Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-rps-dist-01

Curtis Villamizar
UUNET
Cengiz Alaettinoglu
ISI
Ramesh Govindan
ISI
David M. Meyer
Cisco
February 23, 1999

Distributed Routing Policy System

Status of this Memo

Abstract

The RIPE database specifications [2] and RPSL language [1] define languages used as the basis for representing information in a routing policy system. A repository for routing policy system information is known as a routing registry. A routing registry provides a means of exchanging information needed to address many issues on importance to the operation of the Internet. The implementation and deployment of a routing policy system must maintain some degree of integrity to be of any use. The Routing Policy System Security internet-draft [3] addresses the need to assure integrity of the data by proposing an authentication and authorization model. This document addresses the need to distribute data over multiple repositories and delegate authority for data subsets to multiple repositories without compromising the authorization model established in [3].

1 Overview

A routing registry must maintain some degree of integrity to be of any use. The IRR is increasingly used for purposes that have a stronger requirement for data integrity and security. There is also a desire to further decentralize the IRR. This document proposes a means of decentralizing the routing registry in a way that is consistent with the usage of the IRR and which avoids compromising data integrity and security even if the IRR is distributed among less trusted repositories.

Two methods of authenticating the routing registry information have been proposed.

authorization and authentication checks on transactions: The integrity of the routing registry data is insured by repeating authorization checks as transactions are processed. As transactions are flooded each remote registry has the option to repeat the authorization and authentication checks. This scales with the total number of changes to the registry regardless of how many registries exist. When querying, the integrity of the repository must be such that it can be trusted. If an organization is unwilling to trust any of the available repositories or mirrors they have the option to run their own mirror and repeat authorization checks at that mirror site. Queries can then be directed to a mirror under their own administration which presumably can be trusted.

signing routing registry objects: An alternate which appears on the surface to be attractive is signing the objects themselves. Closer examination reveals that the approach of signing objects by itself is flawed and when used in addition to signing transactions and rechecking authorizations as changes are made adds nothing. In order for an insertion of critical objects such as inetnums and routes to be valid, authorization checks must be made which allow the insertion. The objects on which those authorization checks are made may later change. In order to later repeat the authorization checks the state of other objects, possibly in other repositories would have to be known. If the repository were not trusted then the change history on the object would have to be traced back to the object's insertion. If the repository were not trusted, the change history of any object that was depended upon for authorization would also have to be rechecked. This trace back would have to go back to the epoch or at least to a point where only trusted objects were being relied upon for the authorizations. If the depth of the search is at all limited, authorization could be falsified simply by exceeding the search depth with a chain of authorization references back to falsified objects. This would be grossly inefficient. Simply verifying that an object is signed provides no assurance that addition of the object addition was properly authorized.

A minor distinction is made between a repository and a mirror. A repository has responsibility for the initial authorization and authentication checks for transactions related to its local objects which are then flooded to adjacent repositories. A mirror receives flooded transactions from remote repositories but is not the authoritative source for any objects. From a protocol standpoint, repositories and mirrors appear identical in the flooding topology.

Either a repository or a mirror may recheck all or a subset of transactions that are flooded to it. A repository or mirror may elect not to recheck authorization and authentication on transactions received from a trusted adjacency on the grounds that the adjacent repository is trusted and would not have flooded the information unless authorization and authentication checks had been made.

If it can be arranged that all adjacencies are trusted for a given mirror, then there is no need to implement the code to check authorization and authentication. There is only a need to be able to check the signatures on the flooded transactions of the adjacent repository. This is an important special case because it could allow a router to act as a mirror. Only changes to the registry database would be received through flooding, which is a very low volume. Only the signature of the adjacent mirror or repository would have to be checked.

2 Data Representation

RPSL provides a complete description of the contents of a routing repository [1]. Many RPSL data objects remain unchanged from the RIPE and RPSL references the RIPE-181 specification as recorded in RFC-1786 [2]. RPSL provides external data representation. Data may be stored differently internal to an routing registry. The integrity of the distributed registry data requires the use of the authorization and authentication additions to RPSL described in [3].

Some additions to RPSL are needed to locate all of the repositories after having located one of them and to make certain parameters selectable on a per repository basis readily available. These additions are described in Section 5.

Some form of encapsulation must be used to exchange data. The de-facto encapsulation has been that which the RIPE tools accept, a plain text file or plain text in the body of an RFC-822 formatted mail message with information needed for authentication derived from the mail headers. Merit has slightly modified this using the PGP signed portion of a plain text file or PGP signed portion of the body of a mail message.

The exchange that occurs during flooding differs from the initial submission. In order to repeat the authorization checks the state of all repositories containing objects referenced by the authorization checks needs to be known. To accomplish this a sequence number is associated with each transaction in a repository and the flooded transactions must contain the sequence number of each repository on which authorization of the transaction depends.

In order to repeat authorization checks it must be possible to retrieve back revisions of objects. How this is accomplished is a matter local to the implementation. One method which is quite simple is to keep the traversal data structures to all current objects even if the state is deleted, keep the sequence number that the version of the object became effective and keep back links to prior versions of the objects. Finding a prior version of an object involves looking back through the references until the sequence number of the version of the object is less than or equal to the sequence number being searched for.

The existing very simple forms of encapsulation are adequate for the initial submission of a database transaction and should be retained as long as needed for backward compatibility. A more robust encapsulation and submission protocol, with optional confirmation is defined in Section 6.1. An encapsulation suitable for exchange of transaction between repositories is addressed in Section 6. Query encapsulation and protocol is outside the scope of this document.

3 Authentication and Authorization

Control must be exercised over who can make changes and what changes they can make. The distinction of who vs what separates authentication from authorization.

- Authentication is the means to determine who is attempting to make a change.
- Authorization is the determination of whether a transaction passing a specific authentication check is allowed to perform a given operation.

A submitted transaction contains a claimed identity. Depending on the type of transaction, the authorization will depend on related objects. The “mnt-by”, “mnt-routes”, or “mnt-lower” attributes in those related objects reference “maintainer” objects. Those maintainer objects contain “auth” attributes. The auth attributes contain an authorization method and data which generally contains the claimed identity and some form of public encryption key used to authenticate the claim.

Authentication is done on transactions. Authentication should also be done between repositories to insure the integrity of the information exchange. In order to comply with import, export, and use restrictions throughout the world no encryption capability is specified. Transactions must not be encrypted because it may be illegal to use decryption software in some parts of the world.

4 Repository Hierarchy

With multiple repositories, “repository” objects are needed to propagate the existence of new repositories and provide an automated means to determine the supported methods of access and other characteristics of the repository. The repository object is described in Section 5.

In each repository there should be a special repository object named ROOT. This should point to the root repository or to a higher level repository. This is to allow queries to be directed to the local repository but refer to the full set of registries for resolution of hierarchically allocated objects.

Each repository may have an “expire” attribute. Flooding is preferred. The expire attribute is used to determine if a repository must be updated before a local transaction can that depends on it can proceed.

The repository object also contains attributes describing the access methods and supported authentication methods of the repository. The “query-address” attribute provides a host name and a port number used to direct queries. The “response-auth-type” attribute provides the authentication types that may be used by the repository when responding to queries. The “submit-address” attribute provides a host name and a port number used to submit objects to the repository. The “submit-auth-type” attribute provides the authentication types that may be used by the repository when responding to queries.

5 Additions to RPSL

There are very few additions to RPSL defined here. The additions to RPSL are referred to as RPSL "objects". They reside in the repository database and can be retrieved with ordinary queries. Objects consist of "attributes", which are name/value pairs. Attributes may be mandatory or optional. They may be single or multiple. One or more attributes may be part of a key field. Some attributes may have the requirement of being unique.

Most of the data formats described in this document are encapsulations used in transaction exchanges. These are referred to as "meta-objects". These "meta-objects", unlike RPSL "objects" do not reside in the database but some must be retained in a transaction log. A similar format is used to represent "meta-objects". They also consist of "attributes" which are name/value pairs.

This section contains all of the additions to RPSL described in this document. This section describes only RPSL objects. Other sections described only meta-objects.

5.1 repository object

A root repository must be agreed upon. Ideally such a repository would contain only top level delegations and pointers to other repositories used in these delegations. It would be wise to allow only cryptographically strong transactions in the root repository [3].

The root repository contains references to other repositories. An object of the following form identifies another repository.

```
repository:          RIPE
query-address:       whois.ripe.net 43
response-auth-type: rsa-pubkey some-incredibly-long-public-key
response-auth-type: none
remarks:             you can request rsa signature on queries
transaction-auth:    PGPKEY
remarks:             PGP required on submissions
submit-address:      mailto://auto-dbm@ripe.net
submit-address:      rps-query://whois.ripe.net:43
submit-auth-type:    pgp-key crypt-pw mail-from
remarks:             these are the authentication types supported
mnt-by:              maint-ripe-db
expire:              4h
heartbeat-rate:      1h
...
remarks:             admin and technical contact, etc
source:              IANA
```

In each repository there should be a special repository object named ROOT. This should point to the root repository or to a higher level repository. This is to allow queries to be directed to the local repository but refer to the full set of registries for resolution of hierarchically allocated objects.

The attributes of the repository object are listed below.

```
repository      key  mandatory  single    unique
query-address        mandatory  multiple
response-auth-type   mandatory  multiple
submit-address       mandatory  multiple
submit-auth-type     mandatory  multiple
repository-cert      mandatory  multiple
expire               mandatory  single
heartbeat-rate       mandatory  single
descr                optional   multiple
remarks              optional   multiple
admin-c              mandatory  multiple
tech-c               mandatory  multiple
notify               optional   multiple
mnt-by               mandatory  multiple
changed              mandatory  multiple
source               mandatory  single
```

In the above object type only a small number of the attribute types are new. These are:

repository This attribute provides the name of the repository. This is the key field for the object and is single and must be globally unique. This is the same name used in the source attribute of all objects in that repository.

query-address This attribute provides a host name and a port number used to direct queries. Optional fields may follow the port number at a later time and should be ignored. Alternately a URL format may be used. The special protocol identified "rps-query" can be used in URL format.

response-auth-type This attribute provides the authentication types that may be used by the repository when responding to queries.

submit-address This attribute provides a host name and a port number used to submit objects to the repository. Optional fields may follow the port number at a later time and should be ignored. Alternately a URL format may be used. The special protocol identified "rps-query" can be used in URL format.

submit-auth-type This attribute provides the authentication types that may be used by the repository when responding to queries.

repository-cert This attribute provides a reference to a public key certificate in the form of an RPSL key-cert object. This attribute can be multiple to allow the repository to use more than one method of signature.

heartbeat-rate Heartbeat meta-objects are sent by this repository at the rate of one heartbeat meta-object per the period indicated. Time is in seconds if expressed as an integer. Alternately, the letters "h", "m", or "s" can be used to indicate hours, minutes, and seconds.

expire If near real time flooding is temporarily not operational, objects should be considered non-authoritative after this interval, and cannot be used for authorization purposes.

Please note that the “heartbeat” meta-objects mentioned above, like other meta-objects described in this document are part of the protocol to exchange information but are not placed in the database itself. See Section 7.4.5 for a description of the heartbeat meta-object.

The remaining attributes in the repository object are defined in RPSL.

5.2 delegated attribute

For many RPSL object types a particular entry should appear only in one repository. These are the object types for which there is a natural hierarchy, “as-block”, “aut-num”, “inetnum”, and “route”. In order to facilitate putting an object in another repository, a “delegated” attribute is added.

delegated The delegated attribute is allowed in any object type with a hierarchy. This attribute indicates that further searches for object in the hierarchy must be made in one or more alternate repositories. The current repository may be listed. The ability to list more than one repository serves only to accommodate grandfathered objects (those created prior to using an authorization model).

If an object contains a “delegated” attribute, an exact key field match of the object should be contained in each repository listed in the “delegated” attribute. For the purpose of authorizing changes only the “mnt-by” in the object in the repository being modified is considered.

The following is an example of the use of a “delegated” attribute.

```
inetnum:        193.0.0.0 - 193.0.0.255
delegated:      RIPE
...
source:         IANA
```

This inetnum simply delegates the storage of any more specific inetnum objects overlapping the stated range to the RIPE repository. An exact match of this inetnum must exist in the RIPE repository to provide hooks for the attributes referencing maintainer objects. In this example, when adding objects to the RIPE repository, the “mnt-lower”, ‘mnt-routes”, and “mnt-by” fields in the IANA inetnum object will not be considered.

5.3 integrity attribute

The “integrity” attribute can be contained in any RPSL object type. It is intended solely as a means to facilitate a transition period during when some data has been moved from repositories prior to the use of a strong authorization model and is therefore questionable, or when some repositories are not properly checking authorization.

The “integrity” attribute may have the values “legacy”, “no-auth”, “auth-failed”, or “authorized”. If absent, the integrity is considered to be “authorized”. The integrity values have the following meanings:

legacy: This data existed prior to the use of an adequate authorization model. The data is highly suspect.

no-auth: This data was added to a repository during an initial transition use of an authorization model but authorization was with a maintainer that did not have a "referral-by" attribute, or used a discouraged weak form of authentication, or relied upon other objects for authorization whose integrity was not "authorized". Such an addition is being allowed during the transition but would be disallowed later.

auth-failed: The authoritative repository is not checking authorization. Had it been doing so, authorization would have failed. This attribute may be added by a repository that is mirroring before placing the object in its local storage, or can add this attribute to an encapsulating meta-object used to further propagate the transaction. If the failure to enforce authorization is intentional and part of a transition (for example, issuing warnings only), then the authoritative repository may add this attribute to the encapsulating meta-object used to further propagate the transaction.

authorized: Authorization checks were passed. The maintainer contained a "referral-by" attribute, a form of authentication deemed adequate by the repository was used, and all objects that were needed for authorization were objects whose integrity was "authorized".

Normally once an object is added to a repository another object cannot overwrite it unless authorized to do so by the maintainers referenced by the "mnt-by" attributes in the object itself. If the integrity attribute is anything but "authorized", an object can be overwritten or deleted by any transaction that would have been a properly authorized addition had the object of lesser integrity not existed.

During such a transition grandfathered data and data added without proper authorization becomes advisory until a properly authorized addition occurs. After transition additions of this type would no longer be accepted. Those objects already added without proper authorization would remain but would be marked as candidates for replacement.

6 Interactions with a Repository or Mirror

There are a few different types of interactions between routing repositories or mirrors.

Initial submission of transactions: Transactions may include additions, changes, and deletions. A transaction may operate on more than one object and must be treated as an atomic operation. By definition initial submission of transactions is not applicable to a mirror. Initial submission of transactions is described in Section 6.1.

Redistribution of Transactions: The primary purpose of the interactions between registries is the redistribution of transactions. There are a number of ways to redistribute transactions. Transactions can also be rescinded. This is discussed in Section 6.2.

Queries: Query interactions are outside the scope of this document.

Rescinding Transactions Although it is hoped that the feature is never needed, it may be necessary to rescind transactions (Section 6.3).

Transaction Commit and Confirmation: Repositories may optionally implement a commit protocol and a completion indication that gives the submitter of a transaction an response that indicates that a transaction has been successful and will not be lost by a crash of the local repository. A submitter may optionally request such a confirmation. This is discussed in Section 6.4.

6.1 Initial Transaction Submission

The simplest form of transaction submission is an object or set of objects submitted with RFC–822 encapsulation. This form is still supported for backwards compatibility. A preferred form allows some meta-information to be included in the submission, such as a preferred form of confirmation. Where either encapsulation is used, the submitter will connect to a host and port specified in the repository object. This allows immediate confirmation. If an email interface similar to the interface provided by the existing RIPE code is desired, then an external program can provide the email interface.

The encapsulation of a transaction submission and response is described in detail in Section 7.

6.2 Redistribution of Transactions

Redistribution of transactions can be accomplished using three types of requests for redistribution of transactions.

1. A repository snapshots is a request for the complete contents of a given repository. This is usually done when starting up a new repository or mirror or when recovering from a disaster.
2. A transaction sequence exchange is a request for a specific set of transactions. Often the request is for the most recent sequence number known to a mirror to the last transactions. This is used in polling.
3. Transaction flooding is accomplished through a unicast adjacency.

This section describes the operations somewhat qualitatively. Data formats and state diagrams are provided in Section 7.

6.3 Rescinding Transactions

Rescinding a transaction is a manual intervention. The administrators of a repository may find it necessary to request that a specific set of transactions be removed. Database mirrors would have to roll back the entire database to the first transaction being rescinded and then roll forward the transaction log from that point forward. Authorizations in other repositories may be affected.

There are many reasons for having to rescind a transaction whose cause is outside the control of the operator of the repository. For example, a disgruntled employee at a client of the repository may remove all authorization from that clients database objects. There may be opportunities for malicious entries in objects for which there is no authorization hierarchy (See [3]). An example is the anonymous registration of falsified person objects or libelous or obscene person or role objects. In addition, mistakes or program bugs are inevitable.

The most compelling reason to support a rescind is the possibility of a security breach at the repository that results in theft of identity of the repository and falsified transactions. The rescind feature is one that needs to be included but the repositories hope to never need.

6.4 Transaction Commit and Confirmation

If a submission requires a strong confirmation of completion, or if a higher degree of protection against false positive confirmation is desired as a matter of repository policy, a commit may be performed.

A commit request is a request from the repository processing an initial transaction submission to another repository to confirm that they have been able to advance the transaction sequence up to the sequence number immediately below the transaction in the request and are willing to accept the transaction in the request as a further advance in the sequence. This indicates that either the authorization was rechecked by the responding repository and passed or that the responding repository trusts the requesting repository and has accepted the transaction.

A commit request can be sent to more than one alternate repository. One commit completion response is sufficient to respond to the submitter with a positive confirmation that the transaction has been completed however the repository or submitter may optionally require more than one.

7 Data Format Summaries

RIPE-181 [2] and RPSL [1] data is represented externally as ASCII text. Objects consist of a set of attributes. Attributes are name/value pairs. A single attribute is represented as a single line with the name followed by a colon followed by whitespace characters (space, tab, or line continuation) and followed by the value. Within a value all whitespace is equivalent to a single space. Line continuation is supported by a backslash at the end of a line or the following line beginning with whitespace. When transferred externally attributes are generally broken into shorter lines using line continuation though this is not a requirement. An object is externally represented as a series of attributes. Objects are separated by blank lines.

As an example, there are about 80 attribute types in the current RIPE schema and about 15 object types. Some of the attributes are mandatory in certain objects. Some attributes may appear multiple times. One or more attributes may form a key. Some attributes or sets of attributes may be required to be unique. Some of the attributes may reference a key field in an object type and may be required to be a valid reference. Some attributes may be used in inverse lookups.

A review of the entire RIPE or RPSL schema would be too lengthy to include here. Only the differences in the schema are described.

Interactions with the registry either use a legacy format or are encapsulated using sets of name and value pairs that are formated like RPSL objects. These are not part of RPSL and are referred to as “meta-objects”. The meta-objects serve mostly as delimiters to the transactions and to carry information about the type of operation.

7.1 Transaction Submit and Confirm

The de-facto method for submitting database changes has been via email. This method should be supported by an external application for backwards compatibility only. Merit has added the pgp-from authentication method to the RADB (replaced by “pgpkey” in [4]), where the mail headers are essentially ignored and the body of the mail message must be PGP signed. For backwards compatibility objects submitted in an email message, even if signed as a group, should be treated as separate transactions as they are today. RFC–822 encapsulated messages should default to a confirmation of type “LEGACY”.

The meta-objects “transaction-submit-begin” and “transaction-submit-end” delimit a transaction. A transaction is handled as an atomic operation. If any part of the transaction fails none of the changes take effect. For this reason a transaction can only operate on a single database.

A socket connection is used to request queries or submit transactions. An email interface may be provided by an external program that connects to the socket. A socket connection must use the “transaction-submit-begin” and “transaction-submit-end” delimiters but can request a legacy style confirmation. Use of the email interface is discouraged and the email interface will eventually be deprecated. Multiple transactions may be sent prior to the response for any single transaction. Transactions may not complete in the order sent.

The “transaction-submit-begin” meta-object may contain the following attributes.

transaction-submit-begin This attribute is mandatory and single. The value of the attribute contains name of the database and an identifier that must be unique over the course of the socket connection.

response-auth-type See Section 7.6.

date-time-stamp See Section 7.6.

transaction-confirm-type This attribute is optional and single. A confirmation type keyword must be provided. Keywords are “none”, “legacy”, “normal”, “commit”. The confirmation type can be followed by the option “verbose”.

The “transaction-submit-end” meta-object consists of a single attribute by the same name. It must contain the same database name and identifier as the corresponding “transaction-submit-begin” attribute.

Unless the confirmation type is “none” a confirmation is sent. If the confirmation type is “legacy”, then an email message of the form currently sent by the RIPE database code will be returned on the socket (suitable for submission to the sendmail program).

A “normal” confirmation does not require completion of the commit protocol. A “commit” confirmation does. A “verbose” confirmation may contains additional detail.

A transaction confirmation is returned as a “transaction-confirm” meta-object. The “transaction-confirm” meta-object may have the following attributes.

transaction-confirm This attribute is mandatory and single. It contains the database name and identifier associated with the transaction.

confirmed-operation This attribute is optional and multiple. It contains one of the keywords “add”, “delete” or “modify” followed by the object type and key fields of the object operated on.

commit-status See Section 7.6.

date-time-stamp See Section 7.6.

7.2 Transaction Commit

The commit protocol consists of two steps.

1. commit request
2. commit completion

The “commit request” consists of a set of delimiters around a single transaction that has yet to be committed. The delimiters are the “mirror-request-begin” meta-object and “mirror-request-end” meta-object. The “mirror-request-begin” meta-object may contain the following attributes.

mirror-request-begin This attribute is mandatory and single. It contains the database name and sequence number of the transaction about to be committed.

date-time-stamp See Section 7.6.

The “mirror-request-end” meta-object consists of a single attribute of the same name containing the same database name and sequence number provided by the corresponding “mirror-request-begin”.

The “commit-completion” meta-object is sent in response to a “commit request”. Prior to attempting completion the remote database may have some catching up to do to reach the requested sequence number. If so, the remote database will send a “transaction-request” (Section 7.4) to bring its database copy to the sequence number below the transaction being committed.

The “commit-completion” meta-object may contain the following attributes.

commit-completion This attribute is mandatory and single. It contains the same name sequence number provided by the corresponding “mirror-request-begin”.

date-time-stamp See Section 7.6.

commit-status See Section 7.6.

7.3 Database Snapshot

A database snapshot provides a complete copy of a database. It is intended only for repository initialization and and disaster recovery.

A database snapshot request is represented by a "snapshot-request" meta-object. The "snapshot-request" meta-object may contain the the following attributes.

snapshot-request This attribute is mandatory and single. It contains the database name of the database being requested.

response-auth-type See Section 7.6.

A database snapshot is returned. The database snapshot is delimited by a "snapshot-begin" and "snapshot-end" meta-object. The "snapshot-begin" meta-object may contain the following attributes.

snapshot-begin This attribute is mandatory and single. It contains the database name and sequence number of the database snapshot being returned.

transfer-method This attribute is optional and single. It contains one or more of the following keywords "gzip", "uuencode", "base50", "radix64". A byte count may follow.

date-time-stamp See Section 7.6.

The "snapshot-end" meta-object contains a single attribute by the same name containing the same database name and sequence number provided in the corresponding "snapshot-begin".

The body of the snapshot is the set of objects in the form normally exchanged as printed RPSL objects. Each object is separated by a blank line. No signatures are provided except the signature on entire database snapshot from the repository providing the snapshot. The entire body may be compressed and then ASCII encoded. If the body is compressed but not ASCII encoded the last word of the transfer-method attribute value should be a byte count allowing a binary transfer to be used. If a binary transfer is used, the "snapshot-begin" meta-object block should be followed by exactly one empty line, then the binary data, then a trailing pair of newline characters (two linefeeds, not linefeed and carriage return).

7.4 Redistribution of Transactions

There are two ways to track database changes. Both involve making a unicast connection. In one case a request is made for unicast mirroring of a specific repository (see Section 7.4.2). The less preferred method is to poll by requesting a transaction sequence (see Section 7.4.1). To get updated to the current state of the database a request can be made with the end sequence number set to the special value "last".

7.4.1 Polling for Specific Transaction Sequences

A transaction sequence can be requested by sending a “transaction-request” meta-object. A “transaction-request” meta-object may contain the following attributes.

transaction-request This attribute is mandatory and single. It contains the database name and a sequence list. The sequence list is two sequence numbers separated by a dash. The keyword “last” may be used in place of a number to indicate the last sequence number available. The sequence list “last-last” can be requested to simply get the last sequence number in an empty transaction sequence.

response-auth-type See Section 7.6.

7.4.2 Unicast Flooding Redistribution

A unicast mirror request is represented by a “unicast-mirror-request” meta-object which may contain the following attributes.

unicast-mirror-request This attribute is mandatory and single. It contains the database and next sequence number needed. This may optionally followed by an maximum update frequency in seconds, and an idle timeout in minutes. If there are no new transactions and no other activity on the socket before the idle period the connection is dropped.

response-auth-type See Section 7.6.

A unicast mirror request is answered by a unicast mirror response. This is represented in a “unicast-mirror-response” meta-object, which may contain one of the following attributes.

unicast-mirror-response This attribute is mandatory and single. It contains the name of the database.

unicast-mirror-status This attribute is optional and single. It may contain the word “rejected”.

unicast-referral This attribute is optional and multiple. It contains the name of a that are known or likely to provide a unicast feed of the requested database.

A repository may reject a request for a unicast feed for a variety of reasons. Offering an alternative place to look may be helpful to the requester. The alternative could be adjacent repositories providing a feed.

A unicast feed may be canceled without disrupting other use of the socket. See Section 7.4.4.

7.4.3 Transaction Sequence Format

A transaction sequence may contain zero or more transactions. A transaction sequence is delimited by a “transaction-begin” and “transaction-end” meta-object. The complete transaction sequence should be treated as an atomic operation.

The following attributes may be contained in a "transaction-begin" meta-object.

transaction-begin This attribute is mandatory and single. It contains the database name and the next available sequence number. This sequence number will be used for the first transaction in the sequence if the sequence in not empty.

database-sequence This attribute is optional and multiple. It contains the database name and sequence number of a database that is needed for authorization of one or more transaction in the sequence.

transfer-method This attribute is optional and single. It contains one or more of the following keywords "gzip", "uuencode", "base50", "radix64".

date-time-stamp See Section 7.6.

Transactions are encapsulated by embedding the initial transaction submission intact including any authentication.

A "transaction-end" attribute contains the the database name and the next available sequence number. The sequence number in the "transaction-begin" plus the number of transaction submission must be greater than or equal to the next available sequence number included in the "transaction-end" attribute. The next available sequence number can be greater if some RPSL object types are not redistributed by the originating registry for privacy reasons. This may be the case for some registries and objects containing contact information such as "person" and "role" objects.

Transactions can also be rescinded. The operation of rescinding a transaction is represented by a "transaction-rescind" meta-object which itself consumes one sequence number. The "transaction-rescind" meta-object contains one attribute by the same name. The value of the attribute is the sequence number of the first transaction being rescinded and the sequence number following the last transaction rescinded.

7.4.4 Canceling Operations

A request can be made to cancel most operations. The most common would be to cancel a "query" which is returning too much information or cancel a long running operation like a "unicast-mirror-request".

A "cancel-operation" meta object contains only an attribute by the same name. The attribute contains the operation type represented by the key attribute name in the request without the trailing "-begin". The remainder of the "cancel-operation" attribute contains the key field of the request.

When an operation is canceled a "cancel-confirm" meta-object is returned. Any response in progress is ended by the "cancel-confirm" and a "-end" meta-object should not be expected. The "cancel-confirm" attribute contains the same operation type and key field as the corresponding "cancel-operation".

7.4.5 Heartbeat Processing

If there are no new transactions during the heartbeat period a "heartbeat" meta-object is sent. The heartbeat meta-object allows a distinction between lost connectivity to a repository and inactivity. The heartbeat meta-object contains the last sequence number used and the date and time. This object is signed by the originating repository. The date and time prevent replay attacks.

If a repository or mirror receives a heartbeat meta-object, it may need to flood that meta-object. If there is a queue of outstanding transactions still to be process for that repository, then the heartbeat meta-object can be queued behind those transactions.

After a transaction for a remote repository is processed, that repository is know to be current up to the date and time of that last transaction. That date and time must be stored as the most recent update from that repository. If a heartbeat object is queued or arrives later, then the stored date and time is compared to the date and time in the heartbeat meta-object. If the times differ by at least the heartbeat-rate attribute in the repository object (see Section 5.1), then the heartbeat meta-object is flooded. The date and time in the heartbeat object replace date and time stored for the repository.

If the date and time in the heartbeat meta-object is older, then the heartbeat meta-object is disposed of with no action taken. If the date and time is the same as the already known data and time or differs only slightly, then flooding is terminated.

The date and time of last update from a repository "A" is used with the "expire" time in the repository object for "A" to determine when a transaction in another repository "B" with authorization dependencies in repository "A" must be held due to loss of connectivity to repository "A". If the transaction in repository "B" is more recent than the last update to repository "A" plus the expire time for repository "A", then the authorization check cannot be made. A transaction in "A" or a heartbeat meta-object will allow the transaction in "B" to proceed with the authorization check.

The "heartbeat" meta-object contains the following attributes:

heartbeat This attribute is mandatory and single. It contains the database name and the next available sequence number.

date-time-stamp See Section 7.6.

7.5 Authenticating Operations

PGP normally encapsulates text by starting with a line containing "-----BEGIN PGP SIGNED MESSAGE-----" and a blank line and then ending with the signature block. The signature block consists of a blank line, then a line with "-----BEGIN PGP SIGNATURE-----", then a block containing the ASCII radix-64 signature and ending with a line containing "-----END PGP SIGNATURE-----". This encapsulation can be recognized as a meta-objects allowing pgp to be used in normal pipe plumbing using the PGPPASSFD feature to provide a pass phrase.

Alternately, the the PGP delimiters can be replaced with meta-objects. The "signature" meta-objects represent detached signatures. Where multiple signatures are present, these should be signatures over the original contents, not signatures nested over other detached signatures. This allows signatures to be checked in any order.

The meta-objects “signed-block-begin” and “signed-block-end” are used. The attributes “signed-block-begin” and “signed-block-end” are simply containers. The value of these attributes should be the repository name or the fully qualified signing identity followed by a unique string. A fully qualified signing identity is a repository name, the delimiter “::”, and NIC handle or key-cert attribute. The unique string can be a date and time stamp with sufficient time granularity to insure uniqueness (for example, time in microseconds). The value of the “signed-block-begin” and corresponding “signed-block-end” must match.

Between the “signed-block-begin” and “signed-block-end” there may be one or more “signature-block” meta-objects. Multiple identities may sign an object. The same identity may sign using more than one authentication method (for example, using different encryption methods, some of which are not available worldwide).

The “signature-block” meta-objects contains a “signature-block” attribute which contain only the authentication method name. The interpretation of any additional attributes depend on the authentication method.

In the case of PGP, the attributes of the “signature-block” meta-object are the attributes contained in a detached signature file produced by PGP (for example, the “Version”). A “signature” attribute contains the multiline ASCII armored signature.

Any meta-object can be signed, including large sequences of meta-objects and meta-objects such as transaction sequences. Meta-objects may be signed by more than one method. If more than one method is used to sign an object, then all signatures supported by the repository checking authorization must be valid.

Note that the RPSL objects themselves are not signed. What is signed by the submitter is the body of the transaction. In the case of an email submission this is a subtle distinction. When exchanging transactions among registries, the meta-objects that make up requests are signed by one registry and the transaction sequences returned are signed by the other registry. Within the transaction sequences there may be transactions signed by the submitter. There is additional meta-information within the transaction sequences that falls outside of the submitter’s signature-block but within the registry’s signature-block.

Transactions must remain intact, including the signatures, even if an authentication method provided by the submitter is not used by a repository handling the message. It is also essential to retain the transaction sequence signatures and it may be useful to add an addition signature when encapsulating a received transaction sequence for reflooding.

Normally repositories will sign transactions between repositories. When unwrapping the authentication encapsulations, the identities of the signatures must be retained to establish authorization. If at any point the signature of a trusted repository is encountered, no further authorization or authentication is needed and any further nested “signed-block-begin” and “signed-block-end” can be ignored.

7.6 Attributes Common to Meta-Objects

A number of attributes are used by numerous meta-objects. They are described here rather than repeating their descriptions elsewhere.

date-time-stamp This attribute is mandatory and single except were it is noted as being optional. The date and time are given in the form "YYYYMMDD HHMMSS" with an optional numeric timezone represented as "[+-]H". The upper case letters are digits corresponding to the year, month, day of month, hour, minute, second, and hours before or after UTC.

response-auth-type This attribute is optional and multiple. The remainder of the line specifies an authentication type that would be acceptable in the response. This is used to request a response cryptographically signed by the repository.

commit-status This attribute is mandatory and single. It contains one of the keywords "timeout", "error", or "commit". The "error" keyword may be followed by an optional text string.

Error messages included with a commit-status should be clearly worded. Temporary errors such as failure to commit local resources (for example, out of local disk space) should not return an error. The request should be retained and the socket kept open until the resource becomes available. If either side breaks the connection, the transaction must be retried.

A Examples

RPSL provides an external representation of RPSL objects and attributes. An attribute is a name/value pair. RPSL is line oriented. Line continuation is supported, however most attributes fit on a single line. The attribute name is followed by a colon, then any amount of whitespace, then the attribute value. An example of the ASCII representation of an RPSL attribute is the following:

```
route:     140.222.0.0/16
```

An RPSL object is a set of attributes. Objects are separated from each other by one or more blank lines. An example of a complete RPSL object follows:

```
route:         140.222.0.0/16
descr:         ANS Communications
origin:        AS1673
member-of:     RS-ANSOSPFAGGREGATE
mnt-by:        ANS
changed:       tck@ans.net 19980115
source:        ANS
```

A.1 Initial Object Submission and Redistribution

Figure 1 outlines the steps involved in submitting an object and the initial redistribution from the authoritative registry to its flooding peers.

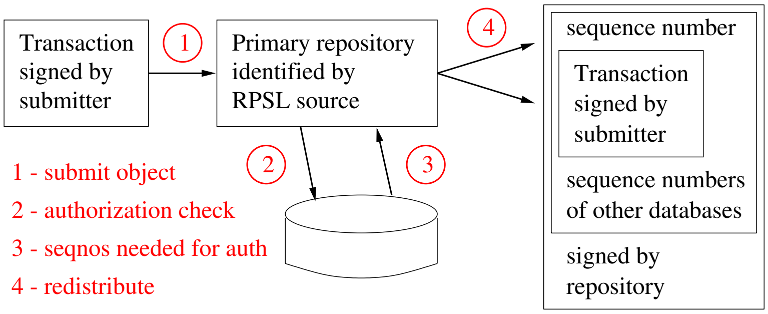

Figure 1: Initial Object Submission and Redistribution

If the authorization check requires objects from other repositories, then the sequence numbers of the local copies of those databases is required for mirrors to recheck the authorization.

To simply resubmit the object from the prior example, the submitter or a client application program acting on the submitter's behalf must submit a transaction. The legacy method was to send PGP signed email. The preferred method is for an interactive program to encapsulate a request between "transaction-submit-begin" and "transaction-submit-end" meta-objects and encapsulate that as a signed block as in the following example:

```
signed-block-begin:        ANS::PGPKEY-3EBC5F29 1

signature-block:  PGP
version: PGPfreeware 5.0i for non-commercial use
signature:
  iQCVAwUANsrwkP/OhQ1cphB9AQFOvwP/Ts8qn3FRRLQQHKmQGzy2IxOTiF0QXB4U
  Xzb3gEvfeg8NWhAI32zBw/D6FjkEw7P6wDFDeok52A1SA/xdP5wYE8heWQmMJQLX
  Avf8W49d3CF3qzh59UCOALtA5BjI3r37ubzTf3mgtw+ONqVJ5+lB5upWbqKN9zqv
  PGBIEN3/NlM=
  =c93c

transaction-submit-begin:  ANS 1
response-auth-type:        PGP
date-time-stamp:           19990401 103000
transaction-confirm-type:  normal

route:          140.222.0.0/16
descr:          ANS Communications
origin:         AS1673
member-of:      RS-ANSOSPFAGGREGATE
mnt-by:         ANS
changed:        curtis@ans.net 19990401
source:         ANS
```

```
transaction-submit-end:    ANS 1

signed-block-end:          ANS::PGPKEY-3EBC5F29 1
```

The signature covers the "transaction-submit-begin" to the "transaction-submit-end" attributes above. If multiple signatures are needed, it would be quite easy to email this block and ask the other party to add a signature-block and return or submit the transaction. Because of delay in obtaining multiple signatures the accuracy of the "date-time-stamp" cannot be strictly enforced. Enforcing accuracy to within the "expire" time of the database might be a reasonable compromise. The tradeoff is between convenience, allowing a longer time to obtain multiple signatures, and increased time of exposure to replay attack.

The ANS repository would look at its local database and make authorization checks. If the authorization passes, then the sequence number of any other database needed for the authorization is obtained.

Having passed the authorization check the transaction is given a sequence number and stored in the local transaction log and is then flooded. The meta-object flooded to another database would be signed by the repository and would be of the following form:

```
signed-block-begin:        ANS::PGPKEY-DEADBEEF 123

signature-block:  PGP
version: PGPfreeware 5.0i for non-commercial use
signature:
  iQCVAwUANsrwkP/OhQ1cphB9AQFOvwP/Ts8qn3FRRLQQHKmQGzy2IxOTiFOQXB4U
  Xzb3gEvfeg8NWhAI32zBw/D6FjkEw7P6wDFDeok52A1SA/xdP5wYE8heWQmMJQLX
  Avf8W49d3CF3qzh59UC0ALtA5BjI3r37ubzTf3mgtw+ONqVJ5+lB5upWbqKN9zqv
  PGBIEN3/NlM=
  =c93c

transaction-begin:         ANS 666

signed-block-begin:        ANS::PGPKEY-3EBC5F29 1

signature-block:  PGP
version: PGPfreeware 5.0i for non-commercial use
signature:
  iQCVAwUANsrwkP/OhQ1cphB9AQFOvwP/Ts8qn3FRRLQQHKmQGzy2IxOTiFOQXB4U
  Xzb3gEvfeg8NWhAI32zBw/D6FjkEw7P6wDFDeok52A1SA/xdP5wYE8heWQmMJQLX
  Avf8W49d3CF3qzh59UC0ALtA5BjI3r37ubzTf3mgtw+ONqVJ5+lB5upWbqKN9zqv
  PGBIEN3/NlM=
  =c93c

transaction-submit-begin:  ANS 1
response-auth-type:        PGP
date-time-stamp:           19990401 103000
transaction-confirm-type:  normal
```

```
route:         140.222.0.0/16
descr:         ANS Communications
origin:        AS1673
member-of:     RS-ANSOSPFAGGREGATE
mnt-by:        ANS
changed:       curtis@ans.net 19990401
source:        ANS

transaction-submit-end:    ANS 1

signed-block-end:          ANS::PGPKEY-3EBC5F29 1

transaction-end:           ANS 666

signed-block-end:          ANS::PGPKEY-DEADBEEF 123
```

Note that the signature above is a detached signature for another file and is illustrative only. The repository signature covers the "transaction-begin" to the "transaction-end" attribute.

If this operation was successful, then a confirmation would be returned. The confirmation would be of the form:

```
transaction-confirm:  ANS 1
confirmed-operation:  change route 140.222.0.0/16
commit-status:        commit
date-time-stamp:      19990401 103001
```

Note that this transaction could be replayed. The only protection is the "date-time-stamp" attribute on the transaction. Figure 2 provides better protection against replay. The repository returns the submission with a sequence number to be resigned by the submitter.

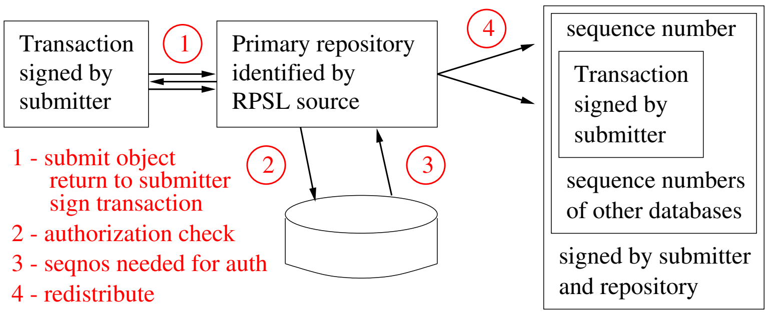

Figure 2: Alternate Initial Object Submission

In Figure 2 the submitter is protected against the possibility of the repository replaying a submission later or a snooper replaying the submission. This method is not specified in this document for two reasons. One is that if the "date-time-stamp" attribute is beyond the "expire" period for the database, the authorization would be rejected. The other is that the sequence number would have to be reserved and a failure to reply with a resigned submission would halt database processing for some timeout period.

An indefinite exposure to replay is present if the PGP signed email method of submission is used unless accuracy of the date stamp in the "changed" attribute is required. This would limit the replay window to approximately a day.

A.2 Transaction Redistribution

The last step in Figure 1 was redistributing the submitter's transaction through flooding (or later through polling). Figure 3 illustrates the further redistribution of the transaction.

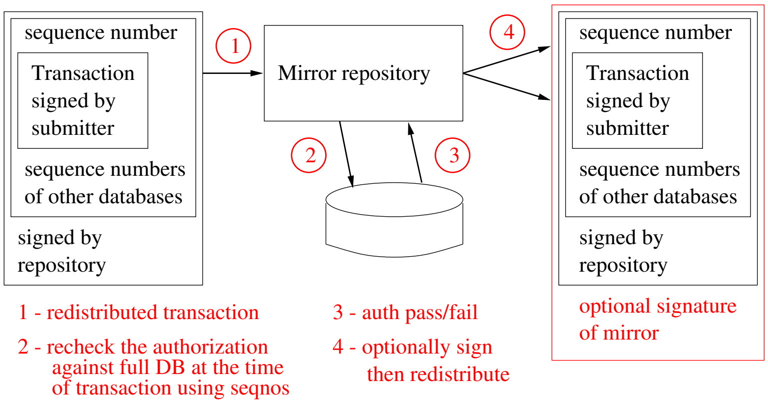

Figure 3: Further Transaction Redistribution

If the authorization check was repeated, the mirror may optionally add a signature before passing the transaction any further. Generally a repository should not pass a transaction on unless the authorization check has passed. Since the outside meta-object is a signed-block, a "signature-block" can be added within that block. The previous signatures should not be signed.

Figure 4 illustrates the special case referred to as a "lightweight mirror". This is specifically intended for routers.

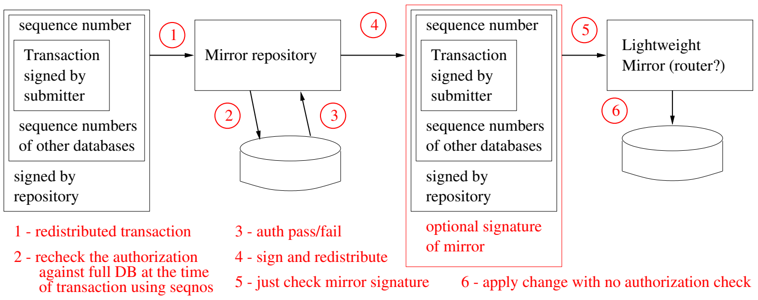

Figure 4: Redistribution to Lightweight Mirrors

The lightweight mirror must trust the mirror from which it gets a feed. This is a safe assumption if the two are under the same administration (the mirror providing the feed is a host owned by the same ISP who owns the routers). The lightweight mirror simply checks the signature of the adjacent repository to insure data integrity.

A.3 Optional Commit and Confirm

In Figure 4 the full commit and confirm cycle is shown. Included is the extra initial submission exchange for added protection against replay previous shown in Figure 2.

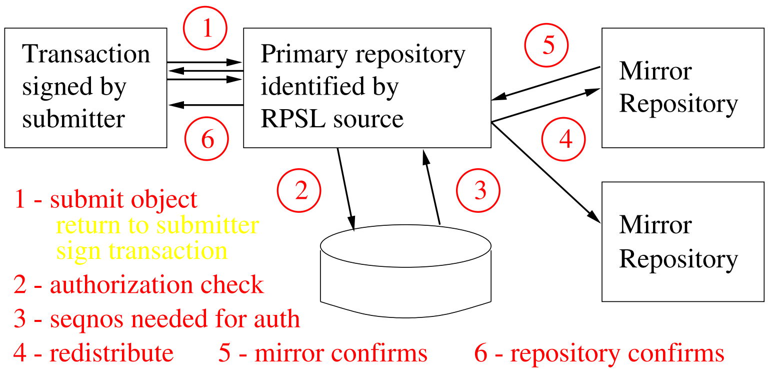

Figure 5: Optional Commit and Confirm

Redistribution differs in that one or more repositories will receive the transaction encapsulated in a commit request. The commit request would be of the following form:

```
signed-block-begin:        ANS::PGPKEY-DEADBEEF 124

signature-block:  PGP
version: PGPfreeware 5.0i for non-commercial use
signature:
  iQCVAwUANsrwkP/0hQ1cphB9AQF0vwP/Ts8qn3FRRLQQHKmQGzy2Ix0TiF0QXB4U
  Xzb3gEvfeg8NWhAI32zBw/D6FjkEw7P6wDFDeok52A1SA/xdP5wYE8heWQmMJQLX
  Avf8W49d3CF3qzh59UC0ALtA5BjI3r37ubzTf3mgtw+0NqVJ5+1B5upWbqKN9zqv
  PGBIEN3/N1M=
  =c93c

mirror-request-begin:      ANS 666
date-time-stamp:           19990401 103000

signed-block-begin:        ANS::PGPKEY-DEADBEEF 123

signature-block:  PGP
version: PGPfreeware 5.0i for non-commercial use
signature:
  iQCVAwUANsrwkP/0hQ1cphB9AQF0vwP/Ts8qn3FRRLQQHKmQGzy2Ix0TiF0QXB4U
  Xzb3gEvfeg8NWhAI32zBw/D6FjkEw7P6wDFDeok52A1SA/xdP5wYE8heWQmMJQLX
  Avf8W49d3CF3qzh59UC0ALtA5BjI3r37ubzTf3mgtw+0NqVJ5+1B5upWbqKN9zqv
  PGBIEN3/N1M=
  =c93c

transaction-begin:         ANS 666
```

```
signed-block-begin:        ANS::PGPKEY-3EBC5F29 1

signature-block:  PGP
version: PGPfreeware 5.0i for non-commercial use
signature:
  iQCVAwUANsrwkP/OhQ1cphB9AQFOvwP/Ts8qn3FRRLQQHKmQGzy2IxOTiF0QXB4U
  Xzb3gEvfeg8NWhAI32zBw/D6FjkEw7P6wDFDeok52A1SA/xdP5wYE8heWQmMJQLX
  Avf8W49d3CF3qzh59UC0ALtA5BjI3r37ubzTf3mgtw+ONqVJ5+lB5upWbqKN9zqv
  PGBIEN3/NlM=
  =c93c

transaction-submit-begin:  ANS 1
response-auth-type:        PGP
date-time-stamp:           19990401 103000
transaction-confirm-type:  normal

route:         140.222.0.0/16
descr:         ANS Communications
origin:        AS1673
member-of:     RS-ANSOSPFAGGREGATE
mnt-by:        ANS
changed:       curtis@ans.net 19990401
source:        ANS

transaction-submit-end:    ANS 1

signed-block-end:          ANS::PGPKEY-3EBC5F29 1

transaction-end:           ANS 666

signed-block-end:          ANS::PGPKEY-DEADBEEF 123

mirror-request-end:        ANS 666

signed-block-end:          ANS::PGPKEY-DEADBEEF 124
```

The confirmation can be returned to the submitter when one of the repositories that was sent a commit request returns a commit completion. This would be of the form:

```
commit-completion:    ANS 666
commit-status:        commit
date-time-stamp:      19990401 103001
```

In the event of a disk crash, the repository has already successfully flooded the transaction before sending the confirm back to the submitter. If a mirror is under the same administration, the repository can recover from disk and roll forward the transactions from the mirror before resuming operation.

B Technical Discussion

B.1 Server Processing

This document does not mandate any particular software design, programming language choice, or underlying database or underlying operating system. Examples are given solely for illustrative purposes.

B.1.1 getting connected

There are two primary methods of communicating with a repository server. E-mail can be sent to the server. This method may be deprecated but at least needs to be supported during transition. The second method is preferred, connect directly to a TCP socket.

Traditionally the whois service is supported for simple queries. It might be wise to retain the whois port connection solely for simple queries and use a second port not in the reserved number space for all other operations including queries except those queries using the whois unstructured single line query format.

There are two styles of handling connection initiation is the dedicated daemon, in the style of BSD sendmail, or launching through a general purpose daemon such as BSD inetd. E-mail is normally handled sequentially and can be handled by a front end program which will make the connection to a socket in the process as acting as a mail delivery agent.

B.1.2 rolling transaction logs forward and back

There is a need to be able to easily look back at previous states of any database in order to repeat authorization checks at the time of a transaction. This is difficult to do with the RIPE database implementation, which uses a sequentially written ASCII file and a set of Berkeley DB maintained index files for traversal. At the very minimum, the way in which deletes or replacements are implemented would need to be altered.

In order to easily support a view back at prior versions of objects, the sequence number of the transaction at which each object was entered would need to be kept with the object. A pointer would be needed back to the previous state of the object. A deletion would need to be implemented as a new object with a deleted attribute, replacing the previous version of the object but retaining a pointer back to it.

A separate transaction log needs to be maintained. Beyond some age, the older versions of objects and the the older transaction log entries can be removed although it is probably wise to archive them.

B.1.3 committing or disposing of transactions

The ability to commit large transaction, or reject them as a whole poses problems for simplistic database designs. This form of commit operation can be supported quite easily using memory mapped files. The changes can be made in virtual memory only and then either committed or disposed of.

B.1.4 dealing with concurrency

Multiple connections may be active. In addition, a single connection may have multiple outstanding operations. It makes sense to have a single process or thread coordinate the responses for a given connection and have multiple processes or threads each tending to a single operation. The operations may complete in random order.

Locking on reads is not essential. Locking before write access is essential. The simplest approach to locking is to lock at the database granularity or at the database and object type granularity. Finer locking granularity can also be implemented. Because there are multiple databases, deadlock avoidance must be considered. The usual deadlock avoidance mechanism is to acquire all necessary locks in a single operation or acquire locks in a prescribed order.

B.2 Repository Mirroring for Redundancy

There are numerous reasons why the operator of a repository might mirror their own repository. Possibly the most obvious are redundancy and the relative ease of disaster recovery. Another reason might be the widespread use of a small number of implementations (but more than one) and the desire to insure that the major repository software releases will accept a transaction before fully committing to the transaction. This may avoid the need to rescind transactions in the face of a newly discovered bug.

The operation of a repository mirror used for redundancy is quite straightforward. The transactions of the primary repository host can be immediately fed to the redundant repository host. For tighter assurances that false positive confirmations will be sent, as a matter of policy the primary repository host can require commit confirmation before making a transaction sequence publicly available.

There are many ways in which the integrity of local data can be assured regardless of a local crash in the midst of transaction disk writes. For example, transactions can be implemented as memory mapped file operations, with disk synchronization used as the local commit mechanism, and disposal of memory copies of pages used to handle commit failures. The old pages can be written to a separate file, the new pages written into the database. The transaction can be logged and old pages file can then be removed. In the event of a crash, the existence of a old pages file and the lack of a record of the transaction completing would trigger a transaction roll back by writing the old pages back to the database file.

The primary repository host can still sustain severe damage such as a disk crash. If the primary repository host becomes corrupted, the use of a mirror repository host provides a backup and can provide a rapid recovery from disaster by simply reversing roles.

If a mirror is set up using a different software implementation with commit mirror confirmation required, any transaction which fails due a software bug will be deferred indefinitely allowing other transactions to proceed rather than halting the remote processing of all transactions until the bug is fixed everywhere or the offending transaction is rescinded.

B.3 Trust Relationships

If all repositories trust each other then there is never a need to repeat authorization checks. This enables a convenient interim step for deployment prior to the completion of software supporting that capability. The opposite case is where no repository trusts any other repository. In this case, all repositories must roll forward transactions gradually, checking the authorization of each remote transaction.

It is likely that repositories will trust a subset of other repositories. This trust can reduce the amount of processing a repository required to maintain mirror images of the full set of data. For example, a subset of repositories might be trustworthy in that they take reasonable security measures, the organizations themselves have the integrity not to alter data, and these repositories trust only a limited set of similar repositories. If any one of these repositories receives a transaction sequence and repeats the authorization checks, other major repositories which trusts that repository need not repeat the checks. In addition, trust need not be mutual to reap some benefit in reduced processing.

As a transaction sequence is passed from repository to repository each repository signs the transaction sequence before forwarding it. If a receiving repository finds that any trusted repository has signed the transaction sequence it can be considered authorized since the trusted repository either trusted a preceding repository or repeated the authorization checks.

B.4 A Router as a Minimal Mirror

A router could serve as a minimal repository mirror. The following simplifications can be made.

1. No support for repeating authorization checks or transaction authentication checks need be coded in the router.
2. The router must be adjacent only to trusted mirrors, generally operated by the same organization.
3. The router would only check the authentication of the adjacent repository mirrors.
4. No support for transaction submission or query need be coded in the router. No commit support is needed.
5. The router can dispose of any object types or attributes not needed for configuration of route filters.

The need to update router configurations could be significantly reduced if the router were capable of acting as a limited repository mirror.

A significant amount of non-volatile storage would be needed. There are currently an estimated 100 transactions per day. If storage were flash memory with a limited number of writes, or if there were some other reason to avoid writing to flash, the router could only update the non-volatile copy every few days. A transaction sequence request can be made to get an update in the event of a crash, returning only a few hundred updates after losing a few days of deferred writes. The routers can still take a frequent or continuous feed of transactions.

Alternately, router filters can be reconfigured periodically as they are today.

B.5 Dealing with Errors

If verification of an authorization check fails, the entire sequence must be rejected and no further advancement of the repository can occur until the originating repository corrects the problem. If the problem is due to a software bug, the offending transaction can be removed manually once the problem is corrected. If a software bug exists in the receiving software, then the transaction sequence is stalled until the bug is corrected. It is better for software to error on the side of denying a transaction than acceptance, since an error on the side of acceptance will require rescinding transactions and rolling forward only those that were valid.

C Deployment Considerations

This section described deployment considerations. The intention is to raise issues rather than to provide a deployment plan.

This document calls for a transaction exchange mechanism similar to but not identical to the existing “near real time mirroring” supported by the code base widely used by the routing registries. As an initial step, the transaction exchange can be implemented without the commit protocol or the ability to recheck transaction authorization. This is a fairly minimal step from the existing capabilities.

The transition can be staged as follows:

1. Modify the format of “near real time mirroring” transaction exchange to conform to the specifications of this document.
2. Implement commit protocol and confirmation support.
3. Implement remote recheck of authorization. Prior to this step all repositories must be trusted.
4. Allow further decentralization of the repositories.

References

[1] C. Alaettinoglu, T. Bates, E. Gerich, D. Karrenberg, D. Meyer, M. Terpstra, and C. Villamizar. Routing policy specification language (rpsl). Technical Report RFC 2280, Internet Engineering Task Force, 1998. ftp://ftp.isi.edu/in-notes/rfc2280.txt.

[2] T. Bates, E. Gerich, L. Joncheray, J-M. Jouanigot, D. Karrenberg, M. Terpstra, and J. Yu. Representation of ip routing policies in a routing registry (ripe-81++). Technical Report RFC 1786, Internet Engineering Task Force, 1995. ftp://ftp.isi.edu/in-notes/rfc1786.txt.

[3] David Meyer, Curtis Villamizar, Cengiz Alaettinoglu, S. Murphy, and Carol Orange. Routing policy system security. Internet Draft (Work in Progress) draft-ietf-rps-auth-01, Internet Engineering Task Force, 5 1998. ftp://ftp.isi.edu/internet-drafts/draft-ietf-rps-auth-01.txt.

[4] Janos Zsako. Pgp authentication for ripe database updates. Internet Draft (Work in Progress) draft-ietf-rps-dbsec-pgp-authent-00, Internet Engineering Task Force, 11 1998. ftp://ftp.isi.edu/internet-drafts/draft-ietf-rps-dbsec-pgp-authent-00.txt.

Security Considerations

@@ later for this.

Author's Addresses

Curtis Villamizar
UUNET Network Architecture Group
<curtis@ans.net>

Cengiz Alaettinoglu
ISI
<cengiz@ISI.EDU>

Ramesh Govindan
ISI
<govindan@ISI.EDU>

David M. Meyer
Cisco
<dmm@cisco.com>

Full Copyright Statement