Efficient Distributed Search for Multicast Session Keywords

Piyush Harsh[1], **and Richard Newman**[2]
[1]Computer and Information Science and Engineering, University of Florida, Gainesville, Florida, USA
[2]Department of CISE, University of Florida, Gainesville, Florida, USA

Abstract— *mDNS is a proposed DNS-aware, hierarchical, and scalable multicast session directory architecture that enables multicast session registration and makes them discoverable in real time. It supports domain-specific as well as global searches for candidate sessions.*

This paper improves mDNS global search algorithm and addresses various security and scalability concerns that remained in mDNS. We propose distributing the overall keyword space among designated MSD servers using hash values. In contrast to other P2P keyword search approaches, we propose IP style prefix routing on the keyword hashes to locate the appropriate MSD server in order to register or retrieve any globally-scoped multicast session. This supports efficient and fast distributed keyword search.

Keywords: keyword routing,multicast,distributed search,hashing

1. Introduction

Multicast currently operates in both ASM (Any Source Multicast) and SSM (Source Specific Multicast) mode. Many researchers as well as several ISPs favor SSM mode over ASM because it removes much of the network complexity that is part of ASM mode. Also IGMP version 3[1] is specially suited for SSM. But with SSM, the source discovery burden rests with the end users. It then becomes critical to come up with a global yet scalable mechanism to enable fast and convenient session (source) discovery by the end users.

In our earlier work we analyzed several existing and proposed systems for maintaining a multicast session database in the Internet. In light of the shortcomings of these approaches, and based on goals we identified for such an architecture, we proposed mDNS[2]. mDNS is a DNS-aware, hierarchical, and globally scalable multicast session directory architecture. It allows multicast session registration and facilitates real-time discovery of these sessions by end users. It addresses many of the concerns that we found in competing proposals in the literature [3] [4] [5] [6] [7] [8] [9]. mDNS is capable of ushering native IP multicast into the next level of deployment.

mDNS allows for two modes of search: domain-specific and global. Domain-specific search is restricted to sessions originating from within the target domain. Global search allows for Internet-wide, globally-scoped multicast sessions to be searched and returned to the requesting end user. To achieve this cuurently, every MSD (Multicast Session Directory) server in every domain is queried for sessions registered with it that match the search criteria. Results are returned directly to the querist.

A drawback of the current approach is that every global search puts a burden on every MSD server, whether or not it has a matching session. This does not scale well, and each query can tie up significant resources. In addition, there are other security concerns, particularly DoS (Denial of service) attacks on any particular end user.

This paper will address these concerns by proposing a smart workload division among MSD servers in various domains using one-way hashes and hash distribution techniques. It further proposes a routing mechanism similar to the IP unicast prefix-based technique that will enable fast routing of global session search queries to the appropriate MSD server(s). This greatly reduces the workload on MSD servers due to global searches. The MSD search algorithm for this solution is provided, and we present a workload analysis (best case and worst case scenarios) among MSD servers for several keyword space division scenarios and end user search patterns.

1.1 A brief introduction to mDNS

mDNS stands for **DNS** aware **m**ulticast directory architecture. It makes use of the existing DNS service in order to enable use of URLs for multicast sessions. It allows sessions to be registered both locally and on a global scale. Unlike other approaches, there is practically no latency in mDNS, i.e., sessions become discoverable as soon as they are registered.

1.1.1 mDNS administrative zone components

Figure 1 shows typical mDNS network components in an administrative domain. These include one or more MSD server, a URS (URL Registration Server), and a DNS server with additional MCAST record entered to facilitate mDNS URL parsing. MSD servers are responsible for storing actual multicast session registration details for multicast sessions originating in their own domain. Additionally the designated MSD server in any domain also stores session details for globally-scoped sessions originating from foreign domains having keywords that fall under its jurisdiction. Details on keyword hashing and distribution are provided in later sections.

The sole purpose of a URS is to maintain uniqueness of identifiers among sessions registered within its own domain. These session identifiers may be duplicated among various

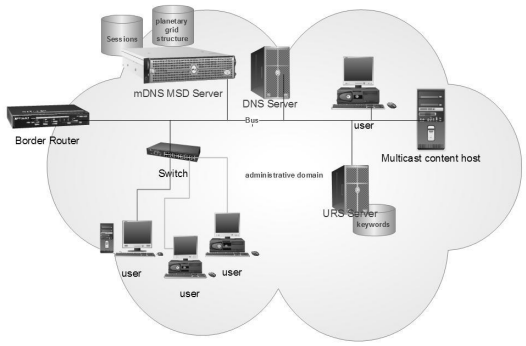

Fig. 1: Typical mDNS domain

domains, but within one domain, the URS prevents duplication of such an identifier among two sessions originating in that domain. This guarantees uniqueness among mDNS URLs and enforces one-to-one mapping of mDNS URLs and multicast sessions.

1.1.2 mDNS hierarchy

For the mDNS scheme to work properly, a DNS server is assumed to be present in every administrative domain in the Internet, or at least in those domains that wish to be part of mDNS hierarchy.

If multiple MSD servers are maintained in a domain, they all are assigned a common anycast address and that allows the mDNS URLs to be translated properly. PMCAST and CMCAST are the multicast channel addresses. They are assumed to be taken from the GLOP[10] range and therefore are ISP assigned. The designated MSD server joins both PMCAST and CMCAST channels, which allows it to form a hierarchy with designated MSD servers in its child subdomains, the designated MSD server in its parent domain, and and those in sibling domains under it's parent domain. Figure 2 shows a typical mDNS hierarchical structure. The

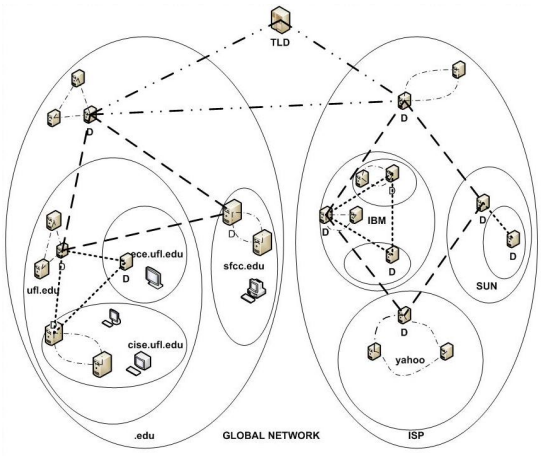

Fig. 2: Typical mDNS hierarchy

complete details of various components of mDNS including session registration, search algorithms, and the bootstrap process have been described in depth in our earlier work [2]. In the rest of this paper we will present enhancements to the global search algorithm to rectify the issues with the current scheme that we identified earlier.

2. mDNS Global Session Search

In the current mDNS design, global searches are propagated on both CMCAST and PMCAST channels. Thus each search query almost surely reaches every designated MSD server in every domain. Once a search query reaches a particular MSD server, it performs a database search and returns the candidate search results directly to the search querist. Clearly the current scheme suffers from security and scalability issues. Depending on the number of MSD servers online, and using IP spoofing, an attacker can swamp any particular host on the Internet using a smurf-like attack[11].

Since each MSD server performs a database search for every global search query that it receives, and given that these searches could be computationally taxing, MSD servers may not be able to sustain a large number of queries. Hence the current scheme raises severe scalability concerns. Another argument against the current scheme is that it is likely that most MSD servers will not have any relevant search results to provide, so why activate these database searches in the first place, if it is unlikely to generate a search query hit?

Nevertheless, the current scheme does avoid session registration data duplication, which is one of the major design goals for mDNS. Can this goal be relaxed a little bit, allowing limited duplication of global session details? In return, can the search algorithm be modified to direct keyword searches to specific MSD servers in the Internet rather than activating every MSD server in the Internet? The next few subsections present a new method to conduct global searches in mDNS that takes this approach.

2.1 Design Goals

mDNS was designed keeping typical Internet design principles in mind:

- usability,
- robustness,
- scalability, and
- maintainability.

In addition to these general design criteria, there were a few service specific goals as well:

- make multicast sessions discoverable in real time,
- prevent multicast sessions' details from being cached all over the Internet,
- make search mechanism sensitive to multicast scope limitations, and
- prevent enforcement of any restrictive session characterization semantics, i.e., give session creators freedom

in choosing keywords to define their session rather than compartmentalizing every session into a small number of predefined categories.

We have tried to uphold these design guidelines set for original mDNS proposal in our modified scheme as well.

2.2 Uniform distribution using Hash schemes

In order to direct global session registration data to a particular MSD server, it is necessary to use a function that provides a uniform and consistent mapping of keywords into fixed length bit string regardless of input keyword length. The mapping must also appear random so as to allow an even distribution of keywords among existing MSD servers. This second property is especially important to maintain fairness in distribution (at least in the keyword distribution). Ensuring a particular server's load fairness is hard to administer *a priori*, as that is dependent on a particular keyword's popularity among end users.

Hashes are functions that have properties well suited for the task, and if they are also one-way, they can provide privacy. Among the many secure hash schemes, we chose SHA2 for mDNS. SHA-256 does not suffer from collisions and APIs are readily available for many platforms.

The hash function is used to hash the keywords that a session creator provides during session registration. The 256-bit hash value is used to locate the appropriate MSD server in the mDNS hierarchy, and the registration details are stored at that particular server. The locally scoped session data are stored locally at MSD servers in the originating domain.

2.3 Keywords Routing using Longest Prefix matching scheme

In order to route registration data to the correct MSD server, we propose a routing approach very similar to current IP unicast routing. To which upstream or downstream multicast channels (PMCAST or CMCAST) to route depends on the longest matching hash prefix in the routing table maintained by every designated MSD server in the mDNS hierarchy. Essentially, the 256-bit hash values take on the role of 32-bit IP addresses in the IP routing analogy.

In order to maintain some fairness, hash space distribution among participating MSD servers is done proportionate to the total number of participating MSD servers in the system and the number of MSD servers in a given subtree in the mDNS hierarchy. We have assumed that MSD servers will be long lived entities and the mDNS hierarchy will be fairly stable over long periods of time. We believe that this assumption is reasonable. Let's take an example scenario to better understand the routing table construction and routing of keywords to appropriate MSD servers. Figure 3 refers to an example tree hierarchy formed by the designated MSD servers in various domains and sub-domains in mDNS. Here node A represents the root DNS server and is the only node in the tree hierarchy that is not a MSD server. Hence it does not store any global session information but does help in the initial hash space division among its children MSD nodes.

2.3.1 Hash Space Division

In the mDNS hierarchy, all children of a parent node join the PMCAST multicast channel, and similarly the parent node joins its CMCAST multicast channel. Needless to say, the CMCAST channel for the parent node and the PMCAST multicast channel for the children nodes are basically the same. This allows the parent node to push down instructions and status to its children nodes and similarly the children nodes are able to communicate amongst themselves and with their parent node.

One of the pieces of information that is furnished to the parent node from its children is the number of MSD nodes in the subtree. This information bubbles up to the root node and thus the root node knows how many MSD servers are maintained under each of its children domains. It uses the information to divide the hash space proportionately among its children domains. Figure 3 shows how the MSD count

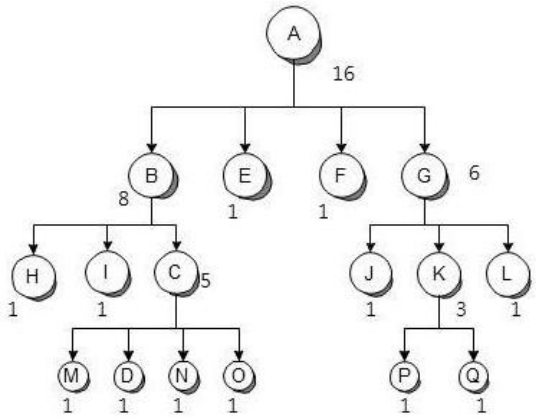

Fig. 3: MSD-D Nodes Count

propagates up to the root node. The number next to each node represents the number of designated MSD servers in the subtree rooted at that node (including that node itself). The only exception is the root node, where the count does not include it, because it does not maintain any session database and hence the hash space division should not include it.

In the example of Figure 3, the 256-bit SHA-256 generated hash space is divided into 16 parts and the space divided into each child domain proportionately.

2.3.2 Routing Table Construction

During the process of hash space distribution, the parent node also constructs the routing table. Let us demonstrate how this is done taking the path of nodes from A to D via nodes B and C. We will use CIDR notation to specify groups of hash addresses in the routing table. The table shown for Node A (i.e., Table 1) is just for illustration. Since any hash

Table 1: Routing table at node A

Hash Address	Node ID	Channel Out
$0\overbrace{xxx...xxx}^{255}/1$	Node B	CMCAST
$10\overbrace{xxx...xx}^{254}/2$	Node G	CMCAST
$110\overbrace{xxx...x}^{253}/3$	Node G	CMCAST
$1110\overbrace{xx...x}^{252}/4$	Node E	CMCAST
$1111\overbrace{xx...x}^{252}/4$	Node F	CMCAST

query will go down via CMCAST channel at the root node, the routing table at root node (i.e., A) can contain just a single entry with * in the **Hash Address** column signifying all remaining address values (in this case every value,) and **Channel Out** set to **CMCAST**.

Node B similarly divides the hash space allotted to it by the root node into 8 parts, assigns one part to itself and allots the remaining hash space to downstream MSD servers. For clarity, the routing table at Node B is also provided. Thus

Table 2: Routing table at node B

Hash Address	Node ID	Channel Out
$0000\overbrace{xx...xx}^{252}/4$	self	MSD-LOCAL-MCAST
$0001\overbrace{xx...xx}^{252}/4$	Node H	CMCAST
$0010\overbrace{xx...xx}^{252}/4$	Node I	CMCAST
$0011\overbrace{xx...xx}^{252}/4$	Node C	CMCAST
$01\overbrace{xxx...xxx}^{254}/2$	Node C	CMCAST
*	PARENT	PMCAST

every designated MSD server knows about the hash space for which it is responsible. It maintains a global multicast session registration database for all sessions with a keyword whose hash value maps to its own subset. It also constructs the routing table in order to direct the query correctly along the path to the appropriate MSD server. mDNS provides for the possibility of multiple MSD servers in any domain. If multiple servers exist, they all subscribe to the administratively scoped multicast channel MSD-LOCAL-MCAST. This provision enables fault tolerance and load balancing in the mDNS architecture at the local domain level.

2.4 Session Registration and Retrieval

A new session registration request is always initiated on the MSD-LOCAL-MCAST channel. The session, regardless of its scope, is always first registered with the domain local MSD servers. If the session is global in scope, the provided keywords are hashed individually using SHA-256 and each generated hash value is used to route the registration details to the appropriate MSD server in the mDNS hierarchy. Each MSD server along the route uses the hashed value and its own routing table to forward the registration request on either PMCAST or CMCAST channel. The routing table can be maintained in main memory in most cases and the hash value check against it can be extremely fast. Once the registration request reaches the correct MSD server (the one responsible for the hash space to which the candidate hash value belongs) the registration details are stored into the session database and the registration process finishes.

The session search proceeds along a similar line. An end user initiates multicast session search by providing session keywords. If the search is for administratively scoped sessions only or is domain specific, then the search algorithm is essentially the same as in the original mDNS proposal (i.e., it is handled by the designated MSD server for that domain). But if the search is Internet-wide, then the keywords are hashed individually, and each hash value is used to route that particular search keyword to the target MSD server that stores session details for globally-scoped multicast channels for that particular keyword. The relevant sessions are sent across from the remote MSD server directly to the end user/querist.

3. Routing table stability and other design considerations

The routing table at each participating MSD node depends on the hash space distribution among children sub domains. The table entries themselves depend on how the space has been distributed from root node downwards. This distribution in turn depends on the ratio of MSD nodes in a particular branch administrative domain compared to other branches of the domain hierarchy tree.

Most of the system state in mDNS is maintained via soft state refreshes. This provides an added benefit of dynamic adaptability of the system to changes in network topology. The system becomes more resilient to partial component failures and network reconfiguration. Let us analyze what effect addition and deletion of MSD nodes might have on a routing table.

mDNS architecture allows for multiple MSD servers at same level in a domain. If multiple MSD servers exist, then using a leader election protocol, one of them is chosen as the designated MSD. It is the designated MSD count that is propagated towards the root node. Therefore addiing MSD servers to an existing administrative domain will not result in any routing table change as the designated MSD count remains same as before.

The only scenarios where the designated MSD count could change are when network administrative domains or sub-domains are added or deleted. We hypothesize that these actions would be infrequent. Administratively independent network zones (domains/sub-domains) are not created or

Algorithm 1: MSD session registration algorithm

```
1  begin
       // always route based on transitional-next 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 local 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 local 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 SHA-256 hash
34                     compare hash value with routing table entry
35                     if CHANNEL OUT = MSD-LOCAL-MCAST then
36                         do nothing
37                     else
38                         route registration request on CHANNEL OUT
39                     end
40                 end
41             end
42         end
43     end
44 end
```

destroyed often. Such decisions have significant monetary implication on any organization or ISP and hence are taken usually as a long-term decision. Hence we conjecture that the network domain hierarchy is a quasi-stable structure. As a result, the mDNS MSD routing table structure generally should be stable over long time periods.

Still any change in the hierarchy could result in hash space redistribution from root node down to all the leaves and interior nodes. Hence we have decided to incorporate threshold-based count propagation, described later, as a stabilizing measure.

3.1 Routing during transition phase

Every session registration gets a lease duration until which the data is maintained at the MSD servers. In order for the same registration data to remain valid, the registrant must renew the lease before the current one expires. There is a tradeoff between communication overhead and lease duration. Without loss of generality let us suppose that a typical lease period is **k** days. Any registration entry at a given MSD may expire in one of two ways - if the session itself is no longer valid after a given time, or if its lease expires and the session has failed to renew its lease within the expired lease period.

If due to some significant network hierarchy change, the hash space is redistributed among existing designated MSD servers, the old routing table is maintained until the next routing table update cycle, which is also a periodic update with periodicity **k** days. The system makes sure that the time gap between existing hash space distribution scheme and next update is at least 1 period. The assignment is done (if needed) during a particular update cycle but the assignments achieve total validity from the next cycle.

In between these two cycle, the mDNS system goes into transition mode behavior. During the transition period, each MSD server depending on its own network load, transfers the registration records that would no longer belong to its future assigned hash values set to the appropriate MSD servers. As and when the registration records are transferred reliably, they are deleted as well from the old MSD servers. Once the transfer of records finishes, the MSD node that transferred all the records marks its routing entry for that space as stale. If every routing entry at a node becomes stale, it sends an explicit prune request to its parent. Once the parent node receives the explicit prune request from one of its children, it marks the corresponding routing entry in the current (soon to be stale) table as stale as well and drops future search requests belonging to that child node's space. This explicit pruning behavior is recursive in nature.

The registration requests always are routed based on the newer hash space distribution if one exists (or transitional-next routing table). The search queries during the transition period are routed to both the assigned MSD server according to newer routing table as well as the existing (soon to be stale) routing table. As soon as the transition cycle has expired, the old invalid routing entries are purged and replaced with the routing table that reflects the newer scheme and the system proceeds from the transition mode to stable mode behavior.

Although this scheme allows for sessions to remain discoverable during transition phases, we would like these transitions to be as few as possible. In order to achieve this goal, we propose to use a threshold-based algorithm to determine how each MSD server reports the MSD count up to its parent node. Algorithm 2 describes the normal mode search process. Transitional mode search is very similar to the normal mode search algorithm, except both **current** as well as **transitional-next** routing tables are checked. If any **current** routing table entry is marked **STALE**, the corresponding action (database search or routing) is not taken in transitional mode.

Algorithm 2: MSD normal mode 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 one of the child
                  sub-domains
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
37     end
38 end
```

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
```

3.2 Threshold scheme and its implications

In order to reduce mDNS architecture transitional phase frequency, any designated MSD node count reporting behavior is governed by a simple threshold based algorithm (see algorithm 3). Two threshold values, α and β are used with $0 < \alpha < \beta < 50\%$. Once the initial mDNS hierarchy buildup stage is over, we conjecture that algorithm 3 will result in more stable and fewer mDNS transitions. The values of α and β are tunable and more experimentation is needed to decide on optimal settings.

4. Performance Evaluation

Suppose at the root level, there are **k** child domains and the total count of designated MSD servers is **N**.

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

The size of the hash space to divide among these servers is 2^{256}. Let the significant bits used for routing between MSD servers be **m**. Then

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

Each child domain at the root level is assigned $\frac{n_i}{N} \times 2^m \times 2^{(256-m)}$ part of the total hash space. Hence the share allotted to $domain_i$ becomes

$$share_i : \frac{n_i}{N} \times 2^{256} \tag{3}$$

Since each domain now distributes its allotted space among the total designated MSD count at that node, each MSD server's share becomes: $share_i \div n_i$. After simplification, each MSD server's hash-space share comes out to be

$$share_{MSD} : \frac{n_i}{N} \times 2^{256} \div n_i \qquad or \qquad \frac{1}{N} \times 2^{256} \tag{4}$$

If the mDNS hierarchy remains stable, then the hash space is shared uniformly among every designated MSD server in the system. As more servers are added, this hash space division could become non-uniform until the next top-down reassignment is done starting at the root node.

A node that appears higher in the mDNS hierarchy does more query routing compared to a node at lower level. Routing load depends on the MSD count at that particular node. At any $node_a$, suppose there are **j** children domains, then the routing load becomes

$$load_a = \frac{1}{N} \times (\sum_{i=1}^{j} count_i + 1) \times 100\% \tag{5}$$

where $count_i$ is the MSD count propagated to it from its $child_i$.

If every keyword is equally likely to be searched, then the workload for any MSD node "a" in the hierarchy over a period of time **t** becomes

$$Workload = t \times rate_{query} \times probability_{range}$$
$$probability_{range} = \frac{share_{MSD}}{2^{256}}$$

Using equation (4), we get

$$Workload = t \times rate_{query} \times \frac{1}{N} \times 2^{256} \qquad (6)$$

Hence under mDNS, the query workload is also evenly distributed provided each keyword is equally likely to be searched.

5. Discussion

Hashing has been used extensively in many peer-to-peer systems such as Chord [12], Tapestry [13] to name two. But those schemes are used to assign space to servers, which could result in gross imbalances. Further, nodes using such schemes need to know all servers eventually for effective routing. In our scheme we use hashing for routing purposes directly. The peer-to-peer schemes work pretty well under dynamic environments. mDNS hierarchy is not very dynamic and peer-to-peer routing mechanisms and space distribution schemes do not fit mDNS requirements well.

Caching can be used to reduce network load on the downstream MSD nodes even further. Instead of target MSD node returning the search query directly to the requesting client application, it may be done recursively, in a fashion similar to DNS domain name translation done recursively instead of iteratively. Those MSD servers along the routing path now can cache popular search keywords, and store the IP address of the target MSD server locally. Cache maintenance could use one of the numerous cache management schemes such as LRU-K, segmented FIFO to name a few. Under such a scheme, the MSD servers along the routing path first look into their local caches; if a keyword hit occurs, they return the corresponding MSD server's IP address directly to the client software. This also has the benefit of reducing the delay incurred in routing all the way to the target MSD server.

Why not store the sessions details in local caches? First of all, the cache size will increase significantly. Secondly, session details usually expire (unless renewed) after the lease period is over; this would result in stale caches and frequent cache maintenance. How much improvement cache implementation in mDNS will bring is debatable. On one hand it increases workload on nodes that are closer to the root which one might argue to be unfair. On the other hand, it has the potential to reduce latency. There is an obvious tradeoff in implementation of caches. Whether decrease in latency is significant enough to warrant serious consideration remains to be seen. An advantage is that this decision can be made locally.

In an earlier section, we asked the question, can we bound the data duplication in order to maintain one of the key goals of mDNS architecture? This upper bound, under the modified global search scheme, is governed by the maximum number of keywords used to describe a session during registration with MSD servers. Since each keyword is hashed, and the different hash values for different keywords would most likely route the registration towards different MSD servers, resulting in session details duplication in those servers. We place no restrictions on the number of keywords provided during registration phase. But we conjecture that the situation under the proposed modification will still be much better compared to some other schemes including sdr.

6. Conclusion

mDNS solves the problem of source discovery and improves usability by permitting the use of user-friendly URLs.

This paper has revised the mDNS search algorithm in order to achieve better scalability and reduce DDoS potential. The routing approach to keyword distribution uses a hashing technique similar to methods used in P2P networks, but with significant differences. A fair distribution scheme was given, and the paper provided some measure of workload assessment on individual servers in the mDNS architecture. The paper has also discussed routing stability issues and provided algorithms for the modified scheme.

References

[1] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, *Internet Group Management Protocol, Version 3*, 2002, internet Engineering Task Force, RFC 3376.

[2] P. Harsh and R. Newman, "mdns - a proposal for hierarchical multicast session directory architecture," in *Proceedings of the 2008 International Conference on Internet Computing*, 2008, pp. 31–37.

[3] M. Handley, "The sdr session directory: An mbone conference scheduling and booking system," April 1996. [Online]. Available: http://www-mice.cs.ucl.ac.uk/multimedia/software/sdr/

[4] A. Swan, S. McCanne, and L. A. Rowe, "Layered transmission and caching for the multicast session directory service," in *ACM Multimedia*, 1998, pp. 119–128. [Online]. Available: citeseer.ist.psu.edu/swan98layered.html

[5] A. Santos, J. Macedo, and V. Freitas, "Towards multicast session directory services." [Online]. Available: citeseer.ist.psu.edu/264612.html

[6] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz, "The harvest information discovery and access system," *Computer Networks and ISDN Systems*, vol. 28, no. 1–2, pp. 119–125, December 1995.

[7] N. Sturtevant, N. Tang, and L. Zhang, "The information discovery graph: towards a scalable multimedia resource directory," *Internet Applications, 1999. IEEE Workshop on*, pp. 72–79, Aug 1999.

[8] P. Liefooghe and M. Goosens, "The next generation ip multicast session directory," *SCI, Orlando FL*, July 2003.

[9] P. Namburi and K. Sarac, "Multicast session announcements on top of ssm," *Communications, 2004 IEEE International Conference on*, vol. 3, pp. 1446–1450 Vol.3, 20-24 June 2004.

[10] D. Meyer and P. Lothberg, *GLOP Addressing in 233/8*, 2001, Internet Engineering Task Force, RFC 3180.

[11] Craig A. Huegen, *The Latest in Denial of Service Attacks: "Smurfing" Description and Information to minimize effects*, May 2008, [Online]. Available: pentics.net/denial-of-service/white-papers/smurf.cgi.

[12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 149–160, 2001.

[13] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: a resilient global-scale overlay for service deployment," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 1, pp. 41–53, Jan. 2004.