University of Innsbruck

Department of Computer Science

STI · INNSBRUCK

Abstraction in Group Based Service Discovery

Bachelor Thesis

Florian Fischer

florian.fischer@student.uibk.ac.at

February 27, 2009

Supervisor: Univ. Prof. Dr. Thomas Strang

thomas.strang@sti2.at

GSD is novel service discovery protocol primarily focuses on mobile ad hoc networks. It is designed to operate under the very specific constraints imposed in such scenarios. However GSD still has potential scalability problems, due to the required "fine-tuning" of certain protocol parameters concerning the abstraction applied in the discovery process.

In this thesis we aim to analyze this scalability problem, in terms of its impact and how to avoid it – in other words how to chose "good" values as parameters in realistic scenarios.

In this scope a closer look at the special properties of mobile ad hoc networks and scalability from a theoretical point of view is needed. Based on this results this thesis also briefly points out how research results from other fields in wireless networking could be used to optimize GSD's performance.

Contents

Chapter 1

Introduction

1.1 Ad Hoc Networks

During the last years portable computation devices and wireless communication have become increasingly widespread, affordable and available in terms of both computational capabilities and bandwidth available. This fact goes hand in hand with roughly (as of 2005) over 2.14 billion mobile phone subscribers worldwide [1].

However cellular networks are still a combination of wired and wireless technology. Users are typically mobile and equipped with power and energy-constrained devices, while there is still a lot of fixed infrastructure in form of base stations and the associated equipment in place.

These are challenging scenarios on their own but an even more complex network type, made possible by the widespread adoption of mobile interconnected devices, are so called *mobile ad hoc networks* (MANETs). They can be concisely defined as following:

> "A Mobile Ad-Hoc Network(MANET) is an autonomous system of mobile nodes."[1]

This means that in these networks mobile units equipped with wireless connectivity usually operate without the needs for fixed infrastructure. This fact sets them apart from traditional wireless networks, like cellular networks and wireless LAN. In the absence of fixed infrastructure nodes have to organize themselves and also route traffic for other participants in the network to facilitate multihop communication.

There are numerous possible applications for such networks. For example rapid exchange of presentations and documents at conferences or meetings, ubiquitous internet access, exchange of traffic information between by passing cars on highways, ...

While the underlying network technology is relatively robust and readily available there are a lot of open issues which slow the practical deployment of applications for ad hoc networks. This is because there is a number of important

[1] `http://www.mobiletracker.net/archives/2005/05/18/mobile-subcribers-worldwide`

factors which set MANETs apart from traditional networks and which need to be taken into special consideration.

As already mentioned nodes in an ad hoc network are expected to be mobile, autonomous, equipped with wireless transceivers and expected to organize themselves dynamically. These underlying assumptions have a number of consequences. Wireless links are inherently unreliable and prone to environmental influence such as interference with other signals, obstacles, ... Furthermore as mentioned nodes are expected to be mobile and thus can freely join or part from the network. Combined with broken wireless connectivity and node failure this leads to a highly dynamic and unstructured network topology. On top of that devices themselves are often limited in two regards. First of all due to their mobility they are usually battery equipped and thus energy limited. Secondly a lot of the devices participating in a MANET might have comparatively little computational power.

So efficiency in regard to these scarce resource is a very important concern when dealing with ad hoc networks. Especially available energy and bandwidth need to be conserved and while these two are obviously related this connection is non-trivial and a complex subject. Nevertheless protocols need to address these issues while still maintaining the scalability needed to support the large number of participants which are envisioned to take part in an ad hoc network.

1.2 Service Discovery

An active research area which is central to the practical usability of ad hoc networks is *service discovery*. Its main objective is to enable the efficient discovery of *services* in a network, wherein a *service* can be loosely defined as any kind of functionality implemented in hard- or software which provides a "benefit" (provides information, performs an action, ...) to other entities in the network. Obvious examples could be a certain file, a printer, an arbitrary information service, a ticket reservation system or a multitude of other services which offer a specific functionality to users.

There are a number of solutions which address this problem for the scope of traditional wired networks, each with specific scenarios and application areas in mind. For example Salutation [2], UPnP [3], SLP [4], Jini [5], UDDI [6], DNS-SD [7], Bluetooth SDP [8], ...

However, as mentioned, wireless ad-hoc networks pose a particular set of restrictions which set them apart from traditional networks. These restrictions also mandate different requirements for service discovery protocols. The traditional solutions do not take these requirements into account since they were often developed to cover a very specific use case only. Thus, they are mostly not very well suited for mobile ad hoc networks.

However an important key insight is that service discovery solutions would be *especially* useful in ad hoc networks, compared to wired networks. While in wired networks static configuration might still be feasible, this approach utterly fails in ad hoc networks. So obviously service discovery solutions which are tailored towards the requirements of ad hoc networks are a cornerstone to

make these networks convenient to use for the end-user.

There are several attempts which try to fill this gap.. One of the most comprehensive approaches which explicitly covers both basic problem domains (network protocol behavior and service description) is GSD [9].

However under certain circumstances GSD suffers from poor scalability, which this thesis tries to more closely identify. The rest of this thesis is structured as following. In Chapter 2 GSD is introduced with a special emphasis on its problematic behavior in certain situations. Chapter 3 covers the theoretical modeling of GSD's scalability and the problems associated with it in wireless ad hoc networks. Chapter 4 presents simulation results in order to evaluate the scalability using realistic scenarios. Finally Chapter 5 concludes the thesis.

Chapter 2

GSD — Group-based Service Discovery

2.1 Motivation

GSD ([9]) is a service discovery protocol developed especially for pervasive environments and mobile ad hoc networks. It tries to take the characteristics of those networks into account and this sets it apart from traditional service discovery protocols which mainly operate well in wired and stable environments. Traditional protocols mostly take a centralized approach and assume reliable infrastructure. Often such protocols employ some type of central server where nodes can register the services they want to offer. And this often goes hand in hand with a request based discovery scheme in which clients actively search for specific services they are interested in (*pull based discovery*). However such a centralized, registry based system does not work well in a fast changing ad hoc network.

Fully decentralized solutions (e.g. [10]) on the other hand often rely on network wide broadcasting to actively advertise available services (*push based discovery*). In turn they "waste" a lot of processing power on devices and a lot of scarce bandwidth, since protocol messages are propagated through the complete network. Thus such solutions are often poorly scalable in regard to large networks and far from optimal solutions in resource constraint environments such as mobile ad hoc networks.

Another major point concerning existing solutions is that they regard service description/service matching and the discovery architecture of the protocol as two largely separate issues. Moreover service descriptions are often very simplistic: Interface descriptions ([5], [3], a set of attributes ([4], [10]), unique identifiers ([8]) or combination of these possibilities.

GSD tries to overcome the previously mentioned problems by coupling *service matching* and the *discovery architecture*. What this means is that GSD takes cached "semantic" descriptions of services into account in order to form functional *service groups* and in turn selectively forward protocol messages. Thus GSD attempts to limit the use of inefficient broadcasting while still retaining a completely decentralized architecture.

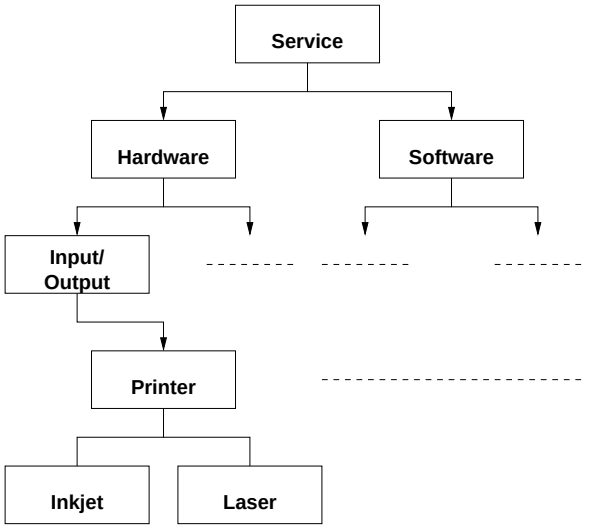


Figure 2.1: A service hierarchy

The next section covers the basic concepts of GSD in more detail.

2.2 Service Descriptions and Discovery

2.2.1 Service Description and Protocol Messages

As mentioned GSD covers the service description format as well as the discovery protocol. It uses the Web Ontology Language (OWL) [11] as format to facilitate semantically rich service descriptions. This ontology based approach offers several advantages. For example since OWL is a standard in the semantic web area a lot of tools (inference engines, modeling tools, ...) and existing ontologies can be reused.

In OWL concepts are expressed using classes with associated properties. These can be used to build semantically rich service descriptions, which can support dependencies and restrictions such as cardinality, domain and range for allowed values, transitivity, inverse, ... to describe the relationship between resources. Overall such a rich description language makes much more sophisticated service matching possible.

Furthermore OWL allows sub-classing in order to relate a more specific class to a more general one. So *owl:Class* and *owl:subClassOf* are two basic building blocks. Also GSD makes use this two concepts in order to group services according to their functionality and build a service hierarchy with a tree structure based on the *owl:subClassOf* relationship. It then uses this service hierarchy and grouping to selectively forward protocol messages only to "appropriate" nodes and limit network traffic. An example of such a possible service hierarchy is shown in Figure 2.1. The underlying discovery protocol of GSD makes use of three main concepts:

1. Service announcements of available services to neighbor nodes.

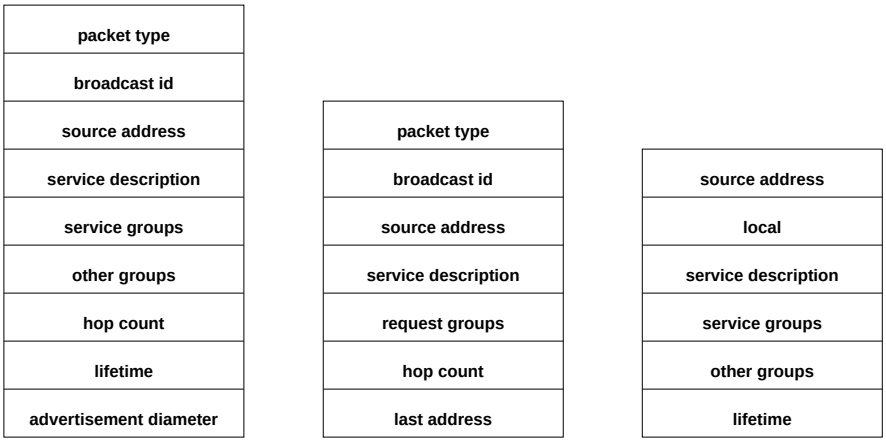


Figure 2.2: Formats for (a) advertisement messages, (b) request messages and (c) cache entries

2. Caching of advertisements and thus of information regarding services in the vicinity.

3. Request based discovery and selective forwarding of discovery requests, instead of broadcasting.

The subsequent part of this chapter briefly covers the messages involved and the key points of GSD, followed by an explanation of how these inter-operate in the service discovery process.

Initially each device has to announce the services it hosts to the nodes in its neighborhood. These periodic advertisement messages are sent as broadcasts and contain (among other data) the following information – each followed by a short explanation where appropriate. The first two basic fields an advertisement message contains are:

- broadcast-id
- source address

Broadcast-id here refers to a monotonically increasing identifier which corresponds to a single broadcast event. In combination with the source address of an advertisement message this serves as an unique identifier which in turn allows a node to discard duplicates of advertisement messages it has already processed.

- service descriptions
- service groups
- other groups

The next two fields simply cover what services are present on a device and in which groups in the service hierarchy they belong. *Other groups* contains information the node has cached about services in it's "vicinity" and summarizes information received from other nodes via advertisements.

- hop count
- advertisement diameter
- lifetime

Finally each advertisement message contains a *hop count* field, which is incremented at each node, and the maximum hop count an advertisement message is supposed to travel until it is discarded (*advertisement diameter*). Thus advertisements are bounded and do not flood the whole network. The *lifetime* included in an advertisement is supposed to be a hint for other nodes how long they should cache the received service information until it expires. However, [9] does not specify the format of the *lifetime* field. Figure 2.2(a) summarizes the format of advertisement messages.

As mentioned information received via advertisements is cached on devices along with the local service information (marked with the *local* field). The format of such a cache entry is depicted in Figure 2.2(c). The *other groups* field stores service groups which the sender has seen in its vicinity and is used to forward requests selectively to appropriate nodes. The caching strategy proposed in [9] is to replace the entry with the least remaining lifetime if the cache is full. However depending on the use case and device capabilities other replacement strategies could be used easily.

Active searching for specific services is done by means of request messages. The information relevant to the discovery processes is the following:

- service description
- request groups

The *service description* is the OWL based service description used for the actual matching of services against the specific request. *Request groups* are the groups in the service hierarchy (revert to Figure 2.1 for an example) to which the wanted service belongs. It is important to note that it is very well possible to deliberately search for a less specific group higher up in the hierarchy and closer to its root, in order to potentially receive more responses. This enables a more fuzzy search. A more detailed description of the discovery process follows later in this thesis. Additionally several other fields in request messages like *source address*, *last address*, ... are used to maintain reverse route tables for replies to requests. However the same functionality could also be implemented by falling back to ordinary ad hoc routing protocols like AODV [12] or DSDV [13] and hence is not really a central point in this thesis. However a complete overview of all the fields is shown in Figure 2.2(b). The subsequent section explains the discovery algorithm in more detail.

2.2.2 Discovery Algorithm

When looking for a specific service a node first examines its local service cache, since the node might also be a service provider itself. If there is no local hit a service discover request is generated and the *other groups* information in cache entries is used in order to try to selectively forward the request to one or multiple other nodes. The key point is that if the *other groups* field of an entry contains the group(s) given in the *request groups* of the service discovery request message, then there is the possibility to find a suitable service close to the node from which the message originated. In this way request messages can be passed on based on service group information, while potentially reducing the need to broadcast, until they finally arrive at a node with a service entry that matches the included service description.

A key issue is that the service group information in the request is determined by the sender, in the sense that it can choose to use a more general service group, further up in the service hierarchy. In this way an can specify how coarse or fine grained the service discovery operation will be performed, and thus also specify a trade-off between then number of results it receives and the network load generated. The intrinsically difficulty in this is that this decision has to be made according to incomplete, local information alone.

It is still possible that a node does not have the necessary information in its cache to selectively forward a request it falls back to broadcasting the request message to all nodes within reach. This can happen if the *request groups* in the message are too specific or also if the node has not received any information regarding suitable groups.

Another case that is possible is that requests are "falsely" forwarded in a direction where no matching service is available. Possible reasons include that either the node with the service is not available anymore due to node mobility, lost connectivity, ... and that the corresponding cache entry has not expired yet. Another possibility which should be noted is however that the service might be classified in a suitable service group in the hierarchy, however its description does not match the requirements in the request – this case would be typical for a too generic service description. In both mentioned cases it already becomes apparent that the abstraction chosen for the *request groups* field fundamentally influences protocol behavior.

An example of this selective forwarding is depicted in Figure 2.3 (from [9], and also explained in more detail there). In this example the request source RS is actively searching for the service S1 which is classified to belong in service group G1 and available at the service provider SP. Below each node the entries of the cache (built from previously received advertisements) are shown. For example

$$S3(G2), G1, G2 \rightarrow N2$$

in N3's cache means that N3 has received an advertisement from node N2. And more particularly that N2 has a service S3 belonging to group G2 and furthermore that N2 has seen services of group G1 and G2 in its vicinity. N2 in turn originally received the information regarding G1 from node N1, whose neighbor SP hosts the service. Thus when the request reaches N3 it is selectively

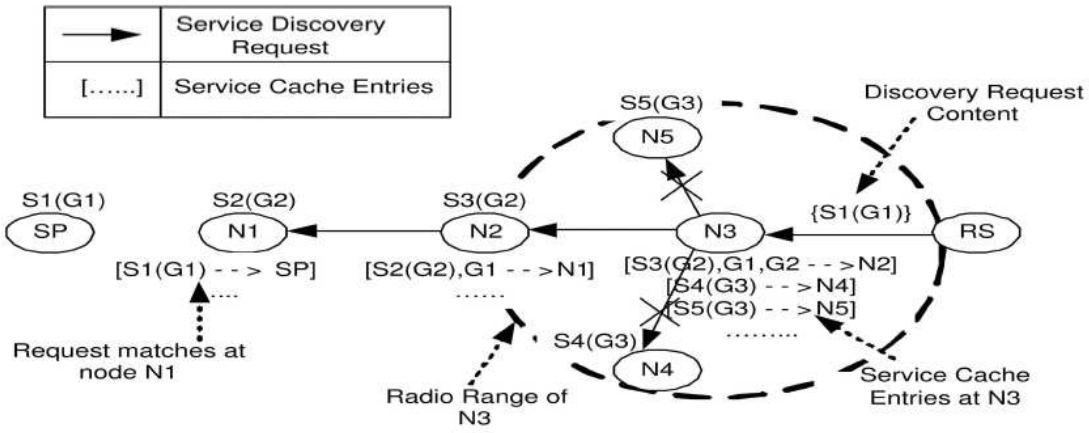


Figure 2.3: GSD request forwarding

forwarded to N2 and then to N1 (according to the information in N2's cache). At N1 the request finally results in a positive match. Another example, based on our concrete simulation setups follows in Section 4.1.

So obviously it is not necessary to *directly* reach the node hosting a wanted service, merely reaching one of its adjacent neighbors is enough. It is also noteworthy that RS initially had to broadcast the request in this example to reach node N3, from whereon it could be selectively forwarded based on the existing group information.

The request in this example is not forwarded to nodes N4 and N5, which host service S5 and S4, since S5 and S4 both belong to service group G3. However talking the hierarchical ordering of service groups into account there are more possible scenarios. Consider the cache entries at N3 to be

$$S3(G2), G1, G2 \rightarrow N2$$

$$S4(G3), G4 \rightarrow N4$$

$$S5(G3), G4 \rightarrow N5$$

where G4 is a further service group, which N4 and N5 have seen in their vicinities' and thus consequently passed this information on to N3. In this case two interesting possible scenarios arise, depending on the specific hierarchical ordering:

1. G4 is a subclass of G1. As a practical example G1 could be $\langle Printer \rangle$ and G4 $\langle Laser \rangle$ in figure 2.1. In this case the request would also be forwarded to N4 and N5. However the service descriptions might not match, resulting in two redundantly forwarded messages.

2. G1 is a subclass of G4. However the request only lists G1 as *request group* and thus the message is not passed on to N4 and N5 albeit they might satisfy the request.

So it becomes clear that the service hierarchy and the level of abstraction chosen when formulating a request have an impact on GSD's protocol behavior regarding (1) discovery efficiency but also (2) the number of generated messages and thus scalability. The following section outlines this impact and points out two possible scalability problems which are subsequently closer examined.

2.3 Potential Scalability Problems

[9] also examines the scalability of GSD regarding various combinations of advertisement diameter, advertisement frequency, ... and derives several recommended values for these parameters. For example, simulation results in [9] suggest that an advertisement diameter of 1 is most efficient in regard to network load. Furthermore it compares GSD to a fictional protocol which transmits requests purely by network-wide broadcast. The authors of [9] establish three main formulas to formalize the network load of GSD and a broadcast based solution.

The total number of messages needed in a fictional, purely broadcast based protocol M_{BCast} is given by

$$M_{BCast} = R_f * m * T * b$$

where R_f is the request frequency, m is the number of messages generated for a single request across the whole network, T is the observed time interval and b is the number of nodes which generate requests.

In the same way the network load M_{GSD} generated by GSD is defined as

$$M_{GSD} = n * N * A_f * T + p * R_f * T * b$$

wherein N is the number of nodes in the network. All nodes are assumed to send out advertisements at rate A_f. n denotes the number of messages generated due to a single advertisement from a node, which is determined by advertisement diameter and node density in the network. Thus the first part of the summation represents the load generated by actively disseminating information in the network via the advertisement component of GSD. The second part expresses the number of messages generated by request based discovery and the only new variable introduced is p which denotes the average number of protocol messages generated throughout the network due to a single request. An important observation in [9] is that generally $p \leq m$ due to the selective forwarding functionality in GSD, however in the worst case $p = m$ since the request is being broadcast across the whole network.

So as a result in order for GSD to be more efficient than a purely broadcast based protocol the following equation has to be satisfied:

$$R_f * (m - p) * b \geq n * N * A_f$$

The simulation results in [9] suggest that GSD outperforms a broadcast based protocol with increasing network size. However they do not clarify the impact of the hierarchical grouping mentioned before but only note the worst

case scenario in which GSD has to fall back to only broadcast based discovery – the case in which it is unable to perform selective forwarding for a request at all. This can be interpreted as a **too specific discovery request** for a service group too far at the bottom in the service hierarchy tree.

But there is also a second case in which GSD performance degrades towards a broadcast based solution as well and p goes towards m: When the **discovery request is too general** and falsely forwarded to nodes which host no matching service. A practical example would be to use $\langle Service \rangle$ (the root in the service hierarchy tree) as group in a request message which searches for a laser printer. Obviously a node receiving such a request would "selectively" forward it to every node it ever received an advertisement from because every other group in the service hierarchy qualifies as $\langle Service \rangle$ as well. However a large part of these messages is clearly redundant and only increases the network load.

The important key insight is that both cases which have a negative impact on GSD's scalability stem from the same underlying reason: Both are a result of the abstraction chosen for the request group in combination with the service hierarchy. Thus the group information used in the service description directly has an impact on the scalability of GSD. In each of the two cases – a too specific query or a too general query – the protocol performance degrades. On one hand by falling back to broadcasting, on the other hand by forwarding redundant messages.

[9] does not state what abstraction was chosen for requests to obtain the simulation results and neither does the paper evaluate the impact of the abstraction level on the scalability of the protocol. So the next chapters try to:

- Identify closer what requirements should be met for good scalability in regard to wireless ad hoc networks.
- Shape a theoretical overview of this abstraction problem.
- Examine the possible consequences and their severity in regard to the protocols scalability by simulation.
- Make suggestions that try to choose the right level of abstraction based on the simulation results.

Chapter 3

Theoretical Background

In this chapter we briefly outline the theoretical background associated with GSD's potential scalability problem when choosing the wrong abstraction for service requests. First of all, we try to clarify the key issues regarding what resources should be conserved ideally and how these resources are consumed. Building on this outline we summarize the desired behavior of a network protocol from a theoretical point of view and point out how and why GSD fails to achieve this behavior.

Then we try to model the problem at hand, that is choosing the correct service abstraction in order to minimize resource consumption throughout the network, in a way that is in line with the requirements defined before. For this, we abstract the most important behavior of GSD in a graph theoretic fashion. We show that different choices in regard to the request abstraction result in different *communication graphs* for the request message.

Based on this we further point out that the problem at hand bears great similarities with well known challenges stemming from the area of *topology control in wireless networks* and more precisely the problem of finding the so-called *critical transmit range* to satisfy certain network wide properties. We argue that finding the right level of abstraction in the service request is isomorphic to certain instances of this problem and thus existing approximation methods could ultimately be reused.

3.1 Optimization Goals and Resource Costs in MANETs

We first want to clarify scalability and the specific costs associated with certain operations in a wireless network. In the constrained environments of manets the following resources should be carefully preserved:

1. Network capacity
2. Energy

Obviously these two resources are correlated. Thus keeping traffic at the minimum ultimately also conserves energy and vice versa.

So a basic insight is that the only way to reduce energy consumption and increase capacity is to minimize the communication between nodes in the network, while still maintaining the desired level of connectivity. The communication links between nodes in the network can be expressed as a *communication graph*. Minimizing the communication graph is analogous to minimizing the needed energy and bandwidth.

It needs to be noted that this optimization tries to satisfy a network wide condition by adjusting the behavior of single nodes, which in a practical scenario will usually result in approximate solutions only, since an individual node hardly has complete information about the state of the network. [14] elaborates on this subject, and furthermore presents a protocol to achieve this goal by adjusting the transmit power of nodes, in order to minimize the resulting communication graph.

This reasoning in general applies to sending any kind of data and the communication graph resulting from this operation, and in particular also to the propagation of service discovery requests throughout the network, since this can be directly influenced by choosing the *service groups* to be included. Section 3.2 explains how sending a service request with a specific level of abstraction directly translates to a specific communication graph wrt. to this abstraction level, along which the request is propagated throughout the network.

Based on this observations, it becomes clear that broadcasting is a suboptimal way to propagate a service request in the network, since it will usually not result in the minimal communication graph. However even more practical reasons, why broadcasting is not a good solution, can be pointed out.

Sending broadcasts in an ad hoc network is a nontrivial task because of the problem of spatial reuse. For example, using the common 802.11 wireless standard, the following problems can be pointed out in association with broadcasting: It can happen that adjacent nodes try to forward a broadcast message at the same time which might often result in:

1. Bandwidth contention, because rebroadcasting hosts are often physically close to each other.

2. Collisions, because there are no RTS/CTS (Request to Send / Clear to Send) control frames used for broadcasting, in contrast to uni-casting. Furthermore the timing of broadcasts on nodes next to each other is strongly correlated.

3. Transmission of redundant messages, since the transmission ranges of nodes often overlap. Thus nodes potentially receive the same message from multiple sources.

Lost messages, caused by these problems, go nearly unnoticed from the sender's point of view, as broadcasts are not confirmed by an ACK. The sum of these issues is well known as the *broadcast storm problem*, and thoroughly discussed, for example in [15].

Performing broadcasts efficiently thus turns out to be non-trivial. Constructing an energy efficient broadcast tree for a certain node is known to be a

NP-hard problem (see [16], [17]). All in all it can be stated that broadcasting should be avoided as much as possible due to the reasons presented above.

3.2 Modelling Request Abstraction as Communication Graphs

As argued previously sending a service request can be modeled as a communication graph. Traditionally such a network graph illustrates the topology of a wireless network. Optimizing this graph and the associated connectivity is a central subject of research in the area of topology control. Formally a communication graph can be defined in the following way (see [18], which also contains a extensive survey on topology control in general)::

> Given a set N of nodes, with $|N| = n$, each node $u \in N$ has a distinct position $L(u)$ in a bounded region R. More precisely $L : N \rightarrow R$ is a function that maps every node in the network to its physical location within R. In combination L and N define a network $M = (N, L)$. Furthermore it is necessary to define a range assignment RA, which is a function and assigns to ever $u \in N$ a value $RA(u)$, representing a certain transmit range.
>
> In turn the communication graph, corresponding to a network $M = (N, L)$ and a range assignment RA is defined as the directed graph $G = (N, E)$,where E denotes the set of edges in the graph–the wireless links in the topology. The directed edge $[u, v]$ is an element of E if and only if $RA(u) >= \delta(L(u), L(v))$, where $\delta(L(u), L(v))$ is the Euclidean distance between u and v. In other words, the directed wireless link (u, v) exists if and only if nodes u and v are at a distance of at most $RA(u)$. In this case v is said to be a one-hop neighbor. A link is said to be bidirectional at time t if $(u, v) \in E$ and $(v, u) \in E$.

Note that the above definition does not account for mobility and furthermore does not place specific restrictions on R. However, it is trivial to extend the above definition to take mobility into account by augmenting it with a variable t in order to express the location $L(u, t)$ of some node u at a certain time, and thus arrive at a communication graph G_t. This extension is not relevant for our purposes in the scope of this thesis. Communication graphs defined in this way correspond directly to the point graph model of wireless networks, presented in more detail in [19].

We will now illustrate how different layers of abstraction in a service discovery request in GSD also result in a specific communication graph G_{GSD}, which is obviously a sub-graph of G. As an example we consider the service hierarchy in Figure 3.1 and the basic network topology in Figure 3.2(a). We consider three specific abstractions in a search for service S1, originating at the request source RS: $S1(G4)$, $S1(G3)$ and $S1(G1)$.

The purpose of this example is not to cover the complete life-cycle of a service, its advertisement, ... but rather to highlight the effect of different service

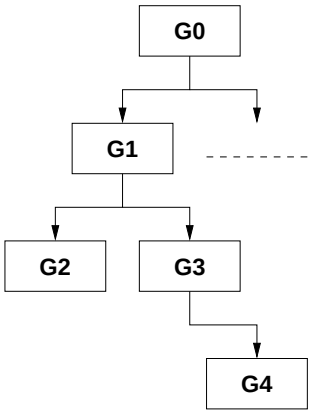


Figure 3.1: Example Hierarchy

groups in a discovery request on the resulting communication behavior. For the sake of explanation and in order to simplify the scenario we further assume that service advertisements have already been distributed throughout the network and therefor enough information is present in caches at the nodes, which illustrates the idealized case.

For this reason no broadcasts occur. Apart from the specific problems in relation with broadcast, mentioned in the previous section, they almost never result in minimal communication graphs for a request. In theory, however, they could be modeled in the same way.

While the concrete advertisement strategy can have significant influence on the performance of the service discovery process it simply results in a sub-optimal communication graph for a specific request. Sub-optimal in the sense that:

- Less information about services is available in the network and thus less potential propagation paths for a request are available, which results in less efficient communication since broadcasts might be necessary.
- Certain service providers might not be discoverable at all.

From Figure 3.2 it becomes obvious how increasing the abstraction results in different communication graphs for the GSD request propagation, which are all sub-graphs of G. Note how certain nodes are elements of the resulting graph even though they are on a redundant path. And even more so, increasing the abstraction level to $G2$ suddenly includes N2 in the graph, which corresponds to a erroneous selective forward, since N2 should not be part of the minimal graph. Based on this graph model, two minimization scenarios for the graph are conceptually possible:

1. Minimize the propagation graph such that **at least one** matching service instance is found.
2. Minimize the propagation graph such that **all reachable** matching service instances are found. This scenario is potentially relevant to allow a

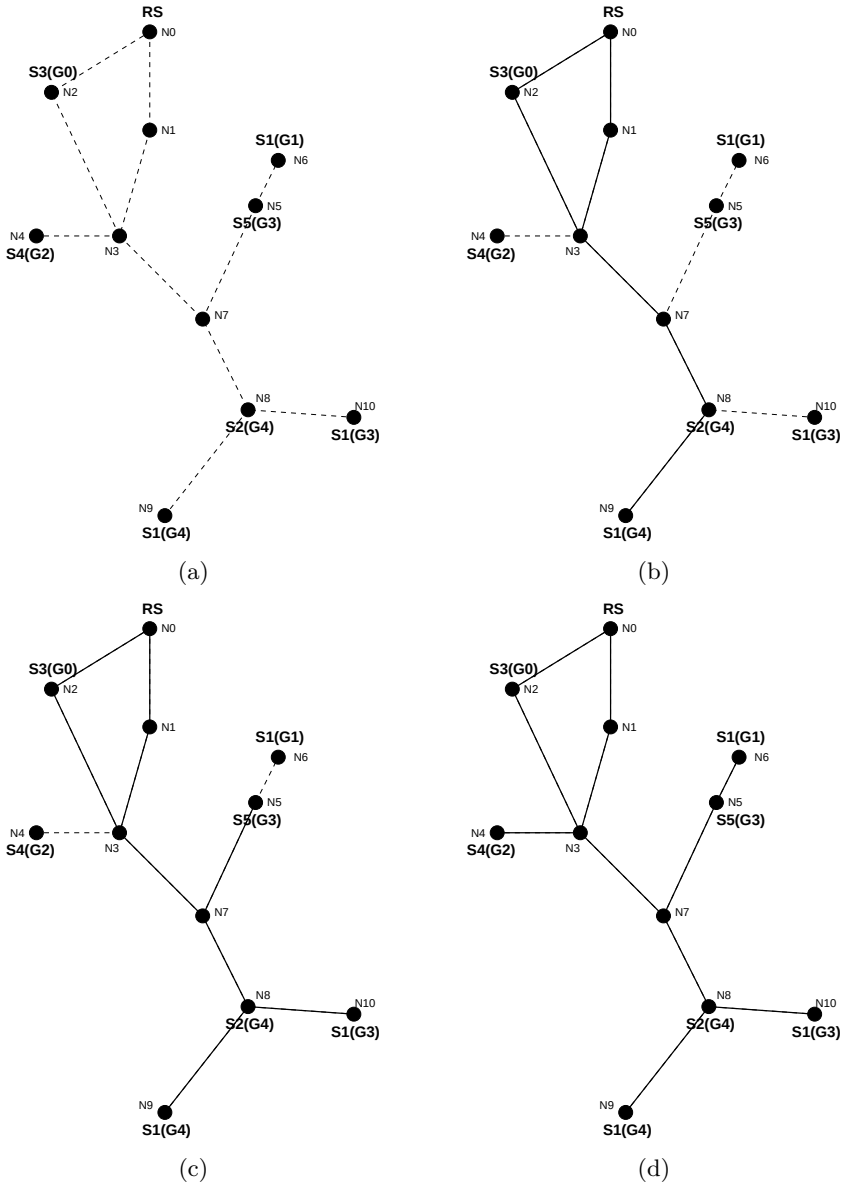


Figure 3.2: The original communication graph G (dashed lines) in (a) and G_{GSD} resulting from a request for (b) $S1(G4)$, (c) $S1(G3)$, (d) $S1(G1)$.

more fine grained selection by the client based on additional parameters (provenance, additional context information, ...).

What can essentially be seen from this simple example is that:

- Physical network topology is important for GSD's performance. Choosing the correct abstraction becomes more difficult in networks with high number of average neighbor nodes and high network density. In a dense network the minimal communication graph will likely include more redundant nodes, so the resulting communication will be suboptimal.
- The service hierarchy and distribution of services in the network are important for GSD's performance. The reason for this is that the communication graph is directly influenced by the distribution of services in the network.

Based on these observations it becomes possible to give a more detailed analysis of GSD's behavior. As pointed out in Section 2.3 the total load generated by GSD can be directly attributed to (i) packets generated for service advertisement and (ii) packets generated for service discovery requests. Furthermore GSD also indirectly causes (iii) reply packets depending on how many service providers a request reaches, however the concrete behavior of that is also tied to the underlying routing protocol used.

Derived from the original formula in Section 2.3 we will now single out the deciding performance factors for GSD's performance. For this we exclude request and advertisement frequency and furthermore as well disregard the observed time interval in the original formula since we are interested in the choosing the right level of abstraction for an individual operation at this point. The remaining key factors are the following:

1. The *average* number a of packets generated for a single service discovery request.
2. The *average* number p of packets generated for a single advertisement.

Since service abstraction does not influence it, p can be roughly regarded as constant and only dependent on the node density in the network. a is more interesting to optimize as both network and abstraction can influence it.

Based on the underlying assumptions of modelling the network as a communication graph several possibilities arise: First of all, it is possible to calculate the likelihood of finding a service match, e.g. in one hop distance, by using the standard formula for a hyper-geometric distribution as following:

$$Chance(N, M, n, k) = \sum_{k=0}^{n} k \frac{\binom{M}{k}\binom{N-M}{n-k}}{\binom{N}{n}}$$

Where N is the total number of services,, M is the number of suitable candidate services (throughout the network) according to the service hierarchy and

abstraction (and can thus be adjusted by increasing abstraction), n is the average number of neighbor nodes, and k is the number of actual services to be discovered. According to the two basic problems described before, k will usually be 1 or M. Combined N and M express the probability that any node has a suitable service or in other words how common a specific service is in the network. Increasing n leads to the according probabilities of finding services in multiple hops.

In order to actually use this model, two properties are needed: First and foremost the availability percentage of a service in a network. For this taking the service hierarchy into account is required because certain services subsume others in their functionality. And secondly knowledge about the network topology, and more precisely its density / the average neighbor count of a node is required. Based on this theoretical model it becomes possible to determine the required abstraction level (and thus also M) to fulfill one of the two optimization goals mentioned above and the chance to find a suitable service is satisfactory, depending on application context.

If required, N can be approximated by (i) observing the number of nodes in relation to the numbers of services in the neighborhood of a node and extrapolating from this, (ii) setting the maximum number of hops a request is allowed to travel, and thus artificially limiting the "size" of the network.

As mentioned, other existing solutions can be applied to solve this problem or at least approximate a good enough solution. A more detailed mathematical analysis along with optimizations is shown in [20], albeit modeled in a slightly different way. As pointed out there exist applicable approaches in the area of topology control which try to optimize the communication graph based on probabilistic information, for example [21] and especially [22], [23].

Chapter 4

Simulation

4.1 Experimental Model

In order to measure the impact of a chosen abstraction level in the service hierarchy on the scalability of GSD and to derive practical results, we implemented GSD's relevant protocol parts in the NS-2 network simulator [24]. More specifically we extended version 2.29 of NS-2 with GSD as an additional application layer protocol. Note that it would also have been possible to implement GSD at the routing layer. However, this has no impact on the simulation results in the scope of this thesis as we are only measuring the network load caused by GSD and not additional traffic caused by the underlying routing protocol.

This section documents the experimental model chosen and the exact purpose of the simulation and its derived results. Thus there are four main aspects covered here:

1. The intended measurement objectives
2. The chosen service hierarchy
3. General scenario setup
4. Individual Node settings

In general we tried to stick to the settings chosen in [9], as close as possible, in order to obtain comparable results. In case where different values were used, either because the original choice is unclear or not documented or because the settings are unsuitable to obtain useful results for the scope of this thesis, we explicitly point out the differences.

4.1.1 Measurement Objectives

As mentioned the purpose of this simulation is to measure the impact of a chosen abstraction level in a service request in regard to the generated network traffic, or more particularly:

> What is the right level of abstraction in order to be specific enough to prevent redundant selective forwards, as well as being

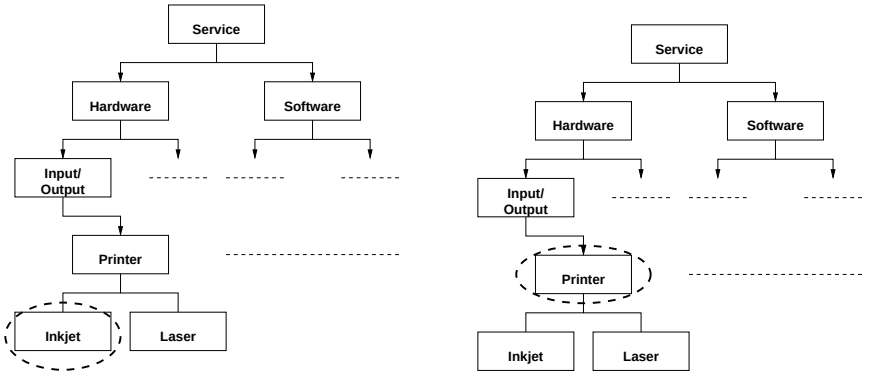


Figure 4.1: Searching (a) directly for a resource or (b) going up one level in the hierarchy

> general enough so nodes do not have to fall back to broadcasting, while at the same time maintaining a high level of discovery quality.

Figure 4.1 illustrates this in a practical example: When searching for an inkjet printer it is possible to either directly specify this by setting the request group to $\langle Inkjet \rangle$. However it might be advantageous to use the more general request group $\langle Printer \rangle$ instead, in order to prevent broadcasts and receive more replies to the service request.

In order to quantify this impact several key numbers where identified and kept track of. First of all we did rough measurements of events triggered at nodes in the network. These include the events shown and explained in Table 4.1.

Advertisements	The number of times a node advertised services.
Requests	The number of active discovery attempts made in the network.
Selective forwards	How often a request was passed on selectively.
Broadcasts	How often a fallback to broadcasting was necessary.
Successful requests	The number of successful requests. Cases in which a match could be found at one or more nodes. Note that one request can result in more than one matching service.

Table 4.1: Events measured.

Each of these events marks a certain important key-point in the discovery protocol. However in order to gain more fine grained data, which closer reflects the real load generated in the network, we also measured data at the packet processing level. The gathered key numbers are shown in Table 4.2.

The only number in this listing that really demands explanation is the last one. It exactly denotes to how many nodes a request was selectively forwarded

Total messages processed
Advertisements processed
Requests processed
Messages dropped
Advertisements dropped
Overall requests dropped
Selective forwards dropped
Broadcasted forwards dropped
Requests received as broadcast
Requests received as selective forward
Average number of targets in selective forward

Table 4.2: Packet processing information collected.

on average during the complete simulation run. This expresses how much more efficient selective forwarding in a certain environment is compared to ordinary broadcast. A high result can be interpreted as a too high abstraction and thus also too generic forwarding.

4.1.2 Service Hierarchy

Another important point is the service hierarchy chosen for the simulation. The authors of [9] only mention that

> "For the purpose of the simulation, we used representative services S0 to S99 to represent actual services and groups G1 to G10 to represent service groups with G10 being equivalent to the parent service group called "Service" at the root of our hierarchical tree."

However, it is clear that the shape and characteristics of the service hierarchy have an effect on the request forwarding and thus on protocol behavior. More specifically because going up one level of abstraction to another service category suddenly includes the whole sub-tree beneath this category in this search.

It is possible to more formally quantify the impact of the service hierarchy on this growth. For this the hierarchy can be regarded as a rooted tree, and the factor determining the impact of increasing or decreasing the abstraction chosen is the average difference of the *size* of a node n and its parent p. The *size* of a node is the total number of its descendants including itself, so the number of nodes in the sub-tree of which the respective node is the root.

For binary trees this growth can be expressed in relation to the level l of a service in the hierarchy, starting with $l = 1$ for services at the leafs, and the number of abstraction steps to "go up" a in the hierarchy for a request:

$$gr(l, a) = 2^{l+a} - 2^l$$

This is the average growth for the case of a *full* and *balanced* binary tree. An example for this growth is shown in Figure 4.2.

For a service hierarchy forming a more general tree, and more precisely an unbalanced tree, this impact becomes hard to measure. Still the average

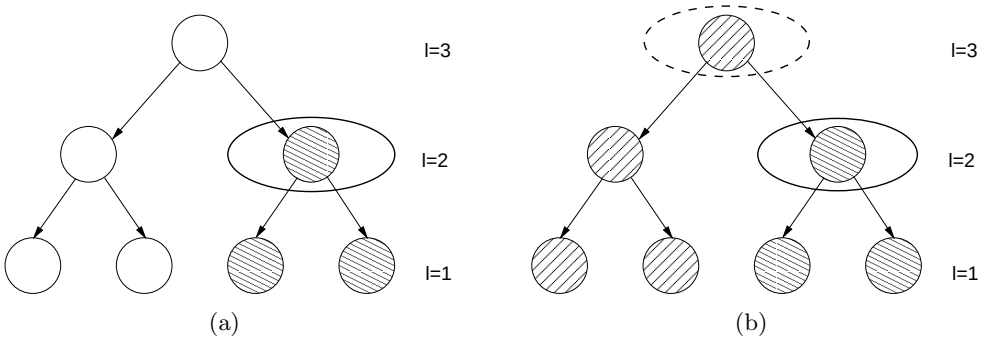


Figure 4.2: Going up one level of abstraction ($a = 1$) from a request for a service at the second level of a service hierarchy ($l = 2$) implicitly adds a total of 4 service groups to the request groups in a balanced binary tree. For a more general tree the difference between the *sizes* of the respective nodes expresses the growth.

number of service groups, implicitly added to a query when going up a level in the tree, is a significant characteristic of a service hierarchy. The reason for this is that this number gives a metric of the "cost" of increasing the service abstraction in a request in terms of caused network load (in combination with a specific network topology).

In order to gain results of practical value we tried to chose a real world example for our simulation. A big practical example for human organized hierarchies are the Big-8 newsgroup hierarchies on Usenet. The Big-8 is a set of discussion groups on Usenet covering a wide variety of topics, organized into eight distinct hierarchies. Albeit the groups are inherently international, discussion is mostly in English because of the newsgroups' historical origin.[1]

The separation into different hierarchies on Usenet is done through a naming convention. That simply means that all groups starting with a specific prefix are part of a common hierarchy and different levels in the hierarchy are separated using a dot. The most significant part of a news group's name is given first and thus also determines the specific hierarchy to which hierarchy a group belongs.

The Big-8 hierarchies start with the following prefixes:

- `comp.*` Computer related topics.
- `news.*` Groups related to the administration of the Big-8 and Usenet in general.
- `sci.*` Science related groups.
- `humanities.*` Groups covering humanities and associated topics.
- `rec.*` Hobbies, sports, games, recreation, ...
- `soc.*` Social issues, socializing.

[1] `http://www.big-8.org/dokuwiki/doku.php?id=history:big-8`

- `talk.*` Wide variety of discussions, mostly politics related.
- `misc.*` Anything that does not fit well into any of the other seven hierarchies.

The authoritative list of groups in these hierarchies is contained in a text file called "Checkgroups" [2], which is posted weekly to the `news.announce.newsgroups` group. For our simulation purposes we used the Checkgroups file posted in the first week of November 2006.

For the sake of our simulation we have selected the `comp.*` hierarchy. In total the `comp.*` hierarchy contains 1814 newsgroups, which result in 2021 distinct service groups in the service tree. This is due to certain groups, as for example `comp.databases.postgresql.admin, comp.databases.postgresql.doc`, that generate "abstract service groups" for which no "service instance" exists – there is no `comp.databases.postgresql` newsgroup. Thus we consider such service groups in the discovery process but do instantiate them on nodes in our simulation either. Practically that means that it is possible to use these groups in a discovery request (as a means of abstraction) but that no actual service corresponding to it is available in the network.

The average depth of the service tree for the `comp.*` hierarchy is 3.67 levels and the average number of groups which are added implicitly to a query when going up one level in the tree is 246.29. It should be noted that the hierarchy is very "shallow" , in the sense that the root node `comp` has an above average child count, and thus increasing the abstraction to include the whole tree in a search suddenly has a huge impact.

We now illustrate a practical discovery request in an example using a subset of the chosen `comp.*` hierarchy, depicted in Figure 4.1.2. We then consider a discover request that is looking for a service S1, which provides information about programming algorithms (its service group is `comp.games.development.programming.algorithms`), and subsequently increase from none abstraction at all, to going up 1 and respectively 2 levels in the service hierarchy. We abbreviate lengthy service groups by their starting letters, i.e.`comp.games.development.programming.algorithms`) by `CGDPA`, or `comp.games.solitair` by `CGS`.

These three scenarios are depicted in Figure 4.4 on page 25. All of them assume the same topology and cache entries, based on previously propagated advertisements. We use the same notation as in Section 2.2, i.e. `[S1(CGPD)]` denotes that the node offers a service S1 belonging to service group `CGPD`, `[CGPD -> N2]` denotes that the node is capable of forwarding a request for a service belonging to group `CGPD` via its neighbor N2 (because it received suitable service information from N2), and finally `[S(CGPD)->N3]` means that the node has the complete service information for its neighbor N3 (because it received a direct advertisement).

In the first scenario, as seen in Figure 4.4(a), no abstraction is used at all. The node RS (the request source) is sending its request, which is in turn propagated to N1, N4 and N5, where a match is found. As noted previously

[2] `http://www.big-8.org/dokuwiki/doku.php?id=faqs:big-8_list`

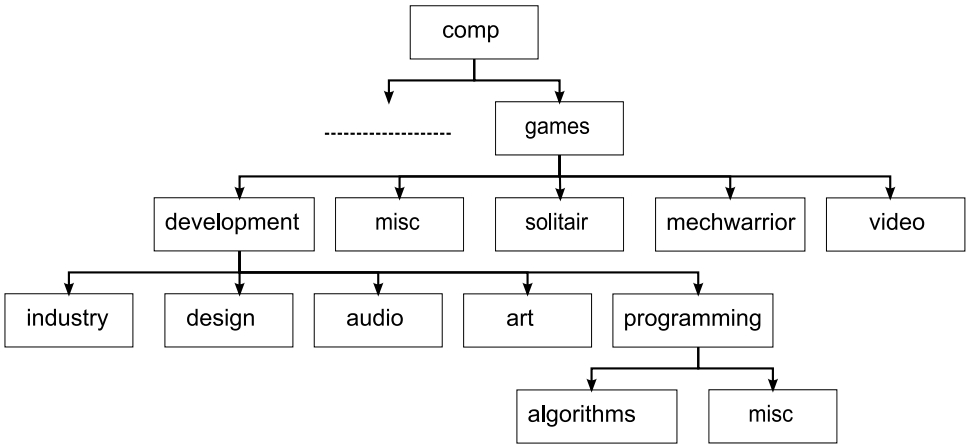


Figure 4.3: A branch of the `comp.*` hierarchy.

the request does not have to find the original service provider but only a node which has the complete service information in its cache. The request cannot be passed on to N7 and N2 because the service group in the request is too specific.

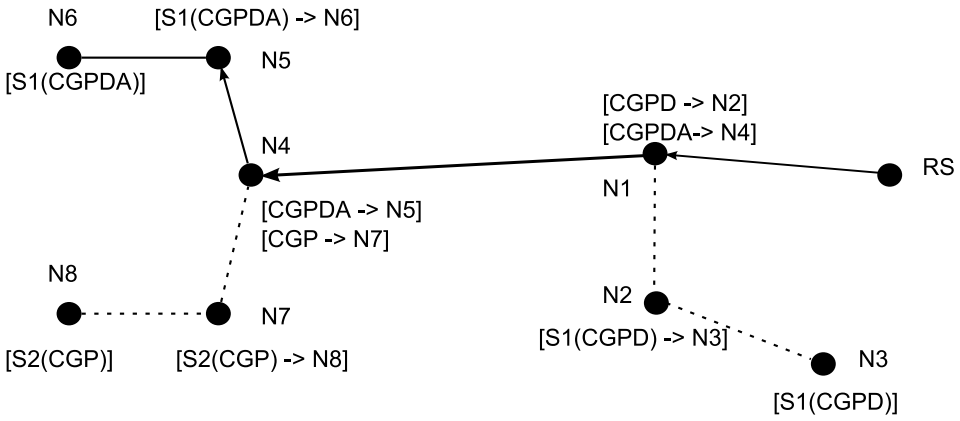
Increasing abstraction, as shown in Figure 4.4(b) makes it possible to also forward to N2 from N1 and to find another suitable match. Increasing abstraction even further (see Figure 4.4(c)) also enables propagation to node N7. However, despite the correct service group no service match is found – the increase in abstraction only resulted in a false forward and a redundant network message.

4.1.3 Simulation Setup

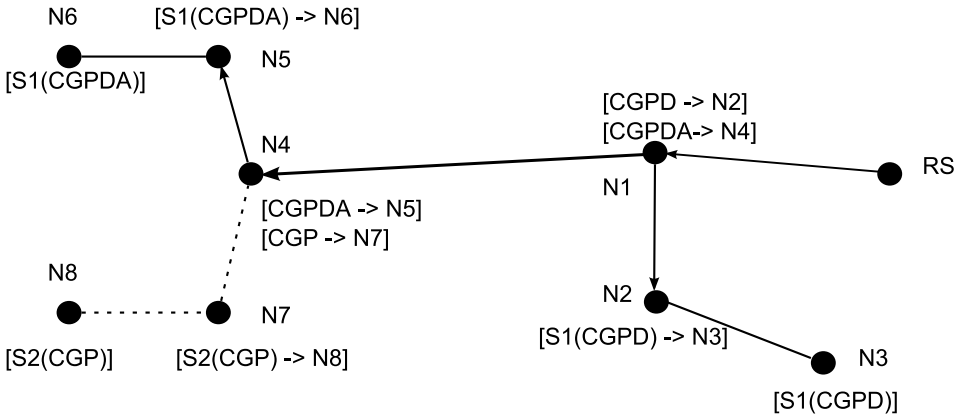
Regarding the general simulation setup we tried to stick to the original paper as much as possible, while increasing the hierarchy and network size to bigger dimensions. Our simulation was carried out as a steady-state simulation, in order to acquire data that realistically reflects the behavior in a deployed network. This poses the problem of an initialization bias since the protocol starts with zero knowledge about services in the network, which basically means that service caches at all nodes are empty. In order to minimize this bias we implemented two changes in our simulation in contrast to [9]:

- We introduced a warm-up period to allow the protocol to reach a reasonable state by distributing service advertisements, after which we stop the advertisement process.
- And more importantly, we increased the total simulation time to collect more data points in order to dilute the effect of the initial bias in the measurements.

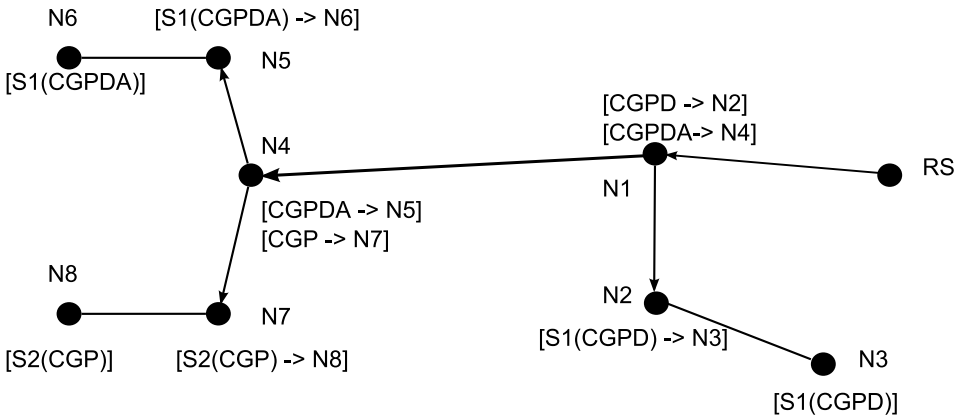
It is only after this warm-up phase that we start to generate requests and collect relevant data. More details regarding this particular type of simulation setup are presented in [25] as a case study, and a more formal and comprehensive treatment is available in [26].



(a)



(b)



(c)

Figure 4.4: An example service discovery request for a service S1 belonging to the service group `CGPDA`, using (a) no abstraction, (b) an abstraction level of 1, and (c) an abstraction level of 5. Scenario (c) does not add further useful services and results in a false forward.

Overall we have chosen three main simulation scenarios. The main difference in regard to the three scenarios was to increase the scale of the simulation compared to the settings in [9] considerably. The different parameter values for these scenarios, including derived values, which highlight why these changes are needed, are presented in the following part of the chapter. For each of the chosen network sizes we examined several different levels of abstraction in the service request in regard to their scalability by analyzing the collected output described in section 4.1.1. The most important of these parameters are presented in Table 4.3 on page 27 and for each case we performed several runs until a 90% confidence interval was reached (but at least 10 runs), and the collected results where averaged. For each of the runs we used a new sub-stream of the NS-2 random number generator class `RNG`, of which each has a periodic sequence of $2^{31} - 2$. Thus we ensure independent simulation runs.

From the basic settings it is possible to derive a number of further key figures which aggregate multiple input parameters. We shortly present each of those derived values in Table 4.4 on page 28 along with the used formula. The main purpose of these values is to facilitate an easier comparison of the characteristics of different simulation scenarios. The calculated results of our simulation setup are given in Table 4.5 on page 29 along with the values corresponding to the settings from [9] (denoted by $\star$) for the sake of comparison.

The nodes are initially distributed randomly in the flat simulation area of varying size. Such a simple scenario is in line with the original settings for the simulations in [9]. Furthermore we have deliberately chosen a topology without any obstacles and the simplest radio propagation model available in NS-2 (FreeSpace), because we are mainly interested in the impact of a certain abstraction for service requests and want to rule out other factors influencing the outcome of the simulation.

For this reason we also conducted the simulations without any node mobility, in addition to runs which stick with the original mobility settings from [9], which is a random way-point model. In order to generate the required node-movement files we used the `setdest`[3] tool included in NS-2. The setdest program generates node-movement files using the random way-point algorithm according to supplied command line arguments. These scenario files, which describe all the node movement for a simulation run, can then easily be included in ordinary simulation scripts.

Furthermore we have chosen the same availability percentage (10%) as the original paper. This means that every service is available on 10% of the nodes in the simulation and in turn also determines how many services each node has to host. The according distribution of services among the nodes was performed randomly. In combination with the initial simulation values (see Table 4.5) another issue becomes clear and illustrates the need to increase the simulation size: Using the original settings a node has an average neighbor count of 14.14 nodes. Given an availability percentage of 10%, a node sending a request has a change of 77% to find a match after the very first hop – this also takes the services a

[3]Originally contributed by CMU, and available in the `indep-utils/cmu-scen-gen/` subdirectory of the NS-2 distribution, in a slightly modified version.

Parameter	Value
General Settings	
Simulation Area (w,h)	(400 x 400m), (800 x 800m)
Simulation Time	1000s
Warm-up Period	100s
Number of Nodes	200, 400, 600
Wireless Transmission Range	30m
Random Broadcast Jitter	0-10ms
Advertisement Interval	15
Advertisement Diameter	1
Service Availability	10%
Request Frequency	7.5s
Request Hop Count	30
Request Abstraction	0,1,2,3
NS-2 Specific Settings	
Radio Propagation Model	FreeSpace
Antenna Model	OmniAntenna
MAC Type	802_11
Network Interface Type	WirelessPhy
Channel Type	WirelessChannel
Interface Queue Type	DropTail//PrioQueue
Link Layer Type	LL
Random Node Motion	Off
Mobility	None, Random way-point with 2m/s and 5s pause time

Table 4.3: Simulation parameters.

node has available locally (see Section 3.2) into account. We decreased the network density (by adjusting the simulation area and node number) drastically in order to lower this chance. The resulting likelihood of finding a suitable service on the first hop, again derived from the average neighbor count and general service availability as above, amounts to to 31%, 24% and 17% respectively.

Requests and advertisements are generated at specified time intervals. Whereas advertisements are generated by each node, requests are generated by only one random node in the network, and the service group in the search is also selected randomly from the service hierarchy. In [9] the advertisements to requests ratio A_f/R_f varied from 0.25 to 2.0 by adjusting the request frequency R_f from 1 requests/minute to 8 requests/minute. We settled to use only the shortest value of 7.5s. However, since we carry our simulation out in a slightly different way – as a steady state simulation – this does not have any significant result. The main purpose of generating a lot of requests is to collect more data points.

One further aspect that should be noted regarding our implementation in NS-2 is that it is not possible to directly set the wireless transmission range. In order to set the communication radius of a wireless node in NS-2, it is necessary to adjust the receiving threshold value for each node's network interface in the

Parameter	Description	Formula
Simulation Area	The area of the simulation.	$w * h$
Node Density	The density of nodes in the area.	$n/(w * h)$
Node Coverage	Area covered by a single node's transmission range.	$\pi * r^2$
Node Footprint	The percentage of the simulation area covered by a single node's transmission.	$(\pi * r^2)/(w * h) * 100$
Maximum Path	The maximum linear path a packet can travel from source to destination.	$\sqrt{(w^2 + h^2)}$
Network Diameter	The minimum number of hops a packet can take along the maximum path from source to destination.	$\sqrt{(}w^2 + h^2)/r$
Neighbor Count	The number of neighbor nodes based on transmission range and simulation area[4].	$(\pi * r^2)/(w * h/n)$

Table 4.4: Derived simulation parameters. r = transmission range, w = width of simulation area, h = height of simulation area, n = number of nodes

simulation. The correct values a radius of 30m were calculated by means of the `threshold` tool included in the NS-2 distribution, which determines the necessary threshold for a given radius and propagation model. For the FreeSpace propagation model, which we used in our simulation, the implementation is based on the Friis transmission equation (see [27] for more details)

$$P\frac{P_r}{P_t} = G_t G_r \left(\frac{\lambda}{4\pi R}\right)^2$$

in which P_r denotes the power at the receiving antenna, P_t denotes the power input at the transmitting antenna, G_t and G_r are the gain at the receiver and transmitter respectively, R is the distance between two nodes (radius) and λ is the wavelength. So it is easy to calculate the required power threshold at a receiver for a certain intended transmit radius.

Scenario	Width	Height	Area	Nodes	Density	Footprint	Maximum Path	Network Diameter	Neighbor Count
(1)	400	400	160000	200	0.0012500	1.77	565.69	18.86	3.53
(2)	800	800	640000	600	0.0009375	0.44	1131.37	37.71	2.65
(3)	800	800	640000	400	0.0006250	0.44	1131.37	37.71	1.77
$\star$	200	200	40000	200	0.00500000	7.07	282.84	9.43	14.14

Table 4.5: Derived simulation parameters for specific scenarios. Node Coverage is not listed, since it only depends on the transmission range r and thus is 2827.43, independent of the particular scenario setup.

4.2 Simulation Results

In this section we present and analyze the data collected in our simulation runs and furthermore try to draw some conclusions based on our observations. First and foremost we want to highlight the effect of different abstraction levels on the network load. We do so by examining:

1. How often a request was passed on as broadcast or selectively.
2. How many requests were received by nodes, in order to get an impression of the resulting overall network load.

Note that selectively forwarding to e.g. three nodes also results in three recorded packets – broadcasts however always only result in one recorded packet. This precisely reflects the link layer behavior of the respective operations. Thus we also illustrate the average number of nodes, to which a packet was selectively forward, in Figure 4.9.

As shown in Figure 4.5, varying the abstraction of the service request has little impact on the number of selective forwards along the propagation path of a request. The reason for this is that service information of a rather shallow service hierarchy is distributed very rapidly across the network. However, increasing the abstraction however dramatically lowers the need to broadcast, while not increasing the abstraction leads to over-proportionally many discarded broadcast packets. Just moving up **one** level in the service hierarchy for a request reduces the amount of broadcast operations by nearly 50% which are instead distributed as a proportionally similar amount of selective forward operations.

So the sweet spot in the protocol behavior is reached as soon as broadcasting is replaced by selective forwarding. Increasing the abstraction even further from this point (i.e. in our simulations by more than two levels) has comparatively little further impact in this regard. This can be explained by the "shape" of the underlying service hierarchy, in the sense that going one step

down from i.e. `comp.lang.java.security` to `comp.lang.java` already makes it very likely to find suitable information and thus perform selective forwarding. So, to conclude, from a certain point on, increasing abstraction further does not impact the decision whether a request is selectively forwarded or broadcasted anymore. In turn the benefits of further increasing the abstraction also show an increasingly smaller benefit.

A noticable impact in further incrasing the abstraction of the service groups can be seen in the number of nodes towards a request is selectively forwarded to. This is shown in Figure 4.5, which depicts "events" at nodes where the specific mode (broadcast or selective forward) in which to pass on a request is decided, or in which a request is received via a certain transmission mode. The number of nodes receiving selective forwards is thus higher as the number of nodes sending selective forwards, as expected. The reason for that is simply that in the case of several neighbor nodes, multiple of them will often have suitable information to further propagate the request, and thus are candidate nodes according to the original GSD protocol description.

There also exists a noticable "dent" at an abstraction level of 2 (this means going towards the root of the service hierarchy by two levels from the service group to which the service actually belond) in the simulation scenario with the highest node densitiy. This sudden decrease in sent selective forwards does however not go hand in hand with an increase in broadcasts – there were strictly less packets sent.

This can be explained with (a combination) two obvservations: (i) An increase by two levels might be equal to using the service root as group in the discovery request some cases, and at the very least already covers a large part of groups in the service hierarchy. (ii) Given the high node densitiy it is very likely to sometimes have a suitable service within one hop distance, or at least very close.

This is augmented by the fact that service discovery requests in GSD do not need to directly reach the service provider – it is enough if they reach a node which has received the original service advertisment and thus has the complete service description. So in this case additional selective forwarding was simply not needed. Increasing the service abstraction even further, all of a sudden includes every service in the network. In the chosen service hierarchy this causes a large jump in candidate services. So, the increasing the service abstraction again raises the number of selective fowards, however many of them can in fact be considered to be redundant.

This also shows the impact of the underlying network topology on the selective forwarding mechanism, since in networks with high average neighbor count, a large amount of selectively forwarded can be considered to be redundant, because they are duplicates of packets already seen before. In this sense selective forwarding quickly degenerates to broadcasting. This is also shown by the average number of nodes towards which a request is selectively forwarded. These results show a nearly direct correspondence to the average number of neighbor nodes. Practically this can be interpreted such that a request was often forwarded to **all** of a node's neighbors. This behavior becomes intuitively clear based on the following two observations:

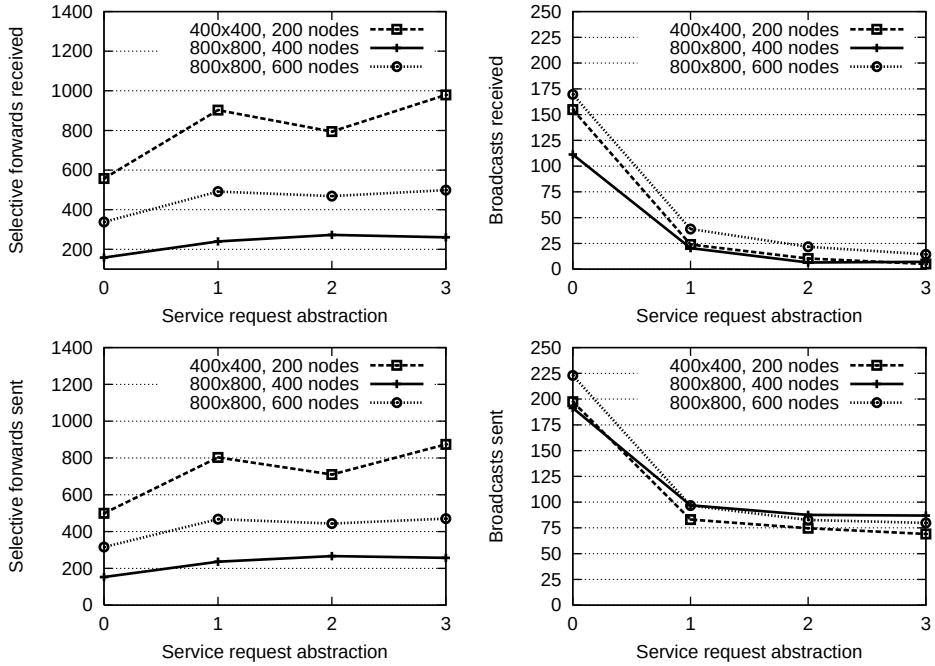


Figure 4.5: Graphs recording sent requests denote the number of times a node decided to selective forward or broadcast a package respectively. Graphs recording received requests depict the number of times a packet was received by a node via one of the two methods.

1. In dense networks, if one neighbor node has information about a service group, it is not unlikely that another neighbor node has the same information as well. Thus nodes with more or less identical information form clusters, based on their wireless connectivity among each other.

2. Directing selective forwards to each of such nodes is unnecessary. It would be enough to forward to one of them, otherwise many selective forwards will just be unnecessary duplicates.

This "cluster behavior" is also reflected in the case where broadcasts are necessary due to the lack of suitable service information. Figure 4.5 shows that less broadcasts clearly were received than sent in all cases. Basically this means that packets were lost due to collisions of multiple broadcasts of neighbor nodes, since broadcast packets are not protected by RTS/CTS handshake mechanism.

So beside the used service hierarchy, the network topology can be identified as another deciding factor for the performance of GSD. To summarize, in networks with high average neighbor count, a large amount of selectively forwarded requests are redundant. We suggest possible improvements to this problem in Section 5.

As already pointed out, from our data it becomes obvious that the average neighbor count in the topology is directly reflected in the success rate of the protocol. This effect on the likelihood of a discovery has already been pointed out in Sections 4.1.3 and 3.2. Thus, topologies with a higher average neighbor

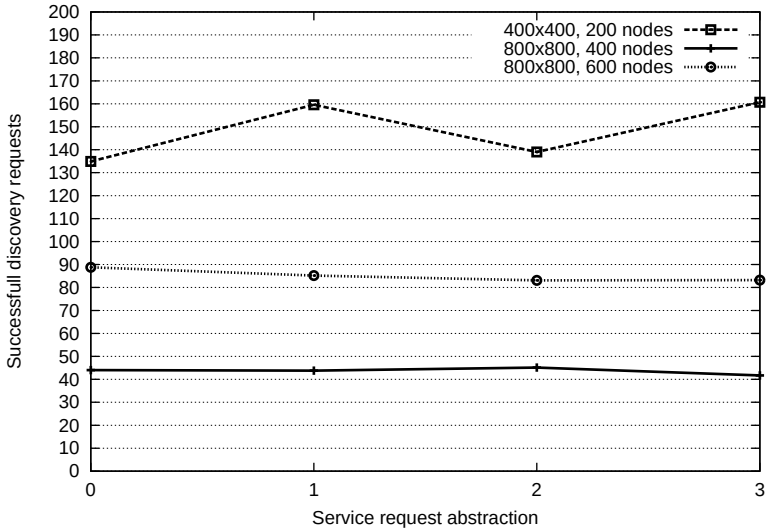


Figure 4.6: The success of requests at different abstractions, showing the number of found service for a total 120 requests sent out.

count per node also have a higher success rate, as seen in Figure 4.6. Furthermore, the chosen request abstraction has comparatively little effect according to our simulations. The reason for this behavior is that at closer inspection the abstraction of a service request does directly influence if it is propagated through the network at all or not. As a last resort GSD will simply fall back to broadcasting and flooding requests through a network, although this involves the usual increase of resource consumption, collisions, lost packets, ... So the chosen abstraction does not directly influence the success rate of the protocol (i.e. if a service is found at all) but it rather determines how **efficiently** and intelligently this is done.

When taking Node mobility into account, the number of selective forwards significantly increases, and thus also emphasizes the previously pointed out problem that too many redundant selective forwards are sent out towards multiple nodes. This is not unexpected since nodes collect more information regarding different services while moving through the network. *Relatively*, the biggest impact of this behavior could be observed in simulation runs in which the request abstraction was not increased at all. In this case having more "direct" service information available clearly has the biggest impact. At the same time, the number of selectively forwarded requests successfully *received* by nodes decreases, see Figure 4.8 – practically speaking this means that many packets are simply lost because the intended recipient has moved away and is not reachable anymore. Not unexpectedly our simulations showed that broadcast is not impacted by node mobility to such a degree. Additionally scenarios with high density, and other parameters that originally favored selective forwarding, degrade more rapidly. Whereas in simulations without mobility selective forwarding often improved efficiency (see Section 3) by a great margin, it suddenly resulted in a lot of dropped packets.

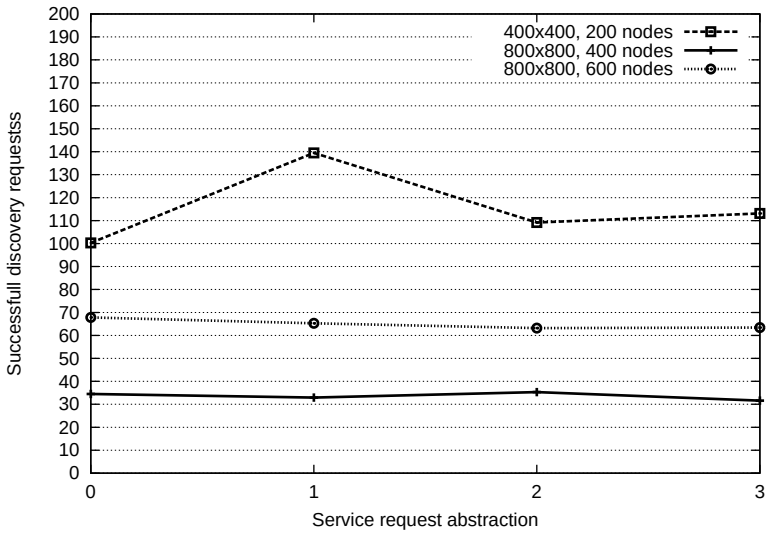


Figure 4.7: Impact of mobility on the success rate of requests, showing the number of found service for a total 120 requests sent out.

A solution for this problem would be to define a specific cache strategy, in which the cache lifetime of an advertisement depends on a function of the sending node's mobility. This would result in less "stale" data in the service cache of a node. An additional possibility would be to employ a cache strategy that attaches a weighting to entries in the service cache, in a way such that mobile nodes are less favored targets for a request. Such an adjustment could go hand in hand with the previously mentioned technique to only chose a certain proportion of neighbors from the service cache, in order to prevent redundant transmission of requests.

So to recapitulate the following main points can be summarized, based on our observations:

- The chosen abstraction has a high impact on the number of necessary broadcasts, for the same discovery rate.
- The chosen request abstraction has little impact on the success rate of the protocol but rather on the network load generated in order to achieve the desired result.
- The chosen request abstraction has impact on the number of packets, which are selectively forwarded, and thus also on the amount of redundant packets generated – this has the potential to let selective forwarding degenerate into broadcast.
- Even increasing abstraction only slightly (e.g. by one to two levels) will lower broadcasts dramatically. In this abstraction has a clear influence on efficiency on the protocol.
- The discovery rate is directly influenced by the underlying network topology.

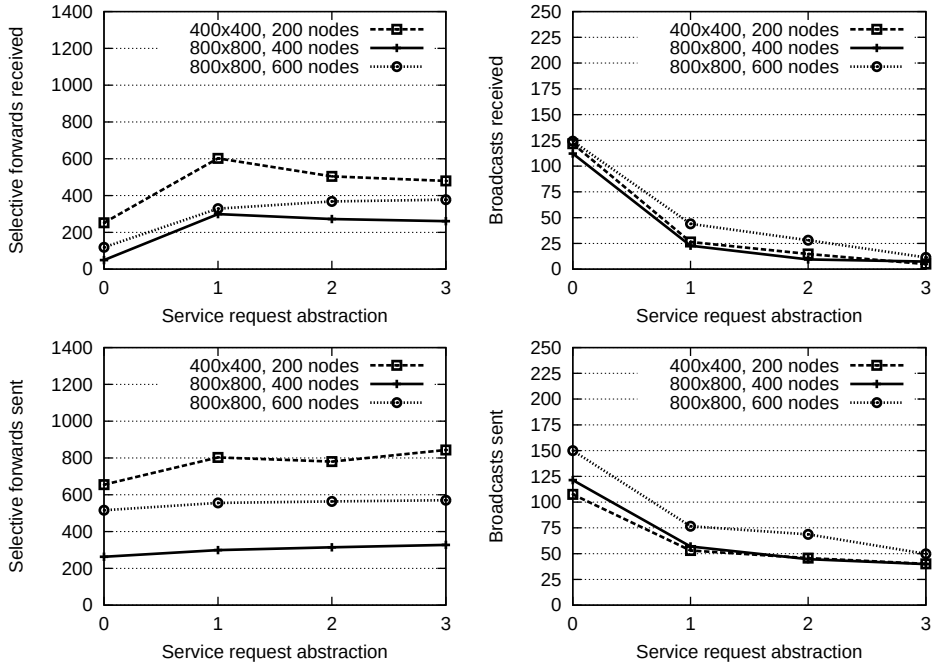


Figure 4.8: Broadcasted and selectively forwarded requests, taking node mobility into account.

- In dense networks selective forwarding quickly degenerates, even when only increasing the abstraction for a request by one to two levels.
- A request should not be forwarded selectively to *all* possible targets, since they likely contain similar cache information (cluster) and thus some of the transmissions are at the end redundant.
- High node mobility has a more severe impact on selective forwarding than broadcasting, since selective forwarding relies on the presence of a specific node in its neighborhood. This situation is made more severe as node mobility in combination with GSD's caching strategy encourages selective forwarding, while at the same time decreasing its success rate. As mentioned previously, selectively forwarding to stable "relay" nodes, which have shown to stay in the vicinity, can be used as a heuristic to construct stable paths towards a service provider.
- Caching needs to be adjusted in order to account not only for device capabilities, but also to consider the mobility of an advertisment's sender.

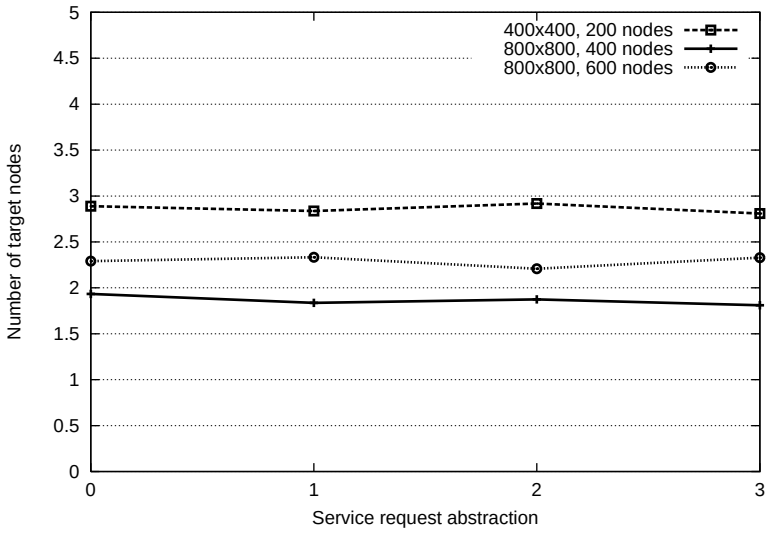


Figure 4.9: Average number of nodes for selective forwarding.

Chapter 5

Summary &Conclusion

In this thesis have we briefly outlined the motivations and challenges faced by service discovery protocols in MANETs. We have subsequently presented GSD, a protocol which tries to incorporate rich semantic service description's and use them to aid in the discovery process. We have shown a way to theoretically model the behavior of GSD and furthermore that parallels to other established areas of research exist, from which possibly existing optimization techniques could be adopted.

Based on our simulations it is also possible to determine that slightly increasing the abstraction, even by just one or two levels can decrease broadcast traffic extensively. More exact values need to take the likelihood of finding a suitable service in n-hops into account, and more concretely for that purpose also (i) the underlying service hierarchy and (ii) the topology of the network (which ultimately decides the communication graph for a request – how request are propagated through the network).

The need for improvements which adjust the static protocol behavior according to the

1. Network topology,
2. Service Hierarchy and
3. Node Mobility

becomes apparent based on the obtained simulation results, which show both a scalability problem especially in regard to very dense networks and a loss of efficiency in mobile scenarios. In such cases too many redundant messages are "selectively" forwarded, and thus the optimization present in GSD degenerates.

So, an obvious improvement over the original forwarding algorithm would be to not forward a request to every suitable neighbor node but rather to a dynamically chosen proportion. More concretely, two main mechanisms could be employed to improve protocol behavior in this regard:

1. A node should keep track of its neighbor nodes by inspecting the ongoing traffic. It should discard nodes from whom it has not seen any traffic for a certain amount of time when deciding how to selectively forward.

2. Beyond that it should select the node as target from whom it has received the most information in its service cache and which has suitable information to further forward the request. The underlying reasoning here is that such nodes are located at favorable positions in the network topology and can act as "relay" nodes for selective forwarding.

Additionally, network density could for example be derived by extending the existing protocol and inspecting the number of messages a node forwards. Based on this and the knowledge about the service hierarchy in use, it would be possible to estimate the likelihood of finding a service after a certain number of hops.

Another piece of context information that could be used to improve the protocol is information regarding the mobility of a certain node. With both, information regarding the general network density (neighbor count) and the mobility of neighbor nodes, more intelligent decision could be made regarding the selective forwarding mechanism. The protocol could be tuned further, to the point at which it can decide, based on available context information, how suitable a neighbor node is as a potential target for request forwarding.

Another insight gained, during the writing of this theses, is the limited scalability of NS-2. Scaling the simulation size to several thousands of nodes for example becomes increasingly time consuming, as NS-2 is limited by its sequential nature which runs in a single event processing loop without taking advantage of more than one processor. Thus the simulator architecture itself becomes a severe bottleneck in large scale simulations and consequently distinct design goals of NS-3[1] are inherent support for parallel and distributed simulations.

[1] `http://www.nsnam.org/`

Bibliography

[1] S. Corson and J. Macker. RFC 2501 : Mobile Ad hoc Networking (MANET) — Routing Protocol Performance Issues and Evaluation Considerations, January 1999.

[2] Salutation Consortium. Salutation Architecture Specification (Part-1), 1999.

[3] Microsoft Corporation. *Universal Plug and Play Device Architecture*, June 2000.

[4] E. Guttman, C. Perkins, J. Veizades, and M. Day. RFC 2608 : Service Location Protocol, Version 2, June 1999.

[5] Sun Microsystems. *Jini 2.0 specification*, April 2003.

[6] OASIS Open. *UDDI Version 3.0*, July 2002.

[7] Marc Krochmal and Stuart Cheshire. Dns-based Service Discovery, 2004.

[8] 3Com, Agere, Ericcson, IBM, Microsoft, Motorola, Nokia, and Toshiba. *Bluetooth V1.1 Core Specifications*, 2003.

[9] Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, and Tim Finin. Towards Distributed Service Discovery in Pervasive Computing Environments.

[10] Choonhwa Lee, Nitin Desai, Sumi Helal, and Varun Verma. Konark — A Service Discovery and Delivery Protocol for Ad-Hoc Networks, 2003.

[11] Sean Bechhofer, Frank van Harmelen, et al. OWL web ontology language reference, 2004.

[12] C.E. Perkins and E.M. Royer. Ad hoc on demand distance vector (AODV) routing. *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, 100, 1999.

[13] C.E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computer. *Comp. Commun. Rev*, pages 234–44, 1994.

[14] S. Narayanaswamy, V. Kawadia, R.S. Sreenivas, and PR Kumar. Power control in ad-hoc networks: Theory, architecture, algorithm and implementation of the COMPOW protocol. *European Wireless Conference*, 2002, 2002.

[15] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162, New York, NY, USA, 1999. ACM Press.

[16] Mario Čagalj, Jean-Pierre Hubaux, and Christian Enz. Minimum-energy broadcast in all-wireless networks: Np-completeness and distribution issues. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 172–182, New York, NY, USA, 2002. ACM.

[17] Weifa Liang. Constructing minimum-energy broadcast trees in wireless ad hoc networks. In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 112–122, New York, NY, USA, 2002. ACM.

[18] Paolo Santi. *Topology Control in Wireless Ad Hoc and Sensor Networks*. John Wiley & Sons, September 2005.

[19] Arunabha Sen and Mark L. Huson. A new model for scheduling packet radio networks. *Wirel. Netw.*, 3(1):71–82, 1997.

[20] Z. Gao, L. Wang, X. Yang, and D. Wen. PCPGSD: An enhanced GSD service discovery protocol for MANETs. *Computer Communications*, 29(12):2433–2445, 2006.

[21] M.D. Penrose. A Strong Law for the Longest Edge of the Minimal Spanning Tree. *Ann. Probab*, 27(1):246–260, 1999.

[22] L. Booth, J. Bruck, M. Cook, and M. Franceschetti. Ad hoc wireless networks with noisy links. *Proc. ISIT*.

[23] C. Bettstetter. Failure-Resilient Ad Hoc and Sensor Networks in a Shadow Fading Environment.

[24] The ns manual: formerly known as notes and documentation. `http://www.isi.edu/nsnam/ns/`, 2003. Page accessed on 12/2007.

[25] LF Perrone and Y. Yuan. Modeling and simulation best practices for wireless ad hoc networks. *Simulation Conference, 2003. Proceedings of the 2003 Winter*, 1, 2003.

[26] A.M. Law and W.D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 1997.

[27] HT Friis. Noise Figures of Radio Receivers. *Proceedings of the IRE*, 32(7):419–422, 1944.