Resource Discovery Protocols for Bluetooth-based Ad-hoc Smart Spaces: Architectural Considerations and Protocol Evaluation

Andreas Brodt, Alexander Wobser, Bernhard Mitschang
IPVS, Universität Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
e-mail: <brodt, mitschang>@ipvs.uni-stuttgart.de, alexander.wobser@alewo-online.de

***Abstract*—Ad-hoc smart spaces aim at resource-rich mobile devices sharing resource and context data with others near them spontaneously. Thus, a device may, e.g., obtain a more complete context model by utilizing sensor data of its neighbors via wireless communication, such as Bluetooth. The highly dynamic device neighborhood challenges resource discovery, as the devices have to organize themselves autonomously.**

This paper evaluates different resource discovery protocols in Bluetooth-based ad-hoc smart spaces. We simulate the protocols in different scenarios taking into account the scatternet structure of the network. We suggest request flooding for small settings and random replication for medium to large spaces.

I. Introduction

Modern mobile devices (smartphones, PDAs, etc.) carry a multitude of sensors, e.g., GPS, cameras, or accelerometers. They are also capable of storing larger amounts of data, both personal (calendar, contacts, messages) and general (maps, traffic, weather). Put together, sensors and stored data tell a lot about the user's current *context* [1]. This allows context-aware applications that provide more user benefit [2]. However, not all aspects of context are available when needed as devices have different capabilities and sensors are not always functional (e.g. GPS indoors). Humans frequently ask other people for help, e.g. to find a train station. Mobile devices possess several means to communicate; they *could* adopt this and ask their neighbors for missing context data.

Ad-hoc smart spaces [3] aim at resource-rich mobile devices sharing *resources* (sensors and context data) with others near them spontaneously and without dedicated infrastructure. They act in the background to improve applications that require resource data. I.e., they must not hinder other activities on the device, e.g., by allocating device features exclusively. To put ad-hoc smart spaces into effect, we focus on practical solutions that cater for easy adoption. I.e., implementations should run on consumer devices without additional hardware or drastic changes of the software stack.

A key challenge of ad-hoc smart spaces is *resource discovery*. Unlike device discovery, which is closely coupled with the underlying communication technology, finding out which devices might share which resources opens a large space. A plethora of discovery mechanisms exist in the literature [4], ranging from home entertainment to internet-scale peer-to-peer nets. This paper evaluates how selected resource discovery protocols perform in ad-hoc smart space settings characterized by (1) autonomous peer-to-peer organization, (2) highly dynamic behavior, and (3) Bluetooth communication: (1) Ad-hoc smart spaces depend solely on the mobile devices and assume no further infrastructure. (2) A space is formed when devices discover each other. Devices leave by moving away. Spaces grow and shrink, split up and merge, so resources appear and disappear very dynamically. (3) We focus on Bluetooth, as it best meets the requirements: Most mobile devices support it and it does not interfere with ulterior communication. Contrarily, a WLAN-solution needs to put the WLAN interface into ad-hoc mode cutting off infrastructure-based communication. Finally, Bluetooth is comparatively energy-efficient, suiting it for long-running background activities.

Our ultimate goal is a practical implementation on a mobile device, such as a Nokia Internet Tablet. This further restricts the design space to solutions that are implementable on top of a state-of-the-art mobile software platform, in our case Linux and the BlueZ Bluetooth stack.

As our contribution, we introduce the characteristics of Bluetooth as well as our system architecture, being the foundations of this work (Section II). We examine different resource discovery protocols (Section III) which we simulated in typical scenarios for ad-hoc smart spaces (Section IV). The simulations show that the chaotic nature of ad-hoc smart spaces is difficult to capture. Yet, we observe that request flooding works well for small networks (≤ 20 devices). For larger networks, more complex discovery structures pay off.

II. Foundations

The foundations of this paper are the Bluetooth networking and the system architecture of an ad-hoc smart space middleware. Both strongly influence our evaluation.

Bluetooth communicates strictly in a point-to-point fashion. Devices organize themselves in *piconets* of one master and multiple slaves. The master coordinates communication in the piconet and routes messages between slaves at the physical link layer. The slaves only communicate with each other on the logical link layer. A device can participate in several piconets at the same time. If the network includes such devices, as shown in Figure 1 (marked with an asterisk), the network is called a *scatternet*. The Bluetooth specification [5]

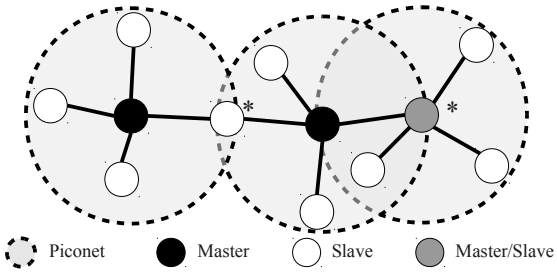

Figure 1. A Bluetooth scatternet consisting of three piconets

does not include routing in a scatternet, and current Bluetooth stacks do not provide it. Applications need to provide their own scatternet routing on top of the Bluetooth stack.

Bluetooth includes the Service Discovery Protocol (SDP), a mechanism to find *services* in the local piconet [5]. However, resources further away in a scatternet are out of reach. Also, SDP is restricted to searching for unique service IDs. Search by arbitrary attributes (e.g., "GPS devices receiving three-dimensional coordinates") is not supported.

We envision the ad-hoc smart space middleware as one Bluetooth service that builds a peer-to-peer overlay on top of the scatternet. We use Bluetooth device discovery and SDP to find running middleware instances first. Then the middleware instances organize each other to share their resources in a second step. Figure 2 shows the resulting architecture.

Put simply, the Bluetooth stack of the operating system comprises three layers. The physical link layer manages master-slave communication. The logical link layer caters for datagram messaging in a piconet and provides the base for the Service Discovery Protocol (SDP) and the RFCOMM profile. RFCOMM provides reliable end-to-end communication.

The ad-hoc smart space middleware uses RFCOMM and needs to provide scatternet routing. Resource discovery builds on top of the routing layer to distribute resource information throughout the scatternet. Resource management uses both resource discovery and routing. An application is presented a unified set of resources; local and remote.

III. Resource Discovery Protocols

Figure 2 depicts resource discovery as a core component in our system architecture that acts on top of the entire scatternet. Our scenario is characterized by a relatively small scale (at most a few hundred devices), decentralized autonomous organization, and very dynamic network topology. This rules out *centralized* client/server approaches, e.g. LDAP [6]. Decentralized *structured* approaches, e.g. Chord [7], distribute resource information using sophisticated structures. The effort to maintain these structures in the highly dynamic network is likely prohibitive, so we did not consider these approaches.

Many peer-to-peer networks build clusters as an overlay network for efficiency [4]. A Bluetooth scatternet provides a cluster structure at the physical link layer, so it seems ideal

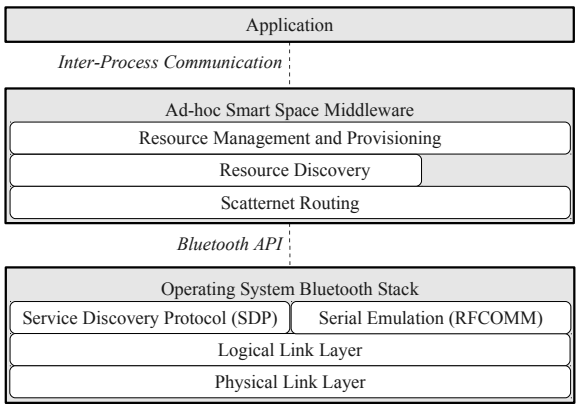

Figure 2. System architecture of an ad-hoc smart space implementation

to exploit it. [8] presents an algorithm that routes Bluetooth SDP queries through scatternets. Devices participating in several piconets offer an SDP service named "bridge". A device looking for resources may use a bridge service to forward SDP searches to other piconets, if the service was not found in the local piconet. Besides adopting the limitations of Bluetooth SDP (search by service IDs only), there is one major problem, however. A device must determine its role in the scatternet to know whether to provide the "bridge" service. This knowledge is available in the physical link layer (see Figure 2), but the Bluetooth API, as provided on a mobile device, does not propagate the knowledge to the logical link layer. In practice, scatternet routing and resource discovery have no chance to determine a bridge device. Of course it is possible to modify the Bluetooth stack of an open operating system. Yet, as stated above, this would render easy adoption of ad-hoc smart spaces impossible on a significant number of devices.

Our scenario leaves three main design variables open: replication, routing, and information propagation. (1) The *degree of replication* varies between zero and full replication. The extremes are pull-based querying of all devices or push-based replication on all devices and search in local directories, respectively. (2) Protocols may integrate *scatternet routing* with discovery, or keep the two tasks separate. Integrating them may optimize communication based on the network structure. Separating them allows exchangeable layer implementations and even other communication technologies. (3) Protocols may *propagate information* in different ways: flooding, tree structures, or point-to-point links. We evaluated resource discovery protocols covering all three design variables.

A. Request Flooding

Request Flooding uses zero replication, integrates scatternet routing, and propagates information through flooding. A searching device emits a request to all neighbors, which either reply with resource information or forward the request recursively. Unique request IDs prevent multiple processing

of requests. Request Flooding does not maintain any support structures or directories and needs no initialization. Yet it is guaranteed to find any existing resource. It also finds the shortest route to the resource.

Due to its simplicity, Request Flooding is considered trivial. At the same time it serves as a benchmark; other protocols should perform at least as good as Request Flooding.

B. Resource Flooding

Resource Flooding is a contrary approach to Request Flooding. It uses full replication, integrates routing and propagates through flooding. Devices publish their resource information to all neighbors, which forward it recursively. This happens whenever resources become available. Devices keep a local directory that is updated by the resource advertisements. Resource requests are answered locally without any communication.

C. Publish/Subscribe

Publish/Subscribe [9] uses full replication, includes routing, and propagates resource advertisements via a tree to avoid the duplicate messages, which Resource Flooding produces.

When two devices encounter each other, they exchange their lists of known devices. For every unknown device in the received list, they issue a subscription to receive all resource advertisements about the (previously) unknown device. Thus, devices need to keep a subscription table to memorize whom to forward incoming resource advertisements. The subscriptions form a *tree* for every resource provider, which must be repaired whenever a device leaves. When the loss is noticed, it is published along the tree, and devices remove the respective resources from their local directories. The devices exchange their lists of known devices regularly so that broken paths in the tree are ultimately replaced.

D. Gnutella-Inspired

Gnutella [10] was the first major peer-to-peer file sharing system that worked completely decentralized. Every participant selects a number of *neighbors* anywhere in the network that replied to a flooded ping message. File Search is done by flooding a request over these neighbors (using a max. hop counter). This scaled poorly for very large networks and lead to enormous delays. Several improvements were proposed including clustering [11] using "ultra-peers" that cache resources of their neighbors.

As ad-hoc smart spaces scenarios are much smaller than internet-scale file sharing nets, the disadvantages of Gnutella might not make much impact. So we adopted concepts from Gnutella in our "Gnutella-inspired" protocol:

A device finds other devices through Bluetooth device discovery and SDP, declares them its neighbors and connects to them. It floods a ping message over all neighbors (using a max. hop counter). Devices reply with the list of resources they offer. The initiating device stores these lists in a local directory together with the route via which the message traveled. Devices routing the messages to their destination update their directories, too. Discovery requests are answered from the local directory. If this does not succeed, the request is flooded over the neighbors. Thus, the protocol uses partial replication and propagates information through flooding. We designed two variants of the protocol w.r.t. routing:

1) Integrated Routing (IR): This variant directly calls the Bluetooth API and utilizes direct links in the scatternet. Thus, "neighbors" are reachable via one hop and flooded discovery requests travel along the physical scatternet structure.

2) Separate Routing (SR): To evaluate the effect of a separate scatternet routing layer, we modified the protocol accordingly. The particular routing algorithm used in our simulations (see Section IV-B) keeps the network slim, resulting in a different network topology. The discovery protocol is the same, devices only connect to fewer neighbors and the routing algorithm creates additional messages.

E. Central Directory

The Central Directory protocol creates a single replication of all resource information, depends on a separate routing layer and uses point-to-point communication. It implements *publish-find-bind* as known, e.g., from UDDI [12]. It works very well in static infrastructures, but requires tweaks in ad-hoc smart spaces. Devices vote one device to host a central directory and to answer all discovery requests. Although the term "vote" implies democracy, the first device needing a directory appoints itself directory host and informs the others via flooding, which reply with their resource lists. When the directory host becomes unavailable, the first device to notice the loss initiates a new voting procedure.

In case of a net merge two directory hosts exist. A net merge always creates a bottleneck at the single bridge device that routes all messages between the two nets. The bridge is not interested in the traffic of discovery requests, so it tells the two directory hosts to exchange their directories. This way, discovery stays in the two old networks, and no directory announcements need to be flooded.

F. Random Replication

Random Replication features a configurable degree of replication, depends on a separate routing layer, and uses point-to-point communication. It creates many *decentralized* directories randomly. Random Replication builds an overlay structure independent of the physical network and assumes a global view on the scatternet. Every device randomly chooses a number of "neighbors" from the entire ad-hoc smart space. Every device queries its neighbors for their resource lists and stores them in a local directory. Search is first done locally, then the neighbors are queried. Thus, the requests may travel quite randomly through the physical network. The parameter r denotes the neighbors per device as a fraction of all devices. Thus, r controls the global degree of replication.

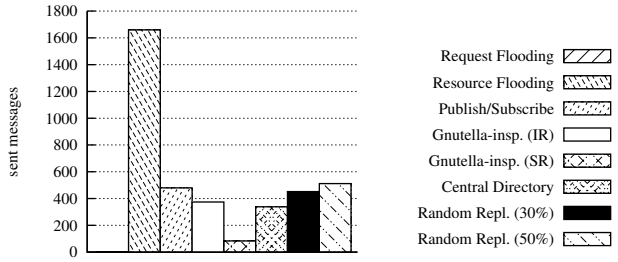

Figure 3. Message load measured in the *Initialization* scenario

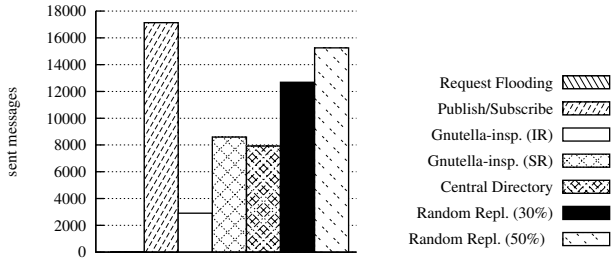

Figure 4. Message load measured in the *Grow* scenario

The number of logical hops of a request in the overlay network largely depends on r. We simulated $r = 30\%$ and $r = 50\%$. I.e., statistically every third or second request can be answered from the local directory, respectively.

IV. Evaluation

We compared the resource discovery protocols in over 2 000 simulations using a custom-made open source simulation environment[1]. We simulated three network topologies (Line, Circle, and Random), and three network sizes: Small, Medium, and Large (5, 15, and 50 devices).

A. Metrics

We exclusively measure the number of messages a device sends and receives on the logical link layer in a piconet. We do not simulate the low level details of Bluetooth; message fragmentation, frequency hopping, or TDMA slots are omitted. This rules out measuring precise latencies and exact energy consumption. However, all simulated protocols act on top of the RFCOMM Bluetooth profile and send messages in the exact same way; a more precise simulation would not reveal additional insights. The piconet messages allow estimating latency and energy consumption; they will be roughly a function of the message load.

B. Scatternet Routing

The scatternet routing layer of our simulation environment uses distance vector routing. Every device keeps a routing table which provides a global view on all devices, but not the optimal route. The routing algorithm minimizes open connections per device to achieve a sparse network topology. This increases network throughput and decreases message loss, as devices require fewer context switches [13], [14].

A device periodically searches for other devices. In the simulation, devices are found based on their geographic distance. Whenever device *A* finds a new device *B*, *A* checks whether *B* is already listed in the routing table. If so, *B* is already reachable, i.e., a direct connection to *B* would only make the network denser. Otherwise *A* connects to *B* and the two devices exchange their routing tables. For every device *D* in *B*'s routing table that is unknown to *A*, *A* adds a route via *B*.

If *A* notices that *B* is no longer reachable, it simply removes all routes via *B* from its routing table. Yet, *A* does not propagate the loss of *B* to other devices, as it is not sure whether this information is ever needed. Only if a device attempts to send a message via *B*, the loss of *B* is signaled along the routing path and the devices on the path remove *B* from their routing tables.

C. Initialization Scenario

For small networks one would assume that the message load to maintain organizational structures does not pay off. Thus, it is interesting to measure this load as a first performance indicator. We created a test scenario using a *Small–Circle* and a *Small–Line* topology, and no discovery requests. We ran the simulation for the first 20 messaging cycles. Thus, all messages serve the purpose to find neighbors, establish routing, create local directories, etc.

Our results, as depicted in Figure 3, show that there are two extremes: First, Resource Flooding creates an excessive amount of messages. This is not a specific outlier of the particular scenario; we obtained similar results from other simulations (omitted for brevity). Second, Request Flooding creates no messages at all, which obvious as Request Flooding only communicates to search for resources.

Furthermore, Figure 3 shows that the Gnutella-inspired protocol with integrated routing *(IR)* generates considerably fewer messages than with separate routing *(SR)*. Initialization of the routing layer clearly leaves its mark.

D. Grow Scenario

To test the scalability of the resource discovery protocols, we simulated three network topologies growing incrementally from one to 50 devices. As in the Initialization scenario, there were no discovery requests, communication happens only to establish organizational structures.

Figure 4 shows the results summed up for all three topologies, except for the flooding protocols. Resource Flooding created such an excessive message load that we decided not to consider the protocol in our simulation any

[1] http://code.google.com/p/bluescatnet/

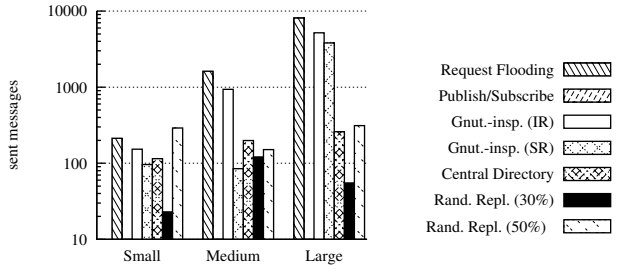

Figure 5. Message load measured in the *Search* scenario

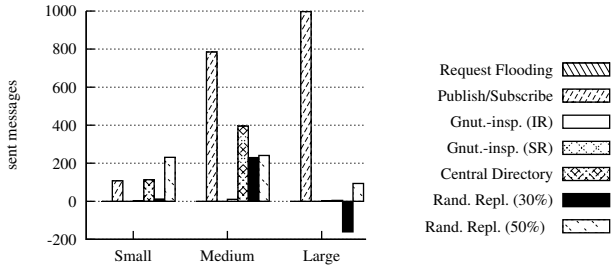

Figure 6. Message load measured in the *Resource Update* scenario

further. As in the Initialization scenario, Request Flooding does not communicate at all.

Even though the tree propagation of Publish/Subscribe improves Resource Flooding considerably, it still shows the highest message load. The Gnutella-inspired protocol performs very well with integrated routing (IR) and mediocre otherwise (SR), due to the establishment of routing structures. The Central Directory performs slightly better than Gnutella SR. Both rely on the established routing structures, but creating the Central Directory obviously requires fewer messages than flooding pings. Random Replication nicely shows the effect of the parameter r, denoting the amount of neighbors per device. $r = 50\%$ generated nearly 3000 more messages than $r = 30\%$, or 20 more messages in average per device in each topology.

E. Search Scenario

The Search scenario consists of numerous simulations that search for resources in various network topologies. Figure 5 shows the combined results grouped by network size. The logarithmic scale of the figure shows the exponentially growing message load of Request Flooding. This is unacceptable in large networks. Yet for small networks, Request Flooding performs well. As Publish/Subscribe keeps all knowledge locally, devices do not need to communicate to find resources. This advantage was bought at the price of very high message load in the Initialization and Grow scenarios. The message load of the Gnutella-inspired protocol is directly proportional to the network size for all network sizes tested. We conclude that the max. hop counter of the tested protocols was too high and thus never came into effect. There is, however, an observable advantage of the separate routing layer (SR) as compared to integrated routing (IR). Thus, the higher initial effort pays off here. The Central Directory achieves mediocre results which are slightly better than Random Replication with $r = 50\%$. The measures for Random Replication do not increase significantly with network size. A value of $r = 30\%$ constantly beats $r = 50\%$. On the other hand the randomness in the protocol makes the Random Replication difficult to predict.

F. Resource Update Scenario

Not only the network topology of an ad-hoc smart space changes constantly, the resources of devices may also come, change and go dynamically. We simulated the message load caused by resource updates in three network topologies having three different sizes. We conducted one simulation for adding, updating, and deleting a resource each, resulting in a total of 29 simulations. To determine the load of the resource updates without maintenance of support structures, we subtracted the load of an empty simulation.

The simulated message loads of the Resource Update scenario, as shown in Figure 6, are directly related on the degree of replication of the protocols. Request Flooding does not communicate at all. Publish/Subscribe, which fully replicates the global resource list, produces the highest message load. The Gnutella-inspired protocols do replicate resource information, but do not actively update the directories. Instead, they exploit resource listings contained in messages they forward for others, thus saving explicit update communication. The Central Directory protocol keeps a single (or very few) replications of the global resource list causing some update load, but considerably less than Publish/Subscribe. Even though the replication rate of the Random Replication protocol is relatively high (depending on the parameter r), it requires comparatively few messages. As expected, $r = 30\%$ caused more update messages than $r = 50\%$.

Figure 6 illustrates the randomness of Random Replication. The medium-sized network produced more messages than the large network, as devices happened to chose different neighbors. Also, the empty test run occasionally caused more messages than the update tests. This explains the negative value for $r = 30\%$ in the large network.

G. Random Actions Scenario

To gain a more realistic picture of the protocols we attempted to simulate a real-world scenario by applying a randomly chosen (but constant) scenario of 30 actions in a network of 50 devices and different topologies. Figure 7 shows the resulting message load, with and without network initialization. Not considering initialization, all approaches that separate routing perform better: Random Replication, Gnutella-inspired (SR), and the Central Directory.

Publish/Subscribe produces more messages as it keeps all replications up to date. Request Flooding and Gnutella-inspires (IR) produce a lot more messages, as they are based on a denser network and replicate less. They perform almost identically, as Gnutella-inspired (IR) is similar to flooding, if the network is smaller than the max. hop counter.

If we consider the initialization effort, the figures change. Now, Request Flooding and Gnutella-inspired (IR) perform best, as they neither require routing nor replicate much. Gnutella-inspired (SR) and Central Directory follow; they establish routing, but their replication effort is low. Random Replication and Publish/Subscribe need significantly more messages, as their degree of replication is higher.

H. Summary

The results differ considerably depending on the simulated network size, update and search rate. We observed that routing structures and replication do not pay off for small networks. The exponential growth of flooding does not make much impact for small networks. For medium to large networks the effect of support structures becomes visible and clearly helps the respective protocols to scale, as observed with Random Replication and the Central Directory. Due to its robustness, we prefer Random Replication for medium and large ad-hoc smart spaces. Its randomness can make its performance hard to predict, but it also helps the protocol cope with changes in the network. The high message load of Publish/Subscribe in presence of updates shows that full replication of resource information does not pay off, even though search is local.

V. Conclusions and Future Work

Ad-hoc smart spaces enable mobile devices to share their resources and context data with others near them spontaneously. This creates a highly dynamic network structure, which challenges resource discovery. We examined self-organizing decentralized resource discovery protocols for Bluetooth-based ad-hoc smart spaces. We pointed out the particularities of Bluetooth and presented an overall system architecture for an ad-hoc smart space middleware. We compared a number of resource discovery protocols with different strategies towards information replication, scatternet routing and information propagation. Our simulations show that Request Flooding performs best for small network sizes (≤ 5 devices), in spite of its exponential growth. In medium to large networks dedicated routing structures and an increased degree of replication pays off. The Random Replication protocol wins here, as its randomness makes it robust to changes in the network.

Further work is required to determine the ideal degree of replication r of the Random Replication protocol. Also, we will experiment with different routing algorithms. Finally, simple static context data, e.g., *weather="sunny"*, could be included in resource advertisements directly, omitting the need to ask for it explicitly. We will examine such optimizations in the future.

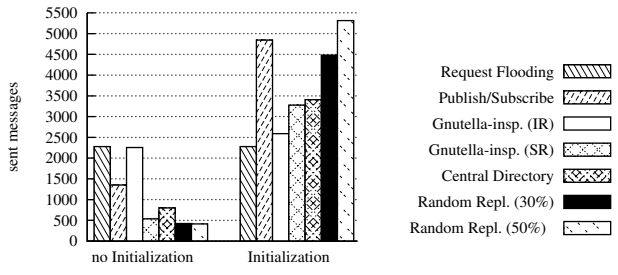

Figure 7. Message load measured in the *Random Actions* scenario

References

[1] A. K. Dey, "Understanding and using context." *Personal and Ubiquitous Computing*, vol. 5, no. 1, 2001.

[2] G.D. Abowd, et. al., "Towards a better understanding of context and context-awareness," in *Proc' Handheld and Ubiquitous Computing*, 1999.

[3] A. Brodt and S. Sathish, "Together we are strong – towards ad-hoc smart spaces," in *Proc' PERCOM*, 2009.

[4] E. Meshkova et. al., "A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks," *Comput. Netw.*, vol. 52, no. 11, 2008.

[5] Bluetooth Special Interest Group, "Specification of the Bluetooth System, Core. Version 1.1," 2001.

[6] T. Howes and M. Smith, *LDAP: programming directory-enabled applications with lightweight directory access protocol*. Macmillan Publishing Co., Inc., 1997.

[7] Stoica, I. et. al., "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc' SIGCOMM*, 2001.

[8] A. Greede and D. Mahony, "A service driven routing protocol for bluetooth scatternets," in *Proc' Personal Mobile Communications Conference*, 2003.

[9] A. Herms and M. Schulze, "Publish/Subscribe Middleware für Selbstorganisierende Drahtlose Multi-Hop-Netzwerke," in *Selbstorganisierende, Adaptive, Kontextsens. vert. Sys.*, 2008.

[10] T. Klingberg and R. Manfredi, "Gnutella 0.6," 2002, http://rfc-gnutella.sf.net/src/rfc-0_6-draft.html.

[11] Y. Chawathe et. al., "Making gnutella-like p2p systems scalable," in *Proc' SIGCOMM*, 2003.

[12] L. Clement, et. al., "Universal Description, Discovery, and Integration (UDDI)," 2008, http://www.uddi.org/pubs/uddi_v3.htm.

[13] S. Baatz et. al., "Bluetooth scatternets: An enhanced adaptive scheduling scheme," in *Proc' INFOCOM*, 2002.

[14] F. Cuomo and A. Pugini, "A linux based Bluetooth scatternet formation kit: from design to performance results," in *Proc' REALMAN*, 2005.