mDNS: A comprehensive multicast session directory service

Piyush Harsh*†and Richard Newman‡

November 22, 2008

*Computer and Information Science and Engineering, University of Florida, Gainesville, FL USA

†Contact Address: Piyush Harsh, E301 CSE Building; PO Box 116120; Gainesville, FL 32611; Tel: (352) 392-7298; Fax: (352) 392-1220; Email: pharsh@cise.ufl.edu

‡Computer and Information Science and Engineering, University of Florida, Gainesville, FL USA

Abstract

IP multicast is generally accepted as more efficient mode of data distribution for live streaming data to large group of subscribers. Yet it lacks universal deployment by ISPs around the world. Lack of demand from end users and network complexity are most cited reasons for its low penetration. With SSM and IGMP v3, most of the complexity has been removed from the network stack. Yet lack in end users enthusiasm towards this technology remains.

Absence of DNS like service for seamless address translation for a URL like multicast session name and a global search infrastructure that allows end user to search for live sessions of interest could be perceived as major roadblocks towards widespread deployment. In this article we present a new service scheme that has been designed in order to overcome these roadblocks. mDNS is a truly global, hierarchical and distributed service infrastructure. Our design leverages existing DNS infrastructure in supporting URL like names for multicast sessions.

Further our scheme uses smart keyword routing scheme in order to speedup global session search. Hash based keyword space distribution among participating servers lends to equitable workload distribution. And tight integration with geo-coding allows us to tag sessions with geographically specific data. This in turn allows some interesting new services and usage for multicast streams to become apparent.

1 Introduction

IP Multicast was invented by Steve Deering as part of his PhD dissertation in 1984. Since then more than two decades have elapsed and still IP multicast has not seen widespread deployment on a global scale by ISPs as anticipated in the research community. Even though network researchers are in agreement on the efficiency of content distribution to a group of receivers from a single source via IP multicast, still demands from end users for this technology is lacking. It could be argued that usability and ease of content discovery plays a significant role in the general acceptance of such technology. Domain Name Service (DNS) definitely helps improve usability of web-based portals by allowing users to remember easily recallable mnemonics such as www.yahoo.com and www.microsoft.com instead of more difficult dotted decimal notation 209.191.93.52 and 207.46.192.254. DNS also allows an user to bookmark these names for later translation by the web browser using DNS. These mnemonics allows for a level of indirection and spare the end-user with arcane details of changes in server's IP address and its location to name a few. Further existence of search engines like $Google^{©}$ and $Yahoo^{©}$ allows an end user to easily search for content s/he is interested in quickly.

Another argument that is usually cited is the inherent network complexity that core routers must incorporate to support multicast. While that could have been true for Any Source Multicast or ASM mode of IP multicast where the burden of multicast source discovery lies on the network using MSDP and complex membership tree construction and maintenance using rendezvous points (RP), much of the complexity has been removed in the source specific multicast (SSM). Much of the complexity such as source discovery now lies with the end users. The standardization of IGMPv3 has been a major step towards shifting some of the complexity from core networks to the edge nodes and end users.

Therefore it becomes apparent that in order for IP multicast to be successful, one has to come up with scheme to allow end users to easily discover multicast sources. Keeping in mind the fact that most of multicast sessions are not long lived entities, and multicast addresses are also not permanently assigned and in fact are shared among several services, such a proposal must also allow source discovery in real time. Additionally, if such a service could allow assignment of URL like mnemonics to multicast sessions, it could go a long way in improving usability of multicast. Further such a scheme could also allow for sessions to be bookmarked.

In this article we will present our scheme named "mDNS" for such a service that provides all the above features and some more. In the next few sections, we would provide a brief summary of various competing schemes that have been proposed in literature and we would argue about their limitations. We will describe each of mDNS components design in details, presenting formal analysis into fair distribution of workload and supported operations' complexity wherever relevant. Extensive pseudocodes are also included to precisely describe the behavior of each unit and it's interaction with other components in the overall mDNS architecture scope. We would also describe in detail, addressing and routing algorithm used in our scheme to speedup

content retrieval and also keeping records duplication at its minimal.

Another important feature built in MSD servers is auto-geocoding capability. Using this feature, MSD servers all over the Internet tag multicast sessions with relevant geo-spatial information such as (latitudes, longitudes) as part of the registration data for the life of that session. This enables mDNS to foster in a new communications paradigm where information can be retrieved based not just on "content" but also on that information's geo-spatial location.

2 Related Work

3 mDNS Hierarchy Construction

The global mDNS architecture is designed keeping scalability and phased deployment over a period of time into consideration. Just like the Internet is made up of independent administrative network domains interconnected with each other, mDNS design similarly is made up of several mDNS domain interconnected to each other. mDNS domain could be any traditional network domain that houses a DNS (Domain name service) server. In fact, mDNS has been designed to be tightly integrated with the existing DNS hierarchy. A typical mDNS domain consists of a DNS server, MSD (multicast session directory) server(s) and a URS (URL Registration) server. Figure (1) shows a typical mDNS domain. The hierarchy of mDNS design is essentially similar to DNS hierarchy. We have proposed to include an additional record "MCAST" in a domain's DNS server that contains essential information needed to facilitate inter and intra domain communication between various mDNS components and seamless data retrieval by multicast session end users in any domain under mDNS global hierarchy. Disregarding the exact DNS records format, our proposed "MCAST" record will contain these essential details:

```
@MCAST
{
    ANYCAST=a.b.c.d
    CMCAST=233.[ASN Byte1].[ASN Byte2].XXX
    PMCAST=233.[ASN Byte1].[ASN Byte2].???
    URS=x.y.z.w
}
```

We have deliberately omitted port numbers as we envision our effort to be standardized and assigned well known port numbers by IANA eventually. In the "MCAST" record shown above, "ANYCAST" refers to the IP address that would enable communication channel to be opened with one of possibly several MSD servers running in a typical mDNS domain. When there are several MSD servers running, one of them is elected as a designated MSD server, others running in standby mode if in future the designated MSD server goes offline,

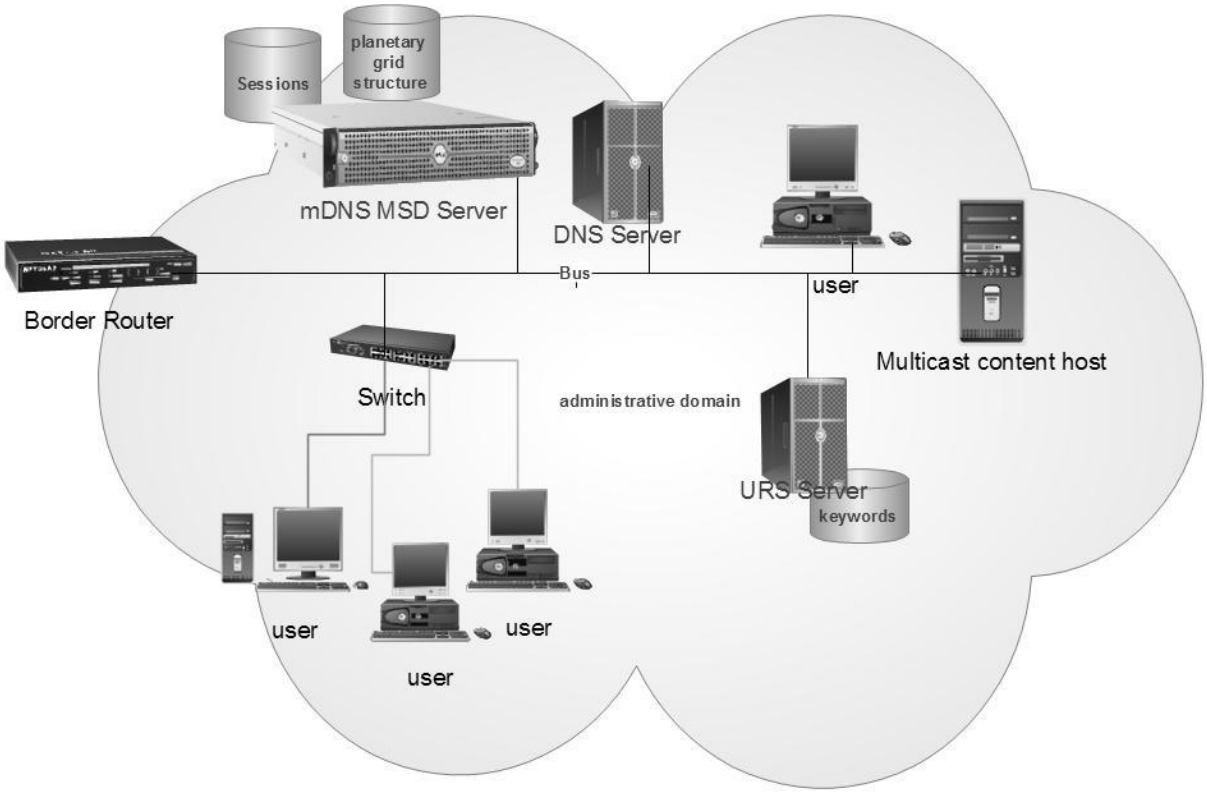

Figure 1: a typical administrative domain in mDNS architecture

to take over the responsibility. "CMCAST" and "PMCAST" are globally scoped GLOP multicast channels that serve as communication channels between the designated MSD server in a domain with designated MSD servers in any child domains and its parent domain. Since these addresses are from GLOP addressing block, these have to be ISP assigned and also ensures that multicast communication on these channels can cross domain boundaries (unlike administratively scoped addresses). "URS" entry refers to the IP address of the URS server in that domain.

Assuming that every domain that has a DNS server running, and that DNS servers know about their parent DNS server's connection details, has mDNS components installed and running as well, figure (2) shows an example mDNS hierarchy in operation. The figure also shows various communication links in operation between various domains shown. Because of inclusion of "MCAST" record in DNS servers, the communication link setup is almost seamless. It is to be noted that as far as mDNS functionality is considered, full Internet wide deployment is not mandated, although full scale deployment certainly demonstrates the true capability of the proposed scheme. In the figure shown above, PMCAST channel details of any child domain is essentially same as CMCAST channel details of its immediate parent domain. In the global scale, the hierarchy would terminate at the root DNS, which is marked by "TLD" (Top Level Domain) in the

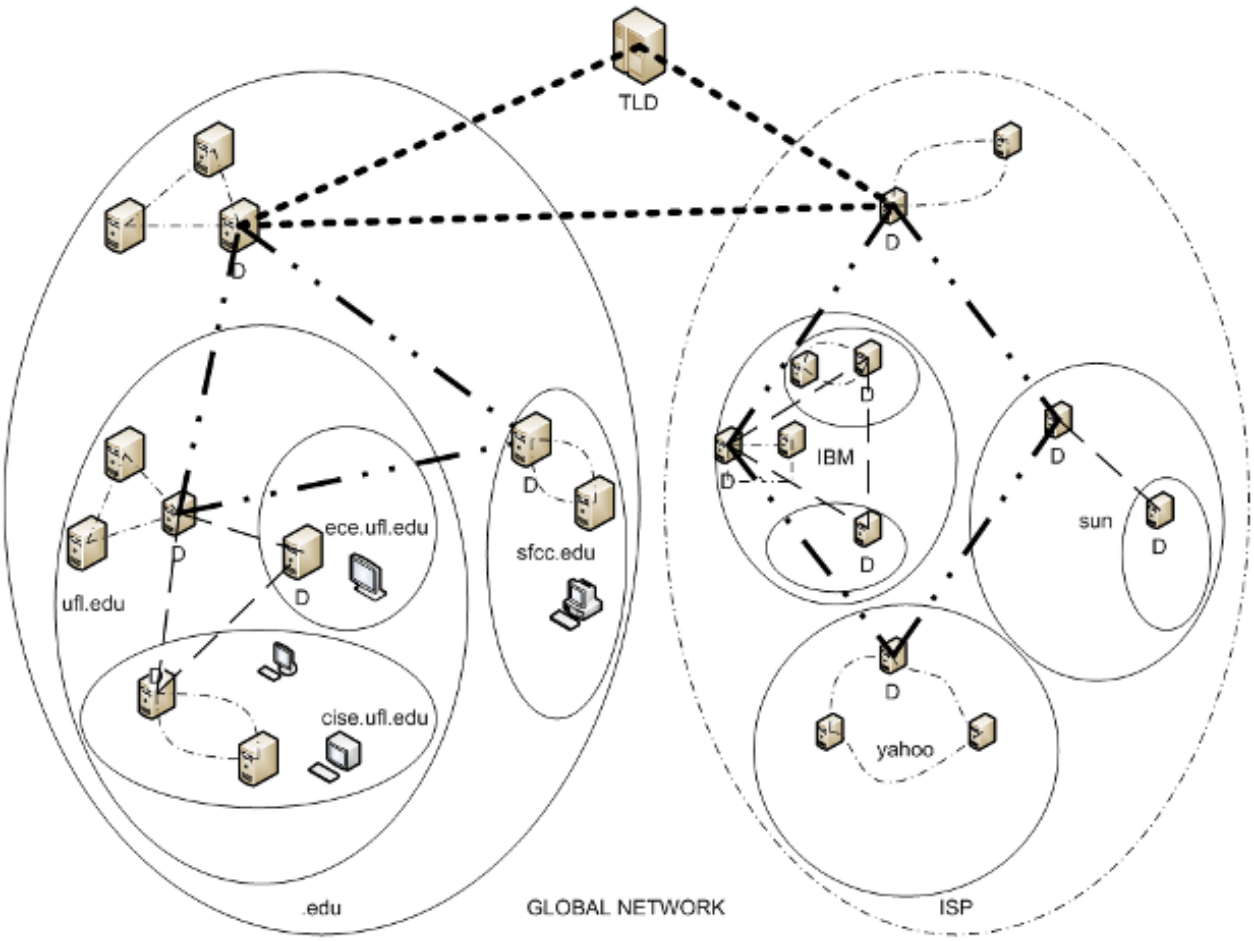

Figure 2: a typical mDNS hierarchy

shown graphic. But in partial deployment scenarios, the hierarchy could terminate at any domain and the next higher DNS could be marked as "null". In that case however the session retrieval/search is restricted to the participating domains only. However specific multicast session connection using URL strings can be retrieved from disconnected mDNS domains. This ability will be explained in greater detail later.

Whenever a MSD server becomes online, it joins MSD-LOCAL-MCAST channel. This is a administratively scoped multicast channel that is known a priori. It then starts a local leader election algorithm that runs the whole time the server is online. Election messages are sent to other prospective MSD servers in the same domain listening in on MSD-LOCAL-MCAST channel on a periodic basic. Provisions have been made to ensure minimal service disruption, in case the designated MSD server (winner of the leader election) goes offline. Each MSD server has a priority value that can be set between 1 and 3 by the system administrator when MSD servers are first started. Each MSD server also maintains several internal status flags, one such flag is "param.isDesignate" that is initiated automatically to "true" when the servers starts.

Algorithm (1) shows the basic steps that is carried out by the election thread that runs for the entire duration the server is running. Clearly it is a soft-state refresh algorithm, and thus ensures minimum service

disruption in case the designated MSD server fails. The job of the designate-MSD server is then taken up by one of the remaining MSD servers (next winner) based on the outcome of the ever-running algorithm. Of course that is when there are multiple MSD servers running.

Algorithm 1: MSD Leader Election Algorithm

```
1  begin
2      set SEED = random(0, 100,000)
3      set HASH = MD5(SEED)
4      set lastSentTime = CurrentSystemTime
5      while loop flag = true do
6          if param.isDesignate == true and CurrentSystemTime - lastSentTime ≥ cycleTime then
7              set lastSentTime = CurrentSystemTime
8              send "ELECT " + priority + " " + HASH on MSD-LOCAL-MCAST channel
9          end
10         set Socket timeout to cycleTime/2
11         receive incoming message on MSD-LOCAL-MCAST channel
12         if incoming message is type ELECT and has same HASH value then
               // ignore message and do nothing
13         end
14         if incoming message is type ELECT and has different HASH then
15             if priority < remotePriority then
16                 set param.isDesignate = false
17             else
18                 if priority == remotePriority then
19                     if hash < remote HASH then
20                         set param.isDesignate = false
21                     end
22                 end
23             end
24         end
25         if no remote ELECT message has been received for 2*cycleTime then
26             set param.isDesignate = true
27         end
28     end
29 end
```

The designate-MSD server also joins "PMCAST" and "CMCAST" multicast channels. These channels allows communication between designate-MSD servers in the parent's domain and in child domain(s) thus completing the hierarchy construction. In the next few sections we will describe in details the various components that make up a MSD server and an URS server.

4 URS Server

Enabling easy to bookmark and rememberable session names just like any typical domain name, for example yahoo.com, is the principal goal on an URS server. The name stands for "URL Registration". Its functions include maintaining uniqueness among session names registered with it. Within any domain there is only one URS server running. The only communication it permits is connection oriented TCP unicast. It supports

session name registration and query. It maintains every registered name in priority queue that is prioritized on a multicast session's expiration time. Hence at any given point in time, the URS database contains names of sessions that have not expired at that instance.

Multicast session creator is required to register the session details and choose a representative name for that session. The URS server notifies the session registration agent in case the representative name has already been used, following which it must provide another name and try the whole URS registration sequence over again. Let us try to understand what URS server achieves through an example:
Let us assume that there is a DNS server in a domain whose canonical name is cise.ufl.edu. Further this DNS server has the additional "MCAST" record added in its records database. And this administrative domain is also running a MSD server and an URS server. Next suppose there is a multicast session that originates from within this domain and the session creator registers the session details with URS server and MSD server. Each session record maintained at URS server has this structure:

```
@URS-RECORD
{
    Expiration Time
    URS Keyword
    Channel IP
    Channel Port
    Unicast Host IP
    Unicast Host Port
    Session Scope
    Geographic Location name
    Geographic Latitude
    Geographic Longitude
}
```

The last two components are generated by automatic geocoding of address provided by the session creator during URS name registration process. Now let us say that the session that originated from within cise.ufl.edu domain, registers "algo-club" with the URS server successfully. Hence as a result, under mDNS scheme that particular session can now be accessed by a much more usable URL "**mcast.cise.ufl.edu/algo-club**".

Figure (3) shows the typical mDNS URL resolution process that enables an end user to access the multicast content. The user types in the mDNS URL, in this example that was "mcast.cise.ufl.edu/algo-club" in mDNS capable multicast application. The user software first resolves "mcast.cise.ufl.edu" in the typical DNS domain name resolution sense. Once the MCAST record is available at the end user software, it then knows the connection details for contacting the URS server. It next contacts the URS server and queries for "algo-club" session. The URS server searches in its database and if found returns the record for "algo-club" to the requesting user agent. Now the user application has all the necessary details to join the desired multicast session. It should be clear that, for mDNS URLs to be resolved correctly, it is not necessary

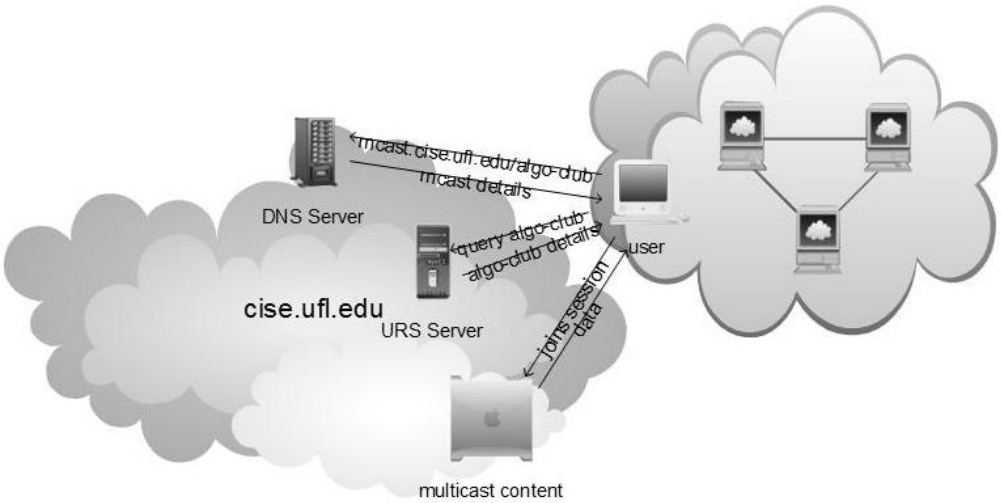

Figure 3: mDNS URL resolution process

that domains in the path to the correct mDNS domain must themselves be mDNS domains. But a session in a disconnected domain can not be searched using typical mDNS search process outside that domain. We will describe that next.

5 MSD Server

MSD (Multicast Session Directory) server is the most important and complex component of the mDNS architecture. It not only maintains session directory of all sessions that originates in its own domain but also maintains a share of the globally scoped sessions depending on the keyword hash distribution scheme. The main components that make up a MSD server are:

- Election Thread
- Local Session Database
- Global Session Database
- MSD-LOCAL-MCAST Thread
- PMCAST Thread
- CMCAST Thread
- Keywords Routing Table Manager
- Geo-Spatial Session Database
- Search Handler

Apart from these major components that help MSD servers perform actions that form the basic umbrella of services provided under mDNS architecture, there are few other modules such as “Lazy-Sync Thread”,

"Cache Manager" to name a few that functions to increase performance and improve resilience of the overall mDNS infrastructure. The function of the "Election Thread" has been discussed in previous sections. Below we will provide detailed information on some of the remaining modules and algorithms that they run.

5.1 Database Subsystem (Global / Local)

MSD servers maintain 3 separate session databases, namely "local", "global" and "geo-tagged" database. The sub-components that constitute "local" and "global" multicast session databases are almost similar. Each maintain a hash table of keywords and associated count of sessions for each keyword against all sessions that are stored with it. Actual session details are stored as records that have this format:

```
@session-record
{
    Session Expiration Time
    Session Start Time
    Keyword
    URS Registered Representative Name
    Channel IP
    Channel Port
    Host Unicast IP
    Host Port
    Stream Scope
    Geographic Location Common Name
    Latitude
    Longitude
}
```

Each database has a "maintenance" thread that is executed periodically and that purges all those session records whose expiration time has elapsed. Figure (4) shows the structure of "global" database in MSD

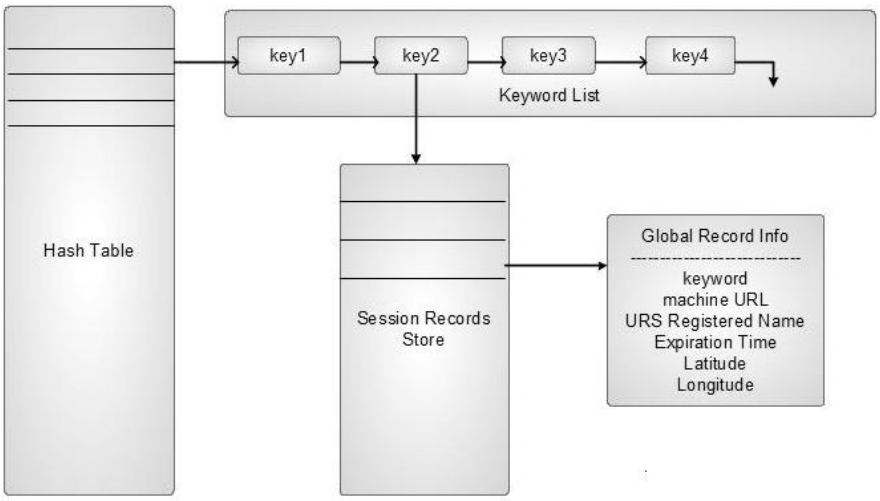

Figure 4: mDNS Global Database Design

server. It shows a 3 level data structure. When a search keyword is received, first the keywords hash table

metadata is checked. If no such entry is found, then there is no point to proceed further. Keywords metadata is not shown in the figure. If an entry is found in the metastore, that keyword is hashed to compute the table index. Since multiple keywords could possibly hash onto same location, the overflow is maintained as a linked list of such keywords, which is the level 2 structure shown in the figure. The keyword entry in the overflow linked list then points to the level three data structure that is another linked list where the session records are stored. All such session records form the candidate sessions to be returned against that particular keyword search request. Depending on the search parameters, additional computation or cross-referencing against the "geo-tagged" database entries could be performed. The "local" database is very similar in construction to "global" database with the difference in the record structure stored in the session records store. The expanded session entry format is already shown above.

Since the global session records might need to be migrated between designate-MSD servers between different domains depending on the routing table changes (explained later), global session database record maintains minimal information needed compared to the local session database records. This design decision was made to keep amount of data to be moved at a minimum. The machine URL and "URS Registered Name" enables session expanded record to be retrieved from the original MSD server where the session was first registered. Session expiration time has been included to enable "maintenance" threads to cull such sessions which have become stale. Latitude and longitude information is maintained to enable cross-referencing against entries in geo-tagged database (if needed) depending on the search query criteria.

5.2 Database Subsystem (Geo-Tagged)

Earth geographic locations can be addressed precisely using latitude and longitude coordinates. Latitudes vary from -90°to +90°along south-north corridor. Similarly longitudes vary from -180°to +180°along west-east corridor. Latitudes are parallel to each other and are equidistant. Every degree separation between latitudes equals 110.9 km in ground distance. The distance relationship between longitudes is not that straightforward because they converge at the poles. This relationship is further complicated as earth is not a perfect sphere.

$$\frac{\pi}{180^\circ} \times \cos\phi \times \sqrt{\frac{a^4 \cos(\phi)^2 + b^4 \sin(\phi)^2}{(a\cos\phi)^2 + (b\sin\phi)^2}} \tag{1}$$

Equation (1) shows the east-west distance between every degree change in longitudes at latitude ϕ with $a = 6,378,137$m and $b = 6,356,752.3$m. Figure (5) shows the earth coordinate system and the schematic of the geo-tagged database. Under the current grid map where major lines being latitudes and longitudes, each being 1°apart, earth can be mapped into 180x360 grid space. Since almost 70% of earth surface is covered by water, 70% of the grid locations naturally would map to water bodies. Of the remaining 30% of landmass, research shows only 50% of land area is inhabited by human. Therefore, we foresee only 15% of

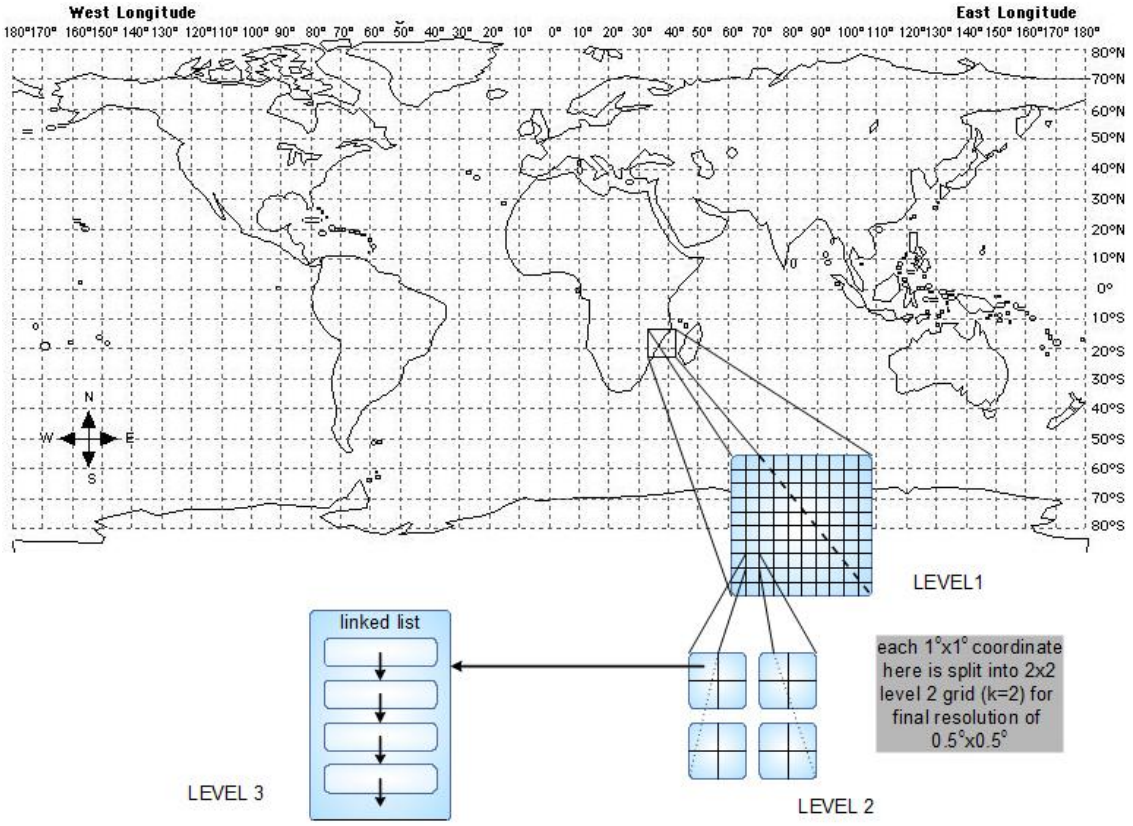

Figure 5: Geo-Tagged Data Structure

full grid locations to be ever used to group multicast sessions belonging to such grid position depending on their geo-tags. Hence sparse-matrix implementation of planetary grid is justified.

5.2.1 Construction

Since a 1°x 1°grid at equator represents an area of 111.3x110.9 km^2, it might be necessary to further subdivide the area into smaller zones. The grid subdivision or node branching factor "k" determines how a larger grid area is subdivided. The depth of tree and choice of "k" depends on the final areal resolution desired. For instance, if an arial resolution of at most 5 x 5 km^2 is desired, and let us say that the branching factor is 2, i.e. k = 2, then a tree with height 5 would result in an areal resolution of 3.48 x 3.47 km^2 at equatorial plane. In general, areal resolution at depth "n" for branching factor "k" is governed by these equations below:

$$\frac{110.9}{K^n} km \tag{2}$$

$$\frac{\pi}{180^\circ} \times \cos\phi \times \sqrt{\frac{a^4 \cos(\phi)^2 + b^4 \sin(\phi)^2}{(a\cos\phi)^2 + (b\sin\phi)^2}} \times \frac{1}{K^n} km \tag{3}$$

Equation (2) governs the north-south resolution at tree depth "n" and equation (3) governs the east-west resolution at same depth at latitude ϕ°.

Any session that gets stored at either "global" or "local" databases also keeps a corresponding geo-reference in the "geo-tagged" database. These references are maintained at correct grid location in the level 0 structure and at correct leaf linked-list in the tree rooted at corresponding level 0 grid position. The "maintenance" threads while removing stale sessions from "global" and "local" databases removes the corresponding reference from "geo-tagged" database as well. We will see how maintaining this additional structure allows new service paradigm to be supported that was previously not possible in later sections.

5.3 Search Routing

One of the goals of mDNS is to reduce unnecessary duplication of session information all over the Internet. In mDNS, every designated-MSD server manages and maintains the overall keyword space cooperatively and in a systematic manner. At any point in time, there is only one MSD server in the mDNS hierarchy that is responsible for storing the session record corresponding to any given keyword. Therefore in order for session searches to retrieve records based on keywords, and for global session registrations to reach desired MSD server that is assigned to store session records against that particular keyword, a keyword routing scheme is needed.

The keyword based routing is performed using MD5 hash of a keyword. 128 bits hash result is used in the same sense as unicast IP addresses for IP routing. mDNS routing is very similar to IP routing in the sense that it is also prefix routing. The hash space assignment to MSD servers is done keeping fair workload distribution as guiding principle. In order to explain the routing table construction, let us consider

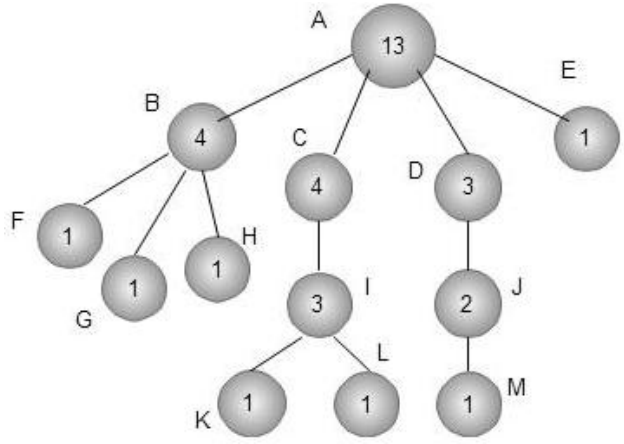

Figure 6: Example mDNS domain hierarchy

an example mDNS domain hierarchy as shown in figure (**??**). The count inside each node refers to the number of designated-MSD servers in the tree rooted at that particular node including itself. This count is transmitted by each node to its parent node using "PMCAST" multicast channel. The root node, may or may not be a MSD server. An example of such a scenario would be when the root node happens to be in

top level domain (TLD) and maintained by IANA. In such a scenario, the root node just helps in hash space assignment to child nodes in fair manner and the count at the TLD domain level may not include the root node. Keeping things in perspective, this scenario would result in root count being 12 and not 13 in the sample hierarchy in figure (6).

Algorithm 2: Routing: Hash Space Distribution Algorithm

```
1  begin
2      set array = list of node counts from child domains
       // in figure ?? array = [4, 4, 3, 1]
3      set sum = 1
4      add to sum <- contents of array
       // in our example figure, sum = 13
5      set significantBits = minimum power of 2 that is ≥ sum
6      set float AddRange = 2^significantBits
7      set float Ratio = AddRange ÷ sum
8      set int Number2Allot = (int)Ratio
9      set float FractionRemaining = Ratio - (int)Ratio
10     set CurrentValue = 0
11     print SelfHashRange: CurrentValue to Number2Allot -1
12     set Currentvalue += Number2Allot
13     for i for each array location do
14         Number2Allot = (int)ratio×array[i]
15         FractionRemaining += ratio×array[i] - (int)ratio×array[i]
16         while FractionRemaining ≥ 0.99999 do
17             Number2Allot++
18             FractionRemaining -= 0.99999
19         end
20         print array[i]: HashRange: CurrentValue to CurrentValue + Number2Allot -1
21         CurrentValue += Number2Allot
22     end
23 end
```

Algorithm (2) shows how the hash-space is distributed proportionately by a parent node for its children nodes in order to balance the workload across participating MSD-designate nodes. The hash-space information is send to child nodes on "CMCAST" channel by the parent nodes. The routing table essentially contains hash-space divisions along with what channel to forward the search / registration request to. It also maintains supporting information such as significant bits (leading bits whose value determine where to route to next), Hash Range Start Value and End Value. These supporting information are used to quickly determine whether some actions are needed based on address range information coming on the "PMCAST" channel.

Table 1 shows a sample routing table maintained at root node A. Given a 128 bit keyword hash, using the outing table at a MSD server, one can determine on what channel to forward that request to. If no matching table entry is found, it is forwarded on "PMCAST" channel provided the forwarding node is not the root node. In that case the request is dropped and an error message is sent to the requesting node.

Table 1: Routing table at node A

Hash Start	Hash End	Channel Out	Node ID	Is Orphan
$0000\overbrace{xxx...xxx}^{124}/4$	$0000\overbrace{xxx...xxx}^{124}/4$	MSD-LOCAL-MCAST	Self ID (A)	NO
$0001\overbrace{xxx...xxx}^{124}/4$	$0101\overbrace{xxx...xxx}^{124}/4$	CMCAST	Node B	NO
$0110\overbrace{xxx...xxx}^{124}/4$	$1010\overbrace{xxx...xxx}^{124}/4$	CMCAST	Node C	NO
$1011\overbrace{xxx...xxx}^{124}/4$	$1101\overbrace{xxx...xxx}^{124}/4$	CMCAST	Node D	NO
$1110\overbrace{xxx...xxx}^{124}/4$	$1111\overbrace{xxx...xxx}^{124}/4$	CMCAST	Node E	NO

"Is Orphan" entry in the routing table indicates whether that particular routing entry is in use or not. Routing table are maintained on a periodic basis, and in order to minimize routing instability, current node count at a particular node is forwarded to parent node guided by the algorithm 3 specified next.

A route entry can be set orphaned if the child node to which it was assigned gets disconnected and its parent node fails to receive periodic refresh messages for a while, and the routing table has yet to be reorganized.

Algorithm 3: Node count reporting algorithm

```
1  begin
2      set values for α and β
3      set COUNT_a = current count
4      set COUNT_b = current count
5      begin periodic behavior
6          listen for count reporting from children
7          update COUNT_b
8          if (COUNT_b − COUNT_a)/COUNT_a × 100 ≤ α then
9              do nothing
10             if mode = TRANSITIONAL then
11                 set mode = NORMAL
12                 purge stale routing table
13             end
14         else if α < (COUNT_b − COUNT_a)/COUNT_a × 100 ≤ β then
15             do proportional hash space reassignment among children domains and self
16             create new routing table
17             set mode to TRANSITIONAL
18         else
19             report COUNT_b to parent node
20             set COUNT_a = COUNT_b
21         end
22         if new hash space assignment comes from parent then
23             create new routing table
24             set mode to TRANSITIONAL
25         end
26     end
27 end
```

MSD Server routing modes “TRANSITIONAL” and “NORMAL” is set depending on the state of routing flux. These flags affect the search and session registration algorithms that we will discuss later. Generally each MSD server maintains a single routing table, but when in “TRANSITIONAL” mode, there could be two routing tables temporarily, the old, just made stale, routing table and a newly constructed routing table. When the mode changes back to “NORMAL” from “TRANSITIONAL”, the stale routing table is discarded and the newly constructed table is made the stable routing table.

6 Putting it all together

Now that we have described in details major components that make up mDNS service architecture, let us examine how these components work from an end-user perspective. For an end user to be able to use the services of mDNS, he/she must use a software that implements the communication protocol used in mDNS. For test purposes, we have developed two separate applications, one for end users to be able to search for desired sessions and access any session using mDNS style URLs, and another for session creators to be able to register their content with MSD servers in their domain.

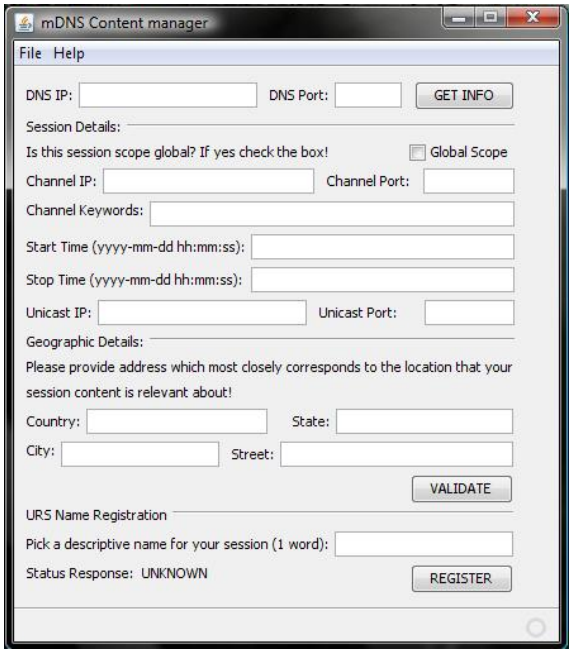

Figure 7: Content Manager

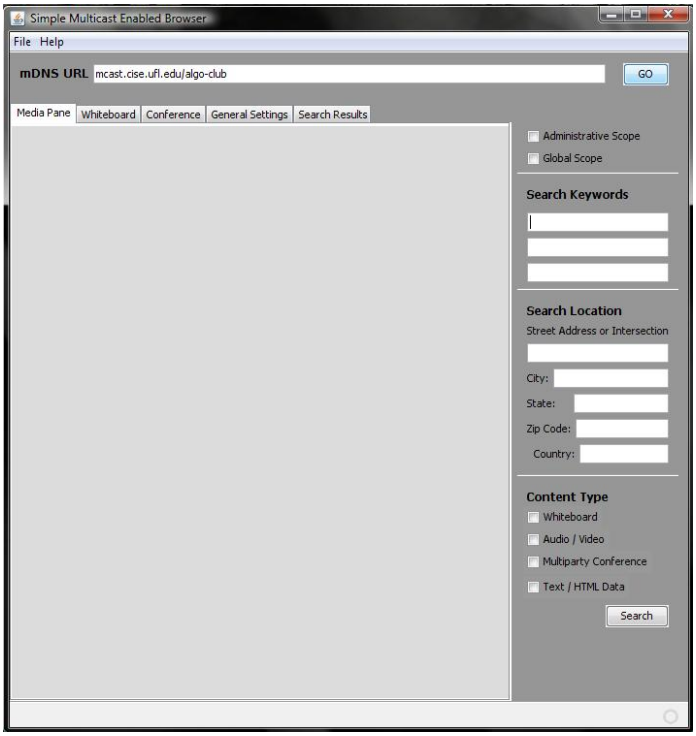

Figure 8: mDNS Content Browser

Figure (7) and (8) shows the mDNS Content manager and Session Browser. Session browser allows users to just type in the mDNS URL and it resolves the URL and retrieve the session details corresponding to the typed URL and starts the session. It also allows users to perform content search based on keywords and

geographical criterion as well.

The search requests to the MSD servers in the local domain is made on MSD-LOCAL-MCAST channel which is an administratively scoped channel. The request has this structure:

```
@SEARCH-REQUEST
{
    Keywords List
    Boolean Operator: AND/OR/NOT
    Location String
    Search Radius (in km)
    Session Scope: Global/Local
    Content Type
}
```

In this request format, location string refers to the street/city location around which the search has to be preformed. Search radius specifies how far from the specified location the session search results must be included. Location String and radius are optional. If location has not been specified then search radius is not used as a filter. Boolean operators by default are assumed to be "OR" if missing. mDNS search semantics specifies operator precedence as NOT, AND and then OR in that order. For example if search is done on "gators AND florida OR algo-club", it is grouped into two searches whose results are combined and returned. These searches would be: "gators AND florida" and "algo-club".

mDNS supports two search modes namely locally-initiated and domain specific. A locally-initiated search is the kind where search requests are sent to domain local MSD servers listening on MSD-LOCAL-MCAST channel. Depending on the search criteria, a locally-initiated search could be forwarded to domains outside the domain where the search has been initiated, the search request based on the routing table entries could be forwarded to parent domain as well to children domains.

In a domain-specific session search, as the name specifies, the search scope is limited to sessions that have been registered and ones originating from within the domain in question. These kind of searches are triggered using mDNS style URLs with trailing search criteria embedded in the URL string itself. An example domain specific search query could look like:

mcast.cise.ufl.edu/search?key=gators+algo-club&location=gainesville+fl+usa&radius=5km

The MSD servers determine based on the incoming request origin whether the request is coming from within the domain or outside the domain. If the request is coming from another domain then locally scoped sessions are not returned. Further, in a domain specific search, only a local database is searched and not the global database. Reason being global database might also contain sessions from other domains depending on how the keyword hash-space was distributed and routing table constructed. Contrary to that, every session originating within a domain always gets stored during registration in the local database without exception. Algorithm (4) shows what steps are taken at any MSD server in a locally-initiated search case. For domain

Algorithm 4: MSD locally-initiated search algorithm

```
1  begin
2      Read boolean mode flag
3      if mode = NORMAL then
4          if search arrives on MSD-LOCAL-MCAST then
               // search came from one of nodes in self domain
5              check routing table
6              if CHANNEL OUT = MSD-LOCAL-MCAST then
7                  search database
8                  return search result
9              else if CHANNEL OUT = CMCAST then
10                 route request on CMCAST
11             else
12                 route request on PMCAST
13             end
14         else if search arrives on PMCAST then
               // search came from the parent node
15             check routing table
16             if CHANNEL OUT = MSD-LOCAL-MCAST then
17                 search database
18                 return search result
19             else if CHANNEL OUT = CMCAST then
                   // narrow search diameter
20                 route request on CMCAST
21             else
22                 drop the query
23             end
24         else
               // search query came on CMCAST channel
25             check routing table
26             if CHANNEL OUT = MSD-LOCAL-MCAST then
27                 search database
28                 return search result
29             else if CHANNEL OUT = CMCAST then
30                 drop the query
31             else
                   // expand search diameter
32                 route request on PMCAST
33             end
34         end
35     else
           // mode = TRANSITIONAL
36         follow transitional mode search pseudocode
           // now routing is done based on both newly computed routing table as well as
              soon to be stale table
           // otherwise the pseudocode is essentially same as for NORMAL mode given above
37     end
38 end
```

specific case, the URL is resolved and the designated MSD server in the target domain is contacted directly by the end-user client, for instance, the session browser (see figure 8) shown above.

Session registration using mDNS content manager is required before any multicast session becomes dis-

coverable by end users. Once a session is registered successfully, it becomes discoverable in a fairly short amount of time. This is another major strength of mDNS hierarchical design. Algorithm (5) shows the steps

Algorithm 5: MSD session registration algorithm

```
1  begin
       // always route based on transitional routing table if it exists
2      Read incoming session registration request
3      if request arrives on PMCAST channel then
4          compare hash value with routing table entry
5          if CHANNEL OUT = MSD-LOCAL-MCAST then
6              register session
7              make entry into global database
8          else
9              if CHANNEL OUT = PMCAST then
10                 drop request
11             else
                   // CHANNEL OUT = CMCAST
12                 route request on CMCAST channel
13             end
14         end
15     else
16         if request arrives on CMCAST channel then
17             compare hash value with routing table entry
18             if CHANNEL OUT = MSD-LOCAL-MCAST then
19                 register session
20                 make entry into global database
21             else
22                 if CHANNEL OUT = PMCAST then
23                     route request on PMCAST channel
24                 else
                       // CHANNEL OUT = CMCAST
25                     drop request
26                 end
27             end
28         else
               // request arrives on MSD-LOCAL-MCAST
29             register session
30             make entry into local database
31             if session scope = global then
                   Data: keyword list
32                 foreach keyword in the list do
33                     compute MD5-128 hash
34                     compare hash value with routing table entry
35                     if CHANNEL OUT = MSD-LOCAL-MCAST then
36                         make entry into global database
37                     else
38                         route registration request on CHANNEL OUT
39                     end
40                 end
41             end
42         end
43     end
44 end
```

involved in session registration. The routing decisions are always done using one routing table, transition-table if MSD server is in "TRANSITION" mode or using stable routing table otherwise. This is unlike search, where search requests might be routed using two routing tables while the MSD server is in "TRANSITION" mode. This is to ensure that global sessions that are in transition and in the process of being routed to different MSD servers because of hash-space redistribution are also discovered by the search algorithm.

7 Analysis

7.1 Geo-Tagged Search Complexity

Let us begin by analyzing the search complexity in mDNS scheme especially when "geo-tagged" database is used. This complexity depends on many factors such as sparse matrix representation format for the planetary grid, each grid location branching factor "k" as one traverses down the tree rooted at that position, search radius "r" desired and database areal resolution parameter "d" that along with "k" determines the tree depth. Where do session originate is generally not known a priori. Therefore we have implemented sparse matrix using array of linked lists. Given the geographic coordinates, using integral part of latitude degree, row index is computed in $O(1)$ time. And since the linked list size for every occupied longitude corresponding

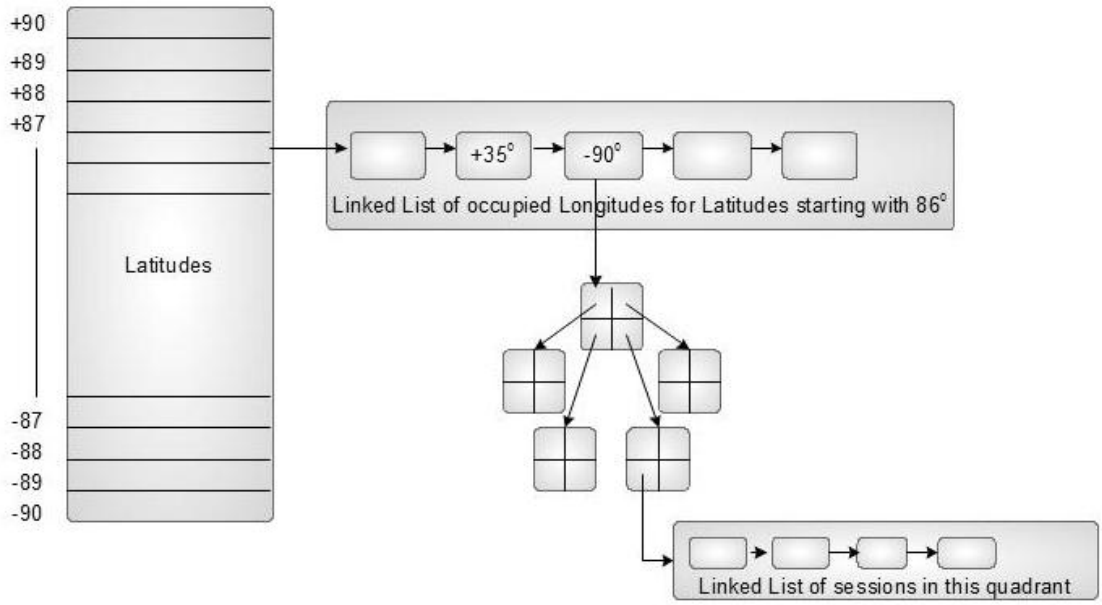

Figure 9: A sample data-structure for sparse-matrix grid in Geo-tagged DB

to a latitude could be at most 360, a linear traversal to find the correct longitude location in the list would also take $O(1)$ time.

Using equations (2) and (3) and specified values for "k" and "d" the maximum depth of the tree "h" rooted at linked-list entry representing a (latitude, longitude) pair can be computed using: $d \geq \frac{110.9}{k^h}$ or

$h = \lceil log_k \frac{110.9}{d} \rceil$. Because session references are maintained only at the leaf node linked-lists (see figure 9), one must traverse the complete tree height regardless of the search radius specified. Now the depth **h'** of tree where the grid vertical resolution is greater than search diameter **2 x r** can be computed using $h' = \lfloor log_k \frac{110.9}{2 \times r} \rfloor$. Now at this depth, the grid's horizontal resolution can be computed using equation 3 by replacing "n" in that equation by **h'**. Because the horizontal "east-west" distances decreases as one moves towards the poles, number of sub-grids **N** that we must traverse laterally in east-west direction in our sparse-matric representation can be computed using:

$$N = \lceil \frac{2 \times r}{\frac{\pi}{180^\circ} \times \cos\phi \times \sqrt{\frac{a^4 \cos(\phi)^2 + b^4 \sin(\phi)^2}{(a\cos\phi)^2 + (b\sin\phi)^2}} \times \frac{1}{k^{h'}}} \rceil \tag{4}$$

And hence the number of possible linked lists at tree leave nodes that one must traverse to find out candidate session records can be easily found out using $N_{leafgrids} = N \times k^{h-h'}$.

The actual geo-search complexity depends on the length of linked-lists, $O(list)$ rooted at the tree-leaves in our sparse-matrix representation, it can be approximated by -

$$C \times (N \times k^{\lceil log_k \frac{110.9}{d} \rceil - \lfloor log_k \frac{110.9}{2 \times r} \rfloor} \times O(list)) \tag{5}$$

where **C** is a constant that can vary between 1 and 4 depending on search-criteria (read:coordinates) proximity to the grid edges or corners of the target quadrant at tree height **h'**. The search complexity can be reduced greatly if we replace leaf linked-lists by hash-tables and using perfect hashing functions.

7.2 Hash-based Keyword Routing: Fairness Analysis

Although its nearly impossible to find out a priori the relative popularity of keywords used to do session searches, let us for the time being assume that every keyword is equally likely to be searched. Further because routing is done using MD5 hash of these keywords, the cryptographic nature of the hashing function makes any hash value in the entire hash space to be routed equally likely. Keeping these assumptions in mind, let us analyze the hash space distribution among participating mDNS MSD servers and the search and routing workload analysis on them.

Suppose the root node has k-child domains such that the sum of MSD-designate count at the root node is N. Let us denote the node count from each child node being reported to the root node by n_i. Thus -

$$\sum_{i=1}^{k} n_i = N \tag{6}$$

Since MD5-128 hash is used, the keyword hash space that must be distributed among participating nodes is

2^{128}. Since we use prefix routing in mDNS, suppose the significant bits that are needed to route appropriately be "m". And therefore -

$$2^m \geq N \qquad or \qquad m \geq log_2 N \tag{7}$$

Without loss of generality, assume that the root node does not participate as MSD server (possibly a TLD server), then the share of hash-space allotted to each child domain at the root level will be $\frac{n_i}{N} \times 2^m \times 2^{(128-m)}$. Therefore for $child_i$ -

$$share_i : \frac{n_i}{N} \times 2^{128} \tag{8}$$

Further, each child node reallocates its assigned hash-space among its children and itself, the space must be divided equally into n_i shares, and thus each participating domain's designated-MSD server share comes out to -

$$share_{MSD} : \frac{n_i}{N} \times 2^{256} \div n_i \qquad or \qquad \frac{1}{N} \times 2^{128} \tag{9}$$

This of course is valid provided the domain hierarchy remains stable over time, as new domains are added and some domain leaves the mDNS hierarchy in real time, there could be times when the above equitable distribution might be violated for short duration of time. This situation should not arise frequently and we conjecture would mostly occur during bootstrapping process. This minor turbulence in stable equitable distribution occurs because the way algorithm 3 has been designed to minimize routing instability.

Let us analyze the workload due to routing of search and registration requests to appropriate MSD servers. Clearly, a node that comes higher up in the tree hierarchy must carry out more routing responsibilities compared to a node that is located close to the leaf domains. At any node in the routing tree, $node_i$, suppose there are **m** children domains, then the routing load at that particular $node_i$ becomes

$$load_i = \frac{1}{N} \times (\sum_{j=1}^{m} count_j + 1) \times 100\% \tag{10}$$

where $count_j$ is the MSD count propagated to $node_i$ from its $child_j$ sub-domain.

Now if every keyword is equally likely to be searched over long period of time, then the workload on a mDNS node $node_i$ over a duration of time "t" becomes -

$$workload_i = t \times rate_{query} \times probability_{range} \tag{11}$$

$$probability_{range} = \frac{share_{MSD}}{2^{128}} \tag{12}$$

Now using equation 9 in equation 12, we get -

$$workload_i = t \times rate_{query} \times \frac{1}{N} \times 2^{128} \tag{13}$$

which shows that the search related workload is also generally equatable provide keywords are equally likely to be searched. Although during short durations, some keywords are more popular than others, how this trends over significant longer period of time remains to be seen.

8 DISCUSSION: mDNS - a paradigm shift in multicast usability

In this section we will discuss new services that mDNS could usher in. Tight coupling of geo-coding in the overall architecture provides the launch-pad for location aware multicast applications and in real time content discovery as and when they are registered with one of the MSD servers allows for some real time ticker or alert services to be offered on a larger scale now.

Consider the case of citizen journalism, because of the services our architecture provides, any person could start reporting using his ever more capable mobile device on the go, register his stream with the nearest MSD servers and the whole world is ready to search and subscribe to his multicast news feed. Take the case of a major motor pileup on a major highway, a person on the scene of news, using his iPhone© could start his own live multicast video feed, register his channel using appropriate keywords say “pileup”, “Georgia” etc. with one of the MSD servers and his session would become searchable in real time. This type of instant multicast feed service paradigm has not been there in the Internet.

Suppose you like listening to a jazz stream multicast from Miami. You can now perform a geo-specific search specifying Miami and the search epicenter and a few miles search radius. The scheme also allows you to bookmark your favorite stream, just like you bookmark a popular webpage. Once you bookmark a session, your content browser will connect to the correct stream in future even if the session connection parameters like channel IP and port number may change over a period of time. Until now the way people would find out about an interesting multicast session was through blog postings, IRC channels and emails. Using such methods the connection information has to be known beforehand and consists of dotted-decimal format and other cryptic details such as what encoding / decoding algorithm to use in order to view a stream. This has made multicast a stomping ground of only researchers and a very tiny fraction of tech savvy people across the globe. The proposed architecture aims to change that and bring multicast to the masses.

Few would argue why not leverage existing search engines like Yahoo© and Google© for the above mentioned scenarios? Table 2 shows categories into which multicast sessions could be classified. Web search

Table 2: Multicast session categories

	Planned	Ad-Hoc
Transient	pre-staged	mDNS
Persistent	Web search engine	any

engines could not be used in cases where the sessions are transient and not pre-planned. If a session is

planned but transient (e.g., a meeting), then details can be advertised or provided out-of-band to invited participants. Only a structured multicast session discovery architecture allows for real-time session discovery of ad-hoc, transient sessions.

Scientists have long argued in favor of multicast. It is a more efficient mode of transmission for stored audio and video broadcasts by large scale content providers like ESPN and SKY. But still deployment of hardware enabling multicast is few and far between. Low user demand of this technology has been one of the reasons. With better usability and seamless integration of search and content delivery applications, the user demand can spike very quickly. This would then make a compelling reason for ISPs around the globe to deploy IP multicast in their network.

References