University of Oslo
Department of Informatics

Service Discovery in On-demand Mobile Ad-hoc Networks

Cand. Scient. Thesis

Yan Zheng
yanz@ifi.uio.no

18th July 2004

Abstract

Discovery of services and other named resources is expected to be a crucial feature for the usability of mobile ad-hoc networks (MANETs). Different types of service discovery architectures are distinguished by the extent that service coordinators (SCs) are implemented in the network. A service coordinator is a node that holds a central repository for caching attributes and bindings for services of servers located in its neighborhood.

In this thesis, we evaluate the performance of different service discovery architectures in terms of service availability, message overhead and latency on reactively routed MANETs. We also discuss different methods that can be used to enhance the service availability and their pros and cons. We have, in this thesis, especially focused on the trade-off between the service availability and the message overhead.

This thesis will also demonstrate the benefits of combining the service discovery with the route discovery, especially on on-demand MANETs where reactive routing protocols are being used.

Preface

This thesis is submitted to the Department of Informatics at the University of Oslo as part of a Cand. Scient. degree.

Acknowledgement

I would like to thank my supervisor, Paal Engelstad, for his incessant encouragement, support, guidance and help throughout the whole period.

I would like to thank, Professor Paal Spilling, for sharing his knowledge.

I would also like to thank the people who are working in Telenor R&D for providing such a warm working environment and for making this whole experience memorable.

A special thank should be given to a fellow master student, Kjetil Marinius Sjulsen, for the great fellowship and discussion during the work.

Finally, I would like to thank my family for always being there for me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
Introduction

A mobile ad hoc network (MANET) is a set of mobile nodes connected by wireless links forming a dynamic autonomous network without any pre-existing infrastructure. Because of the arbitrary and unpredictable movement of the nodes in the ad hoc network, the network topology will be subject to constant changes. Ad hoc nodes are heterogeneous and they function as both routers and hosts.

Discovery of services and other named resources is anticipated to be a crucial feature for the usability of mobile ad-hoc networks. In the dynamic environment of MANETs, different nodes offering different services may enter and leave the network at any time. Efficient and timely service discovery is a prerequisite for good utilization of shared resources on the network.

On a MANET, any node may in principle operate as a server and provides its services to other MANET nodes or as a service requestor and uses the service discovery protocol to discover available services and their service attributes presented on the network. This includes IP addresses, port-numbers and protocols that enable the client to initiate the selected service on the appropriate server.

The Internet community has not yet reached a consensus on one particular service discovery protocol that is likely to be supported by most Internet hosts. There are a number of proposed service discovery mechanisms - such as Jini [10], Service Location Protocol (SLP) [7] [8], Salutation Protocol [11], UPnP/SSDP [13] and Bluetooth SDP [20] [21].

As a slight simplification, one may say that all these protocols are based on two baseline mechanisms for management of service discovery information:

1. Information about services offered on the network is stored in one or a few centralized nodes.
2. Information about each service is stored on the node that is offering the service.

In this thesis we define the service discovery architectures with regard to these two mechanisms. Solution only based on the first mechanism is referred to as a **service coordinator based architecture**, while a solution only based on the second mechanism is referred to as a **pure flooding architecture**. Finally, a solution based on a mixture of both the first and the second is referred to as a **hybrid architecture**.

Existing service discovery mechanisms are normally designed with a fixed network in mind, and might not fit well to mobile ad-hoc networks. Mobile ad hoc networks are normally highly dynamic and without any pre-existing infrastructure. These characteristics call for particular considerations. Hence, before a service discovery mechanism for ad-hoc networks can be designed or selected, one need to evaluate what kind of service discovery architectures are most suitable for mobile ad-hoc networks.

Güichal [34] undertakes an analysis of different service discovery architectures based on simulations. The work concludes that the hybrid architecture normally outperforms both the service coordinator based and the pure flooding architecture. The pure flooding architecture is the second best choice, and yields less messaging overhead. Despite this, Güichal [34] concludes that the hybrid architecture gives an overall better performance, because it yields higher service availability. A shortcoming of the simulations from Güichal's work [34] is that they do not take the importance of underlying routing into consideration. This assumption might be appropriate when a proactive routing protocol is being used, because with proactive routing the traffic patterns and service discovery search patterns do not influence the amount of routing messages. With a reactive routing protocol, on the contrary, this assumption may not hold, and the simulation results are not applicable. Data traffic will trigger messaging by the reactive routing protocol. Hence, service discovery messages will increase the routing overhead. We therefore anticipated that the routing overhead would be much higher with the hybrid architecture than with the pure flooding, simply because the hybrid architecture proved to require more messages on the network.

The layout of the thesis is as follow:
Chapter 2 gives an overview of the major research areas in MANETs including routing and service discovery. Chapter 3 presents relevant work related to service discovery in MANETs. Chapter 4 shows how service coordinators can be introduced to reactively routed MANETs in a bandwidth-efficient way. This chapter also discusses the importance of the placement of service coordinators relative to service requestors and servers. Chapter 5 presents the simulation setup. Chapter 6 presents the results from a simple simulation with five nodes. Chapter 7 presents simulation results that compare the performance between the pure flooding and the hybrid service discovery architecture in networks with static topologies. Chapter 8 repeats the same simulations with mobility added to the network. Conclusions are drawn in Chapter 9, and directions for further work are discussed.

1.1 Research purpose

In this thesis, a new comparison will be made between the pure flooding and the hybrid architecture, to determine if Güichal's conclusion [34] still holds in a reactively routed network. Both the overhead of the service discovery mechanism, as well as the additional routing that is triggered by the mechanism is taken into evaluation. When we evaluate the two architectures, we look for a user-friendly solution that gives a high level of service availability, low discovery delay, and so forth. At the same time, we want a network-friendly solution, i.e. with low messaging overhead and with little additional complexity added to the network. To a certain degree, it is also possible to increase the user-friendliness at the cost of introducing more messaging. Since the service discovery mechanism has an influence on the reactive routing protocol and since the two mechanisms share a lot of similarities, it is possible to make optimizations between the two to reduce the overall routing overhead. Here we use the optimization methods that are based on the proposals from [30] and [31].

Chapter 2
Background

This chapter will briefly present some of the major research topics related to mobile ad hoc networks including routing, service discovery and middleware technology with special emphasis put on the service discovery.

2.1 Mobile Ad Hoc Network

A Mobile Ad Hoc Network (MANET) is a set of mobile nodes connected by wireless links forming a dynamic autonomous network without any pre-existing infrastructure. Because of the arbitrary and unpredictable movement of the nodes in the mobile ad hoc network, the network topology will be subject to constant changes.

MANET nodes are heterogeneous with respect to their processing power, storage capacity, battery life and so forth. They communicate with each other without the need of any centralized access points or base stations. They function as both routers and hosts and they are responsible to cooperate with each other to route network traffic. Multiple hops may be needed when two nodes out of each other's radio range wish to communicate, hence the term multi-hop network.

A MANET is easy to set up because of its minimal dependency on the fixed infrastructure. This makes it ideal in supporting applications that need instant network formation in mobile or temporary environment where fixed infrastructure is unavailable or undesirable, e.g. conventions, construction site, disaster relief etc.

2.2 Routing Protocols for MANETs

Traditional routing protocols for packet switched network using either link state or distance vector algorithms are designed primarily for fixed network with infrequent topological changes and stable and symmetric links. They don't fit so well in MANETs due to several salient restrictions of MANETs, for example dynamic topology, limited bandwidth, constrained energy etc.

In a MANET, a high rate of unpredictable topological changes is expected, which are often caused by the mobility of nodes, power outages etc. In addition, the bandwidth is usually very limited. Thus the dissemination of up-to-date routing information can easily cause network congestion if the routing algorithm should react to the topological changes. Furthermore, it often takes long time for a network to converge by using traditional routing algorithms, which is not considered as an ideal situation for an extremely dynamic environment like the MANET. MANETs call for fast convergence so as to ensure seamless communications between mobile nodes. Routing in MANETs is typically performed using only host specific routes as opposed to network specific routes in fixed networks. Routes should be formed with minimal overhead and bandwidth consumption.

Existing schemes for routing in MANETs can be broadly classified into three categories, namely proactive, reactive and hybrid. They all have their pros and cons. Following subsections will give an overview of some of them.

2.2.1 Proactive routing protocols

Proactive routing protocols bear a strong similarity to the traditional routing algorithms. They are also called table-driven routing protocols because of their concerted effort to keep the various tables updated. Proactive routing protocols maintain consistent routing information from each node to every other node in the network. In order to keep routing information consistent and up to date, they will periodically distribute routing updates throughout the network to reflect the topological changes. Different proactive routing protocols distinguish themselves by the way routing information is handled.

2.2.1.1 Destination-Sequenced Distance Vector (DSDV)

The DSDV [1] routing protocol is a modification of the conventional Bellman-Ford routing scheme. It adapts the traditional distance vector based routing to MANETs. It solves the routing loop and counting-to-infinity problems that often occur in the traditional distance vector based routing due to topological changes.

A node that implements DSDV [1] maintains two important tables, one is the routing table which is used for forwarding packets and the other is the route-settling table which is used for damping the network fluctuation.

The routing table records in each of its entry the address of the available destination node, the next hop towards the destination, number of hops to the destination, a destination generated sequence number, a lifetime indicating the period of time the route is considered to be valid and a pointer to an entry in the route-settling table. The routing table maintains fresh routing information to all the available destinations in the network. Loop freedom is guaranteed through the use of destination generated sequence numbers. When a node receives route updates from its neighbor nodes, it will only update the recorded route to a destination if one of the following two criteria is met.

- The new route has a higher sequence number.
- The sequence number is the same, but the new route exhibits a better metric (i.e. fewer hops to the destination).

The route-settling table holds information concerning the stability of routes to various destinations. For every available destination, it is recorded an average settling time, i.e. the average time taken between the receipt of the first and the best route for the destination. A node should wait twice the average settling time before re-broadcasting the route updates received from the neighbor nodes. In such a way, the network fluctuation will be alleviated and network traffic is reduced by eliminating the unnecessary broadcast of route updates that might occur if a node should always receive the route with worse metric first and a better one right after.

In DSDV [1], route updates are broadcasted periodically or immediately triggered by significant topological changes due to the movement of nodes or alike so as to keep all the tables up to date and consistent. Two types of update packets, full dump and incremental update packet can be sent. The former contains the whole routing table information, which usually consumes several network protocol data units (NPDUs) and should be broadcasted periodically regardless of the existence of any topological changes. These packets can be transmitted infrequently in a rather static network. Incremental update packets contain only those routing information that has been changed since last full dump. Each of these packets should fit in one NPDU, thereby sparing the bandwidth usage. These packets are sent between full dumps. If it should happen that the size of one NPDU is exceeded, a full dump will be scheduled.

Evaluation: Several parameters need to be negotiated for this routing protocol, for example, the updates interval (i.e. full dump updates interval and incremental route updates interval), the settling time for each destination and the route expiration time, so that a balance can be made between route validity and communication overhead. DSDV [1] assumes bi-directional links, which are not always the case in MANETs.

2.2.1.2 Optimized Link State Routing (OLSR)

OLSR [2] is another proactive, table-driven routing protocol worth mentioning.
The optimizations as the name promises are reflected in two ways:

- The protocol engages only a set of nodes called multipoint relays (MPRs) in retransmitting the control messages that are meant to be flooded to the entire network, thereby reducing the total number of duplicate retransmissions.
- The protocol allows control messages from a node to contain only the information about link states to those neighbor nodes that have chosen this node as their multipoint relay, thereby reducing the size of the control messages.

Multipoint Relay (MPR)
MPRs are a subset of a node's one-hop, symmetrical (i.e. bi-directional link) neighbor nodes that are selected independently by the node based on the criterion that they should cover all the two-hop neighbors of the node. In such a way, control messages can be flooded to all the nodes in the network through multipoint relays. As mentioned above, only multipoint relays are engaged in relaying the control messages throughout the network. Accordingly, the smaller the MPR set, the less bandwidth it is consumed and the more optimal it becomes. However, a bigger MPR set can secure eventual link failures.

Neighbor sensing
The neighbor sensing mechanism in OLSR [2] has made it possible for a node to detect its direct connected neighbors. It is done with the help of so-called HELLO messages.

Every node in the network will periodically broadcast HELLO messages to its one-hop neighbors. These messages contain information about sending node’s one hop neighborhood and the link status. The MPR set chosen by the sending node is also announced through these messages. Through the information conveyed in these

HELLO messages, a node is able to keep a neighbor sensing information base that holds the information about its one-hop neighbors, two-hop neighbors, MPRs and MPR selectors. MPR selectors are those neighbor nodes that have chosen this node to be their multipoint relay. The neighbor sending information base will be updated from time to time to reflect the topological changes in the neighborhood.

Network topology
In order to construct routes to all the other nodes in the network, each node keeps a topological information base for the whole network. Topological information is gathered through another periodic message type called Topological Control (TC) messages. TC messages are generated by MPRs (advertising nodes). They must at least contain the reachability information to those one-hop neighbors that have selected the advertising node as their multipoint relay. It is a partial link state.

Besides TC messages, there are two other important control messages that help a node in gaining a complete view of the network topology.

One is called Multiple Interface Declaration (MID) messages, which are broadcasted by nodes that are associated with more than one network interface and all of which are running OLSR [2]. These messages contain the interface addresses that are associated with the sending node. Through these messages, each node can build an interface association information base for the entire network.

The other type of message is called Host and Network Association (HNA) messages that are broadcasted to the entire network by those nodes that act as “gateways” between the ad hoc network and a group of hosts or a subnet that doesn’t run OSLR [2].

Routing tables
Routing tables are constructed with the information acquired through all the aforementioned control messages. Through these messages, a node will record in its topological information base a number of connected pairs in the form of [last-hop, destination node]. Routes are formed by tracking these connected pairs in a descending order. Changes in the neighbor sensing information base, the topological information base or the interface association information base will trigger a routing table update.

Evaluation: OLSR [2] is best optimized in a compact network with random traffic. It is even better if the communication pairs change over time, because route is available all the time as opposed to reactive routings in which significant amount of query traffic may be initiated.

2.2.2 Reactive routing protocols

Considering all the overhead in trying to keep all the routes up to date in the proactive routing protocols and the fact that some of these routes may never even be used, another approach in routing protocols for MANETs is made. They are called source-initiated on-demand or reactive routing protocols. Routes are only created when desired by the source node and maintained under the duration of the communication between the source node and the destination node.

2.2.2.1 Ad hoc On-demand Distance Vector (AODV)

AODV [3] is one of the representative routing protocols that fall under this category. Only routes to those destinations that a node is communicating with are maintained in the node's routing table.

Route request

When a node wishes to communicate with another node, but doesn't yet possess any valid routes to it, the node will then initiate a route discovery by broadcasting a route request (RREQ) to the network. A monotonically increasing broadcast ID is associated with every new RREQ initiated by the node. This broadcast ID together with the IP address of the RREQ initiator uniquely identifies the route request. This information will be stored in every receiving node of the RREQ for a predefined period of time, so that duplicate route requests can be ignored. En expanding ring search technique is used to prevent unnecessary network-wide dissemination of RREQs. The basic idea of this technique is to incrementally increase the flooding scope of a RREQ until a route reply is received or until it reaches a predefined threshold beyond which a network-wide flooding scope will be used. As RREQ traverses the network, reverse routes to the RREQ initiator are generated. Reverse routes will be needed to eventually route back the route reply. This requires bi-directional links.

Loop freedom and route freshness

Loop freedom and route freshness are ensured by the use of destination generated sequence numbers. This is the same idea as that mentioned for DSDV [1].

Route reply

A route reply (RREP) is unicasted back to the RREQ initiator from either the destination node itself or any intermediate node with a “fresh” enough route to the destination. A “fresh” enough route means the cached route to the destination has a valid sequence number that is at least as great as the one from the RREQ packet. Route replies are relayed back using reverse routes that were created along with the RREQ. In the case of RREP by an intermediate node, an unsolicited RREP will be sent to the destination node by the intermediate node as if the destination node has requested a route to the source node. This is to facilitate a bi-directional communication between the source (RREQ initiator) and the destination. As RREP is routed back along the reverse route, a forward route to the destination node will be created. Precursor lists for the source and the destination node will also be created. A Precursor list, as one field of the route entry for a certain destination, is a list of nodes (active neighbors) that have recently utilized this active route to forward packets to the destination. These are the nodes to which route error message (RERR) should be forwarded when the destination becomes unreachable. Multiple route replies may be received by a certain node, only better routes (i.e. routes with greater sequence number or same sequence number yet fewer hops) will be forwarded towards the source node. The source node will begin to use the first discovered route, however, better route will be discovered and used over time.

Route maintenance
Route maintenance concerns only nodes in active routes as opposed to proactive routing protocols where all nodes are engaged in route maintenance. A link breakage can be discovered by failing to receive any kind of broadcast messages (e.g. RREQ, RREP), periodic HELLO messages from the neighbor or by link layer methods. If one of the intermediate nodes on the active path discovers a broken link to the next hop towards the destination, a route error message will be propagated to the node's active upstream neighbors and in turn their active neighbors until the message reaches the source node. Along with the propagation of route error messages, routing tables are searched and routes affected are invalidated.

Evaluation: AODV [3] requires symmetric links between nodes, which cannot be guaranteed in the ad hoc environment. Due to its relative low memory and CPU usage, its scalability is quite promising.

2.2.2.2 Dynamic Source Routing (DSR) protocol

DSR [4] is another on-demand routing protocol. A node that is running DSR [4] will record in its route table a full path to a destination as opposed to all the aforementioned routing protocols in which only next hop information is recorded. This increases the memory usage in individual nodes, which could be a scarce resource for some devices. Furthermore, the complete path to the destination is included in the header of every data packet sent. This might cause the packet size to exceed the maximum MTU of the underlying network, which leads to fragmentations.

DSR [4] consists of two phases; route discovery and route maintenance.

Route discovery
In an on-demand route discovery, a node will broadcast a route request when there are no valid routes cached for the destination it intends to communicate with. The mechanism to avoid duplicate processing and forwarding of route requests is the same as that in AODV [3]. The intermediate node that receives the route request will first append its own address to the route record that is contained in the route request packet and then rebroadcast the packet until it reaches the destination node or an intermediate node with a valid route to the destination. Route replies can be generated either by the destination itself or the intermediate node with a valid cached route. In the former case, the route record will be copied directly to the route reply from the route request. In the latter case, the route record should be appended with the cached route first before being copied to the route reply. There are three alternatives to route the route reply back to the source node. A route reply can be propagated back to the source through a cached route. Alternatively, the node can simply reverse the route record contained in the route request and use the reversed route to route back the reply. A third alternative is to trigger a new route discovery and piggyback the route reply in the route request. The cost of route discovery by flooding the route request is very high in terms of bandwidth, power and time etc. Hence nodes usually choose to cache many learned or overheard routes. In addition, multiple routes for a single destination are cached. In such a way, the cache can be exploited aggressively so as to reduce the need for route discovery.

Route maintenance
DSR [4] does not rely on periodic HELLO messages to supervise the link connectivity. Instead, built-in acknowledgement mechanisms are used. One might use link layer acknowledgements or passive acknowledgements (i.e. overhearing transmissions of neighbors). Alternatively, software acknowledgements (i.e. explicit acknowledgement request messages) can be used. In case of a link failure, a route error message will be sent back to the source host. Upon receiving a route error message, the broken link will be removed from the route cache and all routes containing the broken link will be truncated from that point on.

Evaluation: DSR [4] exhibits a big message overhead and a high memory usage due to the fact that a full path has to be carried in every packet transmitted and has to be stored in the route table. Cached routes are meant to cut the need for route discoveries. However, stale routes may be used due to the aggressive usage of cached routes. Optimizations that require each node to work in promiscuous mode in order to monitor the network traffic within range (i.e. overhear routes from other nodes) will result in more CPU usage, but this problem can eventually be solved using special network interface hardware.

2.2.3 Hybrid routing protocols

In order to provide a better trade-off between the communication overhead and the delay, the hybrid approach comes into being. It partitions the whole network into (overlapping) zones and uses a proactive approach in the intrazone routing while a reactive approach in the interzone routing.

2.2.3.1 Zone Routing Protocol (ZRP)

ZRP [5] is representative in this category. It consists of three basic components, namely intrazone routing protocol (IARP), interzone routing protocol (IERP) and bordercasting resolution protocol (BRP).

ZRP [5] divides the network into overlapping zones called routing zones. Each node specifies a zone radius in terms of radio hops for its own routing zone. Intrazone routing protocols, which can be any of the suitable proactive routing protocols with slight modifications, are used to route the traffic inside the zone while interzone routing protocols, which can be any of the suitable reactive routing protocols, are used by nodes to discovery route to the destination node that lies outside the current routing zone. Bordercasting mechanism is used in interzone routing to relay the query packet across the overlapping zones by directing them towards the uncovered border nodes of the routing zone as opposed to the usual broadcasting mechanism where packets are routed from neighbor node to neighbor node. A node is considered as being covered if the query packet has already been delivered to it. Bordercasting reduces the traffic load caused by route queries.

2.2.4 A general comparison of proactive, reactive and hybrid routing approaches

- With proactive routing protocols, routes from each node to every other node in the network are always available. This will eliminate the initial delay in finding the route and ensure higher quality routes in a static topology. With reactive routing protocols, on the other hand, routes are only created when needed by the source node. When there is no route to the destination node the application wants to communicate with, the delay caused by the route discovery may be significant from an application's point of view. Real time communication will favor proactive routing protocols in this regard.

- Proactive routing protocols incur higher bandwidth and power consumption. Substantial update messages triggered by frequent topological changes and periodic control messages are flooded in network so as to keep the routing tables consistent and up to date. This will consume a huge part of the already scarce bandwidth. Some of the nodes may use most of their processing and battery power to process and relay these routing updates instead of doing any other constructive tasks. Many of these routes may not even be used. Furthermore, routes to every other node in the network are cached in the routing table, which might take up lots of node's memory space if the network is of great magnitude. Memory space is another scarce resource for many mobile devices

- Proactive routing protocols, however, provide more often optimal routes. They continually reevaluate the routes and adjust them according to the topological changes. Reactive routing protocols, on the contrary, will generally stick to the established routes until they can no longer be used even if some other more optimal routes exist.

- In reactive routing protocols, the flooding of route discovery requests might easily saturate a large network. Nodes that don't lie on the final established route will still have to process and relay the route discovery requests, thus wasting the limited processing energy for nothing.

- Proactive routing protocols favor random and sporadic communication patterns while reactive routing protocols prefer relatively long communication sessions between a small set of nodes at any one time. Proactive routing protocols beat reactive protocols especially when the communicating source and destination pairs are changing frequently, since in such case, a lot of control messages (route requests, route replies, etc.) will be initiated so as to find a route between the new source and destination if the reactive routing protocol is used.

- Hybrid routing protocols distinguish themselves the most, yet they bring additional complexities to the network and many factors remain to be considered, e.g. the size of the zone, interplay between interzone and intrazone routing protocols etc.

2.3 Service discovery

With the booming amount of disparate services available in the networks and the increasing mobility they expose, a mechanism for service discovery should be provided for devices to automatically and dynamically "advertise" the services they provide and "discover" the existence, location and configuration of the desired services in the network. The goal of a service discovery is to allow service users to search for services by names, types, attributes etc. instead of IP-addresses and port numbers. In addition, service users usually wish to browse for services and to be freed from the burden of manually reconfiguring the system upon accessing the services. In short, the ultimate goal is to facilitate the task of finding and using the services for the service users. With the proliferation of mobile devices, service discovery mechanisms should support dynamic environments.

Service discovery mechanisms can be broadly classified into three categories:

- Services are registered at a central registry and clients search for services at the registry.
- Servers advertise their services to potential clients through multicasting or limited broadcasting.
- Clients multicast or broadcast service requests to the network. Servers with matching services respond to the service requests usually with unicast service replies.

Many service discovery protocols have been proposed. However, they are more or less designed for fixed networks than for MANETs.

2.3.1 Service Location Protocol (SLP)

SLP [7] [8] is an IETF standard for service discovery and automatic configuration of clients for IP-based networks. SLP [7] [8] has been designed with the intention to enrich the primitive service matching mechanisms and improve the scalability of some proprietary protocols. It allows users to request for services based on characteristics as well as types. Version 2 of SLP [7] [8] has now replaced the first version.

SLP [7] [8] presents a framework that consists of three types of agents, not all of which are mandatory:
A User Agent (UA) is a process that requests for services on behalf of the client applications.
A Service Agent (SA) is a process that advertises the service location and characteristics on behalf of one or more services
A Directory Agent (DA) (optional) is a process that aggregates service information into a central repository. The use of directory agents improves the scalability.

On starting up, UAs and SAs will first check for the presence of DAs. DA related information could be distributed through either static configuration or DHCP service location option (78) [9]. If none such information is configured through these methods, UAs and SAs must initiate either active DA discovery or passive DA discovery. In active DA discovery, UAs and SAs multicast service requests for DAs using multicast convergence algorithm [8]. In multicast convergence algorithm, a service request is attached with a responder list, which is an address list of all the agents that have already responded to the request. The service request will be retransmitted several times so as to collect as many responses as possible. Agents that are already listed in the responder list will discard the service request. The responder list keeps the network especially the requesting node from being inundated with duplicate responses from the same node. In passive DA discovery, UAs and SAs wait for the unsolicited multicast DA advertisements generated once in a while by DAs. If some DAs are present in the network, DA advertisements with DAs' location information, scope information, associated attributes etc. will be received be it an active or a passive discovery approach.

Scope is a concept that improves the scalability. It is a null terminated text string, which is used to group resources by location, network or administrative category [7]. A UA can only discover those services that are configured with at least one of the scopes that are assigned to the UA. UAs configured with "NO SCOPE LIST" can multicast service requests for DAs or SAs so as to retrieve all the available scope information in the network and later to discover all the services within all the scopes. A SA should register all its services with all the discovered DAs provided that the conjunction set of the scope lists of the DA and the SA is not empty.

Service location information is expressed by a service URL, which contains all the needed information (IP address, port number etc.) to contact the service. Legal attributes and their default values for a specific service type are defined using a service template. Service attributes are registered and queried using the same definition as that specified in the service template. A service template defines a common vocabulary between service requestors and service providers.

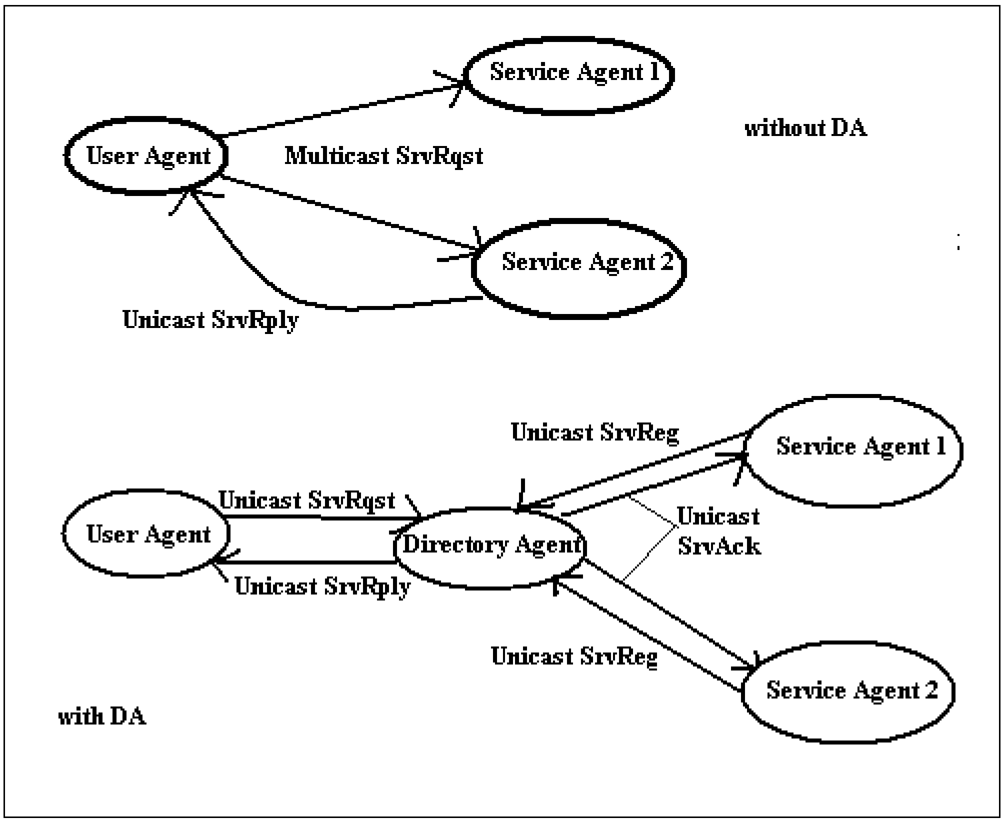

Figure 1: SLP's two different operating modes with or without DAs present

SLP [7] [8] will operate in two modes depending on the existence of DAs. As Figure 1 above illustrates, with the presence of DAs, a SA will register with all the discovered DAs in its scope all its advertised services. Upon successful registration, an acknowledgement will be unicasted from the DA back to the registering SA.

Services are registered with their service URLs, service types, lifetime (the time period during which services are considered to be available), possibly associated attributes and so forth. SAs should refresh their service registrations periodically before their lifetime expire, otherwise service entries will be purged from DAs on expiration. A UA initiates a service discovery by unicasting a service request to a selected DA and the DA will unicast back a service reply if a match is found. Services are matched by service types, scope information and possibly service attributes. A service reply consists of service URLs to the matched services and their lifetime. In the absence of DAs, UAs will query SAs directly by sending service requests using multicast convergence algorithm [8], SAs with the matching services will unicast service replies back to the requesting node.

All SLP [7] [8] messages are sent in UDP datagrams and truncated if they exceed the maximum UDP packet size. However, a TCP connection might be opened when a node receives a truncated service reply. In that case, the service request should be retransmitted. There is no mention of the actual protocol for accessing the service in the specification.

Evaluation: Multicast and DHCP are used in initialization. Neither is scalable as far as Internet is concerned. As a result, SLP in its current form is not scalable either, thus not suitable for MANETs.

2.3.2 Jini Technology

Jini [10] introduced by Sun Microsoft is a Java centered technology. It introduces the concept of a federation, which is a collection of Jini technology-enabled services that co-operate with each other to achieve the goal of resource sharing.

Jini [10] distributed system architecture is comprised of an infrastructure, a programming model and many services. The central components of the infrastructure are a lookup service and a trio of protocols called discovery, join and lookup.

Jini Lookup Service (JLS), which is the counterpart to the DA in the aforementioned SLP [7], serves as a repository for up-to-date service information within the Jini federation. A discovery protocol is used by a newly started service/device, referred to as entity henceforth, to locate lookup services to register with. There are three related discovery protocols, namely a multicast request protocol, a multicast announcement protocol and a unicast discovery protocol. The unicast discovery protocol is used by an entity to contact a lookup service on a known host and it is also used by the other two discovery protocols in the final phase of a lookup service discovery. In unicast discovery, a TCP connection must first be established between the entity and the lookup service on the known host. Then a simple request/response mechanism is used. A proxy of the lookup service through which en entity can invoke different methods of the lookup service will be sent in response. However, if a new entity starts up without any clue of the location of the lookup services, it simply multicasts a UDP request using multicast request protocol in order to obtain one or more references to the lookup services. This resembles the active DA discovery in SLP [7]. A lookup service, upon accepting the request, establishes first a TCP connection with the entity using the enclosed contact information in the request packet. Then the unicast discovery is performed by the entity as described above to get a reference/proxy of the

JLS. Another way to get a JLS proxy is to listen for multicast announcements sent out regularly by the lookup services using multicast announcement protocol. Lookup services will start sending out announcements the moment they start up. This way of getting JLS proxy is similar to the passive DA discovery in SLP [7]. An interested entity can then establish a TCP connection to a JLS using the enclosed contact information in the announcements and followed by the unicast discovery.

After the acquisition of JLS references/proxies, the service joins the federation by registering with the lookup services. It does so by uploading its service object containing the Java programming language interfaces for the service along with other descriptive attributes to the lookup service. This is accomplished by invoking the register method of the received JLS proxy.

Lookup can occur when a Jini client after locating the lookup service through the aforementioned discovery mechanism needs to discover a service that matches a certain interface type and possibly some other descriptive attributes. The node is doing so by invoking a lookup method on the received lookup service proxy. If a match occurs at the lookup service, the service object to the matched service will be downloaded to the client, such that the client can invoke different methods offered by the remote service through the downloaded service object/proxy using Java RMI [37]. This kind of code mobility has simplified the Jini system.

Jini's [10] group concept is a counterpart to SLP’s [7] scope concept. A group is an arbitrary string representing a name. Services can be configured with specific groups to join in.

The programming model of Jini [10] technology comprises a set of interfaces that support reliable service constructions. The leasing interface introduces the leasing concept, so that access to many of the services in the Jini system environment is time bounded. The requested leasing period is proposed by the requestor and negotiated between the requestor and the service provider and finally granted by the service provider. The resources will be freed when leasing period expires unless a renewal is done. The event and notification interface enables an object in one Java Virtual Machine (JVM) to register its interest in the occurrence of some events occurring in another object in some other JVM and receive the notification when the events do occur. The transaction interface allows for the atomicity of a transaction using the two-phase commit protocol, which guarantees that the transaction will either succeed or fail while leaving no inconsistent state in the network.

The Jini [10] specification also mentions that peer lookup can be employed in the lack of lookup services, in such case clients function more or less like lookup services with which services register. It is up to the clients to filter out the unwanted service responses.

Evaluation: Participants of the Jini federation must host a functioning JVM, which may not be feasible for some mobile devices with scarce memory spaces and low processing power. The service proxy concept is tempting, yet it assumes standard interfaces to be always available. Lookup services bear most of the burden in the network and single failures may affect the network performance.

2.3.3 Salutation Protocol

Salutation [11] developed by the Salutation Consortium is another approach to service discovery. It distinguishes itself as opposed to Jini's [10] language dependency on Java and SLP's [7] network transport dependency on TCP/IP. It aspires to solve the problem of service discovery and service utilization among appliances and equipment with dissimilar capabilities in an environment of widespread connectivity and mobility.

The salutation architecture consists of:
Two major components
A Salutation Manager (SLM) serves as a service broker. It mediates among the networked entities (i.e. devices, applications, service or functional units that have access to or may be accessed from other applications, services or devices [11]) to enable the discovering and utilization of the capability of one networked entity by another. It is somewhat like a distributed Jini lookup service [10]. In short, salutation managers let the services register their capabilities with them and they coordinate with each other to locate the desired services for the clients.

A Transport Manager (TM) hides the heterogeneity of the underlying networks from the salutation manager and ensures reliable communication channels to the salutation manager that sits on top of it. It can also locate other remote SLMs that are connected to the same network segment. In this way, the coordination among the SLMs is realized.

Each device can host at most one SLM. If no local SLM (i.e. located on the same device) exists, the device may use a remote SLM through remote procedure call (RPC) [12] mechanism. Depending on the number of different networks physically connected, a salutation manager may sit on more than one transport manager, each responsible for one type of network transport.

A basic and essential building block
A **Function unit** is the minimal meaningful functionality of a client or a service (e.g. [Print]) that can be expressed by a **Function Unit Description Record**. A Function Unit Description Record is assigned a unique handle when being registered at a salutation manager and it defines the type of a function unit. Each Function Unit Description Record is further composed of a collection of **Attributes Records,** which characterize the functionality. Services with several functionalities are described by one or more Function Unit Description Records, all of which constitute the **Service Description Record**.

Two important interfaces
The Salutation Manager Application Program Interface (SLM-API) provides the server and client applications with a transport-independent interface that facilitates the service registration, discovery and access.

A Salutation Manager Transport Interface (SLM-TI) provides transport-independency to the salutation manager with the transport manager dealing with the underlying network details.

Four major service broker tasks
A Service Registry is maintained by a SLM. It is a repository for the information of all the services locally or remotely connected. It is similar to the lookup service in Jini [10] and the DA in SLP [7].

Service Discovery is performed through coordination among the salutation managers. Client communicates with local SLM to request for a service, the local SLM contacts the remote SLM. Required service type specified by the local SLM is matched against the registered service description records at the remote SLM. A list of SLMs with the matching service will be returned, eventually together with function unit handles.

Service Availability is a simplified eventing mechanism. It is especially useful when a client makes a long-term request to a server, i.e. the time from a request is issued until the response is received is significantly long. Then it is essential for the client and the server to know whether the other part is still alive to respectively receive or deliver the response. The client and the server can require their respective local SLMs to perform the availability check by exchanging Remote Procedure Call [12] messages with each other, such that either part will be informed of the unavailability of the other by their respective local SLMs.

Service Session Management is handled by the SLM when the client wants to utilize the discovered services. The local or nearby SLM is asked to establish a service session between the client and the resolved server. There are three modes a communication can take place in.

- ***Native Mode*** SLM is responsible for initiating the session, yet it will not be involved in the data transferring between the client and the server.

- ***Emulated Mode*** The only difference between the emulated mode and the native mode is that SLM will also take part in the data transferring i.e. messages are sent in SLM packet, yet no inspection of content is performed.

- ***Salutation Mode*** In addition to the session initiation and the message streaming mediating, SLM should also determine the data format. Again no inspection of the content is performed.

Evaluation: Salutation protocol is platform, operating system and network independent.

2.3.4 Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) [13] developed by a consortium of companies headed by Microsoft, is an evolving standard that is designed to enable simple, ad hoc communication among distributed devices and services from many different vendors. UPnP builds on existing Internet protocols e.g. TCP/IP, HTTP, XML. Thereby it ensures the interoperability among different vendors.

There are several fundamental building blocks in UPnP [13].
Control Point: The set of software modules that have the ability to discover and control other controlled devices.

Controlled Device: The set of software modules that respond to the control point, e.g. responding to the discovery request, accepting controlling messages etc.

Device: It contains services and maybe other nested devices e.g. TV, VCR. A single device can implement a control point and one or more controlled devices simultaneously.

Service: It exposes actions that can be invoked on it and models its run time state by a list of variables.

Following are the major steps involved in the UPnP [13] networking.
IP address configuration Before the whole UPnP network would work every device should get an IP address. It is done by either contacting a DHCP [14] server in order to be allocated an IP address or for the lack of DHCP server claiming randomly a link local IP address in the range of 169.254/16 (The first and last 256 addresses are reserved) using auto-configuration [15]. After using auto-configuration, DHCP server should be intermittently searched and on discovering one, device should be assigned a new IP address by DHCP server and relinquish the auto-configured one.

Discovery After successfully acquiring an IP address, the device/control point can now participate in the discovery. A newly added device will send out a couple of discovery messages to notify its capabilities (services and embedded devices) to the network. Messages, often referred to as ssdp:alive, are multicasted to the reserved multicast address and port using HTTP extended with Simple Service Discovery Protocol (SSDP) [16] and General Event Notification Architecture (GENA) [18]. The discovery messages provide the network with the information like, among other things, device/service type, lifetime for the advertisement to remain valid and a pointer to a detailed device description file expressed in XML [19]. Control points can listen to the reserved multicast address for such advertisements or notifications. To prevent a network from entering an unwanted state, every device should also multicast several discovery messages (ssdp:bye bye) to revoke the not yet expired advertisements on leaving the network. The same protocol stack is used for the ssdp:bye bye messages. Similarly, a newly added control point can send out a couple of search messages looking for services of interest. These messages (ssdp:discovery) are also multicasted to the reserved address using HTTP extended with SSDP [16]. Any device with the matching services/embedded devices should unicast a response back to the control point that is doing the discovering. The response would contain the same information as the ssdp:alive messages. SSDP [16] is used in both service announcements and discovery, thus it functions similarly to Jini's [10] trio of protocols: discovery, join and lookup.

Description After having discovered the desired device, a control point knows only what was conveyed through the discovery messages (i.e. ssdp:alive) or the unicasted service response. Further detailed device description is provided in a device description file which location is included in the discovery messages or the service response. The most important content of the description file is a list of service types, service names, URLs for service description files and URLs for sending control and eventing messages to the services. If the device has embedded device, it will also contain a description of them. A service description gives the list of actions and corresponding parameters that will trigger responses and the list of variables that model the state of the service at runtime.

Control Given the necessary knowledge of a device and its services, a control point can send control message to the aforementioned control URL of a service to invoke device specific actions or to retrieve associated state variables. Results will be returned by the service. All the messages, be it control messages or results, are expressed in XML [19] and the interaction between the control point and the service is handled by Simple Object Access Protocol (SOAP) [17] using XML and HTTP.

Eventing As mentioned before, each service contains a list of variables that models the state of the service at run time. UPnP allows a service to publish updates whenever there occurs a change in these variables and it allows interested control points to subscribe for these events. A control point expresses its interest by sending a subscription message to the eventing URL of the service. The subscription message contains an event sink URL for receiving the notification. Granted subscription will be assigned a duration (leasing), control points should renew the subscription periodically to keep it valid. All these messages will be formatted in GENA [18].

Presentation If the device offers a presentation URL, control point can then provide a user interface for the device by downloading the page from the presentation URL to the local browser.

Evaluation: UPnP is multicast-based. There is usually no central registry around, which may results in high traffic load. UPnP using SSDP in service discovering limits the discovery to a single subnet. Queries are not aimed at the XML description. The description is only scrutinized after the desired service is discovered.

2.3.5 Bluetooth Service Discovery Protocol (SDP)

Bluetooth [20] [21] is a new low-power, short-range (10M), omni-directional wireless transmission technology operated in the 2.4GHZ ISM band. It contains in its protocol stack a service discovery protocol (SDP). Bluetooth SDP is optimized to discover the services provided by other Bluetooth devices in a dynamic environment.

Every SDP server maintains a repository of service information in the form of service records, one for each service. Each service record is uniquely identified by a 32-bit service record handle within the SDP server and is composed of a list of service attributes which describe different aspects of the service. Each service is an instance of a service class/type represented as a UUID[1]. The specific service class/type defines the legal service attributes and their semantics for that service class.

Bluetooth SDP [20] [21] supports:
<u>*Three kinds of service inquiries*</u>

- **Search for services by service class/type**

[1] The format of Universally Unique Identifiers (UUIDs) is defined by the International Organization for Standardization in ISO/IEC 11578:1996. "Information technology - Open Systems Interconnection - Remote Procedure Call (RPC)"

- **Search for services by service attributes** (i.e. search pattern is a list of UUIDs) Only attributes with values represented as UUIDs are qualified to be searched/matched.
- **Service browsing** is useful for a SDP client when the client has no a priori knowledge of the available services within range. Service browsing is based on a common attribute called BrowseGroupList shared by all the service classes. All the browse groups expressed in UUIDs that a service may be associated with are listed as the value of the BrowseGroupList attribute.

<u>*To possible scenarios*</u>
- Search for services on a connected device
- Search for services on an unconnected device that happens to be in the vicinity i.e. within radio range

Bluetooth SDP [20] [21] employs a request/reply model between a SDP client and a SDP server. A SDP session is comprised of a request protocol data unit (PDU) from a SDP client and the correspondent response PDU from a SDP server.

Before the SDP session can be established between a SDP client and a SDP server, they should first be connected. Firstly, an asynchronous connectionless physical link is established at the Baseband/Radio layer using a procedure called **inquiry** to discover all the discoverable devices within range and followed by a procedure called **paging** to actually establish the connection. Secondly, a link set-up is done at the LMP level. Thirdly, a connection oriented logical channel is established at L2CAP ([20] [21]).

Service discovery is performed as follows: A SDP client issues a service request PDU containing the search pattern. The search pattern can contain either a service class UUID if the client wishes to search for a service by service class or it can contain a list of attribute values also expressed in UDDIs if the client wishes to search for service by attributes or it can contain both. A SDP server with the matching service records (i.e. contains all the UUIDs in the search pattern) will respond with a service response PDU containing, among other things, service record handles for the matching services. The SDP client can use these service record handles to retrieve certain attribute values of the service. These two steps can also be merged into one.

Bluetooth SDP [20] [21] only provides mechanisms to discover services and retrieve diverse service discovery-related information, it doesn't provide any mechanism for service selecting or accessing. However, it does define a standard service attribute **ProtocolDescriptorList** that enumerates the appropriate protocols needed for accessing the service. It doesn't support brokering of services, service advertisements, service registrations or event notification.

Evaluation: Bluetooth SDP aims only at Bluetooth devices.

2.3.6 A comparison of the existing service discovery protocols

Though all the aforementioned protocols share much resemblance, they see things from different angles and they aim at different audiences. All the protocols have their respective pros and cons. A general comparison of these service discovery protocols can be found in [22]. Jini [10] and UPnP [13] emphasize on the pervasive computing environment while Salutation [11] and SLP [7] [8] deal more with service discovering problem. Finally, Bluetooth SDP [20] [21] aims more or less only at Bluetooth enabled devices.

1. SLP, Saturation, UPnP and Bluetooth SDP are all language independent, which means they can be implemented in any language as opposed to Jini, which relies on Java to keep all the promises. Jini requires that all devices should have a working JVM.

2. SLP is designed mainly for TCP/IP networks, so is UPnP that relies on an IP based network and web technology. As to Jini, the current Jini implementation is based on TCP and UDP (multicast), but other network protocols are also possible as long as they provide reliable, stream-oriented communication and multicast facility. Bluetooth SDP works only in the Bluetooth environment. Salutation with the help of Traffic Manager makes it totally independent on the network technology. Thus, saturation will distinguish itself when the non-IP based network makes its way.

3. In a dynamic, distributed network, leasing is very helpful to deal with the partial failure and maintenance problem of the network. SLP, Jini and UPnP all support some kinds of leasing. In SLP, service advertisements either registered at a DA or directly sent to a UA in the lack of DAs are assigned a lifetime which require periodical renewals otherwise the services will be considered unavailable and all the relevant information will be expunged. The same happens in UPnP, there is a header in the service announcements (ssdp:alive) and the service search response called *CACHE-CONTROL* that dictates the lifetime of a device or a service. In Jini, both service registrations at the Jini Lookup Service and access to the service are leased. Leasing period can be negotiated between the requestor and the grantor or decided by the grantor.

4. Jini, UPnP and Salutation all support eventing. In Jini, the event notification is realized by allowing an object in one JVM to register interest in the occurrence of some events in another object in some other JVM and receive notification in the form of remote event object either directly or through a third-party when an event occurs. Eventing in UPnP utilizes the subscription and publishing mechanism as described in section 2.3.4 on page 18 under UPnP eventing. Such subscription is leased (i.e. time-bounded) and should be renewed to remain valid. Salutation offers a simplified eventing mechanism termed service availability check which is one of the service broker tasks performed by SLM, see section 2.3.3 on page 16 for SLM eventing.

5. UPnP is unique for its use of XML to describe device features and capabilities, which provides a more sophisticated and powerful description compared to SLP's service types and attributes, Jini's interface types and attributes, Salutation's function units and SDP's service records. Yet as far as service matching is concerned, UPnP doesn’t use XML. The XML description file is not scrutinized until the requested device is found, i.e. query in UPnP is not based on XML. In this regard, SLP provides a rather powerful matching, it

supports substring, logical operators AND, OR etc. which allows for a more precise service searching compared to equality check in Jini, UPnP and SDP. In Salutation, certain well-defined comparison functions can be associated with queries when searching for services and will be used in service matching. UPnP is the only protocol that doesn't provide any mechanism for searching by server attributes.

6. Service browsing is supported in SLP, Jini, Salutation and Bluetooth SDP. In SLP, there are two kinds of messages that realize the idea of service browsing. One is the Service Type Request (SrvTypeRqst) which can be sent by a UA to discover all the available service types within the assigned scopes, the other is Service Attribute Request (AttrRqst) which can be initiated by a UA to retrieve all or a subset of attribute values associated either with a specific service instance or a generic service type. In Jini, three methods, namely getServiceTypes(), getEntryClasses() and getFieldValues(), enable the clients to browse for services and attributes in the lookup service. While in Salutation, there is a special function description record called “ALL CALL” that enables the clients to discover all the registered services in another Salutation Manager. Browsing in Bluetooth SDP is described in section 2.3.5 on page 19.

7. In SLP, a service can be contacted through the service URL contained in the service reply, but the actual protocol for accessing the service is not mentioned. In Jini, a service object/proxy of the remote service is downloaded from the lookup service. A client can invoke methods through this downloaded service proxy directly. In Salutation, a service session will be established between the client and the server by the local SLM on the client side and the local SLM will be involved in different degree in the communication depending on the mode used. The different modes are described in section 2.3.3 on page 16. In UPnP, control points can invoke commands by sending control messages to the controlled URL of the service, see UPnP control in section 2.3.4 on page 18. Bluetooth SDP doesn't provide any mechanism for service accessing. However it does define a standard service attribute **ProtocolDescriptorList** that enumerates the appropriate protocols for accessing the service.

8. All the aforementioned service discovery protocols have their respective salient features. Jini allows for code mobility. Salutation provides transport independence. UPnP offers automatic configuration and distinguishes itself by its use of XML. SLP has an authentication security feature.

2.4 Service discovery in middleware technologies

2.4.1 The importance of service discovery in Middleware

A middleware is a software layer that seeks to abstract the details of ad hoc communication from applications and enable smooth interactions among the applications regardless of their heterogeneities and the dynamic underlying network topologies. In a MANET, any node may in principle operate as a server and provides its services to other network nodes or as a client and requires services from other network nodes. In this dynamic environment different nodes offering different services may enter and leave the network at any time. In order to efficiently and timely locate the desired services, the middleware must provide some kind of service discovery.

Mature middleware technologies, such as CORBA [23] and SOAP/XML Web services [24] have been designed and used successfully with fixed networks.

2.4.2 CORBA

Common Object Request Broker Architecture (CORBA) [23] promoted by Object Management Group (OMG) provides a flexible communication substrate and platform neutral middleware for distributed, heterogeneous and object-oriented computing environments. In CORBA, applications are modeled as a collection of cooperative objects. These objects contain data and methods that can be invoked by other objects. Services are delivered through these method invocations. Services offered by an object are defined in Interface Definition Language (IDL). The major component in CORBA is an Object Request Broker (ORB), which helps a client object to invoke methods on other objects. An ORB hides the location, implementation and communication details from the applications. In order to access a service, i.e. invoke a method on an object, one has to first obtain an object reference. Obtaining an object reference can be thought of as a kind of service discovery in CORBA. It is realized by the use of naming and trading service. They are two of the many generic services offered by CORBA.

- **Naming services** allow an object to be bound with a friendly name (i.e. service registration) and later allow client to retrieve the object by this name.
- **Trading services** allow a client to find the object by its properties i.e. by its service types and associated attributes.

2.4.3 XML Web Services

XML web service architecture [24] provides another platform neutral middleware for disparate applications to interoperate with each other. A XML web service is an application component that offers a special service to other applications. In CORBA [23] applications are modeled as a collection of objects, while here applications are modeled as a collection of loosely coupled XML web services. XML web service can be best explained by the standards and protocols it leverages.

- **SOAP** [17] (Small Object Access Protocol) is used as the communication protocol for XML web services.
- **WSDL** [25] (Web Services Description Language) is the counterpart to the IDL in CORBA [23]. A WSDL document is a XML document that describes the interfaces of a web service, the location of the service, the protocol needed to access the service etc. In short, WSDL provides all the necessary information to access the web service.
- **UDDI** [26] (Universal Discovery Description and Integration) is the counterpart of CORBA’s trading service for XML web services. It allows a service provider to publish his services to the UDDI registry and later allows the service consumers to discover the published services and use them.

UDDI provides three kinds of search:

- White Page search by name
- Yellow Page search by categories based on standard taxonomies
- Green Page search by technical details of a service interface

2.4.4 Middleware challenges in mobile ad hoc networks

Conventional middleware platforms as mentioned above assume relative static network topology, reliable channels and so forth. MANETs with their special characteristics have, however, posed several new challenges to the middleware technology [27].

- Current generation of middleware is, to a large extent, heavy weight and inflexible, which are too bloated to be ported to the small, often resource-constrained devices participating in MANETs.

- Due to the dynamic changing topology in MANETs, complete transparency of the underlying network may not always be desirable. Many applications may have to adapt to the fluctuation in network resources or the change in location. However, no existing middleware facility has addressed the problem of transparency degree.

- Due to the unpredictable and frequent disconnection in MANETs, communications should be allowed to proceed even in the absence of connection and allows for seamless reconnection. So event-based middleware that support non-blocking/asynchronous communication and publish-subscribe platform will be desired.

- Service discovery should not rely on central registries, since nodes function as central registries might leave the network or become inaccessible due to a sudden network partition. This is an issue addressed by this thesis.

2.5 Service discovery in MANETs

Most of the existing service discovery protocols mentioned above are not specially tailored for MANETs. When designing service discovery protocols for MANETs, one should take into consideration the special characteristics of the MANETs.

Infrastructure-less

Service discovery protocols for MANETs should not reply on any fixed infrastructure. A central register is widely used in many of the aforementioned service discovery protocols, like DAs in SLP [7], JLS in Jini [10] and SLM in Salutation [11]. If central registers should be used in MANETs, they should provide only simple functions so that almost every node with sufficient capacities (i.e. processing power, memory space, battery life etc.) will be able to take on roles as central registers. However, if it requires that nodes functioning as central registers should possess special functions (e.g. the lookup server in Jini [10] has to manage the objects for accessing and the SLM in Salutation [11] has to manage different communication media), it is hardly possible to automatically relocate these functions to other MANET nodes. Thus these nodes form a kind of infrastructure that is inappropriate for MANETs.

Dynamic
MANETs are dynamic in nature. Nodes can join and leave the network at will. Nodes might fail due to, for example, battery failure. Links between the nodes might break due to nodes mobility. All these make the distributed service discovery architectures more appealing to the MANETs.

Heterogeneous
Because of the heterogeneity of the nodes in a MANET, not all the nodes possess the same processing capabilities, battery lives or memory storages. Service discovery architectures should not require additional software to be implemented on every MANET node. In Jini [10] technology, it assumes a running JVM on every network device. Bluetooth SDP [20] [21] depends on a uniform radio technology. These will render extreme difficulties for a MANET with hundreds of heterogeneous nodes.

All the aforementioned service discovery protocols have assumed a routed network. In a MANET, service discoveries will cause extra control messages by the routing protocols. Recent researches have moved towards finding ways to promote co-operations between layers to reduce the overhead caused by repeating similar tasks at various layers. There are for example a lot of similarities between the route discovery in a reactive routing protocol and the service discovery. In this thesis, optimizations are done between the service discovery mechanism and the reactive routing protocols in order to reduce the overall routing overhead. In addition, nodes that function as central registers do not require special functions to be implemented. They only need enough memory spaces to hold the service information and a simple mechanism to look up the information. The effect of having central registers in the network will be discussed through simulations.

Chapter 3
Related works

Several research efforts have been made to propose some suitable service discovery mechanisms for mobile ad hoc networks; we briefly review some of them in this chapter.

3.1 Service discovery architectures

C. K. Toh [28] has in his book Ad Hoc Mobile Wireless Networks outlined different service discovery architectures for managing service information on MANETs, namely service coordinator based, distributed query-based and hybrid service location architectures.

Service coordinator based architecture: Certain nodes in the MANET are chosen to be the service coordinators, a role quite similar to DA in SLP [7] or lookup service in Jini [10]. SCs announce their presences to the network periodically by flooding SC announcement messages. Service providers that receive SC announcements register periodically their services and access information with SCs in their surroundings. A service requestor will choose one service coordinator to be its affiliated SC among all the heard SCs, and it will contact its affiliated SC for desired services.

Distributed query-based architecture: This architecture contains no SCs. Instead, a service requestor floods the service requests throughout its surroundings in the network. Each node that wants to provide services runs its own service discovery server and responds to service requests for its own services.

Hybrid service location architecture: This architecture combines the above two architectures. Service providers within the announcement scopes of SCs will register with them their available services and access information. Service requestors with affiliated SCs will query SCs for services, or simply broadcast the query in the absence of affiliated SCs.

This thesis evaluates the performance of the latter two service discovery architectures on reactively routed MANETs.

3.2 Group-based Service discovery Protocol for MANETs

D. Charkraborty et al. proposed a novel group-based service discover protocol (GSD) [29] for MANETs. The protocol is based on peer-to-peer caching of the service advertisements. Every service advertisement is associated with an advertising radius in terms of hops. Thereby, every node will be able to maintain a cache of all the services within the advertising radius. Services are described using service groups (e.g.

Service/Hardware/IO-Service/Printer-Service). The local cache will be exploited first when a service is requested at the application level in order to enhance efficiency for service discovery. When no matching service is found in the local cache, a service request will be broadcasted to the network.

D. Charkraborty et al. have also proposed a group-based selective forwarding concept for such broadcasted service requests. A service request is only forwarded to those nodes that have seen in their vicinity one or more of the service groups specified in the request. This information about service groups in the vicinity is conveyed through the periodic service advertisements. In this way, the network will not be inundated with request messages, and the bandwidth usage will be spared.

The simulations done in GSD [29] have left out the centralized entities. Each node instead maintains a service cache itself.

3.3 Name Resolution and Service Lookups in on-demand MANETs

A solution to name resolution in on-demand MANETs has been proposed in [32] [33]. The main idea is to streamline name resolution with the underlying reactive routing protocol (e.g. AODV [3], DSR [4]). The objective is to obtain a bandwidth-efficient scheme that reduces the number of broadcasted discovery messages to a minimum.

It has also been proposed to bundle simple service name lookups together with this name resolution mechanism ([31]). This is parallel to DNS SRV lookups for simple service discovery on the fixed Internet [35]. It allows a service name to be resolved into an IP address and a transport protocol number to be used to initiate the service. The transport protocol type is normally encoded into the service name.

3.4 SLP-based service discovery on MANETs

R. Koodli et al. has in their Internet draft [30] proposed a similar solution to service discovery in on-demand MANETs. Here, service discovery requests and replies are carried as an extension to route requests and replies in a similar way. The proposed mechanism for service discovery specifies the message formats that are designed to inter-operate with the Service Location Protocol (SLP) [7]. Thus, it has more capabilities to accommodate advanced service discovery than the DNS-SRV-based scheme for simple service name resolution proposed in [31] has. A drawback, however, is that it requires additional software implemented on the MANET nodes, which may increase complexity and slow deployment. The proposed scheme is a distributed query-based architecture.

3.5 What lacks

Güichal [34] undertakes an analysis of different service discovery architectures based on simulations. The work concludes that the hybrid architecture normally outperforms both the service coordinator based and the distributed query-based approach. The distributed query-based architecture is the second best choice, and yields less messaging overhead. Despite this, the work concludes that the hybrid architecture gives an overall better performance, because it yields higher service availability.

A shortcoming of the simulations is that they do not take the importance of underlying routing into consideration. This assumption might be appropriate when a proactive routing protocol is being used, because with proactive routing the traffic patterns and service discovery search patterns do not influence the amount of routing messages. With a reactive routing protocol, on the contrary, this assumption does not hold, and the simulation results are not applicable. Data traffic will trigger messaging by the reactive routing protocol. Hence, service discovery messages will increase the routing overhead.

The hypothesis of this thesis is that the routing overhead would be much higher with the hybrid architecture than with the distributed query-based distributed, simply because the hybrid architecture proved to require more messages on the network. Since the service discovery mechanism have an influence on the reactive routing protocol, this thesis use the optimization methods proposed in [30] and [31] to reduce the overall routing overhead. A new comparison is made in this thesis between the hybrid and the distributed query-based architecture on reactively routed MANETs in terms of service availability, message overhead and latency.

Chapter 4
Service discovery architectures and mechanisms on a reactively routed MANET

This chapter discusses service discovery architectures, mechanisms and other related issues specific to this research. However, they also apply generally.

4.1 Roles of nodes

In terms of service discovery, each MANET node may take one or several of the following roles:

- A **client (or Service requestor)** is a node that wants to discover a type of service.
- A **Server (or Service provider)** refers to a node that wants to make its services discoverable by other nodes.
- A **Service Coordinator (SC)** is a node that assists with service discovery. It holds a central repository for caching **Service Bindings** (A service binding maps the service type to an IP address and a port number that can be used to initiate the service.).

4.2 Service discovery architectures

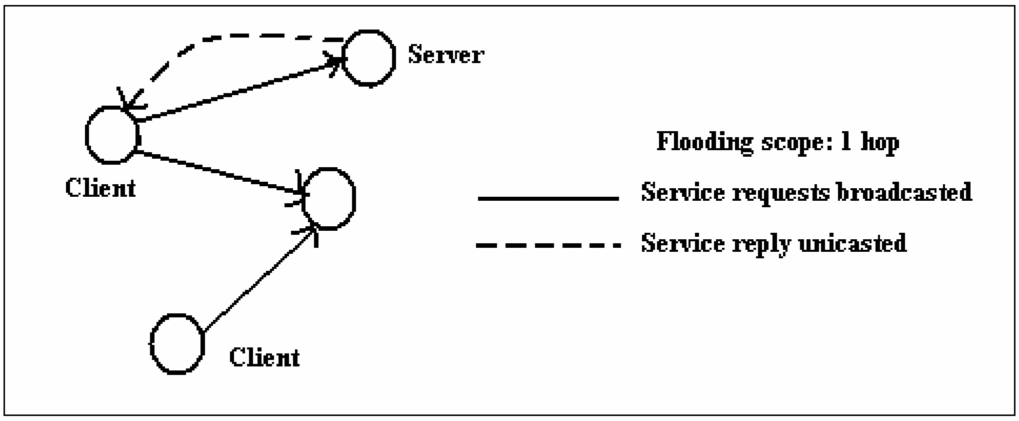

Figure 2: Pure flooding service discovery architecture

The service discovery architectures mentioned in section 3.1 on page 25 apply regardless of the underlying routing protocol, thus apply here too. The service coordinator based architecture is not explored here, because Güichal [34] has showed that it is inferior to both the hybrid architecture and the distributed query-based architecture. Our hypothesis is that the effect of a reactively routing protocol works in favor of the distributed query-based architecture. Thus, in this thesis, we focus on the distributed query-based architecture as shown in Figure 2 on the previous page, referred to as pure flooding henceforth, and the hybrid architecture as shown in Figure 3 below.

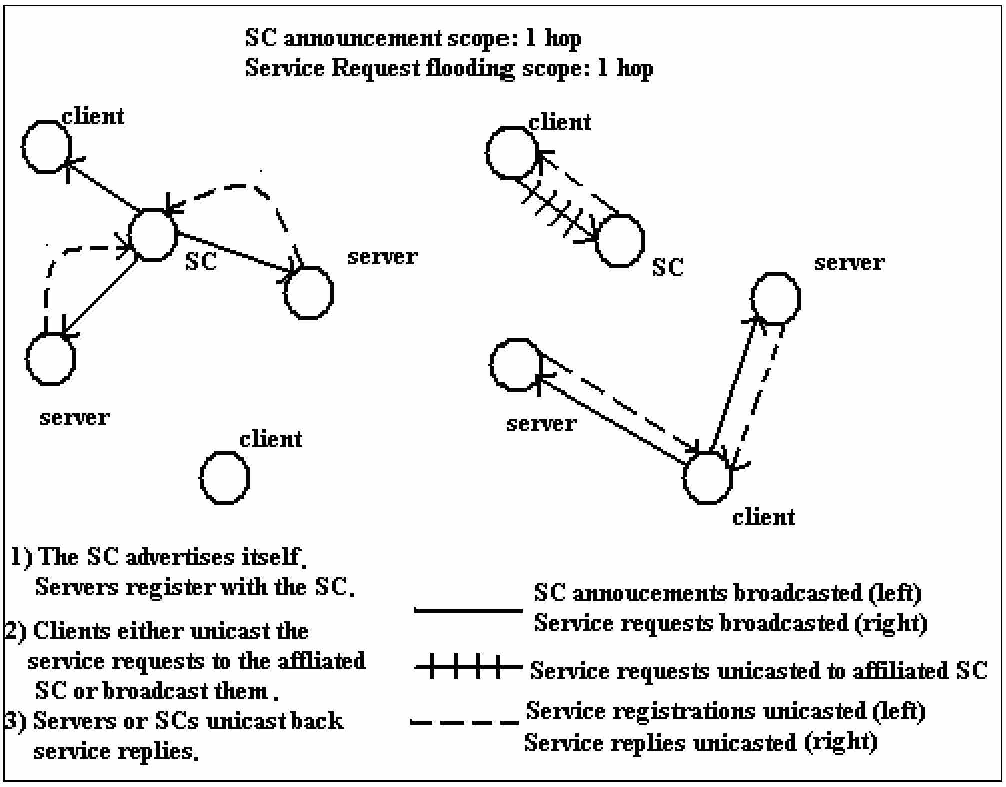
SC announcement scope: 1 hop
Service Request flooding scope: 1 hop
client
SC
server
server
client
client
SC
server
server
client
1) The SC advertises itself. Servers register with the SC.
2) Clients either unicast the service requests to the affliated SC or broadcast them.
3) Servers or SCs unicast back service replies.
SC annoucements broadcasted (left)
Service requests broadcasted (right)
Service requests unicasted to affiliated SC
Service registrations unicasted (left)
Service replies unicasted (right)

Figure 3: Hybrid service discovery architecture

4.3 Message types

The service discovery mechanism includes the following messages:

Service Coordinator Announcements (exist only in the hybrid architecture): Periodically, service coordinators will broadcast announcements to inform the surroundings of their presences. Every service coordinator is associated with an announcement diameter in terms of hops, referred to as SC announcement scope henceforward. An SC will not relay other SC's announcements unless they cover a bigger range than it self does. Nodes within the announcement scope on hearing the SC announcements will cache all the SC contact information encapsulated in the messages. A service requestor will choose one among all the heard SCs as its affiliated SC to which it will direct service requests and this choice is remade every time SC announcements are received. The cached SC information entries are time stamped and will be purged on expiration if no SC announcements are received for the last two SC announcement interval.

Service Registrations (exist only in the hybrid architecture): Servers on hearing the SC announcements will register their own services with **ALL** the heard SCs. These registrations take place immediately after the receipt of the SC announcements. Service bindings contained in the registration packets are also time stamped while being cached at SCs and will be purged on expiration if no service registrations are received for the last two SC announcement period.

Service Request:
Pure flooding architecture: A client in search for a service will simply broadcast the service request to the network. The broadcast scope is limited by a parameter called flooding scope.

Hybrid architecture: A client in search for a service will direct its service request to its affiliated service coordinator. If it so happens that the affiliated service coordinator is beyond reach (left the network, power failure etc.), then the client will proceed with the pure flooding scheme for the current service request i.e. broadcast the service request. At the same time, client will choose, if exists, a new affiliated SC among all the other heard SCs for future service requests. If service coordinator's reply is negative i.e. there is no required service registered in its repository, then client will also proceed with pure flooding scheme for the current service request but no new affiliated SC will be chosen. The client will still stick to its old affiliated SC. If there are no service coordinators heard by the client at all, the client will simply fall back on the pure flooding approach for service discovering.

Service Reply:
Pure flooding architecture: Only servers that offer the desired service will initiate a service reply to the requesting node. It is client's responsibility to choose the best among all the replied servers to contact with for the desired service.

Hybrid architecture: Both SCs and servers can respond to the broadcasted service requests that are not directed to a specific SC if they can provide or have registered matching services. An SC is obligatory to respond to a service request destined for it from the client no matter whether there have been registered any matching services or not. If no matching services exist, the SC will simply send back a negative service reply. An SC will provide the client with all the matched service bindings, either its own or registered by other servers. It is up to the client to decide with which server to establish further contact.

4.4 Relation to reactive routing protocols

Figure 4 on the next page shows how service discovery can be streamlined with the reactive routing protocol in the case where client 1 is affiliated with a service coordinator, while client 2 is not. This is the model used for simulation in this thesis.

The underlying reactive routing protocol used in the simulation is AODV [3]. The service discovery messages are carried by the routing protocol messages as extensions in the form of a type and a type-specific value as being proposed in the AODV specification [3]. Service requests and SC announcements are carried in RREQ extensions, while service replies and service registrations are carried in RREP extensions.

The type and type-specific value for all the four aforementioned messages are listed in Table 1 on the next page.

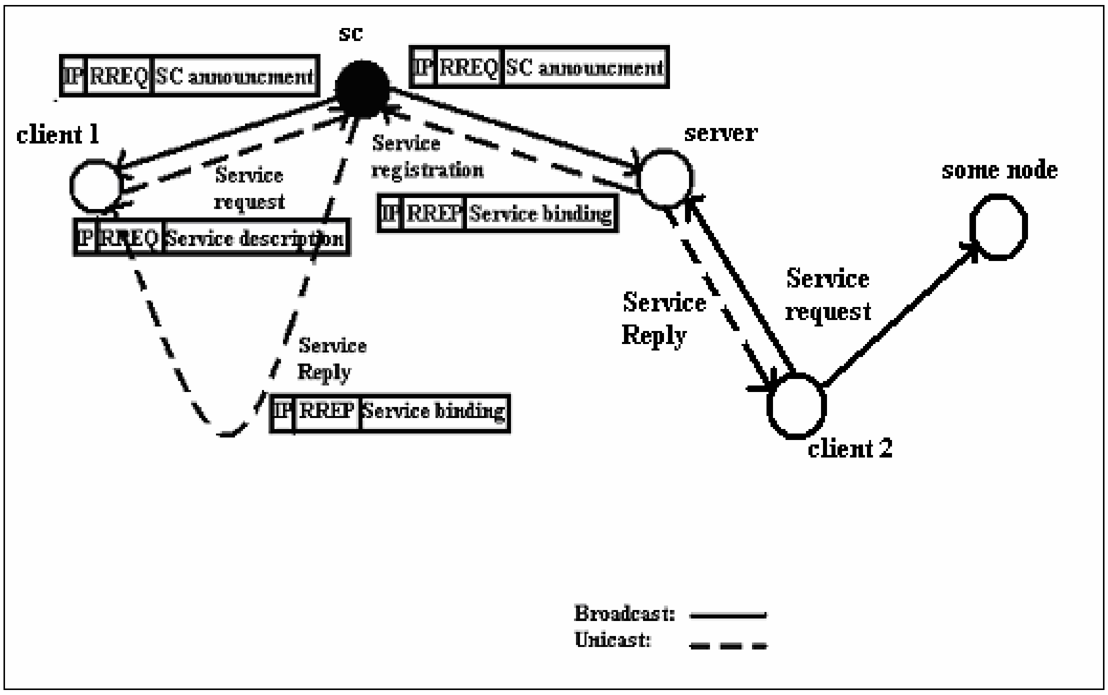

Figure 4: Service discovery model used in the simulation

(SC announcement scope: 1 hop, Service Request flooding scope: 1 hop)

Message type	Type	Type-specific value
Service request	5	Service description
Service reply	6	Service binding
SC announcement	7	SC announcement scope
Service registration	8	Service binding

Table 1: Message types and type-specific values used in simulations

The advantages of piggybacking service discovery on routing messages in this way are as follows:

1. Reverse routes to the service requestor are established along with the service request so that no additional route discovery is necessary to relay the service reply back to the service requestor.
2. Forward routes to the SC are established along with the SC announcements so that service requests and service registrations can be unicasted to the SC.
3. Forward routes towards the server will be built along with the service reply in pure flooding architecture, thus no additional route discovery is needed for further communication with the server (e.g. sending data etc.) In hybrid architecture, forward routes towards the server from the SC will be built along with the service registration so that SC might be able to reply to the route request on behalf of the server itself, which helps in reducing the flooding scope of route request.

One requirement for the nodes running this amended routing protocol is the ability to process AODV message extensions.

4.5 Service Coordinator placement

The placements of the clients, servers and service coordinators in a network can be generally divided into four categories, see Figure 5 through Figure 8. These figures will illustrate the relation between the placement of a service coordinator in the network and its contribution to the network performance.

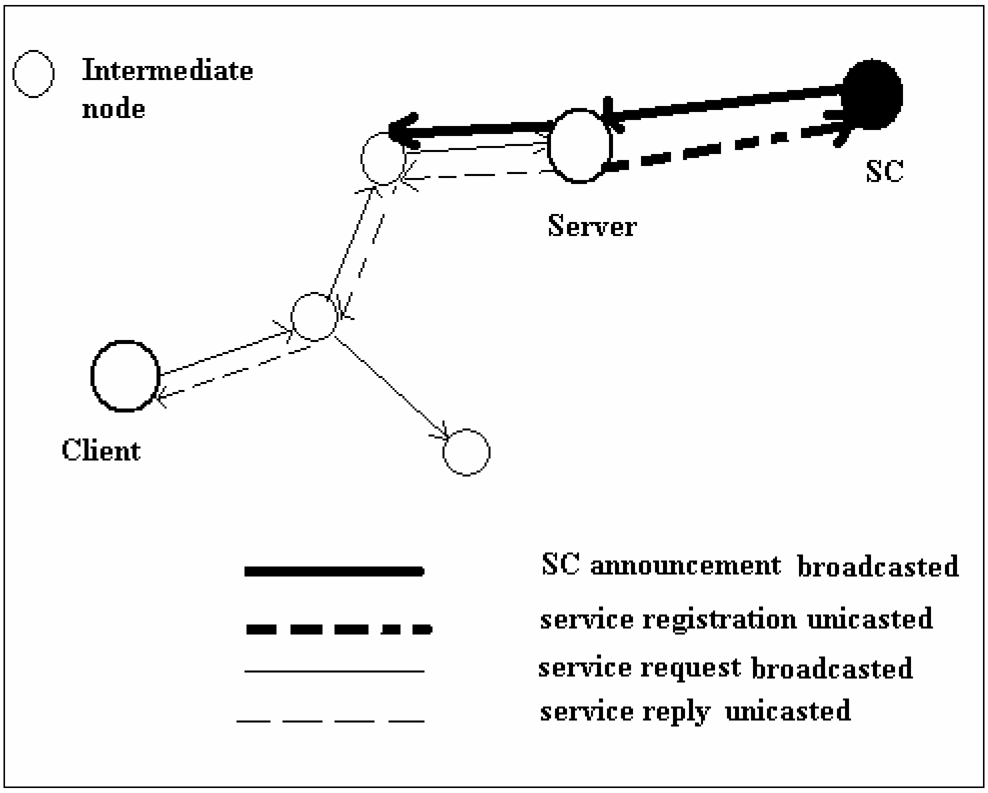
Intermediate node
SC
Server
Client
SC announcement broadcasted
service registration unicasted
service request broadcasted
service reply unicasted

Figure 5: SC placement 1 (client – server – SC)

Figure 5 above illustrates the situation where the server is closer to the service coordinator than the client is. With other words, the server is able to receive the SC announcements but not the client. Figure 6 on the next page illustrates the situation where the client is closer to the service coordinator than the server is. With other words, the client is able to receive the SC announcements but not the server. In both Figure 5 and Figure 6, using the hybrid service discovery architecture will be very unreasonable. The service coordinator is totally superfluous since it is useless to the client. In Figure 5 above, the client has to broadcast the service request since no SC is heard by it. All the SC announcements and server registrations will be merely a waste of the network bandwidth and nodes' processing power for nothing. In Figure 6 on the next page, though the client can unicast the service request to its affiliated SC, yet the server cannot receive the announcements broadcasted by the service coordinator. Accordingly, no service is registered at the SC and the client has to fall back on the pure flooding approach by broadcasting the service request to the network upon receiving the negative reply from its affiliated SC. Of course, it is possible to increase the SC announcement scope in both Figure 5 and Figure 6, so that the client in Figure 5 and the server in Figure 6 can receive SC announcements and be able to unicast the service request or the service registration to the service coordinator respectively. Yet, doing so is not very attractive. Firstly, increasing the SC announcement scope will increase message overhead significantly due to the fact that SC announcements are broadcasted periodically. Secondly, from the figures we can see that if the server is resolved at the SC, no route to the server will be available upon receiving the service reply. An extra route discovery will be needed. Simulation results presented in later chapters will show the effect of increasing the SC announcement scope.

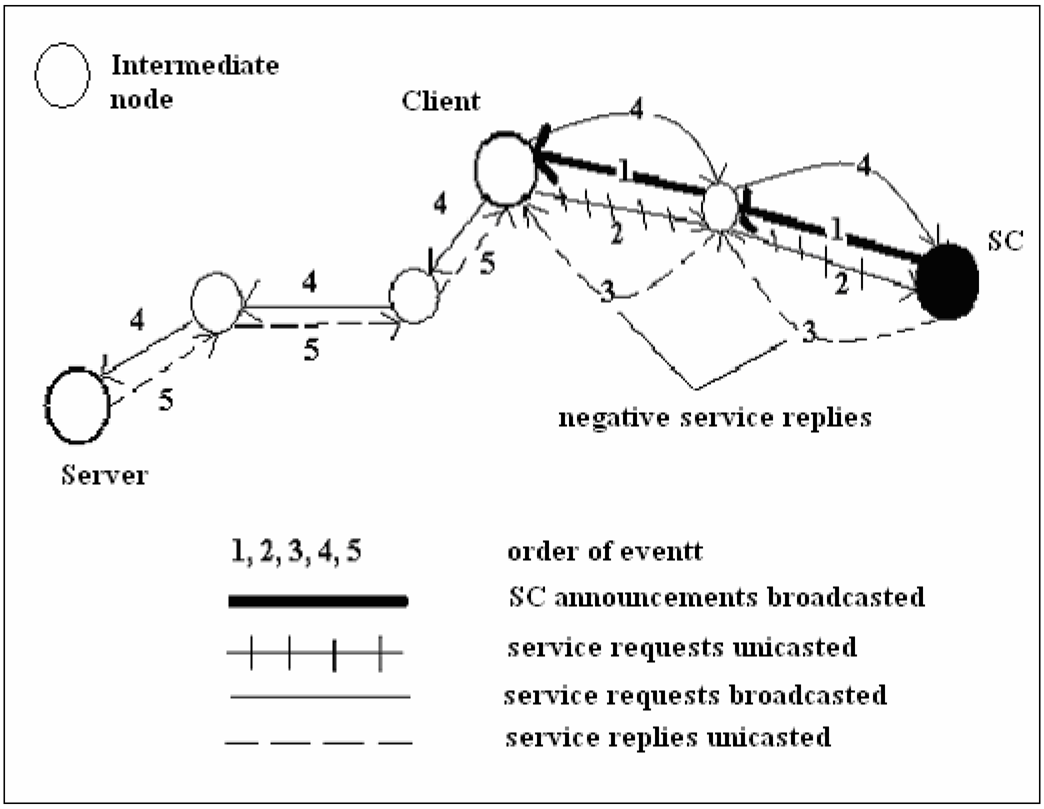
Intermediate node
Client
SC
Server
4
1
2
3
5
negative service replies
1, 2, 3, 4, 5 order of eventt
SC announcements broadcasted
service requests unicasted
service requests broadcasted
service replies unicasted

Figure 6: SC placement 2 (server – client – SC)

Figure 7 on the next page illustrates the "**everyone sees everyone"** situation. Both the client and the server can receive the SC announcements. In addition, the client and the server can reach each other by a route with no service coordinators involved and this route is much shorter than the route with the service coordinator. Firstly, the hybrid approach will not increase the service availability since the server can also be reached through pure flooding. Secondly, if the hybrid approach is used, no route to the resolved server will be available on receiving the service reply from the service coordinator as opposed to the pure flooding approach where forward routes are established along with the service reply from the server itself. Thirdly, since the route between the client and the server with no SC involved is much shorter, it will be easier simply to broadcast the service request and spare the periodic SC announcements and service registrations.

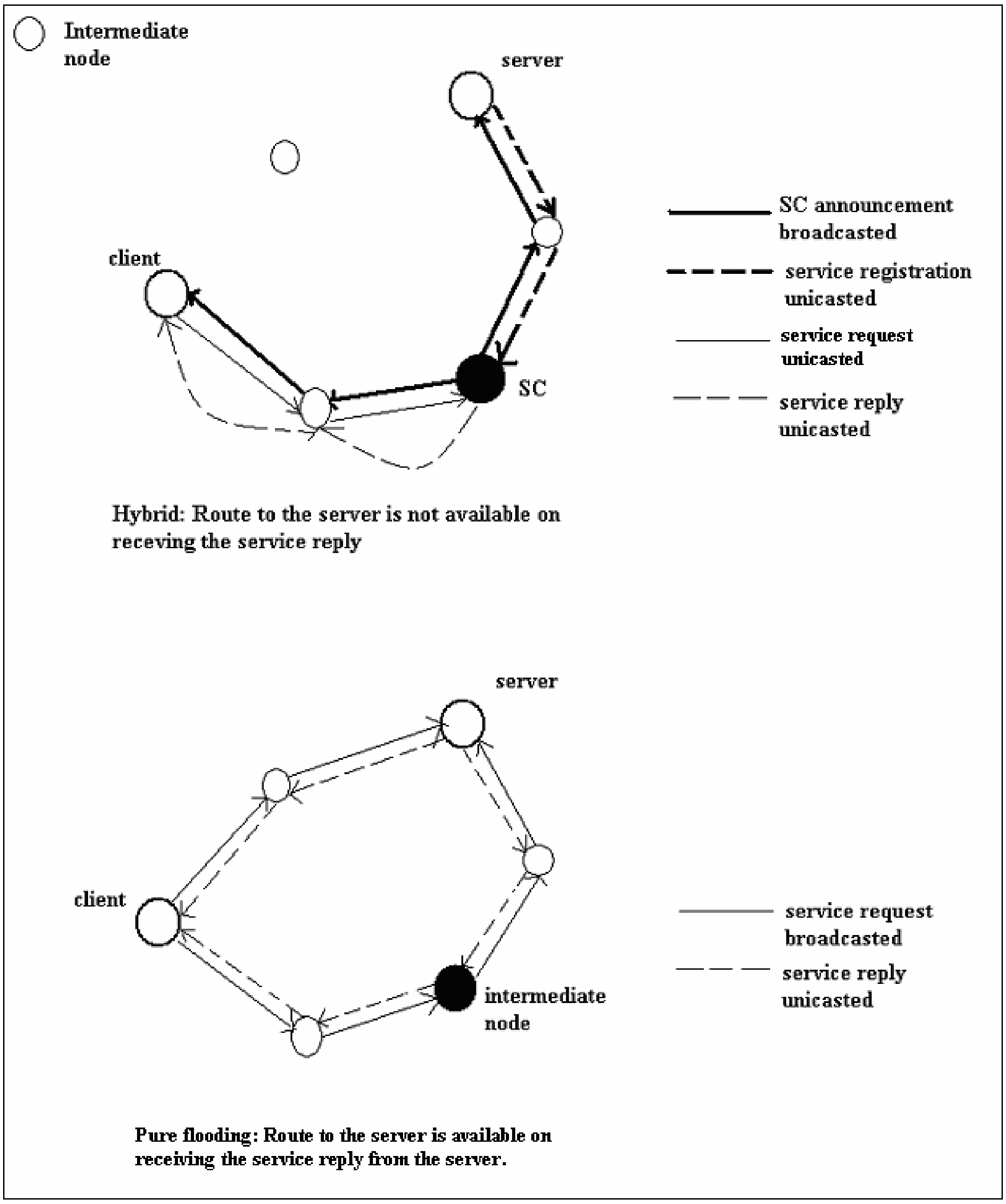
Intermediate node
server
client
SC
SC announcement broadcasted
service registration unicasted
service request unicasted
service reply unicasted
Hybrid: Route to the server is not available on receving the service reply
server
client
intermediate node
service request broadcasted
service reply unicasted
Pure flooding: Route to the server is available on receiving the service reply from the server.

Figure 7: SC placement 3 (everyone sees everyone)

Figure 8 on the next page seems to be the only deployment that makes the service coordinator appear useful. The upper part of the figure shows the situation where the hybrid approach is used and the lower part shows the situation where the pure flooding approach is used. The service coordinator in this deployment does allow service requests to be unicasted instead of broadcasted. In addition, the hybrid architecture may increase the service availability if the server is outside the client's service request flooding scope. For example, if the flooding scope for the service request is set to three hops in Figure 8, then the client will not be able to find the server using the pure flooding approach.

Again we can see from the figure, in the pure flooding approach, after the discovery of the server, the route to the server is also established, so no additional route request is necessary to access the sever. While in the hybrid approach, no route is established to the resolved server. Accordingly an extra route discovery is needed. In addition, if the service requests are relative seldom, many of the network capacity will be wasted in relaying SC announcements and server registrations. Even with a high service requests frequency, we can go for other alternatives than using SCs, for example to cache the service bindings at the client node or intermediate node in order to minimize the overhead.

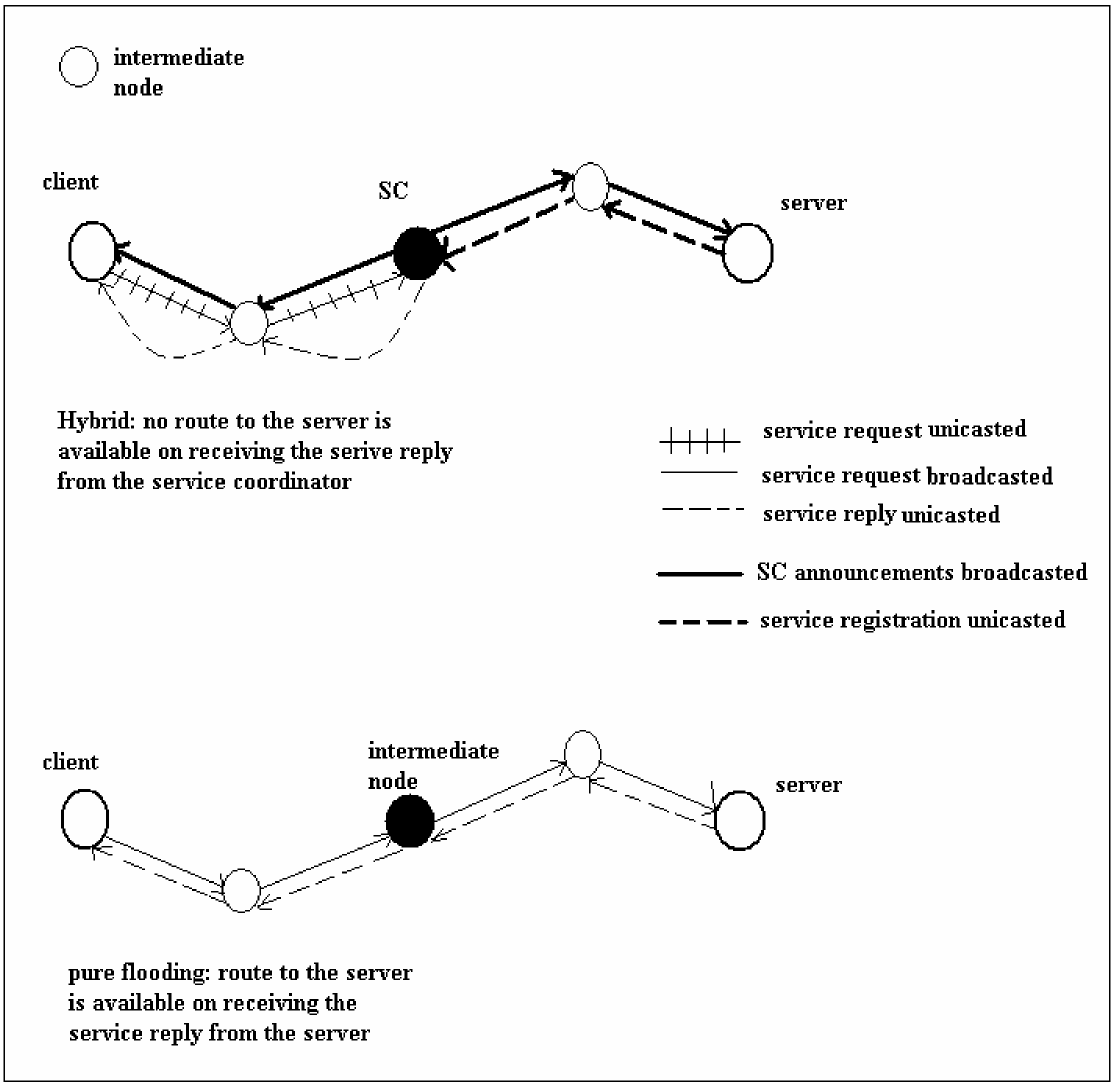
intermediate node
client
SC
server
Hybrid: no route to the server is available on receiving the serive reply from the service coordinator
service request unicasted
service request broadcasted
service reply unicasted
SC announcements broadcasted
service registration unicasted
client
intermediate node
server
pure flooding: route to the server is available on receiving the service reply from the server

Figure 8: SC placement 4 (client – SC – server)

Discussion

The unpredictable and dynamic MANET topology makes least guarantee for the actual placement of different nodes. As discussed above, only one category will possibly show the benefit of adding service coordinators. The overall network performance after adding service coordinators is very doubtful. The simulations done in this thesis are based on random topologies, which might incorporate any of the aforementioned placements. These simulations aim to find out whether implementing service coordinator functionalities to the network will yield a better network performance over the pure flooding approach.

One possible solution to make service coordinators useful might be a dynamic SC election mechanism. Instead of statically assigning the service coordinator role to certain nodes, a lightweight, dynamic SC election mechanism can be implemented in every node participating in the ad hoc network communication. Any node may take on the role as a service coordinator based, for example, on its capacity (e.g. memory, processing power, battery etc.) and its instant network environment (e.g. the number of servers, service coordinators, potential clients etc.). However, SC election mechanism is out of the scope of this thesis.

Chapter 5
Simulation Setup

The simulations were done on the well-known simulator GloMoSim [36], which is shipped with an AODV module.

The simulated network contains 50 nodes randomly located in a 300x300m square. A two ray propagation model for radio waves as well as omni-directional antennas were used at the physical level. The radio range of the nodes is set to 50 meters. The mac protocol used is IEEE 802.11. AODV and UDP are used as the underlying reactive routing protocol and transport layer protocol respectively. There are two different types of services in the network. A node is selected as a client, a server and/or a service coordinator based on the density parameter fed in through the configuration file (see Appendix B). The selection was generated using a random number generator shipped with GloMoSim [36]. SC election mechanism is out of the scope of this thesis.

The mobility model used for the dynamic topology is random waypoint. In the simulations, 20% of the nodes will function as clients and actively initiate service requests every 20 seconds. The time for the first service request is randomly and individually generated for every client node. Another alternative is to allow each client to initiate exactly one service request in the whole simulation period. The reason for not using this alternative is because it will definitely favor the pure flooding architecture over the hybrid architecture. Since every client will only do one service discovery, all control overhead generated by the service coordinators will not be justified. The SC announcement interval is set to be the same as the route timeout value (i.e. 10S) as recommended in AODV [3]. The reason for setting the SC announcement interval alike the route timeout value will be revealed in the next chapter.

The two service discovery architectures simulated are the pure flooding and the hybrid service discovery architecture as shown in Figure 2 on page 29 and Figure 3 on page 30. The architectures can be tuned with (at least) two parameters:

- **SC announcement scope**: This scope regulates the extent to which a service coordinator announcement can reach in terms of hops. This parameter is only used in the hybrid architecture.
- **Flooding scope**: This scope determines how far a service request will be broadcasted in the network in terms of hops. This parameter is used in both architectures. In the hybrid architecture, a service requestor will fall back to use a pure flooding approach by broadcasting the service request based on this flooding scope if no affiliated service coordinator is heard or when a negative service reply has been returned from the affiliated service coordinator.

The following metrics are defined to evaluate the simulation results**:**

- **Request Satisfied Ratio (RSR)**:

$$RSR = \frac{Number\ of\ \ positive\ service\ replies}{Total\ number\ of\ \ service\ requests\ issued\ by\ all\ clients\ in\ the\ network}$$

  A positive service reply means not only the resolution of a service type to a valid service binding (server address, port number), but also a successful contact to this server via the given access information (i.e. a route to the resolved server can be found).
- **Message overhead**: All the non-data messages that are transmitted in the network by all the nodes at the network level. The overhead is counted as the total number of packets over each hop (i.e. the total number of packets times the average number of hops traversed by the packets)
- **Broadcasted message overhead**: All the non-data messages that are *broadcasted* in the network by all the nodes at the network layer.

Simulations are done for both static and dynamic topologies. The simulation programs are written in C (see Appendix C for part of the codes) and every simulation is repeated 500 times with different seed values.

Chapter 6
An initial simulation with five nodes

The purpose of this five nodes simulation is to:

- Illustrate the effect of variable SC announcement frequencies in terms of broadcasted message overhead
- Illustrate the relation between the SC announcement frequency and the active route timeout value
- Fix the SC announcement interval for further simulations

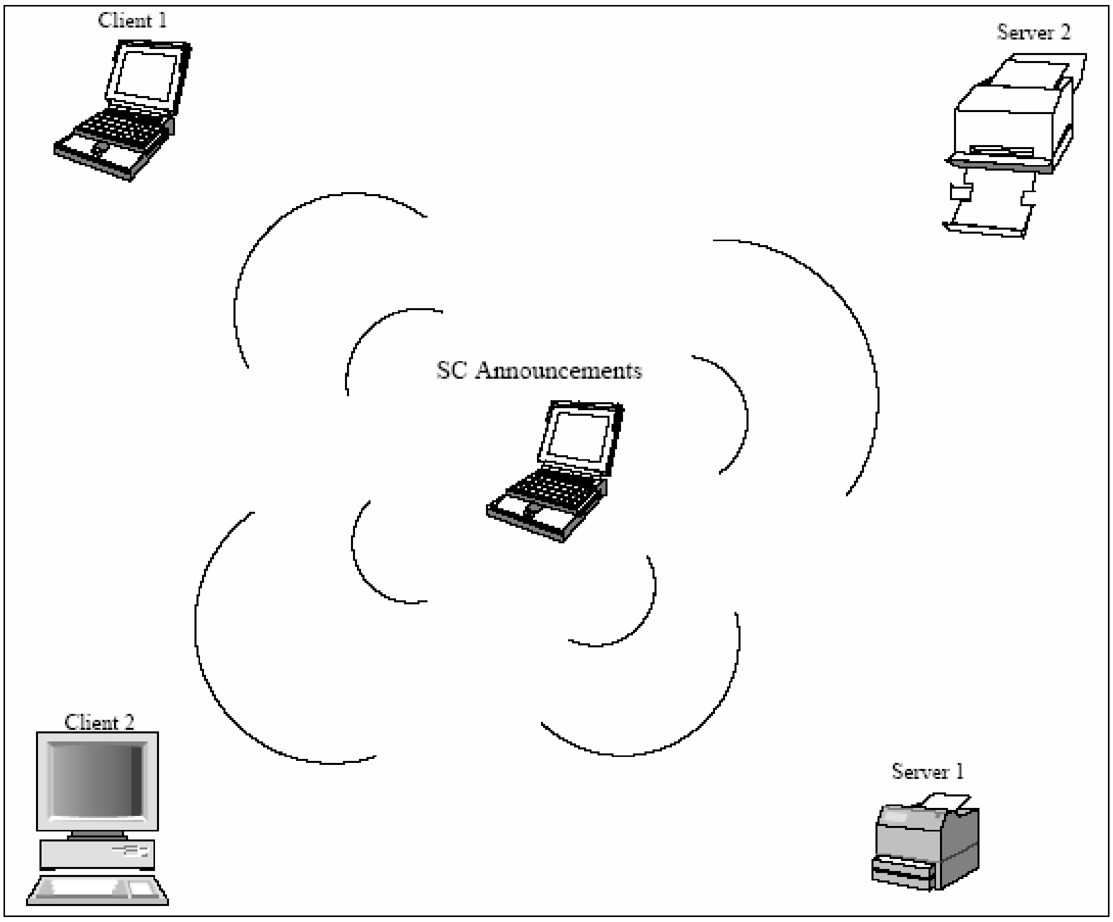

Figure 9: A simulation with five nodes

As illustrated in Figure 9 above, there are five nodes in this simulation, two clients, two servers and one service coordinator. Client 1 and client 2 are supposed to discover server 1 and server 2 respectively. Both clients and servers are within the transmission range of service coordinator but not each other.

As mentioned in section 4.4 on page 32, we choose to piggyback the service discovery on routing messages in order to optimize the overall performance. In addition, a successful service discovery is supposed to end up with a successful access to the resolved server. Accordingly, after every success service discovery, forward routes to the service coordinator and the server will be established if not already existed or updated (AODV [3]). In the following simulations, the service request interval is set to be slightly larger than the route timeout value so as to make sure that all the established routes from the earlier service discovery will be invalidated if not updated by other means. This is to focus on the effect of the SC announcement interval as will be illustrated in the following sections. The simulation parameters are illustrated in Table 2 below.

NET SIZE	100M x 100M
TRANSMISSION RANGE	10M
SIMULATOIN TIME	500S
MOBILITY	NONE
SERVICE REQUEST INTERVAL	10S ~11S, 15S ~ 16S
SC ANNOUNCEMENT INTERVAL	VARIABLE
ACTIVE ROUTE TIMEOUT	10S, 15S
ACTIVE SC TIMEOUT	2 * SC ANNOUNCEMENT INTERVAL
SC ANNOUNCEMENT SCOPE	1
SERVICE REQUEST FLOODING SCOPE	2
ROUTING PROTOCOL	AODV
NUMBER OF NODES	5

Table 2: Simulation parameters for a simulation with five nodes

6.1 Broadcasted Message Overhead vs. SC Announcements Interval

Figure 10 on the next page shows the relation between the broadcasted message overhead and the SC announcements interval. The steep down slope in the beginning of the curve is due to the reduction of announcement messages produced by the SC as its announcement frequency decreases. However, there is a turning point at 10 seconds. Figure 11 and Figure 12 on page 44 further expose the details about what actually happens around this turning point.

One of the reasons that caused this turn in the curve is because of the increase in service requests that have to be broadcasted to the service coordinator. This is due to the timeout of the route from the service requestor to its affiliated SC. As Figure 12 on page 44 illustrates, there are three major message types that have contributed to this variation in the curve, namely SC announcements, service requests broadcasted to the SC from the clients and usual route requests (from the client to the resolved server after the service discovery). As mentioned in section 4.4 on page 32 forward routes to the SC are established and updated along with the SC announcements. Also routes from the SC to the registering servers will be created and updated along with the service registrations that take place immediately after the receipt of SC announcements on the server side. The route timeout value is set to ten seconds (as recommended in AODV [3]). Accordingly, if an SC sends out announcements every ten seconds or less, all these aforementioned routes will be updated before their expirations. Thus, all the service requests from the clients can be unicasted to their affiliated service coordinators. In addition, the service coordinator can also respond to a route request on behalf of the server itself (service registration refreshes the route between the SC and the server), thus reduces the flooding scope of the route request. However, if the SC announcement interval is set to be larger than the route timeout value, then there will exist a time gap between the timeout of the route and the receipt of the next SC announcement or the next service registration. In the meantime, all service requests to the affiliated SCs will have to be broadcasted instead of being unicasted and the SC upon receiving a route request from the client for the resolved server may have to rebroadcast it. The lower the SC announcements frequency, the larger this time gap will be and the larger the risk of unavailable route to the affiliated SC when a client initiates a service request and unavailable route to the server at SC upon receiving a route request from the client for the resolved server will be. All these broadcasted messages, are them service requests or route requests outweigh the benefit of the reduction of SC announcements, thereby cause the curve to go upwards again.

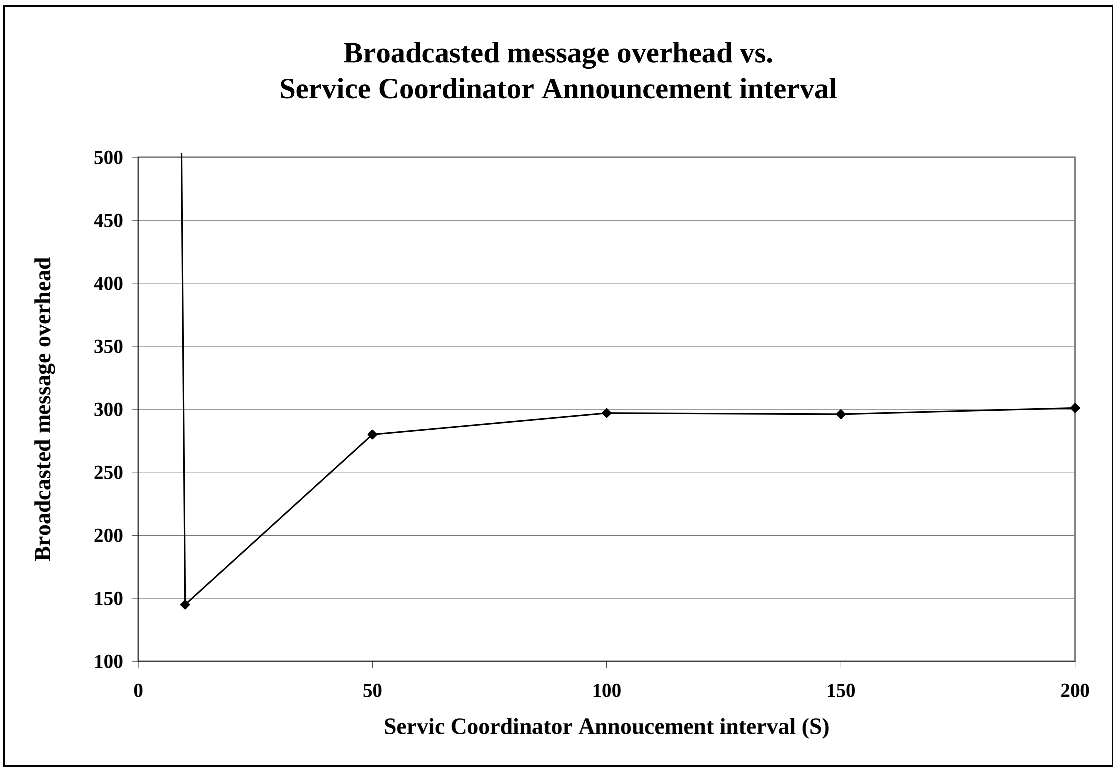

Figure 10: Broadcasted message overhead vs. SC announcement interval

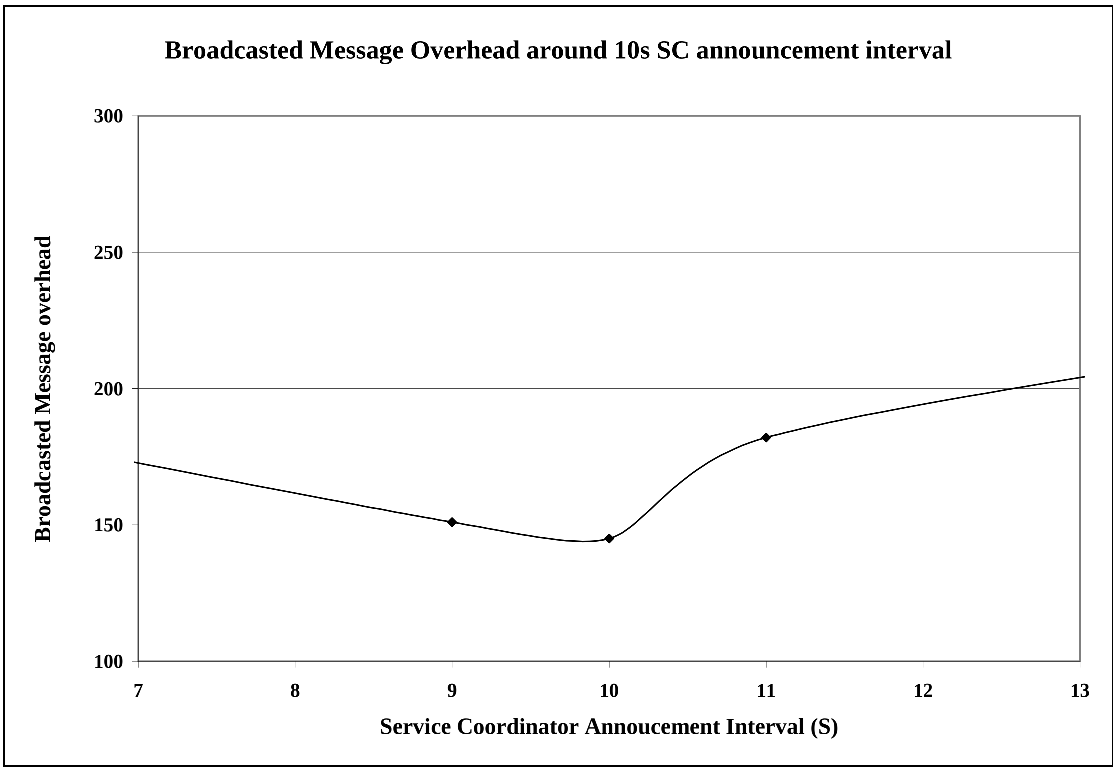

Figure 11: Broadcasted message overhead around 10s SC announcement interval

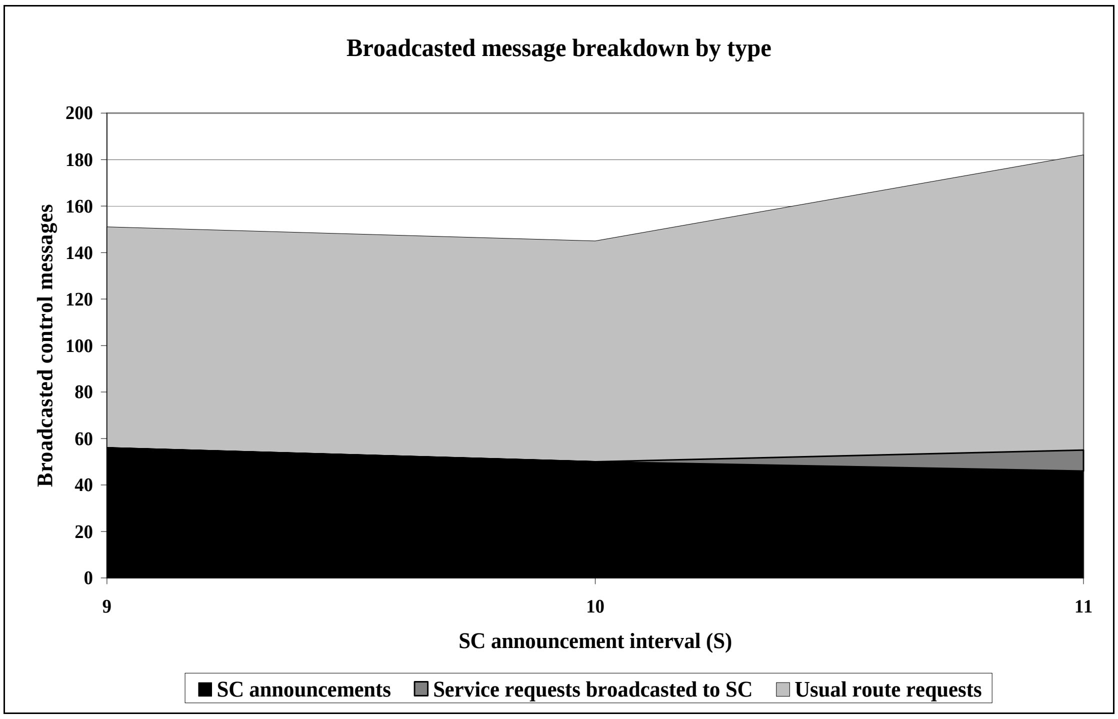

Figure 12: Detail of overhead by message type for the simulation with five nodes

6.2 Broadcasted Message Overhead relative to Active Route Timeout

As Figure 13 below illustrates, when the active route timeout varies, so does the turning point of the curve and at the turning point we get least message overhead per service request. Thus for the further simulation, **the SC announcements interval is set to be the same as active route timeout (i.e. 10 seconds) in order to minimize the overhead**.

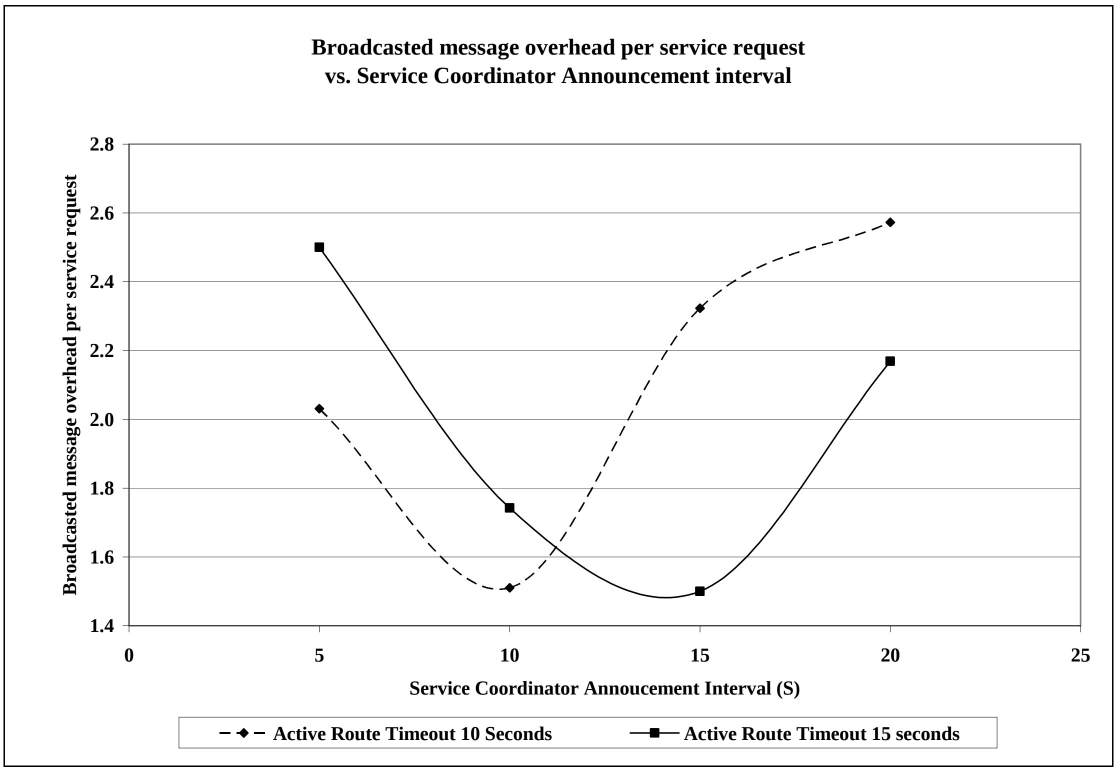

Figure 13: Broadcasted message overhead per service request vs. active route timeout

6.3 Discussion

The factor that is not considered when setting the SC announcement interval is the effect of the service request frequency. At a relative low service request frequency and static network topology, reducing the SC announcement frequency might reduce the overall message overhead. Considering the fact that it is hard to predict the actual service request frequency in a real ad hoc network communication, the extreme scenarios (i.e. very high or very low service request frequency) are excluded for this research. A dynamic topology might favor a higher SC announcement frequency in order to reflect the network dynamics. Considering the fact that the route timeout value of a reactive routing protocol is set taking the underlying network mobility into consideration, setting the SC announcement interval to be the same as route timeout value will be reasonable.

Chapter 7
Simulations with static network topologies

The purpose of the simulations in this chapter is to:

- Compare the performance between the pure flooding and the hybrid service discovery architectures in terms of service availability (i.e. RSR), message overhead and latency under the conditions of no node mobility
- Come to a conclusion about the preference of the two service discovery architectures based on the simulation results under the conditions of no node mobility

Simulation parameters are listed in Table 3 below.

NET SIZE	300M x 300M
TRANSMISSION RANGE	50M
SIMULATOIN TIME	500S
MOBILITY	NONE
ACTIVE ROUTE TIMEOUT	10S
SERVICE REQUEST INTERVAL	20S
SC ANNOUNCEMENT INTERVAL	10S
SERVICE REQUEST FLOODING SCOPE	1, 2, 3 HOPS
SC ANNOUNCEMENT SCOPE	1, 2, 3 HOPS
ROUTING PROTOCOL	AODV
NUMBER OF NODES	50
NODES POSITION	RANDOM
TYPE OF SERVICES	2
CLIENT DENSITY	20%
SERVER DENSITY	VARIABLE
SERVICE COORDINATOR DENSITY	VARIABLE

Table 3: Simulation parameters for static simulations

7.1 Hybrid architecture

7.1.1 RSR relative to server density and SC density

As Figure 14 on the next page shows, the RSR increases as more and more nodes take on roles as servers or (and) service coordinators. However, the increase in total number of servers exhibits a higher impact on the RSR than the increase in total number of service coordinators does.

The RSR is improved by approximately 0.5 in value when the server density increases from 5% to 40% for all SC densities. As to the increase of SC density, for a server density of 20%, the improvement in RSR is 0.023 in value when the SC density increases from 10% to 20%, 0.013 when the SC density increases from 20% to 30% and 0.008 as the SC density increases from 30% to 40%, an overall improvement of merely 0.044 in value. We can see from Figure 14 below that the curves for different SC densities almost overlap with each other.

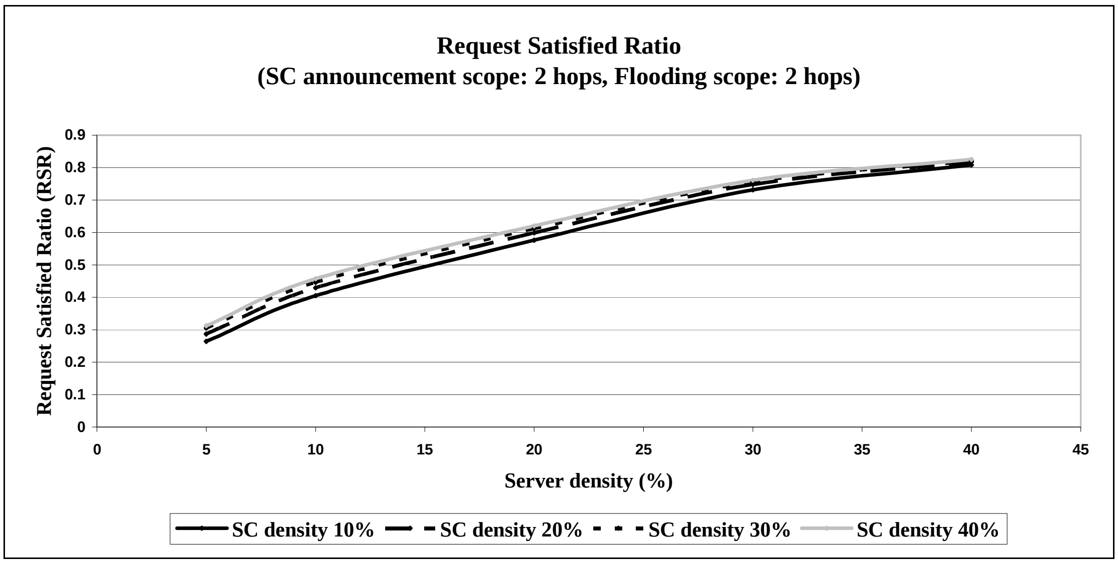

Figure 14: RSR relative to SC and server density for the static network topology (Hybrid)

One of the reasons for this almost negligible improvement in the RSR as SC density increases is that as more and more nodes take on roles as SCs, many may have their impacts on overlapping areas. However, the client will still direct its service request to its old affiliated SC unless either the new one is better compared to the old one based on certain criterion (less hop count etc.) or the old one fails in one way or another. This is better explained in Figure 15 on the next page. In Figure 15, SC 1 and SC 2 have overlapping effecting areas. Client 1 and client 2 will still direct their service requests to their old affiliated service coordinator SC 1. In this case, the presence of SC 2 is redundant. There may exist many such service coordinators, which are just present in the network without actually participating in the service discovery process. Hence they contribute nothing to the improvement in the RSR. However, these service coordinators will still consume a lot of network bandwidth by periodically broadcasting SC announcements and receiving service registrations.

This phenomenon, on the other hand, tells us again as already mentioned in section 4.5 on page 33 how essential the placement of service coordinators in the network should be if they are meant to increase the network performance.

The flattening of the curves at higher server densities is due to a similar reason. As more and more nodes take on roles as servers, nodes that offer the same type of service will register with the same service coordinators or have the same influencing area, which doesn't necessarily improve the RSR.

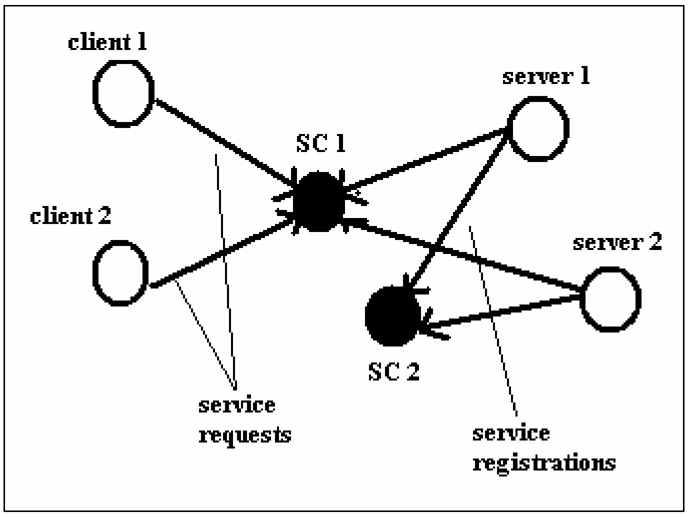

Figure 15: Two service coordinators with overlapping charging areas

7.1.2 Message overhead relative to server density and SC density

Usually, every thing good comes with the bad. The downside of the improved RSR is the increased message overhead as shown in Figure 16 below. At a server density of 20%, the RSR is increased from 0.612 to 0.62 as the SC density increases from 30% to 40%. Along with this negligible improvement in RSR, message overhead is however increased from 5433 to 6118. Comparing the increase ratios, the increase ratio of the message overhead is 11% higher than that of the RSR.

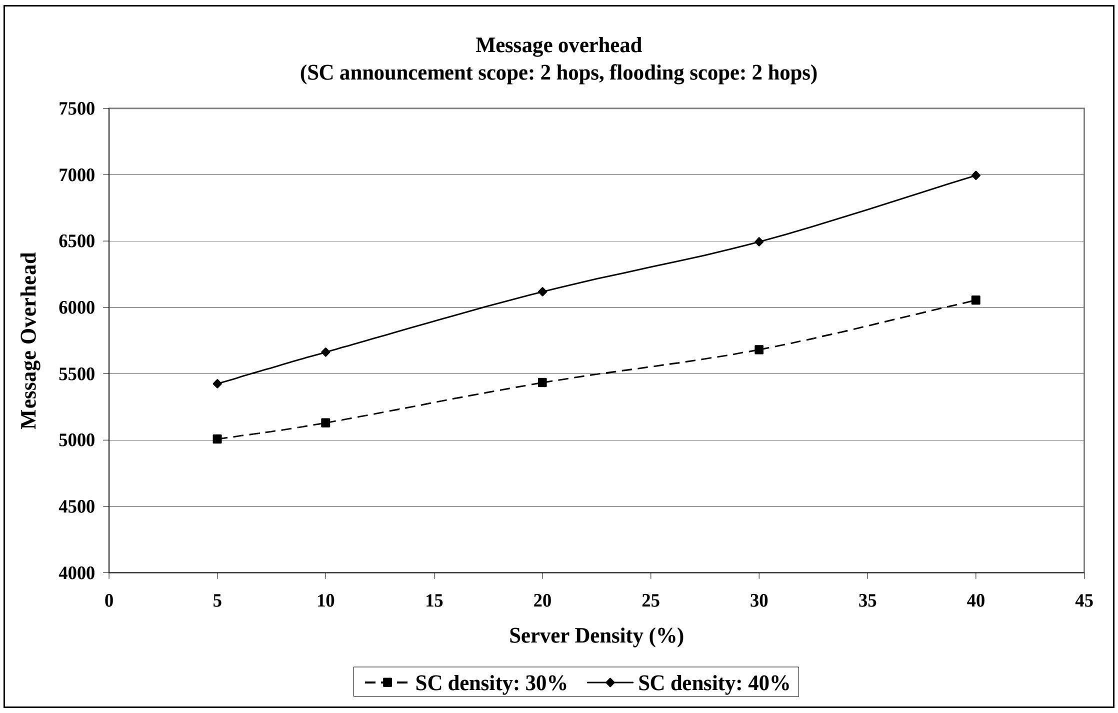

Figure 16: Message overhead relative to SC and server density for the static network topology (Hybrid)

However, when the server density increases from 5% to 40%, the RSR is improved from 0.305 to 0.82 and from 0.312 to 0.825 for an SC density of 30% and 40%, respectively. With this 0.5 increase in the RSR value, the message overhead is increased from 5007 to 6055 and from 5424 to 6995 for the two SC densities, respectively. The increase ratio of the message overhead is, however, 55% and 51% lower than that of the RSR for the two SC densities, respectively. Apparently, the increase in the server density has a more positive effect on network performance than the increase in the SC density. In addition, the broadcasted message overhead decreases as more and more server deployed in the network as shown in Figure 17 below.

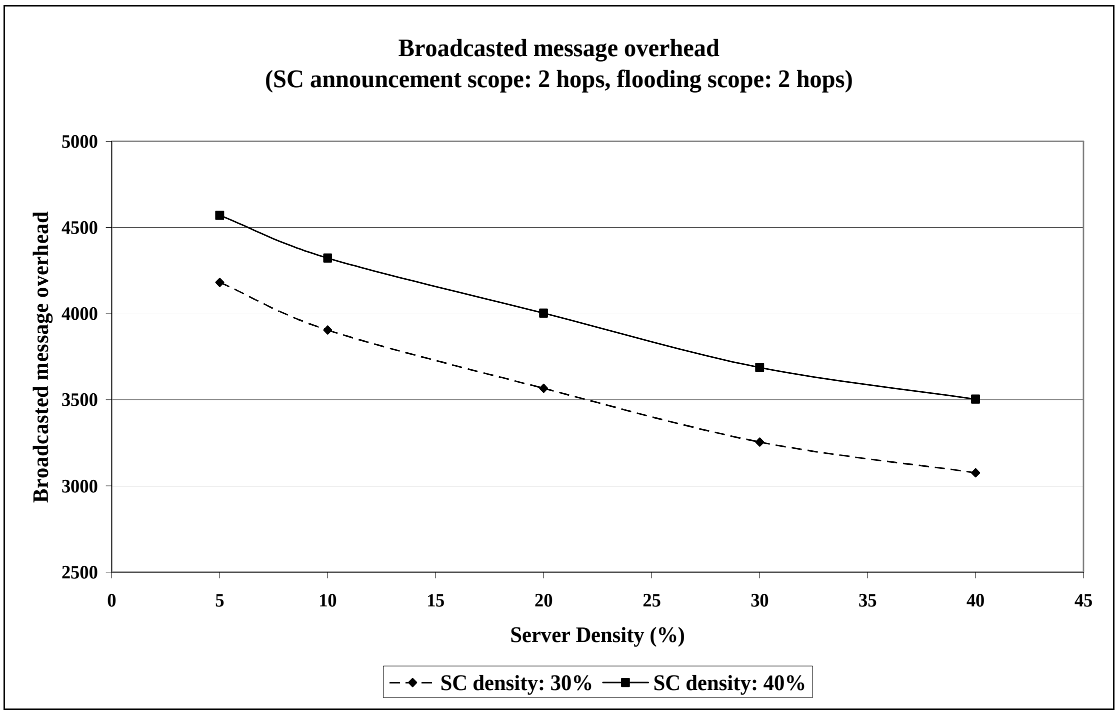

Figure 17: Broadcasted message overhead relative to SC and server density for the static network topology (Hybrid)

The reason for this decrease in broadcasted message overhead, yet still increase in total message overhead, is best illustrated in Figure 18 and Figure 19 on the next page. The total message overhead is broken down according to several major message types. As more and more nodes take on roles as servers, there will be more chances for certain servers to be positioned closer to the client. This will reduce the hops needed for a service request to be broadcasted when a client has to fall back on the pure flooding approach in those cases when there are no service coordinators being heard or a negative service reply has been received from its affiliated SC. Similarly, more servers will register with the service coordinators. This increases the chance for a positive service resolution at the SC. All this explains the decreasing "broadcasted service request". On the other hand, the service registrations increase in line with the number of servers, which outweighs the benefit of the aforementioned decrease and hence the explanation of the increased message overhead yet decreased broadcasted message overhead.

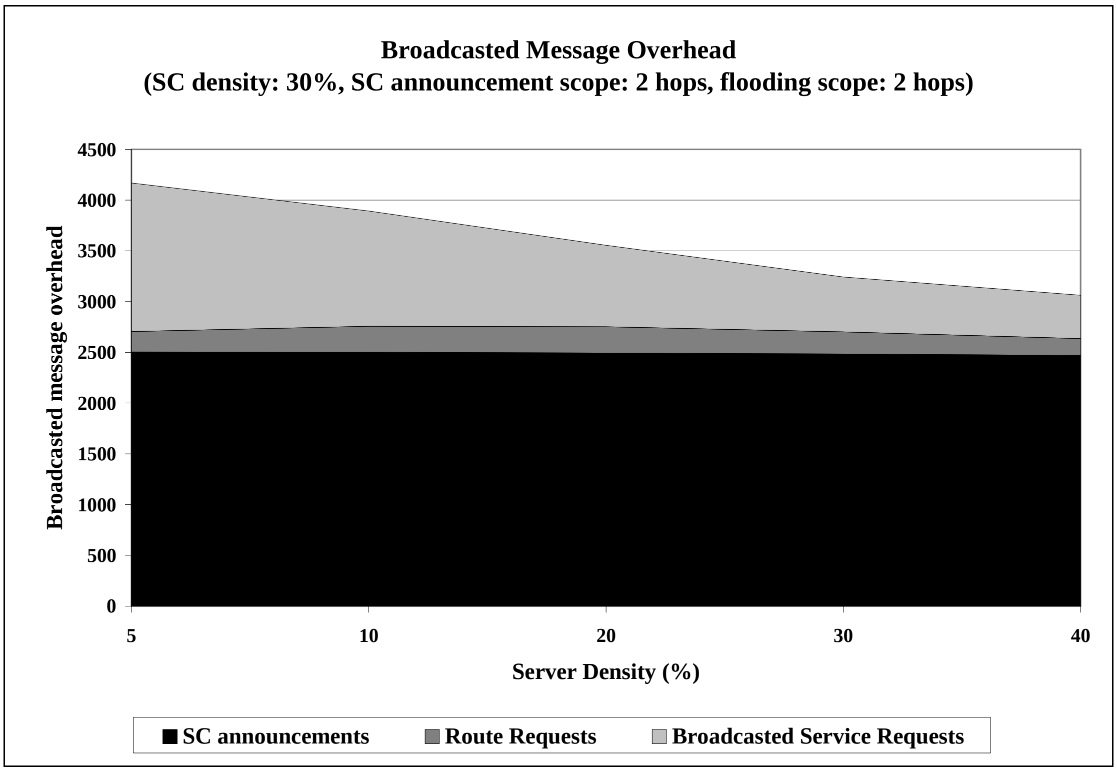

Figure 18: Detail of broadcasted message overhead by message type for the static network topology (Hybrid)

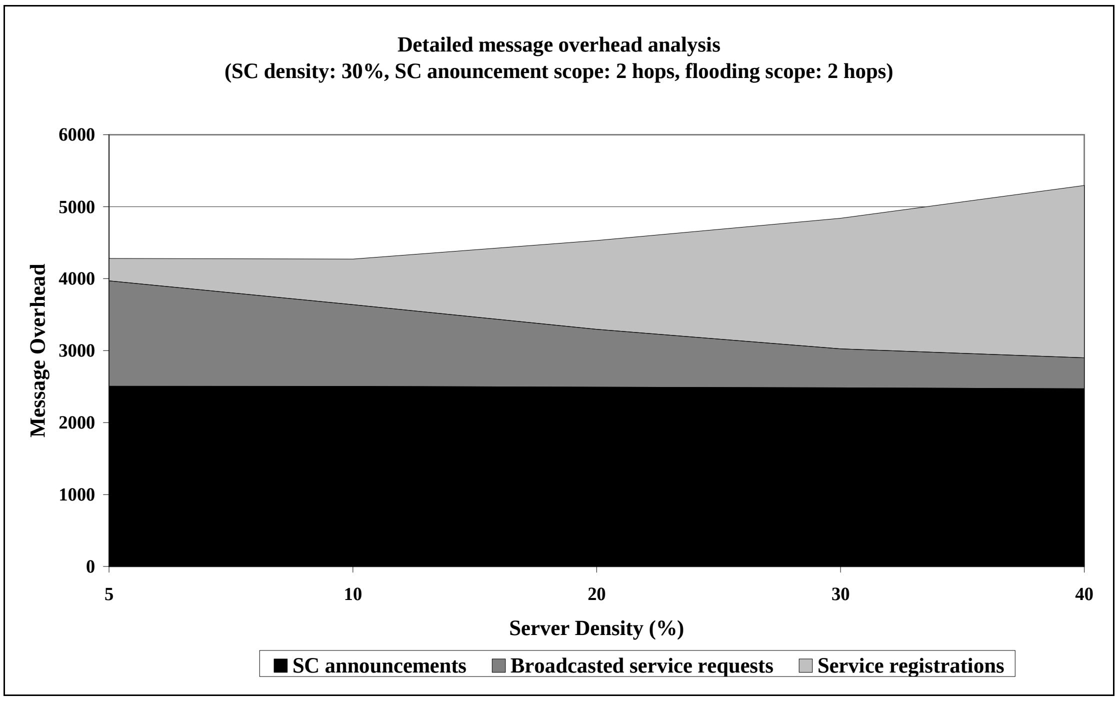

Figure 19: Detail of message overhead by message type for the static network topology (Hybrid)

7.1.3 RSR relative to different scope parameters

Figure 20 below shows the effect on the RSR by varying only the SC announcement scope, the flooding scope or both.

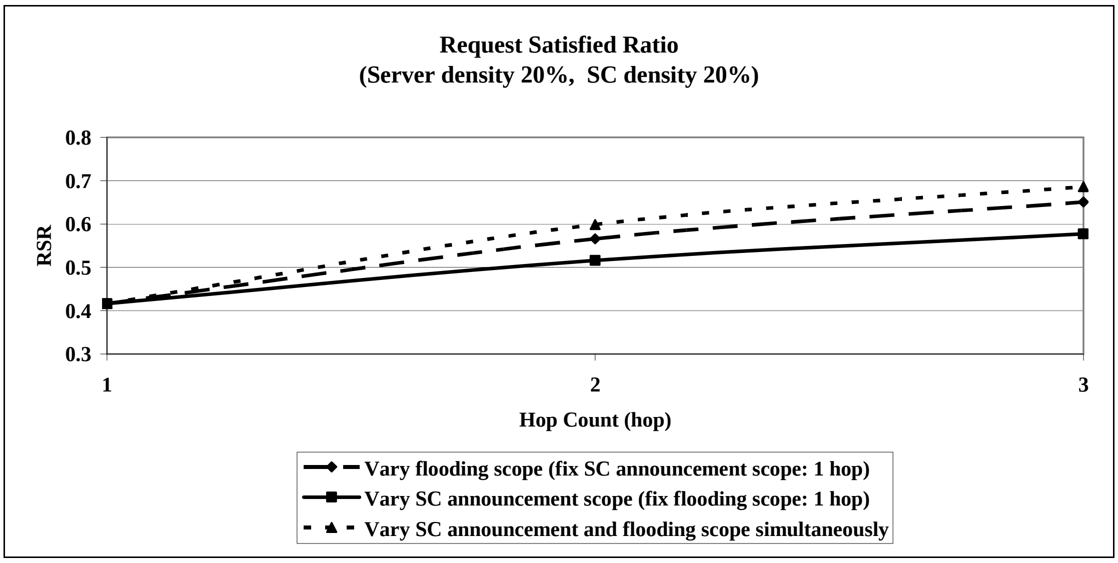

Figure 20: RSR relative to different scope parameters for the static network topology (Hybrid)

Table 4 below lists the detailed increase in the RSR and the message overhead.

	From one hop to two hops (RSR improvement in value)	From two hops to three hops (RSR improvement in value)	Overall RSR improvement (%)	Overall Message overhead increase
Fix flooding scope, vary sc announcement scope	0.1	0.06	38.8%	256%
Fix sc announcement scope, vary flooding scope	0.15	0.085	56.5%	113%
Vary both scopes simultaneously	0.18	0.087	65%	341%

Table 4: The effect of varying different scope parameters for the static network topology (Hybrid)

From Table 4 above we can see that by varying the flooding scope and SC announcement scope simultaneously, we can achieve a maximum overall RSR improvement. Varying the flooding scope alone has more impact on the RSR than by varying the SC announcement scope alone does. Table 4 also shows that the improvement is greater when varying the scope whatever the scope is from one hop to two hops than from two hops to three hops. The reason for this is that in the simulated network, the number of servers that can be discovered by the clients or the number of clients that can affiliate to service coordinators become fewer and fewer as service requests or SC announcements are broadcasted further away. In addition, possible network partitions may hinder higher hop retransmissions to be carried out.

Though the improvement in the RSR is appealing, yet the increase in scope trades off the network bandwidth for the increase in the RSR. The overall message overhead increase is listed in the last column of Table 4. Two of the major contributors to this increase are the increased SC announcements broadcasted in the network and the triggered service registrations.

7.2 Pure flooding architecture

7.2.1 RSR relative to server density and flooding scope

Figure 21 below shows that the RSR increases in line with the server density and the flooding scope, which correspond to the intuition. The improvement in the RSR when we increase the flooding scope from two hops to three hops is less than that when we increase the flooding scope from one hop to two hops. That is due to the same reason as stated in section 7.1.3 above for varying the flooding scopes. As to the flattening of the curves at a higher server density, it is the same reason as stated in section 7.1.1 on page 48 for the hybrid architecture. As more and more servers are deployed in the network, they will have overlapping influencing areas. One or more servers will offer the same type of service to the same area, which doesn’t necessarily improve the RSR.

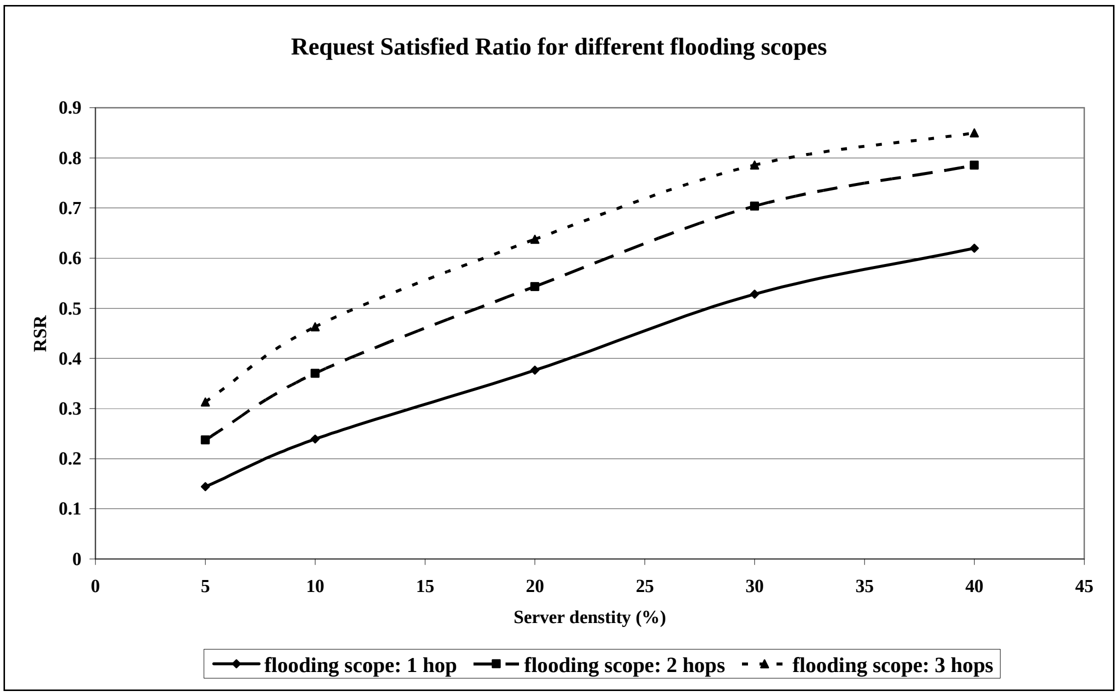

Figure 21: RSR relative to server density and flooding scope for the static network topology (Pure Flooding)

7.2.2 Broadcasted message overhead relative to server density and flooding scope

As Figure 22 below shows, broadcasted message overhead decreases as more and more servers being deployed in the network. This is because as more and more nodes take on roles as servers, there will be more chances for desired services to be located on servers that are closer to the client, which reduces the flooding scope of the service requests. However, the curve for one hop flooding scope is less steep than those for two hops and three hops. This is because both the client density and the service request interval are fixed for the simulations. Thus the number of service requests generated by all the clients in the network would be almost the same regardless of the server density. These service requests will be the only broadcasted messages in the pure flooding architecture. For the flooding scope of one hop, total number of broadcasted messages will stay the same (i.e. equals the total service requests generated). The slight inclination is due to the fact that a node can be a client and a server at the same time, which eliminates the need for broadcasting the service request. As number of servers increases, so does the chance of the collocation of a client and a server with the desired service type on the same node.

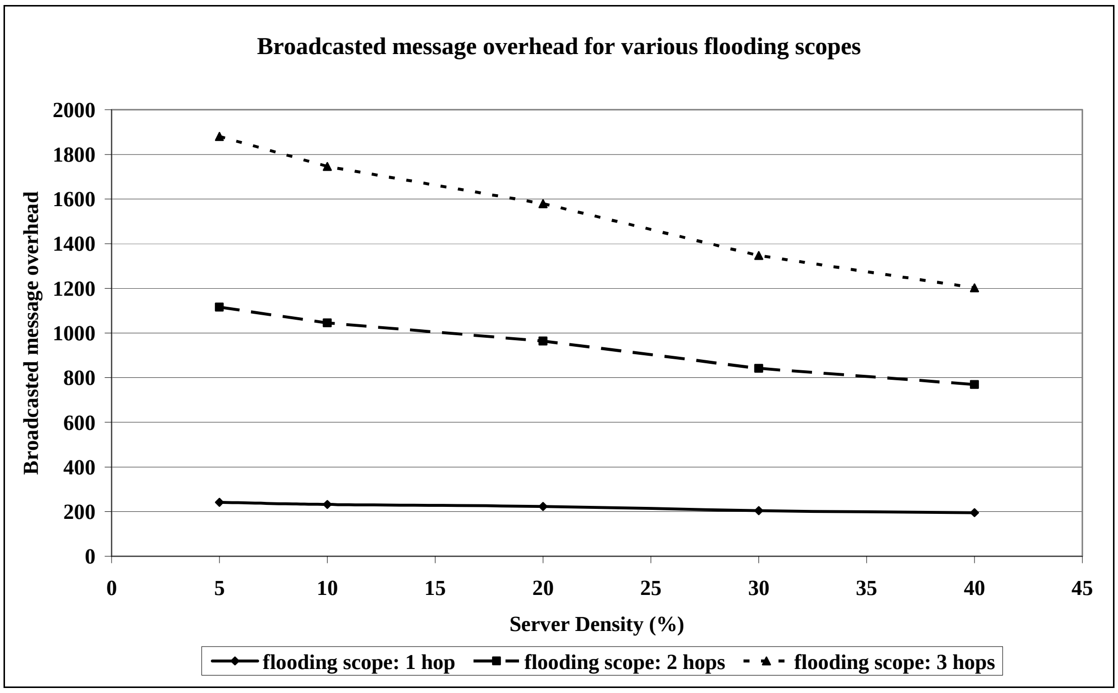

Figure 22: Broadcasted message overhead for different flooding scopes for the static network topology (Pure Flooding)

7.3 Comparison between the pure flooding and the hybrid architecture

7.3.1 RSR comparison

Figure 23 on the next page shows how the presences of service coordinators (i.e. for the hybrid architecture) influence the RSR. As we can see from the figure, the introduction of the service coordinators does improve the RSR. Depending on the announcement scope of the service coordinator, the RSR is improved by 8.3% and 20.8% respectively at a server density of 5%. This was listed in Table 5 below.

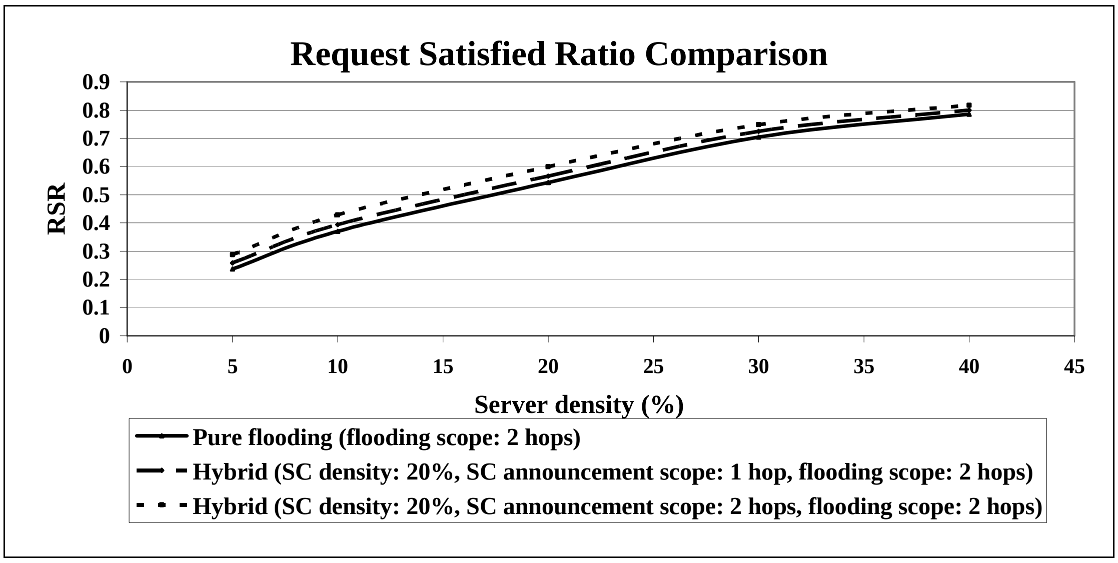

Figure 23: RSR comparison between the pure flooding and the hybrid architecture for the static network topology

	Flooding scope (hops)	SC announcement scope (hops)	Server density (%)	Service Availability/ RSR
Pure flooding	2	-	5%	0.24
Hybrid	2	1	5%	0.26
Hybrid	2	2	5%	0.29

Table 5: RSR comparison at a server density of 5% for the static network topology

The reason that SCs improve the RSR is revealed in Figure 24 and Figure 25 on the next page. Figure 24 illustrates a scenario with a flooding scope of two hops and an SC announcement scope of one hop. Without the service coordinator functionality implemented on the black node in Figure 24, the server would be unreachable from the client. However, with the SC functionality added on the black node, the server will be able to register its service with the service coordinator. And the client's service request will be able to reach the service coordinator and the service coordinator will respond to the client on behalf of the server.

Figure 25 on the next page shows a similar scenario, but the SC announcement scope is expanded to two hops. Without the service coordinator functionality implemented on the black node, neither client 1 nor client 2 will be able to find the server. But with the help of service coordinator functionality implemented on the black node, both clients can direct their service requests to their affiliated service coordinator i.e. the black node, which has cached the server information.

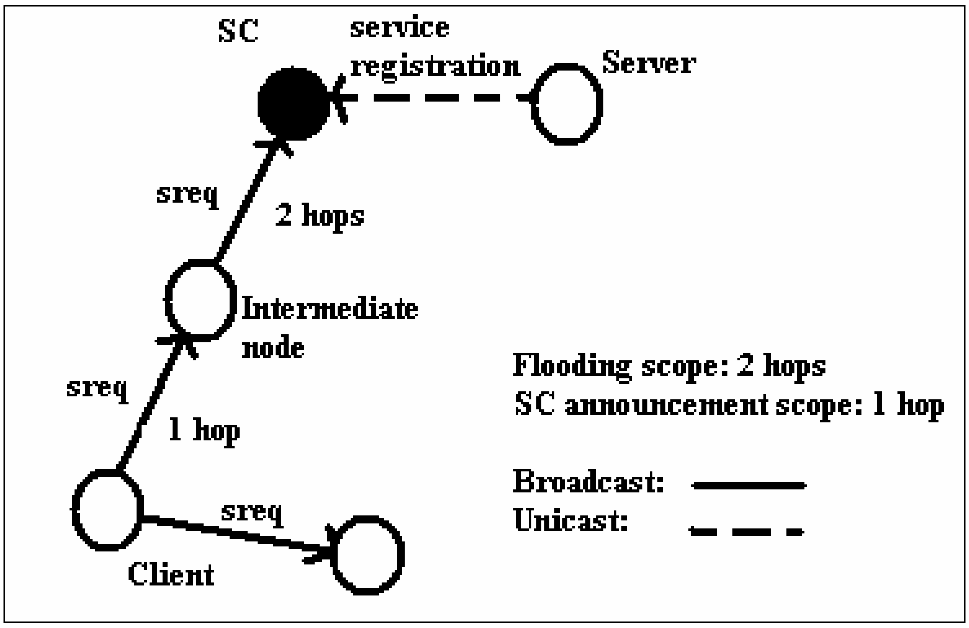

Figure 24: The effect of SC, scenario 1

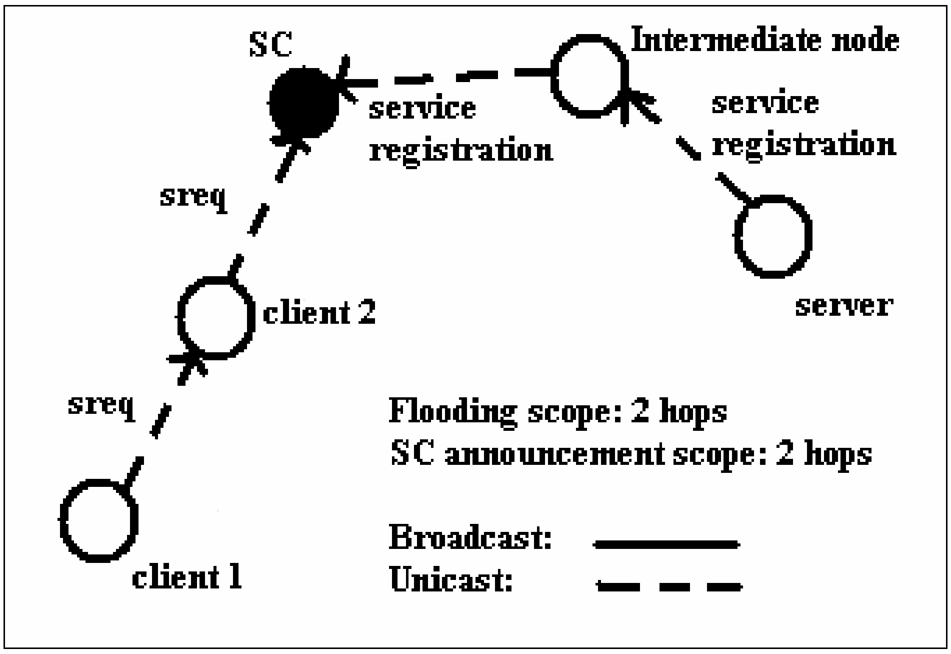

Figure 25: The effect of SC, scenario 2

Our simulation in which the underlying routing overhead is taken into consideration confirms the results obtained in previous work [34], i.e. service availability (RSR) is indeed higher with the hybrid approach.

However, introducing service coordinators to the network also introduces extra message overhead, such as service announcements, service registrations, not to mention the extra route discovery needed to actually contact the server. Whether these message overheads can be justified by the improving RSR will be analyzed in later sections.

7.3.2 Message overhead comparison

As pointed out in [34], the introduction of service coordinators introduces extra message overhead to the network, in terms of service announcements, service registrations and those related to service lookups. However, the routing overheads triggered by these messages are not taken into account in [34]. The objective is to optimize the benefits of additional service availability/RSR against the cost of additional overhead. Here, our analysis differs from [34], as we also take routing messages into account.

As we can see from Figure 26 below, though the introduction of the service coordinators does increase the RSR, yet it also results in a much higher level of messaging overhead. Service coordinators have introduced two proactive elements to the network, namely SC announcements and service registrations. These messages will take up a fixed bandwidth regardless of whether there exist service discoveries or not. From the figure, we can also see that there is no message overhead caused by route discoveries for the pure flooding architecture. This is because in the pure flooding architecture, it is always the service provider itself that responds to the service request and a forward route to the service provider is established along with the service reply. Accordingly, no additional route discovery is needed for the client to access the server. However, in the hybrid architecture, when service coordinators respond to service requests, forward routes are only established towards the service coordinators, not the service providers, so an extra round of route discovery is needed in order to access the server after the resolution.

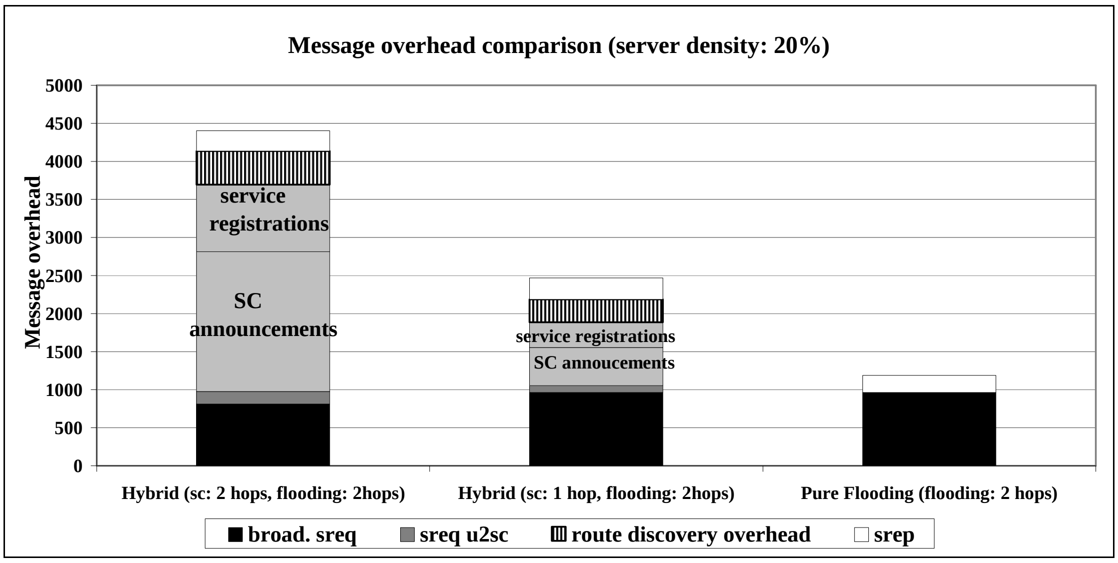

Figure 26: Detail comparison of message overhead by message type for the static topology

The introduction of service coordinators is expected to minimize the need for broadcasting the service requests. Yet from the simulation results, only after the server density reaches a certain level (20%), will the presence of a service coordinator begin to show its benefit as shown in Figure 27 on the next page. This again confirms the importance of the placement of service coordinators relative to the servers and the clients as mentioned in section 4.5 on page 33.

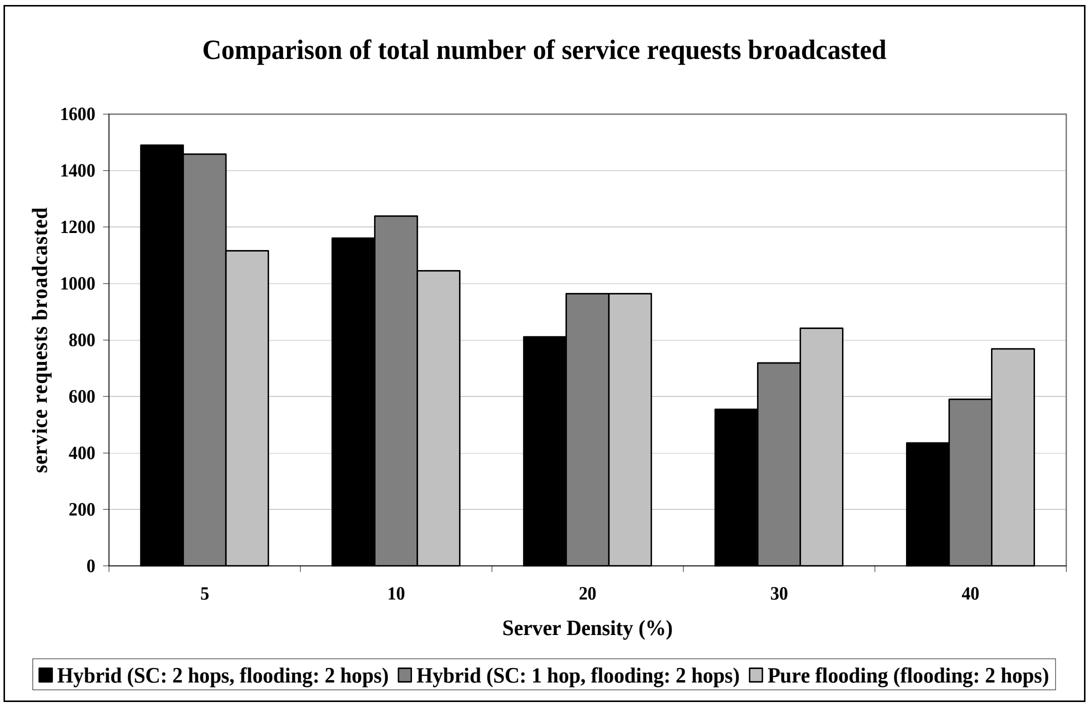

Figure 27: Comparison of total number of service requests broadcasted

7.3.3 Latency comparison

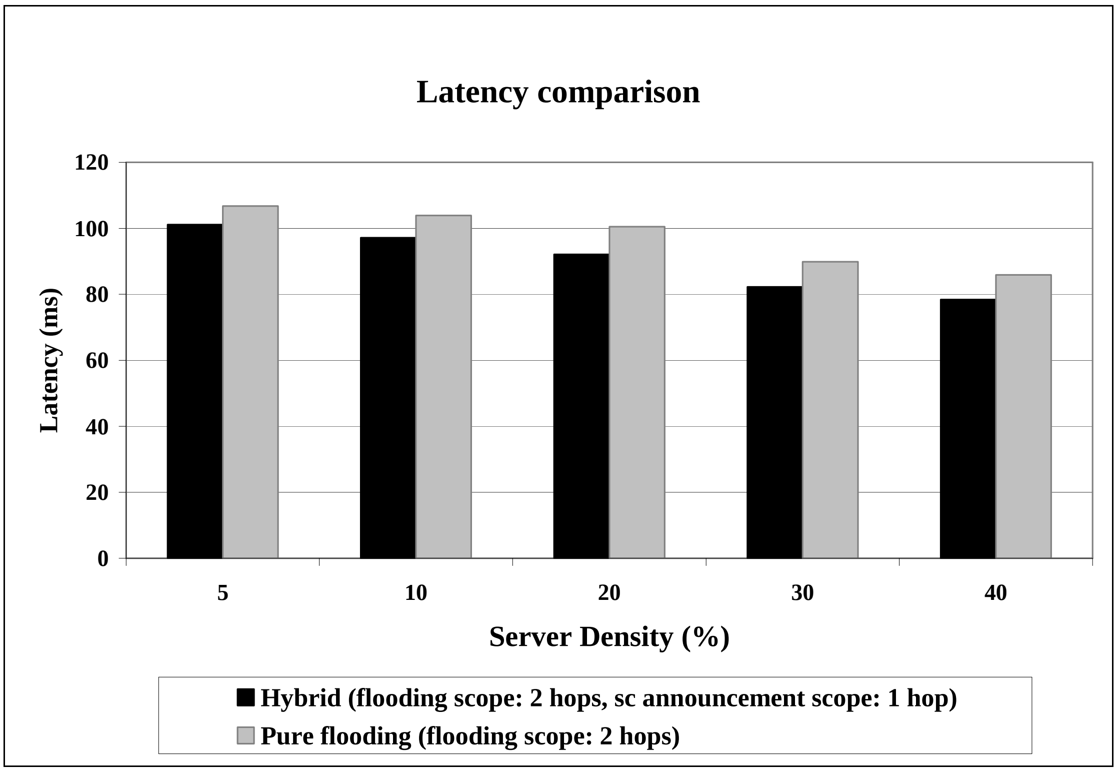

Figure 28: Latency comparison between the pure flooding and the hybrid architecture for the static network toplogy

Figure 28 on the previous page shows the comparison of service discovery latencies between the pure flooding and the hybrid architecture. Service discovery latency is the time from a node generates a service request until that node receives a positive service binding. The introduction of the service coordinators does minimize the service discovery latency. This is because many of the service requests can be satisfied at the service coordinators that are often closer to the client than the server themselves. In addition these service requests are unicasted to the service coordinator, thus no delay is caused by the additional broadcast jitter. The increase in number of servers has enhanced the chances for the client to find the matching service at the service coordinator or at a closer server, which results in a decreasing latency for both architectures.

7.4 Discussion

7.4.1 Latency

Service discovery is normally a step that users go through as part of the initial service initiation. For example: user normally would accept a second of delay when retrieving search results on the Internet (e.g. a Google lookup) or for setting up an IP Telephony call.

Figure 28 on the previous page shows that the service discovery latency is considerably lower than this. Furthermore, the differences in delays between the pure flooding and the hybrid architecture are only in the order of a few milliseconds and should be considered negligible in this context.

Conclusion: Delay is not a factor that distinguishes the one service discovery architecture from the other.

7.4.2 Tradeoff between the service availability and the message overhead

Hypothesis I: The increase in service availability (i.e. RSR) by adding service coordinators is negligible compared to the extra message overhead it caused.

Hypothesis II: There is always a pure flooding scheme that outperforms a hybrid scheme with higher service availability (i.e. RSR) and less message overhead no matter what the combination of tunable parameters (i.e. flooding scope and SC announcement scope) is.

Hypothesis III: The former two hypotheses still hold for an increased SC density.

As demonstrated in earlier sections, the introduction of service coordinators with the hybrid architecture increases the service availability (i.e. RSR) as well as the message overhead, as compared to the pure flooding architecture. The simulations done in this chapter are to verify the aforementioned hypotheses.

Table 6 through Table 8 list the RSR values and the message overhead for the two architectures at three different server densities. Some of these results have already been presented in graphs and tables above.

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.144	263
Flooding	2	-	0.237	1178
Flooding	3	-	0.313	2001
Flooding	4	-	0.38	2799
Flooding	5	-	0.431	3526
Flooding	6	-	0.476	4164
Hybrid	1	1	0.166	1208
Hybrid	2	1	0.258	2456
Hybrid	3	1	0.33	3544
Hybrid	1	2	0.228	2921
Hybrid	2	2	0.287	4235
Hybrid	3	2	0.357	5413
Hybrid	1	3	0.288	4356
Hybrid	2	3	0.334	5609
Hybrid	3	3	0.382	6773

Table 6: Overall comparison between the pure flooding ("Flooding") and the hybrid architecture ("Hybrid") at a 5% server density for the static network topology

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.377	303
Flooding	2	-	0.543	1190
Flooding	3	-	0.638	1992
Flooding	4	-	0.70	2775
Flooding	5	-	0.736	3477
Flooding	6	-	0.756	4069
Hybrid	1	1	0.416	1544
Hybrid	2	1	0.566	2480
Hybrid	3	1	0.651	3281
Hybrid	1	2	0.516	3638
Hybrid	2	2	0.599	4421
Hybrid	3	2	0.668	5099
Hybrid	1	3	0.578	5500
Hybrid	2	3	0.639	6192
Hybrid	3	3	0.686	6802

Table 7: Overall comparison between the pure flooding ("Flooding") and the hybrid architecture ("Hybrid") at a 20% server density for the static network topology

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.62	338
Flooding	2	-	0.785	1152
Flooding	3	-	0.85	1880
Flooding	4	-	0.878	2555
Flooding	5	-	0.89	3126
Flooding	6	-	0.895	3598
Hybrid	1	1	0.658	1824
Hybrid	2	1	0.80	2491
Hybrid	3	1	0.857	3039
Hybrid	1	2	0.742	4298
Hybrid	2	2	0.817	4746
Hybrid	3	2	0.862	5161
Hybrid	1	3	0.768	6660
Hybrid	2	3	0.833	7083
Hybrid	3	3	0.864	7445

Table 8: Overall comparison between the pure flooding ("Flooding") and the hybrid architecture ("Hybrid") at a 40% server density for the static network topology

7.4.2.1 Comparing the pure flooding and the hybrid architecture at a 20% SC density

7.4.2.1.1 Considering single-hop SC announcement scope

For a server density of 5%, adding service coordinators with announcement scopes of 1 hop to the pure flooding architecture with various flooding scopes increases the RSR. As we can see from Table 9 on the next page, the RSR is increased from 0.144 to 0.166 for a flooding scope of 1 hop (i.e. an increase ratio of 1.15), from 0.237 to 0.258 for a flooding scope of 2 hops (i.e. an increase ratio of 1.09) and from 0.313 to 0.33 for a flooding scope of 3 hops (i.e. an increase ratio of 1.05). However, with such minimal increase ratios in the service availability, the message overhead of the hybrid scheme is tremendous higher. The message overhead of the hybrid architecture with a flooding scope of 1 hop is almost 5 times as much as that of the pure flooding architecture with the same flooding scope. The message overhead of the other two hybrid architectures is also doubled compared to the pure flooding architectures with corresponding flooding scopes.

From Table 9, we can see that by expanding the flooding scope of the pure flooding scheme from 1 hop to 2 hops; it will outperform the hybrid scheme that has a flooding scope of 1 hop. The pure flooding scheme exhibits higher service availability, i.e. 0.237 as opposed to 0.166 and less message overhead, i.e. 1178 as opposed to 1208. By further expanding the flooding scope of the pure flooding scheme, the hybrid schemes with multi-hop flooding scopes will also be outperformed. We can see that the hybrid architecture with a flooding scope of 2 hops is inferior to the pure flooding architecture with a flooding scope of 3 hops. Similarly, the hybrid scheme with a flooding scope of 3 hops is inferior to pure flooding architecture with a flooding scope of 4 hops (Table 9).

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.144	263
Flooding	2	-	0.237	1178
Flooding	3	-	0.313	2001
Flooding	4	-	0.38	2799
Hybrid	1	1	0.166	1208
Hybrid	2	1	0.258	2456
Hybrid	3	1	0.33	3544

Table 9: Comparing pure flooding ("Flooding") and the hybrid architecture ("Hybrid") with server density of 5%, SC announcement scope of 1 hop. The values are extracted from Table 6.

We see that the same situation is also representative for other server densities. For example, at a 20% server density (Table 10), the hybrid architecture increases the RSR by 10.3%, 4.2% and 2% for the flooding scopes of one hop, two hops and three hops, respectively, the increase in message overhead is, on the other hand, too large to be justified by the minimal increase.

Again, the hybrid scheme with a flooding scope of 1 hop is outperformed by the pure flooding scheme with a flooding scope of 2 hops, which has a higher RSR of 0.543 and less message overhead of 1190 (Table 10). The hybrid schemes with flooding scopes of 2 hops and 3 hops are inferior to the pure flooding schemes with flooding scopes of 3 hops and 4 hops, respectively.

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.377	303
Flooding	2	-	0.543	1190
Flooding	3	-	0.638	1992
Flooding	4	-	0.70	2775
Hybrid	1	1	0.416	1544
Hybrid	2	1	0.566	2480
Hybrid	3	1	0.651	3281

Table 10: Comparing pure flooding ("Flooding") and the hybrid architecture ("Hybrid") with server density of 20%, SC announcement scope of 1 hop. The values are extracted from Table 7.

As another example, we may look at a service density of 40% (Table 11). Here, we see exactly the same pattern as we saw at lower service densities.

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead

Flooding	1	-	0.62	338
Flooding	2	-	0.785	1152
Flooding	3	-	0.85	1880
Flooding	4	-	0.878	2555
Hybrid	1	1	0.658	1824
Hybrid	2	1	0.80	2491
Hybrid	3	1	0.857	3039

Table 11: Comparing pure flooding ("Flooding") and the hybrid architecture ("Hybrid") with a server density of 40%, SC announcement scope of 1 hop. The values are extracted from Table 8.

Sub-conclusion I: The improvement in service availability (i.e. RSR) exhibited by a hybrid scheme with a single-hop SC announcement scope and a single-hop or multi-hop flooding scope over the pure flooding scheme with the same flooding scope is negligible compared to the message overhead it increased.

Sub-conclusion II: There is always a pure flooding scheme that outperforms a hybrid scheme that has a single-hop SC announcement scope.

7.4.2.1.2 Considering multi-hop SC announcement scope

In section 7.1.3 on page 52, we demonstrated that by increasing the SC announcement scope of the hybrid architecture, the service availability was improved slightly. The downside is a considerable degradation in message overhead.

Comparing with the pure flooding architectures, the hybrid architectures with an SC announcement scope of 3 hops have increased the RSR by 100%, 41% and 22% for flooding scopes of one, two and three hops, respectively as illustrated in Table 12 below. However, they also increase the message overhead by 1556%, 376% and 238%, respectively.

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.144	263
Flooding	2	-	0.237	1178
Flooding	3	-	0.313	2001
Hybrid	1	2	0.228	2921
Hybrid	2	2	0.287	4235
Hybrid	3	2	0.357	5413
Hybrid	1	3	0.288	4356
Hybrid	2	3	0.334	5609
Hybrid	3	3	0.382	6773
Flooding	4	-	0.38	2799
Flooding	5	-	0.431	3526

Table 12: Comparing pure flooding scheme ("Flooding") and different hybrid schemes ("Hybrid") with increasing SC announcement scopes. The values are extracted from Table 6, which covers a service density of 5%

Since we showed that the pure flooding scheme was superior to the hybrid architecture for a single-hop SC announcement scope, it comes at no surprise that the same is the case when the SC announcement scope is of multiple hops. For example, as we see in Table 12 on the previous page, a pure flooding scheme with a flooding scope of 4 hops outperforms all the hybrid schemes with multi-hop SC announcement scopes presented in the table. Though the hybrid scheme with an SC announcement of 3 hops and a flooding scope of 3 hops offers a higher RSR than that offered by the pure flooding scheme with a 4-hop flooding scope, the increase in the RSR of 0.5% is negligible compared to the increase in message overhead of 142%. Therefore, this hybrid scheme is still inferior to the pure flooding scheme with a 4-hop flooding scope. By further expanding the flooding scope of the pure flooding scheme to five hops, it will then offer a higher RSR and less message overhead than the aforementioned hybrid scheme.

The same conclusions are also drawn for other service densities, such as for a service density of 20% shown in Table 7 on page 60 or a server density of 40% shown in Table 8 on page 61. Both tables show that an increase in the SC announcement scope increases the service availability slightly, while the message overhead increases dramatically. Thus, it is easy to see that the flooding architecture outperforms the hybrid architecture. For both service densities, the pure flooding architecture with a flooding scope of 4 hops will outperform all the hybrid architectures with multi-hop SC announcement scopes presented in the tables.

Sub-conclusion III: The improvement in service availability (i.e. RSR) exhibited by a hybrid scheme with a multi-hop SC announcement scope and a single-hop or multi-hop flooding scope over the pure flooding scheme with the same flooding scope is negligible compared to the message overhead it increased.

Sub-conclusion IV: There is always a pure flooding scheme that outperforms the hybrid scheme that has a multi-hop SC announcement scope.

7.4.2.2 Comparing the pure flooding and the hybrid architecture at a 30% SC density

We showed that the pure flooding scheme was superior to the hybrid architecture independent of the SC announcement scope at a SC density of 20%; it comes at no surprise that the same is the case when the SC density is increased. We have shown in Table 13 on the next page an example that covers a server density of 20% and an increased SC density of 30%.

Increasing the SC density will slightly increase the RSR. Comparing, for example, the hybrid schemes with single-hop SC announcement scopes in Table 13 on the next page with those in Table 7 on page 60. The increase ratio in the message overhead is 25%, 18% and 14% more than that in the RSR for flooding scopes of one, two and three hops, respectively.

As already shown in section 7.1.1 on page 48 and section 7.1.2 on page 49 that increasing the SC density will barely increase the RSR, yet the increase in message overhead is rather noticeable. This is shown in Figure 29 below. Here we can see a relatively vertical line, which indicates the increase in the SC density has a much less influence on the RSR than on the message overhead. The reasons are already covered in section 7.1.1 and 7.1.2.

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.377	303
Flooding	2	-	0.543	1190
Flooding	3	-	0.638	1992
Flooding	4	-	0.70	2775
Flooding	5	-	0.736	3477
Hybrid	1	1	0.432	2010
Hybrid	2	1	0.577	2969
Hybrid	3	1	0.658	3768
Hybrid	1	2	0.525	4597
Hybrid	2	2	0.612	5433
Hybrid	3	2	0.678	6156
Hybrid	1	3	0.579	6717
Hybrid	2	3	0.651	7474
Hybrid	3	3	0.695	8149

Table 13: Comparing pure flooding (“Flooding”) and different hybrid schemes ("Hybrid"), which covers a service density of 20% and an increasing SC density of 30% for the static network topology

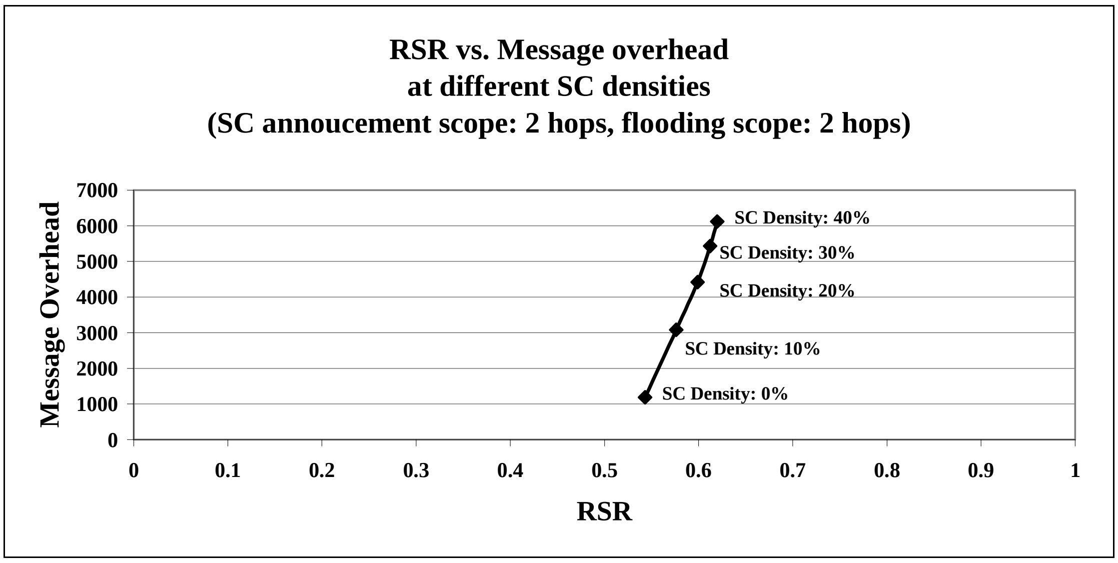

Figure 29: The effects of increasing SC densities for the static network topology

It comes at no surprise that we can always find a pure flooding scheme that outperforms the hybrid scheme with higher RSR and less message overhead. The pure flooding scheme with a 2-hop flooding scope that offers a RSR of 0.543 and a message overhead of 1190 outperforms the hybrid scheme with a single-hop SC announcement scope and a single-hop flooding scope which has a RSR of 0.432 and message overhead of 2010. The pure flooding scheme with a 4-hop flooding scope will outperform all the other hybrid schemes as presented in Table 13 in the previous page. It will even outperform the hybrid scheme with the single-hop SC announcement and the single-hop flooding scope in the sense that the increase in the message overhead can be justified by the increase in the RSR. As RSR is increased from 0.432 to 0.70, an improvement of 62%, the message overhead is increased from 2010 to 2775, an increase of only 38%.

Sub-conclusion V: All the sub-conclusions drawn above still hold for an increased SC density.

7.4.2.3 Flooding scope vs. SC announcement scope in the hybrid scheme

Another phenomenon we can observe from all the simulation data above is that the increase in the flooding scope of the hybrid service discovery architecture offers a better performance than the increase in the SC announcement scope does. This is already covered in section 7.1.3 on page 52. The client will fall back on the pure flooding scheme if there are no service coordinators heard or the service reply from its affiliated SC is negative. The flooding scope controls the performance of the pure flooding part of the hybrid scheme. Since increasing the flooding scope of the hybrid architecture exhibits a better network performance in terms of service availability (i.e. RSR) and message overhead than increasing the SC announcement scope does, the benefit of the pure flooding is again being proved.

Conclusion for the chapter: The pure flooding service discovery architecture is more preferable to the hybrid service discovery architecture under the conditions of no node mobility.

Chapter 8
Simulations with dynamic network topologies

The purpose of the simulations in this chapter is to:

- Compare the performance between the pure flooding and the hybrid service discovery architectures in terms of service availability (i.e. RSR) and message overhead under the conditions of node mobility
- Come to a conclusion about the preference of the two service discovery architectures based on the simulation results under the conditions of node mobility

Simulation parameters are listed in the Table 14 below:

NET SIZE	300M x 300M
TRANSMISSION RANGE	50M
SIMULATOIN TIME	500S
MOBILITY	Random Waypoint
MAX MOVING SPEED	VARIABLE
MIN MOVING SPEED	0M/S
PAUSE TIME	10S
ACTIVE ROUTE TIMEOUT	10S
SERVICE REQUEST INTERVAL	20S
SC ANNOUNCEMENT INTERVAL	10S
SERVICE REQUEST FLOODING SCOPE	1, 2, 3 HOPS
ROUTING PROTOCOL	AODV
NUMBER OF NODES	50
NODES POSITION	RANDOM
TYPES OF SERVICES	2
SERVER DENSITY	VARIABLE
SERVICE COORDINATOR DENSITY	VARIABLE
SC ANNOUNCEMENT SCOPE	1, 2, 3 HOPS
CLIENT DENSITY	20%

Table 14: Simulation parameters for dynamic simulations

The original 300x300m network is partitioned into two areas as Figure 30 on the next page shows in order to simulate the effect of nodes leaving the network.

Nodes are moving according to the mobility pattern of random waypoint within the whole area i.e. 300x300m. The moment the node leaves the active network area, it is considered as having left the network. It will neither initiate any service requests, nor relay any kind of messages until it moves back into the active network area again. All the routing information cached at the node will be invalidated.

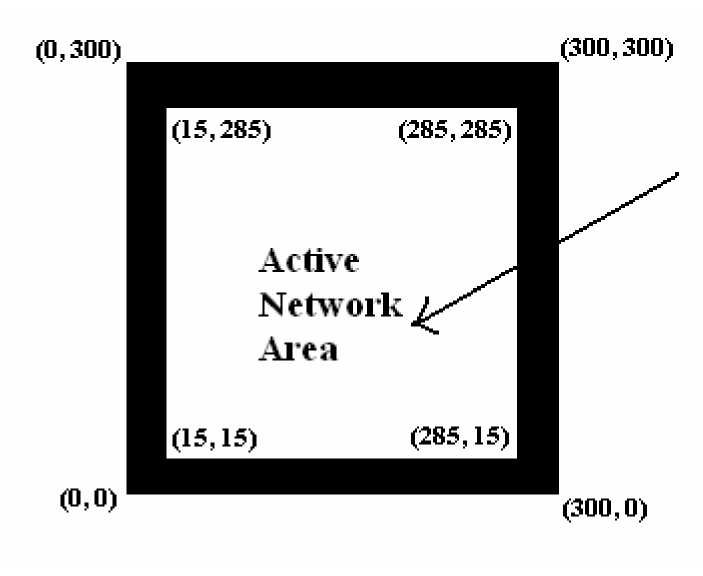

Figure 30: Network partitions for the dynamic network topology

8.1 Hybrid architecture

8.1.1 RSR relative to server density and sc density

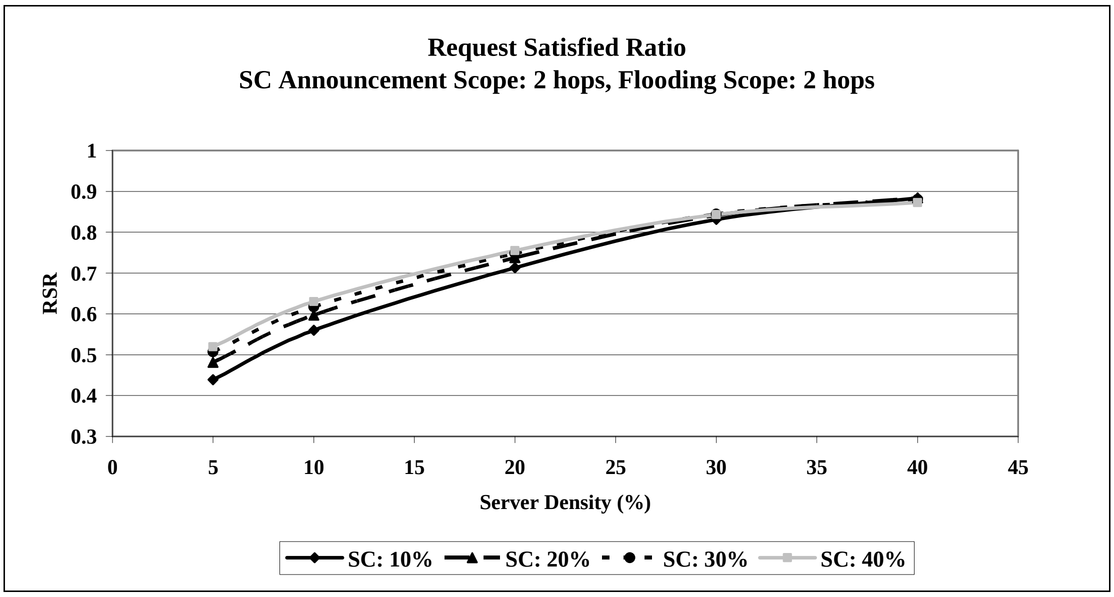

Figure 31: RSR relative to SC and server density for the dynamic network topology (Hybrid)

Figure 31 above shows that with the node mobility added to the network, the conclusions drawn from the static case still hold. The RSR increases as server and service coordinator functions being added to more and more nodes in the network. In addition, the server density is more essential to the increase in the RSR than the SC density. The RSR is improved by 101%, 84%, 74% and 68% when the server density increases from 5% to 40% for SC densities of 10%, 20%, 30% and 40%, respectively.

However, the improvement in the RSR is only 6% when the SC density increases from 10% to 40% for a server density of 20%.

At a higher server density level, the increase by adding service coordinators is negligible, there exhibits even a decrease in the RSR at a server density of 40% when SC density increases from 20% to 40%. This is caused by the stale server information passed out by the service coordinators, which is referred to as false positive replies and will be further discussed in section 8.1.3 on page 70. The server information cached at the service coordinator is considered to be stale if the server is outside the service coordinator's announcement scope or it is outside the active network area. The stale server information still has a non-expired timestamp within the service coordinator. At a higher server density level, increasing the SC density will make it possible for more servers to register their services with the service coordinators and more clients to affiliate themselves to them. Since more servers are registered with one or more service coordinators, the chances for stale server information, which is caused by the node mobility, cached at the service coordinators will increase. Since more clients are affiliated to the service coordinators, service coordinators will be exploited more often. Accordingly, there is more chance for the stale server information to be passed out to the clients by their affiliated service coordinators. All of these have caused the decrease in RSR at a server density of 40%.

8.1.2 Message overhead relative to server density and SC density

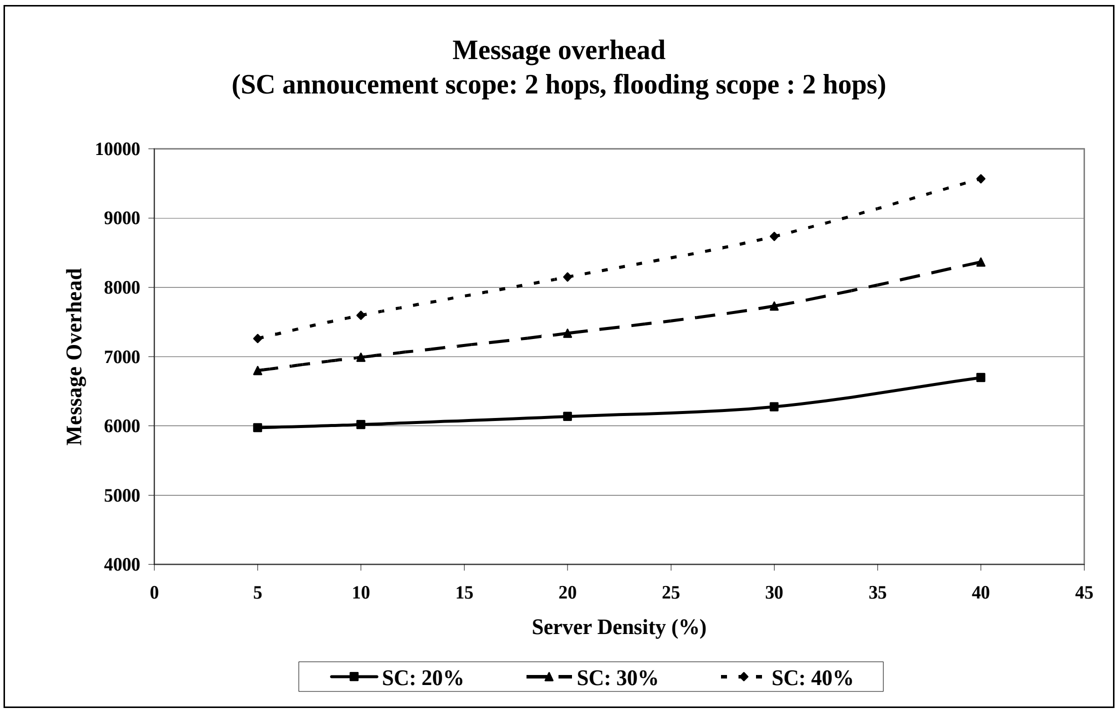

Figure 32: Message overhead relative to SC and server density for the dynamic topology (Hybrid)

Again, the mobility doesn't change the results obtained from the static case. The message overhead increases along with the server and the SC density as shown in Figure 32 above. As we can see from Figure 33 on the next page, the biggest contributors to the increase in message overhead are the service registrations. They increase proportional to the number of servers.

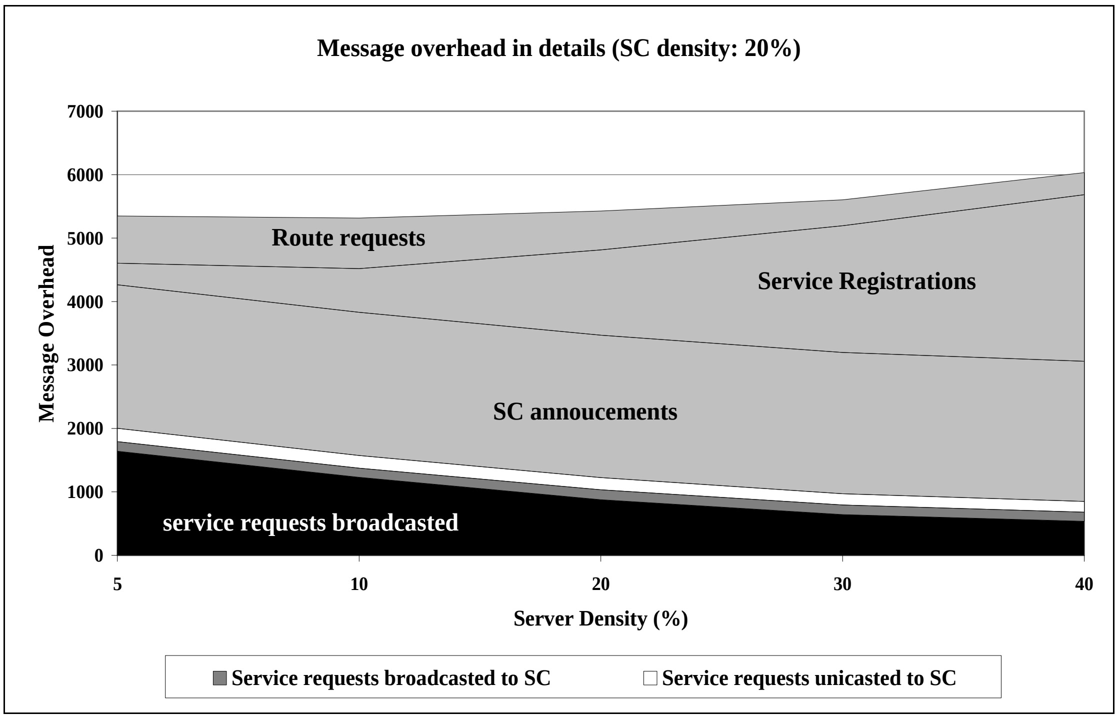

Figure 33: Detail of message overhead by type for the dynamic network topology (Hybrid)

8.1.3 False positive replies from service coordinator

A service coordinator might pass out false positive service replies to the service requestor. False positive service replies contain server information to those servers that the service coordinator claims to be within its reach, but their actual positions are out of the service coordinator’s announcement scope or out of the active network area. As we can see from Figure 34 on the next page, the chance for a service coordinator to pass out false positive replies increases as nodes move faster. As mentioned earlier, the SC will invalidate the server information if there are no service registrations received before the expiration. False positive service replies might be generated in the period after the server node moves out of the reach of the SC and the expiration of the cached server information. Figure 35 on the next page illustrates this time period. If a service request for the server is received during that time period, the service reply will be false positive. The faster the server node moves out, the longer this period might be and the higher the possibility for the SC to give out stale server information. False positive service replies do not always lead to bad consequences. If the server is still in the active network area, as long as the network is not partitioned between the client and the server, the client will still be able to access the server and the service discovery is still considered to be successful. However, if the network is partitioned between the client and the server or the server is outside the active network area, the client will assume the destination unreachable and discard the packet. In the latter case, the service discovery is considered to have failed, though there may exist some other servers in the network that offer the same service and can be reached by the client. This will cause a decrease in the service availability (i.e. RSR).

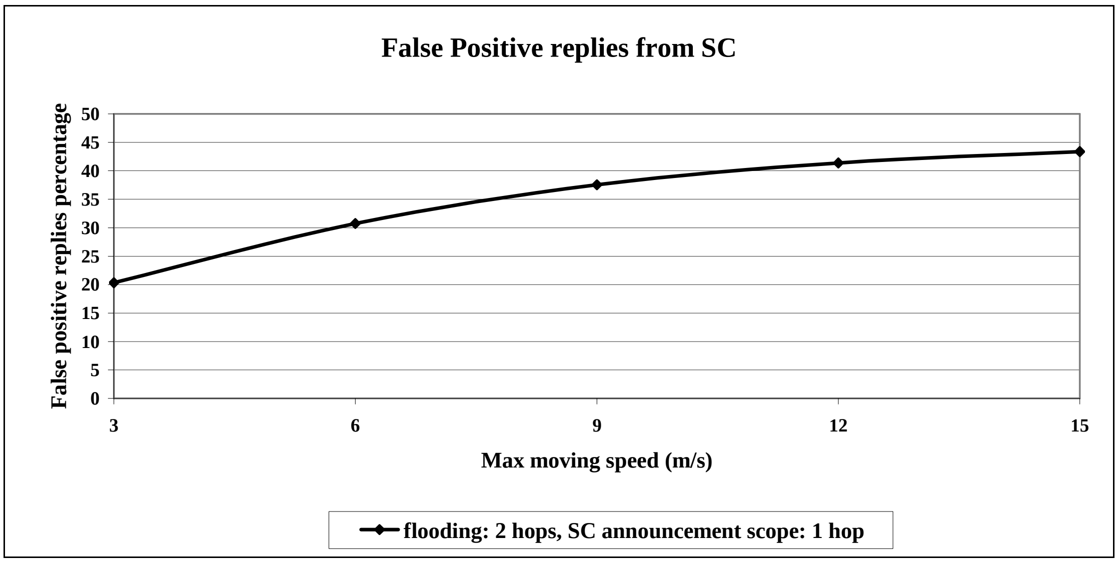

Figure 34: False positive replies percentage

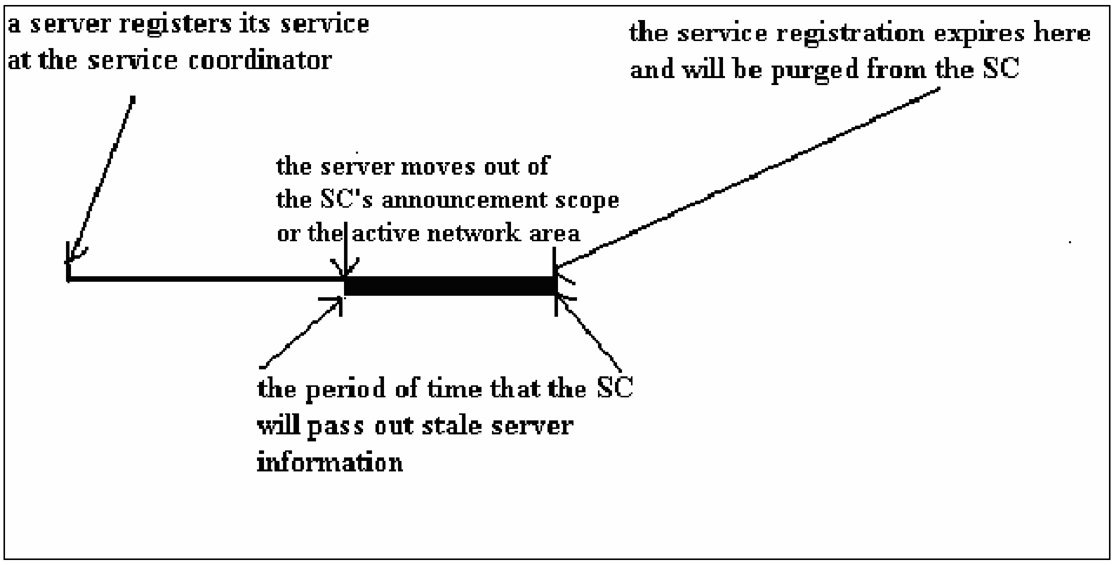
a server registers its service at the service coordinator
the service registration expires here and will be purged from the SC
the server moves out of the SC's announcement scope or the active network area
the period of time that the SC will pass out stale server information

Figure 35: The period during which false positive replies are passed out

8.2 Pure flooding architecture

8.2.1 RSR relative to server density and flooding scopes

With no surprise, the adding of mobility to the network doesn’t change the fact that the RSR increases along with the increase in flooding scope and server density as shown in Figure 36 on the next page. The same arguments apply here as those stated in section 7.2.1 on page 53 for the static network topology.

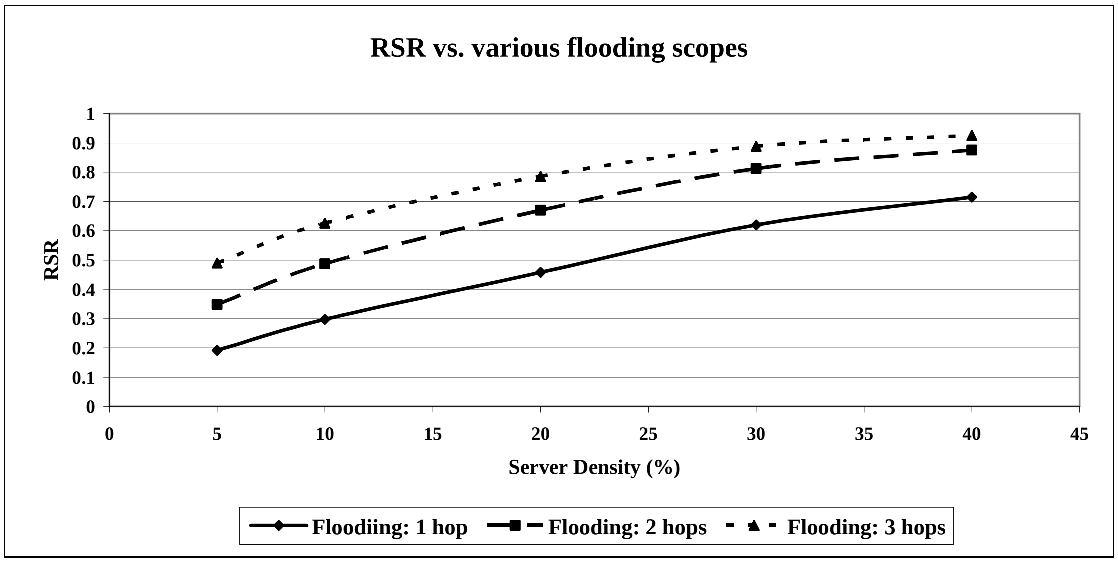

Figure 36: RSR relative to server density and flooding scope for the dynamic network topology (Pure Flooding)

8.3 Comparison between the hybrid and the pure flooding architecture

8.3.1 RSR comparison

Adding service coordinators to the dynamic network shows the same effect as with the static network. The RSR is improved by introducing service coordinators to the network as Figure 37 below shows. The reason for the increase is the same as that stated in 7.3.1 on page 55. However, the increase is less significant at a higher server density level.

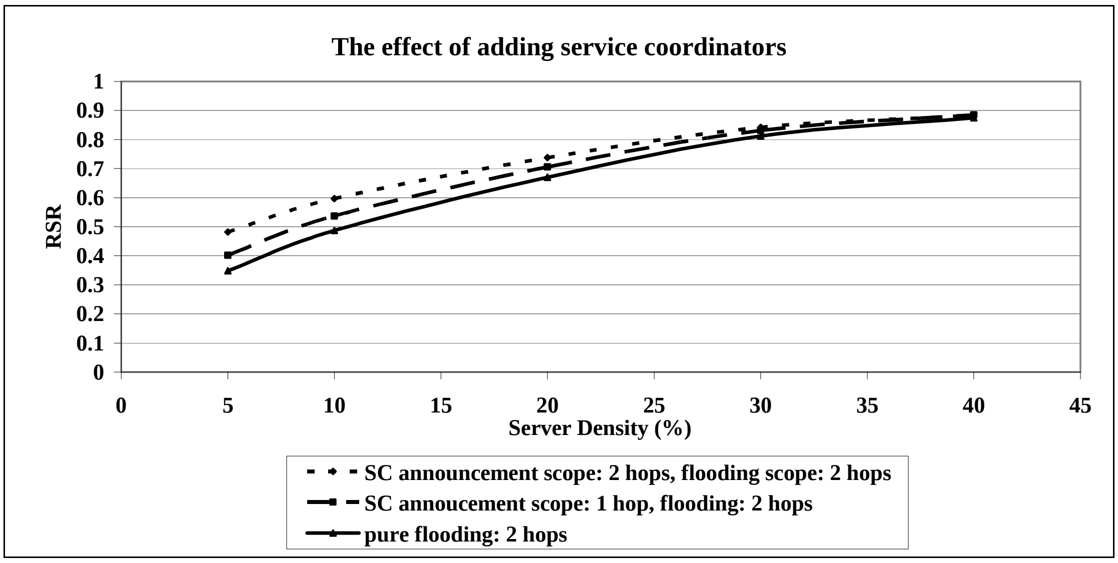

Figure 37: RSR comparison between the pure flooding and the hybrid architecture for the dynamic network topology

Table 15 below lists the detailed RSR values for the two architectures at a 40% server density. The hybrid scheme with a single-hop SC announcement scope has increased the RSR by 1% over the pure flooding. At such a higher server density level, most of the service requests can be satisfied with the predefined flooding scope in the pure flooding architecture, which makes the adding of service coordinator unnecessary as far as RSR is concerned. By further expand the SC announcement scope, the RSR value is even decreased by 0.1% compared to the single-hop SC announcement scope. The reason for the decrease is the same as that stated in section 8.1.1 on page 69 concerning the stale server information passed out by the service coordinator.

	Flooding scope (hops)	SC announcement scope (hops)	Server density (%)	Service Availability/RSR
Pure flooding	2	-	40%	0.875
Hybrid	2	1	40%	0.885
Hybrid	2	2	40%	0.884

Table 15: RSR comparison at a server density of 40% for the dynamic network topology

Our simulation for the mobility case again confirms the results obtained in previous work [34], i.e. service availability (RSR) is indeed higher with the hybrid approach.

8.3.2 Message overhead comparison

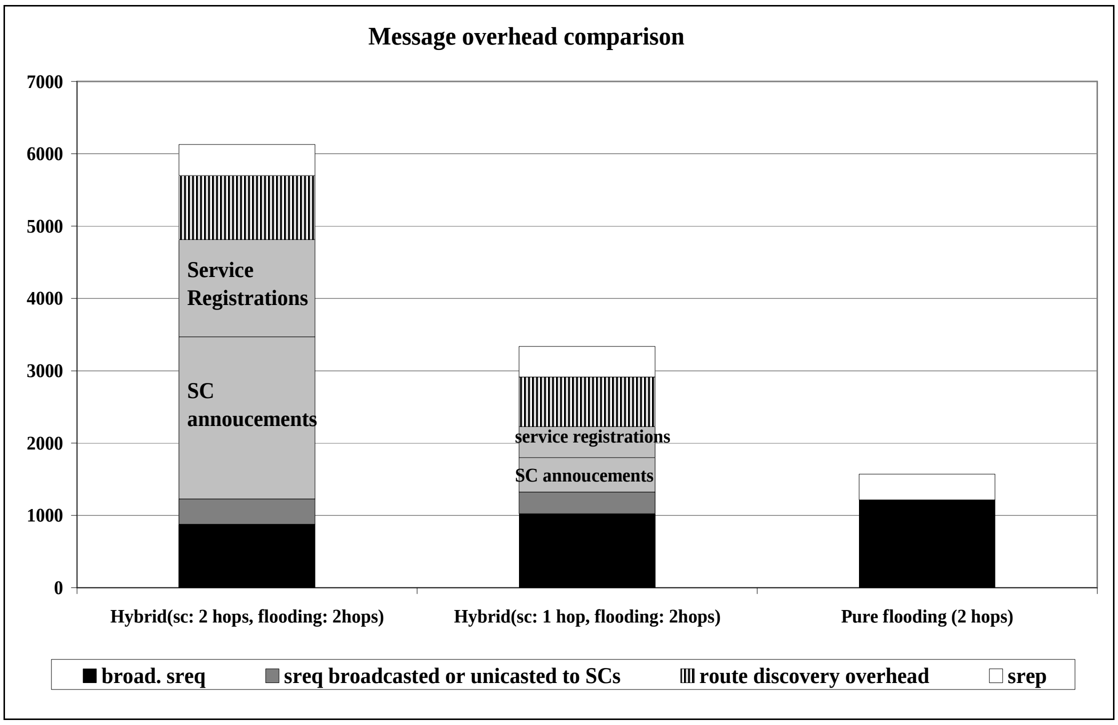

Figure 38: Message overhead comparison between the pure flooding and the hybrid architecture for the dynamic network topology

As Figure 38 above shows, it comes at no surprise that adding service coordinators also results in a higher messaging overhead under the conditions of node mobility.

The proactive elements, namely SC announcements and service registrations, and the extra route discovery messages, are introduced, just as the situation with the static case.

8.3.3 RSR and message overhead relative to max moving speed

We can see from Figure 39 below that RSRs decrease as nodes move faster and faster for both service discovery architectures. Higher mobility causes more frequent broken routes, which decreases the RSR. Considering the way RSR is calculated, only successful service resolution followed by a successful access to the resolved server will be counted as a satisfied service request. Accordingly, even the desired service can be found, later access to the server may fail because the server has moved out of the active network area or the route to the server is broken because of one or more of the intermediate nodes has left the network in the meantime or the network is partitioned between the client and the server. For the hybrid architecture, the client usually has to initiate a route discovery to find a route to the resolved server. All the aforementioned cases might cause route replies to be dropped before reaching the client. In addition, the route discovery mechanism may cause the client to flood the route requests to the network up to several times before giving up. The increase in total number of route requests and decrease in total number of route replies as shown in Figure 41 on page 76 reflects this.

However, the pure flooding architecture appears to be more stable than the hybrid architecture. This is because nodes in pure flooding architecture don’t have to worry about getting any stale server information. It will always be the server itself that responds to a service request. This is not the case in hybrid architecture where service coordinators are involved. As stated in 8.1.3 on page 70, chances for a SC to give out stale server information are bigger if the network becomes more and more dynamic. The stale server information might decrease the RSR as stated in the end of section 8.1.3.

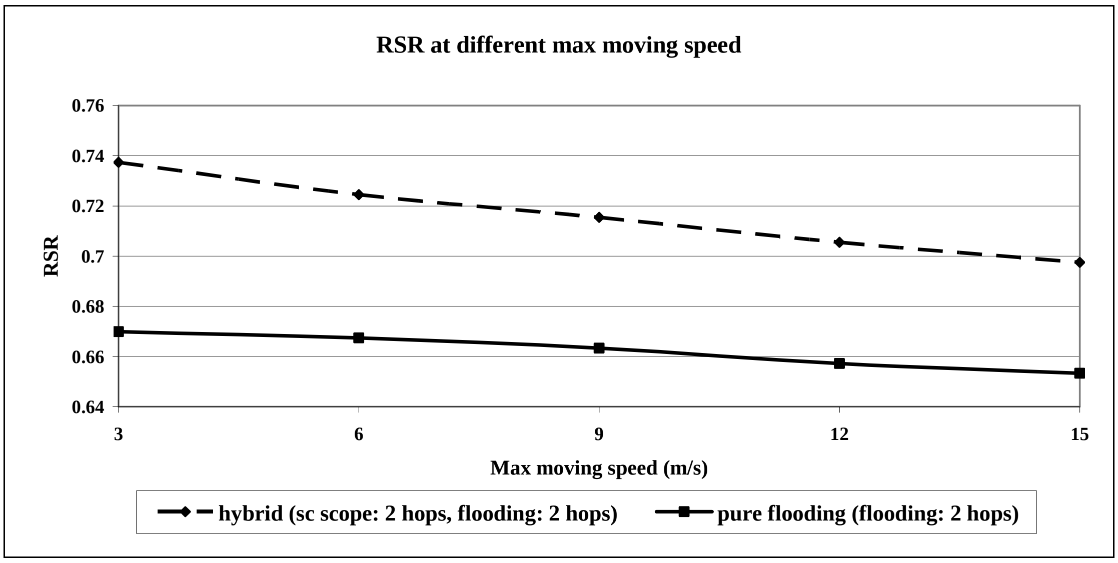

Figure 39: RSR relative to max moving speed

Figure 40 on the next page shows that there is an increase in the message overhead for the hybrid architecture as nodes moves faster and faster, while a slight decrease in the overhead for the pure flooding architecture.

The increase in overhead with the hybrid architecture is caused by the increase in route requests and service requests that have to be broadcasted to the SC as Figure 41 on the next page illustrates. One of the reasons for the increase in route requests has already been explained earlier in this section. The other reason for the increase in route requests is because of broken routes between the affiliated SC and the resolved server, which is caused by high mobility. Accordingly, if the SC lies between the client and the resolved server, the SC has to rebroadcast the route request from the client instead of responding on behalf of the server. The increase in the number of service requests that has to be broadcasted to the SC[2] is due to the fact that high mobility will cause the route between the client and its affiliated SC to be more easily broken. So instead of unicasting the service request to its affiliated SC, the client has to broadcast it.

The slight decrease in overhead with the pure flooding architecture is caused by the decrease in total service requests broadcasted. The mobility model random way point shipped with the GloMoSim [36] has a tendency to move nodes closer towards the center of the region. Table 16 below lists the message overhead for the pure flooding with a single-hop flooding scope under both static and dynamic topology. The decrease in service requests is due to the fact that nodes outside the active routing area are not allowed to initiate any service requests. Even with a decrease in total service requests broadcasted, there is an increase in total number of service replies generated. Since the flooding scope is one hop, only a node’s one-hop neighbors can generate service replies. Mobility model in the simulator has caused nodes to move closer to each other, thus more servers can respond to the service requests. The increase in moving speed will speed up this process, which causes nodes to move closer to each other faster towards the center. This increases chances for services to be found at a closer server (i.e. services can be found in node’s one-hop neighborhood instead of two-hop neighborhood), which leads to fewer retransmissions, thus fewer service requests broadcasted. This is a flaw that needed to be corrected in GloMoSim [36], which is out of the scope of this thesis. Another possible explanation could be that high mobility may more easily cause network partition, which hinder the higher hop (i.e. 2-hop) broadcasts to be done. This will also lead to fewer service requests being broadcasted. More exact analysis requires a thorough study of the mobility model shipped with the simulator, as well as a study of nodes’ behaviors under different moving speed. These studies are considered to be one of the future work.

	flooding scope (hops)	service requests broadcasted	service replies unicasted
Pure flooding (without mobility)	1	223	80
Pure flooding (with mobility)	1	213	109

Table 16: The effect of mobility

[2] Another alternative would be to make the client fall back on the pure flooding approach immediately if there is no valid route cached for the affiliated SC.

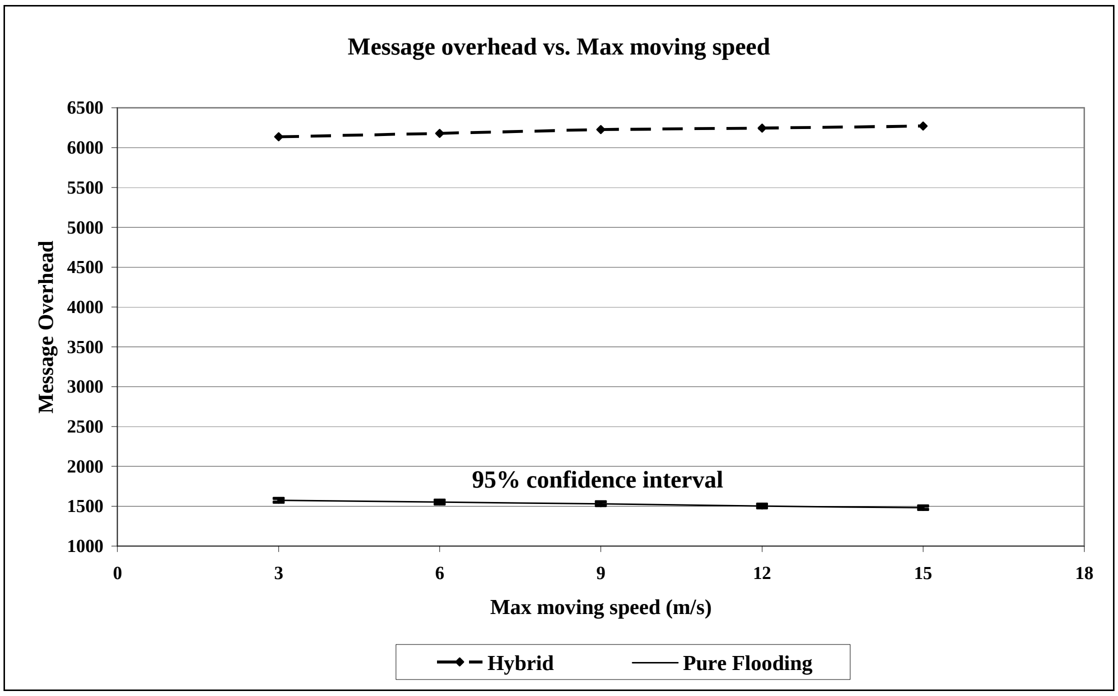

Figure 40: Message overhead relative to max moving speed

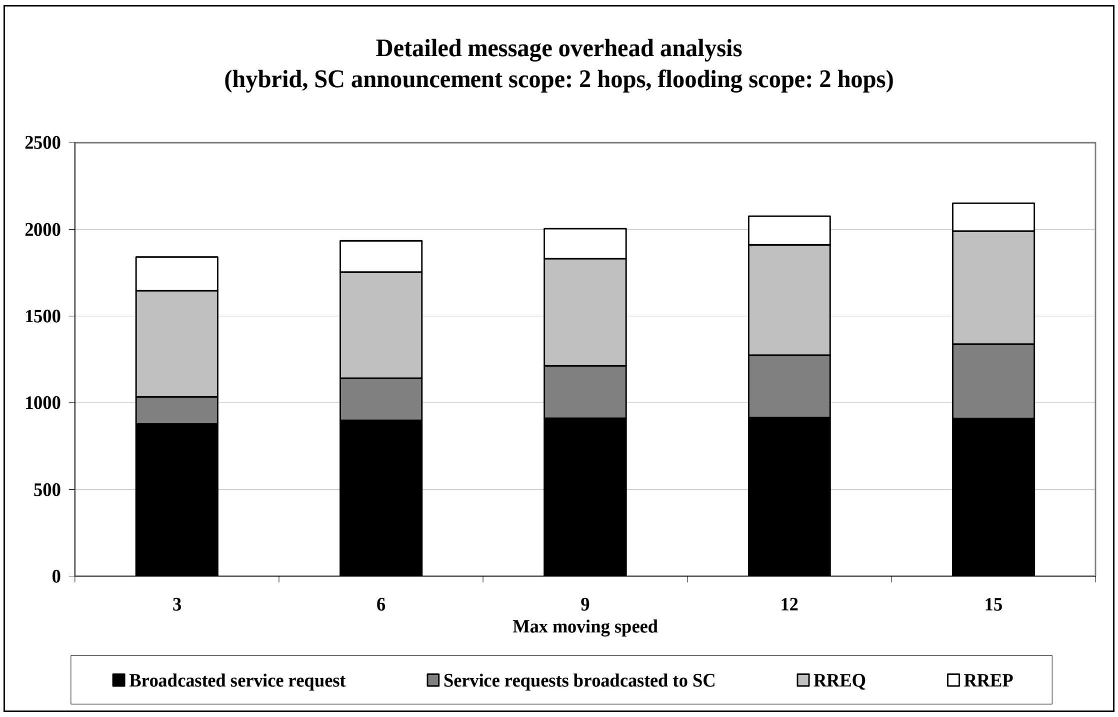

Figure 41: Detail of message overhead by type for the hybrid architecture vs. max moving speed

8.4 Discussion

8.4.1 Tradeoff between the service availability and the message overhead

Hypothesis: The conclusions drawn in section 7.4 for the network with static topology will still hold for the network with dynamic topology.

As stated earlier, with a static network topology, the introduction of service coordinators with the hybrid architecture increases the service availability (i.e. RSR) as well as the message overhead, as compared to the pure flooding architecture. The dynamic network topology is no exception. The key question is still whether the increased message overhead can be justified by the improved service availability when mobility is added to the network.

We have come to a conclusion in section 7.4 that the pure flooding service discovery is more preferable than the hybrid service discovery architecture when no mobility is involved. We will in this section come to a conclusion for the dynamic topology.

Table 17 through Table 19 list the RSR values and the message overhead for the two architectures under the conditions of node mobility at three different server densities. Some of these results have already been presented in graphs and tables above.

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.191	261
Flooding	2	-	0.348	1507
Flooding	3	-	0.49	2888
Flooding	4	-	0.6	4428
Flooding	5	-	0.68	5910
Hybrid	1	1	0.253	1634
Hybrid	2	1	0.402	3437
Hybrid	3	1	0.539	5362
Hybrid	1	2	0.372	4137
Hybrid	2	2	0.481	5973
Hybrid	3	2	0.596	7909
Hybrid	1	3	0.456	6617
Hybrid	2	3	0.548	8302
Hybrid	3	3	0.637	10176

Table 17: Overall comparison between the pure flooding ("Flooding") and the hybrid architecture ("Hybrid") at a 5% server density for the dynamic network topology

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.458	323
Flooding	2	-	0.67	1574
Flooding	3	-	0.786	3000
Flooding	4	-	0.844	4585
Flooding	5	-	0.872	6097
Hybrid	1	1	0.528	2046
Hybrid	2	1	0.706	3343
Hybrid	3	1	0.806	4684
Hybrid	1	2	0.61	4973
Hybrid	2	2	0.737	6136
Hybrid	3	2	0.819	7363
Hybrid	1	3	0.628	8106
Hybrid	2	3	0.75	9227
Hybrid	3	3	0.816	10400

Table 18: Overall comparison between the pure flooding ("Flooding") and the hybrid architecture ("Hybrid") at a 20% server density for the dynamic network topology

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.715	383
Flooding	2	-	0.875	1574
Flooding	3	-	0.925	2950
Flooding	4	-	0.94	4434
Flooding	5	-	0.942	5850
Hybrid	1	1	0.757	2314
Hybrid	2	1	0.885	3190
Hybrid	3	1	0.927	4186
Hybrid	1	2	0.791	5939
Hybrid	2	2	0.884	6698
Hybrid	3	2	0.921	7578
Hybrid	1	3	0.763	10127
Hybrid	2	3	0.857	10979
Hybrid	3	3	0.893	12001

Table 19: Overall comparison between the pure flooding ("Flooding") and the hybrid architecture ("Hybrid") at a 40% server density for the dynamic network topology

8.4.1.1 Comparing the pure flooding and the hybrid architecture at a 20% SC density

8.4.1.1.1 Considering single-hop SC announcement scope

Again, we begin with the hybrid architectures with SC announcement scopes of one hop. For a server density of 5%, adding service coordinators with a single-hop SC announcement scope to the pure flooding architecture with various flooding scopes increases the service availability. As we can see from Table 20 below, the RSR is increased from 0.191 to 0.253 for the flooding scope of 1 hop, from 0.348 to 0.402 for the flooding scope of two hops and from 0.49 to 0.539 for a flooding scope of three hops. Though RSRs are improved by 32%, 16% and 10%, respectively. The increase in message overhead is much higher. For the single-hop flooding scope, the increase is 526%. For the flooding scopes of 2 hops and 3 hops, the increases are 128% and 86%, respectively. The increase in the RSR can hardly be justified by the increase in the message overhead.

From Table 20, we can also see that the pure flooding scheme with a flooding scope of 2 hops will outperform the hybrid scheme that has a flooding scope of 1 hop in terms of higher RSR value and lower message overhead. A pure flooding scheme of 3-hop (4-hop) flooding scope will be superior to the hybrid architecture that has a 2-hop (3-hop) flooding scope.

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.191	261
Flooding	2	-	0.348	1507
Flooding	3	-	0.49	2888
Flooding	4	-	0.6	4428
Hybrid	1	1	0.253	1634
Hybrid	2	1	0.402	3437
Hybrid	3	1	0.539	5362

Table 20: Comparing pure flooding ("Flooding") and the hybrid architecture ("Hybrid") with server density of 5%, SC announcement scope of 1 hop. The values are extracted from Table 17.

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.458	323
Flooding	2	-	0.67	1574
Flooding	3	-	0.786	3000
Flooding	4	-	0.844	4585
Hybrid	1	1	0.528	2046
Hybrid	2	1	0.706	3343
Hybrid	3	1	0.806	4684

Table 21: Comparing pure flooding ("Flooding") and the hybrid architecture ("Hybrid") with server density of 20%, SC announcement scope of 1 hop. The values are extracted from Table 18.

It comes at no surprise that the same pattern can be seen for other server densities. For example, with a server density of 20%, the increase in message overhead is too large to be justified by the minimal increase in RSR (Table 21).

Again, the hybrid scheme with a flooding scope of 1 hop is outperformed by the pure flooding scheme with a flooding scope of 2 hops, which has a higher RSR of 0.67 and less message overhead of 1574. The hybrid schemes with flooding scopes of 2 hops and 3 hops are inferior to the pure flooding schemes with flooding scopes of 3 hops and 4 hops respectively.

A service density as high as 40% (Table 22) does not change the superiority of the pure flooding scheme over the hybrid scheme.

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.715	383
Flooding	2	-	0.875	1574
Flooding	3	-	0.925	2950
Flooding	4	-	0.94	4434
Hybrid	1	1	0.757	2314
Hybrid	2	1	0.885	3190
Hybrid	3	1	0.927	4186

Table 22: Comparing pure flooding ("Flooding") and the hybrid architecture ("Hybrid") with a server density of 40%, SC announcement scope of 1 hop. The values are extracted from Table 19.

Sub-conclusion I: The sub-conclusions drawn in section 7.4.2.1.1 on page 63 also apply under the conditions of node mobility.

8.4.1.1.2 Considering multi-hop SC announcement scope

Comparing with the pure flooding schemes, the hybrid architectures with the 3-hop SC announcement scope have increased the RSR by 139%, 57% and 30% for flooding scopes of one, two and three hops, respectively as illustrated in Table 23 on the next page. However, the message overhead of the hybrid schemes is increased by 2435%, 451% and 252% respectively, which is a tremendous degradation in message overhead.

It comes at no surprise that the pure flooding scheme is still superior to the hybrid scheme when the SC announcement scope is of multiple hops under the conditions of node mobility. For example, as we see in Table 23, a pure flooding scheme with a flooding scope of 4 hops outperforms all the hybrid architecture schemes with multi-hop SC announcement scopes presented in the table with higher RSR and lower message overhead except for two. Though the pure flooding with a flooding scope of 4 hops exhibits a higher message overhead than the hybrid scheme with a flooding scope of 1 hop and an SC announcement scope of 2 hops, yet the increase in the message overhead of 7% can be justified by the improvement in the RSR of 61%. The hybrid architecture is thus still inferior. Actually, a pure flooding with a flooding scope of 3 hops can already outperform this hybrid scheme. The hybrid scheme with an SC announcement scope of 3 hops and a flooding scope of 3 hops offers a higher RSR than the pure flooding scheme with a 4-hop flooding scope. However, the increase in message overhead of 130% can hardly be adjusted by the improvement in the RSR of 6%. Therefore, this hybrid scheme is still inferior to the pure flooding scheme with a 4-hop flooding scope. We can also choose to further expand the flooding scope of the pure flooding scheme to five hops, so that it will also beat the hybrid scheme in the RSR.

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.191	261
Flooding	2	-	0.348	1507
Flooding	3	-	0.490	2888
Hybrid	1	2	0.372	4137
Hybrid	2	2	0.481	5973
Hybrid	3	2	0.596	7909
Hybrid	1	3	0.456	6617
Hybrid	2	3	0.548	8302
Hybrid	3	3	0.637	10176
Flooding	4	-	0.6	4428
Flooding	5	-	0.68	5910

Table 23: Comparing pure flooding scheme (“Flooding”) and different hybrid architecture schemes ("Hybrid") with increasing SC announcement scopes. The values are extracted from Table 17, which covers a service density of 5%

The same conclusions are drawn for other service densities as shown in Table 18 and Table 19 on page 78. It is easy to see that the flooding architecture outperforms the hybrid architecture. For both service densities, the pure flooding architecture with a 4-hop flooding scope will outperform all hybrid architectures with multi-hop SC announcement scopes presented in the tables.

A further observation made from Table 19 is that at a server density as high as 40%, an increase in the SC announcement scope will result in a decrease in service availability, an effect contrary to the initial purpose of adding the service coordinators. The reason for this is quite similar as that stated in section 8.1.1 on page 69 concerning increasing the SC density at a high server density level. The decrease is caused by the stale server information passed out by the service coordinators. The increase in the SC announcement scope is somewhat similar to the increase in the SC density. More servers will be registered with the SCs especially at a high server density level. In addition, increasing the SC announcement scope will also cause more clients to affiliate with service coordinators. Due to node mobility, an SC might hold server information that is stale (i.e. the server is outside the SC announcement scope or outside the active network area) yet still valid as far as the lifetime for the entry is concerned. Since more clients will utilize the SCs for service discovery, chances for passing out such stale server information will increase. The worst case for getting such stale server information will be that the resolved server has left the active network area or in a network partition that is beyond reach from the client. This will then result in a failure in service discovery. This explains the decrease in service availability along with the increase in SC announcement scope.

Sub-conclusion II: The sub-conclusions drawn in section 7.4.2.1.2 on page 64 also apply for the network with dynamic topology.

8.4.1.2 Comparing the pure flooding and the hybrid architecture at a 30% SC density

We have shown in section 7.4.2.2 on page 64 that increasing the SC density doesn't change the fact that the pure flooding scheme is superior to the hybrid scheme for the static case. The same applies for the mobility case too. We have shown in Table 24 below an example that covers a server density of 20% and an increased SC density of 30%.

Increasing the SC density will in most cases increase both the service availability and the message overhead. Comparing, for example, the hybrid schemes with single-hop flooding scopes in Table 24 with those in Table 18 on page 78. RSRs are increased by 3.9%, 1.7% and 0.7% for flooding scopes of one, two and three hops, respectively. However, message overhead are increased by 24%, 15% and 10.5%, respectively. At a higher SC announcement scope (i.e. 3 hops), the increased SC density even causes a reduction in the RSR. The reasons are the same as stated in sections 8.1.1 on page 69 and 8.4.1.1.2 on page 81 about increasing the SC density or SC announcement scope at a high server density.

Service discovery architecture	Flooding scope (hops)	SC announcement scope (hops)	Service availability (RSR)	Message overhead
Flooding	1	-	0.458	323
Flooding	2	-	0.67	1574
Flooding	3	-	0.786	3000
Flooding	4	-	0.844	4585
Hybrid	1	1	0.549	2531
Hybrid	2	1	0.718	3847
Hybrid	3	1	0.812	5176
Hybrid	1	2	0.62	6102
Hybrid	2	2	0.748	7337
Hybrid	3	2	0.824	8566
Hybrid	1	3	0.625	9803
Hybrid	2	3	0.748	10995
Hybrid	3	3	0.81	12488

Table 24: Comparing pure flooding ("Flooding") and different hybrid architecture schemes ("Hybrid") under the conditions of node mobility, which covers a service density of 20% and an increasing SC density of 30%

As for the static case, we can always find a pure flooding scheme that outperforms the hybrid scheme with higher RSR and less message overhead. The pure flooding scheme with a 2-hop flooding scope that offers a RSR of 0.67 and a message overhead of 1574 outperforms the hybrid scheme with a single-hop SC announcement scope and a single-hop flooding scope that has a RSR of 0.549 and message overhead of 2531. Similarly, the pure flooding scheme with a 3-hop flooding scope is superior to the hybrid scheme with a singe-hop SC announcement scope and a 2-hop flooding scope. The pure flooding scheme with a 4-hop flooding scope will outperform all the other hybrid schemes as presented in Table 24 above.

The mobility does not change the fact that increasing the flooding scope of the hybrid architecture offers a better performance than increasing the SC announcement scope does. This confirms the benefit of the pure flooding scheme also under the conditions of node mobility.

Conclusion: All the sub-conclusions drawn above still hold for an increased SC density.

8.4.2 Comparing the static and the dynamic network topology

8.4.2.1 RSR comparison

After adding mobility to the network, we can see that service availability (i.e. RSR) is higher than that of the static case. However, this does not argue for a preference for mobility, simply because the increase in the RSR is caused by the following reasons.

- This is due to the way RSR is calculated. If all the servers that offer the requested service are outside the active network area at the end of the service request cycle, that particular service request is not taken into account for the RSR calculation.
- The active network area is smaller than the original network used for the static case. Accordingly, nodes are closer to each other, which leads to a higher service availability.
- The mobility model random waypoint shipped with the GloMoSim [36] has a tendency to move nodes towards the center of the region, thus making nodes closer to each other and easier to discovery each other

8.4.2.2 Message overhead comparison

Table 25 on the next page shows that with the mobility added to the nodes in the network, the total message overhead is increased for the hybrid architecture. The table also shows the major message types that have contributed to the increase in message overhead. Because of nodes mobility, the route between a client and its affiliated SC might be broken at the time when the client triggered a service request. Accordingly, the service request will have to be broadcasted to the SC as contrary to the static case where service requests will always be unicasted by the client to its affiliated service coordinator if the route is not broken by other means.

The increase in route requests is due to the following:

1. Since some service coordinators might give out stale server information for servers that don’t exist (i.e. outside the active network area). When the client tries to find a route to the resolved server, the route request mechanism causes the client to flood the route request to the whole network up to several times before giving up, thus increasing the total route requests.
2. If the SC lies on the route between the client and the resolved server, the SC can usually reply the route request on behalf of the server itself, thus reducing the flooding scope of the route request. However, in the mobility case, the route between the SC and the registered server might be broken, so that route request has to be re-broadcasted.

There is no significant change in the message overhead for the pure flooding architecture and the difference is mainly caused by the flaw in the mobility model shipped with the simulator.

	SC announcement scope (hops)	Flooding scope (hops)	Total message overhead	Service requests broadcasted to SC	Route request
Hybrid (with mobility)	1	2	3343	174	497
Hybrid (static)	1	2	2480	0	195

Table 25: Overall message overhead comparison between static and mobility case at a server density of 20%

8.4.2.3 Negative effects caused by the service coordinator under the conditions of node mobility

There are several negative effects caused by service coordinators in a dynamic network.

1. Stale server information will be passed out by the service coordinator, which might decrease the service availability. The more dynamic the network is, the big the risk is. This is shown in Figure 34 on page 71.
2. Due to node mobility, routes between the client and its affiliated SC break easily. This forces the client to broadcast the service request to the affiliated SC instead of unicasting. The benefit of using the SCs is thus reduced.

8.4.3 Considering service request interval

The trade-off between a hybrid and a pure flooding architecture is largely dependent on the number and the pattern of service requests generated. For our simulations, 20% of the nodes are actively doing service discoveries every twenty seconds. The elements, namely SC announcements and service registrations, introduced by the hybrid architecture will be justified by the increased number of service requests generated. The client density of 20% and the service request interval of 20s are relatively high values compared to the real life scenarios. Still, the simulation results favor the pure flooding architecture to the hybrid architecture. In real life scenarios, fewer nodes might engage in service discovery activities and clients may prefer longer communication sessions with the resolved servers. This will favor the pure flooding architecture even more, simply because for the first, the elements (i.e. SC announcements and service registrations) introduced by the hybrid architecture will consume a fixed amount of bandwidth, which can hardly be justified by the infrequent service requests and for the second, a pure flooding architecture makes the service discovery purely on-demand, which reduces the message overhead caused by service discoveries to the minimum.

Conclusion for the chapter: The pure flooding service discovery architecture is still preferable to the hybrid service discovery architecture under the conditions of node mobility.

Chapter 9
Conclusion and Future work

By means of simulations, we have shown that the increase in service availability (i.e. RSR) by adding service coordinators is negligible compared to the extra message overhead it caused. In addition, one can always find a pure flooding service discovery scheme with a reasonable service request flooding scope that outperforms the hybrid scheme with higher RSR and less message overhead. Accordingly, the pure flooding service discovery architecture is preferable to the hybrid architecture on reactively routed MANETs. The conclusion applies to both the static and the dynamic network topology.

Even on a proactively routed MANET, a pure flooding architecture might still be preferable, although the routing effects are lower. Firstly, it is considerably less complex. Secondly, the hybrid approach may call for a separate complex mechanism for electing service coordinators, which might require a substantial amount of network resources.

There are several issues that deserve further investigation:

- An opportunity that has not been explored in this thesis is to allow caching of service binding information on intermediate nodes that forwards service replies and on the requestor nodes themselves. This seems to be a promising compromise between the pure flooding and the hybrid architectures for on-demand MANETs, and the issue deserves further investigation.
- As mentioned earlier, the placement of service coordinators relative to the clients and the servers are critical to the network performance. Instead of nodes taking on roles as service coordinators statically, a lightweight, dynamic mechanism for election of service coordinators is desired.
- The flaw in the mobility model (i.e. random waypoint) shipped with GloMoSim [36] should be fixed.
- Further studies about the effect of increasing the moving speed of nodes should be carried out.
- Further evaluations of pure flooding and hybrid architectures under different mobility patterns, server and client distribution patterns should be considered.

References

[1] G. Y. He, “Destination-Sequenced Distance Vector (DSDV) Protocol”, Networking Laboratory, Helsinki University of Technology.

[2] T. Clausen, P. Jacquet, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum and L. Viennot, “Optimized Link State Routing Protocol”, Internet Draft, IETF MANET Working Group, draft-ietf-manet-olsr-07.txt, November 2002.

[3] C. Perkins, E. Royer, and S. Das, “Ad hoc On-Demand Distance Vector (AODV) Routing”, IETF Internet Draft, draft-ietf-manet-aodv-12.txt, November 2002.

[4] J. Broch, D. B. Johnson, D. A. Maltz, “The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks”, IETF Internet Draft, draft-ietf-manet-dsr-01.txt, December 1998.

[5] Z. J. Haas, M. R. Pearlman and P. Samar, “The Zone Routing Protocol (ZRP) for Ad Hoc Networks”, IETF Internet Draft, draft-ietf-manet-zone-zrp-04.txt, July 2002.

[6] E. M. Royer, “A review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks”, IEEE Personal Communications, April 1999.

[7] E. Guttman, C. Perkins, J. Veizades, and M. Day, “Service Location Protocol, Version 2”, IETF, RFC 2608, June 1999, available at http://www.rfc-editor.org/rfc/rfc2608.txt.

[8] E. Guttman, “Service Location Protocol: Automatic Discovery of IP Network Services”, IEEE Internet Computing, http://computer.org/internet, July 1999.

[9] C. Perkins and E. Guttman, “DHCP Options for Service Location Protocol”, IETF, RFC 2610, June 1999, available at http://www.rfc-editor.org/rfc/rfc2610.txt.

[10] Sun Microsoft, “Jini Network Technology”, http://www.jini.org.

[11] Salutation Consortium, “Salutation Architecture Specification v2.1 (Part-1)”, http://www.salutation.org/specordr.htm.

[12] Sun Microsystems, “RPC: Remote Procedure Call Protocol Specification”, Network Working Group-RFC 1050, April 1998, available at http://www.faqs.org/rfcs/rfc1050.html.

[13] Microsoft Corporation, “Universal Plug-and-Play (UPnP) Forum”, http://www.upnp.org.

[14] R. Droms, “Dynamic Host Configuration Protocol”, Network Working Group-RFC 2131, March 1997.

[15] E. Guttman, S. Cheshire (chairs), “Zero Configuration Networking (zeroconf)”, June 1999.

[16] Y. Goland, T. Cai, P. Leach, Y. Gu, S. Albright, “Simple Service Discovery Protocol/1.0”, available at http://www.ietf.org/internet-drafts/draft-cai-ssdp-v1-02.txt.

[17] W3N Note, “Simple Object Access Protocol (SOAP) 1.1”, available at http://www.w3.org/TR/SOAP.

[18] J. Cohen, S. Aggarwal, Y. Goland, “General Event Notification Architecture Base: Client to Arbiter”, draft-cohen-gena-p-base-01.txt.

[19] “Extensible Markup Language”, http://www.xml.com.

[20] Bluetooth SIG, “Specification of the Bluetooth System - Core, Version 1.1 volume 1, 2001. Part E”.
[21] Bluetooth SIG, “Specification of the Bluetooth System - Profiles, Version 1.1 volume 1, 2001. Part K: 2”.
[22] C. Bettstetter and C. Renner, “A comparison of service discovery protocols and implementation of the service location protocols”, TUM, Munich, Germany.
[23] S. Vinoski, “CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments”, IEEE Communication Magazine, Vol.14, No. 2, February 1997.
[24] Web Service Basics, http://msdn.microsoft.com/webservices/understanding/webservicebasics/.
[25] W3C Note, “Web Services Description Language (WSDL) 1.1”, http://www.w3c.org/TR/wsdl.
[26] “UDDI Version 3 specification”, http://www.uddi.org.
[27] K. Geihs, “Middleware Challenges Ahead”, IEEE Computer, June 2001.
[28] C-K. Toh, “Ad Hoc Mobile Wireless Networks: Protocols and Systems”, Prentice Hall, NJ 2002, pp. 231-242.
[29] D. Chakraborty, A. Joshi, Y. Yesha and T. Finin, “GSD: A Novel Group-based Service Discovery Protocol for MANETS”, Computer Science And Electrical Engineering, University of Maryland, Baltimore County.
[30] R. Koodli, C. E. Perkins, “Service Discovery in On-Demand Ad Hoc Networks. Manet”, Working Group Internet Draft, draft-koodli-manet-servicediscovery-00.txt, 02 October 2002.
[31] P. E. Engelstad et al., “Service Discovery and Name resolution Architectures in On-Demand MANETs”, Proceedings of 2003 Workshop on Mobile Wireless Networks (MWN), Providence, Rhode Island, May 19 – 22, 2003.
[32] P. E. Engelstad, G. Egeland, R. Koodli, C. Perkins, “Name Resolution in On Demand MANETs and over External IP Networks”, IETF Internet draft, draft-engelstad-manet-name-resolution-01.txt, February 2004 (Work in Progress)
[33] P. E. Engelstad, D. V. Thanh, G. Egeland, “Name Resolution in Mobile Ad-hoc Networks”, Proceedings of IEEE 2003 International Conference on Telecommunication, ICT’2003, Tahiti, February 23-28, 2003.
[34] G. E. Güichal, “Service Location Architectures for Mobile Ad-hoc Networks”, Master Thesis, Georgia Institute of Technology, July 2001.
[35] A. Gulbrandsen, P. Vixie, L. Esibov, “A DNS RR for specifying the location of services (DNS SRV)”, RFC 2782, Internet Engineering Task Force (IETF), February 2000.
[36] M. Gerla, X, Zeng, R. Bagrodia, “GloMoSim: A library for parallel simulation of large-scale wireless networks”, Proc. 12th Workshop on Parallel and Distributed Simulations, 1998.
[37] Sun Microsoft, “Java Remote Method Invocation”, http://java.sun.com/products/jdk/rmi/.

Appendix A
Article submitted for review

Evaluation of Service Discovery Architectures on reactively routed MANETs

Yan Zheng, Paal Engelstad
University of Oslo (UniK) / Telenor R&D, 1331 Fornebu, Norway
yanz@ifi.uio.no, Paal.Engelstad@telenor.com

Abstract— **Discovery of services and other named resources is expected to be a crucial feature for the usability of mobile ad-hoc networks (MANETs). Different types of service discovery architectures are distinguished by the extent that service coordinators (SCs) are implemented in the network. SCs are nodes that hold a central repository for caching attributes and bindings of services of servers located in its neighborhood. This paper describes and evaluates the performance of different architectures in terms of service availability, messaging overhead and latency. The paper shows that on a reactively routed MANET where the service discovery mechanism has a direct impact on the routing protocol, the routing effects have a major impact on the evaluation result. The paper also demonstrates the benefits of combining the service discovery with route discovery, especially in on-demand MANETs where reactive routing protocols are being used.**

Keywords - ***simulations, ad hoc networks, service discovery architectures, cross-layer optimisations.***

I. Introduction

Discovery of services and other named resources is anticipated to be a crucial feature for the usability of mobile ad-hoc networks (MANETs). In this dynamic environment different nodes offering different services may enter and leave the network at any time. Efficient and timely service discovery is a prerequisite for good utilization of shared resources on the network.

On a MANET, any node may in principle operate as a server and provide its services to other MANET nodes or as a service requestor and uses the service discovery protocol to discover available services on the network and their service attributes. This includes IP addresses, port-numbers and protocols that enable the client to initiate the selected service on the appropriate server.

The Internet community has not yet reached a consensus on one particular service discovery protocol that is likely to be supported by most Internet hosts. There are a number of proposed service discovery mechanisms - such as Jini [1], Service Location Protocol (SLP) [2], Salutation Protocol [3] and UPnP/SSDP [4].

As a slight simplification, one may say that are all these protocols are based on two baseline mechanisms for management of service discovery information:

1. Information about services offered on the network is stored on one or a few centralized nodes, referred to as Service Coordinators (SCs) in this paper.
2. Information about each service is stored on the node that is offering the service.

In this paper we define the service discovery architectures with regard to these two mechanisms. A solution only based on the first mechanism is referred to as a *service coordinator based architecture,* while a solution only based on the second mechanism is referred to as a *distributed query-based architecture*. Finally, a solution based on a mixture of both the first and the second mechanism is referred to as a *hybrid service location architecture*.

Existing service discovery mechanisms are normally designed with a fixed network in mind, and might not fit well to MANETs. MANETs are normally highly dynamic and without any preexisting infrastructure. These characteristics call for particular considerations. Hence, before a service discovery mechanism for ad-hoc networks can be designed or selected, one need to evaluate what kind of service discovery architectures are most suitable for ad-hoc networks.

Güichal [5] undertakes an analysis of different service discovery architectures based on simulations. The work concludes that the hybrid architecture normally outperforms both the service coordinator based and the distributed query-based architecture. The distributed query-based architecture is the second best choice, and yields less messaging overhead. Despite this, the work concludes that the hybrid architecture gives an overall better performance, because it yields higher service availability.

A shortcoming of the simulations from Güichal's work is that they do not take the importance of underlying routing into consideration. This assumption might be appropriate when a proactive routing protocol is being used, because with proactive routing the traffic patterns and service discovery search patterns do not influence the amount of routing messages.

With a reactive routing protocol, on the contrary, this assumption does not hold, and the simulation results are not applicable. Data traffic will trigger messaging by the reactive routing protocol, and service discovery messages will increase the routing overhead. It is therefore anticipated that the routing overhead would be higher with the hybrid architecture than with the distributed query-based architecture, simply because the hybrid architecture proved to require more messages on the network.

In this paper we make a new comparison between the distributed query-based architecture and the hybrid architecture, to determine if Güichal's conclusion still holds in a reactively routed network. Both the overhead of the service discovery mechanism, as well as the additional routing that is triggered by the mechanism, is taken into account. To minimize the routing overhead triggered by service discovery, we have used the optimisation methods proposed in [6] and [7].

When we evaluate the two architectures, we look for a user-friendly solution that gives a high level of service availability, low discovery delay, and so forth. At the same time, we want a network-friendly solution, i.e. with low messaging overhead and with little additional complexity added to the network. To a certain degree, it is also possible to increase the user-friendliness at the cost of introducing more messaging.

In section 2 we present relevant work related to service discovery in ad-hoc networks. Section 3 presents the simulation setup. Section 4 presents simulation results that compare the distributed query-based and the hybrid service discovery architecture. Discussion of the results is presented in Section 5. Section 6 presents a discussion for the dynamic network topology. Conclusions are drawn in Section 7, and directions for further work are discussed.

II. Related work

A. Service discovery architectures

C. K. Toh [8] has outlined different service discovery architectures for managing service information on MANETs. In terms of service discovery, a MANET node may act as a *Client* (or *Service Requestor*) that wants to discover a type of service, a *Server* (or *Service Provider*) that wants to make its services available to other MANET nodes, or a *Service Coordinator (SC)* that assists with service discovery. SCs are nodes that hold a central repository for caching *Service Bindings*, which maps a service name to an IP address(es) and a port number(s) that can be used to initiate the service.

Three possible service discovery architectures are outlined in [8]:

- *Service coordinator based architecture*: Certain nodes in the MANET are chosen to be service coordinators, a role quite similar to the DA in SLP [2] or the lookup service in Jini [1]. SCs announce their presences to the network periodically by flooding SC announcement messages. The flooding is limited to a certain number of hops, determined by the *SC announcement scope* parameter. A service provider (i.e. server) that receives SC announcements unicasts *Service Registration* messages to register periodically its services and access information with SCs in its surroundings. A service requestor (i.e. client) that has received SC announcement messages may unicast a Service Request to a selected SC to discover desired services. The SC responds with a unicast Service Reply. The selected SC is referred to as an affiliated SC.
- *Distributed query-based architecture*: This architecture contains no SC. Instead, a service requestor (i.e. client) floods the Service Request throughout its surroundings in the network. The flooding is limited by the *flooding scope* parameter. Each service provider responds to a Service Request for its own services with a unicast Service Reply.
- *Hybrid service location architecture*: This architecture combines the above two architectures. Service providers within the announcement scope of one or more SCs will register with them their available services and access information, but must also be ready to respond to flooded service requests. When a service requestor unicasts a Service Request to its affiliated SC in line with the *Service Coordinator based architecture*, the SC responds with a positive or negative Service Reply. However, if there is no SC in the service requestor's surroundings or if the affiliated SC returned a negative Service Reply, the service

requestor will simply fall back to the *Distributed Query based architecture*. Both SCs and servers may respond to a flooded Service Request with a positive Service Reply that matches the requested service.

This paper evaluates the performance of the two latter architectures in a reactively routed MANET.

B. Group-based service discovery protocol

D. Charkraborty et al. proposed a novel group-based service discover protocol (GSD) [9] for MANET. The protocol is based on peer-to-peer caching of the service advertisements, which are associated with an advertising radius, i.e. every node maintains a cache of all the services within a certain number of hops (the advertising radius). Services are described using service groups (e.g. Service/Hardware/IO-Service/Printer-Service).

The local cache will be exploited first when a service is requested at the application level in order to enhance efficiency for service discovery. When no matching service is found in the cache, a service request will be broadcasted to the network.

D. Charkraborty et al. have in [9] also proposed a group-based selective forwarding concept for such broadcasted service requests, i.e. the service request is forwarded only to those nodes that have seen one or more of the service groups to which the request belongs. This information is conveyed through the periodic service advertisements. In this way, the network will not be inundated with request messages, and the bandwidth usage will be spared.

C. Name Resolution and Service Lookups

A solution to name resolution in on-demand MANETs has been proposed in ([10], [11]). The main idea is to streamline name resolution with the underlying reactive routing protocol (e.g. AODV [12], DSR [13] or TORA [14]). The objective is to obtain a bandwidth-efficient scheme that reduces the number of broadcasted discovery messages to a minimum.

It has also been proposed to bundle simple service name lookups together with this name resolution mechanism ([6]). This is parallel to DNS SRV lookups for simple service discovery on the fixed Internet [15]. It allows a service name to be resolved into an IP address and a transport protocol number to be used to initiate the service. The transport protocol type is normally encoded into the service name.

Figure 1 shows how service discovery can be streamlined with the reactive routing protocol in the case where the client is affiliated with a service coordinator based on the ideas from [6] [10] [11]. This is the model used for simulation in this paper.

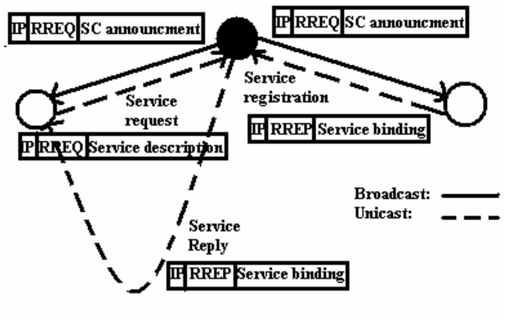

Figure 1: Streamlining the service discovery with the reactive routing.

Service discovery messages can be carried in routing message extensions in the form of a type and a type-specific value as proposed in the AODV specification [12]. Service requests and SC announcements are carried in RREQ extensions, service replies and service registrations, on the other hand, are carried in RREP extensions.

The advantages of piggybacking service discovery on routing messages in this way are:

1. Reverse routes to the service requestor are established along with the service request so that no additional route discovery is necessary to relay the service reply back to the requestor.
2. Forward routes to the SC are established along with the SC announcements so that service requests and service registrations can be unicasted to the SC.
3. A forward route is established along with the service reply so that no additional route discovery is necessary for further communication with the node issuing the reply.

D. SLP-based service discovery

R. Koodli et al. have in their internet draft [7] proposed a similar solution to service discovery in on-demand MANETs, where service discovery requests and replies are also carried as an extension to RREQs and RREPs in (Figure 1).

The proposed mechanism for service discovery specifies message formats that are designed to inter-operate with the Service Location Protocol (SLP) [2]. Thus, it has more capabilities to accommodate advanced service discovery than the DNS-SRV-based scheme for simple service name resolution proposed in [11] has. A drawback, however, is that it requires additional software implemented on the MANET nodes, which may increase complexity and slow deployment.

III. Simulation setup

Simulations were done on the well-known simulator GloMoSim [16], which is shipped with an AODV [12] module.

The simulated network contains 50 nodes randomly located in a 300mx300m squire. A two ray propagation model for radio waves as well as omni-directional antennas were used at the physical level. The radio range of the node is set to be 50 meters. The MAC protocol used is IEEE 802.11. AODV and UDP are used as the underlying reactive routing protocol and transport layer protocol respectively. Every simulation is repeated 500 times with different seed values.

There are two different types of services in the network. A node is selected as a client, a server and/or a service coordinator based on the density parameter fed in through the configuration file. The selection was done using a random number generator shipped with GloMoSim [16]. SC election mechanism is out of the scope of this paper.

The two service discovery architectures simulated are distributed query-based architecture and hybrid architecture. The architectures can be tuned with (at least) two parameters:

- SC announcement scope: This scope regulates the extent to which a service coordinator announcement can propagate in terms of hops. This parameter is used only in the hybrid architecture.
- Flooding scope: This scope determines how far a service request will be broadcasted in the network in terms of hops. This parameter is used in both architectures. In a hybrid architecture, a service requestor will fall back to use a distributed-query based architecture by broadcasting the service request based on this flooding scope if no affiliated service coordinator is heard or when a negative service reply is returned from the affiliated service coordinator.

In the simulations, 20% of the nodes will function as clients and actively initiate service requests every twenty seconds. The time for the first service request is randomly and individually generated for every client node. The SC announcement interval is set to be the same as the route timeout value (i.e. 10S) recommended by the ADOV [12] specification. The reason for setting the SC announcement interval alike the route timeout value is because it yields minimal routing message overhead.

IV. Simulation results

A. Service availability (SA)

The service availability (SA) is defined as:

$$SA = \frac{Number\ of\ positive\ service\ replies}{Total\ number\ of\ service\ requests\ issued\ by\ all\ clients\ in\ the\ network}$$

A positive service reply means not only the resolution of a service type to a valid service binding (server address, port number), but also a successful contact to this server via the given access information (i.e. A route to the resolved server can be found).

Figure 2 shows how the presences of service coordinators (i.e. for the hybrid architecture) influence the service availability. As we can see from the figure, the introduction of service coordinators does improve the service availability. Depending on the announcement scope of the service coordinator, the service availability is improved by 8,3% and 20,8% respectively at a server density of 5%.

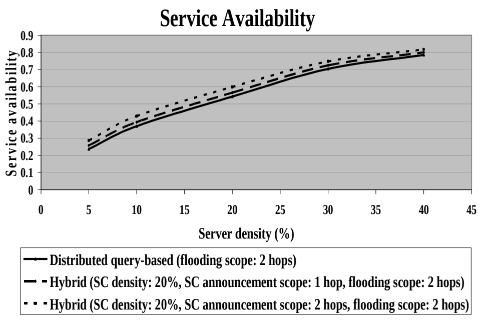

Figure 2: Service availability comparison between the Distributed query-based and the Hybrid architecture.

The reason that SCs improve the service availability is revealed in Figure 3 and Figure 4. Figure 3 illustrates a scenario with a flooding scope of 2 hops and a SC announcement scope of 1 hop. Without service coordinator functionality implemented on the black node in Figure 3, the server would be unreachable from the client. With SC functionality on the black node, on the other hand, the server will be able to register its service with the service coordinator and the client's service request will be able to reach the service coordinator, which will respond on behalf of the server.

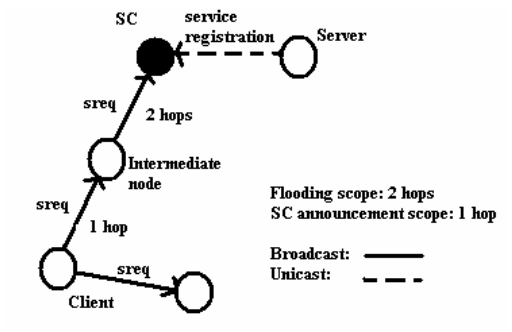

Figure 3: The effect of the SC. Scenario 1

Figure 4 shows a similar scenario, however, here the SC announcement scope is 2 hops. Without the service coordinator, neither client 1 nor client 2 will be able to find the server. But with the help of service coordinator functionality implemented on the black node, both clients can direct service requests to their affiliated service coordinator i.e. the black node, which has cached the server information.

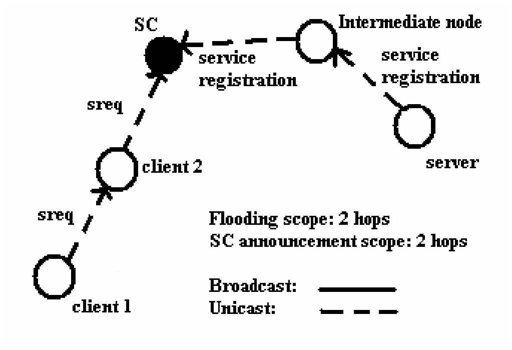

Figure 4: The effect of the SC. Scenario 2

As expected, our simulations confirm the results obtained in previous work [5], i.e. service availability is indeed higher with the hybrid approach.

B. Message overhead

All the non-data messages that are transmitted in the network by all the nodes at the network level are considered to be message overhead. The overhead is counted as the total number of packets over each hop (i.e. the total number of packets times the average number of hops traversed by the packets).

As pointed out in [5], the introduction of service coordinators introduces extra message overhead to the network, in terms of service announcements, service registrations and those related to service lookups. However, routing overheads triggered by these messages are not taken into account in [5]. Here, our analysis differs from [5], as we also take routing messages into account.

Although the introduction of service coordinators does increase the service availability, Figure 5 shows that it also results in a much higher level of messaging overhead. Service coordinators have introduced two proactive elements to the network, namely SC announcements and service registrations. These messages will take up a fixed bandwidth regardless of whether there exist service discoveries or not.

From Figure 5, we can also see that there is no message overhead caused by route discoveries for the distributed query-based architecture. This is because in the distributed query-based architecture, it is always the service provider itself that responds to the service request and a forward route to the service provider is established along with the service reply [11]. Accordingly, no additional route discovery is needed for the client to access the server after the resolution. However, in the hybrid architecture, service coordinators are expected to respond to the service requests. Accordingly, forward routes are only established towards the service coordinators, not the service providers, so an extra round of route discovery is needed in order to access the server after the resolution.

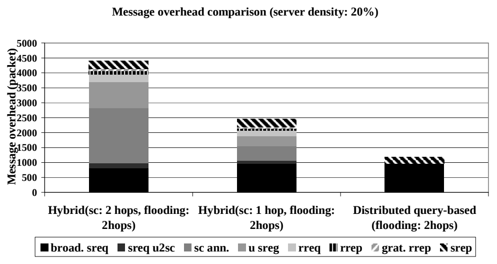

Figure 5: Detailed comparison of message overhead by message type

C. Latency

Figure 6 shows the comparison of the service discovery latency (i.e. from the moment a node generates a service request until that node receives a positive service binding) between the distributed query-based architecture and the hybrid architecture. The introduction of service coordinators does minimize the latency, because many of the service requests can be satisfied at the service coordinators, which are often closer to the client than servers themselves. In addition, these service requests are unicasted to the service coordinators, thus no delay is caused by any additional broadcast jitter. The increase in number of servers has enhanced the chances for the client to find the matching service at the service coordinator or at a closer server, which results in a decreasing latency.

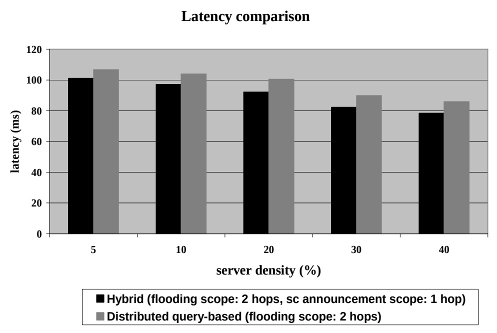

Figure 6: Latency comparison between the distributed query-based and the hybrid architecture.

Service discovery is normally a step that users go through as part of the initial service initiation. For example: users would normally accept a second of delay when retrieving search results on the (e.g. a Google lookup) or for setting up an IP Telephony call. Figure 6 shows that the service discovery latency is considerably lower than this. Furthermore, the differences in delays between the two architectures are only in the order of a few milliseconds and should be considered negligible in this context. Thus, delay is not a factor distinguishes the one service discovery architecture from the other.

V. Discussion of results

Our objective is to optimize the benefits of additional service availability against the cost of additional overhead. The key question to be answered is whether the increased service availability can be justified by the increase in message overhead.Table 1 below lists the service availability values and the message overhead for the two architectures at a 5% server density.

Service discovery architecture	Flooding scope (hops)	SC ann. scope (hops)	Service avail-ability	Message overhead
Distributed	1	-	0.144	263
Distributed	2	-	0.237	1178
Distributed	3	-	0.313	2001
Distributed	4	-	0.38	2799
Distributed	5	-	0.431	3526
Distributed	6	-	0.476	4109
Hybrid	1	1	0.166	1208
Hybrid	2	1	0.258	2456
Hybrid	3	1	0.33	3544
Hybrid	1	2	0.228	2921
Hybrid	2	2	0.287	4235
Hybrid	3	2	0.357	5413
Hybrid	1	3	0.288	4356
Hybrid	2	3	0.334	5609
Hybrid	3	3	0.382	6773

Table 1: Overall comparison between the distributed query-based ("Distributed") and the hybrid architecture ("Hybrid") at a 5% server density

A. Single-hop SC announcement scope

Adding service coordinators with 1-hop announcement scopes to the distributed query architecture with various flooding scopes increases service availabilities. As we can see from Table 1, service availabilities are increased from 0.144 to 0.166 for a 1-hop flooding scope (i.e. an increase ratio of 1.15), from 0.237 to 0.258 for a 2-hop flooding scope (i.e. an increase ratio of 1.09) and from 0.313 to 0.33 for a 3-hop flooding scope (i.e. an increase ratio of 1.05). However, with such minimal increase ratios in service availability, the message overhead of the hybrid scheme is tremendous higher. The message overhead of the hybrid architecture with a flooding scope of one hop is almost 5 times as much as that of the distributed query-based architecture with the same flooding scope. The message overhead of the other two hybrid architectures is also doubled compared to the pure flooding architectures with correspondent flooding scopes.

From Table 1, we can see that by expanding the flooding scope of the distributed query-based scheme from 1 hop to 2 hops, it will outperform the hybrid scheme with a 1-hop flooding scope and a 1-hop SC announcement scope. The distributed query-based scheme exhibits higher service availability, i.e. 0.237 as opposed to 0.166, and less message overhead, i.e. 1178 as opposed to 1208. By further expanding the flooding scope of the distributed query-based scheme, the hybrid scheme with multi-hop flooding scope will also be outperformed. We can see that the hybrid architecture with a 2-hop flooding scope is inferior to the distributed query-based architecture with a 3-hop flooding scope. Similarly, the hybrid architecture with a 3-hop flooding scope is inferior to the distributed query-based architecture with a 4-hop flooding scope.

B. Multi-hop SC announcement scopes

By increasing the SC announcement scope of the hybrid architecture, the service availability will be improved slightly. The downside is a considerable degradation in message overhead.

Comparing with the distributed query-based architectures, the hybrid architectures with the 3-hop SC announcement scope have increased the service availability by 100%, 41%% and 22% for flooding scopes of one, two and three hops, respectively as illustrated in Table 1. However, it also increases the message overhead by 1556%, 376% and 238%, respectively.

Since we showed that the distributed query-based architecture was superior to the hybrid architecture for a SC announcement scope of one hop, it comes at no surprise that the same is the case when the SC announcement scope is of multiple hops. For example, as we see in Table 1, a distributed query-based scheme with a 4-hop flooding scope outperforms all the hybrid architecture schemes with multi-hop SC announcement scopes presented in the table. Though the hybrid scheme with a 3-hop SC announcement and 3-hop flooding scope offers a higher service availability than that offered by the distributed query-based scheme with a 4-hop flooding scope. The increase in service availability of 0,5% is negligible compared to the increase in message overhead of 140%. Therefore, this hybrid scheme is still inferior to the distributed query-based scheme with 4-hop flooding scope. By further expanding the flooding scope of the distributed query-based scheme to five hops, it will then offer higher service availability and less message overhead than the aforementioned hybrid scheme.

C. Higher server densities and higher SC densities

The same patterns were shown for other service densities and SC densities. More simulation results are provided in [17].

Figure 7 shows how service availability and message overhead are affected by the increase in the SC density. The five points on the curve represent, from the bottom up, SC densities of 0% (i.e. distributed query-based architecture), 10%, 20%, 30% and 40%, respectively. Here we can see a relatively vertical line, which indicates the increase in the SC density has a much less influence on the service availability than on the message overhead. One of the reasons for this almost negligible improvement in service availabilities as the SC density increases is that as more and more nodes take on roles as service coordinators, many may have their impacts on overlapping areas. However, the client will still direct its service request to its old affiliated service coordinator unless either the new one is better compared to the old one based on certain criterion (less hop count etc.) or the old one fails in one way or another. There may exist many such service coordinators in the network, which are just present without actually participating in any service discovery process, hence not improving the service availability. However, these service coordinators are still consuming lots of network bandwidth by periodically broadcasting SC announcements and receiving solicited service registrations, which explains the noticeable increase in message overhead.

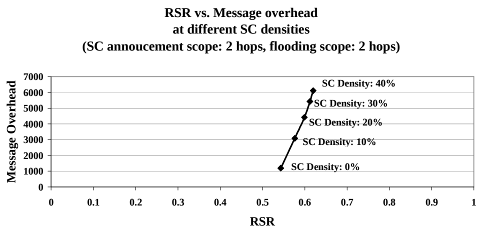

Figure 7: The effect of increasing the SC density.

D. Considering SC announcement interval and service request interval

The trade-off between a hybrid and a distributed query-based architecture is largely dependent on the number and the pattern of service requests generated. For our simulations, 20% of the nodes are actively doing service discoveries every twenty seconds. The elements, namely SC announcements and service registrations, introduced by the hybrid architecture will be justified by the increased number of service requests generated. The client density of 20% and the service request interval of 20s are relatively high values compared to the real life scenarios. Still, the simulation results favor the distributed query-based architecture to the hybrid architecture. In real life scenarios, fewer nodes might engage in service discovery activities and clients may prefer longer communication sessions with the resolved servers. This will favor the distributed query-based architecture even more, simply because for the first, the elements (i.e. SC announcements and service registrations) introduced by the hybrid architecture will consume a fixed amount of bandwidth, which can hardly be justified by the infrequent service requests and for the second, a distributed query-based architecture makes the service discovery purely on-demand, which reduces the message overhead caused by service discoveries to the minimum.

VI. Discussion for the dynamic network topology

Our evaluation up till now has not considered node mobility. When mobility is added in our simulations, the original 300x300m network was partitioned into two parts as Figure 8 shows. Nodes that move outside the active network area are considered to have left the network and will not participate in any network activity. The mobility model used for simulations is random waypoint.

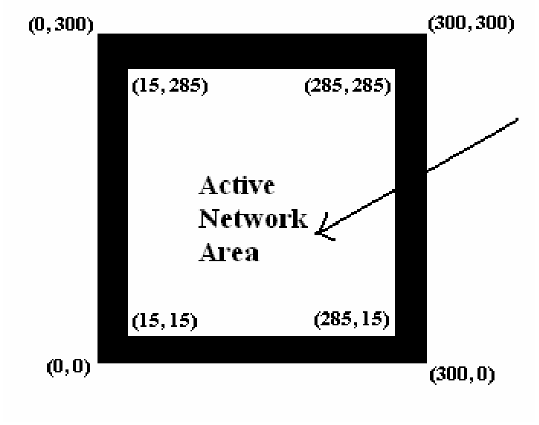

Figure 8: Network partitions under the conditions of mobility

Adding service coordinators to the dynamic networks shows the same effect as with the static networks. Service availability is indeed higher with the hybrid architecture, but it also exhibits a higher messaging overhead. The proactive elements (SC announcements, service registrations) and the extra route discovery messages are introduced, just as the situation with the static case. Due to the space limit, figures and simulation data are not presented here, but can be found in [17].

However, there are some negative effects caused by service coordinators under the conditions of mobility.

1. Stale server information will be passed out by the service coordinator, which might decrease the service availability. The more dynamic the network is, the big the risk is.
2. Due to nodes mobility, routes between the client and its affiliated service coordinator break easily. This forces the client to broadcast the service requests to the affiliated service coordinator instead of unicasting. The benefit of using service coordinators is thus reduced.

VII. Conclusions and future work

By means of simulations, we have shown that the increase in service availability by adding service coordinators is negligible compared to the extra message overhead it causes. In addition, one can always find a distributed query-based service discovery scheme with reasonable service request flooding scope that outperforms the hybrid scheme with higher service availability and less message overhead. Accordingly, the distributed query-based service discovery architecture is preferable to the hybrid architecture on reactively routed MANETs. The conclusion applies to both the static and the dynamic network topology.

It is also interesting to note that by taking into account the additional routing protocol overhead induced by the service discovery architecture, we reach different conclusions than those of previous work [5] where these important effects have not been taken into consideration.

Even on a proactively routed MANET, a distributed query-based architecture might still be preferable, although the routing effects are lower. Firstly, it is considerably less complex. Secondly, the hybrid approach may call for a separate complex mechanism for electing service coordinators, which might require a substantial amount of network resources.

An opportunity that has not been explored in this paper is to allow caching of service binding information on intermediate nodes that forward service replies and on the requestor nodes themselves. This seems to be a promising compromise between the distributed query-based and the hybrid architectures for on-demand MANETs, and the issue deserves further investigation. Another issue that worth further research is the design of a lightweight, dynamic mechanism for election of service coordinators so as to fully exploit their benefits, which might result in an improved performance of a hybrid architecture.

[1] Jini Network Technology, Sun Microsystems, http://www.jini.org.

[2] Guttman, E., Perkins, C., Veizades, J., Day, M., "Service Location Protocol, version 2", *RFC 2608, Internet Engineering Task Force (IETF)*, June 1999.

[3] Salutation Consortium, "Salutation Architecture Specification Version 2.1", 1999.

[4] Universal Plug-and-Play (UPnP) Forum, Microsoft Corporation, http://www.upnp.org.

[5] Güichal, G.E., "Service Location Architectures for Mobile Ad-hoc Networks", Master Thesis, Georgia Institute of Technology, July 2001.

[6] Engelstad et al. “Service Discovery and Name resolution Architectures in On-Demand MANETs”, Proceedings of 2003 Workshop on Mobile Wireless Networks (MWN), Providence, Rhode Island, May 19 – 22, 2003.

[7] Koodli, R., and Perkins, C.E., "Service Discovery in On Demand Ad Hoc Networks", *IETF Internet draft*, draft-koodli-manet-servicediscovery-00.txt, October 2002 (Work in Progress).

[8] Toh, C.-K., "Ad Hoc Mobile Wireless Networks. Protocols and Systems", *Prentice Hall PTR*, New Jersey, 2002, pp. 231-242.

[9] D. Chakraborty et al., “GSD: A Novel Group-based Service Discovery Protocol for MANETs”. *Computer Science And Electrical Engineering.* University of Maryland, Baltimore County.

[10] Engelstad, P.E., Egeland, G., Koodli, R., Perkins, C. "Name Resolution in On Demand MANETs and over External IP Networks", *IETF Internet draft*, draft-

engelstad-manet-name-resolution-01.txt, February 2004 (Work in Progress).

[11] Engelstad P.E., Thanh, D.V., Jønvik, T.E., "Name Resolution in Mobile Ad-hoc Networks". *Proceedings of IEEE 2003 International Conference on Telecommunication*, *ICT'2003*, Tahiti, February 23 – 28, 2003.

[12] Perkins, C.E., Royer, E.M., Das, S.R., "Ad-hoc On Demand Distance Vector (AODV) Routing", *IETF Internet draft*, draft-ietf-manet-aodv-13.txt, February 2003 (Work in Progress).

[13] Johnson, D.B., Maltz, D.A., Hu, Y.C.., "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)", *IETF Internet draft,* draft-ietf-manet-dsr-09.txt, April 2003 (Work in Progress).

[14] Park, V.D., and Corson, S.M., "Temporally-Ordered Routing Algorithm (TORA) version 1: Functional Specification", *IETF Internet Draft*, November 2000 (Work in progress).

[15] Gulbrandsen, A., Vixie, P., Esibov, L., "A DNS RR for specifying the location of services (DNS SRV)", *RFC 2782, Internet Engineering Task Force (IETF)*, February 2000.

[16] Gerla, M., Zeng, X. Bagrodia, R., "GloMoSim: A library for parallel simulation of large-scale wireless networks", *Proc. 12th Workshop on Parallel and Distributed Simulations, 1998.*

[17] Zheng, Y, "Service discovery in Mobile Ad hoc Networks*", Master thesis, University of Oslo, July 2004.*

Appendix B
Sample configuration file for simulations

```
SIMULATION-TIME 500S
TERRAIN-DIMENTIONS (300,300)
NUMBER-OF-NODES 50
NODE-PLACEMENT RANDOM
MOBILITY NONE
MOBILITY RANDOM-WAYPOINT
MOBILITY-WP-PAUSE 10S
MOBILITY-WP-MIN-SPEED 0
MOBILITY-WP-MAX-SPEED 3
MOBILITY-POSITION-GRANULARITY 0.3333
PROPAGATION-LIMIT –111.0
PROPAGATION-PATHLOSS TWO-RAY
NOISE-FIGURE 10.0
TEMPERATURE 290.0
RADIO-TYPE RADIO-ACCNOISE
RADIO-FREQUENCY 2.4E9
RADIO-BANDWIDTH 2000000
RADIO-RX-TYPE SNR-BOUNDED
READIO-RX-SNR-THRESHOLD 9.1
RADIO-TX-POWER –6.974
RADIO-ANNTENNA-GAIN 0.0
RADIO-RX-SENSITIVITY –91.0
RADIO-RX-THRESHOLD –81.0
MAC-PROTOCOL 802.11
NETWORK-PROTOCOL IP
NETWORK-QUEUE-SIZE 100
ROUTING-PROTOCOL AODV

PRINT_CLIENT 20 20S 128  /* client, client density, service request interval, packet size*/
SERVICE_COORDINATOR 20 0  /* SC, SC density, SC announcement scope */
PRINT_SERVER 20  /* server, server density */
```

Appendix C
Partial implementation codes for service discovery on reactively routed MANETs

A.1 Initiate RREQs with or without extensions

```
void RoutingAodvInitiateRREQ(GlomoNode *node,
                             NODE_ADDR destAddr,
                             NODE_ADDR nextHop,
                             GlomoAppServiceDesc *serviceDesc)
{
  GlomoNetworkIp *ipLayer = (GlomoNetworkIp *) node->networkData.networkVar;
  GlomoRoutingAodv *aodv = (GlomoRoutingAodv *) ipLayer->routingProtocol;
  Message *newMsg;
  AODV_RREQ_Packet *rreqPkt;
  char *pktPtr;
  int pktSize = sizeof(AODV_RREQ_Packet);
  int ttl;
  char clockStr[GLOMO_MAX_STRING_LENGTH];
  BOOL isSCAnn = destAddr == SC_ANN_ADDRESS;
  NewMsg = GLOMO_MsgAlloc(node,GLOMO_MAC_LAYER, 0, MSG_MAC_FromNetwork);
  GLOMO_MsgPacketAlloc(node, newMsg, pktSize);
  pktPtr = (char *) GLOMO_MsgReturnPacket(newMsg);
  rreqPkt = (AODV_RREQ_Packet *) pktPtr;
  RoutingAodvIncreaseSeq(node);
  rreqPkt->pktType = AODV_RREQ;

  if (nextHop == ANY_DEST)
  {
    rreqPkt->bcastId = RoutingAodvGetBcastId(node);
  }
  else
  {
    rreqPkt->bcastId = -1;
  }
  rreqPkt->destAddr = destAddr;
  rreqPkt->destSeq = RoutingAodvGetSeq(destAddr, &aodv->routeTable);
  if (rreqPkt->destSeq == -1)
  {
    rreqPkt->unknownSeqNo=TRUE;
  }
  else
  {
    rreqPkt->unknownSeqNo=FALSE;
  }
  rreqPkt->srcAddr = node->nodeAddr;
  rreqPkt->srcSeq = RoutingAodvGetMySeq(node);
  rreqPkt->lastAddr = node->nodeAddr;
  rreqPkt->hopCount = 0;
```

```
if (serviceDesc == NULL)
{
  if (!isSCAnn)
  {
    rreqPkt->numExtensions = 0;
  }
  else
  {
    rreqPkt->numExtensions = 1;
    rreqPkt->reqExt[0].extType = SERVICE_COORDINATOR_ANN;
  }
}
else
{
  rreqPkt->numExtensions = 1;
  rreqPkt->reqExt[0].extType = SERVICE_REQUEST;
  memset(&rreqPkt->reqExt[0].ServDesc, 0, sizeof(GlomoAppServiceDesc));
  memcpy(&rreqPkt->reqExt[0].ServDesc, serviceDesc, sizeof(GlomoAppServiceDesc));
}

if (destAddr == SERVICE_RESOLUTION_ADDRESS)
{
  ttl = FLOOD_SREQ_SCOPE;
}
else if (destAddr == SC_ANN_ADDRESS)
{
  ttl = GLOMO_GetSCAnnDiameter(&node->affiliatedServiceCoordinator);
}
else if (nextHop == ANY_DEST)
{
  if (RoutingAodvCheckSent(destAddr, &aodv->sent))
  {
    ttl = RoutingAodvGetTtl(destAddr, &aodv->sent);
    RoutingAodvIncreaseTtl(destAddr, &aodv->sent);
  }
  else
  {
    if (RoutingAodvCheckRouteEntryExist(destAddr,&aodv->routeTable))
    {
      ttl = RoutingAodvGetHopCount(destAddr, &aodv->routeTable);
      ttl += TTL_INCREMENT;
    }
    else
    {
      ttl = TTL_START;
    }
    RoutingAodvInsertSent(destAddr, ttl, &aodv->sent);
    RoutingAodvIncreaseTtl(destAddr, &aodv->sent);
  }
}
else
{
  assert(serviceDesc != NULL && destAddr != SERVICE_RESOLUTION_ADDRESS
          && nextHop!=ANY_DEST);
  ttl = 1;
}

if (nextHop == ANY_DEST)
{
  if (ttl > 0)
```

```
    {
      NetworkIpSendRawGlomoMessage(node, newMsg, ANY_DEST,
                                   CONTROL, IPPROTO_AODV, ttl);
    }
    else
    {
      return;
    }
    RoutingAodvInsertSeenTable(node, node->nodeAddr,
                               rreqPkt->bcastId, &aodv->seenTable);
    if (destAddr == SERVICE_RESOLUTION_ADDRESS)
    {
      RoutingAodvSetTimerForService(node, MSG_NETWORK_CheckServiceReplied,
                                    serviceDesc,
                                    (clocktype)RoutingAodvGetRingTraversalTime(ttl));
      aodv->stats.broadcastedServiceRequest++;
    }
    else if (destAddr == SC_ANN_ADDRESS)
    {
      aodv->stats.numSCAnnouncementSent++;
    }
    else
    {
      if (ttl==NET_DIAMETER)
      {
        RoutingAodvSetTimer(node, MSG_NETWORK_CheckReplied, destAddr,
                            (clocktype)NET_TRAVERSAL_TIME);
        RoutingAodvIncreaseTimes(destAddr,&aodv->sent);
      }
      else
      {
        RoutingAodvSetTimer(node, MSG_NETWORK_CheckReplied, destAddr,
                            (clocktype)RoutingAodvGetRingTraversalTime(ttl));
      }
      if (serviceDesc != NULL)
      {
        RoutingAodvInsertServer(node, destAddr, serviceDesc,&aodv->serverRoute);
        aodv->stats.broadcastedSC++;
      }
      else
      {
        aodv->stats.numRequestSent++;
      }
    }
  }
  else
  {
    NetworkIpSendRawGlomoMessageToMacLayer(node, newMsg, nextHop, CONTROL,
                                           IPPROTO_AODV, ttl,
                                           DEFAULT_INTERFACE, nextHop);
    aodv->stats.unicastedSC++;
    RoutingAodvSetTimerForService(node, MSG_NETWORK_CheckSCReplied, serviceDesc,
                                  (clocktype)RoutingAodvGetRingTraversalTime
                                   (RoutingAodvGetHopCount
                                    (destAddr, &aodv->routeTable)));
  }
}
```

A.2 Initiate RREPs with or without extensions

```
void RoutingAodvInitiateRREP(GlomoNode *node, Message *msg,
                             Service_BT_Node *binding, BOOL isSreq)
{
  GlomoNetworkIp *ipLayer = (GlomoNetworkIp *)node->networkData.networkVar;
  GlomoRoutingAodv *aodv = (GlomoRoutingAodv *)ipLayer->routingProtocol;
  Message *newMsg;
  AODV_RREQ_Packet *rreqPkt;
  AODV_RREP_Packet *rrepPkt;
  char *pktPtr;
  int pktSize = sizeof(AODV_RREP_Packet);
  rreqPkt = (AODV_RREQ_Packet *) GLOMO_MsgReturnPacket(msg);
  newMsg = GLOMO_MsgAlloc(node, GLOMO_MAC_LAYER, 0, MSG_MAC_FromNetwork);
  GLOMO_MsgPacketAlloc(node, newMsg, pktSize);
  pktPtr = (char *) GLOMO_MsgReturnPacket(newMsg);
  rrepPkt = (AODV_RREP_Packet *) pktPtr;

  if ((rreqPkt->unknownSeqNo == FALSE)
     &&(rreqPkt>destSeq->RoutingAodvGetMySeq(node)))
  {
    aodv->seqNumber = rreqPkt->destSeq;
  }
  rrepPkt->pktType = AODV_RREP;
  rrepPkt->srcAddr = rreqPkt->srcAddr;
  rrepPkt->destAddr = node->nodeAddr;
  rrepPkt->destSeq = RoutingAodvGetMySeq(node);
  rrepPkt->hopCount = 0;
  rrepPkt->lifetime = (clocktype)MY_ROUTE_TO;

  if (binding == NULL)
  {
    rrepPkt->numExtensions = 0;
  }
  else
  {
    int i = 0;
    assert(binding != NULL);
    rrepPkt->numExtensions = 1;
    memset(rrepPkt->repExt, 0, sizeof(RREP_EXT));
    if (isSreq)
    {
      rrepPkt->repExt[0].extType = SERVICE_REPLY;
    }
    else
    {
      rrepPkt->repExt[0].extType = SERVICE_ADV;
    }
    while (binding != NULL && i < 5)
    {
      memcpy(&rrepPkt->repExt[0].ServBinding[i], &binding->binding,
                sizeof(GlomoAppServiceBinding));
      binding = binding->next;
      i++;
    }
    rrepPkt->repExt[0].numChoise = i;
  }


   NetworkIpSendRawGlomoMessageToMacLayer(node, newMsg, rreqPkt->lastAddr,
                                           CONTROL, IPPROTO_AODV, 1,
                                                          DEFAULT_INTERFACE,
                                                          rreqPkt->lastAddr);
  if (rrepPkt->numExtensions == 0)
  {
    aodv->stats.numReplySent++;
  }
  else if (isSreq)
  {
    aodv->stats.numServiceReplySent++;
  }
  else
  {
    aodv->stats.numServiceRegistrationUnicasted ++;
  }
  GLOMO_MsgFree(node, msg);
}
```

A.3 Handle RREQs with or without extensions

```
void RoutingAodvHandleRequest(GlomoNode *node, Message *msg, int ttl)
{
  GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node->networkData.networkVar;
  GlomoRoutingAodv* aodv = (GlomoRoutingAodv *) ipLayer->routingProtocol;
  AODV_RREQ_Packet *rreqPkt = (AODV_RREQ_Packet *)
                                   GLOMO_MsgReturnPacket(msg);
  Message *newMsg;
  char serviceName[100];
  Service_BT_Node *binding1 = NULL;
  Service_BT_Node *binding2 = NULL;
  BOOL isSreq = rreqPkt->destAddr == SERVICE_RESOLUTION_ADDRESS ;
  BOOL isSCAnn = rreqPkt->destAddr == SC_ANN_ADDRESS;
  BOOL withExt = rreqPkt->numExtensions != 0;
  BOOL serviceFound = FALSE;

  RoutingAodvReplaceInsertRouteTable(node, rreqPkt->lastAddr, -1, FALSE, TRUE, 1,
                                        rreqPkt->lastAddr,
                                        simclock()+(clocktype)ACTIVE_ROUTE_TO);

  if(RoutingAodvLookupSeenTable(rreqPkt->srcAddr,rreqPkt->bcastId,&aodv->seenTable))
  {
    GLOMO_MsgFree(node,msg);
    return;
  }
  rreqPkt->hopCount++;
  if (rreqPkt->bcastId != -1)
  {
    RoutingAodvInsertSeenTable(node, rreqPkt->srcAddr, rreqPkt->bcastId,
                                 &aodv->seenTable);
  }

  if (!RoutingAodvCheckNbrExist(rreqPkt->lastAddr, &aodv->nbrTable))
  {
    RoutingAodvInsertNbrTable(rreqPkt->lastAddr, &aodv->nbrTable);
    RoutingAodvIncreaseSeq(node);
  }
  if (!RoutingAodvCheckRouteExist(rreqPkt->srcAddr,&aodv->routeTable))
  {
    clocktype lifetime = RoutingAodvGetMinimalLifetime(rreqPkt->hopCount);
    RoutingAodvReplaceInsertRouteTable(node, rreqPkt->srcAddr,rreqPkt->srcSeq, TRUE,
                                          TRUE, rreqPkt->hopCount,
                                          rreqPkt->lastAddr,lifetime);

  }
  else
  {
    clocktype lifetime = max(RoutingAodvGetLifetime(rreqPkt->srcAddr,&aodv->routeTable),
                                                               RoutingAodvGetMinimalLifetime
                                                               (rreqPkt->hopCount));
    int seq = RoutingAodvGetSeq(rreqPkt->srcAddr,&aodv->routeTable);
    RoutingAodvReplaceInsertRouteTable(node, rreqPkt->srcAddr, max(seq,rreqPkt->srcSeq),
                                          TRUE, TRUE, rreqPkt->hopCount,
                                          rreqPkt->lastAddr, lifetime);
  }


  if (isSreq || isSCAnn || ((node->nodeAddr == rreqPkt->destAddr)
      && node->isServiceCoordinator && withExt))
  {
    int i;
    for (i = 0; i<rreqPkt->numExtensions; i++)
    {
      if (rreqPkt->reqExt[i].extType == SERVICE_REQUEST)
      {
        memcpy(serviceName, rreqPkt->reqExt[i].ServDesc.nameStr,
                    rreqPkt->reqExt[i].ServDesc.nameLen+1);
        if (GLOMO_CheckBindingExist(&node->myService, serviceName))
        {
          binding1 = GLOMO_GetBinding(&node->myService, serviceName);
        }
        if  (GLOMO_CheckBindingExist(&node->cachedService, serviceName))
        {
          binding2 = GLOMO_GetBinding(&node->cachedService, serviceName);
        }
        if (binding1 != NULL)
        {
          binding1->next = binding2;
        }
        else
        {
          binding1 = binding2;
        }
        if (binding1 != NULL)
        {
          serviceFound = TRUE;
        }
        break;
      }
      else if (rreqPkt->reqExt[i].extType == SERVICE_COORDINATOR_ANN)
      {
        RoutingAodvHandleSCAnn(node, msg, ttl);
        break;
      }
    }

    if (serviceFound)
    {
      if(node->nodeAddr == rreqPkt->destAddr && node->isServiceCoordinator)
      {
        Service_BT_Node *cu = binding1;
        while (cu!= NULL)
      {
        aodv->stats.totalSCreplied ++;
        if (!RoutingAodvCheckServerWithinRange(node, cu->binding.serverAddr))
        {
          aodv->stats.falsePositive++;
        }
        cu = cu->next;
      }
    }
    RoutingAodvActivateRoute(rreqPkt->srcAddr, &aodv->routeTable);
    RoutingAodvSetTimer(node, MSG_NETWORK_CheckRouteTimeout,
                                rreqPkt->srcAddr, (clocktype)ACTIVE_ROUTE_TO);
    RoutingAodvInitiateRREP(node, msg, binding1, TRUE);
  }
  else if (isSCAnn)
  {
    RoutingAodvActivateRoute(rreqPkt->srcAddr, &aodv->routeTable);
    RoutingAodvSetTimer(node, MSG_NETWORK_CheckRouteTimeout,
                                rreqPkt->srcAddr, (clocktype)ACTIVE_ROUTE_TO);
    if (node->myService.size!=0)
    {
      RoutingAodvSetTimer(node, MSG_NETWORK_SendSregTimeout, rreqPkt->srcAddr,
                                  (clocktype)(1 * MICRO_SECOND));
    }
    if ((!node->isServiceCoordinator && ttl > 0)
        || (node->isServiceCoordinator
            &&  ttl > GLOMO_GetSCAnnDiameter(&node->affiliatedServiceCoordinator)))
    {
      RoutingAodvRelayRREQ(node, msg, ttl);
    }
  }
  else if (!isSreq)
  {
    assert(node->nodeAddr == rreqPkt->destAddr && node->isServiceCoordinator);
    RoutingAodvActivateRoute(rreqPkt->srcAddr, &aodv->routeTable);
    RoutingAodvSetTimer(node, MSG_NETWORK_CheckRouteTimeout,
                                rreqPkt->srcAddr, (clocktype)ACTIVE_ROUTE_TO);
    binding1 = (Service_BT_Node *)checked_pc_malloc(sizeof(Service_BT_Node));
    memcpy(binding1->binding.serviceName, serviceName, strlen(serviceName) + 1);
    binding1->binding.serverAddr = INVALID_ADDRESS;
    binding1->next = NULL;
    RoutingAodvInitiateRREP(node, msg, binding1, TRUE);
  }
  else if (ttl > 0)
  {
    RoutingAodvRelayRREQ(node, msg, ttl);
  }
  else
  {
    GLOMO_MsgFree(node, msg);
  }
  }
  else if (node->nodeAddr == rreqPkt->destAddr)
  {
    assert(!withExt);
    RoutingAodvActivateRoute(rreqPkt->srcAddr, &aodv->routeTable);
    RoutingAodvSetTimer(node, MSG_NETWORK_CheckRouteTimeout,
                                rreqPkt->srcAddr, (clocktype)ACTIVE_ROUTE_TO);
    RoutingAodvInitiateRREP(node, msg, NULL, FALSE);
  }
```

```
else if (rreqPkt->bcastId == -1)
{
  NODE_ADDR  nextHop = RoutingAodvGetNextHop(rreqPkt->destAddr,
                                             &aodv->routeTable);
  RoutingAodvActivateRoute(rreqPkt->srcAddr, &aodv->routeTable);
  RoutingAodvSetTimer(node, MSG_NETWORK_CheckRouteTimeout,
                      rreqPkt->srcAddr, (clocktype)ACTIVE_ROUTE_TO);
  if (nextHop != ANY_DEST)
  {
    rreqPkt->lastAddr = node->nodeAddr;
    NetworkIpSendRawGlomoMessageToMacLayer(node, msg, nextHop,
                                           CONTROL, IPPROTO_AODV, 1,
                                           DEFAULT_INTERFACE, nextHop);
    aodv->stats.unicastedSC++;
  }
  else
  {
    GLOMO_MsgFree(node,msg);
  }
}
else if ((RoutingAodvCheckRouteExist(rreqPkt->destAddr,&aodv->routeTable))
        &&(RoutingAodvIfSeqValid(rreqPkt->destAddr,&aodv->routeTable))
        &&(RoutingAodvGetSeq(rreqPkt->destAddr,&aodv->routeTable)
            >=rreqPkt->destSeq))
{
  if (withExt)
  {
    NODE_ADDR nextHop = RoutingAodvGetNextHop(rreqPkt->destAddr,
                                              &aodv->routeTable);
    if (nextHop != ANY_DEST)
    {
      rreqPkt->lastAddr = node->nodeAddr;
      NetworkIpSendRawGlomoMessageToMacLayer(node, msg, nextHop,
                                             CONTROL, IPPROTO_AODV, ttl,
                                             DEFAULT_INTERFACE, nextHop);
      aodv->stats.unicastedSC++;
    }
    else
    {
      RoutingAodvRelayRREQ(node, msg, ttl);
    }
  }
  else
  {
    RoutingAodvActivateRoute(rreqPkt->srcAddr, &aodv->routeTable);
    RoutingAodvSetTimer(node, MSG_NETWORK_CheckRouteTimeout,
                        rreqPkt->srcAddr, (clocktype)ACTIVE_ROUTE_TO);
    RoutingAodvInitiateRREPbyIN(node, msg);
    RoutingAodvInitiateGratuitousRREP(node, msg);
  }
}
else
{
  if(ttl==0)
  {
    GLOMO_MsgFree(node,msg);
  }
  else
  {
    RoutingAodvRelayRREQ(node,msg,ttl);
    }
  }

  while(RoutingAodvLookupBuffer(rreqPkt->lastAddr, &aodv->buffer))
  {
    newMsg = RoutingAodvGetBufferedPacket(aodv, FALSE, rreqPkt->lastAddr,
                                          &aodv->buffer);
    RoutingAodvTransmitData(node, newMsg, rreqPkt->lastAddr);
    aodv->stats.numDataSent++;
    RoutingAodvDeleteBuffer(rreqPkt->lastAddr, &aodv->buffer);
  }
  while(RoutingAodvLookupBuffer(rreqPkt->srcAddr, &aodv->buffer))
  {
    newMsg = RoutingAodvGetBufferedPacket(aodv, FALSE, rreqPkt->srcAddr,
                                          &aodv->buffer);
    RoutingAodvTransmitData(node, newMsg, rreqPkt->srcAddr);
    aodv->stats.numDataSent++;
    RoutingAodvDeleteBuffer(rreqPkt->srcAddr, &aodv->buffer);
  }
}
```

A.4 Handle RREPs with or without extensions

```
void RoutingAodvHandleRep(GlomoNode *node, Message *msg,
                           NODE_ADDR srcAddr, NODE_ADDR destAddr)
{
  GlomoNetworkIp *ipLayer = (GlomoNetworkIp *) node->networkData.networkVar;
  GlomoRoutingAodv *aodv = (GlomoRoutingAodv *) ipLayer->routingProtocol;
  AODV_RREP_Packet*rrepPkt = (AODV_RREP_Packet *)GLOMO_MsgReturnPacket(msg);
  BOOL causedNewRoute = FALSE;
  clocktype lifetime;
  Message *newMsg;
  Message *serviceRequest;
  Message *serviceReply;
  char serviceName[100];
  BOOL isSrep, isSreg;
  IpHeaderType *sreqIpHeader;
  IpHeaderType *srepIpHeader;
  TransportUdpHeader *sreqUdpHdr;
  TransportUdpHeader *srepUdpHdr;
  GlomoAppServiceBinding *binding;
  GlomoAppServiceBinding *servreq;
  isSrep = (rrepPkt->numExtensions != 0)
           && (rrepPkt->repExt[0].extType == SERVICE_REPLY) ;
  isSreg = (rrepPkt->numExtensions != 0)
           && (rrepPkt->repExt[0].extType == SERVICE_ADV);
  memmove(&lifetime, &rrepPkt->lifetime, sizeof(clocktype));
  RoutingAodvReplaceInsertRouteTable(node, srcAddr, -1, FALSE, TRUE, 1,
                                     srcAddr, simclock() + lifetime);
  if (srcAddr == rrepPkt->destAddr && RoutingAodvIfSeqValid(srcAddr,&aodv->routeTable))
  {
    causedNewRoute = TRUE;
  }
  rrepPkt->hopCount++;
  if (!RoutingAodvCheckRouteEntryExist(rrepPkt->destAddr, &aodv->routeTable))
  {//Forward route does not exist. so creating new entry
    if (rrepPkt->destAddr != node->nodeAddr)
    {
      RoutingAodvReplaceInsertRouteTable(node, rrepPkt->destAddr,
                                         rrepPkt->destSeq, TRUE, TRUE,
                                              rrepPkt->hopCount, srcAddr,
                                              simclock()+lifetime);
    causedNewRoute = TRUE;
  }
  else {return;}
}
else
{
  BOOL seqInvalid = FALSE, seqGreater=FALSE,
           routeInactive=FALSE, smallerHopCount=FALSE;
  seqInvalid = !RoutingAodvIfSeqValid(rrepPkt->destAddr,&aodv->routeTable);

  if (!seqInvalid)
  {
    int seq = RoutingAodvGetSeq(rrepPkt->destAddr, &aodv->routeTable);
    if (seq < rrepPkt->destSeq)
    {
      seqGreater = TRUE;
    }
    if ((seq == rrepPkt->destSeq)
        && (RoutingAodvIfRouteInactive(rrepPkt->destAddr,&aodv->routeTable)))
    {
      routeInactive = TRUE;
    }
    if ((seq == rrepPkt->destSeq)
        &&(rrepPkt->hopCount
             <RoutingAodvGetHopCount(rrepPkt->destAddr,&aodv->routeTable)))
    {
      smallerHopCount = TRUE;
    }
  }

  if (seqInvalid || seqGreater || routeInactive || smallerHopCount)
  {
    RoutingAodvReplaceInsertRouteTable(node, rrepPkt->destAddr, rrepPkt->destSeq,
                                              TRUE, TRUE, rrepPkt->hopCount,
                                              srcAddr, simclock()+lifetime);
    causedNewRoute = TRUE;
  }
}


if (rrepPkt->srcAddr == node->nodeAddr)
{
  if (isSrep)
  {
    aodv->stats.numSREPreceived++;
  }
  else if (isSreg)
  {
    aodv->stats.numSREGreceived++;
  }
  else
  {
    aodv->stats.numRREPreceived++;
  }

  if (causedNewRoute)
  {
    while (RoutingAodvCheckServer(rrepPkt->destAddr, aodv->serverRoute.head))
    {
      GlomoAppServiceDesc *desc = RoutingAodvGetServerRouteNode
                                                (rrepPkt->destAddr, aodv->serverRoute.head)
                                                 ->serviceDesc;
      RoutingAodvDeleteServer(rrepPkt->destAddr, desc->nameStr,&aodv->serverRoute);
    }
    RoutingAodvDeleteSent(rrepPkt->destAddr, &aodv->sent);
    while (RoutingAodvLookupBuffer(rrepPkt->destAddr, &aodv->buffer))
    {
      newMsg = RoutingAodvGetBufferedPacket(aodv, FALSE,
                                                        rrepPkt->destAddr, &aodv->buffer);
      RoutingAodvTransmitData(node, newMsg, rrepPkt->destAddr);
      aodv->stats.numDataSent++;
      RoutingAodvDeleteBuffer(rrepPkt->destAddr, &aodv->buffer);
    }
    while (RoutingAodvLookupBuffer(srcAddr, &aodv->buffer))
    {
      newMsg = RoutingAodvGetBufferedPacket(aodv, FALSE, srcAddr, &aodv->buffer);
      RoutingAodvTransmitData(node, newMsg, srcAddr);
      aodv->stats.numDataSent++;
      RoutingAodvDeleteBuffer(srcAddr, &aodv->buffer);
    }
  }//if (causedNewRotue)

  if (isSrep || isSreg)
  {
    int i;
    for (i = 0; i<rrepPkt->numExtensions; i++)
    {
      if (rrepPkt->repExt[i].extType == SERVICE_REPLY)
      {
        memset(serviceName, 0, sizeof(serviceName));
        memcpy(serviceName, rrepPkt->repExt[i].ServBinding[0].serviceName,
                    strlen(rrepPkt->repExt[i].ServBinding[0].serviceName)+1);
        if ((serviceRequest = RoutingAodvGetServiceRequest
                                  (serviceName, &aodv->serviceTable)) != NULL)
        {
          if ((rrepPkt->repExt[i].ServBinding[0].serverAddr != INVALID_ADDRESS))
          {
            Service_BT_Node *previous = NULL;
            Service_BT_Node *current = NULL;
            Service_BT_Node *head =NULL;
            int j;
            GlomoAppServiceBinding *bestBinding;

            for (j = 0; j < rrepPkt->repExt[i].numChoise; j++)
            {
              current = (Service_BT_Node *)checked_pc_malloc(sizeof(Service_BT_Node));
              current->binding = rrepPkt->repExt[i].ServBinding[j];
              if (previous == NULL)
              {
                head = current;
              }else
              {
                previous->next = current;
              }
              previous = current;
            }
            if (current != NULL)
            {
```

```
          current->next = NULL;
        }

        if (RoutingAodvExistSrep(serviceName, &aodv->srepRecv))
        {
          Service_BT_Node *theNode = (Service_BT_Node *)
                                     checked_pc_malloc(sizeof(Service_BT_Node));
          theNode->binding = * (RoutingAodvGetServiceBindingFromSrep
                                (serviceName, &aodv->srepRecv));
          theNode->next = head;
          head = theNode;
          bestBinding = RoutingAodvChooseBestBinding(node, head);
          RoutingAodvReplaceInsertSrepRecv(serviceName, bestBinding->serverAddr,
                                           *bestBinding, &aodv->srepRecv);
        }
        else
        {
          bestBinding = RoutingAodvChooseBestBinding(node, head);
          RoutingAodvReplaceInsertSrepRecv(serviceName, bestBinding->serverAddr,
                                           *bestBinding, &aodv->srepRecv);
          RoutingAodvSetTimerForSreq(node, MSG_NETWORK_SendSrepTimeout,
                                     bestBinding,
                                     (clocktype)SREP_SETTLE_TIME);
        }

        while (head!=NULL)
        {
          Service_BT_Node *toFree = head;
          head = head->next;
          pc_free(toFree);
        }
      }
      else
      {
        IpHeaderType *ipHeader = (IpHeaderType *)serviceRequest->packet;
        GlomoAppServiceDesc *servDesc = (GlomoAppServiceDesc *)
                                        (serviceRequest->packet
                                        + IpHeaderSize(ipHeader)
                                        + sizeof(TransportUdpHeader));
      }
    }
    break;
  }
  else if (rrepPkt->repExt[i].extType == SERVICE_ADV)
  {
    RoutingAodvHandleServiceReg(node, msg);
    break;
  }
 }
}
else
{
  node->allowToMove = TRUE;
}
GLOMO_MsgFree(node, msg);
}
else
{
  if (causedNewRoute)
  {
     RoutingAodvActivateRoute(rrepPkt->srcAddr, &aodv->routeTable);
     RoutingAodvSetTimer(node, MSG_NETWORK_CheckRouteTimeout,
                              rrepPkt->srcAddr, (clocktype)ACTIVE_ROUTE_TO);
     RoutingAodvRelayRREP(node, msg, srcAddr);
   }
   else if (isSrep || isSreg)
   {
     RoutingAodvActivateRoute(rrepPkt->srcAddr, &aodv->routeTable);
     RoutingAodvSetTimer(node, MSG_NETWORK_CheckRouteTimeout,
                              rrepPkt->srcAddr, (clocktype)ACTIVE_ROUTE_TO);
                              RoutingAodvRelayRREP(node, msg, srcAddr);
   }
   else
   {
     GLOMO_MsgFree(node, msg);
   }
  }
}
```

A.5 Hanlde SC annoucements

```
void RoutingAodvHandleSCAnn(GlomoNode *node, Message *msg, int ttl)
{
  GlomoNetworkIp *ipLaye;
  GlomoRoutingAodv *aodv;
  AODV_RREQ_Packet *rreqPkt;
  NODE_ADDR SC;
  clocktype bindingTime;

  ipLayer = (GlomoNetworkIp *)node->networkData.networkVar;
  aodv = (GlomoRoutingAodv *)ipLayer->routingProtocol;
  rreqPkt = (AODV_RREQ_Packet *)GLOMO_MsgReturnPacket(msg);
  SC = GLOMO_GetServiceCoordinator(&node->affiliatedServiceCoordinator);
  bindingTime = GLOMO_GetSCBindingTime(&node->affiliatedServiceCoordinator);

  if (SC == INVALID_ADDRESS)
  {
    GLOMO_UpdateServiceCoordinator(&node->affiliatedServiceCoordinator,
                                          rreqPkt->srcAddr, rreqPkt->hopCount,
                                          simclock() + ACTIVE_SC_TO);
    RoutingAodvSetTimer(node, MSG_NETWORK_CheckSCTimeout,
                             rreqPkt->srcAddr, ACTIVE_SC_TO);
  }
  else if (SC == node->nodeAddr)
  {
    GLOMO_ReplaceInsertServiceCoordinator(&node->heardServiceCoordinators,
                                               rreqPkt->srcAddr, rreqPkt->hopCount,
                                               simclock() + ACTIVE_SC_TO);
    RoutingAodvSetTimer(node, MSG_NETWORK_CheckSCTimeout,
                             rreqPkt->srcAddr, ACTIVE_SC_TO);

  }
  else if (SC == rreqPkt->srcAddr)
  {
    GLOMO_UpdateServiceCoordinator(&node->affiliatedServiceCoordinator,
                                          rreqPkt->srcAddr, rreqPkt->hopCount,
                                          simclock() + ACTIVE_SC_TO);
    RoutingAodvSetTimer(node, MSG_NETWORK_CheckSCTimeout,
                             rreqPkt->srcAddr, ACTIVE_SC_TO);
  }
```

```
else
{
 if (!RoutingAodvCheckRouteExist(SC, &aodv->routeTable))
 {
  if (RoutingAodvCheckRouteExist(rreqPkt->srcAddr,&aodv->routeTable))
  {
   GLOMO_DeleteServiceCoordinator(&node->heardServiceCoordinators,
                                  rreqPkt->srcAddr,FALSE);
   GLOMO_ReplaceInsertServiceCoordinator(&node->heardServiceCoordinators,
                                         SC, GLOMO_GetSCAnnDiameter
                                         (&node->affiliatedServiceCoordinator),
                                         bindingTime);
   GLOMO_UpdateServiceCoordinator(&node->affiliatedServiceCoordinator,
                                  rreqPkt->srcAddr,rreqPkt->hopCount,
                                  simclock() + ACTIVE_SC_TO);
   RoutingAodvSetTimer(node, MSG_NETWORK_CheckSCTimeout,
                       rreqPkt->srcAddr,ACTIVE_SC_TO);
  }
  else
  {
   GLOMO_ReplaceInsertServiceCoordinator(&node->heardServiceCoordinators,
                                         rreqPkt->srcAddr,rreqPkt->hopCount,
                                         ACTIVE_SC_TO + simclock());
   RoutingAodvSetTimer(node, MSG_NETWORK_CheckSCTimeout,
                       rreqPkt->srcAddr, ACTIVE_SC_TO);
  }
 }
 else
 {
  if (RoutingAodvCheckRouteExist(rreqPkt->srcAddr, &aodv->routeTable))
  {
    if (RoutingAodvGetHopCount(rreqPkt->srcAddr, &aodv->routeTable)
      < RoutingAodvGetHopCount(serviceCoordinator, &aodv->routeTable))
   {
    GLOMO_DeleteServiceCoordinator(&node->heardServiceCoordinators,
                                   rreqPkt->srcAddr,FALSE);
    GLOMO_ReplaceInsertServiceCoordinator(&node->heardServiceCoordinators,
                                          SC, GLOMO_GetSCAnnDiameter
                                          (&node->affiliatedServiceCoordinator),
                                          bindingTime);
    GLOMO_UpdateServiceCoordinator(&node->affiliatedServiceCoordinator,
                                   rreqPkt->srcAddr,rreqPkt->hopCount,
                                   ACTIVE_SC_TO + simclock());
    RoutingAodvSetTimer(node, MSG_NETWORK_CheckSCTimeout,
                        rreqPkt->srcAddr, ACTIVE_SC_TO);
   }
   else if  (RoutingAodvGetHopCount(rreqPkt->srcAddr, &aodv->routeTable)
             == RoutingAodvGetHopCount(SC, &aodv->routeTable)
             && RoutingAodvGetLifetime(serviceCoordinator, &aodv->routeTable)
                < RoutingAodvGetLifetime(rreqPkt->srcAddr, &aodv->routeTable))
   {
    GLOMO_DeleteServiceCoordinator(&node->heardServiceCoordinators,
                                   rreqPkt->srcAddr, FALSE);
    GLOMO_ReplaceInsertServiceCoordinator(&node->heardServiceCoordinators,
                                          SC, GLOMO_GetSCAnnDiameter
                                          (&node->affiliatedServiceCoordinator),
                                          bindingTime);
    GLOMO_UpdateServiceCoordinator(&node->affiliatedServiceCoordinator,
                                   rreqPkt->srcAddr,rreqPkt->hopCount,
                                   ACTIVE_SC_TO + simclock());
```

```
          RoutingAodvSetTimer(node, MSG_NETWORK_CheckSCTimeout,
                                   rreqPkt->srcAddr, ACTIVE_SC_TO);

        }
        else
        {
          GLOMO_ReplaceInsertServiceCoordinator(&node->heardServiceCoordinators,
                                                   rreqPkt->srcAddr, rreqPkt->hopCount,
                                                   ACTIVE_SC_TO + simclock());
          RoutingAodvSetTimer(node, MSG_NETWORK_CheckSCTimeout,
                                   rreqPkt->srcAddr, ACTIVE_SC_TO);
        }
      }
      else
      {
        GLOMO_ReplaceInsertServiceCoordinator(&node->heardServiceCoordinators,
                                                 rreqPkt->srcAddr, rreqPkt->hopCount,
                                                 ACTIVE_SC_TO + simclock());
        RoutingAodvSetTimer(node, MSG_NETWORK_CheckSCTimeout,
                                 rreqPkt->srcAddr, ACTIVE_SC_TO);
      }
    }
  }
}
```

A.6 Handle Service registrations

```
void RoutingAodvHandleServiceReg(GlomoNode *node, Message *msg)
{
  GlomoNetworkIp* ipLayer = (GlomoNetworkIp *) node->networkData.networkVar;
  GlomoRoutingAodv* aodv = (GlomoRoutingAodv *) ipLayer->routingProtocol;
  AODV_RREP_Packet* rrepPkt = (AODV_RREP_Packet*)GLOMO_MsgReturnPacket(msg);
  int i;
  GlomoAppServiceBinding *binding;
  NODE_ADDR serverAddr;
  short servicePort;
  clocktype bindingtime;

  assert(node->isServiceCoordinator);
  for(i = 0; i < rrepPkt->repExt[0].numChoise; i++)
  {
    serverAddr = rrepPkt->repExt[0].ServBinding[i].serverAddr;
    servicePort = rrepPkt->repExt[0].ServBinding[i].servicePort;
    bindingtime = min(rrepPkt->repExt[0].ServBinding[i].bindingtime,
                         simclock() + ACTIVE_SERVICE_BINDING_TO);
    GLOMO_ReplaceInsertService(&node->cachedService,
                                  rrepPkt->repExt[0].ServBinding[i].serviceName,
                                  serverAddr,servicePort, bindingtime);
    RoutingAodvSetTimer(node, MSG_NETWORK_CheckServiceTimeout,
                           serverAddr, bindingtime-simclock());
  }
}
```

A.7 Client, Server, Service Coordinator Selection

```
void GLOMO_AppInitApplications(GlomoNode *node, const GlomoNodeInput *nodeInput)
{
  GlomoNodeInput appInput;
```

```
char appStr[GLOMO_MAX_STRING_LENGTH];
BOOL retVal;
int  i;

retVal=GLOMO_ReadCachedFile(nodeInput,"APP-CONFIG-FILE", &appInput);
if (retVal == FALSE)
{
  fprintf(stderr, "Application: Needs APP-CONFIG-FILE.\n");
  assert(FALSE); abort();
}
node->appData.uniqueId = 0;
for (i = 0; i < appInput.numLines; i++)
{
  sscanf(appInput.inputStrings[i], "%s", appStr);
  if strcmp(appStr, "PRINT_CLIENT") == 0)
  {
    char type[GLOMO_MAX_STRING_LENGTH];
    char intervalStr[GLOMO_MAX_STRING_LENGTH];
    long itemSize;
    clocktype startTime,interval;
    int dd;
    double  density;
    double probability;

    memset(type, 0, GLOMO_MAX_STRING_LENGTH);
    retVal = sscanf(appInput.inputStrings[i],
                         "%s %d %s %ld",
                         appStr, &dd, intervalStr, &itemSize);
    if (retVal != 4)
    {
      fprintf(stderr,
              "Wrong PRINT_CLIENT configuration format!\n"
              "PRINT_CLIENT <density> <interval> <itemSize>");
      assert(0); abort();
    }
    density = (double) dd / 100.0;
    probability = pc_erand(node->seed);
    memset(type, 0, GLOMO_MAX_STRING_LENGTH);
    if (probability < density)
    {
      int t = ((int)(pc_erand(node->seed) * 10))%2;
      switch (t)
      {
        case 0: memcpy(type, "INK_BW", strlen("INK_BW") +1); break;
        case 1:memcpy(type, "INK_COLOR", strlen("INK_COLOR") +1); break;
      }
      startTime = (clocktype)(pc_erand(node->seed) * SECOND);
      interval = GLOMO_ConvertToClock(intervalStr);
      AppPrintClientInit(node, type, startTime, interval, itemSize);
    }
  }
  else if (strcmp(appStr, "PRINT_SERVER") == 0)
  {
    char type[GLOMO_MAX_STRING_LENGTH];
    double probability, density;
    int dd;
    char *serviceName = (char *)checked_pc_malloc(sizeof(char));
    APP_TYPE port;

    retVal = sscanf(appInput.inputStrings[i], "%s %d",appStr, &dd );
  if (retVal != 2)
  {
    fprintf(stderr,
            "Wrong PRINT_SERVER configuration format!\n"
            "PRINT_SERVER <density> ");
    assert(0); abort();
  }
  density = (double) dd / 100.0;
  probability = pc_erand(node->seed);
  if (probability < density)
  {
    int t = ((int)(pc_erand(node->seed) * 10))%2;
    switch (t)
    {
      case 0: memcpy(type, "INK_BW", strlen("INK_BW") +1);
              port = APP_PRINT_SERVER_INK_BW;
              node->partitionData->totalBW++;
              node->bwServer = TRUE;
              break;
      case 1: memcpy(type, "INK_COLOR", strlen("INK_COLOR") +1);
              port = APP_PRINT_SERVER_INK_COLOR;
              node->partitionData->totalCOLOR++;
              node->colorServer = TRUE;
              break;
    }
    AppPrintServerInit(node, type, port);
    strcpy(serviceName, "PRINT_SERVER_UDP_");
    strcat(serviceName, type);
    GLOMO_ReplaceInsertService(&node->myService,serviceName,
                               node->nodeAddr,port,
                               simclock()+SERVICE_BINDING_TIME);
  }
}
else if (strcmp(appStr, "SERVICE_COORDINATOR") == 0)
{
  double probability, density;
  int dd;
  int annDiameter;

  retVal = sscanf(appInput.inputStrings[i],
                  "%s %d %d",
                  appStr, &dd, &annDiameter);
  if (retVal != 3)
  {
    fprintf(stderr,
            "Wrong SERVICE_COORDINATOR configuration format!\n"
            "SERVICE_COORDINATOR <density> <annDiameter>");
    assert(0); abort();
  }

  density = (double)dd / 100.0;
  probability = pc_erand(node->seed);
  if (probability < density)
  {
    node->isServiceCoordinator = TRUE;
    GLOMO_UpdateServiceCoordinator(&node->affiliatedServiceCoordinator,
                                   node->nodeAddr, annDiameter,
                                   simclock() + SC_BINDING_TIME);
  }
}
```

```
    else
    {
      printf("Application: Unknown application %s\n", appStr);
      assert(0); abort();
    }
  }
}
```