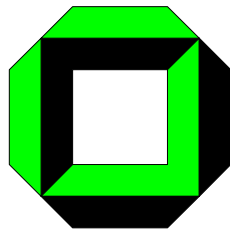


Comparison of Overlay Mechanisms for Service Trading in Ad hoc Networks

Michael Klein, Markus Hoffman, Daniel Matheis, Michael Müssig
{kleinm,hoffman,matheis}@ira.uka.de, michael.muessig@web.de

October 11, 2004

Technical Report Nr. 2004-2



University of Karlsruhe
Faculty of Informatics
Institute for Program Structures and Data Organization
D-76128 Karlsruhe, Germany

Contents

- 1 Introduction** **1**

- 2 Explanation** **2**
 - 1 What tool has been used for measuring? 2
 - 2 Which scenario has been used? 2
 - 3 What has been measured? 3
 - 4 Which settings have been used? 3

- 3 Benchmarks** **4**
 - 1 Service Benchmark SB6 4

- 4 Design Points** **6**
 - 1 Service Rings 6
 - 2 Lanes 7
 - 3 Flooding 7

- 5 Results** **9**
 - 1 Results on Benchmark SB6 9
 - 1.1 Response Time 9
 - 1.2 Message Efficiency 10

- 6 Conclusion** **14**

Chapter 1

Introduction

In this report, the efficiency and effectivity of different service trading overlays is determined by simulation. These overlays have been developed in the DIANE project [1]. Especially, we test two approaches: Service Rings as presented in [2] and Lanes as presented in [3, 4]. We compare the results with the trivial service trading protocol of flooding the query through the network.

Chapter 2

Explanation

1 What tool has been used for measuring?

For the measures, the tool *DIANEmu* [5] was used. This high-level simulator for distributed protocols was developed in the DIANE project, too, and is specialized for running protocols above the routing layer. Its strength lies in the upper layers, i.e. a realistic user and user agent model as well as support for writing application protocols. On the other hand, the network layer is modelled very simply for two reasons: First, the number of network parameters is very small, which helps to concentrate on the effects of the higher layers, second, the runtime performance of the simulation is increased. The simulator performs an event-based simulation, i.e. all events like the arrival of a message, usage of protocol functions etc. are put into an event queue together with the time when they will occur. The event queue processes them step by step, which commonly leads to further events being enqueued. As an advantage, the simulation time is strictly separated from the real time, making the measurements independent from the underlying machine.

2 Which scenario has been used?

In our project DIANE, we use a campus scenario, i.e. the users are students moving around the campus with their mobile devices. All devices are forming an ad hoc network. Here, no infrastructure is used, but the devices provide these infrastructural tasks (like routing) as a group. Moreover, each user is able to *publish* services and therefore make them available for other users in the network. On the other hand, he is able to *unpublish* these services later. Services of other users can be accessed by *using* them.

The functionality of publishing, unpublishing and using a service is not directly provided by the trading protocols. Indeed, these trading protocols offer more basic functions like logging into the overlay structure, logging out of the overlay structure, searching for a service in the overlay structure, offering a service to the overlay structure and revoking a service from it. Therefore, a *user agent* mediates between user and protocol by correctly invoking the appropriate functions. For example, if the user wants to use a service, the agent tries to log into the overlay structure and repeats this process until it is successful, then searches for an appropriate service, invokes one of the results and logs out of the overlay structure.

The protocols use the network layer to exchange messages between the different entities. The network layers offers different sending possibilities: unicast, broadcast and anycast. In the simulation, a a perfect routing protocol is assumed, i.e. the message is routed on the shortest

way to its destination using the omniscience of the simulator, which keeps track of the positions and radio ranges of all devices all the time.

3 What has been measured?

To gain values, the simulator uses the concepts of announcements and gauges. If certain events occur within the simulator (e.g., a message is sent or a service search is started), the simulator throws an announcement with details of the event. These are processed by a set of gauges that have registered for this type of events. For this report, we used three gauges:

- *MessageGauge*. It protocols each message in the network together with its sender, receiver, size, hopcount and time delay. This can be used to calculate the number of messages that were used. We normalized the value by dividing it through the number of service searches.
- *TimeDiffGauge*. It protocols the time difference between the start announcements of a service usage and its finish announcement. Thus, this gauge measures the response time.

4 Which settings have been used?

The settings were separated in two classes:

- A *benchmark* captures all parameters concerning the “external world”, for example the number of users, their motion model, the radio range of the devices etc. They try to provide a realistic picture of the world and are used as a basis for comparing the protocols. Therefore, benchmarks are not altered very often, but are fixedly given and standardized by the DIANE project. Typically, they have no or only a few parameters like the number of users. For this report, we used the benchmarks SB6¹. In the next chapter, its settings are explained in detail.
- Other parameters are captured in a *design point*. Thus, a design point specifies which trading protocol is to be used and how to configure it. Typical parameters are the maximum length of a lane, the time between ping messages and so on. Design points are not fixed, but should be altered by the developers to improve their protocols. In Chapter 4, we present the design points in more detail.

¹SB means service benchmark

Chapter 3

Benchmarks

For this report, we used a benchmarks which tries to capture our campus scenario in a realistic manner: SB6.

1 Service Benchmark SB6

In SB6, users represent students moving around on the campus of the university of Karlsruhe. They randomly pick a location on the campus (like the cafeteria, a lecture hall, the library etc.) and move to that area in a realistic manner, i.e. using the existing roads and bridges on the campus. After having arrived, they wait for a certain amount of time, chose another location and start to walk to that point. Speed and waiting time resemble values of typical pedestrians.

From time to time, the user perform operations on services. This behavior is modelled by the state machine depicted in Figure 3.1. After a certain waiting time ($timeBeforeFirstPublication$), each user starts to publish a service, which he unpublishes after the $timeBeforeUnpublication$. At the same time, he publishes a further service after the $timeBetweenPublications$. Simultaneously, he uses services with a temporal difference given in $timeBetweenUsage$. None of these time values is a fixed value, but are taken from a uniform distribution $U(a, b)$, where a is the smallest and b the largest possible value. The benchmark SB6 can be configured by two parameters, given as

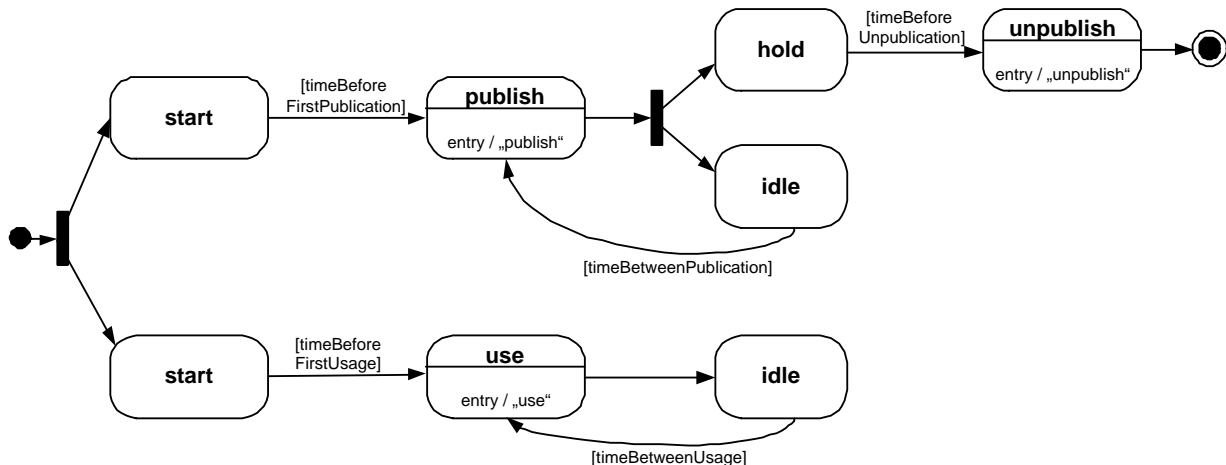


Figure 3.1: State machine of users to simulate service usage, publication and unpublication.

Parameter	SB6.f.nou
timeBeforeFirstUsage	$U(300, 6000)$
timeBetweenUsage	$U(f, 2f)$
timeBeforeFirstPublication	$U(60, 900)$
timeBetweenPublication	$U(2, 4)$
timeBeforeUnpublication	∞
number of users	<i>nou</i>

Table 3.1: Settings for SB6. $U(a, b)$ represents a uniform distribution, where a is the smallest and b the largest possible value. All times are given in seconds. ∞ denotes that published service are never unpublished during simulation time.

SB6.f.nou. f is given in seconds and determines the *timeBetweenUsage* as $U(f, 2f)$. The second parameter *nou* is the number of users. *nou* is not restricted, but typical values are 16, 32 and 64. The details of the parameters can be found in Table 3.1. SB6 is modelled to be realistic, i.e. users don't change their service publications all the time, but publish their service offers very quickly at the beginning, holding them all the time until the simulation stops.

In SB6, 15 different services were used, which were clustered in 3 categories. When using or publishing a service, one of these 15 services was chosen randomly. A service offer o matches to a service request r , if both r and s represent exactly the same service. To perform a rough comparison, the category of r and s could be used.

The network layer in SB6 is modelled as a WLAN in ad hoc mode using a omniscient routing protocol (see Section 1). The radio range of the users is 250 m. The network is protected against partition, i.e. if there is no route between two users, the network layer will deliver the message nonetheless using a penalty hop count of $nou + 1$. Overhearing of messages is switched off.

The user agent, which mediated between user and trading protocol (see Section 2), operates with the following values in SB6: (1) It retries a login into the overlay structure every 60 s until it is successful. (2) When the user does not need to be logged in anymore because there are no more pending usage or publication requests, the agent waits 120 s before actually calling the logoff command at the trading protocol. If a new usage or publication request arrives from the user in the meantime, the existing connection to the overlay structure can be used and the logoff process is postponed.

Chapter 4

Design Points

A design point specifies the trading protocol and its configuration. For our tests, we used three different trading protocols: Service Rings, Lanes and a trivial flooding approach. In the following chapter, they are shortly presented.

1 Service Rings

Service Rings [2] are a typical semantical overlay structure. Devices that offer similar services are grouped together to form a logical ring. Each ring has a designated device, the service access point (SAP), which acts on behalf of the ring by processing all service queries. This is possible as each SAP knows all services (or at least a summary of them) offered in its ring. Thus, it routes a query through its ring if there might be a service in it that could *possibly match*. SAPs themselves form rings of a higher level, which leads to a hierarchical structure (see Figure 4.1). Here, three basic rings have formed: R_1 , R_2 , and R_3 . Also, their SAPs v_3 , v_6 , and v_{10} are forming a ring on the next higher level. Its SAP v_6 is marked with a white asterisk.

Maintaining such a ring structure is a difficult task. Periodically, ping messages are sent through every rings to detect broken or inefficient links and vanished nodes. Repairing these errors can be very costly, especially when SAPs vanish. Moreover, to get an optimal structure,

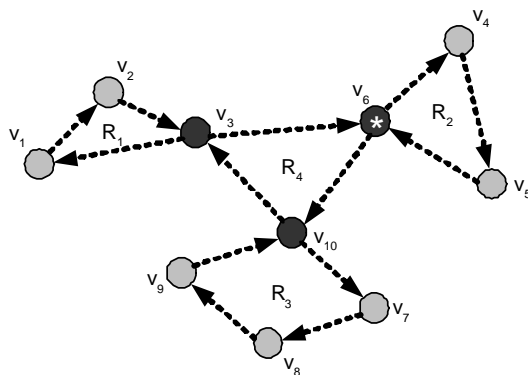


Figure 4.1: Example of a service ring overlay. Three basic rings have formed: R_1 , R_2 , and R_3 . Their SAPs v_3 , v_6 , and v_{10} are forming a ring on the next higher level. Its SAP v_6 is marked with the white asterisk.

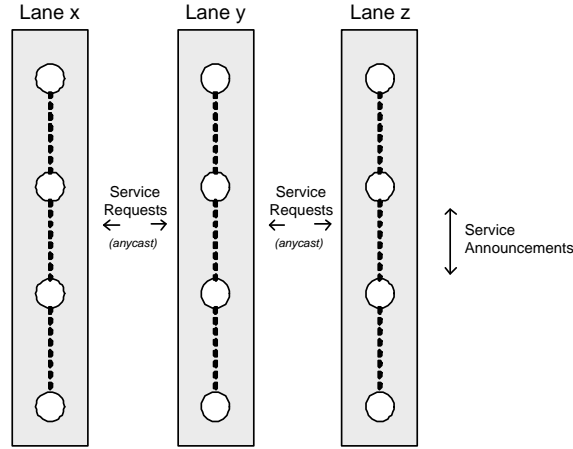


Figure 4.2: Schema of a Lanes overlay.

rings are split or merged if their size is too big or too small.

Typical parameters are the maximum size of the rings, the time between ping messages as well as the waiting time for the arriving result messages.

2 Lanes

Lanes [3, 4] is a non-semantic overlay, i.e. it is not built according to the offered services of the devices. It is organized along two dimension: one for service announcements and one for service requests. Devices are grouped into lanes of devices – within a lane devices maintain overlay links to their predecessor and their successor. Each device member knows the addresses of neighbored lanes, but no single device from it. Thus, communication between lanes uses anycast (see Figure 4.2).

In order to announce a service, a device has to send its description to the predecessor and successor in the lane. They store it and forward it within the lane. Thus, the description is distributed in the whole lane. To search for a service, first, the requestor compares the request with the offers stored on his own device. These are the services offered within the own lane. If nothing is found here, the search message is sent to neighboring lanes using anycast.

Maintenance of lanes is easier than service rings as the structure has been designed to be explicitly lightweight. Each lane member sends a periodical ping message to its successor in the lane. Vanished nodes and broken links can be repaired locally with little effort. If a lane is too big or too small, it is split or merged which can be done in a zipper like fashion.

Typical parameters of lanes are the maximal and minimal length of a lane as well as the time between two ping messages.

3 Flooding

As comparison for service rings and lanes, a very simple protocol shall be used: flooding. Here, service offer descriptions are not distributed within the network at all but stay on the offering device exclusively. Thus, requests have to be forwarded to all members of the network, where they are matched against the offers. This approach does not need any overlay: If a request arrives

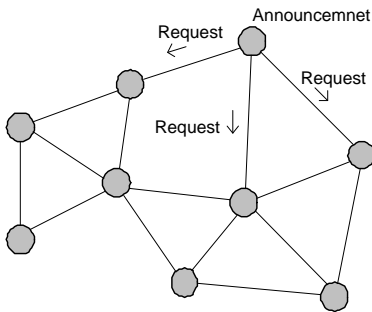


Figure 4.3: Flooding does not need any overlay but relies on the existing hop cop connections.

at a device, it checks whether it has already seen this request. If yes, the request is discarded, if not, a comparison with the own offered services is performed and the request is forwarded to all devices that are reachable within one hop.

This approach is very robust, as no overlay structure has to be maintained and optimized. However, the flood of request messages originating from one service search can be very costly.

Flooding only has one parameter: the time to wait between the start of a service search and the return of the result. Only result messages arriving within this time period are aggregated and returned to the user.

Chapter 5

Results

1 Results on Benchmark SB6

In the following, the results on Benchmark SB6 are presented.

1.1 Response Time

Figure 5.1 shows the average response time of search queries in SB6 for the different protocols and different search frequencies. The response time is the time between starting the service search and the arrival of the complete result set. The rings are performing very bad, which leads to a very high values on the y axis.

Thus, in Figure 5.2, the rings have been dropped. The values behave like expected: The flooding approach always has a response time of 3000 ms, which is exactly the chosen timeout value, i.e. the protocol collects incoming response messages for 3 seconds before returning them

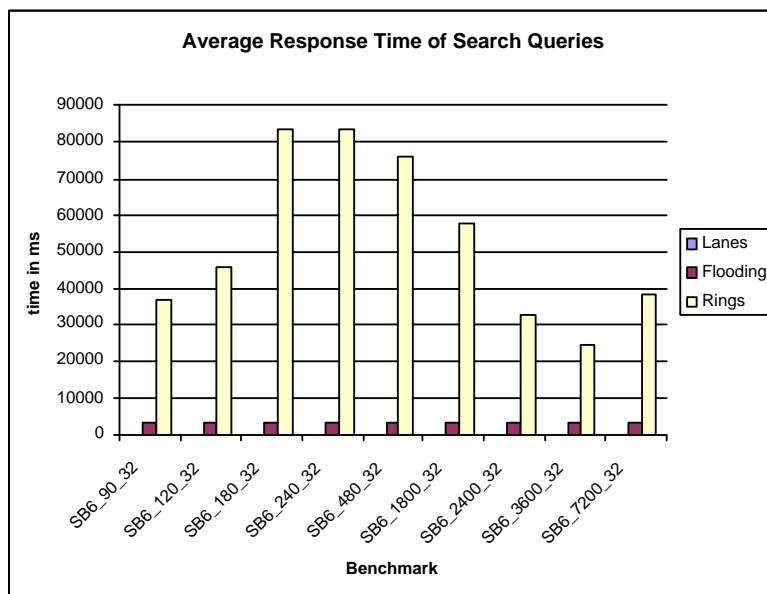


Figure 5.1: Average response time of Lanes, service rings, and flooding in SB6.

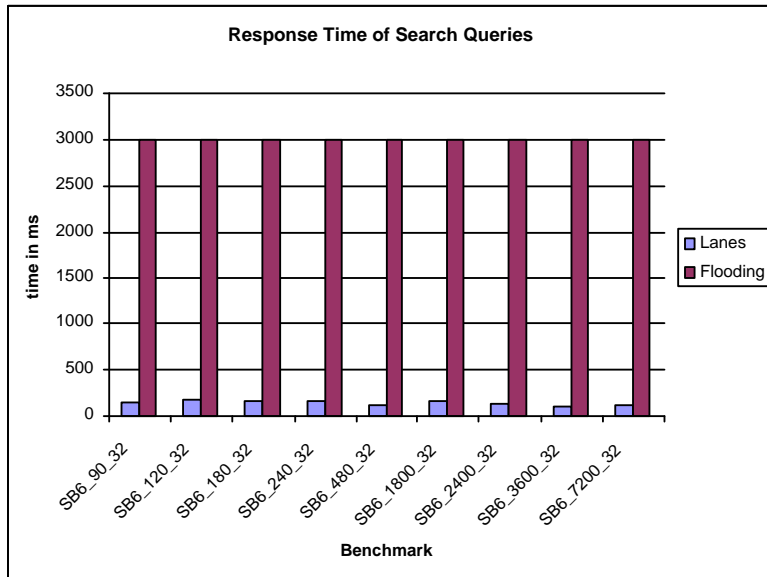


Figure 5.2: Average response time of Lanes and flooding in SB6.

to the user. In contrast to that, Lanes performs much better: All response times are around 200 ms. This improvement stems from the fact that in Lanes no timeout mechanism is used for service search. Two query messages are traversing the lanes on both sides of the initiating devices in parallel while collecting matching offers. When they reach the right and left outer lane, their results are directly sent back to the requestor.

As a conclusion, we can derive the following result: As overlays set up and maintain a guaranteed structure, operations that use this structure (like service search or service announcement) need not rely on timeouts, which have to be set to rather high values in order to preserve effectivity. In overlays, the end of an operation typically is determinable by the arrival of certain messages.

1.2 Message Efficiency

Figure 5.3 shows the average number of messages per service search. The number contains all messages, i.e. functional messages like search and result messages as well as maintaining messages like ping and repair messages in case of overlays. Again, the service rings perform very bad. They are relatively good in case of a high search rate (a search each 480 seconds = 6 minutes or more often) using 120 to 200 messages per search. In situations of a low search frequency (a search every 1800 s = 30 minutes or more seldom), the message number increases rapidly to over 1,000 messages per search. In these cases, the high effort to set up and maintain the complex overlay structure cannot be amortized by the searches anymore. However, the service rings are worse than the two other approaches in any case, which must be explained by their structure that seems to be too complex for ad hoc networks. It could be better in cases of far more user numbers as higher hierarchies of rings could be formed.

In Figure 5.4, the values for service rings are omitted, which leads to a more detailed view on Lanes and flooding. As expected, the flooding approach is nearly independent from the search frequency: It uses up to 40 messages per search as each devices forwards a query exactly once

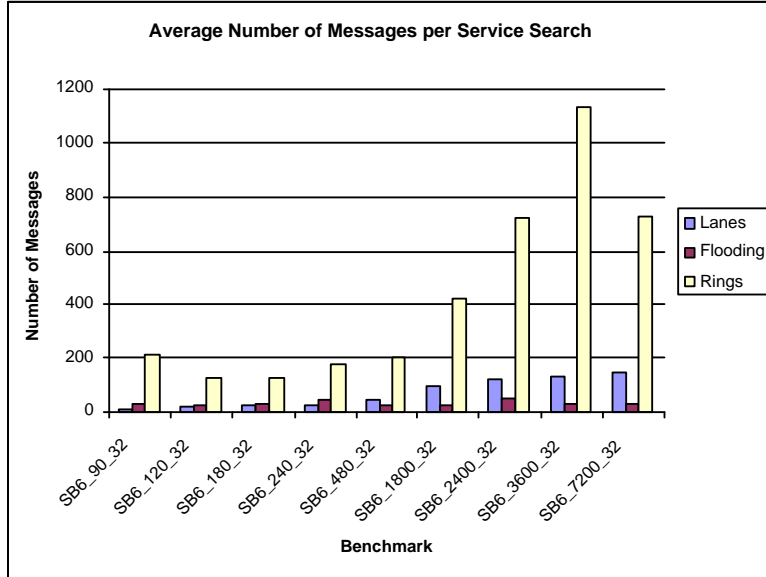


Figure 5.3: Average number of messages per service search in Lanes, service rings, and flooding in SB6.

and some devices additionally return a result message to the requestor. On the other hand, Lanes is dependent from the search frequency in the same fashion as service rings. With a high search frequency (a search each 90 to 240 seconds), 22 or less messages per search were used. With decreasing search frequency (a search every 480s or more seldom), the effort to set up the overlay structure is not justified anymore as more than 100 messages per search have to be used in average.

When doubling the number of users to 64, the efficiency of Lanes further improves. The results are shown in Figure 5.5. Here, we also differentiated between two maximum lane lengths: Lanes.3 denotes an overlay with lanes of maximum 3 devices, Lanes.12 denotes an overlay with lanes of maximum 12 devices. The flooding approach behaves as expected and needs about 90 messages per service search (i.e. 64 for forwarding the message and about 26 result messages), which is roughly the double amount of messages than in the case of 32 users. Lanes, however, scales much better: The number of messages per search increases very slightly to approximately 25 messages (claret curve, Lanes.3). This results from the fact that the two dimensions of lanes roughly lead to a message effort of $O(\sqrt{n})$ where n is the number of users. Further notice that Lanes.3 now is more efficient than the flooding approach in a broader range: searches need to be done every 900 seconds or more often.

Comparing the two different maximum lane lengths 3 and 12 leads to further interesting insights. When the search frequency is very high (searches each 30 to 120 seconds), it is affordable to set up a structure with long lanes (Lanes.12, i.e. 5 to 6 lanes with 64 users), even if service offers need to be distributed to 12 devices. When search frequency is medium (searches each 240 to 900 seconds), shorter lanes become better (Lanes.3, i.e. more than 20 lanes with 64 users) as offer description only need to be copied to three devices. As the search frequency further decreases to low values (searches more seldom than every 1800 seconds), it is not reasonable to build an overlay at all – flooding becomes the algorithm to use. As a result, this leads to the idea of self-tuning parameters: According to the current search frequency, the maximum lane length

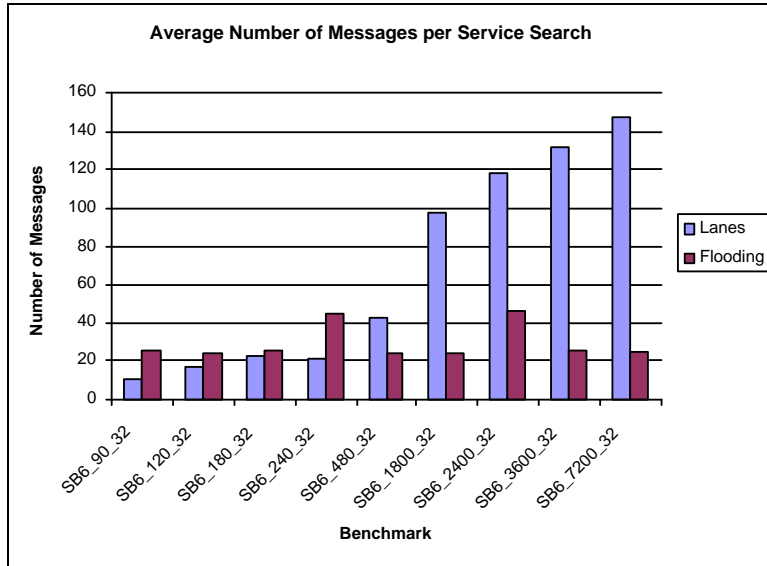


Figure 5.4: Average number of messages per service search in Lanes and flooding in SB6.

is adapted dynamically. We will implement these features in the next version of Lanes.

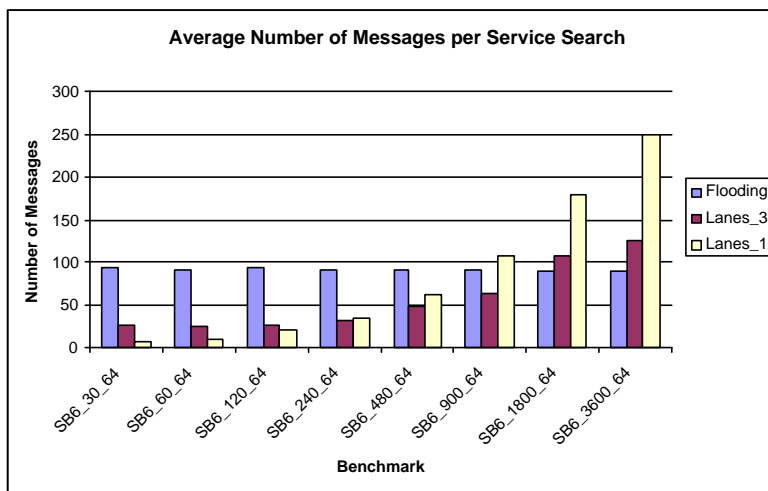


Figure 5.5: Average number of messages per service search in Lanes with different lane lengths compared to flooding in SB6. Here: 64 users.

Chapter 6

Conclusion

In this report, we compared three trading mechanisms for service trading in ad hoc networks: the overlay-based service rings and Lanes as well as the trivial flooding approach as comparison. As basis, we chose a realistic scenario (captured in the Benchmarks SB4 and SB6) of students moving around on a campus while performing different service operations.

We achieved the following results:

- The overlay structure of *Service Rings* seems to be too complex for ad hoc networks. Even high search frequencies could not amortize the effort that is needed to set up and maintain their structure. Perhaps a substantially greater amount of users (more than 1,000) could help to bring out the strengths of the protocol as hierarchies of more than 2 levels would be formed. However, in reality, ad hoc networks of such sizes do not seem to be reasonable.
- Lanes on the other hand was explicitly designed as an lightweight overlay. As the results show, it had a better *message efficiency* compared to flooding when the search frequency of the users is not too small. With 32 users, searches should be done at least every 6 minutes, with 64 users, searches should be done at least every 15 minutes. These values seem to be realistic for most scenarios.
- In general, Lanes *scales better* with the number of user than the flooding approach. While in flooding the messages per search doubled when the number of users had been raised from 32 to 64, in Lanes, the average number of messages increased by approximately 15% only (from 22 to 25).
- Interestingly, Lanes can be *adapted* to varying situations by changing the maximum length of a lane. At the moment, this can only be done manually by altering this protocol parameter. For the next release of Lanes, we plan to self-tune this parameter during run-time.
- Finally, Lanes has *a substantially better response time* than the flooding approach as its well defined overlay structure helps avoiding a usage of longish timeouts.

In a nutshell: lightweight overlays like Lanes can fasten service search tremendously and reduce the message overhead at the same time in scenarios with realistic user motion and frequent service usage.

Bibliography

- [1] Institute for Program Structures and Data Organization, Universität Karlsruhe: DIANE Project. <http://www.ipd.uni-karlsruhe.de/DIANE/en> (2003)
- [2] Klein, M., König-Ries, B., Obreiter, P.: Service rings – a semantical overlay for service discovery in ad hoc networks. In: Proc. of the Sixth Intl. Workshop on Network-Based Information Systems (NBIS2003), Workshop at DEXA 2003, Prague, Czech Republic. (2003)
- [3] Klein, M., König-Ries, B., Obreiter, P.: Lanes – a lightweight overlay for service discovery in mobile ad hoc network. In: Proc. of the 3rd Workshop on Applications and Services in Wireless Networks (ASWN2003), Berne, Switzerland (2003)
- [4] Klein, M., König-Ries, B., Obreiter, P.: Lanes – a lightweight overlay for service discovery in mobile ad hoc networks. Technical Report 2003/6, Universität Karlsruhe, Faculty of Informatics, Angers, France (2003)
- [5] Klein, M.: DIANEmu – a java-based generic simulation environment for distributed protocols. Technical Report 2003-7, Universität Karlsruhe, Faculty of Informatics (2003)