A Large Scale Presence Network for Pervasive Social Computing

Honguk Woo, Hongsoo Kim, Kyusik Kim, Dongkyoung Kim
Software R&D Center
Samsung Electronics
Suwon, Korea
{honguk.woo, herald.kim, kyusik80.kim, dk9586.kim}@samsung.com

***Abstract*— In this paper, we present a scalable presence network architecture for emerging pervasive social applications in which the interaction of not only people but surrounding physical objects seamlessly occurs. The architecture combines the server-based messaging and the peer-to-peer communication schemes, thereby rendering itself deployable at the large scale beyond social networks so as to incorporate a sheer number of embedded devices in a ubiquitous environment.**

***Keywords*-presence; XMPP; scalability; pervasive computing;**

I. INTRODUCTION

With the growing popularity of smart devices, social networks, and cloud computing technology, the demand for seamless inter-connectivity, collaboration, and synchronization across users' devices and surrounding physical objects has been radically increased. Such technology trends facilitating the convergence of the physical world and the social networks can lead to a number of innovative functionalities not yet commercialized for today's pervasive social applications. Recently several projects [1], [2], [3] consider social network platforms for realizing various collaboration scenarios. However, the lack of consideration regarding the networks of large scale causes the difficulty in achieving a generic platform that can satisfy the levels of scalability, extensibility, and performance desired to fulfill requirements on pervasive social applications where a large number of devices and physical objects are incorporated.

In this paper, we address the architectural issues of large scale pervasive social applications by describing a managed overlay network in which the capabilities and resources of devices can be securely accessed and composed. Specifically to mitigate the inherent challenges of scalability due to the sheer number of connected devices, our proposed architecture incorporates sever-based messaging and resource-efficient peer-to-peer networking into an overlay structure. In this approach, a standard messaging protocol, XMPP (eXtensible Messaging and Presence Protocol) [4] is essentially used. Moreover, STUN (Session Traversal Utilities for NAT) and TURN (Traversal Using Relays around NAT) are used for NAT (Network Address Translation) traversal on the present Internet with widely deployed NAT equipments and firewalls. It should be noted that sole reliance on peer-to-peer technologies leads to management problems trading off control for the amount of centralized resources.

The main contributions of our work are as follows: first, we describe the XMPP-based presence network of which motivation is to efficiently manage hundreds of millions of connected devices around the globe for deploying large scale pervasive social applications. The challenge of such scale is well investigated in this paper by the hierarchical construction of regional and inter-regional server clusters, hence ensuring the linear scalability for dealing with devices as the cluster size increases. The clean separation between the control streams with server-based messages and the data streams with peer-to-peer communication refines the complex problem of scalability as the architectural choice on server clustering. Second, we demonstrate the lightweight web service model for programmable device interfaces. This device model is intended to foster the device collaboration and to allow them to be further extended with fine-grained capabilities. It is not trivial to support efficiently the device collaboration particularly when transferring bandwidth-heavy contents is involved. The presence network manages the collaboration of devices in near real-time through server-based messaging, making data transfer happen in a peer-to-peer manner.

The remainder of this paper is organized as follows. Section II includes related work and Section III presents the overall system architecture in which XMPP-based presence servers form a hierarchical cluster so as to efficiently support connections from a large set of devices. Section IV describes our implementation and evaluation. A few pervasive social scenarios spanning user experiences across the network of devices are explained as the usage of the presence network in Section V. Then we conclude this paper.

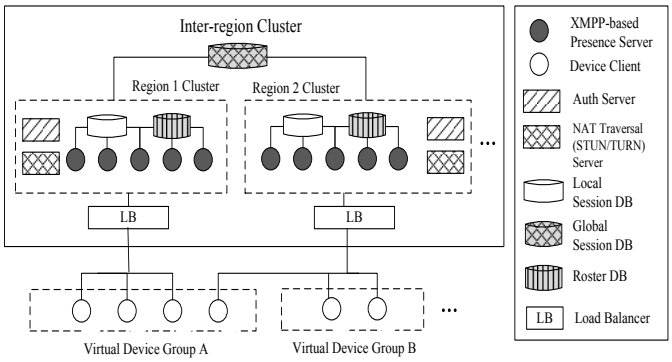


Figure 1. Hierarchical presence network architecture

978-1-4673-5077-8/13/$31.00 ©2013 IEEE

II. Related Work

XMPP is an open standard communications protocol based on XML, by which messages and presence information can be shared among users through centralized servers. It has been broadly extended to multimedia communications including VoIP and signaling for file transfer [5], [6] and cloud services [4]. Wagener *et al*. [7] demonstrated how XMPP can be applied for cloud services, exploring the promising features such that services can be discovered without service registration. As a collaboration scenario, Bendel *et al*. [8] proposed a toolkit *WatchMyPhone* which provides data synchronization and collaboration mechanisms based on XMPP. A pervasive monitoring system relying on XMPP was introduced particularly for post disaster management in [9]. As a similar study, Kirsche *et al*. [10] proposed a solution to unify the world of sensors and actuators with the Internet through the use of the XMPP. While these researches cover necessary functional properties on collaboration services and commonly exploit XMPP as the basis of the system implementation, their concentration is not fully made on performance and scalability issues of collaboration services.

Recently social network platforms for pervasive computing have been an emerging trend. In [1], Mobilis has been explored to provide an XMPP-based social service platform for developers of mobile social applications. It supports temporal as well as spatial restrictions for the visibility of group formation. SAMOA [2] supports the creation of social network applications that can reflect the reality of social interactions. The middleware framework MobiSoC [3] provides the rapid development and deployment of mobile social computing applications. There also has been much work on incorporating the presence functionality into Internet environments. Ford *et al*. [11] proposed UIA (Unmanaged Internet Architecture) that allows nontechnical users to connect their personal devices via a simple and intuitive way using personal device name domains. Similarly, Wu *et al*. [12] proposed PresenceCloud, a scalable server-to-server architecture that can be used as a building block for mobile presence services with DHT. Furthermore, the scalability issue of presence services on peer-to-peer networks has been addressed in [13]. Recently, Eyo extended a personal storage system to support the device transparency to end users with multiple consumer electronics [14]. Such device transparency where user contents are reflected in the namespace is enabled by the replication of metadata across multiple devices and the high availability of metadata. Our approach is different from Eyo in that the functionality is not restricted to a personal storage; any device capability and resource can be abstracted for collaboration on the network through the RESTful service interface.

Especially in the context of home networks, UPnP and DLNA have been prevalently known as an industry standard for device-to-device interoperability and content sharing. HomeOS [15] and dynamically scaling applications are examples of extending traditional UPnP and DLNA based home networks to a variety of areas. As Internet connectivity of devices becomes common, recently many efforts to cover beyond local networks in terms of the scope of device connectivity have been made including SIP-based remote sharing [16] and P2P-based DLNA extension [17]. Our goal of providing seamless cross device experiences to users is shared by such UPnP and DLNA extensions. However, they are generally specific to the semantics of interoperability and sharing between devices, and most importantly are targeted at lower scale.

III. System Design

Fig. 1 briefly illustrates the presence network architecture which consists of two different types of clusters: regional clusters in which XMPP-based servers are tightly connected on the high bandwidth networks and inter-regional clusters of which the connectivity is restricted by common WAN. Each server is capable of the device management including discovery (creating a virtual device group), presence (notifying the device status information updates in a group), and signaling (messaging for controlling functions in a group). The signaling is used to initiate the peer-to-peer communication, combined with NAT traversal. The local session database in a regional cluster supports the near real-time references to the pairs of a device client and its connected server process, but the global session database does the pairs of a device client and its regional cluster. This hierarchical clustering allows widespread devices to be grouped together regardless of their geographical location, e.g., the virtual group B in Fig. 1. In our empirical set-up on a public cloud, workloads are fairly distributed in that each device client is directed to make a session to a server according to the policy of the load balancer in front of its regional cluster; and before that, the regional choice is dynamically made in either a location basis or a user basis. To satisfy the system requirements on time-varying load characteristics, the clusters have been operated with a monitoring and auto-scaling tool in our real deployment.

A device client can update its presence status and communicate with servers and other clients through its established session. When the device client initially logs into the network or changes its status, e.g., on/offline availability, the status information is firstly sent to the presence server via the session, and then forwarded to other online device clients of which location information can be found by using the lookup interface to the session database. More specifically, the status is sent to the corresponding proxy process in the regional cluster for each respective device client, and then actually forwarded to the recipient. If the recipient is not in the same regional cluster, the propagation additionally requires inter-regional routing that inquires of the global session database to locate the gateway of the target regional cluster and then forwards packets (the presence update packet in this case) to the gateway. Once the target gateway receives the packet, the remaining

procedure basically follows the same pattern as that in a regional cluster. Notice that the procedure of the presence propagation above is commonly applied to other XMPP primitives between device clients, e.g., instant messaging, signaling, etc. Whereas the presence associated with the session database makes message delivery work properly, it is the concept of inter-region routing (with the global session database and the gateways) that allows presence servers to be hierarchically clustered, which can scale efficiently. The scalability evaluations of our hierarchical architecture will be presented in Section IV.

On the device client side, our current implementation includes two sets of API libraries. One library prototype supports the service-style interface; the capabilities and resources of individual devices and their compositional functions are represented in the lightweight RESTful service APIs [18]. The goal of this library is to simplify the development of web applications that realizes various cross-device scenarios, e.g., remotely accessing contents on smart devices, synchronizing data in a group, manipulating the properties of multiple devices, and many other cases. In a device group, the RESTful service is executed mostly through peer-to-peer communication although there is no restriction to utilize server-based messages like Jabber-RPC [19] especially for reliability and timeliness. By using the RESTful service APIs, complex functions can be executed in a consistent way that the compositions of device functionalities can be made on the fly. The other library includes the socket-style APIs that support the connectivity among devices behind NAT equipments. The library employs NAT traversal techniques and is implemented based on an open source XMPP-Jingle client [20]. Note that the RESTful APIs of a device client can be executed on top of this socket-style library for direct device communication.

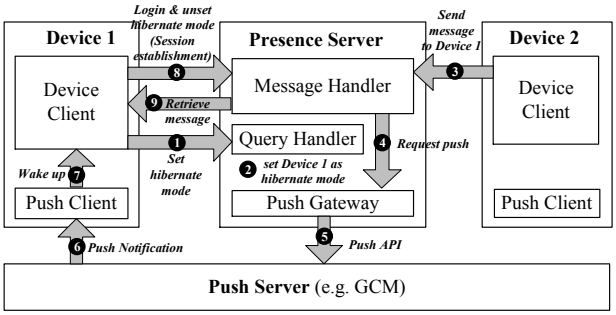


Figure 2. Integration with push notification for mobile devices

IV. EVALUATION

A. System Implementation

Our implementation is based on an open source XMPP server running on erlang virtual machine [21]. The hierarchical clustering is implemented by extending the XMPP server-to-server communication module for inter-regional routing and by integrating in-memory databases into the network for caching session information. The session information is maintained in an open source distributed database [22] in server clusters and cached for the performance improvement.

Each session for a device client requires maintaining a persistent TCP connection to the sever cluster. To cope with constrained resource availability at mobiles, we leverage push notification e.g., Android's GCM, as a triggering mechanism for timely establishing a session only when requested. When a device client turns into the hibernate mode, it can terminate its session. In general, the push client module of mobiles is configured to be always alive as depicted in Fig. 2, and therefore, upon a message targeting to the hibernated device client, the presence server rather can send the push notification so as to make the device client back to online (3~8 in the figure).

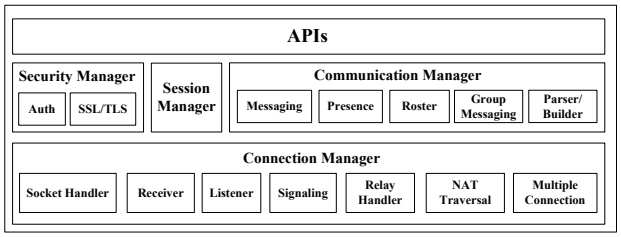


Figure 3. Device client architecture

The device client comprises several components as shown in Fig. 3. The security manager supports authentication of users and devices through the auth server and the presence server. The session manager and the communication manager provide the main functionalities of a device client including session establishment, messaging, and presence update which are mostly XMPP compatible interactivities. The connection manager handles persistent connections to a server and peer-to-peer communication by using NAT traversal and multiplexing. Our client libraries include the socket-style APIs that provide NAT traversal for peer-to-peer connectivity in the form of secure channels between device clients. The socket-style API library is named "plink" and it has similar functionalities as BSD socket APIs, except NAT traversal. In addition, our implementation includes a few messaging APIs and management APIs at client-side. For example, the below messaging and callback API signatures hide the complexity of XMPP messaging, providing a simple semantic of sending and receiving data.

```
// Messaging API
RESULT  msg_send(handle_t handle, std::string &jid, std::string &msg)
// Callback API
RARESULT  callback_register(handle_t rahandle, callback_pfn_t callback)
```

In a constrained environment, it is effective to adaptively control physical connections for sending multiple data streams between device clients. When UDP hole punching is needed as part of NAT Traversal procedures, the client pair might want to maintain a single connection but send multiple streams on it. This allows clients to reduce the set-up time of the communication as well as efficiently manage the resource usage. To do so, our device client is implemented to support thread-based multiplexing and

demultiplexing of data streams. Furthermore, in order to efficiently support multiple device clients running on a single physical device (in case of multiple pervasive social applications on a physical device), we refine the structure of device connections to the XMPP-based presence server; the device-to-server proxy in Fig. 4 is implemented as a middle proxy process at the client-side for handling messages of multiple device clients through a single TCP connection to a presence server. To do so, each application, e.g., App 1, 2, and 3 in the figure, has its own thread and message queue for sending message packets to the server. Keeping an individual message queue per application is aimed at avoiding unfair latency in messaging, considering a common situation where the bandwidth limitation is a bottleneck for real-time messaging performance. Otherwise, one application could fail in sending messages due to the timeout incurred by heavy messages from the others.

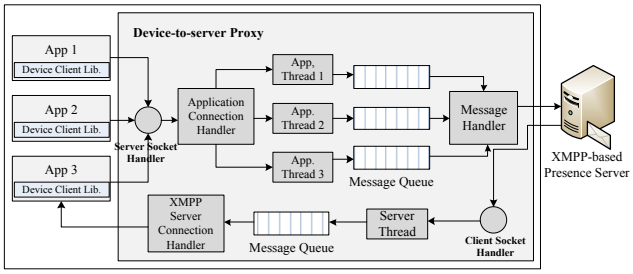


Figure 4. Device-to-server proxy

B. Performance Evaluation

The performance and scalability of the presence network has been measured by deploying our solution on a public cloud with globally distributed multiple data centers, against the workloads that have been designed based on real application characteristics.

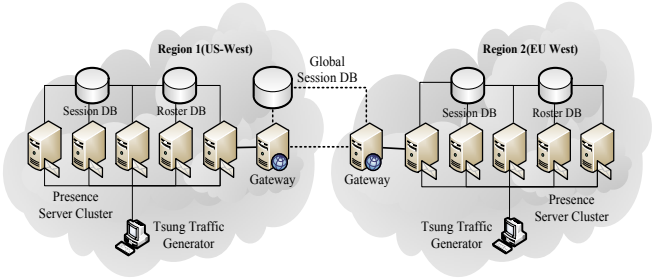


Figure 5. System configuration in clouds

Fig. 5 illustrates the deployment and configuration of our benchmark test system in a multi-region cloud environment. The test system in a region, e.g., US-West region in the left side of the above figure, consists of two database servers (local session and roster), five XMPP-based presence servers, and at least two servers for emulating a group of device clients. Each of these servers runs on the virtual machine instances provided and managed by a public cloud infrastructure and Tsung [23] is configured to generate the workloads from the large sets of device clients. Notice that our hierarchical clustering architecture extends the above configuration in a single region so as to encompass multiple regions in the network by deploying a set of gateway servers along with the global session database. The workload by an emulated client in Tsung is designed to follow the client behavior model below, which is derived from the empirical trace data.

- Update session: read user credentials in the authentication server, update the session entries into the local session database, check the existence of the routing entry in the global session database, and add the routing entry into the global session database if no entry exists.
- Initial Presence: retrieve the roster information of which average size is 4, and send the initial presence to all the entities in the roster.
- Presence update: send the latest presence to all the online entities in the roster every 30 min on average.
- Messaging: send a message (200~1024bytes) to another client every 10 min on average.
- Reconnection: reconnect every 1 hour on average.

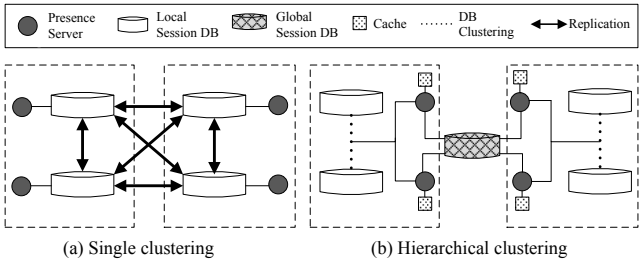


Figure 6. Clustering architecture

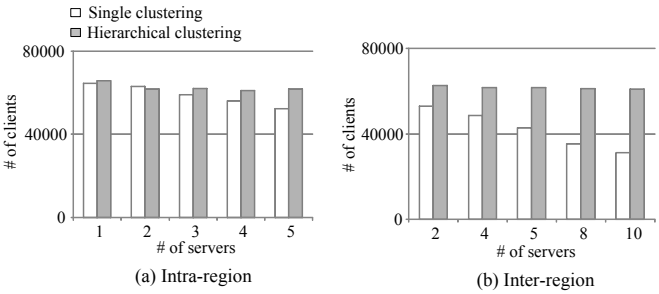


Figure 7. Scalability of clustering architecture

In these tests, the hierarchical clustering has been compared with a common architecture that has a set of identical servers in a single cluster, hereafter simply called the single clustering. In order for any XMPP-based presence server to be able to appropriately locate the target server for forwarding a message, each server in the single clustering is deployed with its own replica of the session database as shown in Fig. 6 (a). It should be noted that servers in the single clustering can be geographically distributed, e.g., multiple data centers in different continents. Fig. 7 depicts the comparison test results where X-axis represents the number of servers, and Y-axis represents the maximum number of online clients per server. As shown, the hierarchical clustering yields relatively undeviating results both for the intra-region case and the inter-region case,

leading to the conclusion that it scales linearly along with the cluster size. On the contrary, the single clustering reveals relatively adverse results; as the number of servers in the cluster increases, the overhead from database replication of the single clustering could make a detrimental impact on the number of concurrent connections possibly managed by a single server. Such impact is highly aggravated for the inter-region case where clustering and database replication can incur more overhead due to the limited network bandwidth. For production use where thousands of millions of devices could be connected, it is important to have such a system property of linear scalability against device clients.

Due to the common fact that a complex hierarchical structure may increase the number of queries over topology information for locating an entity in the structure, the potential overhead from such queries needs to be studied and mitigated. The single clustering in which each server owns a full replica of all necessary session information is an ideal structure, particularly when the number of required queries on session databases for forwarding messages is mainly considered for overhead estimation. In this regarding, we first compare the message latency of the hierarchical clustering with that of such an ideal case, the fully replicated single clustering. Then we conduct tests with a cache scheme for reducing the message latency. Specifically in the hierarchical clustering as conceptually illustrated in Fig. 6 (b), the local session information is partially cached in the server memory. Fig. 8 presents the inherent message delay of the hierarchical clustering (without cache), and the improvement by exploiting the session cache in the same clustering architecture. In the figure, X-axis represents the number of messages per second and Y-axis represents the message latency in milliseconds. Since a group of device clients are emulated by Tsung running on public cloud virtual machines, the message latency does not include delay in wide-area networks. It implies that in practice, end-to-end message delays must vary significantly by additional network delays from the last mile between devices and servers. Overall, the hierarchical clustering with cache yields the stability of message latencies against various message rates, and the improvement by cache gets much significant with the higher message rates as expected.

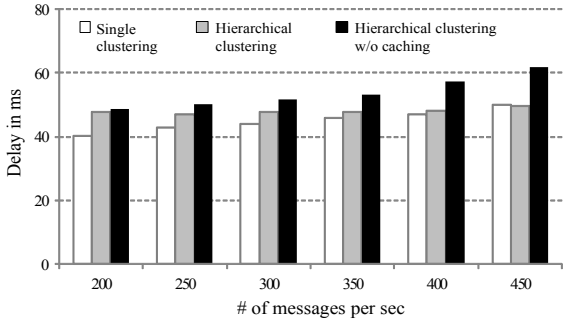


Figure 8. Message latency of clustering architecture

The auto-scaling mechanism is deployed to provide usage monitoring and automatic scale-out of a server instance when a certain threshold is reached. In practice, the threshold is defined as a runtime condition imposed on resource utilization. Fig. 9 shows that one server instance is added around time 21 when the resource utilization reaches the threshold, e.g., 40% in memory usage, and then the new workload (connection requests from device clients) is successfully migrated to the new server from time 24. The migration of existent connections can be expensive due to required updates on session databases and caches. So we rather consider new connections and fail-and-retry connections as candidates for migration. Combined with the linear scalability demonstrated previously, the auto-scaling function provides the practical foundation for operating the production system at large scale.

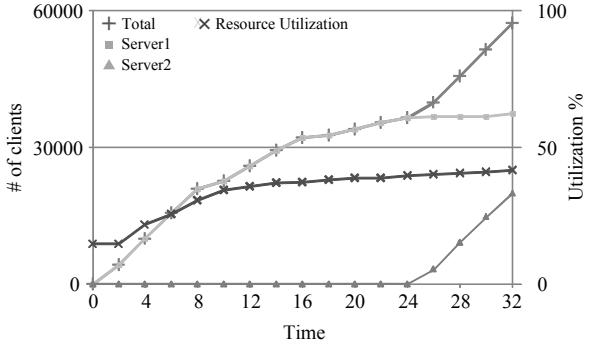


Figure 9. Auto scaling of presence network

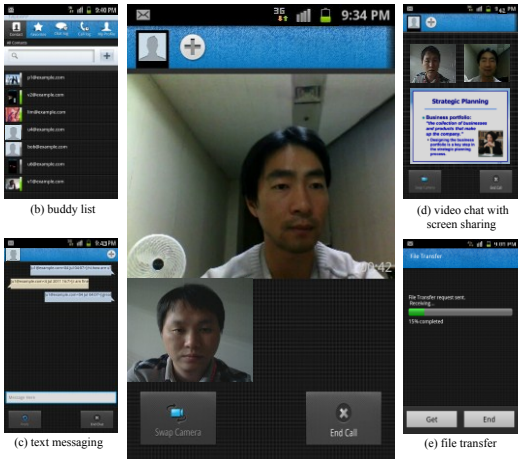


Figure 10. Real-time communication and collaboration

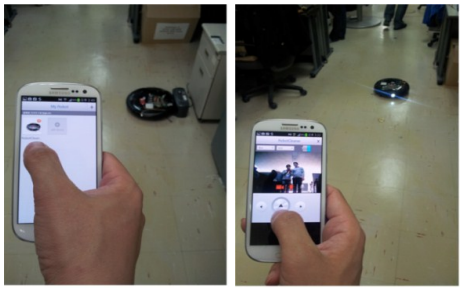
V. USE CASES

Although it is still in the evolving phase, our XMPP-based presence network has been implemented and deployed as a common networking platform for operating a few pervasive social services. Some of the following service scenarios have been recently implemented and tested for the feasibility of the presence network.

- ***Pervasive Collaboration***: With the presence network, it is possible to build a real-time communication and collaboration environment in which a user can access her

own or her friend's devices and play contents at any time regardless of the device locations. Devices can form a group by exploiting owners' social relations. In this group, the device capabilities and resources can be dynamically composed for realizing a variety of device collaboration scenarios. Remotely playing favorite video contents or synchronizing documents between heterogeneous devices at different network settings is a basic one, which can be extended with functional diversity, e.g., GPS, gyro sensors, etc. Moreover, collaborative executions can be seamlessly migrated to another composition of device capabilities. Fig. 10 shows the captured images of such a real-time collaboration application, of which clients have been implemented on Android smartphones, demonstrating various features such as instant messaging, video chatting, file sharing, screen sharing and so on. In our implementation, XMPP of the presence network is used for messaging and signaling in group communication and NAT traversal techniques are used for setting up peer-to-peer collaborations among devices.

- ***Smart Object Interaction***: A user can control and monitor smart objects such as intelligent appliances, e.g., PC, connected TV, robot-style vacuum cleaner, from outside through her mobiles. In Fig. 11, the presence network is used as the core component for seamlessly discovering and controlling devices i.e., robot vacuum cleaners and smartphones, by exploiting the roster information of XMPP between a user and her controllable devices. Note that video streams and control command data streams are separately maintained by the XMPP session and the peer-to-peer channel, consistent with our principal architectural feature of the presence network. From our experiences in deploying and operating such controlling services for smart devices in our production environment, we have learned that such a clear separation between control and data streams alleviates the system complexity, thereby improving the service scalability.



(a) device discovery (b) controlling and monitoring

Figure 11. Remote control and collaboration of smart appliances

VI. CONCLUSION

Despite all the opportunities in enhancing ambient user experiences by the proliferation of smart devices and cloud computing technologies, the lack of generic solutions for accommodating the sheer number of device connections often calls for cumbersome efforts for development and deployment of pervasive social applications. The proposed presence network architecture incorporates server-based discovery and messaging as well as peer-to-peer overlay in the network of devices, hence enabling existing social network scenarios to be applicable for ubiquitous networks where many heterogeneous devices are connected.

REFERENCES

[1] R. Lubke, D. Schuster, and A. Schill, "Mobilisgroups: Location-based group formation in mobile social networks," in *Proc. IEEE PERCOM Workshops*, 2011.

[2] D. Bottazzi, R. Montanari, and A. Toninelli, "Context-Aware Middleware for Anytime, Anywhere Social Networks," in *IEEE Intelligent Systems*, Vol. 22, 2007.

[3] A. Gupta, A. Kalra, D. Boston, and C. Borcea, "MobiSoC: A Middleware for Mobile Social Computing Applications," *Mobile Networks and Applications*, Vol. 14, 2009.

[4] A. Hornsby and R. Walsh, "From Instant Messaging to Cloud Computing, an XMPP review," in *Proc. IEEE ISCE*, Braunschweig, 2010.

[5] P. Saint-Andre, "Streaming XML with Jabber/XMPP," *IEEE Internet Comput.*, vol. 9, no. 5, 2005.

[6] P. Saint-Andre, "XMPP: lessons learned from ten years of XML messaging," *IEEE Commun. Mag*., vol. 47, no. 4, 2009.

[7] J. Wagener, O. Spjuth, E. L. Willighagen, and J. ES. Wikberg, "XMPP for cloud computing in bioinformatics supporting discovery and invocation of asynchronous web services," *BMC Bioinformatics*, vol. 10, no. 1, 2009.

[8] S. Bendel and D. Schuster, "WatchMyPhone - Providing developer support for shared user interface objects in collaborative mobile applications," in *Proc. IEEE PERCOM Workshops*, 2012.

[9] R. Klauck and M. Kirsche, "XMPP to the rescue: Enhancing post disaster management and joint task force work," *in Proc. IEEE PERCOM Workshops*, 2012.

[10] M. Kirsche and R. Klauck, "Unify to bridge gaps: Bringing XMPP into the Internet of things," in *Proc. IEEE PERCOM Workshops*, 2012.

[11] B. Ford *et al.*, "Persistent personal names for globally connected mobile devices," in *Proc. USENIX OSDI*, 2006.

[12] C. Wu, J. Ho, and M. Chen, "A Scalable server architecture for mobile presence service in social network applications," *IEEE Trans. Mobile Comput*., vol. 12, no. 2, 2013.

[13] C. Raniery Paula dos Santos *et al.,* "On the impact of using presence services in P2P-based network management systems," in *Proc. IEEE GLOBECOM*, 2010.

[14] J. Strauss *et al.*, "Eyo: Device-transparent personal storage," presented at *USENIX ATC*, 2011.

[15] C. Dixon *et al.*, "An operating system for the home," presented at *USENIX NSDI*, 2012.

[16] Y. Oh *et al.*, "Design of an extended architecture for sharing DLNA compliant home media from outside the home," *IEEE Trans. Consum. Electron.*, vol. 53, no. 2, 2007.

[17] C. Rus, K. Kontola, I. D.D. Curcio, and I. Defee, "Mobile TV content to home WLAN," *IEEE Trans. Consum. Electron.*, vol. 54, no. 3, 2008.

[18] C. Riva and M. Laitkorpi, "Designing web-based mobile services with REST," *LNCS*, vol. 4907, 2009.

[19] Jabber-RPC, http://xmpp.org/ex-tensions/xep-0009.html, 2011.

[20] Jingle, http://xmpp.org/extensions/xep-0166.html, 2009.

[21] Ericsson AB, http://www.erla-ng.org, 1997.

[22] Cassandra, http://cassandra.apache.org, 2009.

[23] Tsung, http://tsung.erlang-projects.org, 2005.