ABSTRACT

RAO, RANJANA. Integration of On-Demand Service and Route Discovery in Mobile Ad Hoc Networks. (Under the direction of Dr. Douglas Reeves).

Integration of On-Demand Service and Route Discovery in Mobile Ad Hoc Networks. This thesis studies the idea of adding on-demand service discovery to on-demand routing protocols for mobile ad hoc wireless networks. When on-demand service discovery is used, the service discovery process is initiated only when a node requires access to a service. The address of the service provider is returned to the initiator of the service discovery process. However, the initiator still has to discover a route to the service provider. By integrating service and route discovery, both the identity of the node providing the service and the route to the service are discovered together. This approach differs from related work on service discovery in ad hoc networks which perform service discovery at the application layer and do not discover a route to the service. A combined process for service provider identification and discovery of a route to this provider saves additional message transmissions and reduces the delay incurred if the two processes were done separately at the application and network layers respectively.

The idea of adding on-demand service discovery to on-demand routing protocols for mobile ad hoc networks was earlier proposed in an Internet-Draft and simulated using Ad Hoc On-Demand Distance Vector Protocol (AODV) as the routing protocol. In this thesis, we propose modifications and extensions to the Internet-Draft. We discuss solutions to the problem of notifying the nodes in the network when a service provider terminates its service. We also present a modified simulation of the integration of service discovery with AODV (we discovered a number of bugs in the original simulation). We then present a simulation of the integration of service discovery with the Dynamic Source Routing Protocol (DSR).

Lastly, we compare the performance of service discovery with AODV and DSR. We find that integrating service discovery with AODV gives better results in terms of service route acquisition latency. In terms of path length and the ratio of the path length to the shortest path, DSR gives better results for a smaller network size, while AODV gives better results for a larger network size. Further, the number of service discovery failures is higher when DSR is used, especially in larger networks. However, due to the aggressive caching in

DSR, a node which requires a service is more likely to have a route to the service provider and hence may not need to initiate a service discovery. Since the DSR implementation in ns-2 that we consider does not actively purge stale routes from its cache, such a cached route is likely to be valid only in conditions of low mobility. Overall, we conclude that integration of service discovery with AODV gives better results than integration with DSR.

Integration of On-Demand Service and Route Discovery in Mobile Ad Hoc Networks

by

Ranjana Rao

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Computer Science

Raleigh

2004

Approved By:

Dr. Mihail Sichitiu	Dr. Khaled Harfoush

Dr. Douglas Reeves
Chair of Advisory Committee

To the memory of

my brother,

Sandeep Rao

Biography

Ranjana Rao grew up in Bangalore, India. She did her undergraduate degree in telecommunications engineering at R. V. College of Engineering, Bangalore. She then worked for two years as a software engineer at Cisco Systems, India. During this time, she got more interested in the field of computer networking and decided to come to North Carolina State University, Raleigh for a Masters in Computer Science.

Acknowledgements

I would like to thank my advisor and committee chair, Dr. Reeves for his valuable help and guidance. I would like to thank him for his patience and support throughout this process. I would also like to thank my committee members, Dr. Sichitiu and Dr. Harfoush for their support and helpful comments and feedback.

I express my gratitude to Sandeep Gupta who shared his code on which my work is based.

I am grateful for the support I have always received from my parents and the rest of my family. Lastly, I would like to thank all my friends who have always been there for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis discusses the design and performance of a protocol which integrates on-demand service and route discovery in mobile ad hoc wireless networks (MANETs).

There is an increasing demand among users for networks with minimal deployment and configuration time. A MANET is formed when a group of mobile nodes decide on-the-fly to start communicating with each other. Such a network enables communication without the need for infrastructure or any centralized control. Thus, MANETs could be the answer to present and future needs for on-demand networking.

Users of today's networks expect certain services from the network. Indeed one of the purposes of the network is to enable nodes to access remote services and resources not locally available. This sharing of functionality is even more relevant in MANETs where nodes act as peers and every node is responsible for ensuring the smooth functioning of the network as a whole. Certain nodes in a MANET may take on the task of providing a service to the other nodes. Given the significance and potential benefits of MANETs, it is necessary to design robust protocols which ensure that users of such networks are able to find the services they require. Several service discovery protocols have been proposed for this purpose. In this thesis, we take this idea one step further by studying the effects of combining the discovery of a service with the discovery of a route to the service.

In this chapter, we summarize the main features, applications and challenges of MANETs. We also discuss routing and service discovery protocols and the motivation for integrating them. For further reading, we recommend Toh's book [41] which discusses ad

hoc networks in detail and Perkin's book [35] which provides an in depth analysis of routing protocols for ad hoc networks.

1.1 Mobile Ad Hoc Networks

1.1.1 Features

Nodes in a MANET are not only mobile but may have limited capabilities and may be interested in participating in the network only for a short period of time. This behavior gives rise to a number of interesting features outlined in this section.

Nodes may voluntarily join or leave a MANET at any time. Also, nodes may lose battery power or fail or decide to shut down at any time. Each node has a wireless transmission range which is the region around it where its transmissions can be heard. A node or a group of nodes may move out of transmission range of other nodes at any time, thus partitioning the network. Similarly, node movement may cause two partitions to be reconnected, thus merging the partitions. Thus, the topology and membership of a MANET is dynamic is nature. Hence, any protocols for MANETs which make use of centralized nodes are prone to failure. In order for a protocol for MANETs to be robust in the face of continual change, it should be distributed in nature.

A node is assigned an address when it joins the MANET and may be reassigned an address when the network partitions or partitions merge. For more information on address assignment schemes for MANETs, refer to [3], [29], [44], [45], [36] and [26].

If two nodes are within the wireless transmission range of each other, they can communicate directly. Otherwise, intermediate nodes must forward packets between the two nodes. The routing mechanisms in MANETs differ from those in wired networks. In conventional hierarchical routing in wired networks, only a few nodes act as routers and addresses are aggregated when storing routing table entries. Given the highly dynamic nature of MANETs, it would not be suitable for a few nodes to function as routers. These router nodes could move or leave the network or shut down (voluntarily or involuntarily) at any time. Hence in MANETs, every node has knowledge of routes to other nodes in the network. Thus, each node acts as a router and participates in routing a packet to its destination.

1.1.2 Applications

As mentioned earlier, MANETs have tremendous potential for on-demand networking. They can be used in situations where there is no infrastructure or the infrastructure has been damaged. This may include communication between participants at a meeting or in a classroom. They can also be used by emergency rescue teams in places of natural disasters and for communication on the battlefield. Sensor networks which are a kind of ad hoc network are used in military, weather prediction, agriculture and food processing. These networks consist of a number of sensors which can detect changes in environmental or weather conditions or the presence of impurities or chemical agents. The sensors send their statistical data to a collection point, which analyzes the data.

1.1.3 Challenges

Since by definition a MANET is formed on an as needed basis, any node willing to run the required protocols could join such a network. Thus, the nodes in a MANET could be heterogeneous in nature. The nodes may have limited computation, memory, communication and battery power. Also, the bandwidth in wireless networks is typically at least an order of magnitude less than that in wired networks. In addition, the wireless medium is more susceptible to transmission errors. All these factors combined with the mobility of nodes poses a challenge for designing robust protocols for MANETs.

Due to bandwidth constraints, we would like to limit the number of control messages exchanged between nodes. Also, every message sent consumes battery power and requires computation capability at each node along its path. All the nodes in the network would have to incur this overhead in the case of a broadcast message. Since nodes have limited capabilities, it is important to minimize the control overhead in the network. We would like to use as much of the available resources for the transmission of data traffic.

However, since the topology and membership of the network is continually changing, nodes need to keep track of these changes in order to adapt quickly to them. For nodes to keep track of these changes as they happen, topology or status update messages would need to be sent periodically.

There is thus a tradeoff between the need to minimize the amount of communication and processing overhead and the need to have fresh knowledge of the status of a

dynamic network. We need to keep this is mind when designing a protocol for an ad hoc network.

1.2 Routing Protocols for Mobile Ad Hoc Networks

In wired networks, topology updates are sent periodically between routers which then update their routing tables. This type of routing protocols are called proactive or table driven. In MANETs, bandwidth needs to be conserved. Hence, periodic update of topology information may result in too much overhead. For this reason, reactive or on-demand routing protocols were proposed. In this case, only when a node needs a route to a destination, it tries to acquire one.

The benefit of proactive protocols is that the route acquisition latency is low. This is because there is already a route for the destination in the routing table. On the other hand, if an on-demand routing protocol is used, a route discovery process has to be completed before the source gets a route to the destination. Thus, there is a tradeoff between the route acquisition latency and the bandwidth consumed.

The link state and distance vector routing protocols used in wired networks have been adapted for MANETs. Also, new approaches for routing in MANETs have been proposed.

An example of a proactive distance vector routing protocol for MANETs is Destination-Sequenced Distance Vector Protocol (DSDV) [33]. In this protocol, destination sequence numbers are used to solve the count to infinity problem. An example of a reactive protocol is Dynamic Source Routing Protocol (DSR) [20]. This uses sources routes instead of next hop routing and does on-demand route discovery. Another example of a reactive protocol is Ad Hoc On-Demand Distance Vector Protocol (AODV) [37]. This uses destination sequence numbers as in DSDV but does on-demand route discovery as in DSR.

A protocol which combines proactive and reactive mechanisms to form a hybrid protocol is Zone Routing Protocol (ZRP) [18]. Here the network is divided into a number of zones. Within a zone, proactive routing is used, while outside the zone, reactive routing is used. A protocol which makes use of link reversal is Temporally Ordered Routing Algorithm (TORA) [32]. An example of a link state protocol which uses bandwidth efficiently is Source Tree Adaptive Routing (STAR) [11].

1.3 Service Discovery for Mobile Ad Hoc Networks

1.3.1 Motivation for Service Discovery Protocols

Nodes in an ad hoc network need access to servers such as web servers, mail servers, name servers, print servers and tftp servers. We cannot statically configure nodes with a list of servers. This is because the node which needs the service (client) and the node providing the service (provider) may move, fail, shut down or run out of battery power. Node movement may even cause a node to get partitioned from the rest of the network. Also, a provider may decide to terminate its service at any time.

One approach for service discovery in wired networks is for a set of nodes (directory agents) to store information about providers. Clients then query the directory agents to discover providers. A static and hierarchical structure of directory agents can be maintained as in Domain Name Service (DNS). However, such a structure would not be suitable for an ad hoc network which is dynamic in nature. Service discovery protocols are needed which take into account the dynamic topology of an ad hoc network. A fully distributed protocol would be more robust for such a network.

Some service discovery protocols for wired networks are Service Location Protocol (SLP) [17], Jini [27], Salutation [6] and Simple Service Discovery Protocol (SSDP) [12]. SLP is not fully distributed due to the use of directory agents which aggregate service information. Jini uses Java and provides a Jini lookup service which clients use to discover services. The network delay is assumed to be low, which would not be the case in a wireless network. Also, Jini is bulky and would not be suitable for nodes in an hoc network which have limited memory and computation power. Salutation makes use of an entity called Salutation Manager (SLM) in each node. The SLMs communicate using Remote Procedure Calls (RPCs). This would not work in an ad hoc network. SSDP is used in Universal Plug and Play (UPnP) [8]. This involves clients and servers communicating through a multicast group. However, maintenance of a multicast group involves a lot of overhead in ad hoc networks. The Windows operating system uses UPnP with SSDP.

The Bluetooth Service Discovery Protocol (SDP) [39] is suitable for discovering a service within a wireless range of one hop. MIT's Intentional Naming System (INS) [2] is designed for dynamic networks. It makes use of an overlay network of self configuring resolver nodes which can answer service queries. However, the list of resolver nodes has to

be maintained centrally and there should be an underlying routing protocol with periodic exchange of topology updates.

1.3.2 Motivation for Combining Service and Route Discovery

Service discovery protocols for MANETs are studied in detail in the next chapter. In all these protocols, service discovery results in a client discovering the address of the node which provides the desired service. In some protocols, more detailed information about the service may also be obtained by the client, such as service attributes and an access mechanism. However, a route to the provider has to be obtained separately as part of a route discovery process. This means that for a client to get a mapping from a service type to the address of a provider and then to obtain a route to the provider, two separate processes are required. In this thesis, we examine the effects of combining these two processes, so as to obtain the IP address of a provider and a route to it at the same time.

As mentioned earlier, in an ad hoc network, each message consumes significant network bandwidth as well as computation and battery power at each node along its path. We will see in the next chapter that service discovery itself requires the exchange of a large number of messages. The subsequent discovery of a route to the service by the underlying routing protocol would require another exchange of messages. However, by integrating service and route discovery we are able to discover a service provider and a route to it using the same set of messages. Hence, an integrated approach will significantly reduce the bandwidth consumed and the overall latency in discovering a service and a route to the service.

The drawbacks of strict layering in the design of network protocols have been pointed out by Cooper in [7] and a soft layering approach has been proposed where information is exchanged between the layers. Also, in [13], Goldsmith and Wicker point out that energy constrained ad hoc networks can benefit greatly from cross-layering. This is because due to energy constraints in such networks, the control message transmission in the network and the overhead of processing these messages at each node must be minimized. Thus, we see that in an ad hoc network, communication between the layers of the network protocol stack or integration of the functionality of two or more layers is sometimes implemented, if doing this makes it possible to significantly improve the performance of such a network.

Traditionally, service discovery is considered an application layer function. However, we examine doing service discovery at the network layer in order to reduce the communication and processing overhead. Such a deviation from the strictly defined functionality of each layer is reasonable in the case of an ad hoc network.

In the initial stages of this thesis work, we realized that adding on-demand service discovery to on-demand routing protocols for ad hoc networks would be a good idea. Subsequently, while looking at the ns-users mailing list, we found that the integration of service discovery with on-demand routing protocols had been proposed by Koodli and Perkins in Internet-Draft [21]. A simulation of this technique using AODV as the routing protocol had been done by Sandeep Gupta. In this thesis, we propose modifications and enhancements to the Internet-Draft specification. We also note that the integrated approach given in the draft does not take care of purging of stale information about a terminated service. We propose solutions to this problem and provide details of the solution we have chosen. We identify a number of bugs in the simulation of the integrated approach using AODV. We modify this simulation and also simulate the integrated approach using DSR as the routing protocol. We then compare the performance of the integrated approach using the two different routing protocols.

Note that ideally, we would have liked to compare the integrated approach with an existing service discovery protocol at the application layer. We contacted the authors of a couple of such protocols which had been simulated in ns-2. However, we did not get a response to our request for using their simulation code for comparison purposes. Hence, we decided instead to compare the integrated approach using the two different routing protocols.

1.4 Thesis Chapter Outline

The rest of the thesis is organized as follows. In chapter 2, we discuss existing service discovery protocols for MANETs. Chapter 3 outlines the features of the two on-demand routing protocols that we consider in this thesis, namely DSR and AODV. Chapter 4 discusses the Internet-Draft for service discovery integration with on-demand routing protocols and an earlier simulation of this approach using AODV. This chapter also presents a modified version of the simulation using AODV and a simulation using DSR. Modifications

and additions to the draft are presented, including a solution for the service termination problem. The simulation results of the integrated approach using the two routing protocols are discussed in chapter 5. Finally, our conclusions and future work are presented in chapter 6. Note that in order to better understand the simulation details of the integrated approach, the ns-2 simulation details of both the routing protocols used, namely DSR and AODV are presented in Appendix A.

Chapter 2

Service Discovery Protocols for MANETs

In this chapter, we discuss a classification for service discovery techniques for MANETs and then use this to classify some of the existing service discovery protocols for MANETs. We also present in detail the working of these protocols and an analysis of their drawbacks.

2.1 Classification of Service Discovery Techniques

In [38], Preuß classifies service discovery techniques for all networks whether static or dynamic. Since we are only concerned with ad hoc networks, only that part of the classification relevant to such networks is presented here.

As shown in Figure 2.1, there are three types of service discovery for ad hoc networks - mediated, immediate and hybrid. In the mediated type of service discovery, directory or broker nodes store information about service providers. The brokers periodically send advertisements and the providers periodically register with the brokers. A client receives information about the service from a broker instead of from the service provider. The scope (number of hops) and frequency of a broker advertisement and frequency of provider

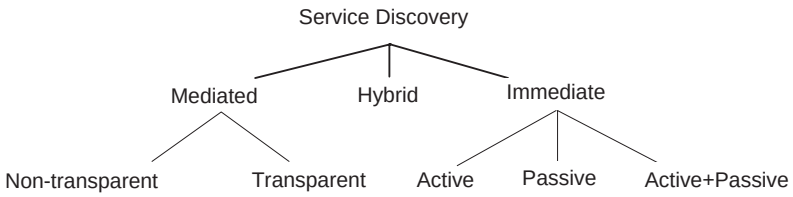

Figure 2.1: Classification of Service Discovery in Ad Hoc Networks

registrations have to be carefully selected. The mediated type of discovery is scalable to large networks but not good for high mobility scenarios.

There are two types of mediated discovery - non-transparent and transparent. Non-transparent means that the client knows that it is communicating with a broker. Transparent means that the client thinks it is communicating with a service provider but actually it is communicating with a broker.

In immediate service discovery, the clients and service providers interact directly. This technique is suitable for highly mobile networks. Since it requires broadcasts, it may not scale well for large networks. There are three types of immediate discovery - active, passive and active+passive. In active discovery, the client sends a request and service providers reply. In passive discovery, the service providers periodically announce their service and clients do not send any requests. It is also possible for a service discovery protocol to use both active and passive discovery and hence we have the classification "active+passive".

It is possible for a service discovery protocol to use both the mediated and immediate techniques. Such a hybrid protocol would scale well and be robust in the face of high mobility. In such a protocol, brokers sends advertisements to their neighbors and providers which receive these advertisements register with the brokers. When a client needs a service, if it knows of a broker, it sends a query to the broker. Otherwise the client sends a broadcast query which may be replied to by providers of the service and brokers which know of the service. Service availability is defined as the number of client queries which are successfully answered. Simulation results presented by Guichal in [15] show that service availability in the hybrid approach is better and overhead is less than in the pure mediated or immediate

Service Discovery Protocol	*Mediated*		*Immediate*	
	Non-transparent	*Transparent*	*Active*	*Passive*
SLP	+		+	
Integration with ODMRP			+	+
Nom			+	
GSD			+	+
QoS-Aware	+			
Mediator Based	+			+
Service Awareness			+	+
JESA		+	+	+
Integration with DSR/AODV			+	

Table 2.1: Summary of Service Discovery Protocols for Ad Hoc Networks

approaches.

The classification of service discovery techniques like SLP, Jini, Salutation, SSDP, etc. based on the above types is presented in [38].

2.2 Service Discovery Protocols

Table 2.1 classifies some of the existing service discovery protocols for ad hoc networks based on the classification presented in the previous section. In the following sections, we look at the working of each of these protocols and point out some of their drawbacks.

2.3 Service Location Protocol

Service Location Protocol (SLP) for mobile users is described by Perkins in [34]. There are three SLP entities : User Agent (UA), Service Agent (SA) and Directory Agent (DA). UAs reside in applications requiring service, SAs reside in nodes providing service and DAs reside in nodes which have a directory listing of available services. There are a number of service types defined and each type has a number of attributes.

UAs and SAs learn of DAs through DHCP or through DA advertisements or through request messages. SAs send service registration messages to DAs with the service

type and attributes of the service they provide. To find a service, a UA can unicast a service request to a DA. The DA sends a service reply with the list of services which match the service type and attribute values. Alternatively, the UA can multicast a request to a well known multicast address at which all DAs and SAs listen. An SA which provides the required service will then reply. A service reply contains the URL for accessing the service.

2.3.1 Analysis

This protocol is for mobile users but not specifically for ad hoc networks. In ad hoc networks a fully distributed approach is desired and hence the use of DAs would not be suitable. DHCP is mentioned as a method for UAs and SAs to find DAs. However, in an ad hoc network, DHCP servers are usually not available for address configuration. The other alternative is periodic DA advertisements which would consume a lot of bandwidth. Also, if service request messages are multicast, depending on the multicast routing protocol used there could be significant overhead in maintaining the multicast tree in an ad hoc network.

2.4 Combining service discovery with ODMRP

A mechanism for combining service advertisement and discovery with On-Demand Multicast Routing Protocol (ODMRP) is presented by Cheng in [5]. The first method is the push model in which nodes providing a service and the possible users of this service form a multicast group. Service advertisements are sent in multicast group join query messages. These advertisements include the service name, IP address, port number and the multicast group address. Those nodes which may need the service, store the advertisement and send a service awareness reply in a join reply message. Services may send advertisements whenever there is a change in the service.

The second method is the pull model in which all nodes willing to reply to query messages form a multicast group. When a node needs a service, it sends a service query message in a join query. A reply message in the form of a service advertisement is sent by nodes in the multicast group.

2.4.1 Analysis

In this paper, service discovery is combined with a routing protocol as in the protocol studied in this thesis. However, the method used is different since this paper uses a multicast protocol to find the unicast address of the service. This means that a separate route discovery may then be needed to find a route to the service. This is unlike the protocol studied in this thesis in which the address and a route to the service are obtained using a single request and reply. Moreover, sending messages to a multicast group requires significant overhead. No simulation/implementation results were presented in this paper.

2.5 Nom Resource Discovery Protocol

Nom [10] is a fully distributed resource discovery protocol proposed by Doval and O'Mahony. It is designed to provide name resolution i.e. mapping between a string and a node id within an ad hoc network. Nom operates above the network layer and below the application layer.

When a query-initiate message is received from an application, the node creates a query message with a certain ttl value and broadcasts it to its neighbors. When a query message is received from the network, if the node can resolve the string in the query, it broadcasts a query-reply, otherwise it forwards the query to its neighbors. If a node receives a query-reply for which it did not send a request, it forwards the query, otherwise it sends the reply to the application. Duplicate messages are identified using a message id.

2.5.1 Analysis

This protocol operation is similar to the on-demand route discovery process of sending a route request and reply. However, the Nom algorithm only returns the identity of the node providing the resource. A route to the node has to then be obtained using the underlying routing protocol. Also, the query-reply is broadcast thus consuming a lot of bandwidth. In order to unicast the query-reply, the underlying routing protocol will have to be used which may anyway trigger a broadcast route request.

2.6 Group-based Service Discovery Protocol

Group-based Service Discovery Protocol (GSD) is presented in [4] by Chakraborty, et al. DARPA Agent Markup Language (DAML) + Ontology Interchange Language (OIL) is used to describe a service. Services belong to a number of groups which are based on the hierarchy of classes and subclasses in DAML. The highest level group is "Service".

If syntactic matching is used, services are matched based on interface, identifier or attribute. However in an ad hoc network, services may implement several different interfaces. So a specified interface in a request may not match the interface implemented in a provider. Semantic matching as provided by DAML is desired.

Providers send service advertisements over a limited number of hops. These advertisements contain descriptions of the services provided by the node and the service groups that the node has received advertisements about. Nodes store this information in a service cache. The frequency of these advertisements can be adjusted depending on the mobility of the provider.

When a node needs a service and there is no entry for it in its cache, it sends a service request over a limited number of hops. As the request propagates a reverse route is set up and stored in a reverse route table. If a node has a matching service description in its cache it sends a service reply which follows the path in the reverse route table. Otherwise, if a node has an entry for the requested service group in its cache, it forwards the request to the node which provided the cache entry. Thus requests are forwarded selectively.

2.6.1 Analysis

Periodic sending of service advertisements consumes a lot of bandwidth even if the broadcast is limited. Also, it is mentioned that service requests are selectively forwarded to nodes which have seen a service group. However, it is not mentioned how a route to such nodes is obtained. Often a route discovery will trigger a broadcast by the underlying routing protocol. Also, a reverse route is stored but no forward route is stored. Hence, a route to the discovered service will still have to be found after completion of the discovery process.

2.7 QoS-Aware Resource Discovery

In [25], Liu, et al. present a protocol for discovering a resource in an ad hoc network and selecting the best possible resource provider based on some QoS metric. A client is a node which needs a resource and a resource provider (RP) is a node which provides a resource. A discovery agent (DA) is a node which stores information about RPs.

The network is divided into domains which are formed dynamically. The DAs periodically broadcast their address to their neighboring nodes. Every non DA node chooses the closest DA as its home DA. All RPs register with their home DA and send QoS information to it periodically. The home DA uses a hash function to decide the set of DAs to send the RP information to. When a client needs a resource it sends a directory query to its home DA. If the home DA can't reply, it forwards the query to the nearest DA from among the set of DAs storing information about this resource. If this DA doesn't have the required information, it forwards the query to the next nearest DA and so on until the required RP information is found. The DA with the list of RPs then sends a QoS query to all other DAs. Those DAs which are home DAs for the listed RPs then reply to the client's home DA with the RP addresses and corresponding QoS information. The information about the RP with the best QoS metric is then returned to the client.

2.7.1 Analysis

The use of DAs results in an architecture that is not fully distributed. The DA selection process and periodic DA announcements require broadcast messages. This incurs a lot of overhead. The periodic sending of QoS information from each RP to its home DA also results in a lot of bandwidth utilization. Also, in the protocol studied in this thesis, a client may receive multiple replies for different providers of a resource and can then choose the best provider. This mechanism is automatically built into the protocol unlike the QoS-Aware discovery which requires additional QoS queries to be sent after the list of RPs is obtained. Also, only the address of a RP is obtained through this protocol and not a route.

2.8 Mediator Based Service Location Protocol

In [22], a mediator based service location protocol is presented by Koubaa and Fleury. Mediators store information about providers. Each provider forms a cluster consisting of all providers in its vicinity. All nodes in a cluster share the medium. The network is divided into a number of clusters. Certain providers are elected to be mediators such that all providers are covered by one or more mediators. Regions which do not have any providers, do not need to have any mediators.

Providers periodically broadcast their id and their service to their cluster and mediators periodically broadcast their id and the provider ids they cover. Nodes cache this information. If a provider is not covered by a mediator or has only a unidirectional link to one, it becomes a mediator. Each mediator maintains a list of providers it covers and each provider maintains a list of providers that are connected to it and a list of mediators that cover it.

If a node needs a service and there is no entry in its cache and the mediators in its zone don't know of a provider for this service, the request needs to be forwarded. The request can be proactively processed by maintaining a multicast tree of mediators. The request can be broadcast in the tree with a designated mediator replying or the mediators can exchange their provider information and the client can anycast the request to a mediator. Alternatively, the request can be processed reactively by forming a multicast tree of mediators on-demand i.e. when a request is to be forwarded. The request is broadcast in the tree and a number of mediators may reply or replies can be aggregated.

2.8.1 Analysis

This protocol is quite complex. The term "zone" was used without giving a definition of it. The protocol involves periodic limited broadcasts from providers and mediators. This could consume significant bandwidth. Also, if a client doesn't have routes to the mediators in its zone, it may need to find routes or do a limited broadcast. As mentioned in the paper, the proactive method of request handling involving multicast tree maintenance requires a lot of message overhead. This protocol also does not provide a route to the provider.

2.9 Service Awareness

The concept of service awareness is presented by Wu and Zitterbart in [47]. Nodes have an awareness cache with service types, provider ids, provider status and the status update time. Providers send advertisements whenever they start or terminate providing a service. Nodes store the list of received ⟨provider id, advertisement sequence⟩ pairs in an advertisement table. This is used to detect the reception of a duplicate advertisement. If a node needs a service for which it doesn't have an entry in its cache, a service discovery message is broadcast. A node which has an entry for the service, sends a reply.

There are a number of ways of updating the awareness cache. When a node receives or overhears a data packet from a provider, it refreshes the corresponding entry in its cache. Also, when a routing packet indicates that a provider can still be reached or can no longer be reached, the cache is updated. Alternatively, a service poll request can be included as an IP option in data packets. The number of such poll requests can be limited. The destination node then sends a service awareness packet with a service poll reply option and its list of providers. Intermediate nodes can update their cache based on the information in the reply and can add additional providers that they know of to the reply. Alternatively, a provider can send a message indicating that it no longer provides service if a data packet for its service is sent to it after it has terminated its service.

In [46], the authors consider this method as a form of anycast. The service type could be thought of as an anycast group. All providers of a particular service type belong to that anycast group. The awareness cache in each node is used for anycast resolution i.e. to map the anycast group name to the IP address of a provider.

2.9.1 Analysis

In an ad hoc network, a node may not know ahead of time that it is about to terminate a service. Also, the node may lose battery power or fail or move away. So the assumption that a node can send an advertisement before it terminates a service is often not correct. Also, merely receiving a data packet from a provider or a routing update indicating that the provider is reachable is not sufficient to conclude that the provider still provides the service. It is possible that the provider no longer provides the service but is still sending data packets not associated with the service or the routing update indicates

that the provider is reachable. Also, through this protocol only the id of the provider can be determined and not a route to it.

2.10 JESA Service Discovery Protocol

Java Enhanced Service Architecture (JESA) is a Java based service platform. It includes a service discovery protocol called JESA Service Discovery Protocol (JSDP) which is presented in [38] by Preuß. In JSDP, within a local network, immediate discovery can be used and beyond the network segment, transparent discovery can be used. Immediate discovery could be active or passive. Transparent discovery involves a hierarchy of brokers with service providers registering with these brokers. JSDP consists of two protocols, Provider Location Protocol (PLP) and Proxy/Attribute Request Protocol (PARP).

PLP is used to find the location of a service provider based on a service type. The id of a service provider or broker which can provide more information about the service is returned. This id is then used by PARP to obtain attributes of a service or to obtain a service access method. This information is returned in serialized Java objects.

2.10.1 Analysis

PLP can be used in a non Java environment. PARP on the other hand requires Java. Overall, JSDP can be used only on mobile devices which have the Java run time environment. This may not be the case for all nodes in an ad hoc network. Also, two steps are required for obtaining a service access method. This provides flexibility for those nodes which only want to locate but don't want to access the service. However, it involves an unnecessary extra message exchange for those nodes which do want complete service information including an access mechanism.

2.11 Integrated Approach

In Internet-Draft [21], an approach for integrating on-demand service and route discovery is presented. We have chosen this approach for our study due to the significant reduction in overhead that can be gained by using this approach. For a more detailed

discussion of the motivation for such an integrated approach, please refer to Section 1.3.2. We discuss this protocol along with the details of our work in Chapter 4.

2.12 Comparison with the Integrated Approach

Having presented the details of a number of service discovery protocols, we now take a look at how they compare with the integrated approach we study in this thesis. As mentioned earlier, all the protocols discussed perform service discovery at the application layer and discover only the identity of the provider and not a route to the provider. Also, we observe that SLP, QoS-aware resource discovery, Mediator based service location and JESA, all make use of directory agents and hence are very different from the integrated approach. GSD uses selective forwarding and semantic matching which differs from the integrated approach we study. The integration of service discovery with ODMRP uses the same concept of integration with a routing protocol but not for the purpose of discovering a route to the service provider. The integrated approach which is the focus of this thesis has some similarities to the Nom resource discovery protocol and to the active discovery in the Service awareness protocol.

This concludes our discussion of service discovery protocols for MANETs. In the next chapter, we discuss on-demand routing protocols for MANETs.

Chapter 3

Routing Protocols for MANETs

In this thesis, we study the integration of service discovery with two on-demand routing protocols, namely Dynamic Source Routing Protocol (DSR) and Ad Hoc On-Demand Distance Vector Protocol (AODV). This chapter presents the basic features of these two routing protocols. For a detailed discussion of the simulation details of these two protocols in the network simulator ns-2, please refer to Appendix A.

3.1 Dynamic Source Routing Protocol

Dynamic Source Routing Protocol (DSR) was proposed by Johnson and Maltz in [20]. It is a reactive routing protocol which uses source routing. This means that the source of a packet inserts into the packet header the entire route to be followed by the packet to its destination. When a node receives such a packet, if it is on the source route to the destination, it forwards the packet to the next node on the route.

3.1.1 Route Request

Nodes learn of source routes and store them in their route cache. Each entry in the cache has an expiry time. When a source node needs a route to a destination and it doesn't have one in its route cache, it broadcasts a route request. This request contains the initiator address, target address, request id and a route record. The initiator is the

node originating the request while the target is the node for which a route is desired. The request id is a sequence number associated with a node and is incremented each time the node sends a request. A request is uniquely identified by the pair ⟨initiator address, request id⟩. The route record stores the path taken by the request as it traverses the network.

If a node receives a request in which the route record already has the node's address, a loop has been traversed and hence the request is dropped. If the node has previously processed a request with the same ⟨initiator address, request id⟩ pair, this is a duplicate request and hence it is dropped. Otherwise, if the node is not the target, it adds its address to the route record and rebroadcasts the request.

3.1.2 Route Reply

If the node which receives the route request is the target, it sends a route reply to the initiator of the request. This reply must contain the route to the target which is the route record from the request. In order to send the reply, the target must have a route to the initiator. This route can be obtained in several ways. The target's route cache may have a route to the initiator or the route record from the request can be reversed and used. Alternatively, a route request for the initiator can be originated by the target and the reply can be piggybacked on this new request.

3.1.3 Route Error

Link layer acknowledgements can be used to detect when a packet has been lost and attempt retransmission. If the packet cannot be successfully retransmitted, a route error packet is sent to the original source of the packet. This route error contains the addresses of the nodes at both ends of the failed link.

For a node to send a route error to the source, it must have a route to the source. The node can look for a route to the source in its route cache or can use the reversed route record from the packet which encountered the error. Alternatively, the route error can be piggybacked on a route request targeted at the source. Another method is to buffer the route error packet and send a route request for the source. When a reply is obtained, the buffered route error packet can be sent using the route from the reply.

When the source node receives the route error, it checks its route cache for routes

containing the failed link and truncates all such routes at the node before the failure. It may then retransmit a route request for the same target. However, the target may not be reachable due to partitioning of the network. To prevent excessive retransmissions of route requests, an exponential backoff is used while retransmitting route requests.

If link layer acknowledgements are not available, the nodes can operate their wireless network interface cards in promiscuous mode. Now when a node sends a packet and its neighbor receives and forwards the packet to its neighbors, the node will also receive the forwarded packet and this serves as an acknowledgement. This is known as passive acknowledgement.

Another alternative is to use a bit in the packet header to indicate to the next hop to send an acknowledgement.

In cases where links are not symmetric, link-layer acknowledgements are not possible. Hence, end-to-end acknowledgements at the transport or application layer can be used. However in this case, the route error message will not contain the hop in error and only the last hop in the route can be assumed to be in error.

3.1.4 Optimizations

1. **Replying from Route Cache**

   The route cache does not have to store separate entries for each route. Instead all the links in the known routes can be stored as a tree. Hence, if a shorter path is found between two nodes, all routes which pass through these two nodes, will automatically have a shorter path.

   When a node forwards a reply or data packet it can add the route from the packet to its route cache. When a node forwards a route error it can check its cache for routes containing the link in error. All such routes are then truncated before the failed link.

   When a node receives a route request and it is not the target but it has a route to the target in its route cache, it can send a reply and drop the route request. The route to be sent in the reply consists of the route record from the request, with the route from the cache appended to it. If the route thus obtained has a loop in it, the reply is not sent.

   Before replying from its cache, the node waits for an amount of time which is propor-

tional to the number of hops in the route to be sent in the reply. Hence, nodes with longer routes will wait for a longer duration. During this time, the node operates in promiscuous mode. If the node receives a packet for the target and the source route in this packet is shorter than the route to be sent in the reply, then the node doesn't send the reply. Otherwise, the node sends the reply at the end of the delay period. This delaying of the reply is done to ensure that a number of nodes do not reply from their cache at the same time causing congestion and to reduce the number of replies sent with routes which are not the shortest.

To reduce the number of messages in the network, packets can be piggybacked on a route request. Since a request is broadcast, the size of the piggybacked packet should be small. A route reply, route error or small data packet can be piggybacked on a request. In this case, if a node replies from its cache, it must make sure to extract the piggybacked packet and forward it to the target of the request. The source address and route in the forwarded packet should be set so that it appears as if the packet was originated by the original initiator and forwarded normally.

In order to avoid the network wide broadcast of the request if the target is one hop away or the node's neighbors know of a route to the target, the route request is first sent with a ttl of 1. Such a request is said to be non propagating. The timeout set for such a request can be small. If the request times out, another request is sent with ttl set to the maximum possible path length in the network. Alternatively, an expanding ring search can be used where the ttl in successive retransmissions of route requests is gradually increased.

2. **Promiscuous Mode**

   A node may operate its wireless network interface in promiscuous mode. This ensures that the interface will not filter packets based on the destination address, but will pass all received packets up to the network driver software.

   A node can now overhear reply and data packets and add routes from these packets to its cache. Also, if a node overhears a route error, it can update its cache so that all routes with the failed link are truncated at the node before the failure.

   If two nodes (sender and receiver) move closer to each other, they may be able to communicate with each other directly. The source routes containing these two nodes

may not have been updated to reflect this. In such a case, even though the sender does not forward a packet directly to the receiver, if the receiver is operating in promiscuous mode, it will overhear the packet. The receiver will be on the source route in the packet but there will be one or more intermediate nodes listed in the route between the sender and the receiver. The receiver node can then send an unsolicited route reply to the original source of the packet, informing it that the route can be shortened.

3. **Route Error Processing**

   The path taken by the route error to the source of the packet which encountered the error, may be different from the path originally followed by the packet which encountered the error. In this case, intermediate nodes along the failed path will not be notified of the error. When a source node receives such a route error, it sends a route error packet along the original path of the packet which encountered the error.

   If the network interface is not operated in promiscuous mode, nodes close to the path in error may not get to know of the error. These nodes may still regard the route as valid and respond to a new request with the failed path. To prevent this, negative information can be stored in the cache when a route error is received. Instead of just truncating the cache entries with the failed link, the failed link information itself can be stored in the cache. Now when the node receives a route which contains a failed link, it knows that this route is not valid. The failed link information should expire after a short period.

3.1.5 Flow State

In Internet-Draft [19], Hu, et al. propose adding flow state to DSR so as to eliminate the need for source routes to be included in every data packet. Each route between a source and a destination is assigned a unique flow id and the first data packet along a route sets up an association between the flow id and the next hop at each node along the route. Subsequent data packets along the same route include only the flow id and not the source route.

3.2 Ad Hoc On-Demand Distance Vector Protocol

Ad Hoc On-Demand Distance Vector Protocol (AODV) was proposed in [37] by Perkins and Royer. It is a reactive routing protocol that uses next hop routing. This means that unlike DSR where each packet has to contain the source route, in AODV, each packet has only the next hop and each node knows of only the next hop to reach a destination. Each node has an associated sequence number and broadcast id. The use of sequence numbers ensures that loop free routes are obtained. AODV finds routes consisting of symmetric links. The protocols details are given in the following sections.

3.2.1 Route Request

When a source node needs a route to a destination and it doesn't have one in its routing table, it broadcasts a route request with the following fields - ⟨source address, source sequence number, broadcast id, destination address, destination sequence number, hop count⟩. Each route request is uniquely identified by the pair ⟨source address, broadcast id⟩. When a node sends a new route request, it increments the broadcast id. A node may receive the same route request multiple times through different paths. A node identifies such duplicate requests from the ⟨source address, broadcast id⟩ pair and drops all such duplicates. The destination sequence number in the request is used to indicate the required freshness of the desired route to the destination.

When a node receives a route request it sets up an entry in its route table for the source node with the next hop set to the node from which it received the request. This is the reverse path route entry. If the node that received the route request is the destination, it sends a route reply. An intermediate node sends a route reply only if the destination sequence number in its route table entry is same or greater than the destination sequence number in the request. Otherwise, the intermediate node increments the hop count and rebroadcasts the route request.

3.2.2 Route Reply

The route reply is unicast to the request source and it has the following fields - ⟨source address, destination address, destination sequence number, hop count, lifetime⟩. When a node receives a reply, it sets up an entry in its route table for the destination node

with next hop set to the node from which it received the reply. This is the forward path route entry.

If the node which receives the route reply is not the request source, it increments the hop count and forwards the reply using the reverse path entry. It also refreshes the reverse path entry's lifetime. The reverse path entry along all paths other than the path of the route reply will timeout.

If an intermediate node receives another route reply for the same destination, it will update its routing table and forward the reply only if the destination sequence number is greater than that in the previous reply or if the destination sequence numbers are the same but the number of hops in the reply is smaller.

Each entry in the route table has the following fields - ⟨destination address, destination sequence number, next hop, number of hops, expiry time, active neighbors⟩. The expiry time indicates when the entry will be purged from the route table. Whenever a route entry is used for sending data, its expiry time is reset. The active neighbors for a particular destination are all those neighbors who have originated or forwarded a packet for this destination in the last active timeout period.

The route discovery process has a timeout value. If a route reply is not obtained within this time, another route request is sent. The optimal value for the number of route request retries is found to be 2.

3.2.3 Hello Packets

A node keeps track of its neighbors by means of hello messages. If a node hasn't sent any packets to its active neighbors within an interval known as the hello interval, it sends a hello packet. This is an unsolicited route reply. The ttl is set to 1 to limit the propagation of the hello packet to only the node's neighbors. The sequence number of the node is not incremented before sending the hello packet. If a node does not receive more than the allowed hello loss number of consecutive hello messages from the next hop for a particular route entry, it assumes link failure. The optimal value of allowed hello loss is found to be 2. If the node does not receive hello messages from an inactive neighbor, it does not take any action.

3.2.4 Link Failure

Link failure can be detected using hello messages or link layer acknowledgements. When a link fails, the node upstream of the failure sends an unsolicited route reply to all active neighbors in routes for which this link is the next hop. The destination sequence number is incremented and the hop count is set to infinity in this reply. The active neighbors in turn forward the route reply to their active neighbors and so on until the source is notified. When the source node receives the reply, it may choose to initiate a route discovery if it still needs a route to the destination. The destination sequence number must be incremented in the new route request to ensure that old invalid routes are not returned.

3.3 Comparison of DSR and AODV

In [9], the differences in the working of the DSR and AODV implementations in ns-2 are presented and their performance is compared. While both DSR and AODV discover routes on-demand, the main difference between them is that DSR uses source routing while AODV uses next hop routing. Due to the use of source routing in DSR, nodes which forward packets learn of the source routes to all nodes in the route record of the received packet. On the other hand, in AODV, a node that receives a route request learns of only the route to the source of the request. Moreover, DSR uses promiscuous reception of packets, thus enabling nodes to cache the source routes in all packets they hear. Due to this aggressive caching in DSR, the routing load is lower in DSR than in AODV.

In DSR, multiple routes to a destination are stored while in AODV only a single route is maintained for each destination. Also, AODV uses timers to purge route entries which have not been used recently. In the DSR implementation in ns-2, stale route entries are not purged, except on reception of a route error. In conditions of low mobility, routes may remain valid for long periods of time and hence DSR performs well. However, in high mobility situations, the entries in the route cache in DSR may be invalid.

AODV uses destination sequence numbers to specify the freshness of a route and to prevent the formation of routing loops. In DSR, no freshness information is maintained.

In AODV, a route error is broadcast to all the precursors of a route whose next hop has failed. These nodes then rebroadcast the error to their precursors and so on until all source nodes are notified. On the other hand in DSR, a route error is unicast to the

source of the packet which encountered the error. Hence, all sources which may use the failed link are not notified immediately.

In DSR, the destination of a route request, replies to multiple receptions of the same request from a source. This ensures that multiple routes are returned by the destination to the source. However, as a result of this, a major portion of the routing overhead in DSR is due to route replies. In AODV, route requests contribute a major portion of the routing overhead. If 802.11 is used as the MAC protocol, before transmitting a unicast packet, a RTS/CTS exchange has to take place. Since the number of route replies is high in DSR and route errors are unicast, the number of unicast packets is much higher in DSR than in AODV. Hence a large number of RTS/CTS messages are generated in DSR resulting in a higher MAC load than in AODV.

This concludes our discussion of on-demand routing protocols for MANETs. In the next chapter, we conclude our discussion of background work with a discussion of an Internet-Draft specification for integrating service discovery with on-demand routing protocols in MANETs. We then go on to discuss the details of our work.

Chapter 4

Integrating On-Demand Service and Route Discovery

The motivation for integrating service discovery with route discovery was discussed in Section 1.3.2. In this chapter, we discuss the Internet-Draft that specifies the details of this approach and an earlier simulation of this draft specification by Sandeep Gupta, using AODV as the routing protocol. We also point out the bugs in this simulation. We then present a modified version of the simulation using AODV and a simulation using DSR as the routing protocol. We highlight the modifications and additions made to the draft. We also discuss the service termination problem and propose some solutions for it. We describe the solution we have chosen and propose this an extension to the Internet-Draft.

4.1 Internet-Draft Specification

In Internet-Draft [21], Koodli and Perkins propose a method for integrating service discovery with on-demand route discovery. This method allows a node which needs a service to discover the address of a service provider and to discover a route to this provider at the same time.

Integration of service discovery with a proactive routing protocol requires adding

service information to the periodic route update messages. The service information would need to be processed by the receiving nodes. However, no separate service discovery process is required.

On the other hand, integration of service discovery with an on-demand routing protocol requires a separate service discovery process. Service extensions have to be added to the route request and reply messages and a protocol has to be defined for the processing of these messages.

4.1.1 Service Specification

A service is specified using either the TCP/UDP port at which the service can be accessed or the service type and predicate as defined in the Service Location Protocol RFC [17].

4.1.2 Service Binding

Service discovery involves finding a mapping from the service specification to the IP address of a node providing the service. This mapping is also known as a service binding. Each service binding has an associated lifetime after which the binding is no longer considered valid.

4.1.3 Combined Service and Route Discovery

If a node which needs to access a service (initiator) has a valid binding and a valid route to the service, it doesn't need to initiate any discovery process. Otherwise, the node broadcasts a route request with a service request extension. Such a request is called a service request (SREQ). The discovery process is complete when the node receives a route reply with a service reply extension. Such a reply is called a service reply (SREP).

4.1.4 Service Request Processing

There are two kinds of service request extensions. The service request port extension has the service port while the service request url extension has the service type and

predicate. Depending on how the service is specified, either the port or the url extension is used.

There are three different cases in which the initiator broadcasts a SREQ.

1. If the node doesn't have either a valid binding or a valid route to the service, it must do both service and route discovery. In this case, the node sends a SREQ with destination address set to 0 and destination sequence number (if required) set to unknown.

2. If the node has a valid binding but no valid route to the service, it must do only route discovery. In this case, the node sends a SREQ with destination address set to the IP address from the binding and destination sequence number (if required) set to the last known sequence number from the route for this destination.

3. If the node has a valid route but no valid binding for a service, it is not aware that the route is to the desired service. Hence it must do both a service and route discovery. In this case, the node sends a SREQ with destination address set to 0 and destination sequence number (if required) set to unknown.

When a node receives a SREQ, if it doesn't have a service binding for the specified service, it rebroadcasts the SREQ. If the node has a service binding, there are two cases to consider as follows.

1. If the node has a valid route to the resolved IP address, it sends a SREP back to the initiator of the request. Note that the node sends a SREP if it has a route to either a non zero destination specified in the received SREQ or a possibly different destination obtained from the node's service binding for the specified service.

2. If the node doesn't have a valid route to the resolved IP address, it sets the destination address to the resolved address and destination sequence number (if required) to the last known sequence number from the route and rebroadcasts the SREQ.

4.1.5 Service Reply Processing

The service reply extension contains a service URL as defined in [17]. It also contains the lifetime of the service binding. When an intermediate node on the path back to the initiator receives a SREP, there are two cases to consider.

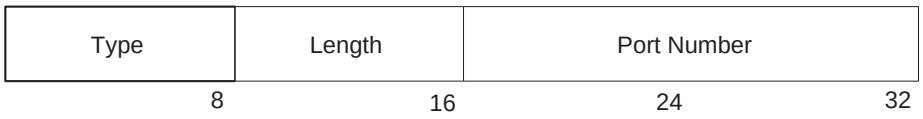

Figure 4.1: Service Request Port Extension

1. If the node has a better binding for the service specified in the service reply extension, it replaces the binding in the SREP with the better binding and forwards the SREP towards the initiator.

2. Otherwise, the node simply forwards the SREP towards the initiator.

4.1.6 Message Formats

1. **Service Request Port Extension**
   The format of the service request port extension is given in Figure 4.1. The purpose of each field is as follows:

   Type : To Be Determined (1 byte)
   Length : Length of the extension (1 byte)
   Port Number : TCP/UDP port number used by the service (2 bytes)

2. **Service Request URL Extension**
   The format of the service request url extension is given in Figure 4.2. The purpose of each field is as follows:

   Type : To Be Determined (1 byte)
   Length : Length of the extension (1 byte)
   ServiceTypeLen : Length of the service type field (1 byte)
   Reserved : For future use (1 byte)
   Service Type : Service type as defined in RFC [17] (variable length)
   Service Predicate : Service predicate as defined in RFC [17] (4 bytes)

3. **Service Reply Extension**

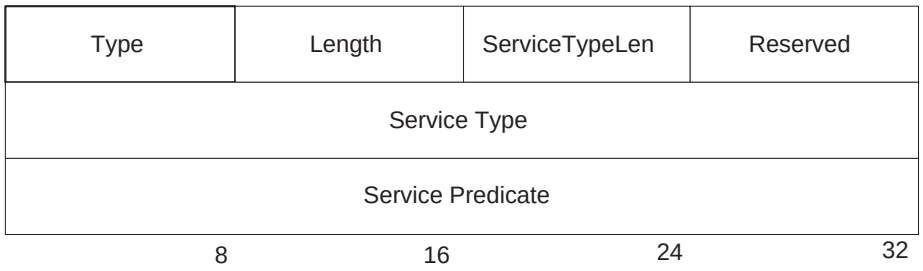

Figure 4.2: Service Request URL Extension

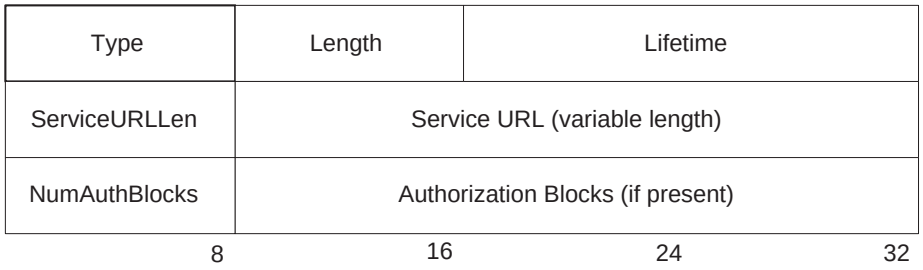

Figure 4.3: Service Reply Extension

The format of the service reply extension is given in Figure 4.3. The purpose of each field is as follows:

Type : To Be Determined (1 byte)
Length : Length of the extension (1 byte)
Lifetime : Lifetime of the service binding (2 bytes)
ServiceURLLen : Length of the service URL field (1 byte)
Service URL : Service URL as defined in RFC [17] (variable length)
NumAuthBlocks : Number of authorization blocks as defined in RFC [17] (1 byte)
Authorization Blocks : Authorization blocks as defined in RFC [17] (3 bytes)

4.2 Routing Loops

In the description of DSR in [20], Johnson and Maltz specify that a node that is not the target of a RREQ can send a RREP. The route in such a RREP is formed by appending the route to the target taken from the node's route cache to the route record from the request. However, they also mention that if the complete route thus formed contains a loop, the RREP should not be sent. In this way, routing loops are avoided in DSR. In the DSR implementation in ns-2, we observe that if the complete route contains loops, all such loops are removed before sending the RREP.

In the description of AODV in [37], Perkins and Royer prove that the use of destination sequence numbers in AODV ensures that loop-free routes are obtained. This proof depends on the use of destination sequence numbers and the mechanism of a node forwarding a RREP only if the destination sequence number in the reply is greater than that in its routing table entry for the destination. Since this is done even in the integrated approach, we infer that loop free routes will still be obtained when service discovery is integrated with AODV.

4.3 Integration of Service Discovery with AODV

The Internet-Draft [21] specification described in the previous section was simulated in ns-2 [14], [30] by Sandeep Gupta [16] using AODV as the routing protocol. The details of this simulation are presented in this section. For details of the simulation of AODV in ns-2, please refer to Appendix A.

We note that Sandeep Gupta proposed some modifications and additions to the Internet-Draft and implemented the integrated protocol in AODV with these modifications. However, there are a number of bugs in this simulation which have been highlighted using bold italics.

4.3.1 Modifications to Internet-Draft Specification

The following modifications and additions to the Internet-Draft specification have been proposed by Sandeep Gupta.

1. The Internet-Draft states that a node should not reply to a SREQ if it doesn't have a service binding for the specified service. However, this simulation assumes that if the request destination is known, a node can reply to a SREQ even if it doesn't have a service binding for the specified service.

2. When a SREP is received by a node which doesn't have a binding for the specified service or if the binding in the SREP is better, the node stores the binding from the SREP.

3. In [16], Sandeep Gupta mentions that before a node terminates its service, it must notify all active connections. This extension was however not implemented by Gupta in his integration with AODV. We discuss the service termination problem in detail and propose solutions in Section 4.6.

Note that care must be taken to make sure that a binding at a node is not replaced by a better binding from a SREP with a different destination while the service is being used by an application. Otherwise, the application would have to change from one service provider to another provider of the same service midway through its exchange of data. In this thesis, we consider only the service discovery process and not the subsequent exchange of data with the service provider. Hence, this issue doesn't arise, but needs to be kept in mind when the invocation of the service is also considered.

4.3.2 Assumptions

The following assumptions were made -

1. A node which adds a service binding to a SREQ, uses a service request url extension. This is because the service request port extension can only include the service port and not the service type. The service request url extension on the other hand, has a service type and a service predicate field. The service predicate field can be used to store the port number.

Based on the original Internet-Draft specification and the above modifications to the draft and assumptions, the signal flow graphs for the integrated service and route

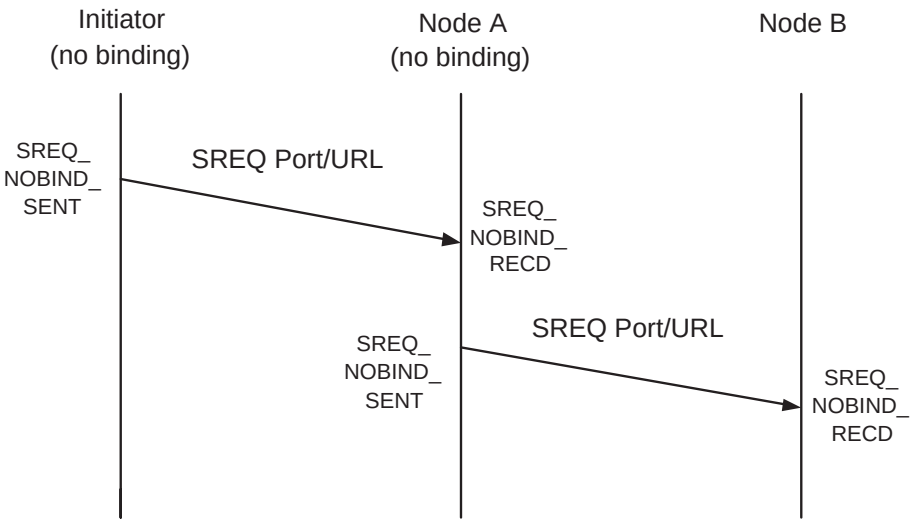

Figure 4.4: Initiator doesn't have a binding, intermediate node also doesn't have a binding

discovery process are shown in Figures 4.4, 4.5, 4.6, 4.7 and 4.8. The meaning of the node states shown in these figures is as follows.

SREQ_NOBIND_SENT : The node has sent a SREQ without a binding (with a port or url extension)

SREQ_NOBIND_RECD : The node has received a SREQ without a binding (with a port or url extension)

SREQ_BIND_SENT : The node has sent a SREQ with a binding (with a url extension)

SREQ_BIND_RECD : The node has received a SREQ with a binding (with a url extension)

SREP_SENT : The node has sent a SREP

SREP_RECD : The node has received a SREP

4.3.3 Service Table

This is the table used for storing the service bindings. Each entry in the service table has the service port, service string (service type), address of the node providing the service, the expiry time and the number of hops to the service. The entries in the service discovery table expire after a maximum of 10 seconds. However, when a node itself provides

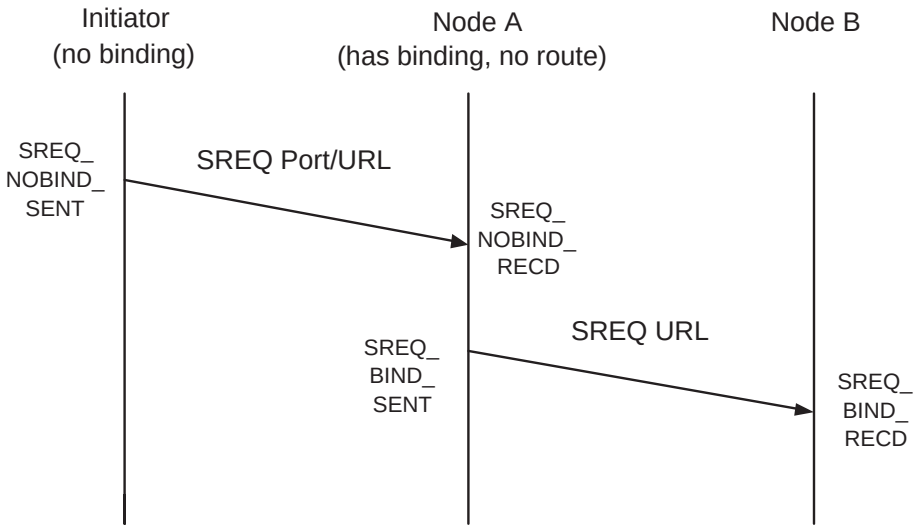

Figure 4.5: Initiator doesn't have a binding, intermediate node has a binding but no route

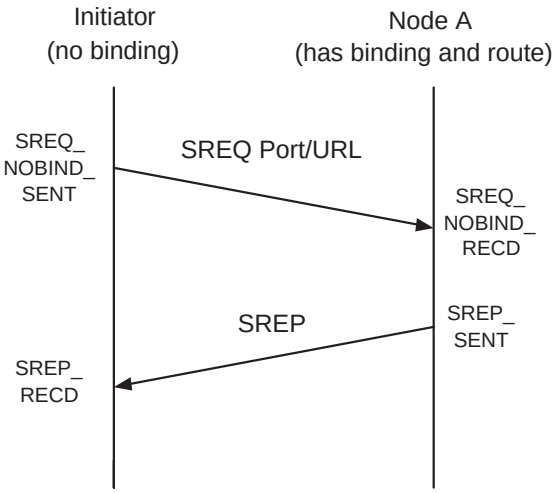

Figure 4.6: Initiator doesn't have a binding, intermediate node has a binding and route

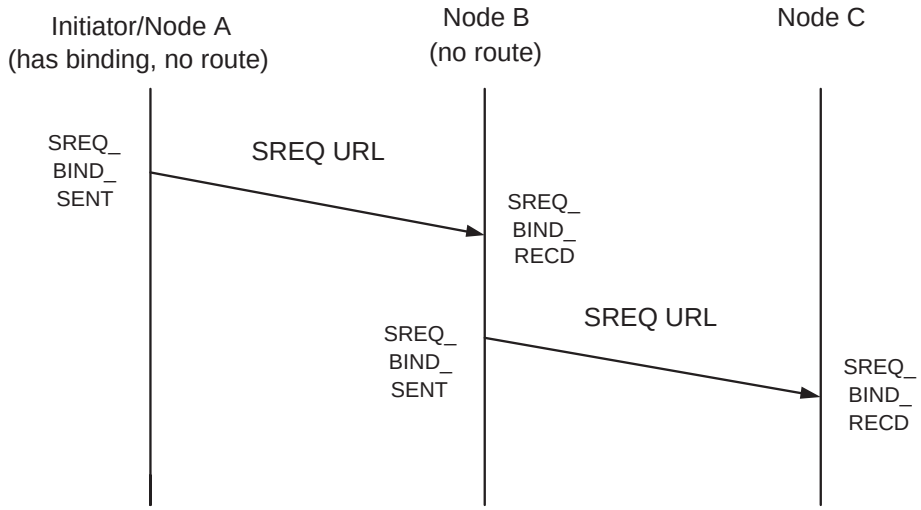

Figure 4.7: Initiator has a binding, intermediate node doesn't have a route

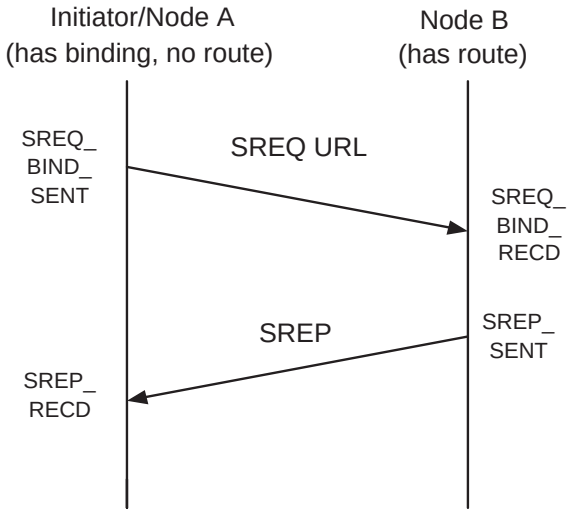

Figure 4.8: Initiator has a binding, intermediate node has a route

a service, the lifetime of the entry for this service in its service table is set to 600 seconds.

4.3.4 Packet Formats

All service extensions contain the type and length of the extension. The RREQ header contains either the service request port extension or the service request url extension. The RREP header contains the service reply extension.

- Service Request Port Extension - This contains the service port number.
- Service Request URL Extension - This contains the length of the service string, the service string and the service predicate. The maximum length of the service string is set to 32 bits. In this simulation, the service predicate is being used only for storing the service port number if available.
- Service Reply Extension - This contains the lifetime of the binding, the url length, the url, the number of authorization blocks and the authorization block data. The maximum length of the url has been set to 255 bits. Also the maximum length of authorization block data has been set to 24 bits. In this simulation, the number of authorization blocks and the authorization block data are not being used.

4.3.5 Service Discovery Queue

This queue stores the pending service requests. Each entry in the queue has the port number, service string and expiry time of the service request. The service discovery process times out 5 seconds after a service request is sent.

4.3.6 Timer

The purging of expired entries from the service queue and the service table has been added to the broadcast timer. This timer runs every 6 seconds. Before purging an entry from the service queue, a message saying that the service could not be found is displayed.

4.3.7 AODV Agent with Service Discovery

The agent has a service discovery queue and a service table. Destination "-1" is considered to be an unknown destination. A route entry for destination "-1" is added to the route table (with status set to "down" and all fields except destination address set to 0).

4.3.8 SREQ Transmission

If the service string is known, create a service request url extension. The service string is filled in the service type field and if available the port no is filled in the service predicate field. If only the port number is known, create a service request port extension.

Look up the service table for the specified port number or service string. If no entry is found in the service table, a SREQ is to be sent with destination address set to "-1".

If an entry is found and it hasn't expired, display a message that the service provider has been found. ***Note that this is a bug since we also need to check for a route to the provider and if there is no route, we need to send a SREQ.***

If an entry is found but it has expired, a SREQ is to be sent with destination address set to the provider address from the entry. ***Note that this is a bug since we should not be using expired entries in the service table. Another bug is that before the SREQ is sent, we need to make sure that there is an entry for the destination of the request in the route table. If there is no entry, one needs to be added (with status set to "down" and all fields except destination address set to 0). This is required because the function which transmits the SREQ assumes that there is an entry for the destination of the request in the route table.***

The process for sending the SREQ is the same as that for sending a RREQ except that the service request extension is added to the route request. The process is as follows.

If there is a valid route for the destination, don't send a SREQ.

If there is a pending request for the destination, don't send a SREQ. ***Note that this is a bug since we may need to send multiple SREQs with destination "-1" at around the same time. Since the SREQs all have the same destination they share the same route table entry and hence the same request timeout values. So***

there may be a pending SREQ for service type "A" with destination "-1". Now a SREQ for service type "B" with destination "-1" will not be sent due to the pending SREQ which shares the same entry in the route table. Thus if a node needs to send multiple SREQs for different services but all with destination "-1" at around the same time, only the first SREQ will be sent and the others will not be sent at all since the first one would not have timed out.

If the request count (number of network wide requests sent) is greater than the maximum allowed, reset the request count and set timeout to the maximum request timeout. Drop all queued packets for the destination and do not send the SREQ. ***Note that there is a bug in this due to the same reason as mentioned before. Since all SREQs ever sent by a node with destination "-1" share the same route table entry, the request count may be incremented by SREQs for different services and hence would not be correct.***

Set the last ttl used to the maximum of the last ttl used and the last valid hop count. If the last ttl used is 0, set the ttl to ttl_start (5 hops). Otherwise, if the ttl is less than the ttl_threshold (7 hops), increment the ttl by ttl_increment (2 hops). Otherwise, set the ttl to the network diameter (30 hops) and increment the request count. Store the last ttl value used. ***Note that there is a bug in this due to the same reason as mentioned before. Since all SREQs ever sent by a node with destination "-1" share the same route table entry, the value of the last ttl used may be changed by SREQs for different services and hence would not be correct.***

Set the request timeout value based on the ttl used and the per hop delay. If we have done network wide broadcast before, set the timeout to a higher value, but make sure it is not greater than the maximum request timeout. ***Note that there is a bug in this due to the same reason as mentioned before i.e. the ttl and the request count used may not be correct.***

Set the request hop count to 1 and increment the broadcast id. If there is a route to the destination, set the destination sequence number to its value from the route, otherwise set it to 0. Increment this node's sequence number by 2. Send the SREQ as a broadcast packet without any delay.

Add an entry to the end of the service discovery queue for the service port and string. Set the expiry time to 5 seconds from the time of sending the request. ***Note that this is a bug since an entry will still be added to the queue even if the SREQ***

is not sent due to there being a route or a pending request or because we have exceeded the maximum number of network wide requests allowed. Also there is a bug in the code for adding an entry to the tail of the queue — the tail pointer is not updated to point to the new entry.

Note that a major bug is that no retransmission of SREQs will take place since there are no packets waiting in the route queue for a route to the service. In the case of RREQs, there are packets waiting in the route queue for a route to the destination. These packets will trigger the retransmission of a RREQ. Retransmission of SREQs has to be handled in a different manner.

4.3.9 SREQ Reception

The initial processing of the SREQ is the same as for a RREQ. It is as follows. If this node is the source of the SREQ, drop the SREQ (it has traversed a loop). If this node has received the ⟨source address, broadcast id⟩ pair before, drop the SREQ (it is a duplicate). Otherwise, store the received ⟨source address, broadcast id⟩ pair.

Check for an entry for the source of the SREQ in the route table. If there is no entry, add one. This is the reverse route entry. Set the expiry time to the maximum of the route's expiry time and the reverse route lifetime (6 seconds). If the request has a higher source sequence number or the same sequence number but a smaller number of hops than the route table entry, update the entry with the request's route details for the source. If there is a pending request for the source, reset the request count and timeout and set the last ttl to the request hop count. Set the expiry time to the active route timeout (10 seconds). Forward any packets that were waiting for a route to the source, without any delay.

If there is a service request port or url extension, the following processing is done as shown in Figures 4.9 and 4.10.

The service table is looked up for either the port or string depending on whether the service request extension is a port or a url extension. If the request destination is unknown and there is no entry in the service table, the SREQ must be rebroadcast. The IP source address is changed to this node's address and the request hop count is incremented. The SREQ is then broadcast with a delay of 1 second.

Otherwise, a service reply extension is created. If an entry was found in the

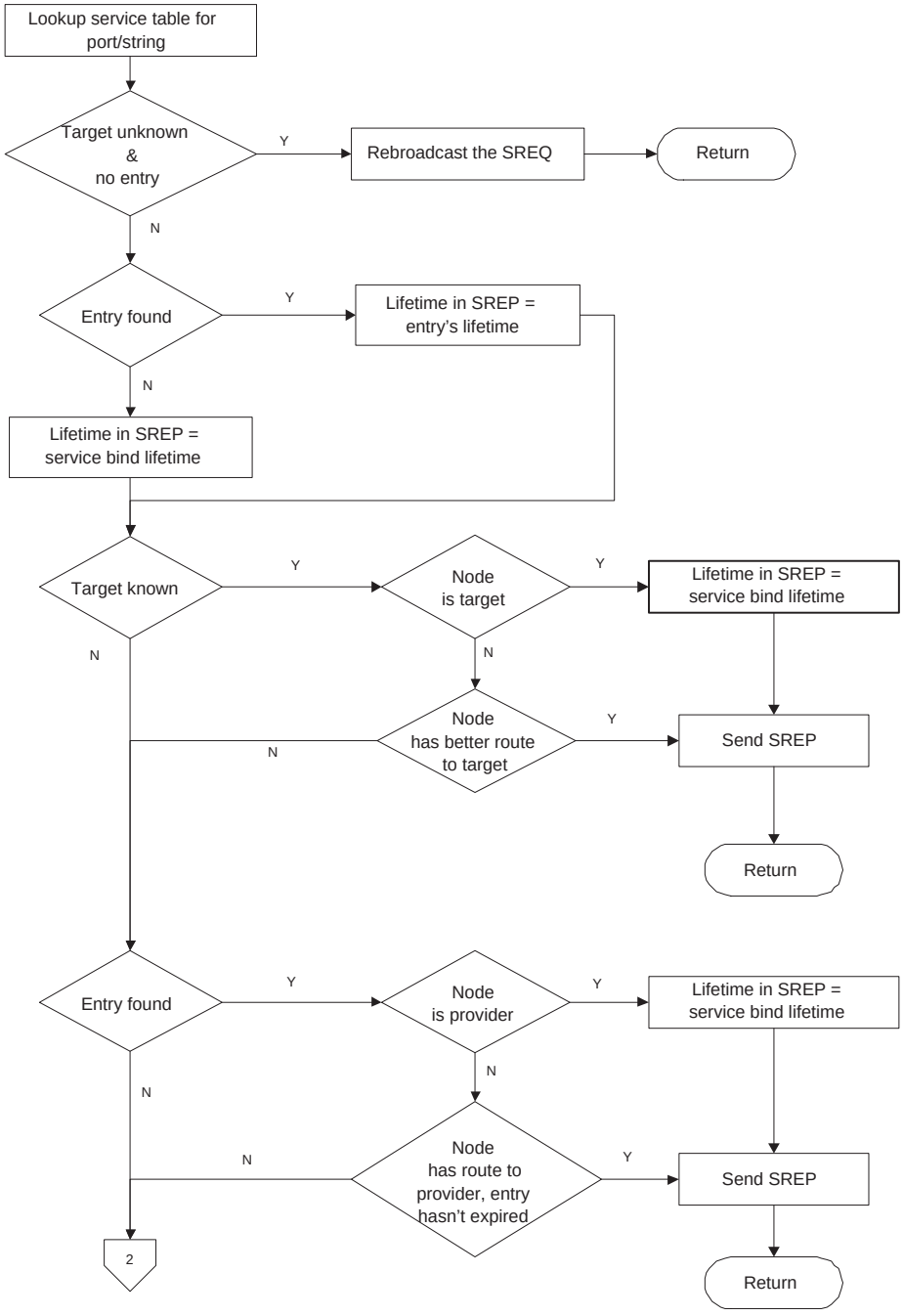

Figure 4.9: Processing of received SREQ

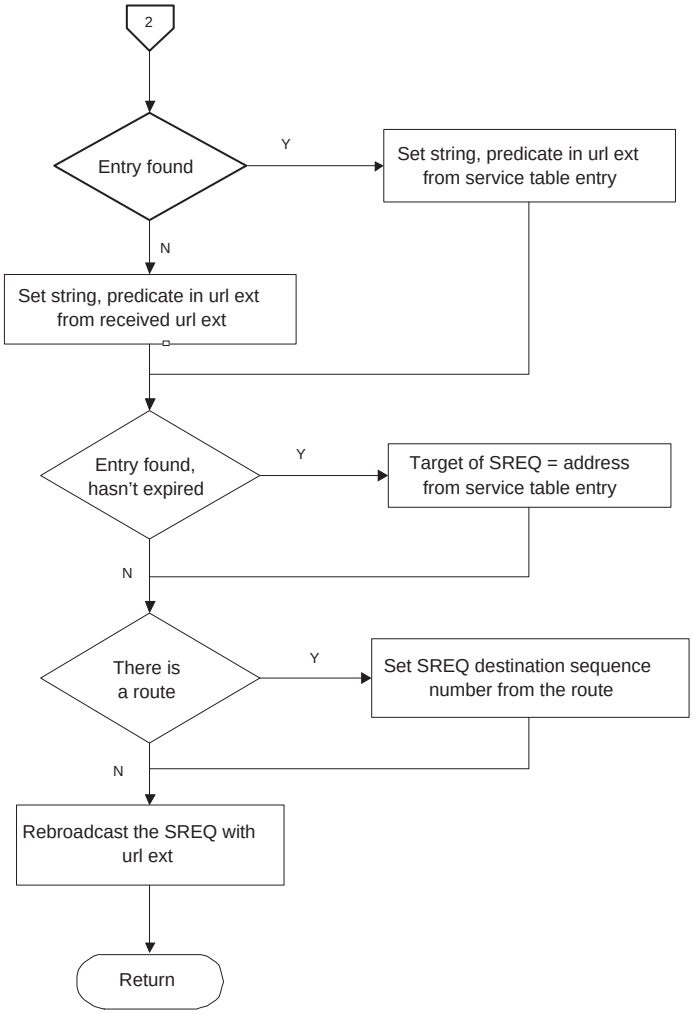

Figure 4.10: Processing of received SREQ continued

service table, the binding lifetime in the SREP is set from the entry, otherwise it is set to the maximum service binding lifetime (10 seconds). ***Note that this is a bug since the replying node should not set the binding lifetime in the SREP when it doesn't have an entry in its service table for this binding. Actually, the message formats defined in the Internet-Draft need to be modified so that a lifetime field is added to the service request url extension. Once this is done, if a node resolves the service port/type to the provider address but does not send a SREP, it can add the binding lifetime to the service request url extension. This way if the node which sends the SREP is not the node which provided the binding, it is not responsible for adding the binding lifetime to the SREP.***

If the request destination is known, check for a route to the destination. If this node is the destination or has a route to the destination with a valid number of hops and the same or higher sequence number than that in the request, a SREP must be sent.

If there is an entry in the service table and this node is the provider, a SREP must be sent.

Also, if there is an entry in the service table which hasn't expired and there is a route to this provider with a valid number of hops, a SREP must be sent. ***Note that this is a bug since the destination sequence number in the request and route are not compared before sending the SREP. The request destination and destination sequence number may have been inserted by the source of the SREQ. In this case, if the SREQ reaches a node with a stale route to the destination but which has a service table entry for the requested service, a SREP will be sent by the node.***

In all the above cases, before sending a SREP, the service url must be filled. If there is no entry in the service table, the service url can be set using the fields from the service request url extension. The service url is of the form "service:serv_type://address:port". For example, if node "45" provides service "web_server" at port "80", the service url will be "service:web_server://45:80".

Also, if the node sending the SREP is the provider, the node's sequence number is incremented and the binding lifetime in the SREP is set to the maximum service binding lifetime. The reply hops is set to 1, the destination sequence number is set to this node's sequence number and the reply route lifetime is set to my_route_timeout (10 seconds).

If the node sending the SREP is not the provider, the reply hops is set to the hop

count from the route entry + 1. The destination sequence number and reply route lifetime are set from the route entry. The precursor of the route to the source is set to the next hop of the route to the destination. The precursor of the route to the destination is set to the next hop of the route to the source.

Otherwise, the SREQ must be rebroadcast with a service request url extension. First, a service request url extension is created. If there is an entry in the service table, the service type and predicate fields in the url extension are set to the service string and port number respectively from the entry. If there is no entry in the service table, the service type and predicate fields are copied from the received service request url extension.

The IP source address is set to this node's address. If there is an entry in the service table which hasn't expired, the request destination is set to the provider address from the entry. ***Note that the Internet-Draft does not explicitly state whether a node should replace a binding in a received SREQ with a different binding if the node has one. This simulation assumes that the binding in the received SREQ should be replaced.***

If there is a route to the provider, the destination sequence number is set to that in the route. ***Note that this is a bug, since the destination sequence number in the route entry may be smaller (more stale) than the destination sequence number in the request. Hence, the SREQ would now request a staler route than the original SREQ. Also a route lookup is not done for the new destination which could be different from the original destination. So the route entry being checked may be for the old destination. In this case, the destination sequence number in the request will correspond to that of the old destination.***

The request hop count is incremented and the SREQ is broadcast with a service request url extension with a delay of 1 second.

4.3.10 SREP Transmission

The process for sending the SREP is same as for the RREP except that a service reply extension is added. The process is as follows.

Look up the routing table for a route to the destination of the SREP. The next hop in the SREP is set to the next hop from the route and ttl is set to the network diameter (30 hops). The SREP is sent without delay.

4.3.11 SREP Reception

The reception of the SREP is initially handled in the same way as for a RREP. The process is as follows.

Look up the routing table for a route to the reply route destination. If there is no route, add one. If the reply route has a higher sequence number than that in the route entry or the same sequence number but a smaller number of hops, update the route entry with the reply route. Also, reset the request count and timeout and set the last ttl to the reply hop count. If this node is the destination of the reply, update the per hop route discovery latency. Forward all packets which were waiting for a route to the reply route destination. Delay successive packets by arp_delay (10 milliseconds).

If there is a service reply extension, the following processing is done.

Extract the service string and port from the service url in the reply. Look up the service table for the service string. If there is no entry in the service table, add one. Otherwise, if the service table entry has a smaller binding lifetime and greater number of hops than than that in the reply, delete the service table entry and add an entry based on the SREP.

Otherwise, the service table entry is better than the reply binding. Hence, we will replace the binding in the SREP with the one from the service table. If this node is the provider, set binding lifetime in the SREP to the maximum service binding lifetime, otherwise set it to the lifetime from the service table entry. Fill the service url based on the string, port and provider from the service table entry. Set the reply route destination to the provider from the service table entry. ***Note that this is a bug since the provider in the received service reply may be different from the provider in the service table entry. Hence, the node needs to check whether it has a route to the provider in the service table entry, before updating the binding lifetime, url and reply route destination. Also, if the reply route destination is modified, other fields in the SREP like destination sequence number, number of hops and route expiry time have to be modified to correspond to the new reply route destination.***

Iterate through the service discovery queue and find the entry which corresponds to the port or string in the received SREP. If there is an entry, display a message saying the service has been found and remove this entry from the queue.

The rest of the processing of the SREP is the same as that of the RREP. It is as

follows.

If this node is the destination of the SREP or the reply route is not better than the route table entry, drop the SREP. Otherwise, check for a route entry for the destination of the reply. If there is no route entry, drop the SREP. If there is an entry with valid number of hops, increment the reply hop count, change the source address to this node's address and forward the SREP without any delay. Also, set the precursor of the route to the reply route destination as the next hop of the route to the destination of the SREP.

4.3.12 Interface to AODV Agent

The following three commands provide the OTcl interface to the AODV Agent service discovery process.

- To start the service discovery process for a particular port number:
  $agent svc-disc-port ⟨port_num⟩
- To start the service discovery process for a specific service string:
  $agent svc-disc-string ⟨serv_str⟩
- To add an entry to the service table for a service which this node provides:
  $agent svc-bind ⟨port_num⟩ ⟨serv_str⟩
  An entry is added with the specified port number and service string. The lifetime is set to 600 seconds and the number of hops is set to 0.

4.4 Modified Integration of Service Discovery with AODV

We propose a number of additions and modifications to the Internet-Draft specification for integration of service discovery with on-demand routing protocols. The modified integration of service discovery with AODV is presented in this section. In this modified simulation, the bugs which were highlighted in the previous section have been corrected and all the proposed modifications and additions to the Internet-Draft have been implemented. Note that the service discovery queue is no longer used, since pending service requests are stored in the service table.

We retain the same maximum service binding lifetime of 10 seconds as we feel that 10 seconds is neither too high nor too low. A binding lifetime that is too low would cause entries to be purged from the service table too frequently. However, a binding lifetime that is too high would cause stale bindings for services which have terminated to remain in the network for too long. Note that later on in this chapter, we propose an extension to the Internet-Draft for notifying all nodes in the network when a node terminates the service it provides. However, this extension assumes that a node knows ahead of time that it is going to terminate its service. If this is not the case, there may be stale bindings in the network and we do not want these to be retained for long. Also, both the AODV and DSR specifications include the purging of route entries after a certain duration. In the AODV implementation in ns-2, the lifetime of a route is 10 seconds while the DSR implementation in ns-2 does not implement purging of route entries. In a dynamic network, we cannot assume that a node will continuously provide service and hence this selection of service binding lifetime is justified.

4.4.1 Modifications to Internet-Draft Specification

We propose the following additions and modifications to the Internet-Draft.

1. A service binding lifetime field of 2 bytes is added to the service request url extension. This is done so that a node which adds a binding to a request but doesn't have a route to the service, can add the binding lifetime to the request. The node which finally replies with a route, may not have a binding and hence should not be responsible for providing the binding lifetime. The modified format of the service request url extension is given in Figure 4.11.

2. No mention was made in the draft about retransmission of SREQs. The approach we follow for retransmitting SREQs is similar to that followed for retransmitting RREQs. However, in the AODV and DSR simulations in ns-2, RREQ retransmission is never really terminated as long as there is a waiting packet. Instead, the request timeout is calculated based on the number of retransmissions that have been attempted. On the other hand, for SREQ retransmissions, we propose that if a certain number of retransmissions have been attempted, the service discovery process should be terminated.

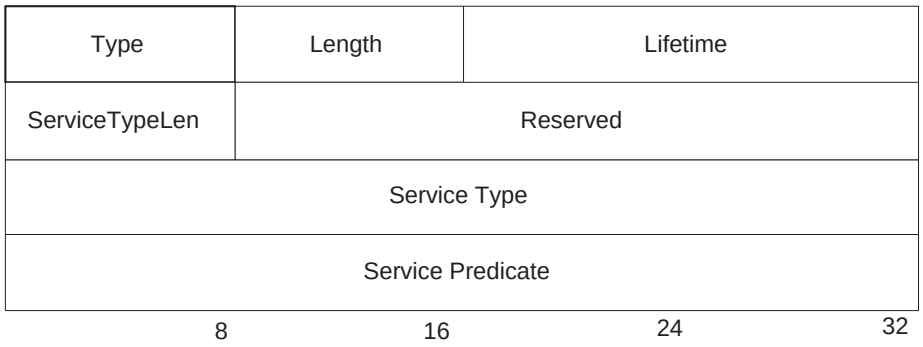

Figure 4.11: Modified Service Request URL Extension

3. If a network wide SREQ with known service destination has timed out, subsequent network wide retransmissions of the SREQ must be sent with unknown destination. This ensures that other providers of the same service can also send a SREP. This way, if the known service provider is not responding or no longer provides the service, alternate providers can be found.

4. When a SREQ is received at a node which is the request destination or which provides the requested service, the node must send a SREP for its service. Only if the request destination is not this node and this node does not provide the service, will a different service destination be considered. This is done to make sure that the request isn't propagated further than is required.

5. When a SREQ is received at the request destination but the node no longer provides the service, it still sends a SREP with binding lifetime set to 0.

6. When a SREQ with a service binding is received at a node, if the node does not have any binding for this service then add the binding from the request to this node. Otherwise, if this node has a binding but it has expired and there is no pending request for this service, then replace this binding with the binding from the request.

7. When a SREP with service binding lifetime set to 0 by the service provider is received by a node which has a valid binding for this service and provider, this binding should

be removed.

8. When a SREP is received by a node which has a better binding for the service and a shorter route to the service destination, the binding and route in the SREP should be replaced with the node's binding and route. Note that the Internet-Draft does mention that a node should replace the binding in a SREP with a better binding, if it has one. However, no mention is made of also checking for a route to the service destination.

Note that, as per the original Internet-Draft, a node which needs to access a service and has a binding but no route to the service, sends a SREQ and not a RREQ. This is because the binding may be stale. If it sends a SREQ and this SREQ reaches the service provider and it no longer provides the service, it will send a SREP with lifetime set to 0 as proposed in the extensions above. Nodes which receive this SREP will then purge a binding for this service provider, if they have one. Hence, it is better to send a SREQ instead of a RREQ.

4.4.2 Modified Service Table

A request count, request timeout and last ttl are added to each entry in the service table. The request count stores the number of network wide service requests that have been broadcast for the service. The request timeout stores the time at which the service request will timeout. The last ttl stores the ttl value used in the last SREQ that was sent for this service. When a service request with destination "-1" is sent, an entry for the service is added to the service table with service address set to "-1". In the following discussion, an entry in the service table is considered "valid" if it has a non zero service port, non null service string, the service destination is not "-1" and the entry hasn't expired.

4.4.3 Service Timer

A service timer has been added to the AODV agent. The purging of expired service table entries is shifted from the broadcast timer to the service timer. This was done because the broadcast timer runs only every 6 seconds. The service timer is set up to expire every 0.1 seconds. Also, an entry is purged only if it has expired and its request timeout is zero

(since we don't want to purge pending requests). The service discovery process doesn't time out after 5 seconds as before, but rather retransmissions are attempted in a similar manner to retransmissions for RREQs. The check for whether a SREQ has timed out has been added to the service timer.

4.4.4 Modified SREQ Transmission

Lookup the service table for either the specified port or the service string. If there is a valid entry, check whether the service destination is this node itself. If it is, display a message that this node itself provides the service. Reset the request count and timeout values in the service table entry.

Otherwise, check for a route to the service destination. If there is a route with a valid hop count, display a message that the service is available. Reset the request count and timeout values in the service table entry.

If there is no valid route, add a route (with status set to "down" and all fields except destination address set to 0). Create a service request url extension with the service port, type and binding lifetime values taken from the service table entry. A SREQ is to be sent with destination set to the provider address from the service table entry.

Otherwise, if there is no entry in the service table, add an entry for the service port and/or service string with destination set to "-1", lifetime set to 0 and number of hops set to 255 (considered as infinity). If we know the service string, fill a service request url extension with lifetime set to 0, otherwise fill a service request port extension. A SREQ is to be sent with destination "-1".

The process for sending the SREQ is same as before, except that the service request count, timeout and last ttl values are stored in the service table entry. In the earlier implementation they were stored in the route table entry and hence multiple requests with destination "-1" shared the same values of request count, timeout and last ttl.

If the request destination is not "-1", the request count, timeout and last ttl values are first copied from the corresponding route table entry before they are used. Also, only if the request destination is not "-1", the last ttl value is initially set to the maximum of the last ttl and the last request hop count.

If the request destination is not "-1", after the values of request count, timeout and last ttl have been set in the service table entry, the modified values are saved in the

corresponding route table entry as well. This is done to make sure that a RREQ and a SREQ for the same destination use and modify the same values. Doing this ensures that a SREQ will not be sent for a destination while a RREQ for the same destination is pending and vice versa.

If the service request count is greater than or equal to the maximum number of service request retries (2), the service table entry is deleted and a message saying that the service could not be discovered is displayed. This is unlike the method for retransmitting RREQs. In the case of RREQs, if the route request count is greater than the maximum number of route request retries (3), the request count is reset to 0 and the timeout is set to the maximum request timeout (10 seconds).

4.4.5 Modified SREQ Reception

The initial processing of the SREQ is same as for a RREQ. The processing of the service request extension has been modified as follows.

If there is a service request port extension, lookup the service table for the port, otherwise lookup for the string.

If the request destination is not known and there is a valid entry in the service table, fill a service request url extension with the string, port and lifetime values from this entry. Set the SREQ destination to the destination from the service table entry.

If the request destination is not known and there is no valid entry in the service table, rebroadcast the SREQ.

If the request destination is known and it is not this node, check whether there is a valid entry in the service table and the destination in the entry is different from the request destination. If this is the case and the destination in the entry is this node, then the request destination must be changed. Otherwise, if there is no route to the request destination with valid number of hops and route destination sequence number greater than or equal to the request destination sequence number, check for a route to the destination in the service table entry. If there is a route with valid number of hops, the request destination must be changed.

If the request destination must be changed, set the request destination and binding lifetime from the service table entry and set the destination sequence number to 0.

If the request destination is known and there is no entry in the service table, add

an entry with the service port, string and lifetime taken from the SREQ url extension and the service destination set to the request destination. Also, if there is an entry in the service table but it has expired and the timeout is 0 (i.e. there is no pending request for it), replace this entry with the entry based on the SREQ url extension.

Fill the service reply extension using the request destination and the service port, string and lifetime from the service request url extension.

If this node is the request destination, check whether there is a valid entry in the service table with the service destination set to this node. If there is such an entry, set the lifetime in the service reply extension to that in the service table entry, while making sure that it is not greater than the maximum service binding lifetime (10 seconds). If there is no such entry in the service table, set the service binding lifetime to 0. Send a SREP.

If this node has a route to the request destination with a valid number of hops and the route destination sequence number is greater than or equal to the request destination sequence number, send a SREP. Otherwise, rebroadcast the SREQ with url extension.

4.4.6 Modified SREP Reception

The initial processing of the SREP is same as for a RREP. The rest of the processing of the SREP is modified as follows.

Extract the service port and string from the url received in the SREP. Lookup the service table for the service port or string.

If the binding lifetime in the reply is zero, check whether the reply route destination is the same as the service destination in the service table entry. If it is, set the service destination to "-1", expiry time to 0 and hop count to 255 (considered as infinity) in the service table entry. If the binding lifetime in the reply is non zero, the following processing is done.

If there is a service table entry and the timeout is non zero, display a message that the service has been found.

If there is no service table entry, add an entry based on the SREP. If there is a service table entry with non zero timeout or whose lifetime is less than that in the SREP, replace the entry with an entry based on the SREP. When adding an entry to the service table, the request count and timeout values are set to 0 and the last ttl value is set to the reply hop count.

Otherwise, check whether there is a valid entry in the service table and binding lifetime in the entry is greater than the SREP binding lifetime and service destination in the entry is different from the reply route destination. If this is the case, check for a route to the destination in the service table entry. If there is a route with smaller hop count than the reply hop count, the binding from the service table must be used in the SREP. The binding lifetime, service destination and reply route destination are set to the values from the service table entry. The reply route destination sequence number, reply hop count and route lifetime are set to the values from the route entry.

The rest of the processing of the SREP is same as for a RREP.

4.4.7 SREQ Retransmission

The retransmission of SREQs is handled in the service timer. We want to make sure that if a network wide SREQ with known destination has timed out, then the retransmitted network wide SREQ must be sent with destination set to "-1". This is because if a service provider is no longer responding or no longer provides a service, we don't want to retransmit a SREQ for the same service provider, but instead we want to send a SREQ without a known request destination. This way other nodes in the network which provide the same service can send a SREP. Hence, the following processing is done in the service timer.

Iterate through the entries in the service table. Check if there is an entry for which the service request has timed out. If there is an entry, check whether the number of network wide service requests attempted is greater than or equal to 1 and the service destination is known. If this is the case, set the service destination to "-1", binding lifetime to 0 and service hop count to infinity in the service table entry. Also, if there is a route entry for the known service destination, reset its route request count and timeout. Now attempt to retransmit the SREQ.

4.4.8 Modified Interface to AODV Agent

When a node terminates the service it provides, the entry for this service in its service table needs to be removed. The following command has been added to the OTcl interface to allow a node to delete the entry in its service table for the specified service port and string.

$agent svc-unbind ⟨port_num⟩ ⟨serv_str⟩

4.5 Integration of Service Discovery with DSR

The same packet formats and OTcl interface are used as in the modified integration of service discovery with AODV. All the modifications to the Internet-Draft proposed in the earlier sections have been implemented in the integration of service discovery with DSR in ns-2. Hence, the integration of service discovery with DSR is similar to the modified integration of service discovery with AODV.

An address of "10000" is used in the simulation to indicate an unknown destination in a SREQ. (Note that ns-2 doesn't use dotted decimal notation for representing node addresses.) This would translate to "0.0.39.16" in dotted decimal notation. In the following sections, the parts that we have added to the simulation of DSR for purposes of service discovery are in *italics*, while the other parts were already implemented for route discovery. For details of the simulation of DSR in ns-2, please refer to Appendix A.

4.5.1 Service Table

Each entry in the service table includes the service port, string, destination address, service expiry time and number of hops. To facilitate retransmission of SREQs, each entry also has the time of the last arp, time of the last service request, the number of outstanding service requests and the type of the last service request. These fields serve the same purpose as the corresponding fields in the request table. Note that the type of the last request could be either a "ring zero" or a "propagating" search. A ring zero search sets the ttl value to 1, while a propagating search sets the ttl to the maximum source route length (16 hops). A third type "none" is added to cover the case when no service request has been sent yet for a particular service. The field for time of the last arp stores the time when a ring zero search was initiated.

4.5.2 Service Timer

A service timer is added to the DSR agent. This timer expires every 0.1 seconds. When the timer expires, all entries in the service table which have expired and for which

there are no pending service requests are deleted. The check for pending service requests is also done in the service timer.

4.5.3 DSR Agent

The DSR agent has a service table and a service timer.

4.5.4 SREQ Transmission

Lookup the service table for the port/string. If there is a valid entry, check whether this node provides the service or has a route to the service. If this is the case, display a message that the service has been found. Otherwise, set the request destination and binding lifetime in the url extension based on the service table entry.

If there is no entry in the service table, add an entry with service destination as unknown, expiry time as 0 and hop count as infinity. Set the request destination as unknown.

If the request destination is known, copy the time of last arp, time of last request and number of outstanding requests from the request table entry for this destination.

If the number of outstanding service requests is greater than or equal to the maximum service retries (3), display a message that the service could not be discovered and remove the entry from the service table.

Otherwise, calculate the timeout till the next SREQ based on the number of outstanding SREQs and the route request backoff period (0.5 seconds), while making sure the timeout doesn't exceed the maximum route request timeout (10 seconds). Based on the time of the last request and the timeout, calculate the time of the next request.

If it is time for a request to be sent, check whether the last request type is "none". If this is the case, set the last request type to "ring zero", the last arp time to the current time and the propagation limit in the request header to 0. Otherwise, set the last request type to "propagating" and the propagation limit in the request header to the maximum source route length (16 hops).

If it is not time for a request to be sent, check whether the last request type is "ring zero" and the arp timeout (30 milliseconds) has been exceeded. If this is the case, set the last request type to "propagating" and the propagation limit in the request header to the maximum source route length. Otherwise, don't send a SREQ.

Increment the number of outstanding service requests and set the time of last service request to the current time. If the request destination is known, copy the time of last arp, time of last request, number of outstanding requests and last request type from the service table entry to the request table entry for this destination.

Set the source address as this node. If the service string is known, fill a service request url extension, otherwise fill a service request port extension.

The rest of the procedure for sending a SREQ is same as for a RREQ. It is as follows.

The node's request id is incremented and added to the SREQ header. The node's address is added to the source route in the header. If the agent has error data that hasn't timed out, then this data is added to the SREQ to form a gratuitous route error. If the interface queue is full, the request is dropped. Otherwise, the request is broadcast with a jitter of 10 milliseconds.

4.5.5 SREQ Reception

If a port extension has been received, lookup the service table for the port, otherwise lookup for the string. If the request destination is not known and there is a valid service table entry, change the request destination and set the binding lifetime to that in the service table entry. The port and string in the url extension are filled.

If the request destination is known and is not this node, check whether there is a valid entry in the service table and the service destination in the entry is different from the request destination. If this is the case and the service destination in the entry is this node, change the request destination and binding lifetime to that in the service table entry. Also, if this node doesn't have a route for the request destination but has a route for the destination in the service table entry, change the request destination and binding lifetime to that in the service table entry.

The rest of the processing of the SREQ is same as that of a RREQ except for a few additions. The whole process is explained below.

- ***Node is Not the Request Destination***

  If the request destination is not this node, the following processing is done.

  If this request is a duplicate, has traversed a loop or its source route is full, it is dropped. *If the request destination is known and there is a url extension, lookup the service table for the service string. If there is no entry, add an entry based on the url extension. If there is an entry but it has expired and it doesn't have any pending requests, replace that entry with an entry based on the url extension.*

  Add the received sequence number and source address to the request table. If this is a ring zero search or this node can reply from its route cache, then no further processing is done.

  The replying from route cache is done as follows. If there is a route to the request destination, append this route to the route record from the request. If the complete route is too long, do not send the reply. Otherwise, remove all loops from the complete route. *If there is a service request url extension, use the url extension and the request destination to form a service url and store the lifetime for the service reply extension.*

If the request also contains data or a route reply or a non gratuitous route error, forward them to the request destination. Set the source route in the forwarded packet to the complete route and *invalidate the* route request and *the service request url extension in the packet.*

Drop the request and form the reply. Add this node's address to the route record from the request and fill the reply route with this path. Reverse this route to get the path to be followed by the reply and fill the source route in the reply with this path. Add this reversed route to the route cache. Set the ttl to 255. *The stored service url and binding lifetime are filled in the service reply extension.* The SREP is then sent to the initiator without any delay. This completes the replying from route cache.

If the SREQ is propagating and the node could not reply from cache, the following processing is done.

If the length of the interface queue is greater than 10, free airtime is less than 15% or the request has exceeded its propagation limit or the source route is full, drop the SREQ. Otherwise, add this node's address to the source route. If the interface queue is full, drop the SREQ. Otherwise, rebroadcast the request with a jitter of 10 milliseconds.

- ***Node is the Request Destination***

  If the request destination is this node, the following processing is done.

  If the request destination is known and there is a url extension, lookup the service table for the service string. If there is no entry, add an entry based on the url extension. If there is an entry but it has expired and it doesn't have any pending requests, replace that entry with an entry based on the url extension.

  Add the received sequence number and source address to the request table. If the source route is full, drop the request. Otherwise, add this node's address to the route record from the request and fill the reply route with this path. Reverse this route to get the path to be followed by the reply and fill the source route in the reply with this path. Add this reversed route to the route cache. Set the ttl to 255.

  If there is a service request url extension, check whether there is a valid entry for the requested service in the service table with the service destination set to this node. If there is such an entry, set the lifetime in the service reply extension to that in the

service table entry, while making sure that it is not greater than the maximum service binding lifetime. If there is no such entry in the service table, set the service binding lifetime to 0 in the reply. Form the service url and send the SREP to the initiator with a jitter of 10 milliseconds.

4.5.6 SREP Reception

- ***Node is Not the Reply Destination***

  If the node is not the reply destination, the following processing is done.

  If we have gone beyond the end of the source route, drop the reply. Otherwise, add the source route in the reply to the route cache. If the ttl is 0, drop the reply. Otherwise, decrement the ttl.

  If there is a service reply extension, extract the service port and string from the received url. Lookup the service table for the service port or string.

  If the binding lifetime in the reply is zero, check whether the reply route destination is the same as the service destination in the service table entry. If it is, set the service destination to unknown, expiry time to 0 and hop count to infinity in the service table entry. If the interface queue is full, drop the reply. Otherwise, forward it without any delay.

  If the binding lifetime is non zero, the following processing is done.

  If there is no service table entry, add an entry based on the SREP. If there is a service table entry whose lifetime is less than that in the SREP and it doesn't have any pending requests, replace the entry with an entry based on the SREP. When adding the entry to the service table, the time of last arp, time of last request and number of outstanding requests are all set to 0 and the last request type is set to "none".

  If there is a valid entry in the service table and the binding lifetime in the entry is

greater than the binding lifetime in the SREP, check whether there is a route to the service destination in the service table entry. If there is a route, check whether the number of hops in this route is smaller than the number of hops in the reply route from this node to the service destination in the SREP. If this is the case, the binding and route in the SREP need to be replaced with the better binding and route available at this node. Replace the binding lifetime in the SREP with that from the service table entry. Set the service destination in the service url to that in the service table entry. Change the reply route to include the route to the destination in the service table entry.

If the interface queue is full, drop the reply. Otherwise, forward it without any delay.

- ***Node is the Reply Destination***

  If the node is the reply destination, the following processing is done.

  Add the reply route to the route cache. Reset the number of outstanding requests and time of last request in the request table entry for the reply route destination. *If there is a service reply extension, reset the time of last arp and set the last request type to "ring zero" in the request table entry.* Forward any packets in the send buffer which were waiting for the reply route, delaying successive packets by the arp timeout (30 milliseconds).

  If there is a service reply extension, extract the service port and string from the received url. Lookup the service table for the service port or string.

  If the binding lifetime in the reply is zero, check whether the reply route destination is the same as the service destination in the service table entry. If it is, set the service destination to unknown, expiry time to 0 and hop count to infinity in the service table entry. Drop the reply.

  If the binding lifetime is non zero, the following processing is done.

  If there is a service table entry and there are pending requests, display a message that the service has been found.

If there is no service table entry, add an entry based on the SREP. If there is a service table entry whose lifetime is less than that in the SREP or there are pending requests, replace the entry with an entry based on the SREP. When adding the entry to the service table, the time of last arp, time of last request and number of outstanding requests are all set to 0 and the last request type is set to "none".

4.5.7 SREQ Retransmission

The retransmission of SREQs is handled in the service timer. We want to make sure that if a propagating (network wide) SREQ with known destination has timed out, then the retransmitted propagating SREQ must be sent with unknown destination. This is because if a service provider is no longer responding or no longer provides a service, we don't want to retransmit a SREQ for the same service provider, but instead we want to send a SREQ without a known request destination. This way other nodes in the network which provide the same service can send a SREP. Hence, the following processing is done.

Iterate through the entries in the service table. Check if there is an entry with pending service requests. If there is an entry, check whether the number of pending service requests is greater than or equal to 2 and the service destination is known. If this is the case, set the service destination to unknown, binding lifetime to 0 and service hop count to infinity in the service table entry. Also, set the time of last arp, time of last request and number of outstanding requests to 0 and the last request type to "ring zero" in the request table entry for the service destination. Now attempt to retransmit the SREQ.

4.6 Service Termination Problem

The integrated approach as given in the Internet-Draft, has the following problem. If a node shuts down its service either voluntarily or involuntarily, other nodes will not be informed of this. So they will continue to regard the service binding as valid and add this binding to SREQs and SREPs. So a node may receive a SREP with the service provider

specified as a node which no longer provides the service. Hence, there is a need to inform all nodes in the network that a service binding is no longer valid, once a service has been terminated. We note that in [16], Sandeep Gupta mentions that before a node terminates its service, it should notify all active connections. However, it is not sufficient to notify only nodes which are part of active connections. Stale bindings may be present at other nodes as well and these need to be purged.

Several solutions are possible. All these solutions assume that a node knows that it is going to shut down its service. When a node terminates its service, it removes its stored service binding for itself and then does one of the following.

1. The node broadcasts an unsolicited SREP in which the service reply extension has a binding lifetime of 0. This indicates that the service binding is no longer valid. Any node which receives this SREP must remove this binding from its service table.

2. The node adds a service reply extension to the next hello message which it sends. This extension contains the service port and type of the service it used to provide and a binding lifetime of 0. This indicates that the service binding is no longer valid. Only the nodes neighbors will receive this hello message. Each neighbor will then add this extension to the hello message which it transmits to its neighbors in the next hello interval. While this solution does not require any new messages to be transmitted, it may take a while before all the nodes in the network are informed that the node no longer provides the service. Moreover, hello messages are used only in AODV and not in other on-demand routing protocols for ad hoc networks.

3. The node broadcasts a service error (SERR) message. When a node receives a SERR message, if it has a binding for the specified service, it removes this binding.

An SREP includes a RREP and a service reply extension. Hence, an SREP includes details about the route to the service like destination sequence number, hop count and expiry time. However, we do not need to send these route details since the route to the node may be unchanged even though it has terminated its service. Hence, a SREP is not the ideal message to send in this case. On the other hand, the use of hello messages would result in a long time elapsing before all the nodes are informed about the service termination. By this time, the service binding for this service may have anyway timed out. Additionally, hello messages already exist only in AODV. The addition of hello messages to

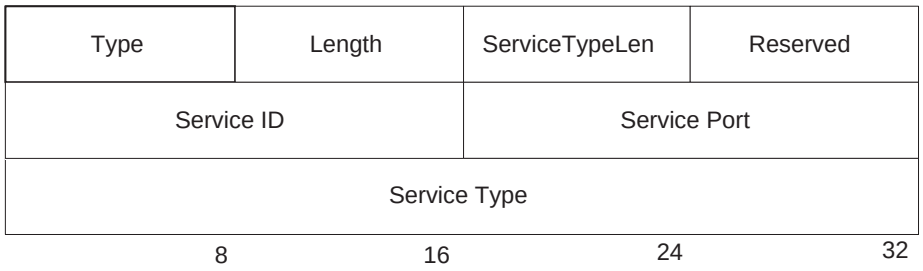

Figure 4.12: Service Error Message

other on-demand routing protocols for the purpose of service termination, would result in a lot of extra bandwidth consumption. Hence, we have chosen the use of a SERR message to solve the service termination problem. We describe this approach in detail in the next section and propose this as an extension to the Internet-Draft.

4.6.1 Service Error Message

The format of the service error message (SERR) is given in Figure 4.12. The purpose of each field is as follows:
Type : To Be Determined (1 byte)
Length : Length of the extension (1 byte)
ServiceTypeLen : Length of the service type field (1 byte)
Reserved : For future use (1 byte)
Service ID : The source node's current service id (2 bytes)
Service Port : TCP/UDP port number used by the service (2 bytes)
Service Type : Service type as defined in RFC [17] (variable length)

4.6.2 Service ID

Each node has a service id which is incremented each time the node terminates a service it provides. The service id is used to identify duplicate service error messages. Each node stores the value of the largest service id that it received from a source of a SERR

message.

4.6.3 Service Error Transmission

When a node terminates a service it provides, it removes its binding for its service. It also fills the service port and string in a service error message. It increments its service id and adds this id to the SERR message. It then broadcasts the SERR message in the network with a specific ttl value.

4.6.4 Service Error Reception

When a node receives a SERR message, it checks what was the largest service id received from the source of the SERR message. If the largest service id received is greater than or equal to the service id in the received SERR message, the node drops the SERR message. Otherwise, the node stores the received service id as the largest service id received from the source of the SERR message. It then checks if it has a binding for the specified service port and type. If it has a binding and the service provider is the source of the SERR message, the node removes this binding. The node then decrements the ttl and if it is 0, drops the SERR message. Otherwise, it rebroadcasts the SERR message in the network.

4.7 Cost and Limitations of Integrated Approach

The addition of service discovery to a routing protocol causes some overhead for the routing protocol. The only way to distinguish between a pure route discovery process and a combined service and route discovery process is by checking for a service request extension or service reply extension. Hence, whenever a RREQ or RREP message is received as part of pure route discovery, it must be checked for a service request extension or a service reply extension respectively. This check would add to the processing time of a RREQ and a RREP.

Another overhead caused due to the addition of service discovery is the periodic expiration of a service timer. This timer handles the purging of expired entries from the service table and the retransmission of SREQs. In our simulations, this timer expires every 100 milliseconds.

Also, we have proposed an additional message for the purpose of service discovery. This is the service error message which is broadcast in the network when a service is terminated. Broadcast messages cause significant overhead in ad hoc networks.

The addition of service discovery also requires additional memory for the service table which stores service bindings and pending service requests.

We discussed the service termination problem in the previous section and proposed solutions for it. However, all these solutions assume that a node knows when it is going to terminate a service and can react accordingly. However, a node may not know ahead of time that it is going to terminate its service. In such a situation, other nodes in the network may still have a stale binding for the terminated service and may reply to a SREQ for this service. Note that if the node terminates its service abruptly and also moves away, leaves the network or shuts down, the normal route maintenance process will take care of notifying the other nodes in the network. The problem arises when a node abruptly terminates its service but the route to it is still valid.

Moreover, it is possible for a malicious node to send a service error message indicating that a particular node has terminated its service. In such a case, all nodes in the network which have a binding for this provider will incorrectly purge the binding even though the node still provides the service. However, this problem is not limited to the SERR message. A malicious node can send an incorrect SREP message as well. Similarly in the case of pure routing, an incorrect RREP or RERR message can also be propagated. Security concerns in on-demand routing protocols and hence in the integrated approach is beyond the scope of this thesis.

Note that in this thesis, we consider only the service and route discovery process and not the subsequent exchange of data with the service provider. It is possible that the service and route discovery process returns a route to a service provider which subsequently terminates its service. Hence, data packets to the service provider may be dropped or may not be processed correctly. It is then the responsibility of the higher layers to initiate a service and route discovery again, so that a valid service provider can be found. This is unlike the mechanism for triggering a pure route discovery. The presence of a waiting data packet for a destination for which a node doesn't have a route, triggers a route discovery for this destination. However, in the case of combined service and route discovery, a node will not often know the address of the service provider and hence a data packet cannot trigger the combined service and route discovery process.

This concludes our discussion of the details of integrating service discovery with on-demand routing protocols. We have discussed the Internet-Draft for the integrated approach, modifications and additions to the draft, the simulation of the integrated approach using AODV and DSR as well as the cost and limitations of the integrated approach. In the next chapter, we present our simulation setup and results for the integrated approach and analyze the results obtained.

Chapter 5

Simulation Results

In this chapter, we present the simulation results for the integrated approach to service discovery using AODV and DSR. We compare the performance of the integration of service discovery with these two routing protocols for different simulation scenarios in ns-2. The performance metrics we consider are route acquisition latency and path length. The time that elapses between the sending of the first request for a service and the reception of the first valid reply is taken to be the route acquisition latency. The reply either sets up or contains a route to the service provider. The path length is the number of hops in this route to the service provider.

5.1 Simulation Setup

We have taken Sandeep Gupta's test script and enhanced it significantly to extensively test the service and route discovery process for different conditions of mobility, number of nodes and replication of services. Some of the simulation parameters are given in Table 5.1. We run our tests for two different sizes of the network — 50 nodes in an area of 700 meters by 700 meters and 100 nodes in an area of 1500 meters by 1500 meters. We consider network sizes of 50 and 100 nodes as these are the sizes considered in the simulation of AODV in [37] and in the simulation of AODV and DSR for comparison purposes in [9]. For each network size, the area of the topology we have chosen was the area for which the

Transmission range	250 meters
Addressing type	flat
Mac layer	802.11
Channel and network interface type	Wireless
Interface queue type	Priority queue
Interface priority queue length	50 packets
Propagation model	Two ray ground
Antenna type	Omnidirectional

Table 5.1: Simulation Parameters

number of service discovery failures was low.

We run each simulation for 600 seconds. This value of simulation time was chosen so that we can generate a sufficient number of service requests for calculating the average route acquisition latency and path length. At the same time, for 600 seconds, the node movement files generated and read for 50 and 100 nodes do not become too large and the simulation runs complete within a reasonable amount of time.

There are five different servers in the network — a web server at port 80, a mail server at port 60, a print server at port 25, a name server at port 53 and a tftp server at port 69. The ids of the nodes providing the services are uniformly distributed over all the node ids. Thus, it is possible for a single node to provide more than one service. All the servers are started at the beginning of the simulation and remain up until the end of the simulation. In other words, for these tests, no service termination is done.

Service requests are randomly generated i.e. the id of the node requesting the service is uniformly distributed over all the node ids and the time of the request is uniformly distributed over the simulation time. The type of service request is also uniformly distributed between a request specifying a service port and a request specifying a service string. The port/string for which the request is made, is uniformly distributed over the 5 services available in the network.

100 service requests are made in the simulation with 50 nodes, while 50 service requests are made in the simulation with 100 nodes. The number of service requests is chosen so that we have sufficient number of requests to calculate the average performance measures and at the same time we do not overload the network. We generate a smaller

number of service requests for the simulation with 100 nodes so as to minimize congestion caused due to the longer path lengths in a larger network. The values of route acquisition latency and path length calculated are the average values over all the requests. However, if a service discovery process is not successful, it is not considered while calculating the averages. Also, since the requests are generated randomly it is possible for a node which provides a service to try to discover the same service. Since the node will find the service available on itself, no service request needs to be generated. Hence, this case is not considered while calculating the averages. Alternatively, a node may find that it has both a service binding and a route to the service it wants to discover. In this case also, no service request needs to be broadcast and hence this case is not considered while calculating the averages.

Each data point plotted in the graphs is an average over 10 simulation replications. In each replication, each of the random numbers is generated using a different sub-stream. This results in a different set of service providers and different time of generation of requests as well as different requests in each replication. The node movement and data traffic patterns used in multiple replications of a particular simulation scenario are the same.

Node movement follows the random waypoint mobility model. This means that a node selects a random destination point within the topology area and moves towards this destination in a straight line with a constant speed which is uniformly distributed between 0 and a specified maximum speed. It then pauses at this destination for a certain amount of time, at the end of which, it selects another random destination point to move towards and the process repeats.

Some other mobility models which can be exported to ns-2 are random direction, random waypoint indirect, random waypoint gaussian and gauss markov. We chose the random waypoint mobility model as we found that it is the most widely used mobility model in research papers on ad hoc networks.

In the simulation with 50 nodes, data traffic is also generated so as to simulate a realistic network and to test the service discovery process in the presence of route requests and replies for non service related data traffic. ns-2 has a traffic generator which can generate either TCP traffic or CBR traffic (over UDP connections). We do not want to set up TCP connections. Hence, we chose to set up UDP connections between pairs of nodes and transmit CBR traffic between them. 5 such UDP connections are set up. This number was chosen so as to provide a realistic amount of data traffic without congesting the network. No data traffic is generated in the simulation for 100 nodes, as it was observed

that if data traffic is also present, a large number of service discoveries fail with DSR for 100 nodes, due to congestion in the network.

The CBR packet size is 512 bytes and a maximum of 10,000 packets are sent. These are the default values set by the traffic generator in ns-2. A packet is sent every 0.25 seconds. This is done to ensure that we don't fill up the network with only data traffic. The maximum value of start time of a connection is 180 seconds. The connection remains up and the data transfer continues till the end of the simulation.

5.2 Varying Server Redundancy

The average pause time between node movements is set to 240 seconds. We chose 240 seconds as the simulation runs for 600 seconds and we don't want the nodes to be continuously moving or continuously stationary during this time. An average pause time of 240 seconds is thus neither too high nor too low. The maximum speed of the nodes is set to 1 meters/second. The number of replicated servers of each service type is considered to be the server redundancy. For example, if a network has only a web server and a mail server and the server redundancy is three, then there are three web servers and three mail servers in the network. The simulation is run for different values of server redundancy. The values of server redundancy considered are 1, 2, 3 and 4. The average route acquisition latency, average path length and the ratio average path length/average shortest path length are plotted for the different values of server redundancy and the results using AODV and DSR for 50 and 100 nodes are shown in Figures 5.1, 5.2, 5.3 5.4, 5.5 and 5.6. Note that the shortest path length in these and other tests is obtained from the General Operations Director (GOD) object in ns-2. This object has global knowledge of the network and hence can compute the shortest path between any two nodes. Hence, the shortest path length is the shortest path between the initiator of the service discovery and the service provider about which it gets the first reply. It is not the shortest path length among all the redundant servers of a particular type in the network.

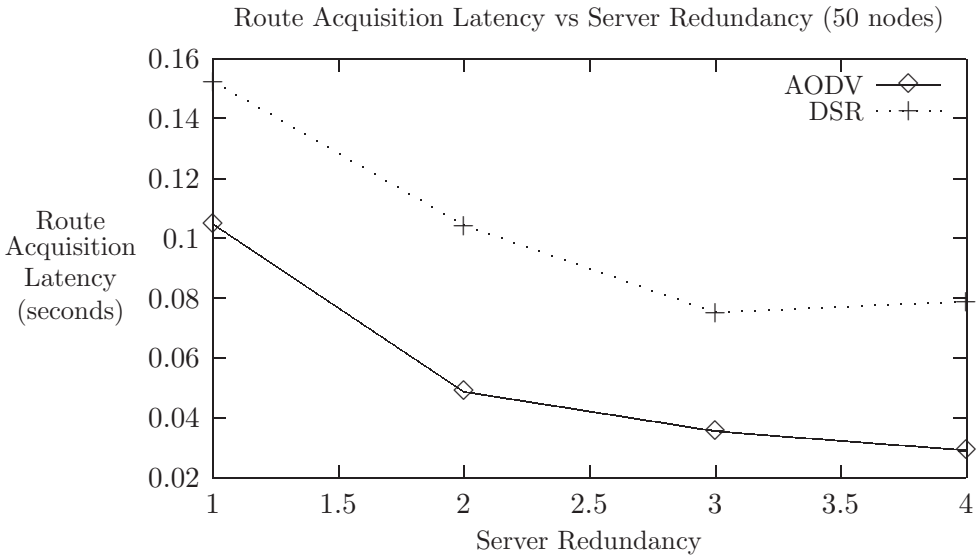

Figure 5.1: Route Acquisition Latency vs Server Redundancy for 50 Nodes

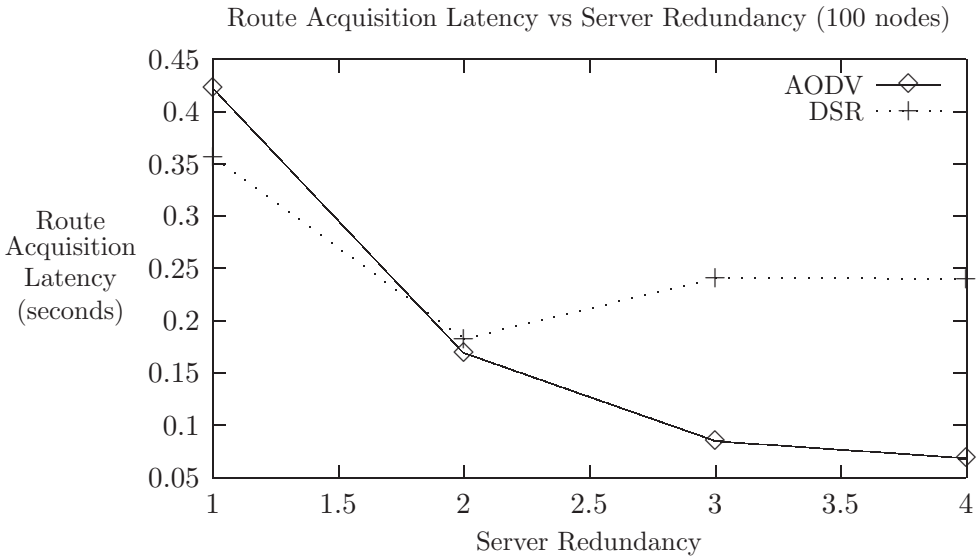

Figure 5.2: Route Acquisition Latency vs Server Redundancy for 100 Nodes

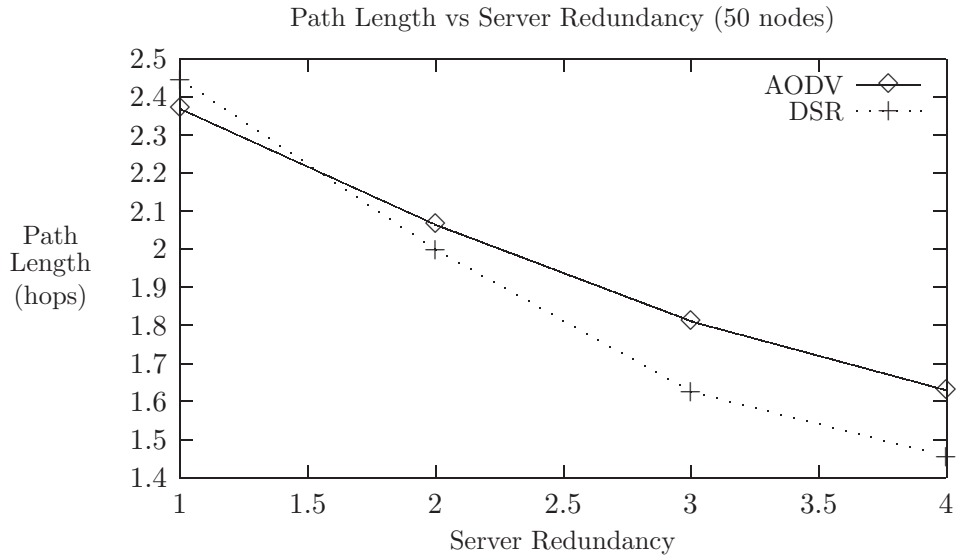

Figure 5.3: Path Length vs Server Redundancy for 50 Nodes

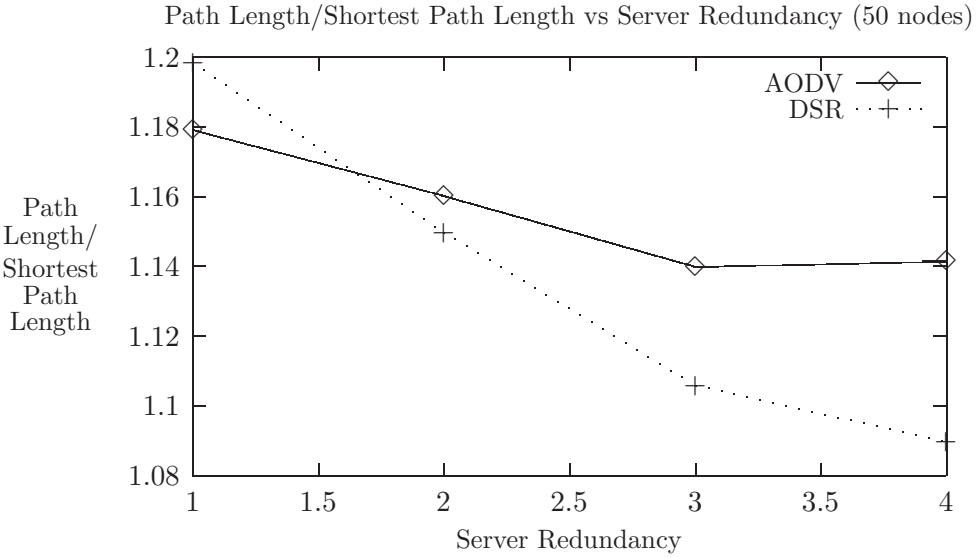

Figure 5.4: Path Length/Shortest Path Length vs Server Redundancy for 50 Nodes

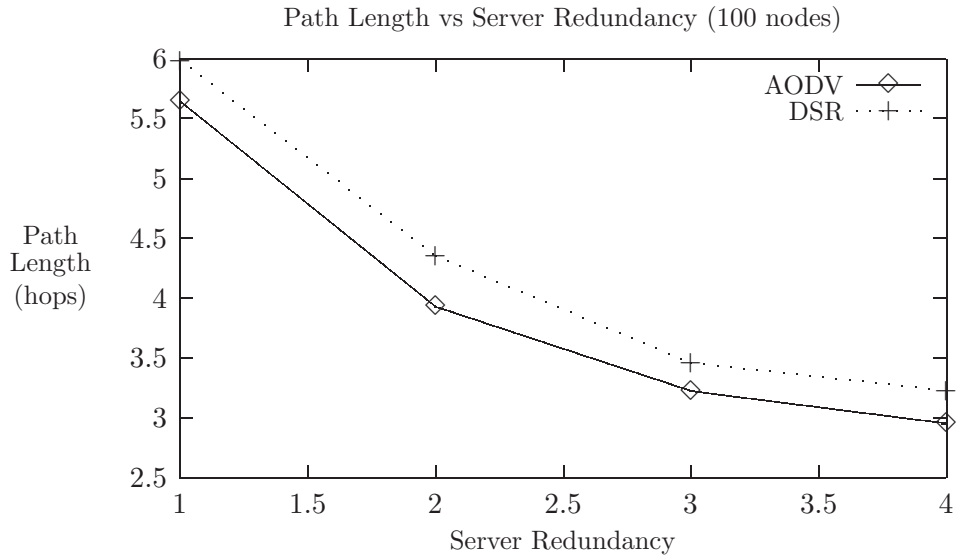

Figure 5.5: Path Length vs Server Redundancy for 100 Nodes

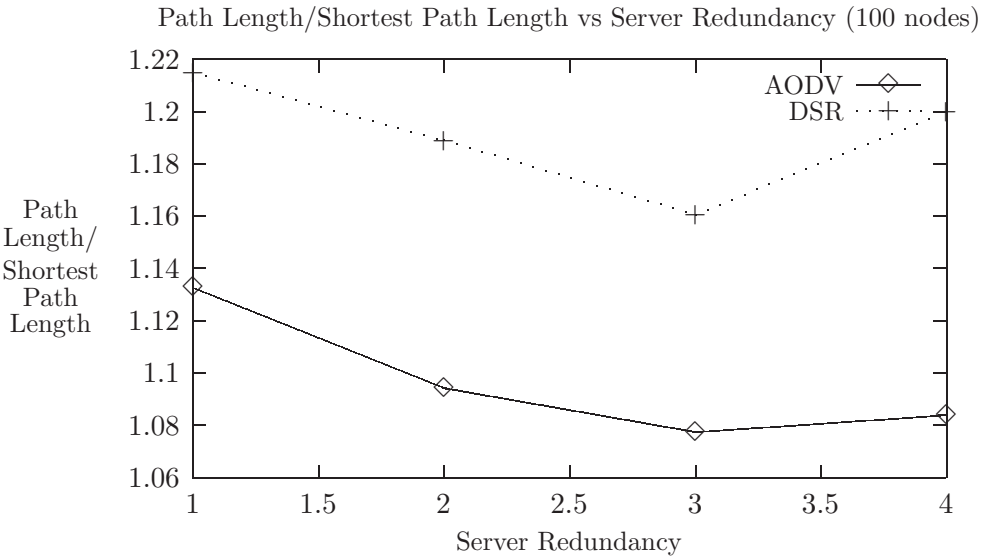

Figure 5.6: Path Length/Shortest Path Length vs Server Redundancy for 100 Nodes

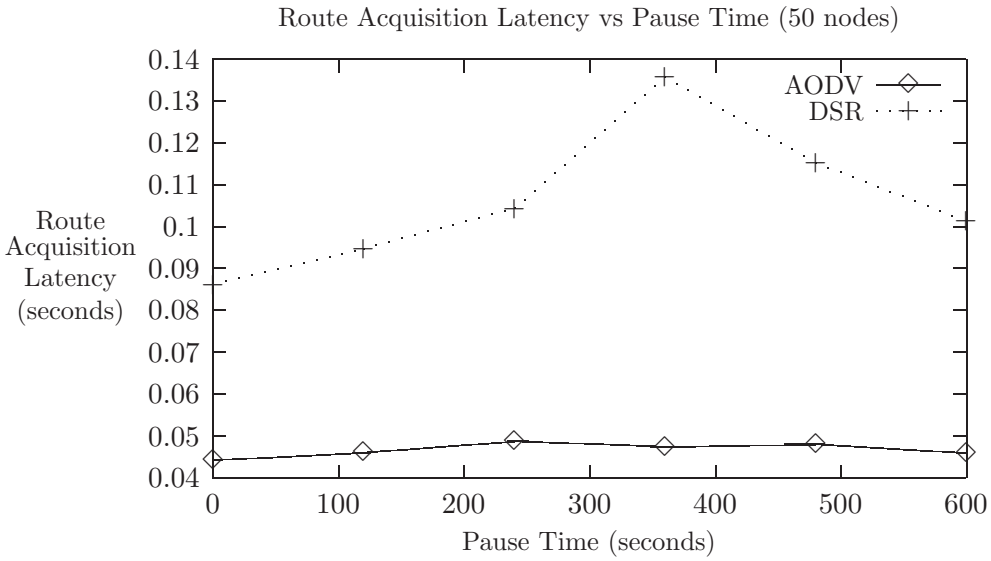

Figure 5.7: Route Acquisition Latency vs Pause Time for 50 Nodes

5.3 Varying Pause Time

The maximum speed of the mobile nodes is set as 1 meters/second. The server redundancy is set to 2. The simulation is run for different values of average pause time between node movements. The values of average pause time considered are 0, 120, 240, 360, 480 and 600 seconds. Note that a pause time of 0 means that the nodes are continuously moving while a pause time of 600 seconds in this simulation set up means that the nodes are always stationary. The average route acquisition latency, average path length and the ratio average path length/average shortest path length are plotted for the different values of pause time and the results using AODV and DSR for 50 and 100 nodes are shown in Figures 5.7, 5.8, 5.9, 5.10, 5.11 and 5.12.

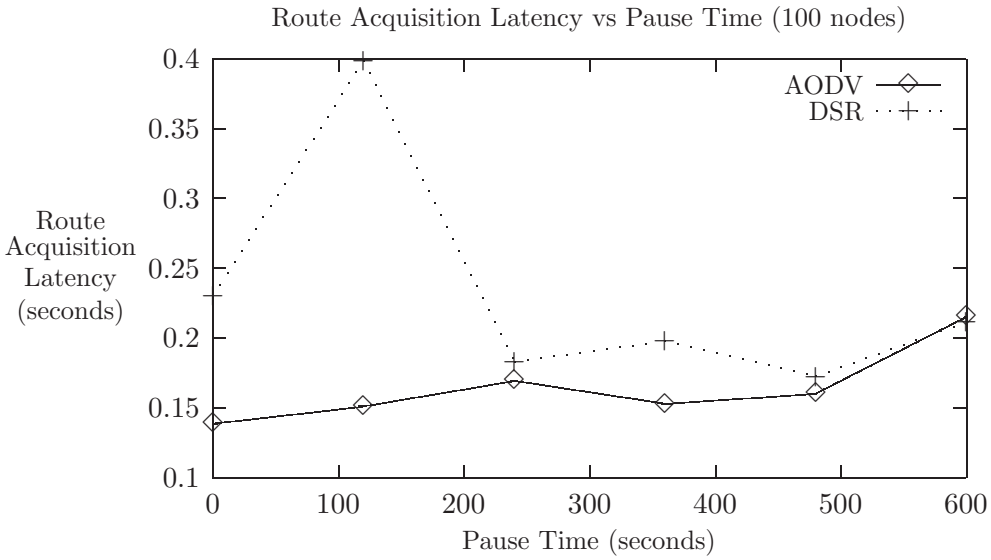

Figure 5.8: Route Acquisition Latency vs Pause Time for 100 Nodes

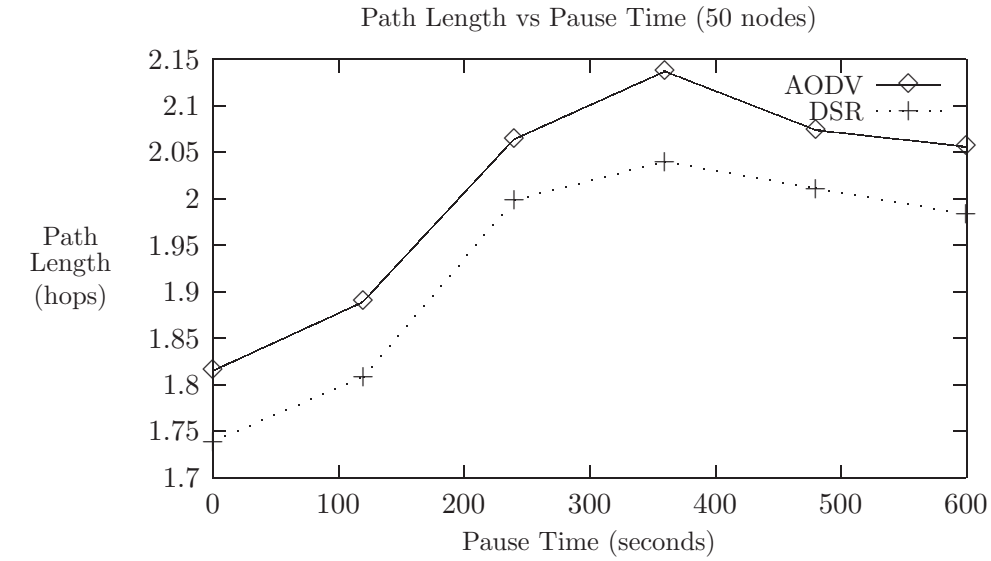

Figure 5.9: Path Length vs Pause Time for 50 Nodes

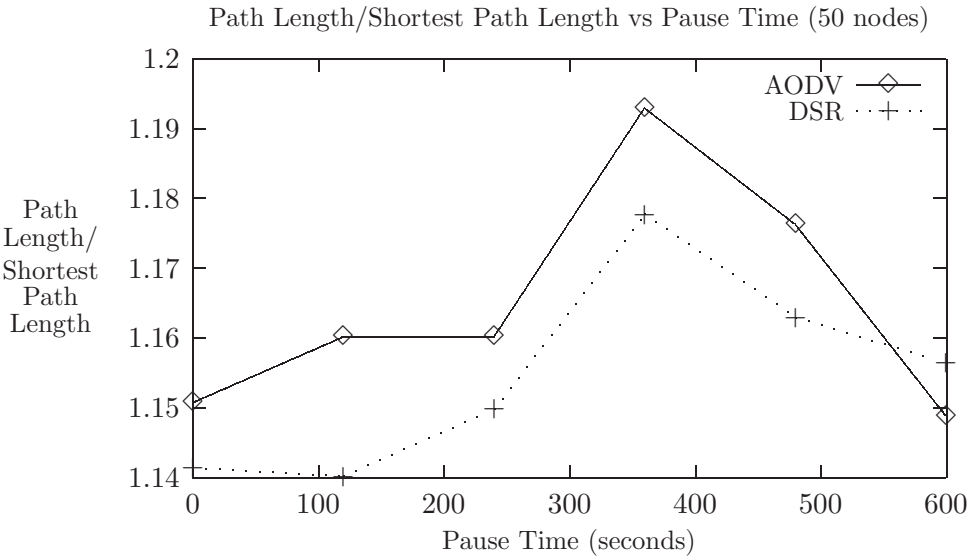

Figure 5.10: Path Length/Shortest Path Length vs Pause Time for 50 Nodes

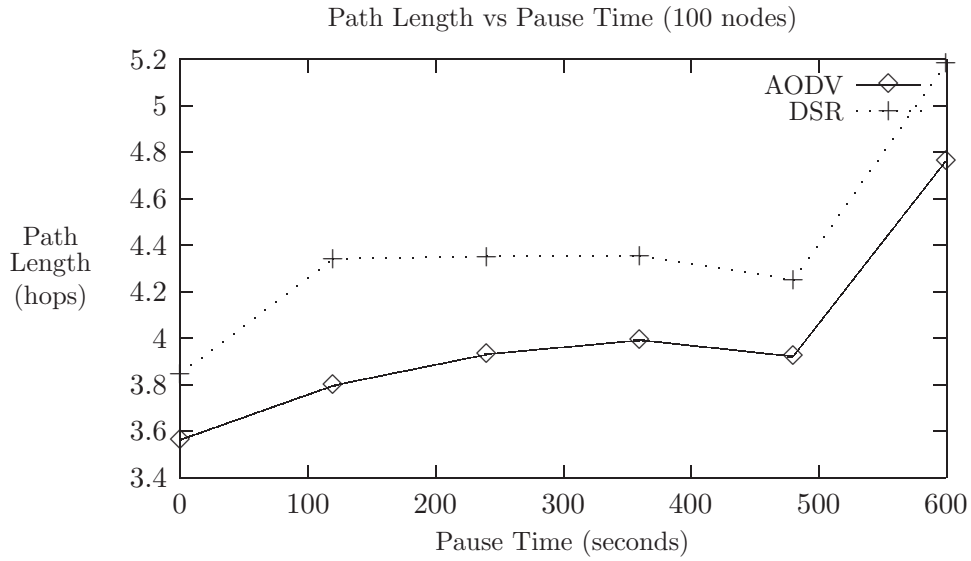

Figure 5.11: Path Length vs Pause Time for 100 Nodes

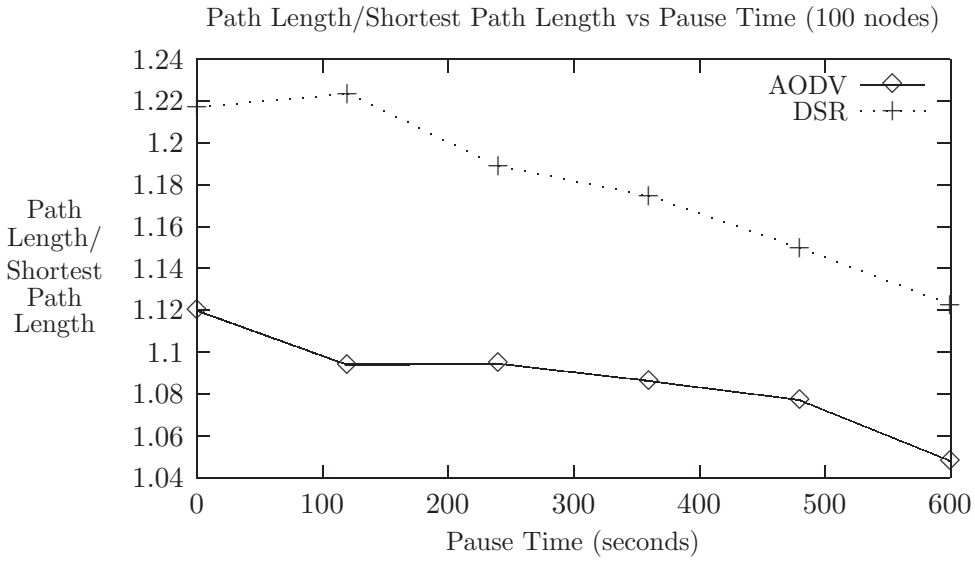

Figure 5.12: Path Length/Shortest Path Length vs Pause Time for 100 Nodes

5.4 Varying Speed

The average pause time between node movements is set to 240 seconds. The server redundancy is set to 2. The simulation is run for different values of maximum speed of the nodes. The values of maximum speed considered are 1, 3, 5, 7 and 9 meters/second. The average route acquisition latency, average path length and the ratio average path length/average shortest path length are plotted for the different values of maximum speed and the results using AODV and DSR for 50 and 100 nodes are shown in Figures 5.13, 5.14, 5.15, 5.16, 5.17 and 5.18.

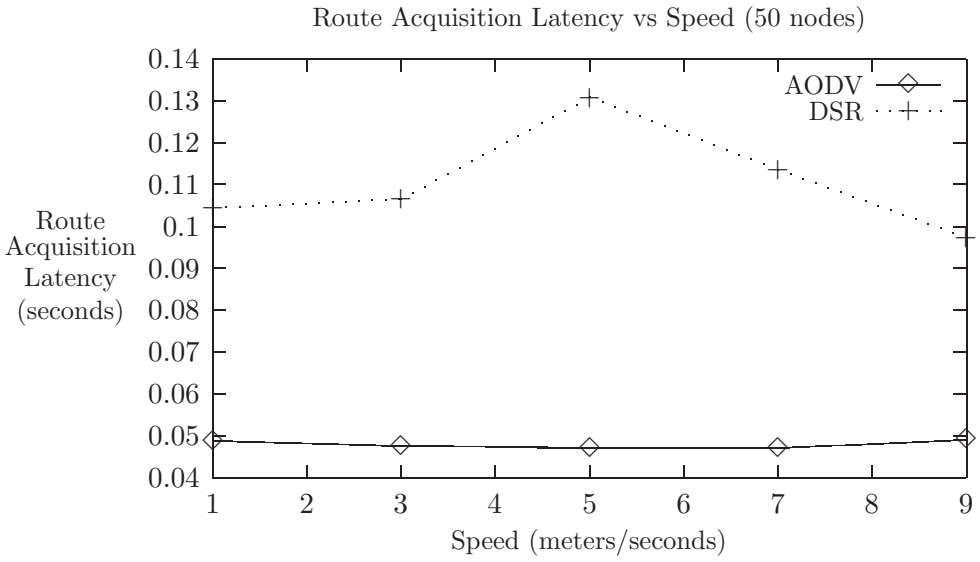

Figure 5.13: Route Acquisition Latency vs Speed for 50 Nodes

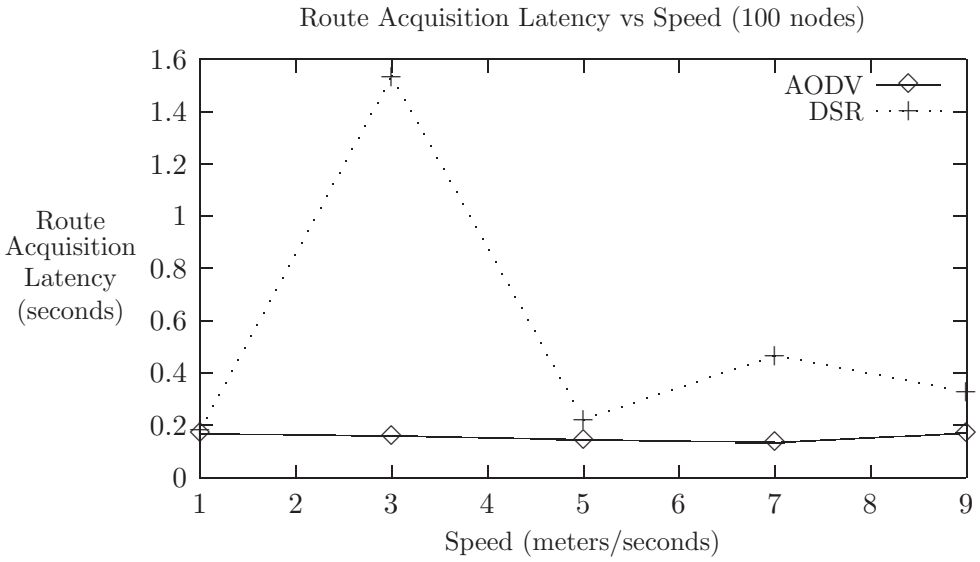

Figure 5.14: Route Acquisition Latency vs Speed for 100 Nodes

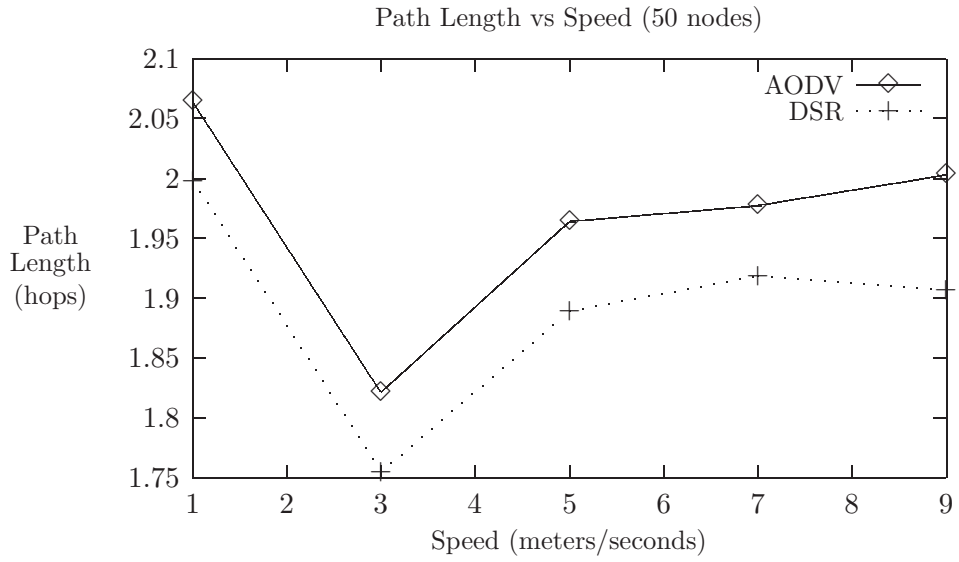

Figure 5.15: Path Length vs Speed for 50 Nodes

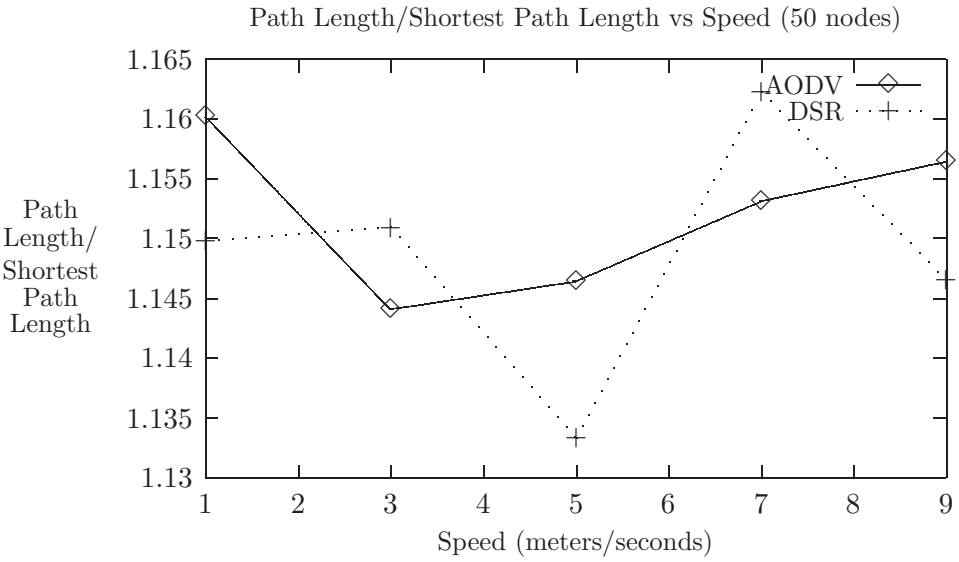

Figure 5.16: Path Length/Shortest Path Length vs Speed for 50 Nodes

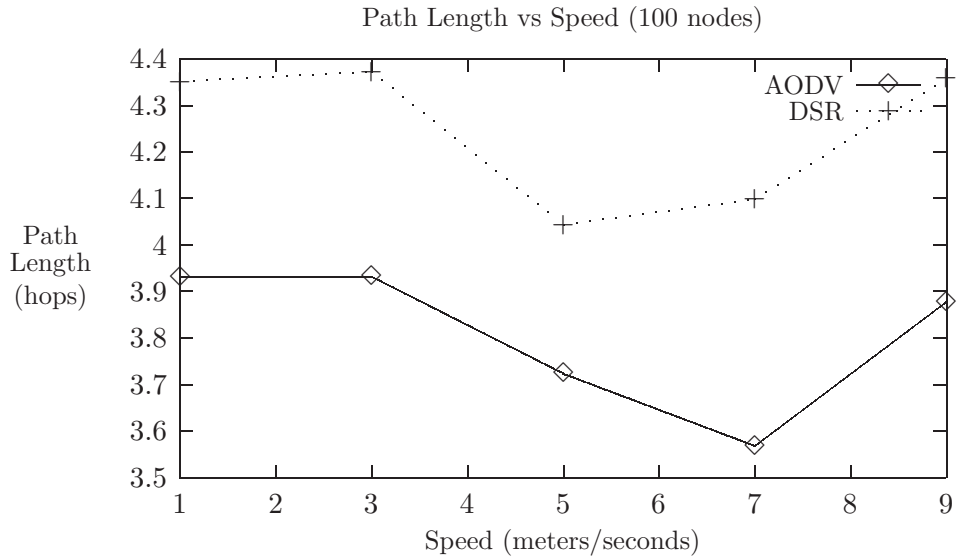

Figure 5.17: Path Length vs Speed for 100 Nodes

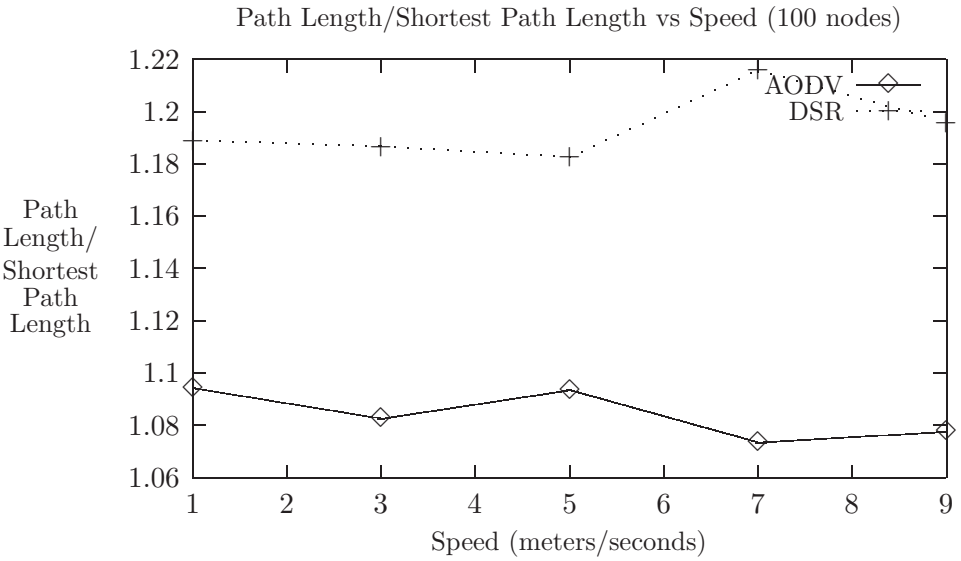

Figure 5.18: Path Length/Shortest Path Length vs Speed for 100 Nodes

5.5 Analysis of Results

For both AODV and DSR, the average route acquisition latency, average path length and the ratio average path length/average shortest path length decreases as the server redundancy increases. This can be explained as follows. When there are more nodes in the network which provide a service, it is likely that a node will take less time to find a server and the server will be closer to the node on average. Hence, this decrease in average route acquisition latency and path length with an increase in server redundancy is expected. However, one must note that the higher the server redundancy, the more SREPs will be generated by service providers and by other nodes replying on their behalf. This is especially true in DSR, where the destination of a request, replies to multiple copies of the same request from the same source. Hence, a higher server redundancy can result in congestion in the network and hence decreased performance. This is the reason we see some increases in route acquisition latency with increase in server redundancy for DSR with 100 nodes. We also observe a decrease in the ratio average path length/average shortest path length with an increase in server redundancy. This can be attributed to the fact that as the server redundancy increases, the path length decreases and a greater percentage of service requests are replied to by the target itself and hence the path length is likely to be more optimal.

We also note that for both AODV and DSR, the average route acquisition latency, average path length and the ratio average path length/average shortest path length do not monotonically increase or decrease with variation in the pause time and speed. We expect route acquisition latency and path length to remain more or less constant with a variation in mobility (change in pause time or speed). This is because we expect to still be able to discover a route to a provider when nodes move faster or more frequently. However, in conditions of high mobility, the discovered route may become invalid after a short period of time and hence subsequent data packets may trigger a route rediscovery. Thus, the number of route requests triggered will increase in conditions of high mobility. Since we do not attempt to exchange data with the discovered provider in our simulations, this situation does not arise. Moreover, we are measuring the route acquisition latency and not the number of route requests triggered.

We do observe some large fluctuations, in particular for DSR with 100 nodes for a pause time of 120 seconds and for a speed of 3 meters/second. We observe that for

these values, there are a number of service discovery failures for some replications of the simulation runs. A service discovery will fail when either the network has partitioned and hence no provider is within range or there is congestion in the network. Hence, when there are a number of service discovery failures, the route acquisition latency and path length of the discoveries which succeed will usually be higher. Another point mentioned in [9] is that in conditions of very low mobility, there tend to be higher delays due to high congestion and interference in certain parts of the network. This is because, in conditions of very low mobility, the traffic is not evenly distributed. With higher node mobility, the load gets distributed more evenly over the whole network.

From our simulation runs, we also notice that the number of service discovery failures is higher when DSR is used as compared to the simulations with AODV, especially in the case of 100 nodes. Also, when DSR is used, a node which requires a service is more likely to have a route to the service provider than if AODV is used and hence may not need to initiate a service discovery. This can be attributed to the aggressive caching in DSR. However, since the DSR implementation in ns-2 that we consider does not actively purge stale routes from its cache, such a cached route is likely to be valid only in conditions of low mobility.

From the graphs, we can conclude that the average route acquisition latency when using AODV is always smaller than that when we use DSR. In terms of average path length and the ratio average path length/average shortest path length, DSR gives better results for 50 nodes, while AODV gives better results for 100 nodes. Overall, we can conclude that integration of service discovery with AODV yields better results than integration with DSR.

In [9], the simulated performance of AODV and DSR in ns-2 is presented and compared. Total data packet delay in the network is compared but route acquisition latency alone is not compared. Due to its aggressive caching, DSR is known to perform better than AODV in networks with small number of nodes, low mobility and light loads. AODV is known to perform better in networks with large number of nodes, high mobility and heavy loads. This is because when nodes are highly mobile, the cached routes in DSR are likely to be invalid, thus causing route errors and rediscovery of routes. Further, DSR is expected to yield shorter path lengths than AODV.

However, we find that overall, the integration of service discovery with AODV performs better than with DSR for the scenarios we have considered. This is because of the inherent differences in the route discovery process and the integration of service discovery

with route discovery. A SREQ may not contain a destination address unlike a RREQ which always contains a destination address. Since the destination is always known in a RREQ, full use can be made of cached routes in DSR. Thus, intermediate nodes can reply to RREQs more often. A SREQ on the other hand, may not contain a destination address and hence a large number of such requests are replied to only by the service provider itself. Thus, the benefits of aggressive caching in DSR are lost to a certain extent in the integrated approach as opposed to pure route discovery.

This concludes our discussion of the simulation results of the integration of service discovery with AODV and DSR. In the next chapter, we summarize our conclusions from this thesis work and present our ideas for future work.

Chapter 6

Conclusions and Future Work

In this chapter, we present our conclusions from this thesis work and proposed future work.

6.1 Conclusions

In this thesis, we studied the design, simulation and performance of an integrated approach to on-demand service and route discovery. We proposed modifications and additions to the Internet-Draft specification for this integrated approach. We then modified the existing simulation of the integrated approach using AODV so as to correct the bugs that we identified and also implemented the modifications to the draft that we proposed. We also simulated the integrated approach along with our modifications to the draft using DSR as the routing protocol. We then discussed the service termination problem and proposed a solution as an extension to the draft.

We compared the performance of the integrated approach using AODV and DSR for different simulation scenarios. We varied the degree of mobility, number of nodes and server redundancy. We note that the values of route acquisition latency and path length obtained for both AODV and DSR for the various scenarios considered are reasonable, thus confirming that the integrated approach is a viable method of service and route discovery. We find that the values of route acquisition latency and path length decrease with an increase

in server redundancy and do not monotonically increase or decrease with varying pause time and speed of the mobile nodes. The results also indicate a better route acquisition latency when using AODV as compared to DSR for all the scenarios considered. For 50 nodes, the path length is better when using DSR, while for 100 nodes, AODV gives better results for path length.

Also, the number of failed service discoveries is higher when using DSR, especially for 100 nodes. However, we observe that when using DSR, more service discoveries can be satisfied by the initiating node itself than in the case of AODV. This is due to the greater likelihood of there being a route to the service provider on the initiating node itself as a result of the aggressive caching techniques of DSR. However, such a cached route may not be valid especially in conditions of high mobility since the DSR implementation in ns-2 that we consider does not actively purge stale entries from the cache. We conclude that overall, the integration of service discovery with AODV performs better than the integration with DSR.

6.2 Future Work

We can run our simulations with different values of certain parameters such as the service binding lifetime and the expiration time of the service timer. Also, we can implement our proposed extension for solving the service termination problem. We can then run our simulations with service terminations and observe the performance under these conditions.

We also propose storing multiple bindings for each service type in the service table and observing whether this affects the performance of the integrated approach. If multiple bindings are stored, we need to specify a rule for selecting a binding for use from among these multiple bindings.

We can also experiment with using an expanding ring search in DSR, so as to improve the performance of service discovery with DSR. We also propose implementing the purging of stale cache entries in DSR, and observing the effect this has on the performance of service discovery.

We observed some sharp fluctuations in our graphs which lead us to believe that 10 replications of our simulation runs may not be sufficient. We plan to increase the number of replications to 30 and compute the confidence intervals $(\hat{X} - (\sigma/\sqrt{n})\epsilon, \hat{X} + (\sigma/\sqrt{n})\epsilon)$,

where $\hat{X}$ is the sample mean, σ is the standard deviation and n is the number of samples (replications). For 95% confidence intervals, $\epsilon = 1.96$. We also realise that in addition to varying the random number sub-streams used for service request generation, we should also vary the random seed used for node movement and data traffic generation in multiple replications of the same simulation run.

In [48], Yoon, et al. point out that in the random waypoint mobility model, the average node speed decreases with time. This is because in this model, the destination and speed of each node are chosen randomly and independently. It is possible that the destination chosen may be far away and the speed may be low. Hence, the node will take a long time to reach this destination. Over a period of time, there may be a number of such slow moving nodes in the network, thus lowering the average node speed. One of the solutions proposed is selecting a non-zero minimum speed. Another solution is to select a high speed when the destination is far away and select a low speed when the destination is near. For future work, we propose implementing one of these solutions or using a different mobility model.

We would also like to compare the performance of the integrated approach to service and route discovery with the conventional approach in which service discovery is done at the application layer and route discovery is done at the network layer. For this purpose, we need to implement an application layer service discovery protocol in ns-2 or obtain an existing simulation of such a protocol.

On-demand routing protocols like DSR and AODV use broadcast of request messages. In a wireless network, broadcast involves a lot of overhead which causes scalability problems for large networks. This is known as the broadcast storm problem. Several solutions to this problem have been proposed in [42], [43], [40] and [28]. Future work could involve using these solutions to reduce the broadcast overhead in service discovery using DSR and AODV.

The service discovery process studied in this thesis could be thought of as a type of anycasting. The service port/string is the anycast group name and nodes resolve this group name to the IP address of a specific service provider. Other schemes for anycast in ad hoc networks can be explored. Also, algorithms for distributed networks can be studied to see whether they can be adapted for service discovery in ad hoc networks.

Service discovery can be integrated with a proactive routing protocol. These protocols tend to consume a lot of bandwidth due to periodic topology update messages. Hence,

the protocol chosen should be one that makes efficient use of bandwidth like Optimized Link State Routing (OLSR) or STAR. When using a proactive routing protocol, the route acquisition latency will be low since no route discovery will be required. When deciding whether to use a proactive or reactive protocol for service and route discovery one needs to consider the tradeoff between bandwidth utilization and route acquisition latency.

Appendix A

Routing Protocol Simulations in ns-2

A.1 DSR Simulation in ns-2

Before integrating service discovery with DSR, we needed to study the simulation of DSR in a network simulator. We decided to use ns-2 [14], [30] for simulation purposes. The simulation details obtained by studying the DSR code in ns-2 are given below.

A.1.1 Summary of Parameters

- ***Send Buffer***

  The send buffer can hold 64 packets.

  The send buffer is checked every 30 milliseconds.

  Packets timeout in the send buffer after 30 seconds.

  Once a packet is in the send buffer it is not piggybacked on future RREQs. It is removed from the send buffer if a route reply provides a route for it.

- ***RREQ***

  Delay RREQ broadcasts by 10 milliseconds.

Do a ring zero search first (1 hop only).

The backoff period is 0.5 seconds.

The maximum time between RREQs is 10 seconds.

- ***RREP***

  Up to 10 route replies can be held off (when replying from cache).

  RREP hold off period is 3 seconds.

  Time we wait before sending a reply from our cache is $U(0.0, rt\_rep\_holdoff\_period) + (our\ route\ length - 1) * rt\_rep\_holdoff\_period$.

  The target will reply to repeated RREQs from the same source.

  An intermediate node that has a route in its cache can send a RREP.

  The RREP is sent along the reversed path in RREQ. This way only bidirectional routes will reach the source.

- ***Route Error***

  The maximum number of errors in one packet is 3.

- ***Gratuitous RREP***

  This is an automatic route shortening RREP sent to the source of the packet.

  The size of the automatic route shortening table is 5 entries.

  Hold down time i.e. minimum time between gratuitous replies for the same route is 1 second.

- ***Gratuitous Route Error***

  A route error we receive is added on to our next route request.

  Maximum hold down time i.e. maximum time from the reception of a route error after which the error is considered to be stale and should not be piggybacked on a RREQ is 1 second.

- ***Flow State***

  Size of flow table is 3000 entries.

  Default flows have an odd flow id.

  Non default flows have an even flow id.

  Order of preference is shorter route over default flow over non default flow.

  Flows can be advertised again after a minimum of 5 seconds.

Default flows timeout after 60 seconds.

Packet transmission over a flow reestablishes the flow (timeout is reset).

- ***Promiscuous Reception of Packets***

  Up to 1024 uids can be stored in the tap cache.

  Bit mask for tap = 1023 = 11 1111 1111.

- ***Salvaging Packets***

  A packet can be salvaged up to 15 times.

  A packet can be salvaged with a route from the cache.

  A node can send only 1 RREQ for a salvaged packet originated at another node. The RREQ cannot be propagating.

  If the packet is a RREP we check if the route in it has the dead link. If it does, we drop the RREP and salvage the rest of the packet only.

- ***Updating the Cache***

  When we forward a route error or a packet with a source route we will give the error/route information to the cache.

- ***Request Table***

  This has 128 entries (default size is 30 entries).

- ***Miscellaneous***

  Requires bidirectional source routes.

  Arp request timeout is 30 milliseconds.

  Maximum length of the source route is 16 addresses.

  For a gratuitous route error the tell address is set to GRAT_ROUTE_ERROR.

A.1.2 Simulation Details

- ***Packet Formats***

  The DSR header contains the source route, its length and the current pointer within the source route. There is also a flag to indicate whether the packet has been salvaged.

  RREQ - This includes a request id and a ttl.

  RREP - This includes the reply route and route length.

RERR - This includes the number of errors and for each error, the address of the node to be notified of the error and the addresses of both ends of the link which is down.

Flow header - This includes the flow id and hop count.

Flow timeout - This includes the time after which the flow will timeout.

Unknown flow - This includes the number of flow errors and for each error, the source and destination address and flow id of the flow.

Unknown default flow - This includes the number of flow errors and for each error, the source and destination address of the flow. (Flow id is not valid for default flow).

The NS packet is placed inside a SRPacket. The SRPacket also includes the source and destination address and the source route.

- ***Request Table***

  Each entry has the source address, last request id received, number of outstanding requests, time of last request, type of last request (ring zero search or network wide) and time of last arp (last ring zero search). The default size of the table is 30 entries. The table has a pointer to the first free entry. Entries are never removed from the request table.

- ***Route Cache***

  This includes the mac and network address. Route/Link add, notice and find counts are maintained. There is a handler which collects these statistics every 1 second.

- ***Automatic Route Shortening (ARS) Table***

  This table can store up to 5 entries. Each entry has the flow id, the number of hops early and a uid. The table also has the index of an entry which can be replaced (victim) i.e. the entry with hops early=0.

- ***Default Route Table***

  This table can store up to 3000 entries. This has the source and destination address and flow id.

- ***Flow Table***

  This table can store up to 3000 entries. Each entry has the source and destination

addresses, flow id, timeout, hop count, expected ttl, flag to indicate if this flow can be used as default, time when this flow was last advertised (by a source routed packet), count to indicate whether the flow has been established end to end, next hop and source route (node can't do automatic route shortening without source route). The flow table also has the next available flow id which can be assigned (flow ids start from 0) and the node's network address. It also contains a default route table. Currently, expired entries are not being removed from the flow table.

- ***Promiscuous Reception of Packets***

  Uids received are stored in a tap cache which can hold up to 1024 entries. The tap cache bit mask is 1023. We snoop on packets with source routes.

- ***Send Buffer***

  Each entry in the send buffer contains a SRPacket and the time it was inserted into the buffer. Up to 64 packets can be stored in the send buffer. Once a packet is in the send buffer it can't be piggybacked on a route request. The send buffer is checked every 30 milliseconds by the send buffer timer handler. Packets timeout in the send buffer after 30 seconds. Whenever the timer expires, if a packet has expired, it is dropped from the send buffer, otherwise we try to get a route for the packet.

- ***RREQ***

  RREQs which are broadcast are delayed by 10 milliseconds. The backoff period is 0.5 seconds and the maximum time between RREQs is 10 seconds. The arp timeout is 30 milliseconds. A ring zero search is done first. The time till the next request is proportional to the number of outstanding requests, but cannot exceed the maximum request timeout. A received RREQ is ignored if it is a duplicate, has traversed a loop, or its source route length is greater than the maximum allowed.

- ***RREP***

  Up to 10 route replies can be held off (when replying from cache or gratuitous route replies). The route to be followed by the RREP is generated by reversing the route in the RREQ. This way only bidirectional routes are returned. Nodes can reply from their cache. Nodes can reply to more than one RREQ from the same source.

- ***RERR***

  We snoop on errors which reach us and notify our cache.

- ***Gratuitous RERR***

  A gratuitous route error is a received route error which is propagated on the next RREQ which is broadcast. The maximum time from the reception of a route error after which the error is considered to be stale and should not be propagated on a RREQ is 1 second.

- ***Held off RREP (when replying from cache)***

  Each entry contains the requester, requested destination, the best length and our length.

- ***Held off Gratuitous RREP***

  Each entry contains the path and the time. The minimum time between gratuitous replies for the same route is 1 second.

- ***Salvaging a Packet***

  A packet can be salvaged up to 15 times. Only 1 RREQ (has to be non propagating) can be sent for a salvaged packet that was originated at some other node. If a packet could not be transmitted and it is a route reply we check to see if it contains the bad link and if it does, we salvage only the rest of the packet.

- ***Flow State***

  Shorter routes are preferred over default flows and default flows are preferred over non default flows. The minimum advertisement interval is 5 seconds. Default flows timeout after 60 seconds. A packet with a source route reestablishes the flow (resets the timeout value).

- ***DSR Agent***

  The DSR agent is the routing agent that runs on each node. It contains the network and mac addresses of the node, a tap cache, a request table, a route cache, a send buffer, a send buffer timer, a flow table, an ARS table, the held off route replies when replying from cache and the held off gratuitous route replies.

  It also includes the route request number to be used, the number of held off replies, the gratuitous reply hold victim (entry in gratuitous route reply list which can be replaced), a flag to indicate whether there is a route error held, and if there is one, the time the error was reported and the node IDs between which there is an error.

A DSR agent also has the link layer output, mac layer, the output interface queue and a trace object. There are some other members for wired cum wireless simulations and mobile IP.

The request table in the DSR agent has 128 entries. The request ids start from 1. When a broadcast packet is originated, it is sent without any delay.

- ***Packet Reception***

  If the packet doesn't have a source route in it, is not a broadcast packet and is not for us, check the cache for a route. If there is no route try to send out a RREQ. Check whether we are allowed to send a RREQ. If we are allowed but we are not the source of the packet and the number of outstanding RREQs > maximum salvage requests, then we don't send a RREQ. However, even if we are not allowed to send a RREQ, but we are the source and the previous request was a ring zero search which timed out, we send a RREQ.

  If the packet is for us and it is a RREP, add the route to cache. Clear the number of outstanding requests in the entry for the request target in the request table. Forward any packets in the send buffer which can be sent using the received route.

  When the packet is for us and it is a RREQ, if we must ignore the RREQ, drop it. Otherwise, add our address to the end of the route in the packet, reverse the route and send a RREP using this route with a delay of up to 10 milliseconds. Add the route to the source to our cache.

  When the packet has a source route, is a RREQ and we are not the target, if we must ignore the RREQ, drop it. Otherwise see if we can reply from cache. If we can reply from cache, make sure that any piggybacked data/RREP/non gratuitous RERR is forwarded to the destination as if it came from the source. Add our address to the end of the route in the RREQ, reverse the route and send a RREP using this route. Add the route to the source to our cache.

  If we can't reply from cache, we need to forward the RREQ. If the queue length > 10 or free airtime < 15% or the RREQ has exceeded its propagation limit, drop the RREQ. Otherwise, add our address to the source route and forward the RREQ.

- ***Packet Transmission***

  When we are originating a packet which isn't a RREQ/RREP/RERR, if the route

from the default flow is shorter than our route, use the default flow route. If our route is not in the flow table or it has timed out, add it to the flow table. If the flow is not established end to end or not advertised recently, use flow timeout in the header, otherwise, don't use flow timeout. If the ttl matches the expected value, use default flow (no flow header), otherwise use non default flow (use flow header).

When we are sending out a RREQ, if we have non stale error data, add this to the RREQ to form a gratuitous RERR.

When we receive a packet for forwarding and there is a flow header, find the entry in the flow table for the flow id. If an entry could not be found in the flow table and there is no source route, send an unknown flow error. If an entry could not be found in the flow table and there is a source route, add an entry to the flow table. If an entry could be found in the flow table, set the next hop from the flow table. Decrement the ttl and if the ttl is 0, drop the packet. If the packet is not salvaged and was heard previously i.e. is in the ARS table, send a gratuitous route reply with the source route from the flow table. Refresh the timeout for the flow table entry and schedule the packet without delay.

When we receive a packet for forwarding and there is no flow header and no source route, if there is no entry in default flow table, send an unknown default flow error. If there is an entry in the default flow table but no entry in the flow table, send an unknown flow error. Otherwise, if the ttl is not equal to the expected ttl, send an unknown flow error. Otherwise, handle the packet as for a non default flow.

When we receive a packet for forwarding and there is no flow header and there is a source route, if we have reached the end of the source route, drop the packet. Otherwise, add the route to the cache and decrement the ttl. If the ttl is 0, drop the packet. Otherwise, forward the packet.

When we are forwarding a packet, if it is a RREQ, set the next hop to ip_broadcast, otherwise set the next hop from the source route. If it is a RREQ, jitter is 10 milliseconds, otherwise there is no jitter.

A.2 AODV Simulation in ns-2

Before integrating service discovery with AODV, we needed to study the simulation of AODV in a network simulator. We decided to use ns-2 for simulation purposes. The simulation details obtained by studying the AODV code in ns-2 are given below.

A.2.1 Summary of Parameters

- ***Route Queue***

  Maximum size of queue is 64 packets.

  Timeout of a packet in the queue is 30 seconds.

- ***RREQ***

  The reverse path setup by a RREQ times out after 6 seconds.

  The expanding ring search starts at 5 hops and increases by 2 hops up to a maximum of 7 hops.

  A RREQ is retransmitted over the whole network 3 times and then times out for 10 seconds.

  The broadcast id from a received RREQ is stored for 6 seconds.

- ***Route Table***

  Entries in the route table timeout after 10 seconds.

- ***Route Errors***

  A route error packet can hold up to 100 errors.

- ***Route Maintenance***

  Intermediate nodes are allowed to repair routes.

  Link layer (802.11) feedback is used.

  Time an intermediate node waits for local route repair to complete is 150 milliseconds.

- ***Local Connectivity Management***

  The hello interval is 1 second.

  The maximum hello interval is $1.25 * hello\_interval$.

  The minimum hello interval is $0.75 * hello\_interval$.

The allowed hello loss is 3 packets.

The lifetime of a bad link is 3 seconds ($hello\_interval * allowed\_hello\_loss$).

- ***Miscellaneous***

  Network diameter is assumed to be 30 hops (when a RREQ is broadcast over the whole network, this value is used).

  Time to traverse a node is assumed to be 30 milliseconds (if there is no route or no history then this is taken to be the per hop round trip discovery latency which is used in calculating the RREQ timeout value).

  The arp delay is 10 milliseconds (used to delay transmission of successive packets from the route queue).

  All route messages are sent to port 255.

  The random delay in forwarding of broadcast packets is up to 10 milliseconds.

A.2.2 Simulation Details

- ***Route Queue***

  The route queue is used for storing packets which are waiting for a route to the destination. This is a drop front queue which can hold up to 64 packets. Packets in this queue timeout after 30 seconds.

- ***Route Table***

  Each entry in the routing table has the destination address, sequence number, next hop, number of hops, expiry time and status of route (up/down/in repair). Each entry also includes the list of precursors for the route and the list of neighbors using the route. It also includes the number of network wide route requests originated for this destination, the time when the next request can be sent and the last ttl value used. The route discovery per hop latency values for up to 3 past requests are also stored. When a local route repair is initiated, the route status is set to "in repair". When the route status is set to "down", the number of hops is set to infinity. (The last value of the number of hops is saved before doing this.)

- ***Packet Formats***

  Route Request (RREQ) - It consists of the source address and sequence number,

destination address and sequence number, broadcast id, hop count and the time the RREQ was sent.

Route Reply (RREP) - It consists of the reply route's destination address and sequence number, source address, hop count, lifetime of route and time when the request was sent.

Route Error (RERR) - It consists of the number of errors, and for each error, the unreachable destination address and sequence number.

- ***Timers***

  There are 5 timers.

  The broadcast timer removes all entries in the broadcast id cache which have expired every 6 seconds.

  The hello timer sends hello packets every randomly generated time interval between minimum (0.75 seconds) and maximum hello interval (1.25 seconds).

  The neighbor timer is supposed to remove all neighbor entries which have expired every hello interval (1 second) but currently it doesn't do this (the code has been commented out).

  The route cache timer checks the route table for expired entries every 0.5 seconds. If an entry is up and has expired, the sequence number is incremented (will now be odd), the route status is set to down and waiting packets are dropped. If a route is up but has not expired, waiting packets are forwarded with the delay incremented by arp_delay (10 milliseconds) for each packet. If the route is down and there are waiting packets, a RREQ is sent. (The logic may be that if the route is down, it may be a long time since it expired, so we need to send a RREQ.)

  The local repair timer frees the packet which was waiting for local route repair. This is done when the route request sent by the intermediate node to repair the route times out.

- ***Broadcast ID Cache***

  Each entry in this cache includes a source address, broadcast id and expiry time.

- ***AODV Agent***

  This is the routing agent that runs on each node. Each node has an IP address, a

sequence number (starts off at 2) and a broadcast id (starts off at 1). It also has a routing table, a broadcast id cache and a neighbor cache. It includes the 5 timers, a route queue and an interface queue. Odd sequence numbers are used to indicate an error. When the node adds a new neighbor or removes an existing neighbor from its neighbor cache, it increments its sequence number by 2.

- ***Link Failure and Local Route Repair***

  If a link fails and the packet to be transmitted is broadcast or non data or there is no route to the destination, the packet is dropped. Otherwise, if the packet is closer to the source than to the destination, the packet and all other packets using the broken link are dropped and the neighbor is removed from the neighbor cache. Also all entries in the routing table for which this link is the next hop are brought down and a route error packet is sent without delay. Otherwise, if the packet is closer to the destination than to the source, local repair is done by putting the packet in the route queue, setting the route status to "in repair", sending a RREQ and scheduling the local repair timer.

- ***Packet Reception***

  If the received packet is not an AODV packet and we are the source, resolve a route for the packet if it hasn't already been forwarded. Otherwise, if the packet has been forwarded, drop the packet (since it has traversed a loop). If the packet is not an AODV packet and we are not the source, decrement the ttl and resolve a route for the packet if ttl is non zero. Otherwise, if ttl is zero, drop the packet. If the received packet is an AODV packet, decrement the ttl.

  Route Resolution - If a node receives a packet and there is a valid route to the destination, the packet is forwarded with no delay. If there is no route but the route is being repaired, the packet is buffered. If there is no route and the node is the source of the packet, the packet is buffered and a RREQ is sent. Otherwise, a route error is sent without delay and the packet is dropped.

  Received Packet is a RREQ - If this node is the source of the RREQ (the packet has traversed a loop) or the source address, broadcast id pair are in the broadcast cache (this is a duplicate RREQ), drop the RREQ. Otherwise, add the source address and broadcast id to the broadcast id cache.

If there is no route to the source of the RREQ, add the reverse route with expiry time set to the reverse route lifetime of 6 seconds. If there is a route but the RREQ route is better, update the route. Forward any packets for the source without delay.

If this node is the destination, increment its sequence number and send a RREP with lifetime set to my_route_lifetime (10 seconds) and hop count of 1. Otherwise, if this node has a better route to the destination than the RREQ source, send a RREP with lifetime from the route and hop count from the route + 1. Otherwise, change the source IP address to this node's address, increment the request hop count and forward the RREQ as a broadcast packet with a delay of 1 second.

Received Packet is a RREP - If the routing table doesn't have a route to the reply route's destination, add the route or if the reply route is better, update the route and reset the number of requests and request timeout. If the reply is for this node's request, calculate the discovery latency. Forward all packets for the reply route's destination with successive delays incremented by the arp_delay (10 milliseconds).

If the reply is for this node's request or the reply route is not better, drop the RREP (it need not be forwarded). If the reply is not for this node's request and there is no route to the destination of the RREP, drop the RREP. Otherwise, increment the reply hop count, change the source address to this node's address and forward the RREP without any delay.

Received Packet is a RERR - Iterate through the errors in the RERR. If there is a route for the unreachable destination and its sequence number is less than or equal to that in the RERR and the next hop is the source of the RERR, set the route status to down and update its sequence number. Drop all packets for the source of the route error. If this route has precursors, then add this unreachable destination to a route error packet and forward that as a broadcast packet with a random delay of up to 10 milliseconds.

Received Packet is a Hello Packet - If the reply route's destination is not in the neighbor cache, add it, otherwise update its expiry time based on the allowed hello loss and hello interval.

- ***Packet Transmission***

  Forwarding a Packet - If the ttl is 0, drop the packet. If there is a route, set the

route expiry time to active_route_timeout (10 seconds) and the common header's next hop from the route (address type is ns_af_inet). If there is no route, set the next hop address type to ns_af_none. If it is a broadcast packet, send it with a random delay of up to 10 milliseconds, otherwise use the specified delay. If no delay is specified, send the packet without any delay.

Packet to be Transmitted is a RREQ - If there is a valid route to the destination or the previous request hasn't timed out, don't send the RREQ. If the number of network wide requests is greater than the maximum allowed, set the request timeout to its maximum value (10 seconds), reset the request count, drop all packets for the destination and don't send a RREQ.

If the last ttl was 0, set the ttl to the start value (5 hops), otherwise if it was below the threshold (7 hops), add the allowed increment (2 hops) to it. Otherwise, set the ttl to the network diameter (30 hops) and increment the request count. Save the ttl value used. Set the request timeout based on the ttl and the per hop time. If we have done network wide broadcast before, set the timeout to a higher value, making sure that it isn't more than its maximum value. Increment the node's broadcast id and sequence number. Set the request hop count to 1. Set the destination sequence number to its value from the route if there is one, otherwise set it to 0. Send the RREQ without any delay.

Packet to be Transmitted is a RREP - Check that there is a route to the destination of the RREP and set the next hop (in the common header) to the next hop from the route. Set the ttl to the network diameter. The packet is sent without any delay.

Packet to be Transmitted is a RERR - The packet is broadcast, the next hop is set to 0 and ttl is set to 1. The packet is sent with a random delay of up to 10 milliseconds if required, otherwise it is sent without any delay.

Packet to be Transmitted is a Hello Packet - A hello packet is basically a route reply packet with the reply route's destination as the address of the node sending the hello and destination sequence number as this node's sequence number. The ttl is set to 1 and hop count is also set to 1. The reply route's expiry time is set based on the allowed hello loss and hello interval. The packet is broadcast without any delay.

Bibliography

[1] Hal Abelson, Philip Greenspun, and Lydia Sandon. Tcl for web nerds. `http://philip.greenspun.com/tcl`.

[2] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP 1999), Kiawah Island, SC*, pages 186–201, December 1999.

[3] Jeff Boleng. Efficient network layer addressing for mobile ad hoc networks. In *Proceedings of International Conference on Wireless Networks (ICWN'02), Las Vegas, USA*, pages 271–277, June 2002.

[4] Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, and Tim Finin. GSD: A novel group-based service discovery protocol for manets. In *4th IEEE Conference on Mobile and Wireless Communications Networks (MWCN 2002)*, pages 140–144, September 2002.

[5] Liang Cheng. Service advertisement and discovery in mobile ad hoc networks. In *Workshop on Ad hoc Communications and Collaboration in Ubiquitous Computing Environments, in conjunction with the ACM 2002 Conference on Computer Supported Cooperative Work, New Orleans, Louisiana, USA*, November 2002.

[6] Salutation Consortium. Salutation architecture specification, version 2.1, 1999.

[7] G. H. Cooper. An argument for soft layering of protocols. Technical Report MIT/LCS/TR-300, MIT, 1983.

[8] Microsoft Corporation. Universal plug and play device architecture, version 1.0, June 2000.

[9] Samir R. Das, Charles E. Perkins, Elizabeth M. Royer, and Mahesh K. Marina. Performance comparison of two on-demand routing protocols for ad hoc networks. *IEEE Personal Communications*, 8(1):16–28, February 2001.

[10] Diego Doval and Donal O'Mahony. Nom: Resource location and discovery for ad hoc mobile networks. In *Proceedings of The First Annual Mediterranean Ad Hoc Networking Workshop, Med-hoc-Net 2002, Sardegna, Italy*, September 2002.

[11] J.J. Garcia-Luna-Aceves and M. Spohn. Source-tree routing in wireless networks. In *Proceedings of IEEE 7th International Conference on Network Protocols (ICNP 1999), Toronto, Canada*, October 1999.

[12] Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright. Simple service discovery protocol/1.0, Internet Engineering Task Force (IETF) Internet-Draft, draft-cai-ssdp-v1-03.txt, work in progress, October 1999.

[13] A. J. Goldsmith and S. B. Wicker. Design challenges for energy-constrained ad hoc wireless networks. *IEEE Wireless Communications*, 9(4):8–27, August 2002.

[14] Marc Greis. Tutorial for the network simulator ns. `http://www.isi.edu/nsnam/ns/tutorial`.

[15] Guillermo E. Guichal. Service location architectures for mobile ad hoc networks. Master's thesis, Georgia Institute of Technology, June 2001.

[16] Sandeep Gupta. Implementation of service discovery in mobile ad-hoc networks in ns-2.26 (with aodv as routing protocol). M. Tech Protocol Implementation Course Report, May 2003.

[17] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2, RFC 2608, June 1999.

[18] Z.J. Haas and M.R. Pearlman. The zone routing protocol (ZRP) for ad hoc networks, Internet Engineering Task Force (IETF) Internet-Draft, draft-ietf-manet-zone-zrp-02.txt, work in progress, June 1999.

[19] Yih-Chun Hu, David B. Johnson, and David A. Maltz. Flow state in the dynamic source routing protocol for mobile ad hoc networks, Internet Engineering Task Force (IETF) Internet-Draft, draft-ietf-manet-dsrflow-00.txt, work in progress, February 2001.

[20] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, volume 353, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[21] Rajeev Koodli and Charles E. Perkins. Service discovery in on-demand ad hoc networks, Internet Engineering Task Force (IETF) Internet-Draft, draft-koodli-manet-servicediscovery-00.txt, work in progress, October 2002.

[22] Hend Koubaa and Eric Fleury. A fully distributed mediator based service location protocol in ad hoc networks. In *Proceedings of Global Telecommunications Conference (GLOBECOM 2001)*, volume 5, pages 2949–2953, November 2001.

[23] Robert Lafore. *Object-Oriented Programming in Microsoft C++*. The Waite Group, 1998.

[24] Leslie Lamport. *LaTeX: A Document Preparation System*. Pearson Education, 1994.

[25] Jiangchuan Liu, Qian Zhang, Wenwu Zhu, Jun Zhang, and Bo Li. A novel framework for qos-aware resource discovery in mobile ad hoc networks. In *IEEE International Conference on Communications (ICC 2002), New York, USA*, volume 2, pages 1011–1016, April 2002.

[26] A.J. Mcauley and K. Manousakis. Self-configuring networks. In *Proceedings of IEEE Military Communications Conference (MILCOM '00)*, pages 315–319, October 2000.

[27] Sun Microsystems. Jini architecture specification, version 1.2, December 2001.

[28] Wuchi Feng Min-Te Sun and Ten-Hwang Lai. Location aided broadcast in wireless ad hoc networks. In *Proceedings of IEEE GLOBECOM 2001, San Antonio, TX*, pages 2842–2846, November 2001.

[29] Sanket Nesargi and Ravi Prakash. MANETconf: Configuration of hosts in a mobile ad hoc network. In *Proceedings of IEEE INFOCOM '02*, pages 1059–1068, August 2002.

[30] The ns manual (formerly ns notes and documentation), the VINT project. `http://www.isi.edu/nsnam/ns/ns-documentation.html`.

[31] Otcl tutorial, version 0.96. `ftp://ftp.tns.lcs.mit.edu/pub/otcl/doc/tutorial.html`, September 1995.

[32] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of IEEE INFOCOM 1997, Kobe, Japan*, pages 1405–1413, April 1997.

[33] Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, September 1994.

[34] Charles E. Perkins. Service location protocol for mobile users. In *The Ninth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, volume 1, pages 141–146, September 1998.

[35] Charles E. Perkins, editor. *Ad hoc Networking*. Addison-Wesley, December 2000.

[36] Charles E. Perkins, Jari T. Malinen, Ryuji Wakikawa, Elizabeth M. Belding-Royer, and Yuan Sun. IP address autoconfiguration for ad hoc networks, Internet Engineering Task Force (IETF) Internet-Draft, draft-ietf-manet-autoconf-01.txt, work in progress, November 2001.

[37] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 1999), New Orleans, LA*, pages 90–100, February 1999.

[38] Stephan Preuß. JESA service discovery protocol. In *Proceedings of Networking 2002, Pisa, Italy*, volume 2345 of *LNCS*, pages 1196–1201. Springer-Verlag, May 2002.

[39] Bluetooth Special Interest Group (SIG). Specification of the bluetooth system - core, November 1999.

[40] Min-Te Sun and Ten-Hwang Lai. Computing optimal local cover set for broadcast in ad hoc networks. In *Proceedings of IEEE ICC 2002, New York, NY*, pages 3291–3295, April 2002.

[41] C.-K. Toh. *Ad hoc Mobile Wireless Networks : Protocols and Systems*. Prentice Hall PTR, 2002.

[42] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. *ACM WINET Wireless Networks*, 8(2–3):153–167, March 2002.

[43] Yu-Chee Tseng, Sze-Yao Ni, and En-Yu Shih. Adaptive approaches to relieving broadcast storms in a wireless multihop mobile ad hoc network. *IEEE Transactions on Computers*, 52(5):545–557, May 2003.

[44] Nitin Vaidya. Weak duplicate address detection in mobile ad hoc networks. In *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 206–216, June 2002.

[45] Kilian Weniger and Martina Zitterbart. IPv6 autoconfiguration in large scale mobile ad-hoc networks. In *Proceedings of European Wireless 2002, Florence, Italy*, February 2002.

[46] Jidong Wu and Martina Zitterbart. Service awareness and its challenges in mobile ad hoc networks. In *GI Jahrestagung*, September 2001.

[47] Jidong Wu and Martina Zitterbart. Service awareness in mobile ad hoc networks. In *11th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN 2001), Boulder, Colorado, USA*, March 2001.

[48] Jungkeun Yoon, Mingyan Liu, and Brian Noble. Random waypoint considered harmful. In *Proceedings of IEEE INFOCOM, San Francisco, CA*, volume 2, pages 1312–1321, April 2003.