Network Working Group
Request for Comments: 3528
Category: Experimental

W. Zhao
H. Schulzrinne
Columbia University
E. Guttman
Sun Microsystems

April 2003

Mesh-enhanced Service Location Protocol (mSLP)

Status of this Memo

Copyright Notice

Abstract

This document describes the Mesh-enhanced Service Location Protocol (mSLP). mSLP enhances the Service Location Protocol (SLP) with a scope-based fully-meshed peering Directory Agent (DA) architecture. Peer DAs exchange new service registrations in shared scopes via anti-entropy and direct forwarding. mSLP improves the reliability and consistency of SLP DA services, and simplifies Service Agent (SA) registrations in systems with multiple DAs. mSLP is backward compatible with SLPv2 and can be deployed incrementally.

Table of Contents

1. Introduction

In the Service Location Protocol (SLPv2 [RFC2608]), Directory Agents (DAs) accept service registrations from Service Agents (SAs) and answer queries from User Agents (UAs); they enhance the performance and scalability of SLPv2. The use of scopes in SLPv2 further improves its scalability. In general, a DA can serve multiple scopes, and a scope can be served by multiple DAs. When multiple DAs are present for a scope, how should they interact with each other? This document describes the Mesh-enhanced Service Location Protocol (mSLP), addressing this open issue in SLPv2.

mSLP defines a scope-based fully-meshed peering DA architecture: for each scope, all DAs serving the scope form a fully-meshed peer relationship (similar to IBGP [RFC1771]). Peer DAs exchange new service registrations in shared scopes via anti-entropy [EPID-ALGO,UPDA-PROP] and direct forwarding. mSLP improves the reliability and consistency of SLP DA services, and simplifies SA registrations in systems with multiple DAs.

1.1. Notation Conventions

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 BCP 14, RFC 2119 [RFC2119].

1.2. Terminology

Peer DAs (or Peers)
: DAs that share one or multiple scopes are peers.

Peering Connection
: A persistent connection (e.g., TCP) that provides reliable and ordered transfers between two peers. The closing of a peering connection terminates the peer relationship.

Mesh-enhanced DA (MDA)
: An MDA carries the "mesh-enhanced" attribute keyword in its DA Advertisement (DAAdvert) message, maintains peering connections to all peers, and properly interacts with peers.

Mesh-enhanced SA (MSA)
: An MSA uses the Mesh Forwarding extension (Section 4.3) when it registers with MDAs.

Registration Update
: A registration update refers to a Service Registration (SrvReg) or Service Deregistration (SrvDeReg) message.

Registration State
: A registration state refers to an entry in the registration database.

Accept DA
: When a DA accepts a registration update from an SA, the DA is the accept DA for the update.

Accept Timestamp
: The arrival timestamp of a registration update at its accept DA is the accept timestamp of the update. All accept timestamps assigned by the same DA MUST be monotonically increasing.

Version Timestamp
: When an MSA sends a registration update to an MDA, the MSA assigns a version timestamp to the update. All version timestamps assigned by the same MSA MUST be monotonically increasing.

1.3. Compatibility

mSLP is designed as a lightweight enhancement to SLPv2. It is backward compatible with SLPv2. mSLP defines two enhanced entities: MDAs and MSAs. They can be deployed incrementally. An enhanced entity supports extended operations without affecting its original functionality as defined in RFC 2608 [RFC2608]. For simplicity and

compatibility, an enhanced entity works as a non-enhanced entity to interact with non-enhanced entities. Table 1 summarizes all interactions involving an MDA or MSA.

Interaction	Equivalent To	Defined In
MDA <--> MDA		mSLP
MDA <--> MSA		mSLP
MDA <--> DA	DA <--> DA	RFC 2608
MDA <--> SA	DA <--> SA	RFC 2608
MDA <--> UA	DA <--> UA	RFC 2608
MSA <--> DA	SA <--> DA	RFC 2608
MSA <--> UA	SA <--> UA	RFC 2608

Table 1. Interactions involving an MDA or MSA

2. Scope-based Fully-meshed Peering DA Architecture

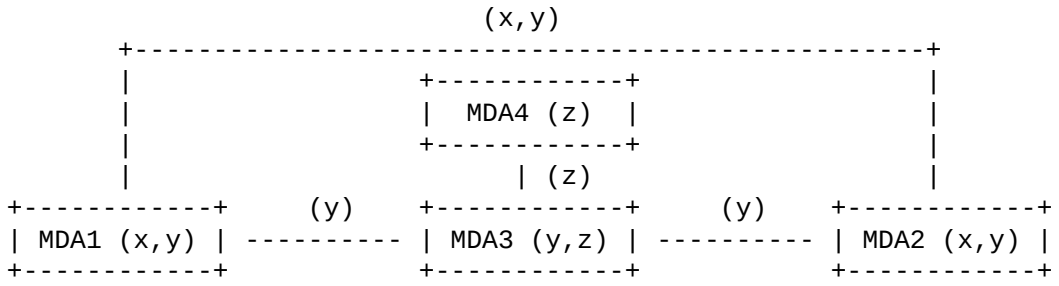


Figure 1. A scope-based fully-meshed peering DA architecture

mSLP employs a scope-based fully-meshed peering DA architecture. For each scope, all MDAs that serve the scope form a fully-meshed peer relationship. Figure 1 shows an example for four MDAs and three scopes (x, y, and z). Note that a single peering connection is needed between two peers for exchanging all service registrations in their shared scopes.

This architecture enhances SLP DA services. First, it improves the consistency among peer DAs by automatically reconciling inconsistent states among them. Second, it enables newly booted and rebooted MDAs to catch up on all new registrations at once from their peers, purely through DA interaction, without involving SAs.

This architecture also simplifies SA registrations. In SLPv2, an SA needs to discover and register with all existing DAs in its scopes, and re-register when new DAs are discovered or old DAs are found to have rebooted. In mSLP, for all MDAs, an MSA only needs to discover and register with a sufficient number of them, such that the union of

their scopes covers its scopes; the registrations will then be propagated automatically to other MDAs in the registration scopes. For example, in Figure 2, MSA1 only needs to discover and register with MDA2, or with both MDA1 and MDA3.

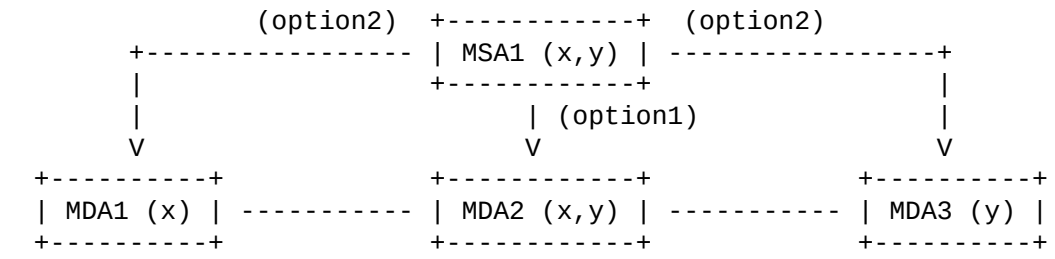


Figure 2. Options for registering with MDAs

Furthermore, this architecture provides scaling advantages. Consider a scope that has N SAs and M DAs, and assume N > M >= 2. Although mSLP and SLPv2 need the same number of SLP messages to distribute registrations from N SAs to M DAs, mSLP can reduce the number of agents needed for taking care of registration distribution, and reduce the number of TCP connections needed if each SA uses TCP for its registrations. More specifically, the agents that need to take care of registration distribution are all N SAs in SLPv2, but only M DAs in mSLP. Also, the number of needed TCP connections is N*M in SLPv2 as each SA has to connect with each DA and register, but only N+M*(M-1)/2 in mSLP as each SA only needs to connect to one contacting DA of a full mesh of M node and register, then registrations are propagated through the DA mesh. For N=100 and M=10, SLPv2 needs 1000 TCP connections, but mSLP only needs 145 such connections.

Note that as mSLP employs full-mesh topology, which is mainly for simplicity and reliability, it cannot scale to a large number of MDAs in a single mesh. In general, mSLP can be applied if the number of MDAs in a mesh is on the order of tens or below. One way to avoid having a large number of MDAs in a mesh is to split the scope into several finer scopes. For example, if we have N MDAs for scope "x", and N is too large, then we can split "x" into two finer scopes: "x-1" and "x-2", with N1 MDAs for "x-1" only, N2 MDAs for "x-2" only, N3 MDAs for both "x-1" and "x-2", and N1+N2+N3=N. Thus, instead of having a large full mesh of size N, we now have two smaller full meshes of size N1+N3 and N2+N3, respectively. Accordingly, a service registration that previously targets for scope "x", now needs to be registered under both "x-1" and "x-2".

3. Peer Relationship Management

3.1. Learning about New Peers

An MDA can learn about new peers via static configuration, DHCP [RFC2610], and DAAdvert multicast and unicast. In any case, an MDA MUST get a peer's DAAdvert before establishing a peer relationship to the peer.

3.2. Establishing a Peering Connection

After getting a new peer's DAAdvert, an MDA establishes a peering connection (if such a connection does not exist yet) to the peer, and sends its DAAdvert via the connection (Figure 3). An MDA can identify a peering connection initiated by a peer by receiving the peer's DAAdvert from the connection. Normally, a single peering connection is set up between two peers, but there is a small possibility that a pair of peering connections might be created between two peers if they try to initiate a connection to each other at almost the same time. Thus, when an MDA identifies a new peering connection initiated by a peer, it SHOULD check whether it has initiated another peering connection to the peer. If this is the case, and it has a lower-numbered IP address than the peer, then the MDA SHOULD terminate the connection it has initiated.

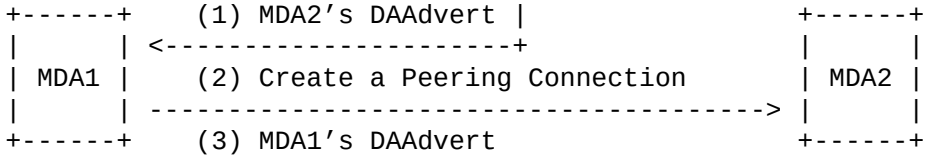


Figure 3. Establishing a peering connection

3.3. Exchanging Information about Existing Peers

After establishing a peering connection, two peers (say, MDA1 and MDA2) exchange information about their existing peers by forwarding peers' DAAdverts via the peering connection (Figure 4). MDA1 will forward the DAAdvert of a peer (say, MDA3) to MDA2 if:

(1) MDA3 shares scopes with MDA2, and
(2) MDA3 is an active peer of MDA1 (i.e., there is a peering connection between MDA3 and MDA1) or an accept DA for registrations currently maintained by MDA1 (i.e., MDA1 has registrations originally accepted by MDA3).

MDA2 operates similarly. Note that all DAAdverts can be sent as one TCP stream for efficiency. Exchanging information about existing peers enables an MDA to learn about new peers incrementally.

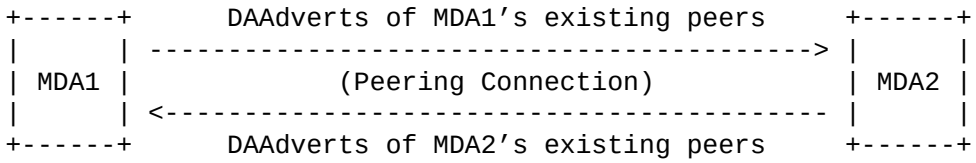


Figure 4. Exchanging information about existing peers

3.4. Maintaining a Peer Relationship

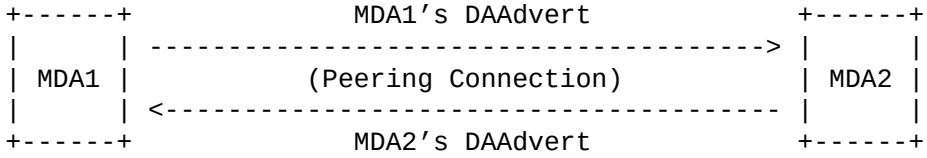


Figure 5. Maintaining a peer relationship

To detect failures (network partitions and peer crashes), mSLP uses a heart-beat mechanism. An MDA sends its DAAdvert to peers (Figure 5) every CONFIG_DA_KEEPALIVE seconds. The timeout value for this message is CONFIG_DA_TIMEOUT seconds (Section 6).

3.5. Tearing Down a Peer Relationship

An MDA SHOULD tear down a peer relationship when it finds that the peer has closed the peering connection, when it receives a DAAdvert multicast from the peer with a DA stateless boot timestamp set to 0 (meaning that the peer is going to shutdown), or when it has not received the peer's DAAdvert for more than CONFIG_DA_TIMEOUT seconds.

4. Registration Propagation Control

4.1. Accept ID and Propagation Order

When an MDA accepts a registration update from an MSA, the MDA assigns a unique accept ID to the update. An accept ID has two components: an accept DA URL and an accept timestamp. The accept timestamp is a 64-bit integer representing elapsed microseconds since 00:00 Coordinated Universal Time (UTC), January 1, 1900. Figure 6 shows the format for an accept ID entry. A registration state has the same accept ID as that of the latest update applied to it.

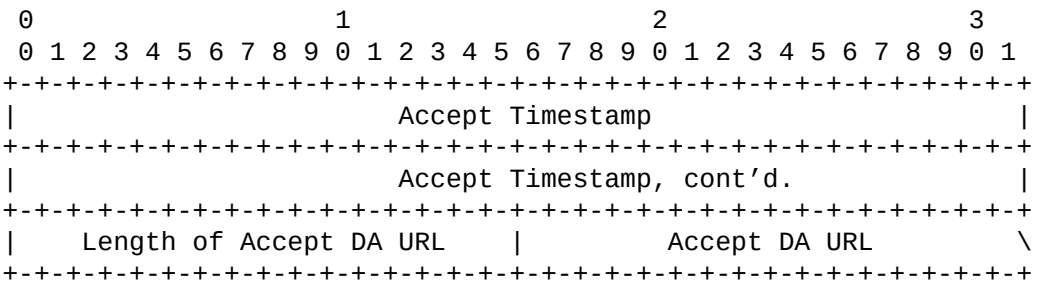


Figure 6. Accept ID entry

An MDA MUST propagate registrations in the increasing order of their accept IDs, i.e., registrations having the same accept DA MUST be propagated in the increasing order of their accept timestamps. Note that registrations having different accept DAs MAY be propagated in any order.

4.2. Version Timestamp and Registration Version Resolution

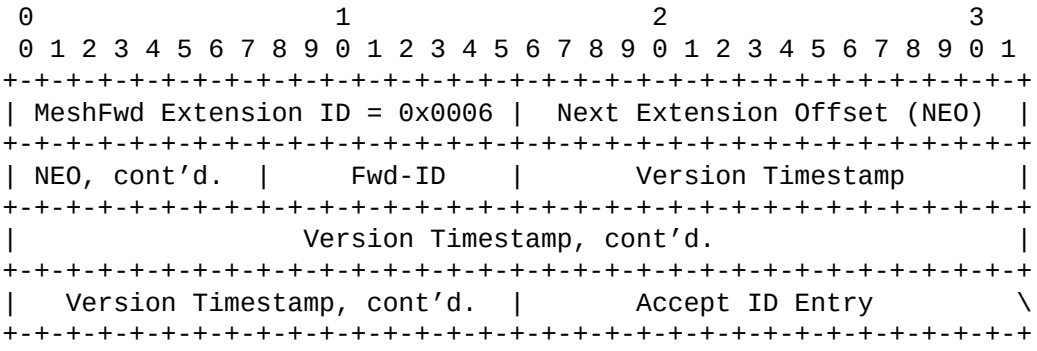
When registrations are propagated among MDAs, their arrival timestamps at MDAs cannot be used for version resolution. For example, assume that MSA1 sends a registration (R1) to MDA1 first, and a new version of the same registration (R2) to MDA2 later. When R1 and R2 are propagated, the arrival timestamp of R1 at MDA2 is later than that of R2, but R1 SHOULD NOT overwrite R2 at MDA2 as R2 is a newer version.

mSLP resolves registration versions using version timestamps. When an MSA sends a registration update to an MDA, the MSA assigns a version timestamp to the update. The version timestamp is a 64-bit integer representing elapsed microseconds since 00:00 UTC, January 1, 1900. mSLP assumes that each registration is updated only by one SA, thus an MDA does not need to compare version timestamps from different MSAs. An MDA installs a registration update if the update has a newer version timestamp (from an MSA), or the update does not have the Mesh Forwarding extension (from a non-MSA).

4.3. Mesh Forwarding Extension

The Mesh Forwarding (MeshFwd) extension carries a version timestamp and an accept ID entry. Figure 7 shows its format and two defined Forwarding IDs (Fwd-IDs).

The MeshFwd extension is used with a Srv(De)Reg message, but it can only be used with a fresh SrvReg, or a complete SrvDeReg.



Fwd-ID	Abbreviation
1	RqstFwd
2	Fwded

Figure 7. MeshFwd extension and its Fwd-IDs

An MSA uses the RqstFwd MeshFwd extension (Fwd-ID = RqstFwd, accept timestamp = 0) in a Srv(De)Reg to explicitly request an MDA (the accept DA) to forward the message.

An MDA uses the Fwded MeshFwd extension (Fwd-ID = Fwded, accept timestamp != 0) in each Srv(De)Reg sent from it to another MDA, either forwarding a Srv(De)Reg received from an MSA (if the message has the RqstFwd MeshFwd extension), or propagating a registration state in its database.

4.4. Summary Vector

An MDA uses a summary vector to represent its received Srv(De)Reg(s) that have a MeshFwd extension. This summary vector records the latest accept timestamp for each accept DA that appears in the MeshFwd extension. For example, consider n MDAs for a scope, if MDAi has a summary vector as ((MDA1, T1), (MDA2, T2), ..., (MDAn, Tn)), then MDAi has received all registrations originally accepted by MDAj up to timestamp Tj, where 1<=i,j<=n.

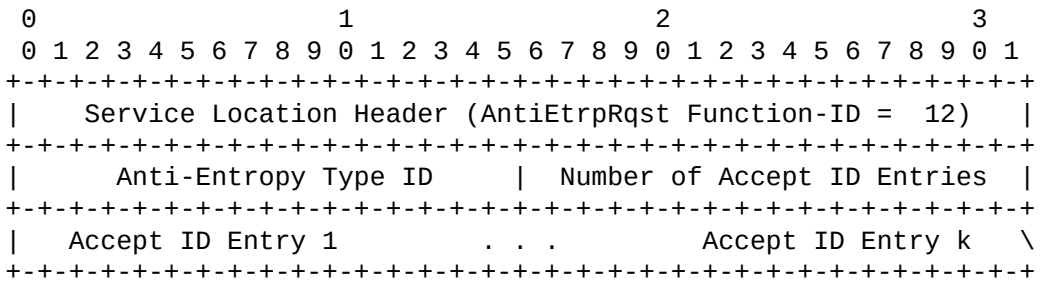
An MDA updates its summary vector when it receives a Srv(De)Reg that has a MeshFwd extension. The MDA adds a new accept ID to its summary vector if the Srv(De)Reg has a new accept DA; the MDA updates the accept timestamp of an existing accept ID in its summary vector if the Srv(De)Reg has an existing accept DA.

4.5. Service Deregistration

When an MDA receives a SrvDeReg that has a MeshFwd extension, it SHOULD retain the corresponding registration in the database, and mark it as deleted. This way, the registration will not appear in any query reply, and an earlier SrvReg will not mistakenly cause the registration to reappear in the database. A registration state will be purged from the database when it expires.

4.6. Anti-entropy Request Message

The Anti-entropy Request (AntiEtrpRqst) message carries an anti-entropy type ID and a list of accept ID entries. Figure 8 shows its format and two defined anti-entropy type IDs.



Anti-Entropy Type	Type ID
selective	1
complete	2

Figure 8. AntiEtrpRqst message and anti-entropy types

The AntiEtrpRqst message is used by an MDA to request new registration states from a peer. The anti-entropy type is either selective or complete. If the anti-entropy type is selective, only registration states that have an accept ID greater than any specified accept ID in the message are requested. If the anti-entropy type is complete, all registration states that have an accept ID greater than any specified accept ID in the message or have an accept DA not specified in the message are requested.

For example, consider three MDAs (MDA1, MDA2, and MDA3) for a scope. MDA2 has registration states originally accepted by MDA1, MDA2, and MDA3. If MDA1 sends a selective AntiEtrpRqst to MDA2 using an accept ID list as ((MDA2, T2)), then MDA1 only requests registration states that are originally accepted by MDA2, and have an accept timestamp greater than T2. If MDA1 sends a complete AntiEtrpRqst to MDA2 using an accept ID list as ((MDA2, T2)), then MDA1 requests all

registration states originally accepted by MDA1 and MDA3, plus those originally accepted by MDA2 and having an accept timestamp greater than T2.

4.7. Anti-entropy

Anti-entropy is used for exchanging initial registration states when two peers recognize each other for the first time, and for updating new registration states after failures.

When an MDA receives an AntiEtrpRqst from a peer, it sends the requested new registration states in the increasing order of their accept IDs. At last a Service Acknowledgment (SrvAck) message is sent to indicate that the processing of a corresponding AntiEtrpRqst has been completed (Figure 9). A new registration state is sent as a fresh SrvReg with its remaining lifetime. A newly deregistered state is propagated as a SrvDeReg. Note that multiple Srv(De)Reg(s) can be sent as one TCP stream for efficiency.

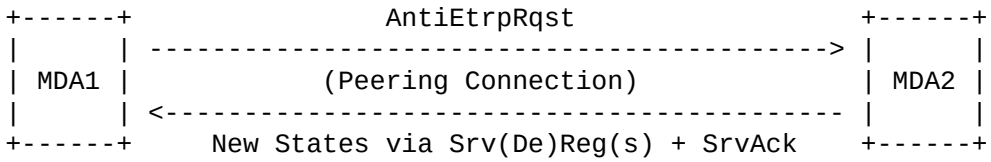


Figure 9. Anti-entropy via AntiEtrpRqst, Srv(De)Reg(s) and SrvAck

4.8. Direct Forwarding

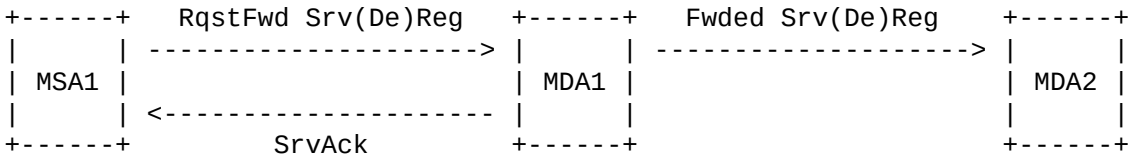


Figure 10. Direct forwarding of a Srv(De)Reg

After sending all new registration states accepted by itself to a peer (via anti-entropy), an MDA directly forwards newly received registration updates from MSAs to the peer until a failure occurs.

In Figure 10, when a Srv(De)Reg is directly forwarded from MDA1 to MDA2, its Fwd-ID is set to Fwded, and its accept timestamp is set to its arrival timestamp at MDA1. Note that a direct forwarding is performed asynchronously: MDA1 can send a SrvAck to MSA1 before it forwards the Srv(De)Reg to MDA2. Also note that the direct forwarding of a Srv(De)Reg goes only one-hop from its accept DA (say, MDA1) to all MDA1's peers that are in the registration scopes.

4.9. SrvAck Message

According to [RFC2608], a DA MUST reply with a SrvAck to a Srv(De)Reg when the message is received from an SA. However, an MDA SHOULD NOT reply with a SrvAck to a Srv(De)Reg if the message is received from a peer. This is for efficiency because peers exchange Srv(De)Reg messages via reliable peering connections. Note that an MDA MUST reply with a SrvAck to an AntiEtrpRqst.

4.10. Control Information

For each registration entry, an MDA maintains the following control information: an accept ID (for registration propagation), a version timestamp (for registration version resolution - rejecting previous updates), and a deletion flag (deregistered or not).

For all registration entries, an MDA maintains a summary vector to reflect its received registrations so far.

5. Summary

mSLP extends SLPv2 with three new definitions: a new attribute - "mesh-enhanced" for DAAdvert, a new message extension - MeshFwd, and a new message type - AntiEtrpRqst.

A UA MAY prefer an MDA to a non-MDA since an MDA is more likely to reliably contain the complete set of current service registrations for the UA’s scopes.

A non-MSA needs to discover and register with all DAs in its scopes. It does not use the MeshFwd extension.

A non-MDA accepts Srv(De)Reg(s) from SAs. It does not forward them.

For all MDAs, an MSA only needs to discover and register with sufficient number of them, such that they cover its scopes. It uses the MeshFwd extension when it registers with MDAs.

An MDA carries the "mesh-enhanced" attribute keyword in its DAAdvert. It maintains a peer relationship to each peer. It accepts registrations from SAs and peers, propagates registrations via anti-entropy and direct forwarding to peers.

6. Protocol Timing Defaults

Interval Name	Default Value	Defined in
CONFIG_DA_KEEPALIVE	200 seconds	Section 3.4
CONFIG_DA_TIMEOUT	300 seconds	Section 3.4

7. IANA Considerations

The Mesh Forwarding (MeshFwd) extension ID, 0x0006, described in Section 4.3, has been assigned by IANA out of the SLP extension space (RFC 2608, Section 9.1) reserved for "optional to implement" extensions (i.e., the 0x0000-0x3FFF range).

The function-ID of the Anti-entropy Request (AntiEtrpRqst) message type, 12, described in Section 4.6, has been assigned by IANA (RFC 2608, Section 15).

8. Security Considerations

mSLP uses standard SLPv2 authentication. First, an MDA SHOULD authenticate other MDAs before setting up a peer relationship with them so as to prevent any malicious MDA from joining the DA mesh. Second, as a successful attack at an MDA may affect all MDAs in the DA mesh, an MDA SHOULD authenticate MSAs before accepting and forwarding their Srv(De)Reg messages to prevent illegitimate modification or elimination of service registrations. Third, as an MSA depends on the MDA with which it registers to forward its Srv(De)Reg messages, it SHOULD authenticate the MDA to avoid using a malicious MDA.

9. Acknowledgments

Thomas Narten, James Kempf, Mike Day, Mikael Pahmp, Ira McDonald, Qiaobing Xie and Xingang Guo provided valuable comments for this document.

10. References

10.1. Normative References

[RFC2608] Guttman, E., Perkins, C., Veizades, J. and M. Day, "Service Location Protocol, Version 2", RFC 2608, June 1999.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10.2. Informative References

[RFC1771] Rekhter, R. and T. Li, "A Border Gateway Protocol 4 (BGP-4)", RFC 1771, March 1995.

[RFC2610] Perkins, C. and E. Guttman, "DHCP Options for Service Location Protocol", RFC 2610, June, 1999.

[EPID-ALGO] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart and D. Terry, "Epidemic algorithms for replicated database maintenance", the sixth ACM symposium on principles of distributed computing, Vancouver, Canada, 1987.

[UPDA-PROP] K. Petersen, M. Spreizer, D. Terry, M. Theimer and A. Demers, "Flexible update propagation for weakly consistent replication", the sixteenth ACM symposium on operating systems principles, Saint Malo, France, 1997.

11. Authors' Addresses

Weibin Zhao
Department of Computer Science
Columbia University
1214 Amsterdam Avenue, MC 0401
New York, NY 10027-7003

EMail: zwb@cs.columbia.edu

Henning Schulzrinne
Department of Computer Science
Columbia University
1214 Amsterdam Avenue, MC 0401
New York, NY 10027-7003

EMail: hgs@cs.columbia.edu

Erik Guttman
Sun Microsystems
Eichhoelzelstr. 7
74915 Waibstadt
Germany

EMail: Erik.Guttman@sun.com

12. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.