Mode Independent Session Directory Service Architecture

a unified approach for ASM and SSM multicast networks

Piyush Harsh
CISE, University of Florida
Gainesville, FL USA
pharsh@cise.ufl.edu

Richard Newman
CISE, University of Florida
Gainesville, Florida 32611-6120
nemo@cise.ufl.edu

ABSTRACT

In this paper we describe architectural changes incorporated into DNS aware Multicast Session Directory (mDNS) that enable it to co-exist in both Any Source Multicast (ASM) and Source Specific Multicast (SSM) environments. mDNS is a distributed, global, scalable and hierarchical approach that allows multicast sessions to be searched based on multiple parameters including keywords, session-type, geo-locality, etc. mDNS design being tightly coupled with existing Domain Name Service (DNS) enables sessions to be assigned a Uniform Resource Locator (URL) that can be book-marked for future access.

We also describe the caching strategy added to mDNS and present arguments on its possible benefits. Afterwards, we will discuss the slight search strategy alteration required by caching and its security implications. This paper also describes our simulation design. We describe how we automated our experiments. The integration of fully implemented mDNS software and our "simulated" network hierarchy will be explained. We provide simulation results and explain their significance with respect to network topography.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Distributed Networks; C.2.4 [**Distributed Systems**]: Distributed applications; C.4 [**Performance of Systems**]: Modeling techniques

General Terms

Session Directory

Keywords

mDNS, multicast, search, DNS, caching

1. INTRODUCTION

SAC'10 March 22-26, 2010, Sierre, Switzerland.

mDNS [7] [8] [9] is a globally scalable, distributed and hierarchical multicast session directory service. It is tightly coupled with the existing DNS architecture and leverages its features that allow multicast sessions to be assigned URLs. These URLs can be book-marked just like any other URL strings. Such sessions can be accessed in the future using their assigned URLs. mDNS also tags sessions registered with it using geographical tags depending on the place of origin of such sessions or the nature of their content [9]. Each session registration entry also contains several keywords provided by the content host that allows the stream to be discovered by the receiver nodes. These features enable end users to search and discover any multicast session registered with mDNS quickly and efficiently. In [8] authors developed a hash-based routing scheme along with the associated data-structures and workload distribution algorithm that tries to balance workload assigned to participating directory servers. Hierarchy construction using soft-state refreshes and self management in the face of intermittent node failures relieves unnecessary burden from system administrators.

mDNS up until now supported only ASM [4] [5] mode of IP multicast. ASM mode support for two-way communication between sender and receiver nodes, along the shared multicast channel, allows for minimal routing table entries, which speeds lookup functions. With increasing support for SSM [2] mode of IP multicast among the network researchers and Internet Service Providers (ISPs), and standardization of Internet Group Management Protocol version 3 (IGMPv3) [3], future multicast capable networks will be configured as SSM networks. Yet, we foresee that ASM will continue to exist for a significant time. The already proposed architecture along with other supporting components in mDNS lend themselves organically towards SSM albeit with minor structural/protocol modifications. These changes are necessary as SSM mode supports only one-way data dissemination, i.e., from source node towards the receivers.

The rest of the paper is organized as follows: we will provide a very brief description of mDNS architecture describing only the salient features. Then we will describe few minor, yet necessary, architectural and protocol changes that allow it to support both ASM as well as SSM modes of operation. We will describe the caching strategy and its benefits for search speedup. We will describe in some detail our simulation setup and integration of the actual implementation with the simulated multicast network hierarchy. We will end this paper by presenting some of the simulation results and

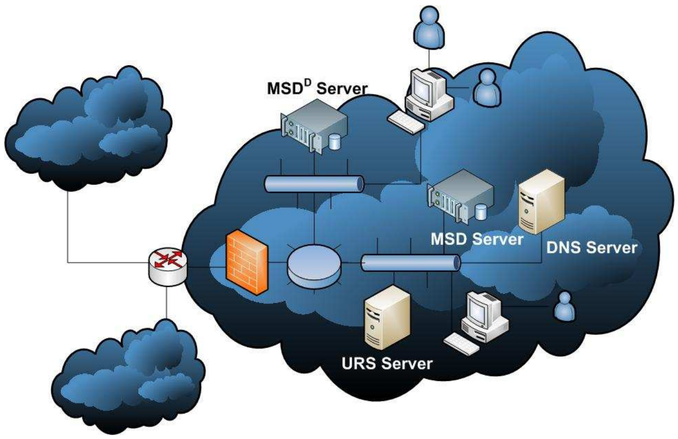

Figure 1: A typical mDNS domain setup.

discuss the impact of network topology on mDNS operating parameters.

2. MDNS: BRIEF INTRODUCTION

A typical mDNS domain must have a DNS server configured and running properly. It must house a mDNS URL Registration Server (URS) and at least one Multicast Session Directory (MSD) server. If more than one MSD server is installed, they are capable of choosing a designated server amongst themselves. Currently the developers have incorporated a simple bully-leader election algorithm [6] in the MSD servers. The DNS server must be configured to point to the URS server. They have chosen the name "MCAST" for the URS server. The URS server aids in multicast session URL translation. We have configured our DNS server to fully support mDNS requirements. A subset bind configuration of our DNS server is given here:

```
; cons.cise.ufl.edu
$TTL    604800
@       IN      SOA     ns1.cons.cise.ufl.edu. (
                     2006020249 ; Serial
                         604800 ; Refresh
                          86400 ; Retry
                        2419200 ; Expire
                         604800); Negative Cache TTL
;
@       IN      NS      ns1
        IN      MX      10 mail
        IN      A       128.227.170.50 ;
mcast   IN      A       128.227.170.43 ;
```

The above DNS configuration allows the URS server to be accessed using the "mcast.domain-URL" string. Our URS server can be accessed thus using "mcast.cons.cise.ufl.edu". Each multicast session must register a session identifier string with its domain local URS server. The URS server guarentees the uniqueness of the registered session identifier within its own domain. Using this (domain-relative) unique session identifier, we can construct the FQDN URL for any multicast session. For example, if we create a multicast session in our subdomain and register an identifier called ***algos-channel*** with our URS server, then one can access our multicast channel using the URL ***mcast.cons.cise.ufl.edu/algos-channel***. Thus using DNS and URS servers, one can resolve the mDNS multicast session URLs to the respective session details and hence will be able to access the session contents.

Network administrators of every domain are required to configure the *PMCAST* and *CMCAST* parameters of the URS server. *PMCAST* refers to the globally scoped GLOP [11] multicast channel that enables the designated MSD server in a particular domain to communicate with its counterpart in its parent domain. Similarly, *CMCAST* is the globally scoped GLOP multicast channel that enables the designated MSD server to communicate with MSD servers in its child domains. In the current version these are the only three parameters that the system administrator must manually configure, namely - the *MCAST* DNS entry for the URS server, and the *PMCAST* and *CMCAST* parameters in the URS server. The rest of the system configuration, including mDNS hierarchy construction and maintenance, is performed automatically without system admin intervention. Our approach is thus very sys-admin friendly.

PMCAST and *CMCAST* channels are globally scoped because the communication on these channels must cross domain boundaries. In contrast the *MSD-LOCAL-MCAST* channel is a well-defined administratively scoped multicast channel over which end-users can communicate with the domain-local MSD servers. Even multiple MSD servers (if they exist) in the domain communicate with each other over this channel. Inter-MSD (intra-domain) server communication is needed for leader elections, backups and synchronization tasks to be performed among installed and online MSD servers.

Using the *PMCAST* and *CMCAST* channels, the designated MSD server in each mDNS domain forms the hierarchical overlay network with other mDNS enabled domains. MSD servers retrieve relevant configuration details by communicating with their domain-local URS server. They exchange domain count details along the child-to-parent path which helps in the calculation of workload distribution (hash-space allotments) among the participating domains. Each MSD server constructs a keyword routing table that facilitates end-user searches. Please refer to [7] [8] [9] for more details on the algorithms, data-structures and protocols involved. Figure (2) shows an example mDNS hierarchy

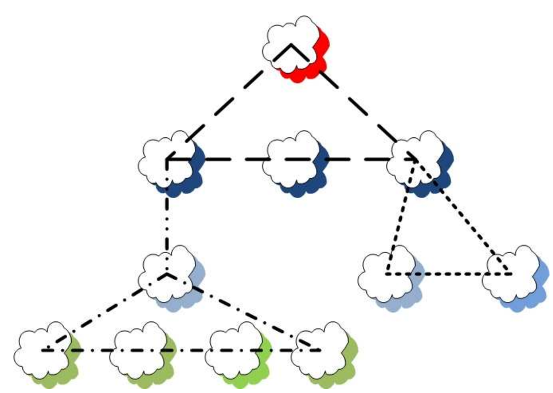

Figure 2: A typical mDNS domain hierarchy.

where different communication channels among participating domains are shown. These overlay hierarchies are self-maintained and constructed using the *PMCAST* and *CMCAST* channel using a soft state approach.

To be more accurate, the above figure shows the scenario if every domain is configured to handle ASM multicast communication. ASM mode allows for two-way communication and the hierarchy is generated by setting the *PMCAST* channel of a child domain to be same as the *CMCAST* channel of its parent domain. Next we will describe a few modifications, that would enable this general approach to work in SSM mode as well, thereby improving its versatility.

3. SSM MODE OPERATIONS

Multicast channels in SSM mode have an additional unicast IP identifying the sender node. A typical SSM mode channel is therefore a three parameter tuple:

```
<UNICAST-IP; MULTICAST-CHANNEL-IP; MULTICAST-CHANNEL-PORT>
```

Compare that to the two parameter tuple that identifies a ASM mode multicast channel:

```
<MULTICAST-CHANNEL-IP; MULTICAST-CHANNEL-PORT>
```

Hence in order for mDNS to operate in an SSM environment, a child node must be able to subscribe to the channel on which the parent node sends critical network information (down-link). Further, the child node can not use the same channel to communicate (up-link) with the parent. SSM channels are one-way communication medium. Therefore the child node must revert to point-to-point unicast transmission to its parent node (up-link). Thus mDNS must be modified to support down-link communication via SSM multicast and up-link communication via point-to-point unicast.

Since no one knows *a priori* the IP address of the designated MSD server in the parent domain, it would be unwise to make it a configurable parameter to be set by system administrators. The issue is compounded by the fact that in the face of multiple MSD servers, the designated MSD server is chosen as a result of leader election. If the leader dies, the remaining MSD servers are capable of detecting the failure. A new leader is chosen after another election. In the previous version, in order to support domain-specific session search [7], mechanisms were already in place for tracking the designated MSD server in the URS server. Tracked information is the unicast IP and the port value of the unicast server running as a thread in the designated MSD server. This feature can be utilized as a solution here. Changes needed to support SSM by mDNS:

- include parent domain URL as a configurable parameter in URS server
- include operating mode information in URS server

Inclusion of the parent URL in URS server enables it to query the upstream (parent) URS server and query for the IP address and the port information of the current unicast server of the designated MSD server in the parent domain. This information, together with *PMCAST* and *CMCAST* channel details, is sufficient for MSD servers in the child domain to subscribe to the SSM multicast channel of the parent node. Furthermore, the unicast server details received from the URS server in the parent domain enable the child domains to communicate upstream via unicast. Provisions are made to refresh the data in the child domains in case the designated MSD server fails in the parent domain and a backup MSD server takes over. This is achieved easily using periodic refreshes among URS servers and between the URS server and the designated MSD server in a given domain.

The *operating mode* parameter in URS servers can be set in one of the following three modes:

```
OPERATING MODE:
  0: SSM Mode
  1: ASM Mode
  2: HYBRID
```

This parameter provides a hint to the MSD server on how to operate. If an MSD server is operating in ASM mode, then both upstream and downstream communication is done via multicast. If however an MSD server is operating in SSM mode, then upstream communication is done via unicast and downstream communication is done via SSM multicast.

There is an provision for a third mode, namely, hybrid mode. This mode could be set if a particular domain is capable of handling both SSM and ASM modes of IP multicast. Such a domain can be used as a bridge between ASM and SSM networks. In hybrid mode, the MSD server sends messages in duplicate, once as if it was acting in ASM mode and again as if it was in SSM mode. Also it subscribes to corresponding ASM and SSM channels for receiving communications from upstream/downstream domains (if any). Duplicate messages are discarded. You should convince yourself that the hybrid mode domain will be able to communicate correctly irrespective of whether the child domains are configured in ASM or SSM mode. The same is valid if the parent domain of the hybrid domain is configured in SSM or ASM modes.

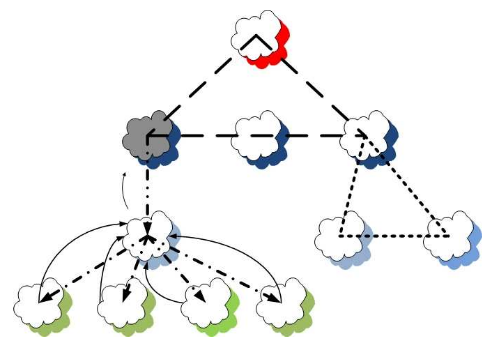

Figure 3: mDNS operating in hybrid environment.

Figures (3) and (4) show some hybrid scenarios. The shaded domains are configured to operate in the hybrid mode. Multicast channels operating in ASM modes are shown without arrows in the communication links. Channels in SSM modes are shown with a downward arrows. The unicast up-links are shown using arcs with arrows showing the direction of communication.

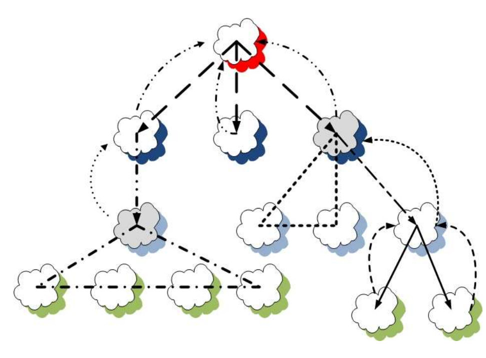

Figure 4: mDNS hybrid environment - example 2.

Figure (4) shows a scenario where a domain configured to operate in hybrid mode has a few children that are configured to run in ASM and another child in an SSM network.

A hybrid domain can handle ASM as well as SSM capable child domains.

4. IMPROVING LOOKUPS & CACHING

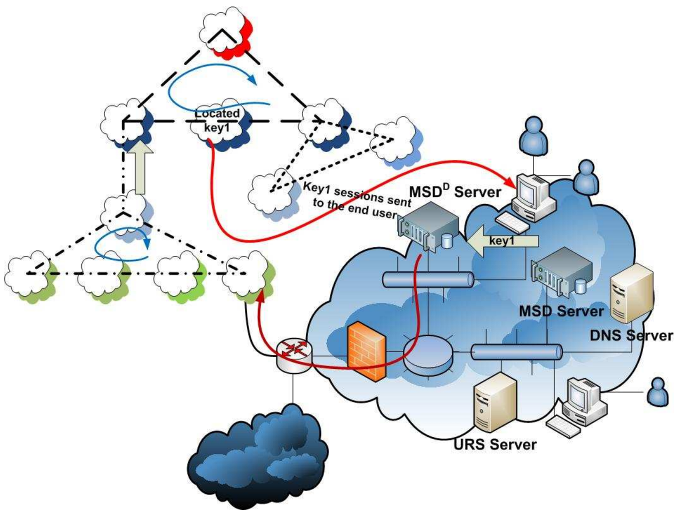

Figure 5: current mDNS keyword search.

Figure (5) shows the previous approach to the end-user session search requests. A user sends a search request to its domain-local MSD server. Using the hash-based keyword routing scheme [8] the request is routed to the appropriate MSD server (one in whose hash-space the search key falls). This MSD server then sends back the queried session data directly to the end user. Connection information to the end-user terminal is part of the search request, which allows the target MSD server to establish a connection in order to send back the desired results.

This approach suffers from several security risks. One such risk is that a malicious node can establish a connection to the end-host using the connection information contained in the search query. Additionally the end-user system firewall as well as the network firewall may block the incoming connection to the end user as a security precaution. The standard practice suggests that the end user must initiate the connection to the target MSD server to query for the keyword and wait for the target MSD server to respond with the search results. The latter approach would resolve the incoming connection problem with the network firewalls as the connection is initiated by a node in their domain, the nature of connection now becomes outgoing compared to an incoming connection in the first approach.

Figure (6) describes the modified approach that resolves the issues with the earlier approach to search. The target MSD server sends the connection details for its unicast server to the MSD server where the search request originated. Firewalls can be easily configured to accept incoming connection to a few well known IPs and ports corresponding to MSD servers running in their domain. The originating MSD server then responds back to the client machine providing the connection details for the target MSD server that manages the session details associated with the queried keyword. Next, in pass 2, the client node can directly connect to the target MSD server and retrieve the search query results. The obvious drawback is the additional delay incurred due to the extra (second) pass. The security and firewall maintenance benefits makes up for the additional delay.

4.1 Offsetting delay: use of caching

The extra delay incurred due to the additional second pass in the modified approach can be greatly offset if the connection details for the target MSD servers can be retrieved quickly. In mDNS, once the hash-space allocation and hash-routing construction phase stabilizes, the MSD connection details would become stable as well. Unless many domains join and leave the mDNS hierarchy in an arbitrary fashion, the hierarchy as well as hash space allotment remain stable. One way a target MSD connection information may change even if the hierarchy itself is stable is if the designated MSD server fails. In this case if a backup MSD server is running, it will soon become the designated MSD server (after a fresh leader election) and thus the IP address will change. But we expect such cases to be very rare. These arguments make *MSD connection details* an excellent candidate for caching.

With caches in place, when a end-user requests a keyword search for multicast sessions, the domain-local MSD server checks the cache. If there is a cache hit, then it immediately sends the cached connection details for the target MSD server to the requesting end-user. The end-user tries to connect to the remote target MSD server; if it succeeds, the delay incurred is reduced significantly. If it fails, most likely due to changed connection information in the target domain (due to primary MSD server failure), or if the target domain is not responsible for the keyword due to more recent hash space reassignments (likely caused due to network topography changes), the end-user prompts the domain-local MSD server to invalidate the stale entry. The original two-pass protocol is then used, which refreshes the stale entry and the process continues from there.

4.1.1 caching strategy

In order to maximize upon the benefits of Least Recently used (LRU) [10] [12] and Least Frequently used (LFU) [10] caching strategies, we used a hybrid caching strategy. Table (1) shows typical cache entries in our hybrid caching strategy.

Table 1: typical cache structure

keyword	access time	freq	score	ip:port
gators	1249607331102	534	247.00424	abc:q
football	1249607331102	61	57.804245	def:w
nfl	1249607377712	712	318.67035	ghi:e
beach	1249607339173	11	37.884953	jkl:r

Assume the current system time is 1249607590678 and the timeout value is 3600000 milliseconds (10 hours). The above table entries correspond to values calculated using $\alpha = 0.4$. The score component for any cache entry is computed using

$$\alpha \times (freq) + (1 - \alpha) \times \left[\frac{timeout - (t_{curr} - t_{last-access})}{60000} \right]$$

$$0 \leq \alpha \leq 1$$

If α equals 1 then the cache operates in accordance to LFU scheme, and if α is 0, the cache operates like a LRU cache. The cache entry with the lowest score value is selected for replacement (if needed). In the above table, *timeout* is a configurable parameter that determines the soft timeout for

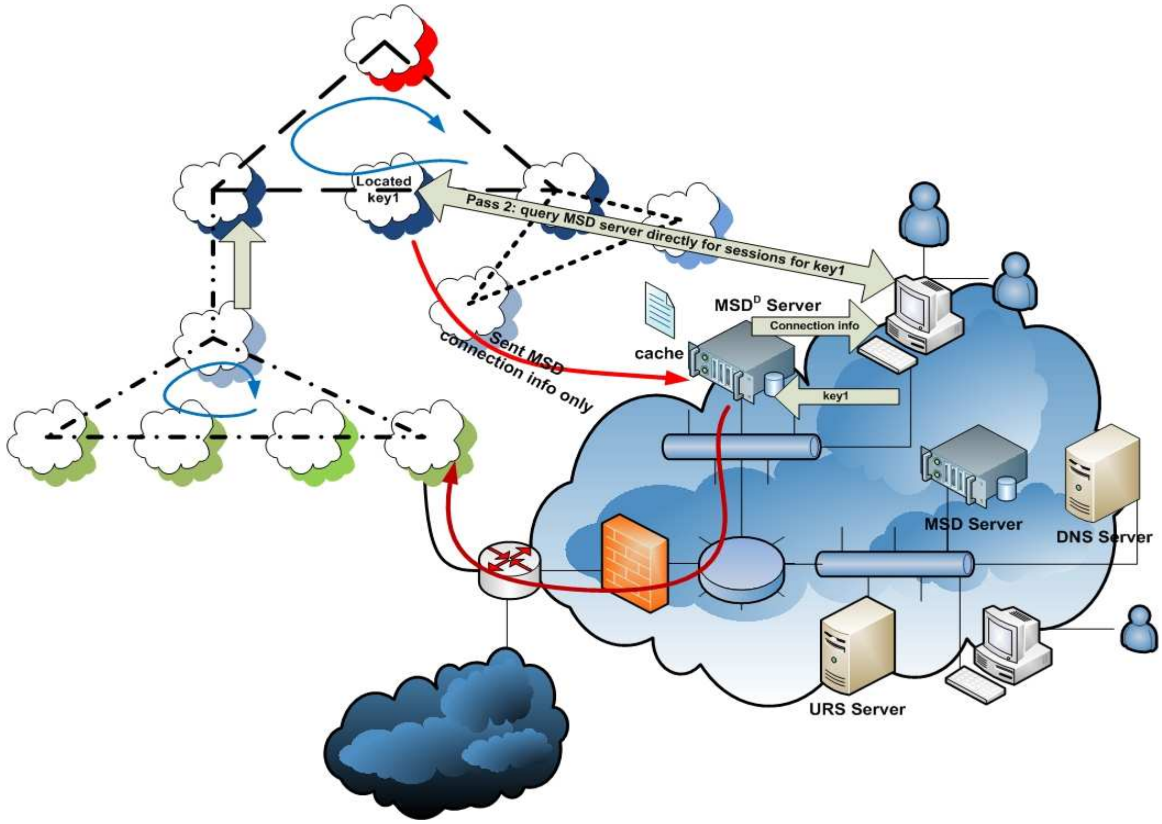

Figure 6: modified mDNS keyword search

any cache entry. $t_{last-access}$ is the time when the cache entry was last accessed or updated (not counting the current access that modifies the entry). And t_{curr} is the current system time when this entry is being accessed. After computing the score, the $t_{last-access}$ is replaced by t_{curr} value and *freq* value updated. The use of caches should reduce the query time significantly compared to a situation where no cache is used.

5. SIMULATION STRATEGY

In order to evaluate the performance of mDNS with respect to various internal parameters, we have designed a clever simulation strategy. Our simulation strategy allows us to test an actual software implementation of mDNS in a virtual domain environment without overburdening the host machine.

We have developed a DNS simulator application. It allows us to simulate multiple domains on a single machine just by running multiple instances of the simulator. It can perform virtual URL translations as well. Using multiple instances of DNS simulator and setting appropriate values for *PMCAST*, *CMCAST* and *MSD-LOCAL-MCAST* we are able to simulate a particular domain. And using pointers to parent DNS and children domain DNS servers in the simulator we are able to simulate a connected hierarchical network domain arrangement.

We then run URS servers on port numbers that match the corresponding entries in the DNS simulators. We start our MSD servers by pointing to the ports on which our DNS simulators are running; this enables the MSD servers to find out about URS servers and then using this information they query the URS servers to locate the details on *PMCAST*, *CMCAST*, and *MSD-LOCAL-MCAST* channels. From here the MSD servers and URS servers execute exactly in same manner as if they were installed and running in a real network domain.

An alternate approach would have been to use multiple instances of virtual machines (VM) to simulate each domain. This approach is highly taxing in terms of host machine resources. Each VM consumes significant resource and so one is limited to running only a few instances on any given machine. Using our strategy, we are able to simulate 10 - 15 domains on our test machine without incurring a significant performance penalty. Since the type of simulation data we seek to measure, e.g., the number of domain hops, number of keyword routing table updates, etc., is not dependent on the finer details of a network simulation such as the physical layer, data links, and lower level interconnections, our simulation strategy does not invalidate our data.

Here are the system details of our host machine used in the simulation runs:

```
CPU: Intel Core2 Quad Q6600 @ 2.40 GHz
Memory: 5120 MB
OS: Windows Vista Home Premium 64 bit
```

We developed a simulator controller application to automate the simulation process. The controller takes parameters such as the order in which to start each of the simulated domains and inter-domain startup delay parameters. Using this we can simulate a scenario where a network domain joins the mDNS hierarchy in a controlled sequence using controlled delays between subsequent joins. One can study the effects of networks joining and leaving the mDNS hierarchy on the overall performance of the architecture.

Between two simulation runs, which includes changing the MSD configurations according to the pre-specified simulation strategy, the simulator controller takes care to ensure the underlying host machine paging system stabilizes. The simulator controller is capable of starting simulated domains, launching mDNS components for each domain, waiting for the mDNS hierarchy to stabilize (in terms of routing table stability), killing all mDNS components, changing system parameters for the next run and restarting the simulation after ensuring the host machine environment is stable (paging-subsystem). All these are fully automated relieving us from manual interventions between simulation runs.

5.1 Initial Simulation Goals

In order to improve stability in the *mDNS* hierarchy, a node reports to its parent the total count of domains in the domain tree rooted at itself (including itself), in a controlled manner. If the true count was to be notified to the parent nodes at every periodic reporting cycle, then just a single addition of a domain in the hierarchy could result in global reassignment of hash-spaces among the participating, domains thus resulting in routing table updates globally. This situation is clearly not desirable. In our design, at every reporting cycle the nodes report to their parent in accordance with this formula:

$$\delta_{count} = count_{actual} - count_{reported}$$

$$\delta_{frac} = \frac{\delta_{count}}{count_{reported}}$$

$$0 < \alpha \leq \beta$$

If $\delta_{frac} \leq \alpha$ then δ_{count} value remains unchanged and the same value continues to be reported to the parent domain. If $\alpha < \delta_{frac} \leq \beta$, the hash space is re-assigned among the existing children nodes while the node count reported to the parent remains unchanged. But if $\delta_{frac} > \beta$ then the node count being reported to the parent node is updated and made equal to the actual (true) node count. The internal reassignment in the second case can lead to routing updates only in the subtree of the hierarchy rooted at the node where this reassignment took place. But in the third scenario, where the count being reported is updated to reflect the true child count, a routing update on a global scale could result. We designed our initial simulation to test the effect of α and β on the overall stability.

5.1.1 Simulation Components & Analysis Strategy

We tested with 10 virtual domains, starting them in multiple permutations and using different delays between consecutive domain startups. We collected the total number of routing table updates any domain underwent before stabilizing, the time taken for routes to be stabilized, and measured the hash-space assignment skew and control bandwidth used up among the participating domains. Hash skew for each domain is measured using this formula:

$$Hash^{skew} = |(Hash_{frac})^{assigned} - \frac{1}{\sum domains}|$$

We then combined the three data, namely, number of route updates, time taken to stabilize and hash skew, using weights of 0.5, 0.3 and 0.2 to arrive at a weighted score. **Lower weighted score represents better performance**. Before linear combination, each data value was normalized to fall in the same approximate range of magnitude. This was done to prevent bias for any one element in computation of the weighted score. For example generally *hash-skew* was of magnitude 10^{-2}, *stabilization-time* (in seconds) was of magnitude 10^2 and *count of routing updates* was in the order of 10^1, so before computing the weighted score, we multiplied *hash-skew* by 100 to convert it into a percent and divided *stabilization-time* by 60 to convert into minutes.

5.1.2 Simulation Results

In order to see the effect of network topology on optimal ranges of α and β, we linked our virtual domains in accordance to schemes shown in figure (7). Scenario 1 shows a somewhat balanced hierarchical domain arrangement, whereas scenarios 2 and 3 shows the two extremes of domain linkage schemes. Scenario 2 is the two level scenario where there is only one parent domain (flat arrangement, tree of height two) and scenario 3 is the other extreme where all the domains are linked in a linear order (tree of height 10). In the figure, the direction of arrow shows the relationship *is a child of*, e.g., A → B means A *is a child of* B.

For all three scenarios we configured the simulation controller to start the virtual domains according to the permutation: [10, 4, 5, 6, 1, 2, 7, 8, 9, 3] and inter-domain delay values [5, 5, 5, 10, 30, 600, 5, 5, 300, 30]. The value in permutation location i acts as pointer to position in the delay-list for locating the delay value to wait before starting the next domain. This is how the simulation controller acts: it first starts virtual domain 10, looks into 10^{th} place in the delay-list, finds the value 30, waits for 30 seconds before starting the virtual domain 4 and so on. Another set of values that we used in our simulation was domain start-up permutation value [10, 1, 4, 5, 2, 3, 6, 7, 8, 9] and delay values [5, 5, 5, 5, 5, 540, 5, 5, 5, 5].

Figure 8: scenario 1 permutation 1 plot.

Figure (8) shows the weighted score 3-D plot for domain

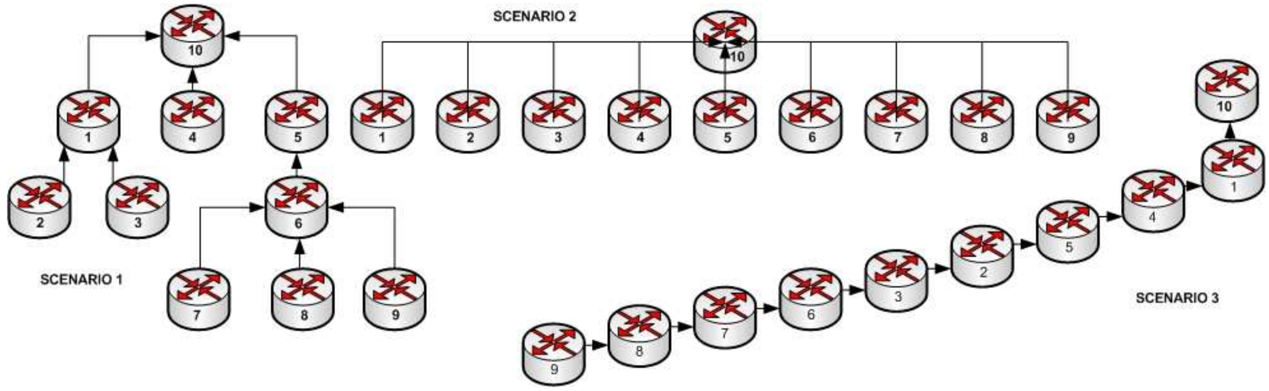

Figure 7: various simulation scenarios.

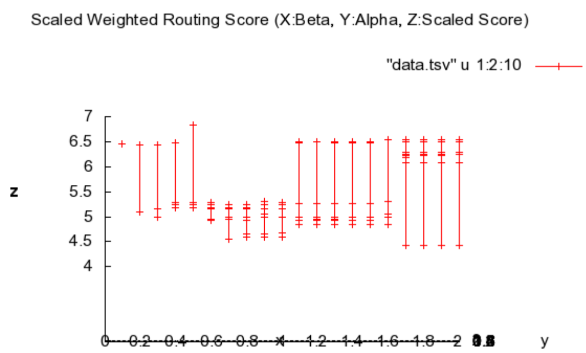

Figure 9: scenario 1 permutation 1 plot : y-z view.

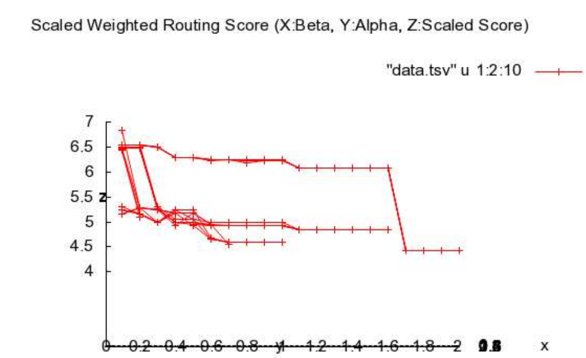

Figure 10: scenario 1 permutation 1 plot : x-z view.

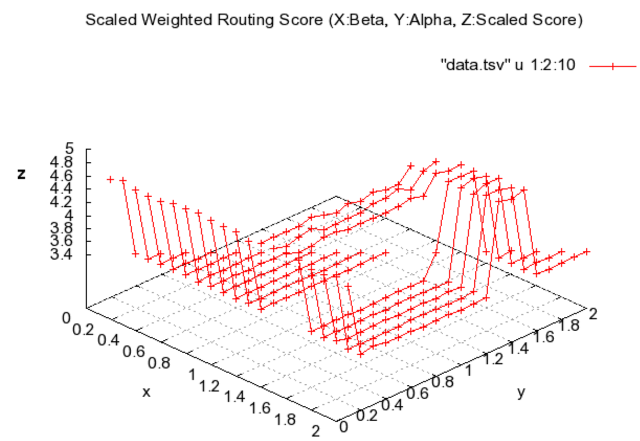

Figure 11: scenario 2 permutation 1 plot : 3-d view.

scenario 1 and for the first permutation and delay list values. The x-axis represents β values from 0.1 to 2.0, y-axis represents α ranging from 0.1 to β, and the z-axis represents the scaled weighted score. After rotation of the above plot, y-z view is shown in figure (9) while figure (10) shows the x-z view.

Looking into the three different orientations of the same plot, one can see that for configuration 1 setup, the best performance is achieved if $\alpha, \beta \in \{1.7, 1.8, 1.9, 2.0\}$ with $\alpha \leq \beta$. For the same scenario, scenario 1 and permutation set 2, the simulation results showed optimal system performance for $\alpha, \beta \in \{0.7, 0.8, 0.9, 1.0\}$ with $\alpha \leq \beta$.

Figure (11) shows that for configuration 2 and permutation 1, the optimal system performance is achieved for $\alpha \in \{0.4 - 1.0\}$ and $\beta \in \{1.6 - 2.0\}$ to report some of the values. For same scenario but with permutation 2, our simulation suggested the optimal system performance with $\alpha \in \{1.0 - 2.0\}$ with $\beta \in \{1.2 - 2.0\}$ with $\alpha \leq \beta$.

Figure (12) shows the 3-d plot for configuration 3 and permutation 1. The system performed better with $\alpha, \beta \in \{1.2 - 2.0\}$ with $\alpha \leq \beta$. And for permutation 2, the system performed better again for $\alpha, \beta \in \{1.2 - 2.0\}$ with $\alpha \leq \beta$ (plots for permutation 2 have not been shown here). Detailed experimental data and plots can be accessed at [1].

Figure 12: scenario 3 permutation 1 plot : 3-d view.

5.1.3 *Interpretations*

Considering the simulation results, it is clear that the choice of α and β depends on the network topology. A system administrator is free to choose a value of his liking although it is advisable to follow the common selection guidelines for the full hierarchy. In order to maintain global routing table stability, a relatively high value of β is suggested, and for routing table stability at the subtree level, a higher value of α is advised.

One drawback for choosing a high value for α and β is the possibility of somewhat uneven hash-space assignments across participating nodes. Hence in order to maintain the balance between workload and routing stability, a comprise value of α and β must be found.

It is important to make clear the fact that routing unstability does not mean that the routing tables will forever be unstable. But addition or removal of a domain to/from the hierarchy will result in hash space reassignments, which will lead to route table updates. But once the hierarchy becomes stable, routing tables will become stable as well. But one would want to reduce this period of instability while the whole hierarchy is still reorganizing, as it could have an adverse impact on quality of service.

6. CONCLUSION

In this paper we have documented our strategy to make *mDNS* support ASM and SSM networks. We gave our simulation model that allows us to efficiently simulate multiple domains on a single host. We presented preliminary data that strongly suggests that the choice of values for route stabilization parameters, α and β, depends on the overall network topography, but a higher value would impart greater route stability at the possible cost of imbalance in hash-space (read work load) assignments to individual nodes.

Simulations involving various user search patterns and server workload assessments have yet to be done. We are in the process of fine tuning the implementation to support SSM networks as well as standardizing the session registration module to support numerous session types.

7. ACKNOWLEDGMENTS

We would like to thank Dr. Randy Chow for helping us during editing phase. We appreciate the candid discussions with other CONS Lab members who critiqued our work, suggested improvements and listened patiently to us over several internal meetings. Notably among those are InKwan Yu, Mahendra Kumar, and Panoat Chuchaisri.

8. REFERENCES

[1] mdns simulation site. http://www.cons.cise.ufl.edu/mdns/.

[2] S. Bhattacharyya. An Overview of Source-Specific Multicast (SSM). RFC 3569 (Informational), July 2003.

[3] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376 (Proposed Standard), Oct. 2002. Updated by RFC 4604.

[4] S. Deering. Host extensions for IP multicasting. RFC 1112 (Standard), Aug. 1989. Updated by RFC 2236.

[5] W. Fenner. Internet Group Management Protocol, Version 2. RFC 2236 (Proposed Standard), Nov. 1997. Obsoleted by RFC 3376.

[6] H. Garcia-Molina. Elections in a distributed computing system. *IEEE Transactions on Computers*, 31(1):48–59, 1982.

[7] P. Harsh and R. Newman. mdns - a proposal for hierarchical multicast session directory architecture. In *International Conference on Internet Computing*, pages 31–37, 2008.

[8] P. Harsh and R. Newman. Efficient distributed search for multicast session keywords. In *International Conference on Internet Computing*, pages 31–37, 2009.

[9] P. Harsh and R. Newman. Using geo-spatial session tagging for smart multicast session discovery. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 22–27, New York, NY, USA, 2009. ACM.

[10] R. Karedla, J. S. Love, and B. G. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, 1994.

[11] D. Meyer and P. Lothberg. GLOP Addressing in 233/8. RFC 3180 (Best Current Practice), Sept. 2001.

[12] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The lru-k page replacement algorithm for database disk buffering. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 297–306, New York, NY, USA, 1993. ACM.