Internet Engineering Task Force
INTERNET DRAFT

Yaron Y. Goland
Ting Cai
Paul Leach
Ye Gu
Microsoft Corporation
Shivaun Albright
Hewlett-Packard Company
October 28, 1999
Expires April 2000

# Simple Service Discovery Protocol/1.0
# Operating without an Arbiter
# <draft-cai-ssdp-v1-03.txt>

## Status of this Memo



## Abstract

The Simple Service Discovery Protocol (SSDP) provides a mechanism where by network clients, with little or no static configuration, can discover network services. SSDP accomplishes this by providing for multicast discovery support as well as server based notification and discovery routing.

## Table of Contents

## 1. Changes Since 02

The entire specification has been extensively re-written. As such the reader is advised to re-read the entire specification rather than to just look for particular changes.

Removed the arbiter and related functionality.

Spec used to contain both ssdp:discover and ssdp:discovery, settled on ssdp:discover.

Changed SSDP multicast message examples to use the reserved relative multicast address "5" provided by IANA. In the local administrative scope, the only scope currently used by SSDP, this address translates to 239.255.255.250.

An application has been made for a reserved port for SSDP but no response from IANA has been received.

## 2. Introduction

[Ed. Note: In my experience, one of the best ways to enable a specification to be quickly and successfully developed is to provide a problem statement, a proposed solution and a design rationale. I came across this three-part design structure when Larry Masinter proposed it to the WebDAV WG. To that end, I have divided this spec in a similar manner. Once the specification is sufficiently mature, the problem statement and design rationale sections will be placed in a separate document and the proposed solutions will be presented for standardization.]

This document assumes the reader is very familiar with [RFC2616], [HTTPUDP], [GENA], [MAN] and [RFC2365].

### 2.1. Problem Statement

A mechanism is needed to allow HTTP clients and HTTP resources to discover each other in local area networks. That is, a HTTP client may need a particular service that may be provided by one or more HTTP resources. The client needs a mechanism to find out which HTTP resources provide the service the client desires.

For the purposes of this specification the previously mentioned HTTP client will be referred to as a SSDP client. The previous mentioned HTTP resource will be referred to as a SSDP service.

In the simplest case this discovery mechanism needs to work without any configuration, management or administration. For example, if a user sets up a home network or a small company sets up a local area network they must not be required to configure SSDP before SSDP can be used to help them discover SSDP services in the form of Printers, Scanners, Fax Machines, etc.

It is a non-goal for SSDP to provide for multicast scope bridging or for advanced query facilities.

## 2.2. Proposed Solution

### 2.2.1. Message Flow on the SSDP Multicast Channel

The following is an overview of the messages used to implement SSDP.

SSDP clients discover SSDP services using the reserved local administrative scope multicast address 239.255.255.250 over the SSDP port [NOT YET ALLOCATED BY IANA].

For brevity's sake the SSDP reserved local administrative scope multicast address and port will be referred to as the SSDP multicast channel/Port.

Discovery occurs when a SSDP client multicasts a HTTP UDP discovery request to the SSDP multicast channel/Port. SSDP services listen to the SSDP multicast channel/Port in order to hear such discovery requests. If a SSDP service hears a HTTP UDP discovery request that matches the service it offers then it will respond using a unicast HTTP UDP response.

SSDP services may send HTTP UDP notification announcements to the SSDP multicast channel/port to announce their presence.

Hence two types of SSDP requests will be sent across the SSDP multicast channel/port. The first are discovery requests, a SSDP client looking for SSDP services. The second are presence announcements, a SSDP service announcing its presence.

### 2.2.2. SSDP Discovery Information Caching Model

The following provides an overview of the data provided in a SSDP system.

Services are identified by a unique pairing of a service type URI and a Unique Service Name (USN) URI.

Service types identify a type of service, such as a refrigerator, clock/radio, what have you. The exact meaning of a service type is outside the scope of this specification. For the purposes of this specification, a service type is an opaque identifier that identifies a particular type of service.

A USN is a URI that uniquely identifies a particular instance of a service. USNs are used to differentiate between two services with the same service type.

In addition to providing both a service type and a USN, discovery results and presence announcements also provide expiration and location information.

Location information identifies how one should contact a particular service. One or more location URIs may be included in a discovery response or a presence announcement.

Expiration information identifies how long a SSDP client should keep information about the service in its cache. Once the entry has expired it is to be removed from the SSDP client's cache.

Thus a SSDP client service cache might look like:

| USN URI | Service Type URI | Expiration | Location |
|---|---|---|---|
| upnp:uuid:k91... | upnp:clockradio | 3 days | http://foo.com/cr |
| uuid:x7z... | ms:wince | 1 week | http://msce/win |

In the previous example both USN URIs are actually UUIDs such as upnp:uuid:k91d4fae-7dec-11d0-a765-00a0c91c6bf6.

If an announcement or discovery response is received that has a USN that matches an entry already in the cache then the information in the cache is to be completely replaced with the information in the announcement or discovery response.

## 2.3. Design Rationale

[Ed. Note: In my own experience one of the most powerful ways to explain design rationale is in a question/answer form. Therefore I have used that format here.]

### 2.3.1. Message Flow on the SSDP Multicast Channel

Please see section 8.3 for more design rationale behind our use of multicasting.

#### 2.3.1.1. Why use multicast for communication?

We needed a solution for communication that would work even if there was no one around to configure things. The easiest solution would have been to build a discovery server, but who would set the server up? Who would maintain it? We needed a solution that could work even if no one had any idea what discovery was. By using multicasting we have the equivalent of a "party channel." Everyone can just grab the channel and scream out what they need and everyone else will hear. This means no configuration worries. Of course it brings up other problems which are addressed throughout this specification.

#### 2.3.1.2. Why use a local administrative scope multicast address?

Multicasting comes in many scopes, from link local all the way to "the entire Internet." Our goal is to provide for discovery for local area networks not for the entire Internet. LANs often are bridged/routed so a link local multicast scope was too restrictive. The next level up was a local administrative scope. The idea being that your administrator decides how many machines should be grouped together and considered a "unit". This seemed the ideal scope to use for a local discovery protocol.

#### 2.3.1.3. Why does SSDP support both service discovery requests as well as service presence announcements?

Some discovery protocols only support discovery requests, that is, the client must send out a request in order to find out who is around. The downside to such solutions is that they tend to be very expensive on the wire. For example, we want to display to our user all the VCRs in her house. So we send out a discovery request. However our user has just purchased a new VCR and, after starting our program, plugged it in. The only way we would find out about the new VCR and be able to display it on our user's screen is by constantly sending out discovery requests. Now imagine every client in the network having to send out a torrent of discovery requests for service they care about in order to make sure they don't miss a new service coming on-line.

Other systems use the opposite extreme, they only support announcements. Therefore, when our user opens the VCR display window we would just sit and listen for announcements. In such systems all the services have to send out a constant stream of announcements in order to make sure that no one misses them. Users aren't the most patient people in the world so each service will probably need to announce itself at least every few seconds. This constant stream of traffic does horrible things to network efficient, especially for shared connections like Ethernets.

SSDP decided to adopt a hybrid approach and do both discovery and announcements. This can be incredibly efficient. When a service first comes on-line it will send out an announcement so that everyone knows it is there. At that point it shouldn't ever need to send out another announcement unless it is going off-line, has changed state or its cache entry is about to expire. Any clients who come on-line after the service came on-line will discover the desired service by sending out a discovery request. The client should never need to repeat the discovery request because any services that subsequently come on-line will announce themselves. The end result is that no one needs to send out steady streams of messages. The entire system is event driven, only when things change will messages need to be sent out. The cost, however, is that the protocol is more complex. We felt this was a price worth paying as it meant that SSDP could be used successfully in fairly large networks.

#### 2.3.1.4. Doesn't the caching information turn SSDP back into a "announcement driven" protocol?

Discovery protocols that only support announcements generally have to require services to send announcements every few seconds. Otherwise users screens will take too long to update with information about which services are available.

SSDP, on the other hand, allows the service to inform clients how long they should assume the service is around. Thus a service can set a service interval to seconds, minutes, days, weeks, months or even years.

Clients do not have to wait around for cache update messages because they can perform discovery.

### 2.3.2. SSDP Discovery Information Caching Model

#### 2.3.2.1. Why do we need USNs, isn't the location good enough?

When a service announces itself it usually includes a location identifying where it may be found. However that location can and will change over time. For example, a user may decide to change the DNS name assigned to that device. Were we to depend on locations, not USNs, when the service's location was changed we would think we were seeing a brand new service. This would be very disruptive to the user's experience. Imagine, for example, that the user has set up a PC program that programs their VCR based on schedules pulled off the Internet. If the user decides to change the VCR's name from the factory default to something friendly then a location based system would loose track of the VCR it is supposed to be programming because the name has changed. By using unique Ids instead we are able to track the VCR regardless of the name change. So the user can

change the VCR's name at will and the VCR programming application will still be able to program the correct VCR.

#### 2.3.2.2. Why are USNs URIs and why are they required to be unique across the entire URI namespace for all time?

In general making a name universally unique turns out to usually be a very good idea. Mechanisms such as UUIDs allow universally unique names to be cheaply created in a decentralized manner. In this case making USNs globally unique is very useful because services may be constantly moved around, if they are to be successfully tracked they need an identifier that isn't going to change and isn't going to get confused with any other service.

URIs were chosen because they have become the de facto managed namespace for use on the Internet. Anytime someone wants to name something it is easy to just use a URI.

## 3. Terminology

SSDP Client - A HTTP client that makes use of a service.

SSDP Service - A HTTP resource that provides a service used by SSDP clients.

Service Type - A URI that identifies the type or function of a particular service.

Unique Service Name (USN) - A URI that is guaranteed to be unique across the entire URI namespace for all time. It is used to uniquely identify a particular service in order to allow services with identical service type URIs to to be differentiated.

In addition, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 4. SSDP Discovery Requests

### 4.1. Problem Statement

A mechanism is needed for SSDP clients to find desired SSDP services.

### 4.2. Proposed Solution

The SEARCH method, introduced by [DASL], is extended using the [MAN] mechanism to provide for SSDP discovery.

The SSDP SEARCH extension is identified by the URI ssdp:discover.

For brevity's sake a HTTP SEARCH method enhanced with the ssdp:discover functionality will be referred to as a ssdp:discover request.

ssdp:discover requests MUST contain a ST header. ssdp:discover requests MAY contain a body but the body MAY be ignored if not understood by the HTTP service.

The ST header contains a single URI. SSDP clients may use the ST header to specify the service type they want to discover.

This specification only specifies the use of ssdp:discover requests over HTTP Multicast UDP although it is expected that future specifications will expand the definition to handle ssdp:discover requests sent over HTTP TCP.

ssdp:discover requests sent to the SSDP multicast channel/port MUST have a request-URI of "*". Note that future specifications may allow for other request-URIs to be used so implementations based on this specification MUST be ready to ignore ssdp:discover requests on the SSDP multicast channel/port with a request-URI other than "*".

Only SSDP services that have a service type that matches the value in the ST header MAY respond to a ssdp:discover request on the SSDP multicast channel/port.

Responses to ssdp:discover requests sent over the SSDP multicast channel/port are to be sent to the IP address/port the ssdp:discover request came from.

A response to a ssdp:discover request SHOULD include the service's location expressed through the Location and/or AL header. A successful response to a ssdp:discover request MUST also include the ST and USN headers.

Response to ssdp:discover requests SHOULD contain a cache-control: max-age or Expires header. If both are present then they are to be processed in the order specified by HTTP/1.1, that is, the cache-control header takes precedence of the Expires header. If neither the cache-control nor the Expires header is provided on the response to a ssdp:discover request then the information contained in that response MUST NOT be cached by SSDP clients.

#### 4.2.1.1. Example

```
M-SEARCH * HTTP/1.1
S: uuid:ijklmnop-7dec-11d0-a765-00a0c91e6bf6
Host: 239.255.255.250:reservedSSDPport
Man: "ssdp:discover"
ST: ge:fridge
MX: 3
```

```
HTTP/1.1 200 OK
S: uuid:ijklmnop-7dec-11d0-a765-00a0c91e6bf6
Ext:
Cache-Control: no-cache="Ext", max-age = 5000
ST: ge:fridge
USN: uuid:abcdefgh-7dec-11d0-a765-00a0c91e6bf6
AL: <blender:ixl><http://foo/bar>
```

### 4.3. Design Rationale

#### 4.3.1. Why is the ST header so limited? Why doesn't it support at least and/or/not? Why not name/value pair searching?

Deciding the "appropriate" level of search capability is a hopeless task. So we decided to pare things back to the absolute minimum, a single opaque token, and see what happens. The result so far has been a very nice, simple, easy to implement, easy to use discovery system. There are lots of great features it doesn't provide but most of them, such as advanced queries and scoping, require a search engine and a directory. This level of capability is beyond many simple devices, exactly the sort of folks we are targeting with SSDP. Besides, search functionality seems to be an all or nothing type of situation. Either you need a brain dead simple search mechanism or you need a full fledged near SQL class search system. Instead of making SSDP the worst of both worlds we decided to just focus on the dirt simple search problem and leave the more advanced stuff to the directory folk.

#### 4.3.2. If we are using the SEARCH method why aren't you using the DASL search syntax?

We couldn't come up with a good reason to force our toaster ovens to learn XML. The features the full-fledged DASL search syntax provides are truly awesome and thus way beyond our simple scenarios. We fully expect that DASL will be the preferred solution for advanced search scenarios, but that isn't what this draft is about.

#### 4.3.3. Why can we only specify one search type in the ST header of a ssdp:discover request?

We wanted to start as simple as possible and be forced, kicking and screaming, into adding additional complexity. The simplest solution was to only allow a single value in the ST header. We were also concerned that if we allowed multiple values into the ST headers somebody would try to throw in and/or/not functionality. Given the minimal byte savings of allowing multiple values into the ST header it seems better to just leave the protocol simpler.

#### 4.3.4. Why do we only provide support for multicast UDP, not TCP, ssdp:discover requests?

We only define what we need to make the discovery protocol work and we don't need TCP to make the discovery protocol work. Besides to make TCP discovery really work you need to be able to handle compound responses which means you need a compound response format which is probably XML and that is more than we wanted to handle. Eventually we expect that you will be able to go up to the SSDP port on a server using a HTTP TCP request and discover what service, if any, lives there. But that will be described in a future specification.

#### 4.3.5. Why do we require that responses without caching information not be cached at all?

Because that was a lot easier thing to do then trying to explain the various heuristics one could use to deal with services who don't provide caching information.

## 5. SSDP Presence Announcements

### 5.1. Problem Statement

A mechanism is needed for SSDP services to be able to let interested SSDP clients know of their presence.

A mechanism is needed to allow SSDP services to update expiration information in cache entries regarding them.

A mechanism is needed to allow SSDP services to notify interested SSDP clients when their location changes.

A mechanism is needed to allow SSDP services to inform interested SSDP clients that they are going to de-activate themselves.

### 5.2. Proposed Solution

#### 5.2.1. ssdp:alive

SSDP services may declare their presence on the network by sending a [GENA] NOTIFY method using the NTS value ssdp:alive to the SSDP multicast channel/port.

For brevity's sake HTTP NOTIFY methods with the NTS value ssdp:alive will be referred to as ssdp:alive requests.

When a ssdp:alive request is received whose USN matches the USN of an entry already in the SSDP client's cache then all information regarding that USN is to be replaced with the information on the ssdp:alive request. Hence ssdp:alive requests can be used to update location information and prevent cache entries from expiring.

The value of NT on a ssdp:alive request MUST be set to the service's service type. ssdp:alive requests MUST contain a USN header set to the SSDP service's USN.

ssdp:alive requests SHOULD contain a Location and/or AL header. If there is no DNS support available on the local network then at least one location SHOULD be provided using an IP address of the SSDP service.

ssdp:alive requests SHOULD contain a cache-control: max-age or Expires header. If both are present then they are to be processed in the order specified by HTTP/1.1, that is, the cache-control header takes precedence of the Expires header. If neither the cache-control nor the Expires header is provided the information in the ssdp:alive request MUST NOT be cached by SSDP clients.

There is no response to a ssdp:alive sent to the SSDP multicast channel/port.

#### 5.2.1.1. Example

```
NOTIFY * HTTP/1.1
Host: 239.255.255.250:reservedSSDPport
NT: blenderassociation:blender
NTS: ssdp:alive
USN: someunique:idscheme3
AL: <blender:ixl><http://foo/bar>
Cache-Control: max-age = 7393
```

### 5.2.2. ssdp:byebye

SSDP services may declare their intention to cease operating by sending a [GENA] NOTIFY method using the NTS value ssdp:byebye to the SSDP multicast channel/port.

For brevity's sake HTTP NOTIFY methods with the NTS value ssdp:byebye will be referred to as ssdp:byebye requests.

The value of NT on a ssdp:byebye request MUST be set to the service's service type. ssdp:byebye requests MUST contain a USN header set to the SSDP service's USN.

There is no response to a ssdp:byebye sent to the SSDP multicast channel/port.

When a ssdp:byebye request is received all cached information regarding that USN SHOULD be removed.

#### 5.2.2.1. Example

```
NOTIFY * HTTP/1.1
Host: 239.255.255.250:reservedSSDPport
```

```
NT: someunique:idscheme3
NTS: ssdp:byebye
USN: someunique:idscheme3
```

## 5.3. Design Rationale

### 5.3.1. Why are we using GENA NOTIFY requests?

We needed to use some notification format and GENA seemed as good as any. Moving forward, GENA gives us a framework to do notification subscriptions which will be necessary if SSDP services are to be able to provide status updates across the wilds of the Internet without depending on the largely non-existent Internet multicast infrastructure.

### 5.3.2. Why is there no response to the ssdp:alive/ssdp:byebye requests sent to the SSDP multicast channel/port?

What response would be sent? There isn't much of a point of having the SSDP clients send response saying "we received your notification" since there may be a lot of them.

### 5.3.3. Could NTS values other than ssdp:alive/ssdp:byebye be sent to the SSDP multicast channel/port?

Yes.

### 5.3.4. Why do we include the NT header on ssdp:byebye requests?

Technically it isn't necessary since the only useful information is the USN. But we want to stick with the GENA format that requires a NT header. In truth the requirement of including the NT header is a consequence of the next issue.

### 5.3.5. Shouldn't the NT and NTS values be switched?

Yes, they should. Commands such as ssdp:alive and ssdp:byebye should be NT values and the service type, where necessary, should be the NTS. The current mix-up is a consequence of a previous design where the NT header was used in a manner much like we use the USN today. This really needs to change.

# 6. SSDP Auto-Shut-Off Algorithm

## 6.1. Problem Statement

A mechanism is needed to ensure that SSDP does not cause such a high level of traffic that it overwhelms the network it is running on.

## 6.2. Proposed Solution

[Ed. Note: We have a proposed solution but it is still a bit rough, so we will be publishing to the SSDP mailing list for further discussion before including it in the draft.]

## 6.3. Design Rationale

### 6.3.1. Why do we need an auto-shut-off algorithm?

The general algorithm for figuring out how much bandwidth SSDP uses over a fixed period of time based on the number of ssdp:discover requests is :

DR = Total number of SSDP clients making ssdp:discover requests over the time period in question.
RS = Total number of services that will respond to the ssdp:discover requests over the time period in question.
AM = Average size of the ssdp:discover requests/responses.
TP = Time period in question.

((DR*3 + DR*9*RS)*AM)/TP

The 3 is the number of times the ssdp:discover request will be repeated.
The 9 is the number of times the unicast responses to the ssdp:discover requests will be sent out assuming the worst case in which all 3 original requests are received.

So let's look at a real world worst-case scenario. Some companies, in order to enable multicast based services such as voice or video streaming to be easily configured set their local administrative multicast scope to encompass their entire company. This means one gets networks with 100,000 machines in a single administrative multicast scope. Now imagine that there is a power outage and all the machines are coming back up at the same time. Further imagine that they all want to refresh their printer location caches so they all send out ssdp:discover requests. Let us finally imagine that there are roughly 5000 printers in this network. To simplify the math we will assume that the ssdp:discover requests are evenly distributed over the 30 seconds.

DR = 100,000 requesting clients
RS = 5000 services
AM = 512 bytes
TP = 30 seconds

((100000*3+100000*9*5000)*512)/30 = 76805120000 bytes/s = 585976.5625 Megabits per second

This is what one would call an awful number.

In a more reasonably sized network SSDP is able to handle this worst case scenario much better. For example, let's look at a network with 1000 clients and 50 printers.

DR = 1000 requesting clients
RS = 50 services
AM = 512 bytes
TP = 30 seconds

((1000*3+1000*9*50)*512)/30 = 7731200 bytes/s = 59 Mbps

Now this looks like an awful amount but remember that that this is the total data rate needed for 30 seconds. This means that the total amount of information SSDP needs to send out to survive a reboot is 59*30 = 1770 Mb. Therefore a 10 Mbps network, assuming an effective data rate 5 Mbps under constant load that means it will take 1770/5 = 354 seconds = 6 minutes for the network to settle down.

That isn't bad considering that this is an absolute worst case in a network with 1000 clients and 50 services all of whom want to talk to each other at the exact same instant.

In either case, there are obvious worst-case scenarios and we need to avoid network storms, therefore we need a way for SSDP to de-activate before it causes a network storms.

### 6.3.2. **Why not just require everyone to support directories and thus** get around the scaling issue?

Many manufacturers stick every protocol they can think of in their clients and services. So if your network administrator happened to buy some clients and servers that supported SSDP but didn't know they supported SSDP then you can imagine the problems. Therefore even if we required directory support there are still many cases where SSDP clients and services may inadvertently end up in a network without anyone knowing it and cause problems.

## 7. ssdp:all

### 7.1. Problem Statement

A mechanism is needed to enable a client to enumerate all the services available on a particular SSDP multicast channel/port.

### 7.2. Proposed Solution

All SSDP services MUST respond to SEARCH requests over the SSDP multicast channel/port with the ST value of ssdp:all by responding as if the ST value had been their service type.

For brevity's sake a SEARCH request with a ST of ssdp:all will be referred to as a ssdp:all request.

### 7.3. Design Rationale

#### 7.3.1. Why would anyone want to enumerate all services?

This feature is mostly for network analysis tools. It also will prove very useful in the feature when directories become SSDP aware. They will be able to discover all services, record information about them and make that information available outside the local administrative multicast scope.

## 8. SSDP Reserved Multicast Channel

### 8.1. Problem Statement

SSDP needs a local administrative multicast channel that will be guaranteed to only be used by SSDP compliant clients and services.

### 8.2. Proposed Solution

IANA has reserved the relative multicast address "5" for the exclusive use of SSDP. In the local administrative scope used by this version of SSDP the relative address translates to 239.255.255.250.

An application has been put in for a SSDP reserved port but IANA has not yet responded.

### 8.3. Design Rationale

#### 8.3.1. Why didn't SSDP just get a static local administrative scope address rather than a relative address?

We got a relative address because we expect that SSDP may be used to discover basic system services such as directories. In that case if you can't find a directory in your local scope you may want to try a wider multicast scope. This is exactly the sort of functionality enabled by MALLOC (http://www.ietf.org/html.charters/malloc-charter.html). MALLOC allows one to enumerate all the multicast scopes that are supported on the network. The SSDP client can then try progressively larger scopes to find the service they are seeing. However this progressively wider discovery only works if SSDP uses a relative address.

#### 8.3.2. Why does SSDP need to use a port other than 80?

There is a bug in the Berkley Sockets design that was inherited by WinSock as well. The bug is as follows: One can not grab a particular port on a particular multicast address without owning the same port on the local unicast address.

The result is that if we used port 80 on the SSDP multicast scope then we would require that the SSDP software also grab port 80 for the local machine. This would mean that SSDP could only be implemented on machines which either didn't have HTTP servers or whose HTTP servers had been enhanced to support SSDP.

We felt this was a unnecessary restriction. Therefore we are choosing to use a port other than 80 on the SSDP multicast channel.

## 9. HTTP Headers

### 9.1. USN Header

USN = "USN" ":" AbsoluteURI; defined in section 3.2.1 of [RFC2616]

### 9.2. ST Header

ST = "ST" ":" AbsoluteURI

## 10. Security Considerations

TBD.

## 11. IANA Considerations

To ensure correct interoperation based on this specification, IANA must reserve the URI namespace starting with "ssdp:" for use by this specification, its revisions, and related SSDP specifications.

IANA has reserved the relative multicast address "5" for exclusive use by SSDP. An application has been made for a registered port.

## 12. Appendix - Constants

MAX_UNIQUE - 50 - Maximum number of unique IP address/port pairs that may be sent over UDP before tripping the auto-shut-off algorithm.

MAX_COUNT - 30 seconds - When the "go quiet" process is begun a message is sent out that is delayed a random interval between 0 to MAX_COUNT seconds.

## 13. Acknowledgements

This document is the result of enormous effort by a large number of people including but not limited to:
Alan Boshier, Babak Jahromi, Brandon Watson, Craig White, Dave Thaler, Holly Knight, Michel Guittet, Mike Zintel, Munil Shah, Paul Moore, Peter Ford, Pradeep Bahl, and Todd Fisher.

## 14. References

[HTTPUDP] Y. Y. Goland. Multicast and Unicast UDP HTTP Requests. Internet Draft - a work in progress, draft-goland-http-udp-00.txt.

[GENA] J. Cohen, S. Aggarwal, Y. Y. Goland. General Event Notification Architecture Base: Client to Arbiter. Internet Draft - a work in progress, draft-cohen-gena-client-00.txt.

[MAN] H. Nielsen, P. Leach, S. Lawrence. Mandatory Extensions in HTTP. Internet Draft - a work in progress, draft-frystyk-http-extensions-03.txt.

[RFC2119] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119, March 1997.

[RFC2365] D. Meyer. Administratively Scoped IP Multicast. RFC 2365, July 1998.

[RFC2396] T. Berners-Lee, R. Fielding and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396, August 1998.

[RFC2518] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring û WEBDAV. RFC 2518, February 1999.

[RFC2616] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1. RFC 2616, November 1998.

[DASL] S. Reddy, D. Lowry, S. Reddy, R. Henderson, J. Davis, A. Babich. DAV Searching & Locating. a work in progress - draft-ietf-dasl-protocol-00.txt.

## 15. Author's Addresses

Yaron Y. Goland, Ting Cai, Paul Leach, Ye Gu
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Email: {yarong, tingcai, paulle, yegu}@microsoft.com

Shivaun Albright
Hewlett-Packard Company
Roseville, CA

Email: SHIVAUN_ALBRIGHT@HP-Roseville-om2.om.hp.com

This document will expire in April 2000.