Network Working Group
Request for Comments: 4533
Category: Experimental

K. Zeilenga
OpenLDAP Foundation
J.H. Choi
IBM Corporation

June 2006

The Lightweight Directory Access Protocol (LDAP) Content Synchronization Operation

Status of This Memo

This memo defines an Experimental Protocol for the Internet community. It does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

Copyright Notice

IESG Note

The IESG notes that this work was originally discussed in the LDUP working group. The group came to consensus on a different approach, documented in RFC 3928; that document is on the standards track and should be reviewed by those considering implementation of this proposal.

Abstract

This specification describes the Lightweight Directory Access Protocol (LDAP) Content Synchronization Operation. The operation allows a client to maintain a copy of a fragment of the Directory Information Tree (DIT). It supports both polling for changes and listening for changes. The operation is defined as an extension of the LDAP Search Operation.

Table of Contents

1. Introduction

The Lightweight Directory Access Protocol (LDAP) [RFC4510] provides a mechanism, the search operation [RFC4511], that allows a client to request directory content matching a complex set of assertions and to request that the server return this content, subject to access control and other restrictions, to the client. However, LDAP does not provide (despite the introduction of numerous extensions in this area) an effective and efficient mechanism for maintaining synchronized copies of directory content. This document introduces a new mechanism specifically designed to meet the content synchronization requirements of sophisticated directory applications.

This document defines the LDAP Content Synchronization Operation, or Sync Operation for short, which allows a client to maintain a synchronized copy of a fragment of a Directory Information Tree (DIT). The Sync Operation is defined as a set of controls and other protocol elements that extend the Search Operation.

1.1. Background

Over the years, a number of content synchronization approaches have been suggested for use in LDAP directory services. These approaches are inadequate for one or more of the following reasons:

- failure to ensure a reasonable level of convergence;
- failure to detect that convergence cannot be achieved (without reload);
- require pre-arranged synchronization agreements;
- require the server to maintain histories of past changes to DIT content and/or meta information;
- require the server to maintain synchronization state on a per-client basis; and/or
- are overly chatty.

The Sync Operation provides eventual convergence of synchronized content when possible and, when not, notification that a full reload is required.

The Sync Operation does not require pre-arranged synchronization agreements.

The Sync Operation does not require that servers maintain or use any history of past changes to the DIT or to meta information. However, servers may maintain and use histories (e.g., change logs, tombstones, DIT snapshots) to reduce the number of messages generated and to reduce their size. As it is not always feasible to maintain and use histories, the operation may be implemented using purely (current) state-based approaches. The Sync Operation allows use of either the state-based approach or the history-based approach on an operation-by-operation basis to balance the size of history and the amount of traffic. The Sync Operation also allows the combined use of the state-based and the history-based approaches.

The Sync Operation does not require that servers maintain synchronization state on a per-client basis. However, servers may maintain and use per-client state information to reduce the number of messages generated and the size of such messages.

A synchronization mechanism can be considered overly chatty when synchronization traffic is not reasonably bounded. The Sync Operation traffic is bounded by the size of updated (or new) entries and the number of unchanged entries in the content. The operation is designed to avoid full content exchanges, even when the history information available to the server is insufficient to determine the client's state. The operation is also designed to avoid transmission of out-of-content history information, as its size is not bounded by the content and it is not always feasible to transmit such history information due to security reasons.

This document includes a number of non-normative appendices providing additional information to server implementors.

1.2. Intended Usage

The Sync Operation is intended to be used in applications requiring eventually-convergent content synchronization. Upon completion of each synchronization stage of the operation, all information to construct a synchronized client copy of the content has been provided to the client or the client has been notified that a complete content reload is necessary. Except for transient inconsistencies due to concurrent operation (or other) processing at the server, the client copy is an accurate reflection of the content held by the server. Transient inconsistencies will be resolved by subsequent synchronization operations.

Possible uses include the following:

- White page service applications may use the Sync Operation to maintain a current copy of a DIT fragment, for example, a mail user agent that uses the sync operation to maintain a local copy of an enterprise address book.

- Meta-information engines may use the Sync Operation to maintain a copy of a DIT fragment.

- Caching proxy services may use the Sync Operation to maintain a coherent content cache.

- Lightweight master-slave replication between heterogeneous directory servers. For example, the Sync Operation can be used by a slave server to maintain a shadow copy of a DIT fragment. (Note: The International Telephone Union (ITU) has defined the X.500 Directory [X.500] Information Shadowing Protocol (DISP) [X.525], which may be used for master-slave replication between directory servers. Other experimental LDAP replication protocols also exist.)

This protocol is not intended to be used in applications requiring transactional data consistency.

As this protocol transfers all visible values of entries belonging to the content upon change instead of change deltas, this protocol is not appropriate for bandwidth-challenged applications or deployments.

1.3. Overview

This section provides an overview of basic ways the Sync Operation can be used to maintain a synchronized client copy of a DIT fragment.

- Polling for changes: refreshOnly mode

- Listening for changes: refreshAndPersist mode

1.3.1. Polling for Changes (refreshOnly)

To obtain its initial client copy, the client issues a Sync request: a search request with the Sync Request Control with mode set to refreshOnly. The server, much like it would with a normal search operation, returns (subject to access controls and other restrictions) the content matching the search criteria (baseObject, scope, filter, attributes). Additionally, with each entry returned, the server provides a Sync State Control indicating state add. This control contains the Universally Unique Identifier (UUID) [UUID] of

the entry [RFC4530]. Unlike the Distinguished Name (DN), which may change over time, an entry's UUID is stable. The initial content is followed by a SearchResultDone with a Sync Done Control. The Sync Done Control provides a syncCookie. The syncCookie represents session state.

To poll for updates to the client copy, the client reissues the Sync Operation with the syncCookie previously returned. The server, much as it would with a normal search operation, determines which content would be returned as if the operation were a normal search operation. However, using the syncCookie as an indicator of what content the client was sent previously, the server sends copies of entries that have changed with a Sync State Control indicating state add. For each changed entry, all (modified or unmodified) attributes belonging to the content are sent.

The server may perform either or both of the two distinct synchronization phases that are distinguished by how to synchronize entries deleted from the content: the present and the delete phases. When the server uses a single phase for the refresh stage, each phase is marked as ended by a SearchResultDone with a Sync Done Control. A present phase is identified by a FALSE refreshDeletes value in the Sync Done Control. A delete phase is identified by a TRUE refreshDeletes value. The present phase may be followed by a delete phase. The two phases are delimited by a refreshPresent Sync Info Message having a FALSE refreshDone value. In the case that both the phases are used, the present phase is used to bring the client copy up to the state at which the subsequent delete phase can begin.

In the present phase, the server sends an empty entry (i.e., no attributes) with a Sync State Control indicating state present for each unchanged entry.

The delete phase may be used when the server can reliably determine which entries in the prior client copy are no longer present in the content and the number of such entries is less than or equal to the number of unchanged entries. In the delete mode, the server sends an empty entry with a Sync State Control indicating state delete for each entry that is no longer in the content, instead of returning an empty entry with state present for each present entry.

The server may send syncIdSet Sync Info Messages containing the set of UUIDs of either unchanged present entries or deleted entries, instead of sending multiple individual messages. If refreshDeletes of syncIdSet is set to FALSE, the UUIDs of unchanged present entries are contained in the syncUUIDs set; if refreshDeletes of syncIdSet is set to TRUE, the UUIDs of the entries no longer present in the content are contained in the syncUUIDs set. An optional cookie can

be included in the syncIdSet to represent the state of the content after synchronizing the presence or the absence of the entries contained in the syncUUIDs set.

The synchronized copy of the DIT fragment is constructed by the client.

If refreshDeletes of syncDoneValue is FALSE, the new copy includes all changed entries returned by the reissued Sync Operation, as well as all unchanged entries identified as being present by the reissued Sync Operation, but whose content is provided by the previous Sync Operation. The unchanged entries not identified as being present are deleted from the client content. They had been either deleted, moved, or otherwise scoped-out from the content.

If refreshDeletes of syncDoneValue is TRUE, the new copy includes all changed entries returned by the reissued Sync Operation, as well as all other entries of the previous copy except for those that are identified as having been deleted from the content.

The client can, at some later time, re-poll for changes to this synchronized client copy.

1.3.2. Listening for Changes (refreshAndPersist)

Polling for changes can be expensive in terms of server, client, and network resources. The refreshAndPersist mode allows for active updates of changed entries in the content.

By selecting the refreshAndPersist mode, the client requests that the server send updates of entries that are changed after the initial refresh content is determined. Instead of sending a SearchResultDone Message as in polling, the server sends a Sync Info Message to the client indicating that the refresh stage is complete and then enters the persist stage. After receipt of this Sync Info Message, the client will construct a synchronized copy as described in Section 1.3.1.

The server may then send change notifications as the result of the original Sync search request, which now remains persistent in the server. For entries to be added to the returned content, the server sends a SearchResultEntry (with attributes) with a Sync State Control indicating state add. For entries to be deleted from the content, the server sends a SearchResultEntry containing no attributes and a Sync State Control indicating state delete. For entries to be modified in the return content, the server sends a SearchResultEntry (with attributes) with a Sync State Control indicating state modify.

Upon modification of an entry, all (modified or unmodified) attributes belonging to the content are sent.

Note that renaming an entry of the DIT may cause an add state change where the entry is renamed into the content, a delete state change where the entry is renamed out of the content, and a modify state change where the entry remains in the content. Also note that a modification of an entry of the DIT may cause an add, delete, or modify state change to the content.

Upon receipt of a change notification, the client updates its copy of the content.

If the server desires to update the syncCookie during the persist stage, it may include the syncCookie in any Sync State Control or Sync Info Message returned.

The operation persists until canceled [RFC3909] by the client or terminated by the server. A Sync Done Control shall be attached to SearchResultDone Message to provide a new syncCookie.

1.4. 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 [RFC2119].

Protocol elements are described using ASN.1 [X.680] with implicit tags. The term "BER-encoded" means the element is to be encoded using the Basic Encoding Rules [X.690] under the restrictions detailed in Section 5.1 of [RFC4511].

2. Elements of the Sync Operation

The Sync Operation is defined as an extension to the LDAP Search Operation [RFC4511] where the directory user agent (DUA or client) submits a SearchRequest Message with a Sync Request Control and the directory system agent (DSA or server) responds with zero or more SearchResultEntry Messages, each with a Sync State Control; zero or more SearchResultReference Messages, each with a Sync State Control; zero or more Sync Info Intermediate Response Messages; and a SearchResultDone Message with a Sync Done Control.

To allow clients to discover support for this operation, servers implementing this operation SHOULD publish 1.3.6.1.4.1.4203.1.9.1.1 as a value of the 'supportedControl' attribute [RFC4512] of the root DSA-specific entry (DSE). A server MAY choose to advertise this extension only when the client is authorized to use it.

2.1. Common ASN.1 Elements

2.1.1. syncUUID

The syncUUID data type is an OCTET STRING holding a 128-bit (16-octet) Universally Unique Identifier (UUID) [UUID].

```
syncUUID ::= OCTET STRING (SIZE(16))
     -- constrained to UUID
```

2.1.2. syncCookie

The syncCookie is a notational convenience to indicate that, while the syncCookie type is encoded as an OCTET STRING, its value is an opaque value containing information about the synchronization session and its state. Generally, the session information would include a hash of the operation parameters that the server requires not be changed and the synchronization state information would include a commit (log) sequence number, a change sequence number, or a time stamp. For convenience of description, the term "no cookie" refers either to a null cookie or to a cookie with pre-initialized synchronization state.

```
syncCookie ::= OCTET STRING
```

2.2. Sync Request Control

The Sync Request Control is an LDAP Control [RFC4511] where the controlType is the object identifier 1.3.6.1.4.1.4203.1.9.1.1 and the controlValue, an OCTET STRING, contains a BER-encoded syncRequestValue. The criticality field is either TRUE or FALSE.

```
syncRequestValue ::= SEQUENCE {
    mode ENUMERATED {
        -- 0 unused
        refreshOnly       (1),
        -- 2 reserved
        refreshAndPersist (3)
    },
    cookie     syncCookie OPTIONAL,
    reloadHint BOOLEAN DEFAULT FALSE
}
```

The Sync Request Control is only applicable to the SearchRequest Message.

2.3. Sync State Control

The Sync State Control is an LDAP Control [RFC4511] where the controlType is the object identifier 1.3.6.1.4.1.4203.1.9.1.2 and the controlValue, an OCTET STRING, contains a BER-encoded syncStateValue. The criticality is FALSE.

```
syncStateValue ::= SEQUENCE {
    state ENUMERATED {
        present (0),
        add (1),
        modify (2),
        delete (3)
    },
    entryUUID syncUUID,
    cookie    syncCookie OPTIONAL
}
```

The Sync State Control is only applicable to SearchResultEntry and SearchResultReference Messages.

2.4. Sync Done Control

The Sync Done Control is an LDAP Control [RFC4511] where the controlType is the object identifier 1.3.6.1.4.1.4203.1.9.1.3 and the controlValue contains a BER-encoded syncDoneValue. The criticality is FALSE (and hence absent).

```
syncDoneValue ::= SEQUENCE {
    cookie          syncCookie OPTIONAL,
    refreshDeletes  BOOLEAN DEFAULT FALSE
}
```

The Sync Done Control is only applicable to the SearchResultDone Message.

2.5. Sync Info Message

The Sync Info Message is an LDAP Intermediate Response Message [RFC4511] where responseName is the object identifier 1.3.6.1.4.1.4203.1.9.1.4 and responseValue contains a BER-encoded syncInfoValue. The criticality is FALSE (and hence absent).

```
syncInfoValue ::= CHOICE {
    newcookie      [0] syncCookie,
    refreshDelete  [1] SEQUENCE {
        cookie         syncCookie OPTIONAL,
        refreshDone    BOOLEAN DEFAULT TRUE
    },
    refreshPresent [2] SEQUENCE {
        cookie         syncCookie OPTIONAL,
        refreshDone    BOOLEAN DEFAULT TRUE
    },
    syncIdSet      [3] SEQUENCE {
        cookie         syncCookie OPTIONAL,
        refreshDeletes BOOLEAN DEFAULT FALSE,
        syncUUIDs      SET OF syncUUID
    }
}
```

2.6. Sync Result Codes

The following LDAP resultCode [RFC4511] is defined:

e-syncRefreshRequired (4096)

3. Content Synchronization

The Sync Operation is invoked when the client sends a SearchRequest Message with a Sync Request Control.

The absence of a cookie or an initialized synchronization state in a cookie indicates a request for initial content, while the presence of a cookie representing a state of a client copy indicates a request for a content update. Synchronization Sessions are discussed in Section 3.1. Content Determination is discussed in Section 3.2.

The mode is either refreshOnly or refreshAndPersist. The refreshOnly and refreshAndPersist modes are discussed in Sections 3.3 and 3.4, respectively. The refreshOnly mode consists only of a refresh stage, while the refreshAndPersist mode consists of a refresh stage and a subsequent persist stage.

3.1. Synchronization Session

A sequence of Sync Operations where the last cookie returned by the server for one operation is provided by the client in the next operation is said to belong to the same Synchronization Session.

The client MUST specify the same content-controlling parameters (see Section 3.5) in each Search Request of the session. The client SHOULD also issue each Sync request of a session under the same authentication and authorization associations with equivalent integrity and protections. If the server does not recognize the request cookie or the request is made under different associations or non-equivalent protections, the server SHALL return the initial content as if no cookie had been provided or return an empty content with the e-syncRefreshRequired LDAP result code. The decision between the return of the initial content and the return of the empty content with the e-syncRefreshRequired result code MAY be based on reloadHint in the Sync Request Control from the client. If the server recognizes the request cookie as representing empty or initial synchronization state of the client copy, the server SHALL return the initial content.

A Synchronization Session may span multiple LDAP sessions between the client and the server. The client SHOULD issue each Sync request of a session to the same server. (Note: Shadowing considerations are discussed in Section 6.)

3.2. Content Determination

The content to be provided is determined by parameters of the Search Request, as described in [RFC4511], and possibly other controls. The same content parameters SHOULD be used in each Sync request of a session. If different content is requested and the server is unwilling or unable to process the request, the server SHALL return the initial content as if no cookie had been provided or return an empty content with the e-syncRefreshRequired LDAP result code. The decision between the return of the initial content and the return of the empty content with the e-syncRefreshRequired result code MAY be based on reloadHint in the Sync Request Control from the client.

The content may not necessarily include all entries or references that would be returned by a normal search operation, nor, for those entries included, all attributes returned by a normal search. When the server is unwilling or unable to provide synchronization for any attribute for a set of entries, the server MUST treat all filter components matching against these attributes as Undefined and MUST NOT return these attributes in SearchResultEntry responses.

Servers SHOULD support synchronization for all non-collective user-application attributes for all entries.

The server may also return continuation references to other servers or to itself. The latter is allowed as the server may partition the entries it holds into separate synchronization contexts.

The client may chase all or some of these continuations, each as a separate content synchronization session.

3.3. refreshOnly Mode

A Sync request with mode refreshOnly and with no cookie is a poll for initial content. A Sync request with mode refreshOnly and with a cookie representing a synchronization state is a poll for content update.

3.3.1. Initial Content Poll

Upon receipt of the request, the server provides the initial content using a set of zero or more SearchResultEntry and SearchResultReference Messages followed by a SearchResultDone Message.

Each SearchResultEntry Message SHALL include a Sync State Control of state add, an entryUUID containing the entry's UUID, and no cookie. Each SearchResultReference Message SHALL include a Sync State Control of state add, an entryUUID containing the UUID associated with the reference (normally the UUID of the associated named referral [RFC3296] object), and no cookie. The SearchResultDone Message SHALL include a Sync Done Control having refreshDeletes set to FALSE.

A resultCode value of success indicates that the operation successfully completed. Otherwise, the result code indicates the nature of the failure. The server may return e-syncRefreshRequired result code on the initial content poll if it is safe to do so when it is unable to perform the operation due to various reasons. reloadHint is set to FALSE in the SearchRequest Message requesting the initial content poll.

If the operation is successful, a cookie representing the synchronization state of the current client copy SHOULD be returned for use in subsequent Sync Operations.

3.3.2. Content Update Poll

Upon receipt of the request, the server provides the content refresh using a set of zero or more SearchResultEntry and

SearchResultReference Messages followed by a SearchResultDone Message.

The server is REQUIRED to:

a) provide the sequence of messages necessary for eventual convergence of the client's copy of the content to the server's copy,

b) treat the request as an initial content request (e.g., ignore the cookie or the synchronization state represented in the cookie),

c) indicate that the incremental convergence is not possible by returning e-syncRefreshRequired,

d) return a resultCode other than success or e-syncRefreshRequired.

A Sync Operation may consist of a single present phase, a single delete phase, or a present phase followed by a delete phase.

In each phase, for each entry or reference that has been added to the content or been changed since the previous Sync Operation indicated by the cookie, the server returns a SearchResultEntry or SearchResultReference Message, respectively, each with a Sync State Control consisting of state add, an entryUUID containing the UUID of the entry or reference, and no cookie. Each SearchResultEntry Message represents the current state of a changed entry. Each SearchResultReference Message represents the current state of a changed reference.

In the present phase, for each entry that has not been changed since the previous Sync Operation, an empty SearchResultEntry is returned whose objectName reflects the entry's current DN, whose attributes field is empty, and whose Sync State Control consists of state present, an entryUUID containing the UUID of the entry, and no cookie. For each reference that has not been changed since the previous Sync Operation, an empty SearchResultReference containing an empty SEQUENCE OF LDAPURL is returned with a Sync State Control consisting of state present, an entryUUID containing the UUID of the entry, and no cookie. No messages are sent for entries or references that are no longer in the content.

Multiple empty entries with a Sync State Control of state present SHOULD be coalesced into one or more Sync Info Messages of syncIdSet value with refreshDeletes set to FALSE. syncUUIDs contain a set of UUIDs of the entries and references unchanged since the last Sync

Operation. syncUUIDs may be empty. The Sync Info Message of syncIdSet may contain a cookie to represent the state of the content after performing the synchronization of the entries in the set.

In the delete phase, for each entry no longer in the content, the server returns a SearchResultEntry whose objectName reflects a past DN of the entry or is empty, whose attributes field is empty, and whose Sync State Control consists of state delete, an entryUUID containing the UUID of the deleted entry, and no cookie. For each reference no longer in the content, a SearchResultReference containing an empty SEQUENCE OF LDAPURL is returned with a Sync State Control consisting of state delete, an entryUUID containing the UUID of the deleted reference, and no cookie.

Multiple empty entries with a Sync State Control of state delete SHOULD be coalesced into one or more Sync Info Messages of syncIdSet value with refreshDeletes set to TRUE. syncUUIDs contain a set of UUIDs of the entries and references that have been deleted from the content since the last Sync Operation. syncUUIDs may be empty. The Sync Info Message of syncIdSet may contain a cookie to represent the state of the content after performing the synchronization of the entries in the set.

When a present phase is followed by a delete phase, the two phases are delimited by a Sync Info Message containing syncInfoValue of refreshPresent, which may contain a cookie representing the state after completing the present phase. The refreshPresent contains refreshDone, which is always FALSE in the refreshOnly mode of Sync Operation because it is followed by a delete phase.

If a Sync Operation consists of a single phase, each phase and hence the Sync Operation are marked as ended by a SearchResultDone Message with Sync Done Control, which SHOULD contain a cookie representing the state of the content after completing the Sync Operation. The Sync Done Control contains refreshDeletes, which is set to FALSE for the present phase and set to TRUE for the delete phase.

If a Sync Operation consists of a present phase followed by a delete phase, the Sync Operation is marked as ended at the end of the delete phase by a SearchResultDone Message with Sync Done Control, which SHOULD contain a cookie representing the state of the content after completing the Sync Operation. The Sync Done Control contains refreshDeletes, which is set to TRUE.

The client can specify whether it prefers to receive an initial content by supplying reloadHint of TRUE or to receive a e-syncRefreshRequired resultCode by supplying reloadHint of FALSE (hence absent), in the case that the server determines that it is

impossible or inefficient to achieve the eventual convergence by continuing the current incremental synchronization thread.

A resultCode value of success indicates that the operation is successfully completed. A resultCode value of e-syncRefreshRequired indicates that a full or partial refresh is needed. Otherwise, the result code indicates the nature of failure. A cookie is provided in the Sync Done Control for use in subsequent Sync Operations for incremental synchronization.

3.4. refreshAndPersist Mode

A Sync request with mode refreshAndPersist asks for initial content or content update (during the refresh stage) followed by change notifications (during the persist stage).

3.4.1. refresh Stage

The content refresh is provided as described in Section 3.3, except that the successful completion of content refresh is indicated by sending a Sync Info Message of refreshDelete or refreshPresent with a refreshDone value set to TRUE instead of a SearchResultDone Message with resultCode success. A cookie SHOULD be returned in the Sync Info Message to represent the state of the content after finishing the refresh stage of the Sync Operation.

3.4.2. persist Stage

Change notifications are provided during the persist stage.

As updates are made to the DIT, the server notifies the client of changes to the content. DIT updates may cause entries and references to be added to the content, deleted from the content, or modified within the content. DIT updates may also cause references to be added, deleted, or modified within the content.

Where DIT updates cause an entry to be added to the content, the server provides a SearchResultEntry Message that represents the entry as it appears in the content. The message SHALL include a Sync State Control with state of add, an entryUUID containing the entry's UUID, and an optional cookie.

Where DIT updates cause a reference to be added to the content, the server provides a SearchResultReference Message that represents the reference in the content. The message SHALL include a Sync State Control with state of add, an entryUUID containing the UUID associated with the reference, and an optional cookie.

Where DIT updates cause an entry to be modified within the content, the server provides a SearchResultEntry Message that represents the entry as it appears in the content. The message SHALL include a Sync State Control with state of modify, an entryUUID containing the entry's UUID, and an optional cookie.

Where DIT updates cause a reference to be modified within the content, the server provides a SearchResultReference Message that represents the reference in the content. The message SHALL include a Sync State Control with state of modify, an entryUUID containing the UUID associated with the reference, and an optional cookie.

Where DIT updates cause an entry to be deleted from the content, the server provides a SearchResultEntry Message with no attributes. The message SHALL include a Sync State Control with state of delete, an entryUUID containing the entry's UUID, and an optional cookie.

Where DIT updates cause a reference to be deleted from the content, the server provides a SearchResultReference Message with an empty SEQUENCE OF LDAPURL. The message SHALL include a Sync State Control with state of delete, an entryUUID containing the UUID associated with the reference, and an optional cookie.

Multiple empty entries with a Sync State Control of state delete SHOULD be coalesced into one or more Sync Info Messages of syncIdSet value with refreshDeletes set to TRUE. syncUUIDs contain a set of UUIDs of the entries and references that have been deleted from the content. The Sync Info Message of syncIdSet may contain a cookie to represent the state of the content after performing the synchronization of the entries in the set.

With each of these messages, the server may provide a new cookie to be used in subsequent Sync Operations. Additionally, the server may also return Sync Info Messages of choice newCookie to provide a new cookie. The client SHOULD use the newest (last) cookie it received from the server in subsequent Sync Operations.

3.5. Search Request Parameters

As stated in Section 3.1, the client SHOULD specify the same content-controlling parameters in each Search Request of the session. All fields of the SearchRequest Message are considered content-controlling parameters except for sizeLimit and timeLimit.

3.5.1. baseObject

As with the normal search operation, the refresh and persist stages are not isolated from DIT changes. It is possible that the entry referred to by the baseObject is deleted, renamed, or moved. It is also possible that the alias object used in finding the entry referred to by the baseObject is changed such that the baseObject refers to a different entry.

If the DIT is updated during processing of the Sync Operation in a manner that causes the baseObject no longer to refer to any entry or in a manner that changes the entry the baseObject refers to, the server SHALL return an appropriate non-success result code, such as noSuchObject, aliasProblem, aliasDereferencingProblem, referral, or e-syncRefreshRequired.

3.5.2. derefAliases

This operation does not support alias dereferencing during searching. The client SHALL specify neverDerefAliases or derefFindingBaseObj for the SearchRequest derefAliases parameter. The server SHALL treat other values (e.g., derefInSearching, derefAlways) as protocol errors.

3.5.3. sizeLimit

The sizeLimit applies only to entries (regardless of their state in Sync State Control) returned during the refreshOnly operation or the refresh stage of the refreshAndPersist operation.

3.5.4. timeLimit

For a refreshOnly Sync Operation, the timeLimit applies to the whole operation. For a refreshAndPersist operation, the timeLimit applies only to the refresh stage including the generation of the Sync Info Message with a refreshDone value of TRUE.

3.5.5. filter

The client SHOULD avoid filter assertions that apply to the values of the attributes likely to be considered by the server as ones holding meta-information. See Section 4.

3.6. objectName

The Sync Operation uses entryUUID values provided in the Sync State Control as the primary keys to entries. The client MUST use these entryUUIDs to correlate synchronization messages.

In some circumstances, the DN returned may not reflect the entry's current DN. In particular, when the entry is being deleted from the content, the server may provide an empty DN if the server does not wish to disclose the entry's current DN (or, if deleted from the DIT, the entry's last DN).

Also note that the entry's DN may be viewed as meta information (see Section 4.1).

3.7. Canceling the Sync Operation

Servers MUST implement the LDAP Cancel [RFC3909] Operation and support cancellation of outstanding Sync Operations as described here.

To cancel an outstanding Sync Operation, the client issues an LDAP Cancel [RFC3909] Operation.

If at any time the server becomes unwilling or unable to continue processing a Sync Operation, the server SHALL return a SearchResultDone with a non-success resultCode indicating the reason for the termination of the operation.

Whether the client or the server initiated the termination, the server may provide a cookie in the Sync Done Control for use in subsequent Sync Operations.

3.8. Refresh Required

In order to achieve the eventually-convergent synchronization, the server may terminate the Sync Operation in the refresh or persist stages by returning an e-syncRefreshRequired resultCode to the client. If no cookie is provided, a full refresh is needed. If a cookie representing a synchronization state is provided in this response, an incremental refresh is needed.

To obtain a full refresh, the client then issues a new synchronization request with no cookie. To obtain an incremental reload, the client issues a new synchronization with the provided cookie.

The server may choose to provide a full copy in the refresh stage (e.g., ignore the cookie or the synchronization state represented in the cookie) instead of providing an incremental refresh in order to achieve the eventual convergence.

The decision between the return of the initial content and the return of the e-syncRefreshRequired result code may be based on reloadHint in the Sync Request Control from the client.

In the case of persist stage Sync, the server returns the resultCode of e-syncRefreshRequired to the client to indicate that the client needs to issue a new Sync Operation in order to obtain a synchronized copy of the content. If no cookie is provided, a full refresh is needed. If a cookie representing a synchronization state is provided, an incremental refresh is needed.

The server may also return e-syncRefreshRequired if it determines that a refresh would be more efficient than sending all the messages required for convergence.

Note that the client may receive one or more of SearchResultEntry, SearchResultReference, and/or Sync Info Messages before it receives a SearchResultDone Message with the e-syncRefreshRequired result code.

3.9. Chattiness Considerations

The server MUST ensure that the number of entry messages generated to refresh the client content does not exceed the number of entries presently in the content. While there is no requirement for servers to maintain history information, if the server has sufficient history to allow it to reliably determine which entries in the prior client copy are no longer present in the content and the number of such entries is less than or equal to the number of unchanged entries, the server SHOULD generate delete entry messages instead of present entry messages (see Section 3.3.2).

When the amount of history information maintained in the server is not enough for the clients to perform infrequent refreshOnly Sync Operations, it is likely that the server has incomplete history information (e.g., due to truncation) by the time those clients connect again.

The server SHOULD NOT resort to full reload when the history information is not enough to generate delete entry messages. The server SHOULD generate either present entry messages only or present entry messages followed by delete entry messages to bring the client copy to the current state. In the latter case, the present entry messages bring the client copy to a state covered by the history information maintained in the server.

The server SHOULD maintain enough (current or historical) state information (such as a context-wide last modify time stamp) to determine if no changes were made in the context since the content

refresh was provided and, when no changes were made, generate zero delete entry messages instead of present messages.

The server SHOULD NOT use the history information when its use does not reduce the synchronization traffic or when its use can expose sensitive information not allowed to be received by the client.

The server implementor should also consider chattiness issues that span multiple Sync Operations of a session. As noted in Section 3.8, the server may return e-syncRefreshRequired if it determines that a reload would be more efficient than continuing under the current operation. If reloadHint in the Sync Request is TRUE, the server may initiate a reload without directing the client to request a reload.

The server SHOULD transfer a new cookie frequently to avoid having to transfer information already provided to the client. Even where DIT changes do not cause content synchronization changes to be transferred, it may be advantageous to provide a new cookie using a Sync Info Message. However, the server SHOULD avoid overloading the client or network with Sync Info Messages.

During persist mode, the server SHOULD coalesce multiple outstanding messages updating the same entry. The server MAY delay generation of an entry update in anticipation of subsequent changes to that entry that could be coalesced. The length of the delay should be long enough to allow coalescing of update requests issued back to back but short enough that the transient inconsistency induced by the delay is corrected in a timely manner.

The server SHOULD use the syncIdSet Sync Info Message when there are multiple delete or present messages to reduce the amount of synchronization traffic.

Also note that there may be many clients interested in a particular directory change, and that servers attempting to service all of these at once may cause congestion on the network. The congestion issues are magnified when the change requires a large transfer to each interested client. Implementors and deployers of servers should take steps to prevent and manage network congestion.

3.10. Operation Multiplexing

The LDAP protocol model [RFC4511] allows operations to be multiplexed over a single LDAP session. Clients SHOULD NOT maintain multiple LDAP sessions with the same server. Servers SHOULD ensure that responses from concurrently processed operations are interleaved fairly.

Clients SHOULD combine Sync Operations whose result set is largely overlapping. This avoids having to return multiple messages, once for each overlapping session, for changes to entries in the overlap.

Clients SHOULD NOT combine Sync Operations whose result sets are largely non-overlapping. This ensures that an event requiring an e-syncRefreshRequired response can be limited to as few result sets as possible.

4. Meta Information Considerations

4.1. Entry DN

As an entry's DN is constructed from its relative DN (RDN) and the entry's parent's DN, it is often viewed as meta information.

While renaming or moving to a new superior causes the entry's DN to change, that change SHOULD NOT, by itself, cause synchronization messages to be sent for that entry. However, if the renaming or the moving could cause the entry to be added or deleted from the content, appropriate synchronization messages should be generated to indicate this to the client.

When a server treats the entry's DN as meta information, the server SHALL either

- evaluate all MatchingRuleAssertions [RFC4511] to TRUE if matching a value of an attribute of the entry, otherwise Undefined, or

- evaluate all MatchingRuleAssertion with dnAttributes of TRUE as Undefined.

The latter choice is offered for ease of server implementation.

4.2. Operational Attributes

Where values of an operational attribute are determined by values not held as part of the entry it appears in, the operational attribute SHOULD NOT support synchronization of that operational attribute.

For example, in servers that implement the X.501 subschema model [X.501], servers should not support synchronization of the subschemaSubentry attribute as its value is determined by values held and administrated in subschema subentries.

As a counter example, servers that implement aliases [RFC4512][X.501] can support synchronization of the aliasedObjectName attribute as its values are held and administrated as part of the alias entries.

Servers SHOULD support synchronization of the following operational attributes: createTimestamp, modifyTimestamp, creatorsName, modifiersName [RFC4512]. Servers MAY support synchronization of other operational attributes.

4.3. Collective Attributes

A collective attribute is "a user attribute whose values are the same for each member of an entry collection" [X.501]. Use of collective attributes in LDAP is discussed in [RFC3671].

Modification of a collective attribute generally affects the content of multiple entries, which are the members of the collection. It is inefficient to include values of collective attributes visible in entries of the collection, as a single modification of a collective attribute requires transmission of multiple SearchResultEntry (one for each entry of the collection that the modification affected).

Servers SHOULD NOT synchronize collective attributes appearing in entries of any collection. Servers MAY support synchronization of collective attributes appearing in collective attribute subentries.

4.4. Access and Other Administrative Controls

Entries are commonly subject to access and other administrative Controls. While portions of the policy information governing a particular entry may be held in the entry, policy information is often held elsewhere (in superior entries, in subentries, in the root DSE, in configuration files, etc.). Because of this, changes to policy information make it difficult to ensure eventual convergence during incremental synchronization.

Where it is impractical or infeasible to generate content changes resulting from a change to policy information, servers may opt to return e-syncRefreshRequired or to treat the Sync Operation as an initial content request (e.g., ignore the cookie or the synchronization state represented in the cookie).

5. Interaction with Other Controls

The Sync Operation may be used with:

- ManageDsaIT Control [RFC3296]

- Subentries Control [RFC3672]

as described below. The Sync Operation may be used with other LDAP extensions as detailed in other documents.

5.1. ManageDsaIT Control

The ManageDsaIT Control [RFC3296] indicates that the operation acts upon the DSA Information Tree and causes referral and other special entries to be treated as object entries with respect to the operation.

5.2. Subentries Control

The Subentries Control is used with the search operation "to control the visibility of entries and subentries which are within scope" [RFC3672]. When used with the Sync Operation, the subentries control and other factors (search scope, filter, etc.) are used to determine whether an entry or subentry appears in the content.

6. Shadowing Considerations

As noted in [RFC4511], some servers may hold shadow copies of entries that can be used to answer search and comparison queries. Such servers may also support content synchronization requests. This section discusses considerations for implementors and deployers for the implementation and deployment of the Sync operation in shadowed directories.

While a client may know of multiple servers that are equally capable of being used to obtain particular directory content from, a client SHOULD NOT assume that each of these servers is equally capable of continuing a content synchronization session. As stated in Section 3.1, the client SHOULD issue each Sync request of a Sync session to the same server.

However, through domain naming or IP address redirection or other techniques, multiple physical servers can be made to appear as one logical server to a client. Only servers that are equally capable in regards to their support for the Sync operation and that hold equally complete copies of the entries should be made to appear as one logical server. In particular, each physical server acting as one logical server SHOULD be equally capable of continuing a content synchronization based upon cookies provided by any of the other physical servers without requiring a full reload. Because there is no standard LDAP shadowing mechanism, the specification of how to independently implement equally capable servers (as well as the precise definition of "equally capable") is left to future documents.

Note that it may be difficult for the server to reliably determine what content was provided to the client by another server, especially in the shadowing environments that allow shadowing events to be coalesced. For these servers, the use of the delete phase discussed in Section 3.3.2 may not be applicable.

7. Security Considerations

In order to maintain a synchronized copy of the content, a client is to delete information from its copy of the content as described above. However, the client may maintain knowledge of information disclosed to it by the server separate from its copy of the content used for synchronization. Management of this knowledge is beyond the scope of this document. Servers should be careful not to disclose information for content the client is not authorized to have knowledge of and/or about.

While the information provided by a series of refreshOnly Sync Operations is similar to that provided by a series of Search Operations, persist stage may disclose additional information. A client may be able to discern information about the particular sequence of update operations that caused content change.

Implementors should take precautions against malicious cookie content, including malformed cookies or valid cookies used with different security associations and/or protections in an attempt to obtain unauthorized access to information. Servers may include a digital signature in the cookie to detect tampering.

The operation may be the target of direct denial-of-service attacks. Implementors should provide safeguards to ensure the operation is not abused. Servers may place access control or other restrictions upon the use of this operation.

Note that even small updates to the directory may cause a significant amount of traffic to be generated to clients using this operation. A user could abuse its update privileges to mount an indirect denial of service to these clients, other clients, and/or portions of the network. Servers should provide safeguards to ensure that update operations are not abused.

Implementors of this (or any) LDAP extension should be familiar with general LDAP security considerations [RFC4510].

8. IANA Considerations

Registration of the following values have been completed by the IANA [RFC4520].

8.1. Object Identifier

The OID arc 1.3.6.1.4.1.4203.1.9.1 was assigned [ASSIGN] by the OpenLDAP Foundation, under its IANA-assigned private enterprise allocation [PRIVATE], for use in this specification.

8.2. LDAP Protocol Mechanism

The IANA has registered the LDAP Protocol Mechanism described in this document.

```
Subject: Request for LDAP Protocol Mechanism Registration
Object Identifier: 1.3.6.1.4.1.4203.1.9.1.1
Description: LDAP Content Synchronization Control
Person & email address to contact for further information:
    Kurt Zeilenga <kurt@openldap.org>
Usage: Control
Specification: RFC 4533
Author/Change Controller: Kurt D. Zeilenga, Jong Hyuk Choi
Comments: none
```

8.3. LDAP Result Codes

The IANA has registered the LDAP Result Code described in this document.

```
Subject: LDAP Result Code Registration
Person & email address to contact for further information:
    Kurt Zeilenga <kurt@OpenLDAP.org>
Result Code Name: e-syncRefreshRequired (4096)
Specification: RFC 4533
Author/Change Controller: Kurt D. Zeilenga, Jong Hyuk Choi
Comments:  none
```

9. Acknowledgements

This document borrows significantly from the LDAP Client Update Protocol [RFC3928], a product of the IETF LDUP working group. This document also benefited from Persistent Search [PSEARCH], Triggered Search [TSEARCH], and Directory Synchronization [DIRSYNC] works. This document also borrows from "Lightweight Directory Access Protocol (v3)" [RFC2251].

10. Normative References

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

[RFC3296] Zeilenga, K., "Named Subordinate References in Lightweight Directory Access Protocol (LDAP) Directories", RFC 3296, July 2002.

[RFC3671] Zeilenga, K., "Collective Attributes in the Lightweight Directory Access Protocol (LDAP)", RFC 3671, December 2003.

[RFC3672] Zeilenga, K., "Subentries in the Lightweight Directory Access Protocol (LDAP)", RFC 3672, December 2003.

[RFC3909] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP) Cancel Operation", RFC 3909, October 2004.

[RFC4510] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.

[RFC4511] Sermersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.

[RFC4512] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Directory Information Models", RFC 4512, June 2006.

[RFC4530] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP) entryUUID Operational Attribute", RFC 4530, June 2006.

[UUID] International Organization for Standardization (ISO), "Information technology - Open Systems Interconnection - Remote Procedure Call", ISO/IEC 11578:1996

[X.501] International Telecommunication Union - Telecommunication Standardization Sector, "The Directory -- Models," X.501(1993) (also ISO/IEC 9594-2:1994).

[X.680] International Telecommunication Union - Telecommunication Standardization Sector, "Abstract Syntax Notation One (ASN.1) - Specification of Basic Notation", X.680(1997) (also ISO/IEC 8824-1:1998).

[X.690] International Telecommunication Union - Telecommunication Standardization Sector, "Specification of ASN.1 encoding rules: Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)", X.690(1997) (also ISO/IEC 8825-1:1998).

11. Informative References

[RFC2251] Wahl, M., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997.

[RFC3928] Megginson, R., Ed., Smith, M., Natkovich, O., and J. Parham, "Lightweight Directory Access Protocol (LDAP) Client Update Protocol (LCUP)", RFC 3928, October 2004.

[RFC4520] Zeilenga, K., "Internet Assigned Numbers Authority (IANA) Considerations for the Lightweight Directory Access Protocol (LDAP)", BCP 64, RFC 4520, June 2006.

[PRIVATE] IANA, "Private Enterprise Numbers", http://www.iana.org/assignments/enterprise-numbers.

[ASSIGN] OpenLDAP Foundation, "OpenLDAP OID Delegations", http://www.openldap.org/foundation/oid-delegate.txt.

[X.500] International Telecommunication Union - Telecommunication Standardization Sector, "The Directory -- Overview of concepts, models and services," X.500(1993) (also ISO/IEC 9594-1:1994).

[X.525] International Telecommunication Union - Telecommunication Standardization Sector, "The Directory: Replication", X.525(1993).

[DIRSYNC] Armijo, M., "Microsoft LDAP Control for Directory Synchronization", Work in Progress.

[PSEARCH] Smith, M., et al., "Persistent Search: A Simple LDAP Change Notification Mechanism", Work in Progress.

[TSEARCH] Wahl, M., "LDAPv3 Triggered Search Control", Work in Progress.

Appendix A. CSN-based Implementation Considerations

This appendix is provided for informational purposes only; it is not a normative part of the LDAP Content Synchronization Operation's technical specification.

This appendix discusses LDAP Content Synchronization Operation server implementation considerations associated with Change Sequence Number based approaches.

Change Sequence Number based approaches are targeted for use in servers that do not maintain history information (e.g., change logs, state snapshots) about changes made to the Directory and hence, must rely on current directory state and minimal synchronization state information embedded in Sync Cookie. Servers that maintain history information should consider other approaches that exploit the history information.

A Change Sequence Number is effectively a time stamp that has sufficient granularity to ensure that the precedence relationship in time of two updates to the same object can be determined. Change Sequence Numbers are not to be confused with Commit Sequence Numbers or Commit Log Record Numbers. A Commit Sequence Number allows one to determine how two commits (to the same object or different objects) relate to each other in time. A Change Sequence Number associated with different entries may be committed out of order. In the remainder of this Appendix, the term CSN refers to a Change Sequence Number.

In these approaches, the server not only maintains a CSN for each directory entry (the entry CSN) but also maintains a value that we will call the context CSN. The context CSN is the greatest committed entry CSN that is not greater than any outstanding (uncommitted) entry CSNs for all entries in a directory context. The values of context CSN are used in syncCookie values as synchronization state indicators.

As search operations are not isolated from individual directory update operations and individual update operations cannot be assumed to be serialized, one cannot assume that the returned content incorporates each relevant change whose change sequence number is less than or equal to the greatest entry CSN in the content. The content incorporates all the relevant changes whose change sequence numbers are less than or equal to context CSN before search processing. The content may also incorporate any subset of the changes whose change sequence number is greater than context CSN before search processing but less than or equal to the context CSN after search processing. The content does not incorporate any of the

changes whose CSN is greater than the context CSN after search processing.

A simple server implementation could use the value of the context CSN before search processing to indicate state. Such an implementation would embed this value into each SyncCookie returned. We'll call this the cookie CSN. When a refresh was requested, the server would simply generate "update" messages for all entries in the content whose CSN is greater than the supplied cookie CSN and generate "present" messages for all other entries in the content. However, if the current context CSN is the same as the cookie CSN, the server should instead generate zero "updates" and zero "delete" messages and indicate a refreshDeletes of TRUE, as the directory has not changed.

The implementation should also consider the impact of changes to meta information, such as access controls, that affect content determination. One approach is for the server to maintain a context-wide meta information CSN or meta CSN. This meta CSN would be updated whenever meta information affecting content determination was changed. If the value of the meta CSN is greater than the cookie CSN, the server should ignore the cookie and treat the request as an initial request for content.

Additionally, servers may want to consider maintaining some per-session history information to reduce the number of messages needed to be transferred during incremental refreshes. Specifically, a server could record information about entries as they leave the scope of a disconnected sync session and later use this information to generate delete messages instead of present messages.

When the history information is truncated, the CSN of the latest truncated history information entry may be recorded as the truncated CSN of the history information. The truncated CSN may be used to determine whether a client copy can be covered by the history information by comparing it to the synchronization state contained in the cookie supplied by the client.

When there is a large number of sessions, it may make sense to maintain such history only for the selected clients. Also, servers taking this approach need to consider resource consumption issues to ensure reasonable server operation and to protect against abuse. It may be appropriate to restrict this mode of operation by policy.

Authors' Addresses

Kurt D. Zeilenga
OpenLDAP Foundation

EMail: Kurt@OpenLDAP.org

Jong Hyuk Choi
IBM Corporation

EMail: jongchoi@us.ibm.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).