

COP5615 – Operating System Principles
Fall 2006
PROJECT 3.1

SERVER PROTOCOL DISCOVERY

Team Number:

06

Team Members:

1. RAJALAKSHMI KRISHNAMURTHI
2. PRAKASH KUMAR
3. NITIN
4. SATISH CHANDRA

Connectionless service.

We know that we are working in the connectionless service environment, which is supported by the connectionless protocol. As we, all know that UDP is a protocol, which supports connectionless services.

UDP (User Datagram Protocol):

A communications protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP). UDP is an alternative to the Transmission Control Protocol (TCP) and, together with IP, is sometimes referred to as UDP/IP. Like the Transmission Control Protocol, UDP uses the Internet Protocol to actually get a data unit (called a datagram) from one computer to another. Unlike TCP, however, UDP does not provide the service of dividing a message into packets (datagrams) and reassembling it at the other end. Specifically, UDP does not provide sequencing of the packets that the data arrive in. This means that the application program that uses UDP must be able to make sure that the entire message has arrived and is in the right order. Network applications that want to save processing time because they have very small data units to exchange (and therefore very little message reassembling to do) may prefer UDP to TCP. The Trivial File Transfer Protocol (TFTP) uses UDP instead of TCP.

DATAGRAM:

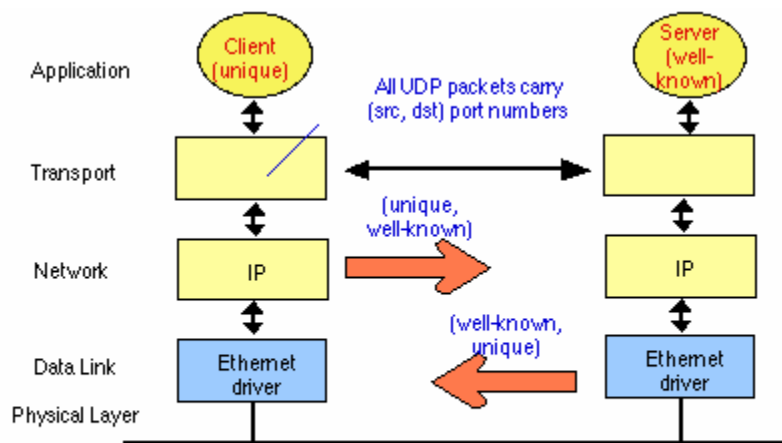
A self-contained, independent entity of data carrying sufficient information to be routed from the source to the destination computer without reliance on earlier exchanges between this source and destination computer and the transporting network." The term is used in several well-known communication protocols, including the User Datagram

Protocol and AppleTalk. A very similar term, packet, is used in the Internet Protocol and other protocols related to the Internet.

A datagram or packet needs to be self-contained without reliance on earlier exchanges because there is no connection of fixed duration between the two communicating points as there is, for example, in most voice telephone conversations. (This kind of protocol is referred to as connectionless.)

UDP Communication Setup

A server process (program) listens for UDP packets received with a particular well-known port number and tells its local UDP layer to send packets matching this destination port number to the server program. It determines which client these packets come from by examining the received IP source address and the received unique UDP source port number. Any responses that the server needs to send back to a client are sent with the source port number of the server (the well-known port number) and the destination port selected by the client.



This Project, i.e. Project 3 is an extension of the project 2 which we have submitted last time. Here we have incorporated more advance features such as:

1. Creation of chat rooms,
2. Joining a room,
3. Leaving a room,
4. A list of users engage in chatting can be generated, and
5. A list of all chat rooms.

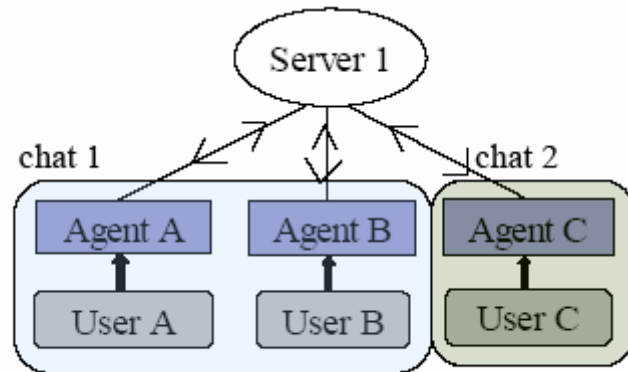
Here we have also implemented the multicasting techniques using IP and its features.

In the Project 1 we were having the two modules as Command module and the Server module in this Project 2 the two modules will be referred as:

The Command module is referred as 'end user' module and the Server module is referred as 'agent' / 'user agent' module.

We also have created a centralized communication management server, which would be referred as 'Server'.

The idea to design Project 2 was:



There could be multiple instances of such 'server' running independently of each other on the network. The Project 1 does not follow the principle of handling the collisions and it does not support the collision detections, which come out, be one of the major drawbacks of the Project 1.

Here what we have implemented is that if we have two chat rooms the centralized 'server' will handle i.e. chat room 1 and chat room 2 collisions. Server maintains information about all users and on chat rooms and the corresponding multicast details.

- A. Implementation of the 'server' that is running all the time and first to be started.
- B. Implementation and addition of the more number of commands in the 'end user' interface.
- C. Implementation and the addition of the number of commands to the of 'agent' or 'user agent' module.

(A) Server Module:

The server module is having two threads. A receiver thread which will start executing as soon as the server application is started and will continue execution till the server module is terminate i.e. both are running concurrently. The second thread will be the sender thread that will be called to send messages as and when needed and will terminate after sending the message to the intended address. The server creates and manages multicast groups. The server will support the following commands.

- I. createroom <one word name> – it creates a chat room and names that chat room
- II. joinroom <chat room name> - it allows particular user to join a chat room
- III. leaveroom <chat room name> - it allows the user to leave a chat room

- IV. `ulist <chat room name>` - it lists all the users that are in a chat room
- V. `listrooms` – it lists all existing chat room names

Message packet corresponding for each command possibly could have the following structure (remember these commands are coming from the user-agent to the server and the format specified is just an example)

`command [argument if any] [user alias name] [ip address] [port number] [other parameters]`

Restrictions we have considered as explained in the project:

- a. We will not assume peer-to-peer or end to end communication between agents / users.
- b. All chat messages will be originating from end users and terminating to some chat room and not to any particular 'user' but control messages may terminate to the server.
- c. In addition, any user at any given time could participate in only 1 chat room.

Server Module operations explained:

I) *createroom command:*

Whenever the server receives a command to create a new chat room. It must first check for any existing rooms with the same name. In addition, it must check if the requesting user is not participating in any other chat room by comparing the user agent's receiver thread ip address and port number. If no such room is found and the user is not part of any other chat room then it creates a new room and generates a unique seed for that room.

II.) *joinroom command:*

When the server receives a joinroom command, it first checks if that user is already participating in any existing chat room or not. This check is done by comparing the user agent's talk receiver IP address and port number in the list of users for any existing chatroom. If the user is already participating in any existing chat room, server responds back by a failure message specifying the name of the chat room the user is already participating. In addition, if some other user agent in that chat room user list already uses the alias, the server responds back indicating name collision to the end user agent further advising it to change its self-alias and try again. Otherwise, it returns a message specifying the chat room seed and the corresponding multicast address. Server also then adds this user to the list of users for that chat room.

III.) *leaveroom command:*

When the server receives a leaveroom command, it first checks if that user is already participating in that particular chat room or not by comparing the ip address, port number fields of the existing users for that chat room. If the user is currently participating in that chat room then the server removes its entry from the list of user for that chat room. It immediately changes that chat room unique seed, again immediately communicates to the remaining user of that chat room the new chat room seed (using UDP unicast messages) in turns. It then indicates to the original user that it has been successfully removed from

that chat room. Otherwise, it returns a failure message. If the server changes the unique seed for that chat room it, also updates the chat room table with the new seed value. Further if after removing the requesting user from the chat room successfully, if the chat room becomes empty, the server must destroy that chat room session by removing that room details from the existing chat room table

IV) *ulist command:*

When the server receives a ulist command from the user agent, it sends back the list of all users specifying their alias name along with the IP address and the port number for that chat room regardless of the requesting user participation status in that chatroom. If no such chat room exists, server returns back an error message.

V) *listrooms command:*

When the server receives a 'listrooms' command, it responds back with the list of all existing chat rooms names to the requesting user-agent talk receiver address.

(B) End User Module:

The End User module operations explained:

The End User commands we have implemented in the Project1 will remain the same for the Project 2 also and there is one more addition we have done here i.e. the 6th command which is as follows

VI csend <message> - sends message to current chat room multicast address

All other commands usage except csend has been explained in Server Module (already explained above) of the project. The csend command is meant to send message to the chat group.

(C) Agent / User Agent Module:

Agent is synonymous to the server module of project 1. In addition, it also has a fourth thread namely mcast sender thread that is called upon to send multicast messages to the chat group. In addition, there exists a fifth thread, which listens to any incoming multicast messages called mcast receiver thread. Fifth thread is also started when the agent joins or creates a chat room and it terminates when the agent receives a command to leave that chat room, let us call it mcast receiver thread. First two threads run continuously when user agent is executed and terminates when it receives 'exit' command.

Agent / User Agent operations explained:

When the agent receives a createroom, joinroom, leaveroom, ulist, listrooms command at its command server thread, it spawns a unicast sender thread passing it the command and the corresponding parameters. The unicast sender thread then appropriately creates a UDP message packet containing necessary information for the “server” which includes

the self-alias information as well as unicast receiver thread's ip address and port number and sends it to the "server's receiver thread ip address and the port number.

If the agent's command server thread receives the csend command, it spawns a mcast sender thread passing to it the message as the argument. The mcast sender thread creates a new multicast message packet to send to the chat room specified by the multicast address global variable. The constructed packet's data portion must contain the unique chat room seed followed by the message in that order. It then sends that packet on the multicast channel and terminates after successful operation; otherwise, it should output an error message indicating send failure. When the mcast receiver thread receives a packet on the multicast channel, it first checks for the validity of the message. If the first field of the data portion matches with the current unique chat room seed, the message component is displayed on the console otherwise the packet is discarded without further action.

Running Server and Clients and Command Syntax

1. Client will ask for alias by default as:

Enter Alias..... user1 /*no space in the screen name i.e alias*/

2. Type list then hit enter:

list /*this give you the list of all connected users*/

3. Setting new alias write setalias enter new alias then hit enter

setalias <aliasname>

4. For sending msg to other type sendto and then type message hit enter

sendto <alias> <msg>

5. For removing from chat just type remove and hit return key

remove <alias name>

6. To reenter in to the chat again follow step 5.

7. To get List of Chat Rooms available just type listrooms and hit return key

listrooms return

8. To Join any Chat Room type "Join" and then enter the room name.

join <room name>

9. To get users present in that Chat room type

"listusers" and return key.

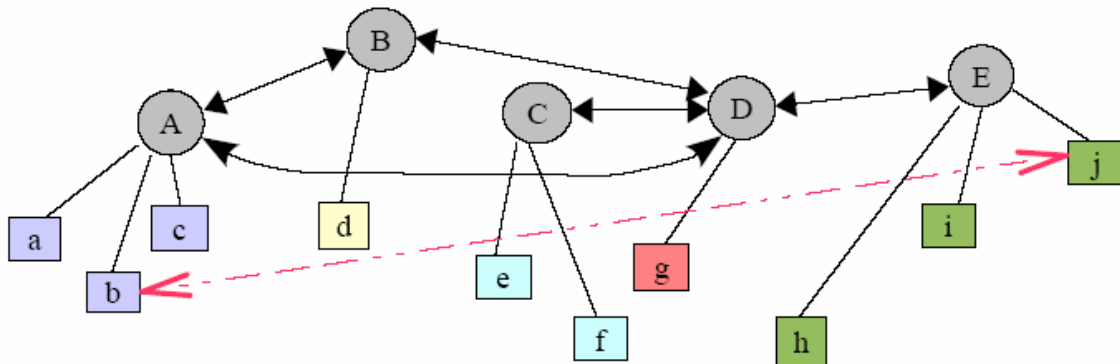
Remember you have to follow step 10 before following this step listusers return

10. To change Chat room follow step 10.

11. To exit please write exit.

For this project we have to follow two protocols one should be UDP and other should be multicasting protocol DVMRP. I am explaining all of them one by one.

Protocol Specification



Explanation of the Scenario

This project is just an extension of what we have developed in the project2. Here the circle in the above diagram represents chat servers, the square elements represent the user nodes connected to the chat servers. Each user node consists of user agent as well as end user modules developed in project 2.

In project 2, peer-to-peer communication was disabled, which will be enabled in project 3. In addition, the servers will have the capability to communicate with each other. Teams are required to come up with a server discovery protocol so that servers can discover any available nearby servers. When a nearby server is discovered, the server is required to distribute its list of attached users among its neighbors in order to facilitate fault tolerance in the design. In addition, when a nearby server is discovered, its details are to be passed to the attached nodes as well.

When a server fails, nodes connected to the server must be able to detect the failure and reattach itself to one of the nearby servers. If they are not able to reconnect to any other server, the nodes must die. Later when the crashed server becomes online, it may query its neighbors (it can dump the neighbor list in a file periodically) to find out what nodes were under its jurisdiction, and then may contact those nodes asking them to reattach

them to itself. In addition, the nodes attached to a server may search for nodes connected to distant servers by querying its own server. That server should then query its neighbor server nodes and so on until either the requested node information is found or no such node exists in the connected server graph. The requesting node then can establish a direct connection to the far-away node via peer-to-peer connection as implemented in project 1.

User nodes a, b, c are attached to server A. User nodes d are attached to server B User nodes e, f are attached to server C. User nodes g are attached to server D. User nodes h, i, j are attached to server E. Server A prevents alias name collision under its domain. Additional requirement will be that name collision now must be enforced across multiple chat rooms as well. Hence nodes a, b and c can now be named hierarchically in a globally unique fashion. Global names of a, b, c will now be A.a, A.b and A.c. Same thing we follow for other i.e. B.d, C.e, C.f, D.g, E.h E.i, E.j Of course this necessitates that the server themselves are named uniquely. This can be enforced during the server discovery phase, when a server discovers a new neighbor, it confirms that the server names are unique; in case of a collision one of the server must automatically change its alias name. A.a, A.b and A.c can still engage in chat room communication using multicast.

Case 1. Now suppose A.b wants to talk to node j, it send a search request to its parent server A, A in turn sends the search request to its neighbors B and D, D propagates the search request to its neighbors and so on. Finally E finds j listed as its user and it responds back to D which responds back to A and it sends the search result to b, now b can send messages directly to j using send command.

Case 2. Now suppose C.e wants to talk to node a, it send a search request to its parent server C, C in turn sends the search request to its neighbors D, D propagates the search request to its neighbors C and so on. Finally A finds a listed as its user and it responds back to B, B to D and which responds back to C and it sends the search result to e, now e can send messages directly to a using send command.

There are many more cases in addition to this.

When we talk about the fault tolerant routing we always talking about the adaptive routing Adaptive routing describes the capability of a system, through which routes are characterized by their destination, to alter the path that the route takes through the system in response to a change in conditions. The adaptation is intended to allow as many routes as possible to remain valid (that is, have destinations that can be reached) in response to the change.

For Fault-tolerance: DVMRP

For making our chat fault-tolerable we will use the specification used by the Distance Vector Multicast Routing Protocol (DVMRP).

DVMRP is the original IP Multicast routing protocol. It was designed to run over both multicast capable LANs (like Ethernet) as well as through non-multicast capable routers.

In this case, the IP Multicast packets are "tunneled" through the routers as unicast packets. This replicates the packets and has an effect on performance but has provided an intermediate solution for IP Multicast routing on the Internet while router vendors decide to support native IP Multicast routing.

The DVMRP Statement

```
dvmrp yes | no | on | off [ {  
  interface interface_list [ {  
    enable | disable ;  
    metric metric ;  
    threshold threshold ;  
    ratelimit rate ;  
    advertise network metric metric ;  
    advertise network mask mask metric metric ;  
    advertise network masklen number metric metric ;  
  } ] ;  
  tunnel host lcladdr local_address [ {  
    enable | disable ;  
    metric metric ;  
    threshold threshold ;  
    ratelimit rate ;  
    advertise network metric metric ;  
    advertise network mask mask metric metric ;  
    advertise network masklen number metric metric ;  
  } ] ;  
  traceoptions trace_options ;  
} ] ;
```

The **dvmrp** statement enables or disables the DVMRP protocol. If the **dvmrp** statement is not specified the default is **dvmrp off**; If enabled, DVMRP will default to enabling all interfaces that are multicast capable. These interfaces are identified by the IFF_MULTICAST interface flag.

Note: only one multicast routing protocol can be configured on a port at a time.
The options are as follows:

interface *interface_list*

Enables or disables DVMRP on this interface or list of interfaces. Currently, the gated implementation of DVMRP cannot detect when there are multiple addresses configured on the same physical interface. It will send DVMRP messages to each logical network. This should be fixed in a future release.

The possible parameters are:

disable

Specifies that DVMRP packets received via the specified interface will be ignored. The default is to listen to DVMRP on all multicast capable interfaces.

enable

This is the default. This argument may be necessary when **disable** is used on a wildcard interface descriptor.

metric

This command provides a way to configure the metric on a physical port or tunnel. This metric will be added to all routes that are learned via this interface.

threshold

This command provides a way to configure the threshold on a physical port or tunnel. A packet will not be forwarded out this interface unless the TTL in the packet exceeds this threshold.

ratelimit

This command provides a way to limit the rate of multicast traffic on an interface. It is an option in the XEROX PARC IP Multicast kernel (Release 3.3 or greater).

advertise

The advertise line is used to advertise a particular network out this particular interface only. This is somewhat dangerous but is necessary for joining non-DVMRP multicast regions to DVMRP regions. It can be specified by network number and netmask or network number and prefix length. If no netmask is specified, the default netmask for the class is assumed.

tunnel *host lcladdr local_address*

The tunnel statement uses the same options as the **interface** statement.

traceoptions *trace_options*

Specifies the tracing options for DVMRP. (See Trace Statements and the DVMRP specific tracing options below.)

Tracing options

Packet tracing options (which may be modified with **detail**, **send** or **recv**):

packets

All DVMRP packets.

probe

DVMRP Router Probe packets

report

DVMRP Route Report packets

mapper

DVMRP Neighbor and Neighbor 2 packets

prune

DVMRP Prune packets

graft

DVMRP Graft and Graft Ack packets

DVMRP is a dense-mode multicasting protocol and therefore uses a broadcast and prune mechanism. The protocol builds a source-rooted tree (SRT) in a similar way to PIM dense mode. DVMRP routers flood datagrams to all interfaces except the one that provides the shortest unicast route to the source. DVMRP uses pruning to prevent unnecessary sending of multicast messages through the SRT.

A DVMRP router sends prune messages to its neighbors if it discovers that:

The network to which a host is attached has no active members of the multicast group. All neighbors, except the next-hop neighbor connected to the source, have pruned the source and the group. When a neighbor receives a prune message from a DVMRP router, it removes that neighbor from its (S,G) pair table, which provides information to the multicast forwarding table. When a host on a previously pruned branch attempts to join a multicast group, it sends an IGMP message to its first-hop router. The first-hop router then sends a graft message upstream.

Identifying Neighbors

In this implementation of DVMRP, a *neighbor* is a directly connected DVMRP router. When you enable DVMRP on an interface, the associated VR adds information about local networks to its DVMRP routing table. The VR then sends probe messages periodically to learn about neighbors on each of its interfaces. To ensure compatibility with other DVMRP routers that do not send probe messages, the VR also updates its DVMRP routing table when it receives route report messages from such routers.

Advertising Routes

As its name suggests, DVMRP uses a distance-vector routing algorithm. Such algorithms require that each router periodically inform its neighbors of its routing table. DVMRP routers advertise routes by sending DVMRP report messages. For each network path, the receiving router picks the neighbor advertising the lowest cost and adds that entry to its routing table for future advertisement.

The cost, or metric, for this routing protocol is the hop count back to the source. The hop count for a network device is the number of routers on the route between the source and that network device. Table 6 shows an example of the routing table for a DVMRP router.

Table 6: Sample Routing Table for a DVMRP Router

Source Subnet	Subnet Mask	From Router	Metric	Time Before Entry Is Deleted from Routing Table	Input Port	Output Port
143.2.0.0	255.255.0.0	143.32.44.12	4	85	3/0	4/0, 4/1
143.3.0.0	255.255.0.0	143.2.55.23	2	80	3/1	4/0, 4/1
143.4.0.0	255.255.0.0	143.78.6.43	3	120	3/1	4/0, 4/1

The DVMRP router maintains an (S,G) pair table that provides information to the multicast forwarding table. The (S,G) pair table is based on:

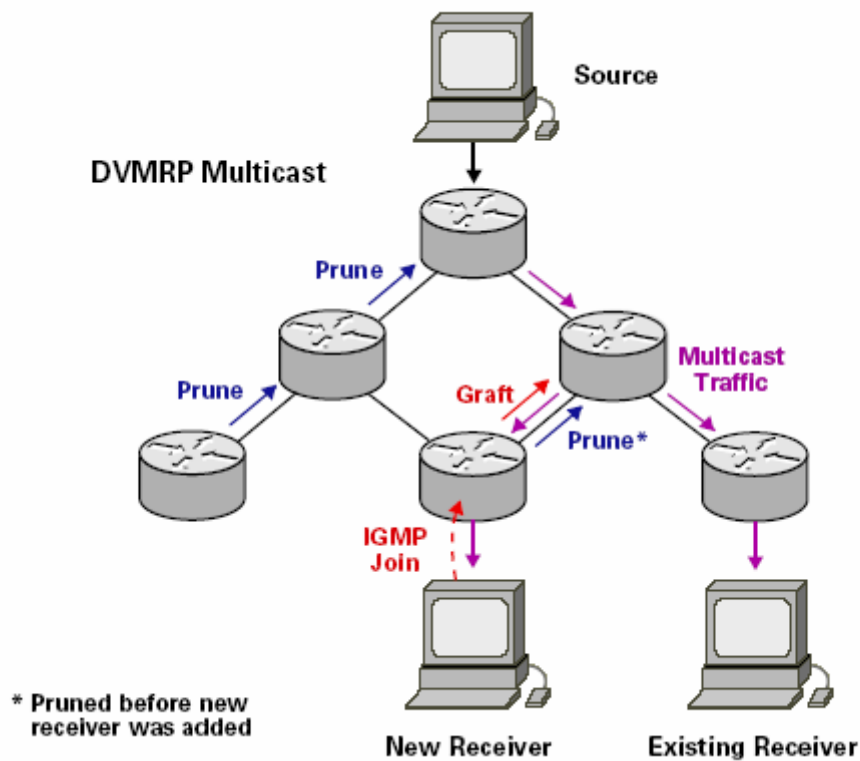
Information from the DVMRP routing table

Information learned from prune messages

If IGMP and DVMRP are on the same interface, group information learned from IGMP The (S,G) pair table includes a route from each subnetwork that contains a source to each multicast group of which that source is a member. These routes can be static or learned routes. Table 7 shows an example of the (S,G) pair table for DVMRP.

Table 7: Sample DVMRP (S,G) Pair Table

Source Subnet	Multicast Group	Time Before Entry Is Deleted from Routing Table	Input Port	Output Port
143.2.0.0	230.1.2.3	85	3/0	4/0, 4/1
	230.2.3.4	75	3/0	4/0, 4/1
	230.3.4.5	60	3/0	4/1
	230.4.5.6	90	1	4/0
143.3.0.0	230.1.2.3	80	3/1	4/0, 4/1



IGMP for Discovering

IP hosts use Internet Group Management Protocol (IGMP) to report their multicast group memberships to neighboring servers. Similarly, multicast servers, use IGMP to discover which of their hosts belong to multicast groups.

IGMP Operation

IGMP exchange the following types of messages between servers and hosts:

- Query
- Report
- Reavegroup

Query

A multicast server can be a querier or a nonquerier. There is only one querier on a network at any time. Multicast servers monitor queries from other multicast servers to determine the status of the querier. If the querier hears a query from a server with a lower IP address, it relinquishes its role to that server.

- General queries to the all-hosts group address (224.0.0.1)
- Specific queries to the appropriate multicast group address

Report

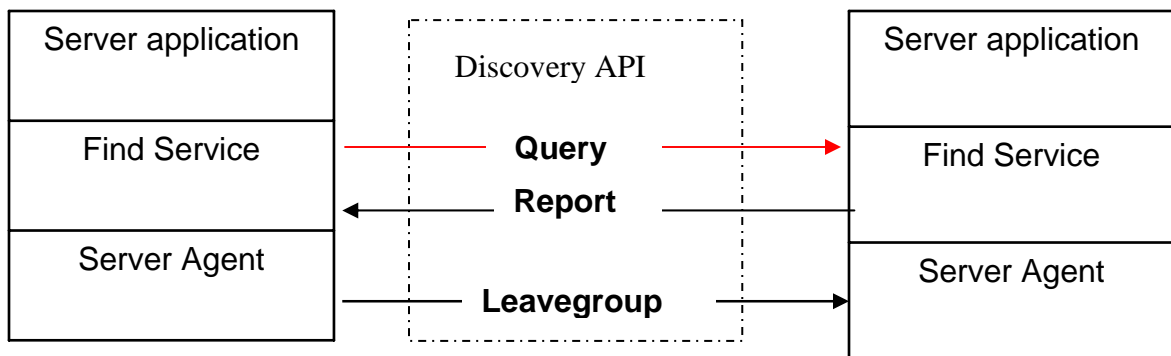
When a host receives a group membership query, it identifies the groups associated with the query and determines to which groups it belongs.

The host multicasts a group membership report to the group addresses. When a multicast server receives a report, it adds the group to the membership list for the network

If the server does not receive any reports for a specific multicast group within the *maximum response time*, it assumes that the group has no members on the network. The server does not forward subsequent multicasts for that group to the network.

Leavegroup

When a host leaves a group, it sends a leave group membership message to multicast servers on the network. A host generally addresses leave group membership messages to the all-servers group address (224.0.0.2).

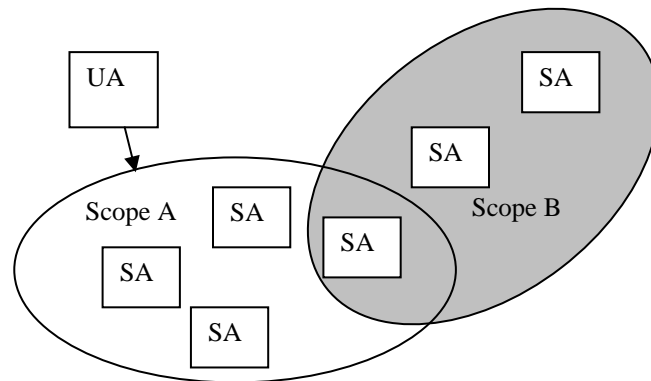


Multicast Group A

Multicast Group B

The Server application uses a Server Agent Query that the remote Server Agent responds to with a Report.

Scope of Sever Agents (SA) and User Agents(UA)



User Agents (UA) can only locate services within the scopes to which they have access.