Integrating Service Discovery with Routing and Session Management for Ad-hoc Networks *

Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, MD 21250
{dchakr1, joshi, yeyesha}@cs.umbc.edu

Abstract

In this paper, we propose GSR: a new routing and session management protocol for ad-hoc networks as an integral part of a service discovery infrastructure. Traditional approaches place routing at a layer below service discovery. While this distinction is appropriate for wired networked services, we argue that in ad hoc networks this layering is not as meaningful and show that integrating routing with discovery infrastructure increases system efficiency. Central to our protocol is the idea of reusing the path created by the combination of a service discovery request and a service advertisement for data transmission. This precludes the need to use separate routing and discovery protocols. GSR also combines transport layer features and provides end-to-end session management that detects disconnections, link and node failures and enables service-centric session redirection to handle failures. This enables GSR to accommodate service-centric routing apart from the traditional node-centric routing. We compare GSR with AODV in terms of packet delivery ratio, response time and average number of hops traveled by service requests as well as data. GSR achieves better packet delivery ratio with a minor increase of the average packet delivery delay.

1 Introduction

The growth of handheld devices ranging from cell phones to portable mp3 players to win CE iPAQs has opened up new research directions in the area of pervasive computing. These devices have varying resource capabilities. However, a large number of them have basic networking capabilities (GPRS, IR, Bluetooth, 802.11) to connect to peer devices. Current usage of these devices vary from localized access of capabilities (mostly) to accessing Internet-based Services (sometimes) to accessing rudimentary services from peer devices like downloading business cards (rarely). However, with the increase in the heterogeneity of information, capabilities and usage of these devices, the future holds an enormous potential for these devices to utilize services in peer devices using ad-hoc networking capabilities. Examples range from mobile commerce environments to battlefront environments to sensor networks. Mobile commerce examples include receiving discount coupons at malls, carrying out automatic checkout in grocery stores. Warfront activities and sensor networks often need to integrate data (that are offered by services on various devices/sensors) from heterogeneous sources to discover meaningful trends.

It can be argued that the fundamental reason for ad-hoc networks (also referred to as Mobile Ad-hoc Networks or MANETs) is for devices to use the services available from their peers in the vicinity. By "Service", we refer to any software component, data, or hardware resource on a device that it makes accessible to others. Service discovery and invocation are thus fundamental operations in an ad-hoc network. While there exists a huge body of work in service discovery in the context of wired-networks, research in the area of service discovery in ad hoc networks is relatively new [1, 2, 3]. Solutions primarily utilize the broadcast-driven nature of the underlying ad-hoc network to carry out service discovery on various devices.

Service invocation is carried out after service discovery and involves sending of service invocation data to the desired service. Service invocation primarily utilizes underlying ad-hoc routing protocols [4, 5, 6, 7] for its

*This work supported in part by NSF Awards 9875433 and 0070802, DARPA DAML program and IBM

operation. Most prior work in the area of service discovery and invocation assumes that the process of service discovery and routing are only loosely coupled. To the contrary, it has been argued in [8, 9] that cross layer integration of protocol stacks improve system efficiency. There has also been some work in utilizing service-centric data to route packets [10, 11] for wired networks. AODV [4] defined a service extension to its routing protocol to incorporate discovery. However, as discussed in section 2, the extension considers only bit-level addressing of services and is primarily based on the broadcast-driven nature of the AODV protocol. We argue that an efficient service discovery protocol can provide further efficiency to an integrated discovery and routing protocol. Furthermore, incorporating some transport layer end-to-end session management with the integrated layer provides greater reliability to end applications in an ad-hoc network.

Apart from the benefits pointed out in [8], integration of the service discovery with routing in ad-hoc networks provides the following benefits: (1) **Usage of available routes:** The discovery infrastructure while trying to discover a service discovers multiple possible paths to reach a service too. Typically, a discovery infrastructure discards this information. While this is not needed in wired networks (since network topology is fixed and there are very few route changes), it could be effectively used by ad-hoc routing protocols; (2) **Service-centric Route Enablement:** Multiple instances of the same service may potentially exist on different ad-hoc nodes. If needed, the integrated layer can use the information in the discovery infrastructure to route the invocation data to a *service instance* instead of a *node address*. This makes the integrated protocol *service-centric* instead of the traditional *node-centric* approach towards routing; (3) **Resilience to Service-Node Failures:** Moreover, all routing protocols are node-centric (they route based on the *node address* or IP address) and hence prone to failure of that node. Service-node failure leads to the service being unavailable leading to a service failure. Ideally, we would like service discovery and invocation to be immune to service-node failure since multiple instances of the same service could be existing on different nodes. We achieve this by combining the service discovery and routing layers. We borrow the notion of *path-repair* which is widely used in optical networks where the switching fabric is aware of multiple paths from source to destination. However, instead of multiple paths, the service discovery layer is aware of multiple instances of a specific type of service and the route to that service. In the event of a service-node failure, this new integrated layer can rediscover another instance of the service and deliver data to it. (4) **Reduced Routing Overhead:** Reuse of the discovery infrastructure to route invocation data results in reduced overhead since certain mandatory actions of standard routing protocols like *route discovery*, *path maintenance* can potentially be integrated with the discovery infrastructure. Typically, ad-hoc service discovery protocols involve local or global broadcasting of service advertisements and service requests until the requestor has discovered the desired service. Once a service is discovered, any standard *on-demand* or *link-state/distance-vector* based routing protocol is used for service invocation. These routing protocols, which reside below the service discovery layer, perform route discovery (in case of on-demand protocols) and link-maintenance (in case of link-state/distance-vector protocols) using a flooding-based approach. The overhead is essentially redundant because these steps could be combined with the broadcasts done during service discovery.

In prior work we have developed a distributed and network efficient peer-to-peer caching based service discovery architecture for ad hoc networks [3] that performs better than traditional broadcast-driven discovery architectures [1, 2]. However, it uses the available underlying ad-hoc routing protocol for service invocation and data transmission. In this paper, we describe our work in enabling our discovery infrastructure to support service invocation and hence obviate the necessity of a separate routing protocol. We also show that integrating the two layers in fact provide us with better system efficiency. We call our protocol Group-based Service Routing Protocol (GSR). Central to our routing protocol is the concept of reusing the path created by a service discovery request and a service advertisement to enable data transmissions. The service request matches a service advertisement at an intermediate node. The route between the source and the destination service is formed by the combination of the path traversed by the service request and the path traversed by the advertisement. We call the path that is actively participating in data transfer as the *ACTIVE_PATH*.

Traditional routing protocols do not encapsulate transport layer features like end-to-end network state maintenance, session management etc. However, since GSR offers upper layer applications the feature to discover and invoke a service at the same time, we have augmented GSR to provide the transport layer features of end-to-end session management during service invocation. We call this protocol Group-based Service Routing Protocol with session management (GSR-S). A session is maintained for each invoked service at each node in the *ACTIVE_PATH*. Each session handles various kinds of failures (service-node and link failures), buffers ongoing data transmissions at the intermediate nodes and performs service-centric session redirection depending on session-

specific preferences from the source of the request. GSR defines two kinds of sessions: *Service-Consistent session* and *Node-Consistent session* to offer various kinds of service guarantees to the end user.

We note that GSR also supports standard node-centric routing. We dont attempt to override the approach of keeping routing at a separate layer in the network stack. We argue that when the upper layer is essentially a service discovery protocol, then integrating the discovery with the routing protocol yields better efficiency.

We present simulation results comparing our integrated routing protocols (with and without session management) with a version where we have our service discovery protocol running over standard Ad-hoc On-demand Distance Vector (AODV) [4] routing protocol. We compare average packet delivery ratio, average service response time, average packet delay and average service response time and packet hops (data and request packets). Our integrated protocol gives almost 100% packet delivery ratio with a minor increase in the average packet delay. This is much better than the standard performance of AODV as a routing protocol in such environments. We also observe that reusing service discovery paths often results in reduced data path length. Drawback of our protocol is in the increase in the average packet delay for GSR-S. This is mostly due to the buffering and retransmission caused due to session redirection by GSR-S. However, an analysis of delay distributions shows that majority of the packets have delays comparable to AODV packet delay. We chose AODV as a baseline for comparison over other protocols because link-state/distance-vector protocols [7, 6] have sufficiently more routing overhead than on-demand protocols (AODV, DSR).

2 Background

There has been ample amount of work in development of routing protocols in ad-hoc networks. Protocols like DSR [5], AODV [4], TORA [7], DSDV [6] are only a few of the routing protocols that have come up over the last 10 years. However, all these protocols are node-centric and the source has to know the destination address before ensuing an interaction. Standard node-centric routing protocols cannot be used for service discovery. This is primarily due to couple of reasons: (1) Discovery protocols do not assume that they know the node addresses of the devices in the vicinity to be able to use standard routing protocols to reach the devices and check whether the required service exists on those devices (2) Unique Address assumption: Routing protocols assume device addresses are unique. However, services are not unique and there could be multiple instances of the same service in a network. However, routing protocols have been traditionally used to route service invocation data once the service has been discovered.

Work in the field of service discovery in ad hoc networks is relatively new. There exists a spectrum of distributed approaches [1, 2, 12, 10] to enable service discovery. On one end of the spectrum lies a request-broadcast-based solution where a service request is globally broadcast to all nodes in the network. Nodes having the particular service reply to a request. On the other end of the spectrum lies an advertisement-broadcast-based solution where a service advertisement is broadcast to all nodes in the network. Each node interested in discovering services caches the advertisements. The advertisements are matched with service requests locally and a result is returned.

Apart from the fact that these protocols are inefficient in terms of bandwidth and resource usage (since the requests/advertisements have to be processed by *all the nodes* which have limited processing capability and battery power), global broadcasting of messages is a very non-scalable solution, especially for large-scale ad-hoc networks [13]. Caching of all advertisements is another bottleneck since many of the nodes have limited memory and are unable to store all the advertisements and soon the cache gets filled up. Our service discovery protocol (referred to as Group-based Service Discovery protocol - GSD) [3] is based on bounded broadcasting of advertisements in the vicinity, peer-to-peer caching of advertisements and intelligent request routing (to ensure maximum reachability) based on service group information that is propagated with an advertisement. It avoids global broadcasting of requests or advertisements and decreases the network load to a large extent. Moreover, network-wide reachability is not compromised too. It is worthwhile to note that even though there is a plethora of work in wired-network based service discovery architectures and protocols [14, 15, 16, 17], centralized/semi-centralized architecture of these protocols, registry-based working model and dependence on a stable underlying network connection make them unsuitable for service discovery in ad hoc environments.

The idea of integrating routing with service discovery has been discussed earlier in [8, 18]. There has been work [8] in looking at the issues and benefits involved in cross-layer optimizations in Bluetooth scatternets. The

principal idea is to integrate the link, routing and service discovery layer so that efficient handling of power is possible. This body of work calls for a unified network stack instead of the traditional protocol design. Our work also follows along these lines except that our integrated protocol is general for any ad hoc network. It also develops on the concept of a "bottom-up" integration where the routing layer is integrated into the discovery infrastructure above it. We also show that careful design of the service discovery protocol can provide better routing support.

In [18], Balakrishnan et. al, present an integrated message routing and service discovery architecture. However, this solution assumes an underlying wired network infrastructure support and solutions like DNS for bootstrap. Moreover, message routing is done by piggybacking service data along with discovery request. Thus, the discovery is in some sense tied to the routing layer. GSR does not rely on any wired network infrastructure and also supports use of the discovery layer as a separate protocol along with other traditional routing protocols if needed.

Content-based Networking [11] mentions about message routing based on message content rather than node-address. They employ a publish-subscribe-based middleware solution for data routing and in some sense is geared towards wired networks. In essence, this is similar to service-driven routing. However, publish-subscribe or middleware solutions do not perform well in a distributed ad-hoc environment due to their centralized/semi-centralized architecture.

Ad-hoc On-Demand Distance Vector Protocol (AODV) [4, 19] has defined an extension that is based on the idea of enhancing the routing protocol by adding a service discovery extension to it. It uses the AODV_RREQ message as a service discovery request when its 'S' flag is set. The IP address field contains the service address and port number. Even though this is one way of performing service discovery and enables service-based routing, the key limitations/differences from GSR are: (1) It assumes only one service per node whereas GSR does not have any such assumption. (2) The discovery protocol is dependent on the broadcast-based architecture of AODV. GSR on the other hand, uses selective forwarding based on the service group information (discussed in section 4) and is more efficient than simple broadcasting used by AODV. This reduces network overhead in the integrated protocol.

The remaining part of the paper is organized as follows: Section 3 defines some key terms used in GSR and GSR-S. In section 4, we briefly cover our Group-based Service Discovery protocol (GSD). Section 5 describes the design and various features of GSR. Section 6 describes implementation components of GSR. We present our experimental evaluation of GSR, GSR-S and compare it with AODV in section 7. We finally conclude in section 8.

3 GSR Protocol Key Terms

In this section, we define some key terms and concepts associated with GSR. GSR uses GSD as the service discovery protocol and incorporates routing support to it by enabling service invocation and data transmission. Some of the definitions in this section are intuitive and well-known. We define them for the purpose of clarity with respect to our work.

- **Request Source (RS):** Node from where a particular service discovery and invocation request originates. Note that a node is referred to as the *Request Source* only with respect to the request that it has originated.
- **Service Provider (SP):** Nodes that contains services that are accessible from other peer nodes.
- **Intermediate Node (IN):** For a particular discovery request, a node where the discovery request finds out a matching service.
- **ADVERTISEMENT_PATH:** Path traversed by a service advertisement starting from a particular SP to an IN. It is measured in *number of hops*.
- **REQUEST_PATH:** Path traversed by a service discovery request starting from the RS. It is also measured in *number of hops*.
- **RESPONSE_PATH:** Path traversed by a service reply that is generated in response to a service discovery request. It is measured in *number of hops*.

- **DATA_PATH:** Path formed by combining an ADVERTISEMENT_PATH and a REQUEST_PATH that meet at an IN. The IN could as well be the SP or the RS (in which case the length of either the ADVERTISEMENT_PATH or the REQUEST_PATH would be 0).
- **ACTIVE_PATH:** It is defined as the DATA_PATH actually employed to transmit service invocation data[1] to the discovered SP. Out of multiple DATA_PATHs, a single path is chosen to be the ACTIVE_PATH for a service invocation.
- **Service-Consistent Session**: It refers to a session that requires all data to be sent to a particular service but does not require it to be sent to a particular node. In case of service-node or link failures, such sessions could be redirected to another node hosting the same service.
- **Service-consistent Discovery Request:** Service discovery request that just specifies the service description and does not contain any node specific information.
- **Node-Consistent Session**: A Node-Consistent Session requires service invocation data to be sent to a particular service at a particular node.
- **Node-Consistent Discovery Request:** Service discovery request that contains node specific information apart from the service that it is trying to discover.

4 GSD: Group-based Service Discovery

For the purpose of completeness, we give a brief description of our Group-based Service Discovery Protocol (GSD) [3]. GSD is based on the concepts of (1) Bounded advertising of services in the vicinity (2) Peer-to-Peer dynamic caching of service advertisements (3) Service group-based selective forwarding of discovery requests.

GSD exploits the semantic capabilities offered by DARPA Agent Markup Language [20] to effectively describe services/resources present on nodes in the MANET. Services are described using an ontology developed using DAML+OIL [21]. Semantic Service description has two purposes in GSD: (1) Services are classified into hierarchical groups depending on their functionalities. This information is used to selectively forward a service request to other nodes in the MANET thus preventing request broadcast. (2) Apart from various advantages provided by semantic service description [22], it enables us to discover services that are functionally *identical* or *similar* to the service specified in the discovery request even if they have different names or invocation mechanisms. Figure 1 shows a snapshot of the extended hierarchical service group used in GSD.

Each Service Provider (SP) periodically advertises a list of its services to all the nodes in its radio range. An advertisement message consists of the following fields:

<Packet-type, Source-Address, Service-Description, Service-Groups, Other-Groups, Hop-Count, Lifetime, ADV_DIAMET

A monotonically increasing identifier called *broadcast-id* along with the *source-address* uniquely identify a broadcast and detects duplicate advertisements. The *Service-description* and *Service-groups* contain information about the local service(s) and their corresponding service groups.

Additionally, each IN receiving the advertisement can forward it to all other nodes in its radio range. The field ADV_DIAMETER determines the number of hops each advertisement travels. Each IN increments the Hop-Count when it forwards an advertisement that is in turn used to compute whether the advertisement can be forwarded any further. ADVERTISEMENT_PATH in this context is the path formed by service advertisements from the SP to an IN. Hence, a single advertisement creates multiple ADVERTISEMENT_PATHs starting from the SP. Each node on receipt of an advertisement stores it in its *Service Cache*. Each entry in the Service Cache contains the following fields:

<Source-Address, local, Service-Description, Service-Groups, Other-Groups, Lifetime>

[1] we use the term *service invocation data* and *service data* interchangeably in the rest of the paper.

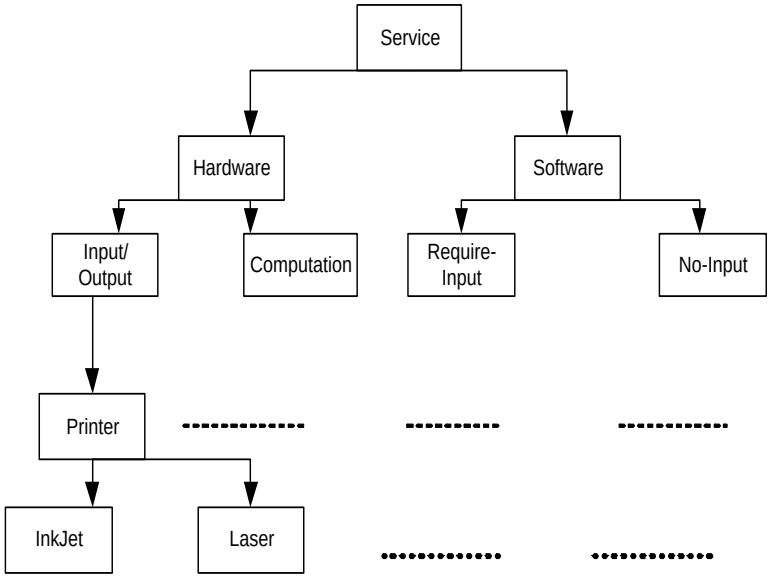

Figure 1: Hierarchical Service Group based on Functionality

Apart from storing advertisements, a Service Cache also stores descriptions of local services in the node(identified by the *local* field in each cache entry). The field *Other-Groups* contain a list of the groups that the corresponding *Source-Address* (sender of the advertisement) has seen in its vicinity. We follow a lowest-remaining-lifetime replacement policy to replace entries when the cache is full. The advertisement frequency, advertisement diameter and advertisement lifetime are user-controlled parameters that enables GSD to be adapted to the necessities of the device and the environment.

Advertising Service Groups: Apart from advertising its own services, GSD also uses the same advertisements to advertise functional group information of services a node has seen in its vicinity. The field *Other-Groups* in an advertisement contains an enumerated list of the service groups of all the *non-local* services seen by sender node. This information is obtained from the advertisements stored by the IN in its service cache. This service group information gets propagated from one IN to another and may potentially cover the whole network (if the network is partition free). Functional group information provides a good abstraction to represent services and is enough to divert a discovery request towards the appropriate region. They also provide a good measure to aggregate the service descriptions and hence save on network bandwidth. Figure 2 shows an example of propagation of service advertisements and the associated service group information for a simple ad-hoc network.

Request Routing: A service discovery request originates from a Request Source (RS) whose application layer requests the service. A request consists of our ontology based description of the service requested that also optionally includes descriptions of service groups to which the requested service belongs. The request is matched with the services present in the local cache of the RS (that might also be a SP). A service discovery request is formed on a local cache miss. A service discovery request contains the following fields:

<Packet-type, broadcastId, Service-Description, Request-Groups, Source-Address, Last-Address, Hop-Count>

The field *Request-Groups* contains the service group(s) to which the requested service belongs. *Hop-Count*, a user-controlled parameter specifies the maximum propagation limit of the request. We use the information regarding *Other-Groups* present in the service cache of each node to selectively forward a discovery request in case of a local cache miss. We recall that each entry in the service cache of a node contains a field *Other-Groups*. Thus, if the request belongs to one of those groups, then there is a chance that the requested service might

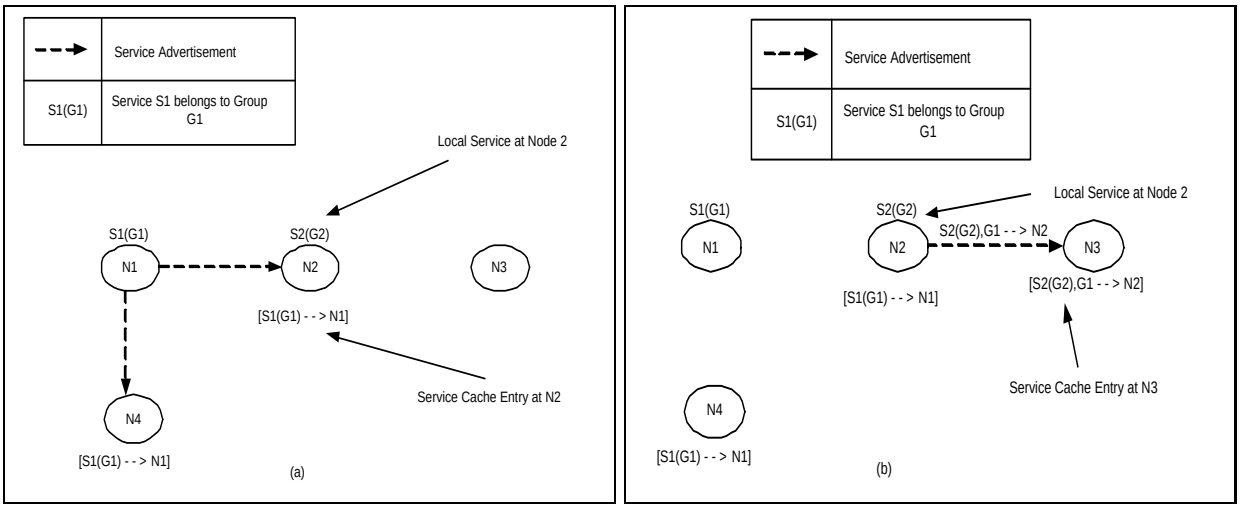

Figure 2: Service Advertisements and propagation of service group information. Figure 2a: Advertisements being sent by node N1. Figure 2b: Service Group information being propagated by node N2 during its advertisement phase.

be available near the IN that sent the advertisement. Consequently, instead of broadcasting the request GSD selectively forwards the request to those nodes.

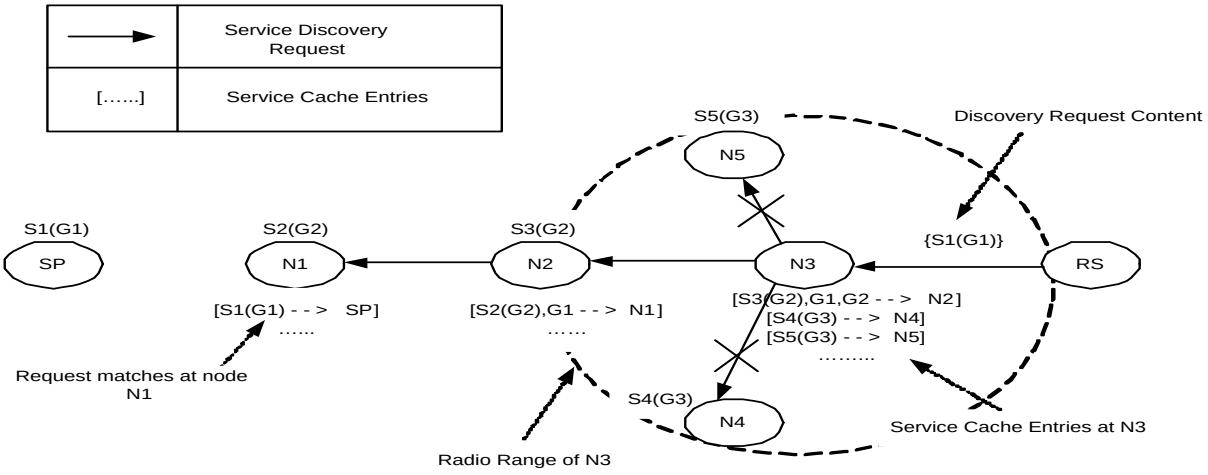

Figure 3: Group-based Selective Forwarding of Service Discovery Request

The selective forwarding process is explained in Figure 3 for a simple ad-hoc network. It shows a sequence of nodes (that are all INs) connected to each other with RS being the requesting source and SP being the service provider where the requested service (S1) is available. For the sake of simplicity, we only display a linear connection of nodes and do not show other nodes that might be present in the vicinity. We do not show the exchange of advertisements in the figure. Assuming that each node has advertised its own services and other remote service groups, Figure 3 shows the partial service cache entries in each node. For example, the entry

$$S2(G2), G1 -> N1$$

means that node N1 has service S2 belonging to group G2 and it has seen a service belonging to group G1 in its

vicinity. When a request belonging to group G1 comes to N3, then instead of broadcasting it to all nodes in its vicinity (N4, N5), N3 selectively forwards it to node N2. This is because only N2 claims to have seen a service belonging to group G1 in its vicinity. This process continues in all other nodes until the request has reached N1 where it finds a direct match of the requested service (present in the service cache of N1).The request is by default broadcast to other nodes when the algorithm fails to determine a set of nodes to selectively forward the request to.

A *Service Reply* is generated by the IN once a service request matches a service advertisement. The reply is transmitted back to the source using either the reverse route used by the discovery request or any standard ad-hoc routing protocol in case of a route failure. We describe the protocol in detail in [3, 13]. In the next section we describe how we incorporate service invocation and service data routing using the discovery infrastructure.

5 GSR Protocol Design

GSR uses the discovery infrastructure of GSD to support service invocation and transmission of service data instead of using a separate ad-hoc routing protocol. The central idea in GSR is to utilize the ADVERTISEMENT_PATH and the REQUEST_PATH (or the RESPONSE_PATH) to transmit service invocation data. Like AODV [4], we assume link reversals on the underlying ad-hoc connection. Hence if a node A is reachable from node B, then node B is also reachable from node A under identical conditions. GSR creates several DATA_PATHs after a service has been discovered by appropriately combining the ADVERTISEMENT_PATH and the REQUEST_PATH. It selects a suitable DATA_PATH for transmission of service data. This becomes the ACTIVE_PATH. It also maintains end-to-end session over the ACTIVE_PATH to detect link failures or service-node failures. We employ the technique of partial path reconstruction that enables session redirection to a different node or reconnection to the same node through a different path depending on service guarantee requirements. We explain the various salient components of GSR in the following subsections.

5.1 Dependence on GSD

Current implementation of GSR depends on GSD for its successful operation. However, the design of the routing protocol is not dependent on GSD. GSR could as well be integrated with any other ad-hoc service discovery protocol. This is because the only condition that GSR imposes on the underlying discovery protocol is the ability to discover a service. However as explained in section 4, using GSD as the driver protocol enables GSR with the following advantages: (1) **Efficient usage of network bandwidth:** GSD performs selective forwarding (instead of broadcasting) and efficiently discovers routes to the discovered service. This is again mostly due to the hierarchical grouping of services in GSD.(2) **Semantic Session Redirection:** GSD performs semantic service matching that enables loose or near matches of services. Thus, if a service request tries discovering service S1 belonging to groups G1, GSD enables the discovery request to match with a service S2 belonging to group G1 that matches most of the functional requirements of service S1. This lets GSR to redirect a session to another service having similar functionality in case of service-node failure. We note that, a hierarchical grouping of services could be replaced by a flat grouping (where there is only one level in the hierarchical tree). However, a hierarchical approach simply enhances the chances of GSD to discover services that are farther and farther away from the specified description and hence enables graceful degradation in case of service unavailability.

5.2 Data Path Setup

A service discovery request matches service descriptions at several intermediate nodes (IN). On a successful match, a service reply is generated by the IN and transmitted back through the REQUEST_PATH in the reverse direction. Each node in the REQUEST_PATH maintains a pointer to its previous hop in the path leading back to the RS. This is setup when the discovery request reaches the nodes. However, REQUEST_PATH only contains the path from the RS to the IN. In GSD, service advertisements are broadcast. GSD maintains no information about the route from the IN to the SP. We have enhanced service advertisements in GSR where in each node receiving an advertisement maintains a pointer to its previous hop leading to the SP.

The service reply is dropped in case of disconnections or link failures. Link failures result when the node upstream in the path has either moved or shut itself down. RESPONSE_PATH refers to the actual path that

the reply traverses to reach the RS from the INs. Each node in the RESPONSE_PATH maintains a forward pointer to the node leading to the IN. This is done when the service reply traverses the corresponding node. Note that these two paths could be different if a standard routing protocol was used to transmit the service reply back to the source. Our protocol reuses the path that was already traversed by the service request instead of having to discover another new one. A DATA_PATH is formed by combining the RESPONSE_PATH with the ADVERTISEMENT_PATH to establish a complete route from the RS to an SP. Figure 4 shows the various steps in involved in the creation of the DATA_PATH.

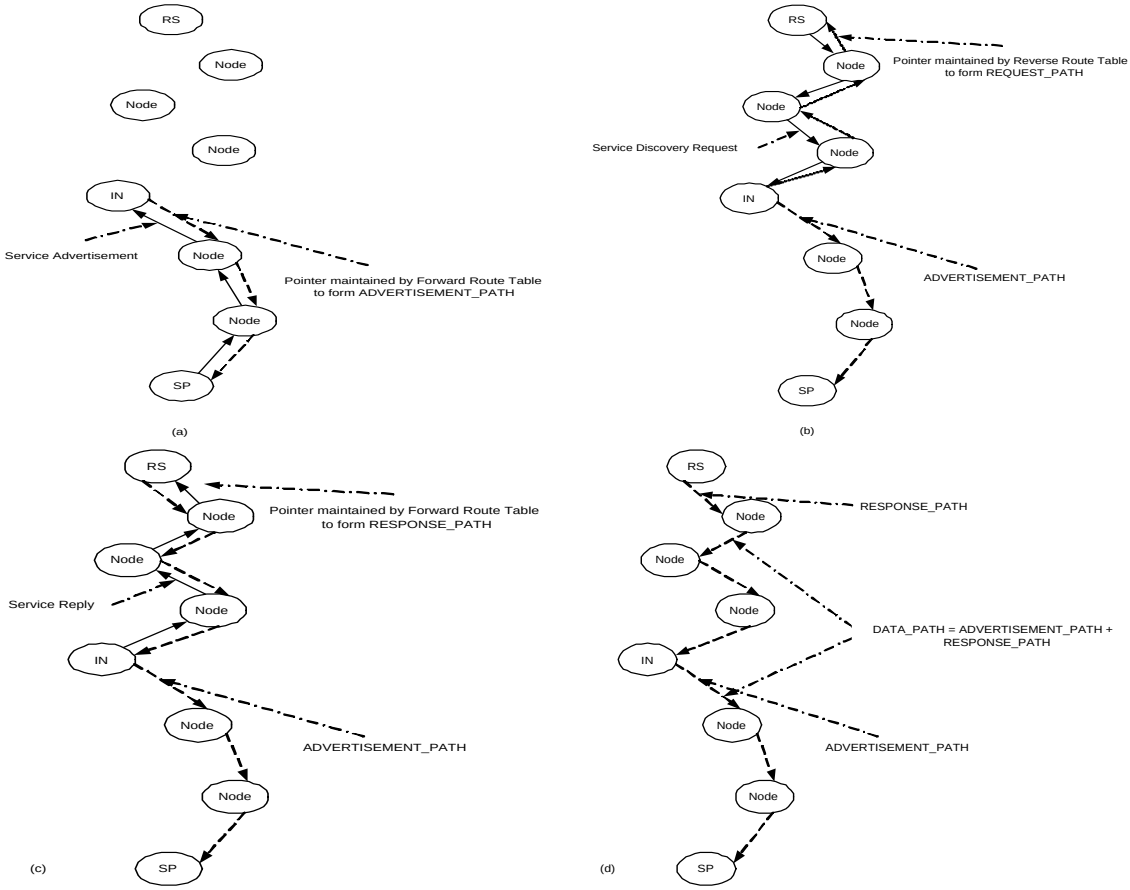

Figure 4: Creation of a single DATA_PATH in GSR. The ADVERTISMENT_PATH is set up first when service advertisement is sent by the SP (Figure 4a). The REQUEST_PATH is set up after that (Figure4b) . The service reply propagates back to the RS using the REQUEST_PATH and in turn sets up the RESPONSE_PATH (Figure 4c). The ADVERTISEMENT_PATH and the RESPONSE_PATH form the DATA_PATH (Figure 4d).

We note that for a certain discovery request, there could be DATA_PATHs established to different *similar* services or *multiple instances* of the same service. There could also be multiple DATA_PATHs to the same service through various INs. The IN bridges the RESPONSE_PATH with the ADVERTISEMENT_PATH.

Each node in the paths maintains time outs corresponding to the paths. A DATA_PATH thus expires if either the ADVERTISEMENT_PATH or the RESPONSE_PATH times out.

5.3 Active Path Selection

We have observed in the previous section that various DATA_PATHs are formed after service discovery due to the presence of potentially multiple instances of the same service. Once the RS receives a service reply, it determines

the best service to perform the service invocation. The details of the best service selection can be found in [22]. In this paper, we are concerned with routing service invocation data to the selected service. However, multiple DATA_PATHs could also be formed with the same instance of the selected service. This is because:

1. Service Requests propagate through multiple outgoing links (since they are selectively forwarded) and could potentially be answered by the same INs leading to formation of multiple DATA_PATHs. Please note that an IN detects duplicate requests. However, the RS might send out multiple discovery requests for the same service.

2. There could be multiple instances of a single unique service cached on different INs. These INs could reply to a service request thus resulting in multiple RESPONSE_PATHs and hence multiple DATA_PATHs.

One of the DATA_PATHs is chosen to be the ACTIVE_PATH for the corresponding service invocation. Currently, an ACTIVE_PATH is chosen from available DATA_PATHs based on *minimal hop count* from the RS to the SP. This information is obtained from the service reply from the IN. The service reply while traversing the RESPONSE_PATH computes the length of the RESPONSE_PATH (in hops). However, it does not know the length of the ADVERTISEMENT_PATH. To accommodate for this, we enhanced each service advertisement to compute the length of the current path and store it in each IN it traverses. An IN, while replying to a discovery request conveys the ADVERTISEMENT_PATH length to the service reply. This is used to compute the final DATA_PATH length by the RS.

Once an ACTIVE_PATH has been selected, the RS uses it to transmit service invocation data. All other DATA_PATHs constructed during the discovery phase time out if they are kept unused and all route information is deleted.

5.4 End-to-end Session Maintenance

GSR uses the selected ACTIVE_PATH to transmit service invocation data. Service invocation in ad-hoc networks requires various kinds of service guarantees. For example, the request might require that all the data be sent to one instance of the discovered service. One example of such service invocation could be data streaming applications in sensor networks where all data of a particular type needs to be sent to one instance of a service only. On the other hand, the request could be also specify that the data could potentially go to multiple instances of a particular service. An example of such service could be audio streaming applications where the audio is sent to the music service nearest to a person. Service guarantees could also be relaxed to service groups where the data could potentially go to any service belonging to a particular functional group. Example of such services could be music streaming service where the music could potentially be sent to any "speaker" in a given location. GSD protocol by virtue of its semantic service matching features is capable of discovering services based on required service guarantees. Furthermore, we have incorporated session management into GSR to provide various levels of service guarantees during service invocation too.

GSR supports two kinds of session, namely: *Service-Consistent session* and *Node-Consistent session*. A Node-Consistent session (as defined previously) requires that all data in a particular invocation be sent to *one instance* of the discovered service. A Service-Consistent session on the other hand only requires that the data be sent to the particular service. It does not impose any restriction on the service instance. In subsection 5.5, we discuss how these strict and relaxed guarantees are enforced in case of failures.

Each node in the ACTIVE_PATH maintains a session for an open connection. A session is initiated by the RS at the time of sending service invocation data. The RS specifies the type of session it desires. Each node in the ACTIVE_PATH maintains a *Session_Handler* for each connection going through it. A *Session_Handler* keeps the following information for each connection:

<Service_Description, Current_SessionId, ACTIVE_PATH_Source, ACTIVE_PATH_Destination, Next_Hop, Previous_Hop, Session_LifeTime, Session_State, Session_Strictness, Session_Buffer>.

Session_Buffer is used to buffer packets during failures. A *Session_Handler* is initiated when a service invocation data packet is received by a node. Thus *Session_Handlers* are initiated at different times at different nodes in the ACTIVE_PATH. Each data packet piggybacks along with it session related information. Session

related information include *SessionId*, *Service_Description* of the service to which this session belongs to, *ACTIVE_PATH_Destination, Session_LifeTime* and *Session_Strictness*. The parameter *Session_Strictness* specifies whether the session has to be node-consistent or the session could be service-consistent. *ACTIVE_PATH_Destination* refers to the address of the selected SP.

5.5 Path Breakage and Session Redirection

End-to-end session management in GSR provides it with connection monitoring to detect link failures and node failures. A *Link Failure* might happen due to disconnections or mobility of nodes in the ACTIVE_PATH. The node that is upstream in the ACTIVE_PATH detects this when it fails to transmit data packets to the next hop. A *Service-Node Failure* on the other hand refers to the failure due to shutting down of the SP or the SP becoming unreachable. We detect this at the node just ahead of the SP in the ACTIVE_PATH when it fails to transmit data packets to the SP.

GSR takes corresponding actions depending on the type of session. Intermediate nodes are not able to distinguish between a Link Failure and a Service-Node Failure. This is because, both the failures are triggered from failure to transmit data packets successfully. Thus, for a Node-Consistent session, the intermediate node that detects the failure uses the discovery infrastructure to discover another route to the required SP. A Node-Consistent discovery request is sent out during this type of failure. All packets coming into the node during this period are buffered in the corresponding *Session_Buffer*. If an alternate route is discovered, then all buffered packets are transmitted through the new path that becomes augmented to the already existing ACTIVE_PATH. Figure 5 describes the redirection of the ACTIVE_PATH. The session is dropped if the node fails to discover a route to the required SP.

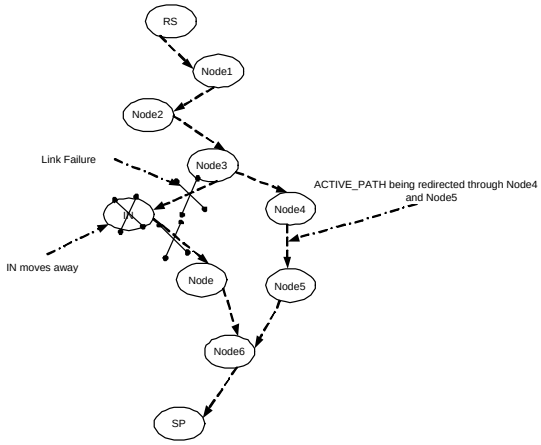

Figure 5: ACTIVE_PATH Redirection in GSR

GSR resorts to service-centric session redirection in case of link or service-node failures for a Service-Consistent session. The node that detects the failure tries using GSD to discover a service that has the same specification as of the service in the ongoing session. The discovery request sent out during this period is Service-Consistent. On a successful discovery, the session is redirected to the new SP. Note that the node detecting the failure redirects the session. This does not impose any load on the RS. Moreover, the part of the ACTIVE_PATH that is usable remains intact. The ACTIVE_PATH from the RS to the new SP is established by combining the old ACTIVE_PATH from the RS to the node that detects the failure and the new ACTIVE_PATH formed from that node to the new SP. This novelty of our routing protocol enables service invocation data to be routed based on service descriptions, thus enabling GSR to be service-centric. Currently, GSR tries discovering services exactly matching the description of the service in the ongoing session. However, we could also redirect a session to a service belonging to the group of the service in the session by utilizing the semantic service matching features and the hierarchical service groups. Figure 6 describes session redirection in GSR. By buffering packets and

retransmitting them, GSR tries to improve the reliability of the protocol. However, it does not guarantee delivery of all the packets in the service invocation data. We show in section 7 that GSR performs reasonably well compared to other routing protocols in terms of data delivery.

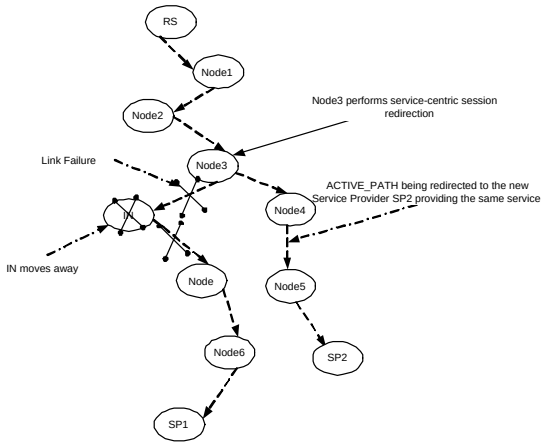

Figure 6: Service-centric Session Redirection in GSR

6 GSR Implementation Components

We have employed the use of several existing and new techniques to augment our discovery infrastructure to enable service-centric data routing. We employ the use of *Forward Route Tables* and *Reverse Route Tables* to maintain path specific information during service discovery and invocation. Additionally, we also maintain a *Session Table* that stores information about ongoing sessions through a node. A *Session_Handler* in each node maintains session data corresponding to each session. A *Session_Handler* could belong to any of the four session states, *Session_Discovery, Session_Active, Session_Dropped* and *Session_Finished.* In this section, we describe the various implementation level components of GSR.

6.1 Reverse Route Table:

Each node in addition to maintaining a *Service Cache* (for GSD) also maintains a *Reverse Route Table.* It is a node-centric route table indexed by the *source address* of a node. Important fields in the Reverse Route Table (RR Table) are:

<Source-Address, Previous-Address, Life-Time, Hops-To-Source>

Reverse Route Table performs the function of reverse routing a service reply back to the source of the request using the REQUEST_PATH. This makes the REQUEST_PATH same as the RESPONSE_PATH in GSR but in the reverse direction. The RR table is updated during the time a service request arrives at a node. Each service request packet contains a *Last-Address* field. Each node seeing a service request changes this field to reflect its own address before forwarding it to other nodes in its vicinity. This value is stored in the RR Table (as *Previous-Address*)of each node the request traverses. The RR Table is used to send a service reply back to the RS. Our protocol offers updating of the RR Table based on shortest route or recency of the request.

- *Shortest Route Updation*: The RR Table for a source address is updated only when it has received a request that has lesser value of *Hops-To-Source* than the existing entry (if there is one).
- *Recency-based Updation*: The RR Table is updated whenever a request received from a source is more *recent* than the previously stored value. Recency is determined using the timestamp of the service request and the

Life-Time of the entry.

These actions are performed only after the discovery layer has handled duplicate requests and replies. The entry in the table is kept for REV_ROUTE_TIMEOUT time units. The REV_ROUTE_TIMEOUT value thus determines the time limit for which intermediate nodes maintains the REQUEST_PATH. Shortest Route Updation could be used for relatively stable ad-hoc networks where as recency-based updation could be used for fast-moving ad-hoc networks where the recency of the route is very important.

6.2 Forward Route Table:

It is a node-centric route table indexed by the *Destination Address* of nodes. Important entries in the Forward Route Table (FR Table) are:

<Destination-Address, nextHop-Address, Life-Time, Hops-To-Destination>

FR Table performs the role of forming the DATA_PATH by combining RESPONSE_PATH and ADVERTISEMENT_PATH. Service Invocation Data is routed through the selected DATA_PATH using FR Table. FR Table determines the *nextHop-Address* when data packets reach a certain node in the path. FR Table is updated in when a service reply or an advertisement arrives a node.

- *Service Reply based Updation*: A service reply packet contains two fields (apart from others) namely: *Last-Address* and *Service-Source-Address*. The *Last-Address* identifies the next hop required for a data packet to reach *Service-Source-Address*. These values are stored in the FR Table. Each node, before forwarding a service reply changes the *Last-Address* field to reflect its own node address.
- *Advertisement based Updation*: The FR Table is also updated when a service advertisement reaches a node. The *Source Address* of the advertisement and the *Last-Address* from where it has been forwarded are stored in the FR Table. This forms the ADVERTISEMENT_PATH. Thus each node on the ADVERTISEMENT_PATH knows the next hop neighbor in case it has to forward data to the source of the service advertisement.

FR Table updation is based on shortest route to destination as well as recency of advertisement and service reply. This is analogous to the updating on RR Table. Each entry is kept for FOR_ROUTE_TIMEOUT time units and that determines the duration of the time the DATA_PATH remains idle. The DATA_PATH is destroyed if it has not been used within this time interval. Apart from the above mentioned updations, the timeout value of entries in the FR Table that participate in an ACTIVE_PATH is updated whenever a session is initiated in an ACTIVE_PATH.

6.3 Session Maintenance:

Session_Handlers at each node manage sessions for each connection through the node. Data transfer is performed by forwarding the data packets through the ACTIVE_PATH using the FR Table. After a Session_Handler is initiated, it constructs an entry in the Session Table. It obtains the required information from the session-related information piggybacked in an invocation packet (explained before) and from the FR and RR Tables. As mentioned before, the Session_Handler can possibly have four states. A session after being initiated belongs to either *Session_Discovery* or *Session_Active* state. The states of the Session_Handler and the corresponding actions performed it are explained in detail in this subsection.

- *Session_Discovery*: This is the state when the required service has not yet been discovered and the *Session_Handler* is waiting for the requested service. *Session_Handler* on entering this state initiates a service discovery using GSD. The *Session_Handler* also buffers incoming data packets in the *Session_Buffer*. A *Session_Handler* can go into this state from either *Session_Active* state (explained below) or right at the inception of the session at the source of the request. During the inception of the session, the discovery request sent out by the *Session_Handler* is always *Service-Consistent*. However, if this state has been reached from *Session_Active* state, the discovery request sent out depends on the type of the currently existing session.

The state goes either into *Session_Active* state if a service meeting the session specifications was discovered, or goes into *Session_Dropped* in case of a discovery failure after repeated trials.

- *Session_Active*: This is the state when a *Session_Handler* engages in transferring data packets in an ACTIVE_PATH. A *Session_Handler* monitors the outgoing link for a certain connection. Whenever, it receives a data packet, it checks the FR Table to determine the next hop for it. It then forwards the data packet to the next hop. In the event of a link failure or node failure, the outgoing packet is buffered. The *Session_Handler* immediately goes back to the *Session_Discovery* phase on detecting this. This state can be reached from either a *Session_Discovery* state or during the creation of the session at a node. A node other than the source node or the destination node, but belonging to the ACTIVE_PATH can go into this state right at the creation of the session. This is because the node already belongs to a DATA_PATH and hence knows a route to the particular service/destination. Hence it can actively participate in data transfer without having to go into the *Session_Discovery* state. It transitions to the *Session_Finished* state in case of a successful transfer or to the *Session_Dropped* state in case of an unsuccessful transfer.

- *Session_Dropped*: This is the state when either the *Session_Handler* has failed in transmitting all the packets to the destination. This may result due to the failure of a node to discover the required service. All session-related information is purged from the participating nodes. Pending data packets for a dropped session are dropped too. A *Session_Handler* that goes from the *Session_Discovery* to this state, has not succeeded in discovering the required service. A session buffer overflow may lead to a session being dropped too. The session then transitions into the *Session_Finished* state at the end.

- *Session_Finished*: This is the end state reached by the *Session_Handler* once it has finished a successful or an unsuccessful session. It is reached either from the *Session_Active* or *Session_Dropped* state.

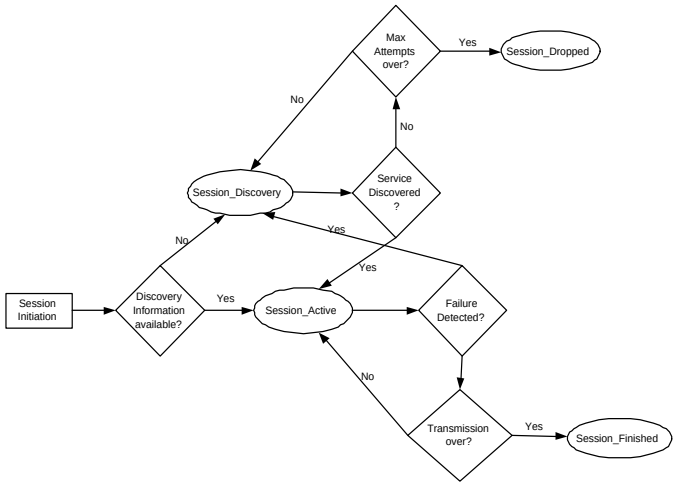

Figure 7: Session State Transition Diagram

Figure 7 depicts the session state changes. Our session maintenance and failure handling detects most failures and resorts to appropriate fault handling mechanisms depending on the session type (as explained in section 5). However, there are certain instances where the node failure might go undetected. For example, if a node is in *Session_Discovery* state, and it goes down after having buffered all incoming packets, it wont be detected by its upstream neighbor in the *Active_Path*. However, if some packets were still being transmitted by the upstream node, it would be able to detect the failure due to packet drops and take necessary actions.

7 Experimental Evaluation

We implemented GSR on the ad hoc network simulator Glomosim [23] under various mobility conditions and different node topologies. We compared GSR with our basic model where we had our service discovery architecture as a layer on top of AODV. We had two versions of our integrated protocol. The basic version does not do any session management. We call this GSR (Group-based Service Routing). The version of our protocol that performs end-to-end session management is termed as GSR-S (Group-based Service Routing with Session Management). We compare these two versions with the basic version where the discovery layer uses AODV for all routing purposes. We call this GSD+AODV (Group-based Service Discovery+AODV).

We implemented the integrated protocol as a routing layer in the Glomosim protocol stack. We used CBR (Constant Bit Rate) messages from the application layer as triggers to invoke discovery and routing in our protocol. Simulation Environment consisted of node topologies ranging from a topology with 25 nodes to a topology with 64 nodes. In this paper, we present results for the two extremes (25 nodes and 64 nodes). We used a random way-point mobility pattern for all the nodes. The initial setup followed a grid structure where all the nodes were distributed in a grid over the terrain. For the purposes of the simulation, we used representative services S0 to S64 to represent actual services and groups G1 to G10 to represent service groups with G10 being equivalent to the parent service group called "Service". Through out all the experiments, we used a pessimistic evaluation strategy by keeping the requested service only on 8% of the nodes. This is because, intuitively with the increase in the availability of the service, the performance of the protocol will improve. All the simulation parameters including the various timeouts used for different tables and link and radio layer parameters have been enumerated in Figure 8.

Duration	600 seconds
Network Area (x,y)	(110 x 110m), (250 X 250m)
Initial Topology	Grid topology
Network Diameter	7, 11
Tx Range	30m
Tx Throughput	20kbps
No. of Nodes	25, 64
Advertisement Interval	10 seconds
Advertisement Lifetime	5 seconds
Reverse Route Timeout	12 seconds
Forward Route Timeout	8 seconds
broadcast jitter	10 milliseconds
GSR-S Session type	Service-Consistent

Figure 8: Experiment Parameters

We compared GSR, GSR-S and GSD+AODV with respect to packet delivery ratio, average response time for request, average response hops, average packet delay and average packet hops. We define mobility of nodes by P(Sm, SMax) where :
P= Pause (in seconds) after the node has moved to a new position
Sm= Minimum speed (meters/sec) of movement of the node
SMax= Maximum speed (meters/sec) of movement of the node
The node moves with a speed within the range (Sm, SMax). In our simulations, we have varied node speed ranging from 1 meters/second to 9 meters/second with a pause time of 5 seconds for all the experiments. All the time calculations (packet delay, response time) are in seconds.

- *Experiment 1: Packet Delivery Ratio vs. Mobility*
  Average Packet Delivery Ratio is defined as the number of data packets received by the final destination as a fraction of the number of data packets transmitted by the source. In Figure 9 we plot the average packet delivery ratio as a function of mobility. GSR-S quite predictably shows a packet delivery ratio of

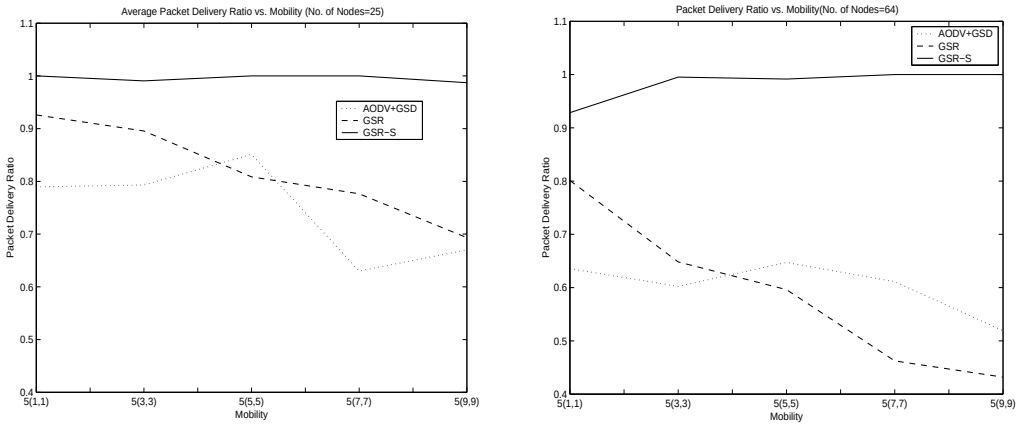

Figure 9: Average Packet Delivery Ratio Comparison graph of GSD+AODV, GSR and GSR-S

almost 1 which is significantly higher than AODV+GSD or GSR. This is mainly due to the end-to-end session management. This shows that integrating session management along with routing provides with improved efficiency in packet delivery. However, it is also noticed that packet delivery ratio of GSR is more than AODV+GSD in low mobility situations. However, AODV seems to be performing somewhat better in high mobility situations (9 meters/second) for networks consisting of large number of nodes (64 nodes in the graph). This is because often times in high mobility scenarios, the reverse routes break due to node mobility contributing to the decrease in the delivery ratio. AODV, on the other hand discovers a fresher route each time it transfers service data.

- *Experiment 2: Average Data Delay vs. Mobility*
  We calculated the Average Packet Delay for data received in Experiment 1. Logically, packet delay should increase in GSR-S since GSR-S buffers dropped packets, and again retransmits them after the session has been re-established. The results have been plotted in Figure 10. As expected, we see that packet delay in GSR-S is greater than delay in GSR or AODV+GSD. However, we also observed that the distribution of delay for GSR-S shows a high proportion of packets (about 75%) being delivered with delays almost similar to AODV or GSR. The average data delay is strongly affected by a small fraction of the packets which take a very long time to be delivered. These are packets that had been buffered by a session during a node or link failure. Figure 11 shows the delay distribution. We also observe that GSR performs consistently better than AODV in terms of packet delay. This is because GSR uses one of the DATA_PATHs to transmit data whereas AODV transmits data only after having finished its route discovery process. This result along with results from Experiment 1 shows that reusing the path already obtained during discovery would result in faster data transmission with a minor decrease in the delivery ratio. The delivery ratio on the other hand drastically improves with session maintenance.

- *Experiment 3: Average Data Hops vs. Mobility*
  The average hops traveled by data packets give an idea of the efficiency of the routing protocol in terms of discovering shorter routes. Our results show that GSR performs best in terms of average data hops both in low mobility as well as high mobility situations with small and large node topologies. GSR-S performs a little worse than GSR. This is again attributed to the fact that some of the data packets are redirected via a different path (these packets are dropped in GSR), resulting in increased hop count. AODV+GSD perform even worse than GSR-S. However, under high mobility conditions (9 meters/second) in a large node topology (64 nodes), AODV performs comparatively better than both the other protocols. We believe this is because high mobility often breaks the reverse routes. AODV, in such situations discovers fresher routes from the RS to the SP. GSR-S, on the other hand employs partial path reconstruction from an intermediate node. In high mobility scenarios, this often results in GSR-S discovering a longer path than what AODV discovers from RS to the SP. The results have been shown in Figure 12.

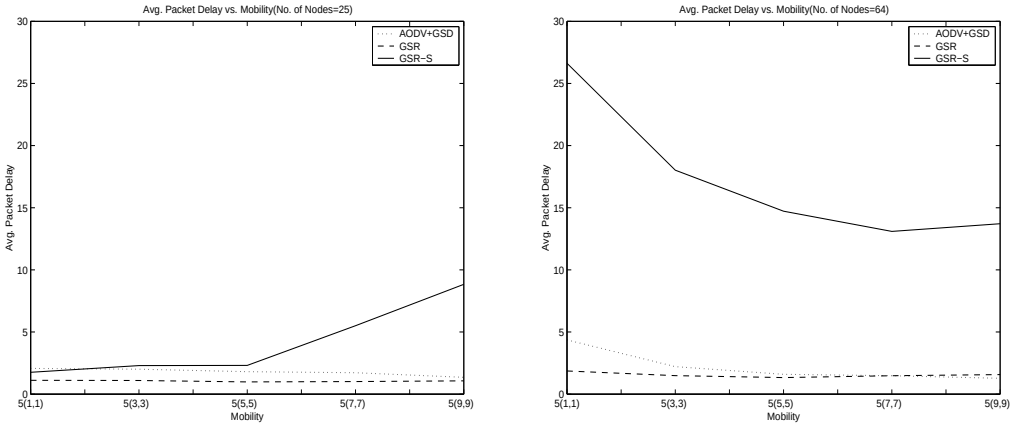

Figure 10: Average Packet Delay of GSD+AODV, GSR and GSR-S with respect to varying mobility

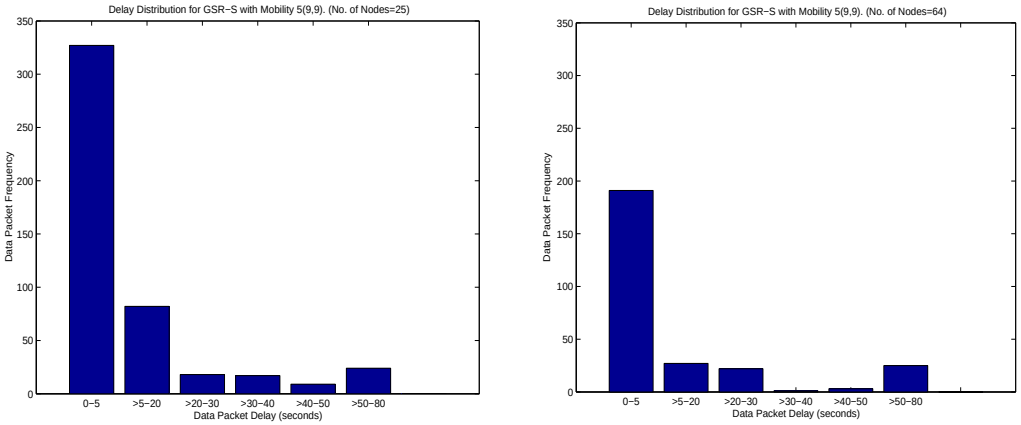

Figure 11: Delay Distribution for GSR-S with Mobility 5(9,9)

- *Experiment 4: Average Request Response Time vs. Mobility*
  Request Response Time refers to the time taken for a service reply to reach the source after a service request had been sent out. This result gives us an estimate of how long AODV takes to transmit a service reply back to the source vis-a-vis our protocol where the reply is reverse-routed back to the source. In Figure 13, we observer that with less number of nodes (25 nodes), GSR/GSR-S performs significantly better than AODV. We also observe that with 64 nodes, the response time decreases with increasing mobility. This is corroborated in Figure 14 where we see a decrease in the average response hops (thus resulting in decreased response time) for large nodes under high mobility conditions. However, with small number of nodes (25) there is no significant decrease in the average response time. GSR/GSR-S, as expected is more efficient in utilizing a shorter route than AODV.

8 Conclusions and Future Work

We have presented the design of an integrated service discovery and routing protocol (GSR). Our integrated protocol has the capability to do both node-centric as well as service-centric routing. Service Discovery is a key component for the development of distributed applications. We show that an integration of service discovery with routing in ad hoc networks offers greater system efficiency.

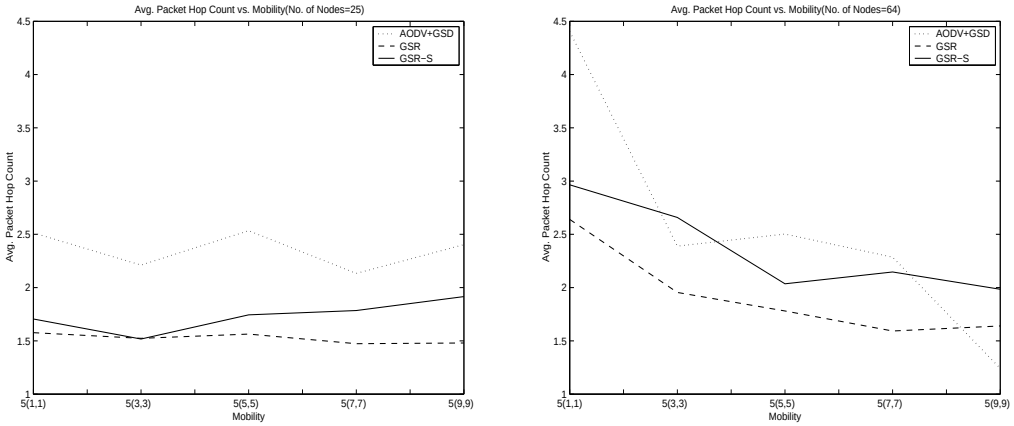

Figure 12: Average Data Packet Hop Count registered in GSD+AODV, GSR and GSR-S with respect to varying mobility

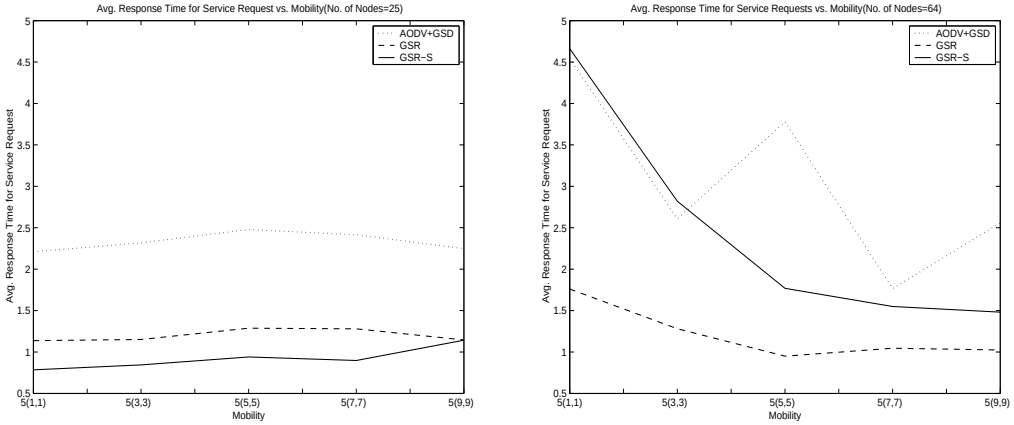

Figure 13: Average Request Response Time in GSD+AODV, GSR and GSR-S with respect to varying mobility

Integrating different layers of the protocol stack has well-known shortcomings. The integrated protocol becomes non-modular and difficult to upgrade. Correctness checks also become complicated and code reuse becomes difficult. While the benefits offered by modularity in wired networked systems outweigh the need for integration, in ad hoc networks, integration of the two layers offers greater justification in terms of the benefits obtained.

Several minor enhancements are possible in our proposed protocol. For example, currently, a service reply is dropped if it does not have a route to the source node. On detection of such a message drop, the node immediate upstream could broadcast the packet to the limited vicinity and expect some node to route the reply to the source node. This enhancement makes the REQUEST_PATH and RESPONSE_PATH different and we have to handle it accordingly. The *Session_Handler* could also be enhanced by providing it with multiple possible destinations where the particular session could be forwarded. Right now, it goes into the *Session_Discovery* state and tries to recover/redirect the session in a reactive manner. We could make a *Session_Handler* cache service replies from multiple nodes and hence pre-calculate possible destinations for an active session. Our future work plan is to integrate these features and try to make the integrated protocol more robust for upper-level applications.

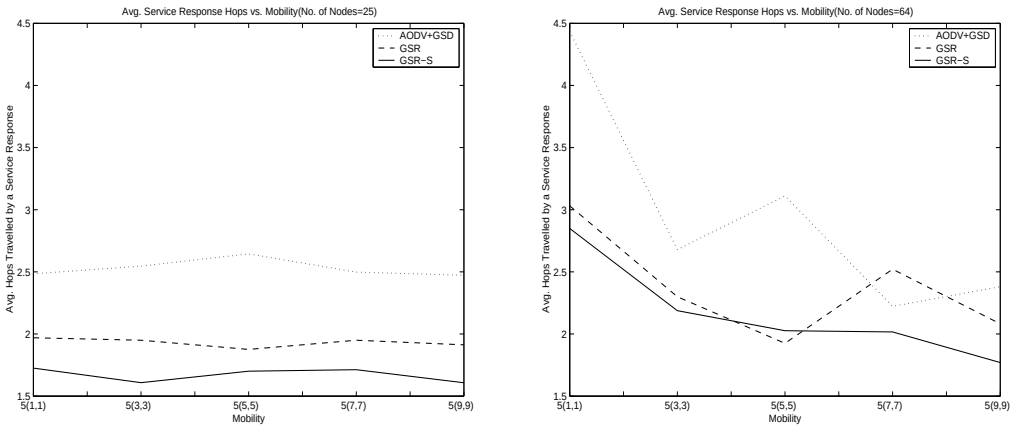

Figure 14: Average Response Hops in GSD+AODV, GSR and GSR-S with respect to varying mobility

References

[1] S. Helal, N. Desai, and C. Lee, "Konark-A Service Discovery and Delivery Protocol for Ad-hoc Networks," in *Third IEEE Conference on Wireless Communication Networks (WCNC), New Orleans, USA*, March 2003.

[2] D. Tang, C. Chang, K. Tanaka, and M. Baker, "Resource Discovery in Ad hoc Networks," Tech. Rep., Stanford University, August 1998, CSL-TR-98-769.

[3] D. Chakraborty and A. Joshi, "GSD: A novel group-based service discovery protocol for MANETS," in *IEEE Conference on Mobile and Wireless Communications Networks, Stockholm, Sweden*, September 2002.

[4] C.E. Perkins and E.M Royer, "Ad-hoc On-Demand Distance Vector Routing," in *2nd IEEE Workshop on Mobile Computing Systems and Applications*, February 1999, pp. 90–100.

[5] D.B. Johnson and D.A Maltz, "The Dynamic Source Routing Protocol for Mobile Ad-hoc Networks," *Mobile Computing, Kluwer Academic Publishers*, pp. 153–181, 1996.

[6] C.E Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," *Computer Communications Review*, pp. 234–44, October 1994.

[7] V. D. Park and M. S. Corson, "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks," in *Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Kobe, Japan*, April 1997.

[8] P. Bhagwat B. Raman and S. Seshan, "Arguments for Cross-Layer Optimizations in Bluetooth Scatternets," in *The 2001 Symposium on Applications and the Internet (SAINT)*, January 2001.

[9] V. Kawadia and P.R Kumar, "A cautionary perspective on cross layer design," in *Submitted to IEEE Wireless Communications Magazine*, July 2003.

[10] M. Balazinska, H. Balakrishnan, and D. Karger, "Instwine: A scalable peer-to-peer architecture for intentional resource discovery," in *International Conference on Pervasive Computing, Zurich, Switzerland*, August 2002.

[11] A. Carzaniga and A.L. Wolf, "Content-based Networking: A New Communication Infrastructure," in *NSF Workshop on an Infrastructure for Mobile and Wireless Systems. In conjunction with the International Conference on Computer Communications and Networks (ICCCN), Arizona, USA*, October 2001.

[12] H. Horiuchi S. Motegi, K. Yoshihara and S. Obana, "Proposal of service discovery for wireles ad-hoc networks," *IPSJ Journal. Vol 33. No. 12*, 2002.

[13] D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, "Towards distributed service discovery in pervasive computing environments," Technical Report, TR-CS-03-26, University of Maryland Baltimore County, April 2003.

[14] K. Arnold, B. Osullivan, R. W. Scheifler, J. Waldo, and A. Wollrath, *The Jini Specification (The Jini Technology)*, Addison-Wesley, Reading, MA, June 1999.

[15] The Salutation Consortium Inc 1999, "Salutation Architecture Specification (Part 1), Version 2.1 Edition," `http://www.salutation.org`.

[16] R. John, "UPnP, Jini and Salutaion - A Look at some popular Coordination Frameworks for Future Network Devices," Tech. Rep., California Software Labs, 1999, http://www.cswl.com/whiteppr/tech/upnp.html.

[17] Universal Description Discovery and Integration Platform, ," `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`, September 2000.

[18] E. Schwartz W. Adjie-Winoto and H. Balakrishnan, "The Design and Implementation of an Intentional Naming System," in *In Proceedings of the Symposium on Operating Systems Principles, South Carolina, USA*, December 1999.

[19] C.E Perkins, E.M Royer, and S. Das, "Ad hoc on-demand distance vector protocol.," in *IETF Internet Draft. Version 12.*, November 2002.

[20] DARPA Agent Markup Language, ," `http://www.daml.org`.

[21] DARPA Agent Markup Language and Ontology Inference Layer, ," `http://www.daml.org/2001/03/daml+oil.daml`.

[22] D. Chakraborty, F. Perich, S. Avancha, and A. Joshi, "DReggie: A Smart Service Discovery Technique for E-Commerce Applications," in *Workshop in conjunction with 20th Symposium on Reliable Distributed Systems*, October 2001.

[23] R. Bagrodia X. Zeng and M. Gerla, "GloMoSim: A Library for Parallel Simulation of Large-scale Wireless Networks," *12th Workshop on Parallel and Distributed Simulations, Alberta, Canada*, 1998.