give hints on the identity of the user), the node identifier and the IP address of the node. Any peer in the network can look up the login information of any other node using a specific object key e.g. hash(*username:Alice*). Like in ICQ or Skype the user name is a cryptic code, but a suitable buddy name can locally be assigned.

Profile Plugin: The profile of an user contains a personal photo and a list of attributes and values. These attributes may be school affiliation, hobbies, gender, geographical position, birthday ... the list of such information is extendible. In order to provide integrity in data storage, users sign the data objects they store in the network. As their node ID is identical to their public key any other node can check whether a node (e.g. 0x2dc41a57, Alice) is the author of its own profile. The personal settings of an user may be confidential. In order to provide a secure access control without a central entity, we encrypt the signed data object symmetrically and encrypt the symmetrical cryptographic key with the public keys of the peers that are allowed to access the data.

Groups Plugin: Groups (and Friend Lists) are special lists stored in the network containing the user names of group members. Every peer can subscribe in open lists, which they identify by the group name and modify directly using the Storage Dispatcher, which we enhanced to support remote list operations. Restricted groups can only be edited by group owners, access permission has to be requested from a specific peer listed as group owner, who can then decide and modify the group list. The plugin manages to calculate proper group keys, the join and leave procedure and to derive the list of participants of a group.

Search Plugin: In order to search for specific groups or users, we implemented a Search Plugin, which operates on the distributed linked lists and uses the functionality of the Friend and Group Plugin. It is a good example how plugin services can be combined to provide more mature services. The Search Plugin can be fed with attributes e.g. from the profile, it establishes distributed lists and registers the corresponding user at the lists related to his profile. An user e.g. with *Darmstadt* as home town, is then registered automatically in the group/list *Darmstadt* under the hash key of *group_Darmstadt*.

The Search Plugin can also be used to search for users by specifying keywords (e.g. *Darmstadt*). All possible group keys are then built within the plugin and queried. Results to corresponding groups are presented jointly or merged.

Messaging Plugin: The Messaging Plugin offers a functionality similar to email, namely offline messaging. Each user has an in- and outbox, by specifying an user name a message object is created, encrypted with the recipients public key and stored in the inbox list of the recipient under a specific key (e.g. *inbox_username:Alice*. Once this user comes online, he checks his inbox and retrieves unread messages.

Photo Plugin: The Photo Plugin is a representative of storable multimedia content. Giving an user name, a corresponding key is generated, under which the list of albums related to this user can be found, this is shown in Fig. 3. Giving an album name retrieves the list of photos in this album. The resulting set of photos (or other multimedia content) can be used by the GUI for further processing, but can also be instantaneously displayed.

Live Chat Plugin: The Live Chat Plugin represents all real-time dependent multimedia content delivery applications. It uses the Message Dispatcher and the node ID of the receiver to communicate with the Live Chat Plugin located at the other peer. The communication can be encrypted with the public key (which is the node ID) of the communication partner. The messages are sent directly and dispatched immediately to the Live Chat Plugin (by naming the plugin identifier).

E. Example for Plugin Interaction

In Fig. 2 some plugin dependencies are shown. These dependencies are marked with letters α, β, γ and δ. The dependencies denote following relation between plugins.

α - During registering, the Login Plugin instructs the Profile Plugin to create a new user profile. Login Plugin provides the necessary data for creation of a new user profile. β - While creating a new profile or editing an existing profile, Profile Plugin instructs the Groups Plugin to adjust the appropriate groups. γ - The Profile Plugin registers the criteria associated with a certain user by providing appropriate instructions to the Groups Plugin. The information contained in a profile is used by the Search Plugin for searching the users associated with this information. δ - The Login Plugin instructs the Search Plugin to register e.g. the email address associated with a certain user as a searchable value in a distributed list.

V. Design Decisions

For building the framework we made several design decision, which we want to discuss in this section. They are helpful for building extendible p2p applications.

A Common GUI Framework: Instead of combining the graphical user interface with the functional part of the modules, we defined clear interfaces for the plugins. They can be used to state requests to the plugin (e.g. request list of all friends of Alice) and to retrieve the results (get the current list of friends of Alice). Using this paradigm helps to build several kinds of graphical user interfaces and to use the results by other plugins. The results can be either presented to the user using a device specific visualization or reused by other plugins. A GUI framework maintains the plugin GUIs and combines them on the screen (like in Eclipse [16]). We have implemented for all plugins user interfaces and integrated them in an user interface framework.

Base Security Solution: Security is a challenging issue in p2p systems, with several open questions, especially if no central server is used for support. For the scenario of multimedia online communities we sketch a solution that fulfills the requirements. We assume pseudonyms and with that explicitly no binding between human identities and user identities in our system. As node IDs we use asymmetric public keys, e.g. based on elliptic curves as they are small.

Nodes can authenticate themselves based on this public key and receive confidential information encrypted with these public keys. Data stored in the network can be signed by their authors and the signature can be verified with the public key of the author. Integrity and authentication are provided. For access control we propose to encrypt confidential signed data with a fresh symmetrical key and this symmetrical key with the public keys of all authorized nodes. The signed and symmetrically encrypted data and the encrypted symmetric key is then stored and replicated in the DHT. Only a node that can decrypt the encrypted symmetrical key can access the data.

Request-based Event Propagation: In the paper we claimed several times that the GUI periodically requests a current state of the queried results. We call this request-based event propagation. An alternative is the event-triggered event propagation.

To give an example, an user triggers his GUI to show him the images of a photo album of Alice. The GUI triggers the plugin which looks up the corresponding image IDs. As a result, data objects with image specific meta data and an image is expected. After a short time, some of the requested data objects arrive at the Storage Dispatcher, which forward them to the Information Cache. Following the event-triggered event propagation idea, this event should then be forwarded to the Photo Plugin. In that case, the Photo Plugin would receive some of the images and decide, whether the partial images are of relevance for the user or not, whether the GUI is still active or maybe some other request should be prioritized higher. Further the images may be interesting not just for the GUI but for some other plugin as well. Should this other plugin be informed as well? Questions resulting in high complexity.

The request-based event propagation is much simpler, yet more useful in this case. A plugin is just activated by the GUI or other plugins, then it uses the information available in the Information Cache, states maybe some requests for missing objects and combines a proper reply based on the available information. In order to support delay-critical communication the Message Dispatcher offers direct plugin to plugin communication between different peers delivering messages instantly.

Implementation Details: We implemented the proposed framework in Java, which enables us to run the code on a variety of devices. We used FreePastry [14] as DHT and PAST [15] as storage and replication module. However, our system is not limited to these two implementations, as we clearly defined the interfaces and any DHT or storage and replication module implementing this methods can be used. The load is well distributed on the peers as the applied distributed linked lists spread the data and the load among the peers. Testing the system under churn leads to evaluating FreePastry and PAST, which have been evaluated in [17] and [18]. In comparison to client-server approaches the costs are totally shared among the users. This is a great benefit in comparison to the millions for administration costs for Facebook and MySpace.

Structured DHTs are widely used in file sharing scenarios to lookup the peer providing a specific object. Their lookup complexity is $O(log\ n)$ with n being the number of peers in the system, the typical lookup time is faster. However, lookup time for combined data objects (e.g. photo albums) has to be summed but is limited with a timeout (2s), in that rare case the objects are marked as missing. As the costs are shared and the quality of service is comparable to client-server solutions, we believe that the proposed p2p based multimedia online community platform may open the door for a new application of the p2p paradigm in multimedia content delivery.

VI. Conclusion and Future Work

The ongoing trend of Internet communities like Facebook and MySpace did not reflect on p2p based applications up to now. Additionally, multimedia systems converge more and more to the domain of p2p, as content distribution using the p2p paradigm has proved itself very powerful. In this paper, we present a framework that can be used to build large-scale multimedia online communities. The framework encapsulates the functionality in plugins, each providing a specialized function (e.g. maintaining photo albums, friend lists, groups, direct multimedia communication ...). These plugins can use each other and all use the underlying p2p functions provided by an Information Cache, Storage and Messaging modules and a Distributed Hash Table. The framework is easily extendible, the development of new plugins is simple and they can be dynamically loaded into the system. Using the framework enables to build high quality, complex p2p applications by reusing existing mature components.

In the future we plan to extend the framework with more plugins enabling audio streaming for friends, collaborative work and more nice features.

References

[1] DFG Research Group 733, "QuaP2P: Improvement of the Quality of Peer-to-Peer Systems," http://www.quap2p.de.

[2] YouTube - Broadcast Yourself, http://www.youtube.com.

[3] Last.fm - the Social Music Revolution, http://www.last.fm.

[4] Alexa.com, "The Web Information Company," 2008.

[5] Jesse Chan, "Facebook Valuation," http://www.answers.com, Nov. 2007.

[6] B. Stone, "Facebook," New York Times, 2007.

[7] N. Liebau, K. Pussep, K. Graffi, S. Kaune, E. Jahn, A. Beyer, and R. Steinmetz, "The Impact of the P2P Paradigm," in *Proceedings of Americas Conference on Information Systems 2007*, Aug 2007.

[8] Sharman Networks, "KaZaA," http://www.kazaa.com, Mar. 2000.

[9] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann, "BubbleStorm: resilient, probabilistic, and exhaustive Peer-to-Peer search," in *Proc. of ACM SIGCOMM Conf.* ACM Press, 2007, pp. 49–60.

[10] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proc. of IFIP/ACM Middleware '01*. Springer, 2001, pp. 329–350.

[11] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *IPTPS*, 2002.

[12] S. A. Baset and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," in *IEEE INFOCOM*, 2006.

[13] J. Kubiatowicz, D. Bindel, Y. Chen et al., "Oceanstore: An Architecture for Global-scale Persistent Storage," in *Proc. of the 9th Int. Conf. on ASPLOS*. ACM Press,, 2000, pp. 190–201.

[14] Freepastry, http://www.freepastry.org/FreePastry/.

[15] PAST, "A Large-Scale, Peer-to-Peer Archival Storage Facility," http://freepastry.org/PAST/default.htm.

[16] Eclipse, "An Open Development Platform," http://www.eclipse.org/.

[17] Daishi Kato and Toshiyuki Kamiya, "Evaluating DHT Implementations in Complex Environments by Network Emulator," in *IPTPS*, 2007.

[18] A. Rowstron and P. Druschel, "Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility," in *18th ACM SOSP'01*, 2001.