

Adaptive Jitter Control for UPnP M-Search

Kevin Mills and Christopher Dabrowski

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

Abstract – Selected service-discovery systems allow clients to issue multicast queries to locate network devices and services. Qualifying devices and services respond directly to clients; thus, in a large network, potential exists for responses to implode on a client, overrunning available resources. To limit implosion, one service-discovery system, UPnP, permits clients to include a jitter bound in multicast (M-Search) queries. Qualifying devices use the jitter bound to randomize timing of their responses. Initially, clients lack sufficient knowledge to select an appropriate jitter bound, which varies with network size. In this paper, we characterize the performance of UPnP M-Search for various combinations of jitter bound and network size. In addition, we evaluate the performance and costs of four algorithms that might be used for adaptive jitter control. Finally, we suggest an alternative to M-Search for large networks.

I. INTRODUCTION

Selected service-discovery systems allow clients to issue multicast queries to locate network devices and services [1, 3]. Qualifying devices and services respond directly to clients; thus, in a large network, potential exists for responses to implode on a client, overrunning available resources. This implosion problem also arises in other protocols that support multicast queries and responses [4-7]. To limit implosion, one service-discovery system, Universal-Plug-and-Play¹ (UPnP) permits clients to include a *jitter bound* (MX) in multicast (M-Search) queries. Each qualifying device jitters its response time by randomly selecting a delay up to MX . Initially, clients lack sufficient knowledge to select an appropriate jitter bound, which varies with network size.

In this paper, we model the UPnP M-Search mechanism and characterize performance for various combinations of jitter bound and network size. The resulting performance curves should help designers of UPnP clients to understand the effects of selecting particular jitter bounds. We also consider four algorithms that might be used to adaptively control jitter in UPnP M-Search. We compare the performance of the adaptive algorithms against each other and against a fixed jitter bound. We discuss the costs associated with adaptation. These costs lead us to suggest an alternative approach to M-Search for large networks. The

insights we provide should help designers of service-discovery systems to create architectures that can scale across a variety of network sizes, while achieving effective and efficient performance.

The remainder of this paper is organized as follows. Section II describes the UPnP M-Search mechanism, defines an experiment and related metrics to characterize M-Search performance, and illustrates M-Search performance for varying jitter bounds and network sizes. Section III outlines four algorithms that might be used to adaptively adjust M-Search jitter bounds, and compares the performance of the algorithms against each other and against a fixed jitter bound. Section III also discusses the costs and assumptions underlying adaptation. Section IV suggests an alternative to M-Search for use in large networks. Section V gives our conclusions.

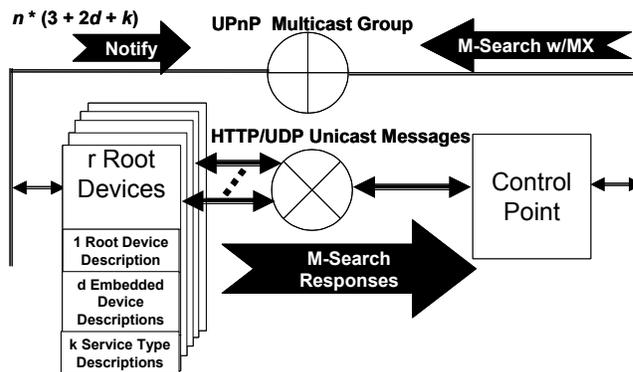


Fig. 1. General Operation of UPnP Discovery

II. CHARACTERIZING M-SEARCH PERFORMANCE

Fig. 1 depicts the general operation of device and service discovery in UPnP. UPnP consists of two main elements: root devices (servers) and control points (clients). A UPnP network may contain $r \geq 0$ root devices. Each root device contains $d \geq 0$ embedded devices and $k \geq 0$ unique service types, where each device and service has a specific type. Each root device also contains a hierarchical description that defines the capabilities of $1 + d + k$ elements: the root device and each of its embedded devices and unique service types. The description can be rather lengthy; thus, UPnP provides a two-step process for obtaining descriptions. A control point first discovers devices or services of interest by type or identity, and then requests the related descriptions.

¹ Certain commercial products and standards are identified in this paper to describe our study adequately. The National Institute of Standards and Technology neither recommends nor endorses these products or standards as the best available for the purpose.

UPnP provides two discovery modes: lazy and aggressive. Lazy discovery uses periodic announcements sent by each root device on the UPnP multicast group (Notify in Fig. 1). At each announcement interval, each root device sends $n(3 + 2d + k)$ Notify messages to identify the root device (and its identity and type), each embedded device (by identity and type), and each unique service type (by type). The UPnP specification recommends a duplicate transmission factor, n , “due to the unreliable nature of UDP” (user-datagram protocol) [1]. Control points listen for announcements to discover the existence of various devices and services. The UPnP specification sets the announcement interval at 30 min. or more. For this reason, control points may use aggressive discovery to get an immediate picture of available services.

Aggressive discovery commences when a control point multicasts an M-Search query, which specifies an interest (that can include specific devices, device types, service types, or all) and a jitter bound (MX in Fig. 1). Root devices listen for M-Search queries to determine if any contained items are of interest. Each root device sends $3n$ responses if the root device qualifies, and $2n$ and n responses respectively for each qualifying embedded device and service type. If the query asks for everything (SSDP_ALL), each root device responds with the same $n(3 + 2d + k)$ messages used in lazy discovery. To mitigate a potential implosion of responses, each root device waits a random time, uniformly distributed in the range $0..MX$ s, before transmitting its responses in a burst.

A. Experiment Definition

To characterize M-Search performance, we used SLX™ [8] to construct a simulation model representing the topology shown in Fig. 1, deployed in a 10-Mbps Ethernet. Since the UPnP specification allows implementation choices, we based those aspects of our model on UPnP software available publicly from Intel [9]. We allow the number of root devices, r , to vary from 10 to 200 by 10-step increments. Each root device includes an identical count of embedded devices ($d = 2$) and service types ($k = 3$). We set $n = 2$, the default value in the Intel implementation of UPnP. We allow MX to vary from 2 to 40 in 2-s increments. For each combination of r and MX , an M-Search task in a single control point issues a query requesting SSDP_ALL, which elicits $n(3 + 2d + k) = 20$ 200-byte response messages from each root device; thus, aggregate implosion ranges from 200 ($r = 10$) to 4000 ($r = 200$) response messages. (To keep our graphs legible, we display results over only $r = 10..100$ and $MX = 2..20$.) We limit the M-Search task to buffer no more than 40 messages, dropping the excess. We allow the control point task to execute every 5 ms, processing one response message at each execution (200 messages/s maximum rate). For each message, the task examines a cache to see if a new discovery occurs, adding items to the cache as required. The task takes c ms to process a message, where c varies with the cache

size. When finished, the task reschedules itself to execute in $5 - c$ ms. If $c \geq 5$, the task executes immediately.

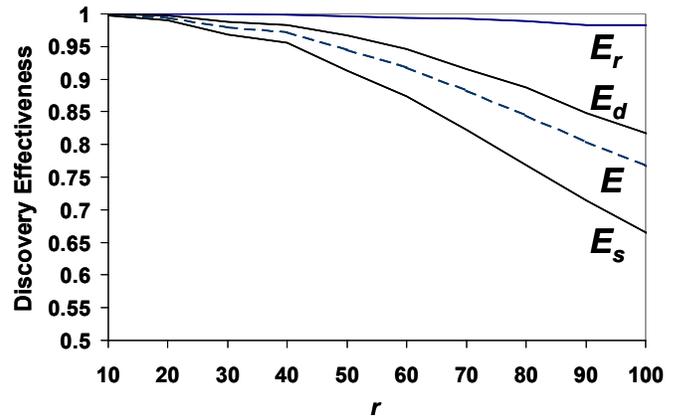


Fig. 2. Overall discovery effectiveness (E) compared against discovery effectiveness by entity type: root devices (E_r), embedded devices (E_d), and services (E_s). [$MX = 10$ s]

B. M-Search Performance

We measure system performance with four metrics: discovery effectiveness (E), discovery latency (L), buffer utilization (B), and processor usage (P). Given a network comprising $e = r + rd + rk$ entities and assuming that a control point discovers $f \leq e$ entities from responses to an M-Search, then $E = f / e$. We can also track discovery effectiveness by entity type, root devices (E_r), embedded devices (E_d), and services (E_s) as shown in Fig. 2. In the Intel implementation, each root device sends M-Search responses in the same order ($3n$ then $2dn$ then kn) and since responses earliest in the sequence are more likely to find buffer space available at the control point, root devices are more likely to be discovered than either embedded devices (next most likely) or services (least likely).

Fig. 2 also reveals that randomly jittering responses does not ensure $E = 1$, even when MX is set to a seemingly suitable value. When $r = 100$, a total of 2000 response messages will implode on the control point, which processes 200 messages/s, suggesting that 10s (2000/200) might be a suitable value for MX . Unfortunately, since each root device picks a random time to respond and then sends a burst of 20 response messages, collision periods can occur during which receive buffers are overrun in the control point. Fig. 3 displays the problem.

Even at $MX = 20$ s, collisions occur with sufficient frequency that E decays significantly beyond $r = 50$. Collisions lead to increased buffer occupancy (Fig. 4), which leads to increased likelihood of message drops. Periods of high buffer occupancy (and therefore message loss) tend to persist, as incoming messages arrive in bursts at random intervals, while the M-Search task reduces the buffer backlog at a steady rate.

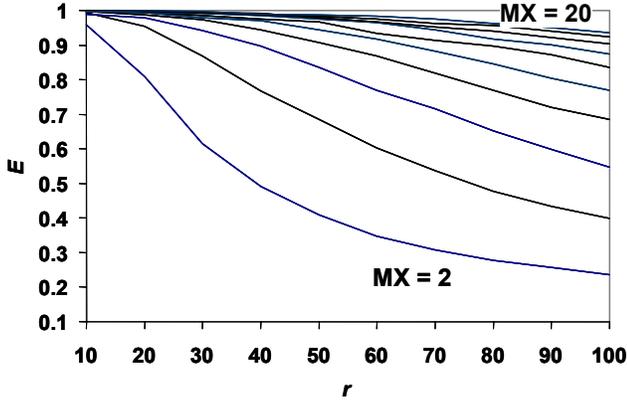


Fig. 3. Discovery effectiveness (E) for various values of MX (2s to 20s in 2s increments) as the number of root devices (r) increases.

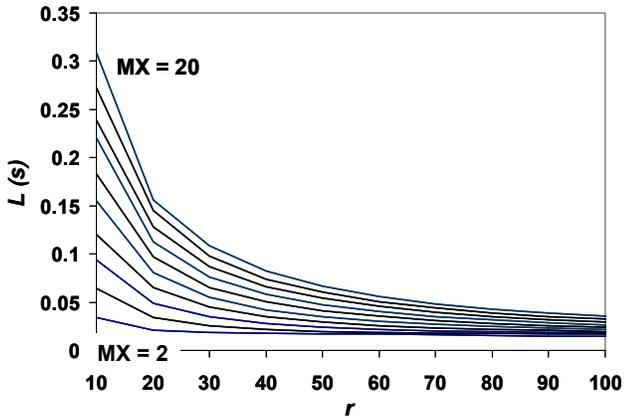


Fig. 4. Average buffer occupancy (B) as a percentage of available buffers (40 messages in this case) for various values of MX (2s to 20s in 2s increments) as the number of root devices (r) increases.

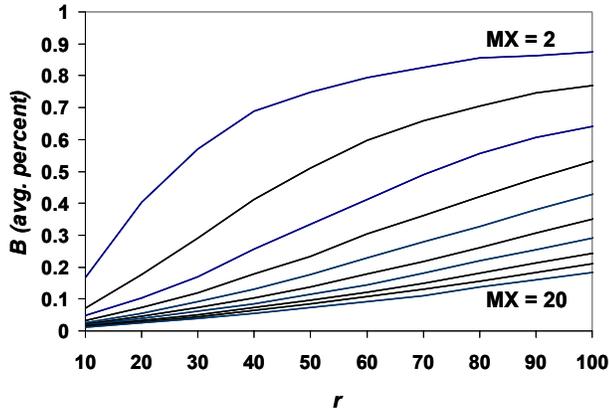


Fig. 5. Average discovery latency (L) for various values of MX (2s to 20s in 2s increments) as the number of root devices (r) increases.

Buffer size at the control point can be augmented to accommodate additional responses; however a suitable buffer size may be difficult to determine given the random nature of response jitter (and unknown network size). Instead, a

control point could increase MX ; but then, as Fig. 5 shows, discovery latency will grow.

We define discovery latency (L) as the time that elapses between successive discoveries of new entities in the network. As Fig. 5 shows, when MX is large compared to network size the gap between new discoveries grows for a control point. An increased MX also leads to fewer buffer overruns, which increases the discovery effectiveness for a control point. As discovery effectiveness increases, the discovery cache in the control point increases in size, which causes the M-Search task to spend more processor cycles examining each response message (Fig. 6). This increase occurs because the M-Search task must look through more cache entries to determine if a new entity has been discovered, and to insert a related cache entry if needed. For relatively large values of MX , processor utilization increases linearly with network size, though this would change if we modeled more efficient search algorithms. For relatively small values of MX , growth in processor utilization levels off with the size of the discovery cache maintained by the M-Search task.

III. ADAPTIVE JITTER CONTROL

We propose four algorithms for adaptive-jitter control, and then illustrate the performance arising from each. We also discuss the costs and assumptions underlying the algorithms. Some other algorithms to address multicast query-response implosion can be found in the literature [10,11].

A. Four Adaptive Jitter-Control Algorithms

In adaptive-jitter control, each root device independently estimates the time it will take for all root devices to respond to each M-Search query. Each root device then uses its estimate to determine a time to send its own responses (if any). Included in each response message is a value recommending how long the control-point M-Search task should listen for responses. With this approach, the M-Search task need not guess an appropriate MX value.

Each root device listens on the UPnP multicast group for Notify messages (which include a caching time, or *max-age*) sent by all root devices, and builds a map (NM) of devices and services in the network. For each root device, NM includes: the identity and type of the root device and all embedded devices and unique service types, a *max-age*, and an estimate of the redundant transmission factor (n). Estimates of n exploit the fact that in the Intel implementation each message is sent n times before the next message. Listening root devices apply a time threshold to identify duplicate messages and then compute an average n for each announcing root device.

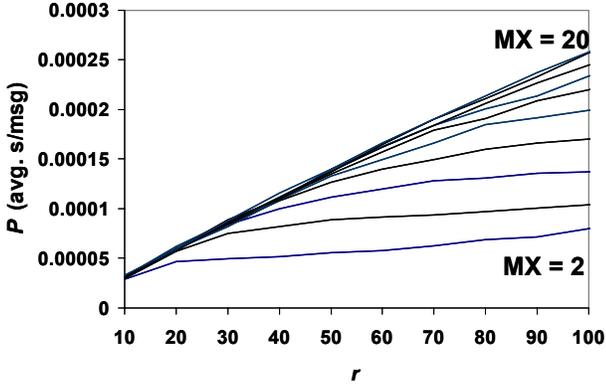


Fig. 6. Average processor time (P) in seconds/message for a Control Point M-Search task to examine a response for various values of MX (2s to 20s in 2s increments) as the number of root devices (r) increases.

M-Search queries issued by the M-Search task include a rate, R , at which the task can consume messages, and also an $MX \geq 0$. Upon receiving an M-Search query, a root device cycles through its NM to estimate how many response messages will be sent by all root devices, using R to also estimate when the last set of responses should commence ($Jstart$) and finish ($Jend$) – under an assumption that messages will be sent consecutively at rate R . During this process, a root device can also note a time (Stx) when it should send its own responses – under an assumption that root devices will send messages sequentially in the ascending order of their unique identities. Using this information, we devised four adaptive jitter-control algorithms: Random Burst (RB), Random Paced (RP), Scheduled Burst (SB), and Scheduled Paced (SP).

In the random algorithms (RB and RP), a root device selects a time, T_r , randomly distributed uniformly on the interval $[0, Jstart]$, to send its response messages. The root device includes $Jend$ in each response so that the M-Search task will learn an appropriate time interval to listen. The root device will not respond if $0 < MX < T_r$. In the RB variant of the algorithm, the root device bursts its response messages. In the RP variant, the root device paces its responses at rate R .

In the scheduled algorithms (SB and SP), a root device sends its response messages at Stx ; however, the root device will not respond if $0 < MX < Stx$. Response messages are sent in a burst (SB) or at rate R (SP). The root device includes $Jend$ in each response message.

B. Performance of Adaptive Jitter Control

Fig. 7 illustrates discovery effectiveness (E) for each adaptive jitter-control algorithm as the number of root devices (r) increases from 10 to 300. For comparison, we include the performance of a fixed $MX = 33$ s, which is the $Jstart$ value estimated by each root device when $r = 300$.

Scheduling transmissions achieves full effectiveness ($E = 1$). On the other hand, randomizing transmissions leads to

collisions in the receive buffers, and then to buffer overflows and lost discoveries. Pacing responses (RP) results in fewer buffer overflows, but fails to eliminate them. While RP more closely matches arrival rate with service rate, Fig. 8 indicates a nearly identical average buffer occupancy for RP and RB.

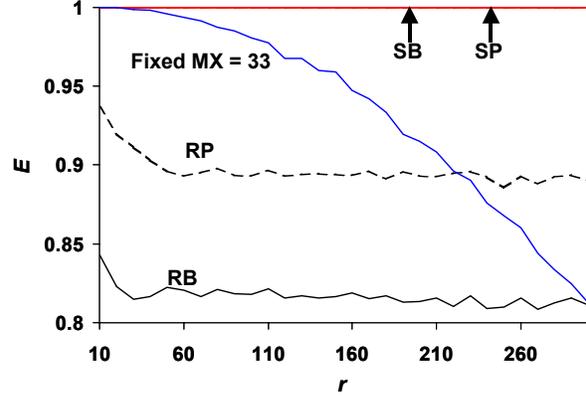


Fig. 7. Discovery effectiveness for four adaptive jitter control algorithms and one fixed jitter bound as the number of root devices increases

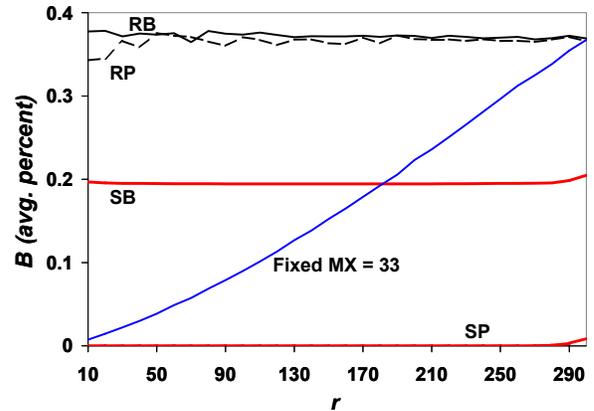


Fig. 8. Average buffer occupancy for various jitter-control algorithms as the number of root devices increases.

Buffer utilization is very low for SP because scheduling eliminates collisions and responses arrive at the rate at which the M-Search task can process them. While SB avoids collisions, responses arrive in (20-message) batches, leading to a higher average buffer utilization. Fig. 8 also shows that each of the adaptive jitter-control algorithms yields a nearly stable average buffer utilization (but at different occupancy levels), while buffer utilization for a fixed MX varies with the relationship between MX and r .

Fig. 9 illustrates that all the adaptive jitter-control algorithms provide consistently low average discovery latency, L , despite variation in network size, which is not the case for a fixed MX , where latency varies with the relationship between MX and r . The scheduled algorithms (SB and SP) perform slightly better than the random algorithms (RB and RP) because buffer overflows resulting

from jitter randomization cause some discoveries to be lost, which tends to lengthen the time between new discoveries.

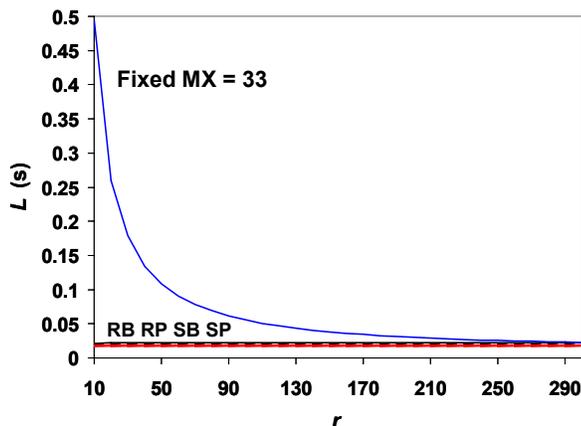


Fig. 9. Average discovery latency of various jitter-control algorithms as the number of root devices increases

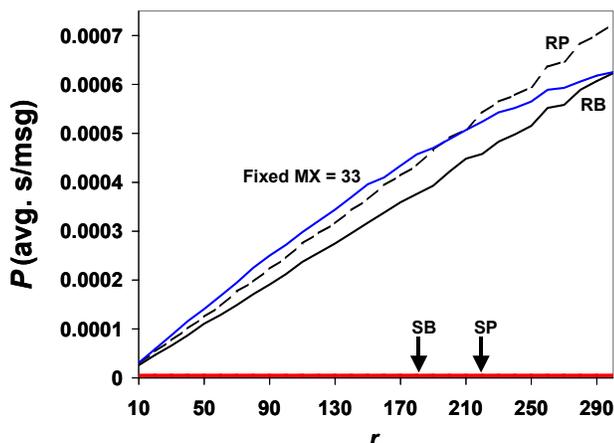


Fig. 10. Average processor seconds per message used by the M-Search task for various jitter-control algorithms and increasing network size.

The scheduled algorithms also lead to some serendipitous effects on processor utilization in the M-Search task (Fig. 10). Since scheduled responses arrive in order, the M-Search task need not conduct a search of its discovery cache for each response message. Instead, the M-Search task checks to see if a response can be inserted into the cache at the current insertion point. Only if this is not the case does the M-Search task need to search its cache. In our experiments all scheduled responses arrived in the expected order; thus, Fig. 10 shows that both SB and SP consume a small, fixed amount of processor time for each message.

Fig. 10 also shows that the cache search required by the random algorithms causes processor utilization to increase as the number of discovered entities increase. Processor utilization for RP always exceeds that for RB because the RP algorithm discovers a greater percentage of entities. Random jitter with a fixed $MX = 33$ s uses more processor time than either RP or RB up until about $r = 200$, where RP proves

more effective and thus requires more processor time. The rate of increase in processor utilization for the fixed MX continues to decline, reaching the same value as RB when $r = 300$ (and $Jstart = MX = 33$ s).

C. Costs and Caveats

Adaptive jitter control comes with two costs: memory and processing time in root devices. Each root device creates, stores, and maintains a network map (NM) of size

$$S = h + \sum_{i=0}^r [p + q \cdot (d_i + k_i + t_i)], \quad (1)$$

where h is the cache-header size, p is the root-device header size, q is per-entry content size, r is the number of root devices, and d_i , k_i , and t_i represent respectively the number of embedded devices, service types, and device types maintained by root device i . In our experiments, S varies from about 1.2 ($r = 10$) to 37 ($r = 300$) Kbytes.

To process an M-Search query, a root device must scan NM to estimate the likely number of responses that will be issued by all root devices. During the scan, a root device also purges stale entries. Thus, for each M-Search query a root device uses processor time

$$C = \sum_{i=0}^r [x \cdot (1 + d_i + k_i)] + (y \cdot O), \quad (2)$$

where x is processor time to scan one entry, y is processor time to purge one root-device, and O is the number of stale root-device entries found during the scan. In our experiments, for SSDP_ALL queries with no stale entries, C varies between 0.3 ($r = 10$) and 9 ($r = 300$) ms.

In addition to memory and processing costs, the scheduled algorithms assume that each root device has the same knowledge about network state (NM). Absent this assumption, root devices would schedule collisions, leading to lower discovery effectiveness. This same- NM assumption should hold in steady state, where all root devices have had a chance to announce themselves and where changes occur infrequently. Of course, when a root device enters a network it must acquire NM to participate effectively in adaptive jitter control.

IV. DISCOVERY IN LARGE NETWORKS

Most discovery protocols provide for recurring announcements at a known interval. For example, the Jini protocol recommends announcements every 120s [11]. Recurring announcements permit a network device to listen for a period of time over which a reasonably complete NM might be constructed. Unfortunately, the minimum announcement interval specified for UPnP is 30 min., which might prove too long a period for a device to wait before participating on the network. To compensate for this lengthy announcement interval, UPnP provides the M-Search

mechanism so that network devices can attempt to gain a sense of the *NM* on demand. We have shown, though, limitations of the M-Search as a means to find all devices and services on the network. Some other discovery protocols [2,3] include feedback mechanisms within their multicast queries in order to provide a means of dampening responses. Using such dampening mechanisms, a short repeated burst of multicast queries (for example, Jini recommends seven queries at intervals of five seconds) might be used to obtain a reasonably complete *NM* (ignoring the possibility of temporary node and channel failures). Unfortunately, UPnP includes none of these mechanisms; thus, acquiring the *NM* needed to permit effective participation in adaptive jitter control for M-Search queries seems to require using the regular UPnP M-Search. Since we have already shown UPnP M-Search to be ineffective for this purpose, we propose an alternative to M-Search for *NM*-bootstrap and for general use in large networks.

Suppose that on startup a root device initiates a network mapping (*NM*) service with probability W . In that case, the network will contain only rW *NM* services. Then a control point, or a newly starting root device, can use M-Search (in fixed or adaptive form) to query only for instances of *NM* services. Each qualifying *NM* service can respond with the count of root devices, embedded devices, and service types known to it. Using this information, a querying node can select one *NM* service and use http-GET (HyperText Transfer Protocol) to retrieve its *NM*. Alternatively, the querying node may issue http-GETs to multiple *NM* services, and then merge the results into a signal *NM*. After retrieving a *NM*, a root device should be sufficiently bootstrapped to participate in adaptive M-Search. For a control point, querying for *NM* services will reduce (or eliminate) the need to issue SSDP_ALL M-Search queries.

A further advantage of using *NM* services can accrue as network volatility increases. As the need arises, due to increase in load or in network or node failures, a root device can choose to start a *NM* service to increase redundancy or to share the load from an increasing number of client queries. Similarly, as volatility diminishes or as network size decreases, root devices with a running *NM* service can elect to terminate the service in order to reduce network overhead.

V. CONCLUSIONS

Given the UPnP M-Search mechanism, we illustrated relationships among network size (r), jitter bound (MX), discovery effectiveness (E) and latency (L), and buffer (B) and processor (P) utilization. Specifically, we showed how an inappropriate jitter bound (MX value) in UPnP M-Search queries could significantly reduce discovery effectiveness or increase discovery latency. We outlined four algorithms that might be used for adaptive jitter control, and we explained the storage and processing costs associated with adaptation. We compared the performance of the adaptive algorithms

against each other and against a fixed MX value. The random paced (RP) algorithm yielded increased discovery effectiveness over random burst (RB). Both scheduled algorithms (SB and SP) led to better performance than either random algorithm. In particular, the scheduled paced (SP) algorithm achieved optimal performance for all metrics. We explained, however, that the performance of the scheduled algorithms would deteriorate if all root devices do not share the same picture (*NM*) of the network state.

We outlined an approach to enable root devices to bootstrap their *NM*. We suggested that control points might also use this approach to replace M-Search SSDP_ALL queries, thus avoiding the potential for an implosion of M-Search responses. Further, we hinted that the *NM*-bootstrap mechanism might be adapted to modulate redundancy and load sharing in support of aggressive discovery in UPnP networks. Further exploration of these ideas remains for future work. We also suggest that these algorithms should be investigated under various types and rates of failure.

REFERENCES

- [1] Universal Plug and Play Device Architecture, Version 1.0, Microsoft, June 8, 2000.
- [2] Erik Guttman and James Kempf. Service Location Protocol, Version 2, RFC 2608bis, January 10, 2002.
- [3] Ken Arnold et al, The Jini Specification, V1.0 Addison-Wesley 1999. Latest version is available from Sun.
- [4] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, Session Invitation Protocol, RFC 2543, March 1999.
- [5] Stuart Cheshire, DNS-based Service Discovery, Internet Draft, December 20, 2002.
- [6] O. Catrina, D. Thaler, B. Aboba, and E. Guttman, Zeroconf Multicast Address Allocation Protocol, Internet Draft, October 22, 2002.
- [7] Stuart Cheshire, Performing DNS queries via IP Multicast, Internet Draft, December 20, 2002.
- [8] James O. Henriksen, "An Introduction to SLX™", *Proceedings of the 1997 Winter Simulation Conference*, ACM, Atlanta, Georgia, December 7-10, 1997, pp. 559-566.
- [9] Preston Hunt and Ulhas Warriar, "UPnP Applications Enhance Mobile Functionality", *Intel Developer Update Magazine*, Intel, April 2002, pp. 1-5. (Intel UPnP Software Development Kit available from: <http://www.intel.com/labs/connectivity/upnp/index.htm>)
- [10] T. Imieliński and S. Goel, "Dataspace - querying and monitoring deeply networked collections of physical objects," Tech. Rep. DCS-TR-381, Rutgers University, July 1999.
- [11] B. R. Badrinath and Pradeep Sudame. "Gathercast: The design and implementation of a programmable aggregation mechanism for the Internet", *Proceedings of IEEE International Conference on Computer Communications and Networks (ICCCN)*, October 2000.