Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 29, 2013

A. Atlas
Juniper Networks
S. Hares
Hickory Hill Consulting
J. Halpern
Ericsson

February 25, 2013

A Policy Framework for the Interface to the Routing System
draft-atlas-i2rs-policy-framework-01

Abstract

A key aspect of the Interface to the Routing System (I2RS) is what mechanisms it includes to carry policy information and to enable policy control. This applies both in the protocol itself and in the services associated with the different components of the routing system. Similarly, the data-models associated with the services must be capable of expressing the appropriate granularity for access and authorization-related policy. This document describes the policy framework for I2RS.

Table of Contents

1. Introduction

The Interface to the Routing System (I2RS) provides read and write access to the information and state that enable the routing components of routing elements. The I2RS is introduced and described in [I-D.atlas-irs-problem-statement] and [I-D.ward-irs-framework].

Policy helps provide filters and control on the access to information and state that is enabled by individual protocol interactions. A clear view of the policy features desirable at the I2RS is important to shape the architecture and requirements for the protocols and services of the I2RS. Policy can be explicitly defined or implicitly assumed in a system, and can be enforced by that system's rules and behavior. Since I2RS provides services to routing sub-systems that already have policy defined (implicitly or explicitly), it is important to consider the existing policy mechanisms and how an I2RS services should interact with them.

I2RS policy has four different aspects that need to be considered.

1. Policy related to the I2RS protocol interactions between different systems.

2. Policy related to the interaction between the I2RS Agent and the local system to which the I2RS Agent is providing an interface.

3. Service policy to support scope and influence restrictions and to preserve necessary policy associated with the related routing sub-system.

4. Policy that can be installed or read via a service's data-model that is associated with the related routing sub-system.

2. Terminology

The following terminology is used in this document.

agent or I2RS Agent: An I2RS agent provides the supported I2RS services to the local system's routing sub-systems. The I2RS agent understands the I2RS protocol and can be contacted by I2RS clients.

client or I2RS Client: A client speaks the I2RS protocol to communicate with I2RS Agents and uses the I2RS services to accomplish a task as instructed by the client's local application. An I2RS client can be seen as the part of an application that supports I2RS and could be a software library.

service or I2RS Service: For the purposes of I2RS, a service refers to a set of related state access functions together with the policies that control its usage. For instance, 'RIB service' could be an example of a service that gives access to state held in a device's RIB.

read scope: The set of information which the particular I2RS entity (agent or client) is authorized to read. This access includes the permission to see the existence of data and the ability to retrieve the value of that data. In the context of an interaction between a client and an agent, the effective read scope is restricted to the intersection of the read scopes of the two entities.

write scope: The set of field values which the particular I2RS entity (agent or client) is authorized to write (i.e. add, modify or delete). This access can restrict what fields can be modified or created, and what specific value sets and ranges can be installed. In the context of an interaction between a client and an agent, the effective write scope is restricted to the intersection of the write scopes of the two entities.

scope: When unspecified as either read scope or write scope, the term scope applies to both the read scope and write scope.

resources: A resource is an I2RS-specific use of memory, storage, or execution that a client may consume due to its I2RS operations. The amount of each such resource that a client may consume in the context of a particular agent can be constrained. Examples of such resources could include: the number of installed operations, number of operations that haven't reached their start-time, etc. These are not protocol-specific resources or network-specific resources.

role or security role: A security role specifies the scope, resources, precedences, etc. that a client or agent has.

identity: A client is associated with exactly one specific identity. State installed by a particular identity is owned by that identity; state ownership can not be transferred. It is possible for multiple communication channels to use the same identity; in that case, the assumption is that the associated client is coordinating such communication. Similarly, an agent is associated with a specific identity.

3. General I2RS Policy

I2RS needs its own implicit and explicit policy. This section articulates some of the those key concepts and policy decisions. The I2RS policy applies to interactions between the agent and clients and between the agent and the local system.

The agent's externally perceivable behavior and associated policy is a key aspect of I2RS that must be described. The client's behavior and functionality is specifically out-of-scope except where it needs to be described with respect to the agent's behavior and the I2RS protocol.

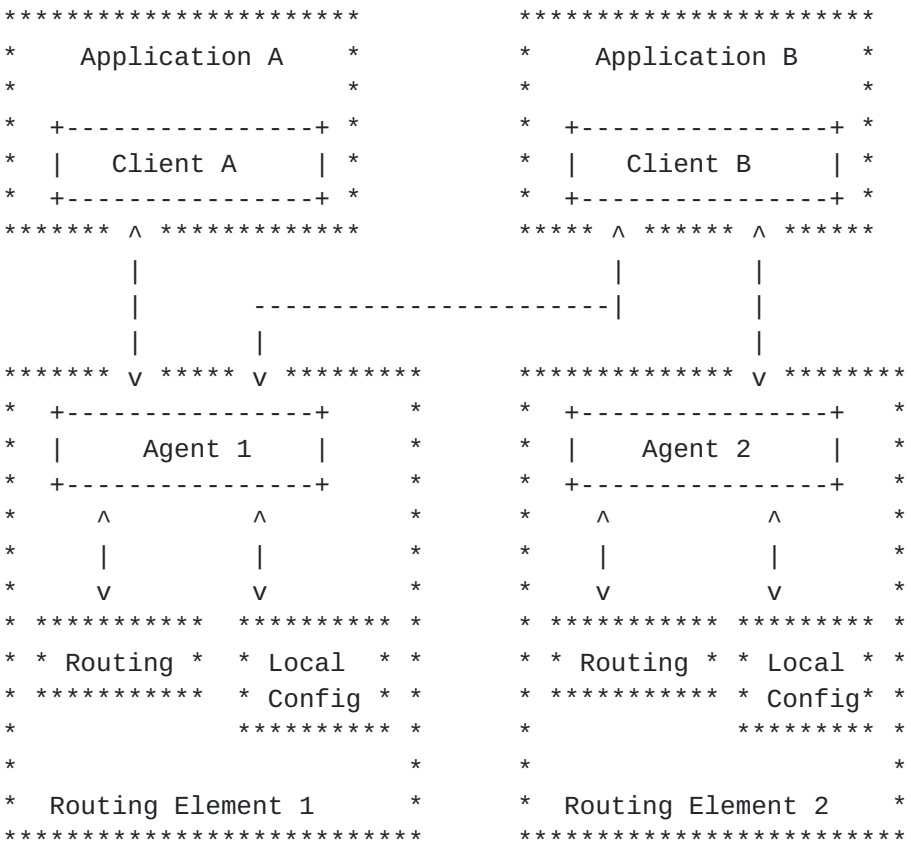

Figure 1: Architecture of clients and agents

As can be seen in Figure 1, a client can communicate with multiple agents. The application associated with a client may have multiple tasks it is accomplishing (separate functions, short-term versus longer-term, etc) and each such task may involve a set of agents which may or may not differ.

As can also be seen in Figure 1, an I2RS Agent may communicate with multiple clients. Each client may send the agent a variety of write operations. The set of write operations received by an agent may overlap and conflict. No simple protocol or policy mechanisms by an agent can completely avoid indirect interactions between different install operations. The functional partitioning between the different clients must be done to avoid undesirable indirect interactions.

3.1. Use-Case of Overlapping Interactions

An I2RS Agent can receive overlapping operations from multiple clients. An example is when there are two applications:

Client A: Special Flow Router: Client A is part of an application that explicitly routes particular special flows using policy-based routing (aka ACLs).

Client B: DDoS Detection and Mitigation: Client B is part of an application that looks for flows that appear to be part of a DDoS attack and explicitly routes them to mitigate the attack. Client B also uses policy-based routing (aka ACLs).

If Client B is told to explicitly route prefix X, because it looks suspicious, and Client A is also explicitly routing prefix X, then the I2RS Agent must determine what to do based upon policy. Even though intelligent functional partitioning has been done, this is an example where the I2RS agent must still make an arbitration decision. This document defines precedence as the policy mechanism by which the I2RS agent can be instructed what to do in such cases.

3.2. Policy between client and agent

Multiple clients can communicate with the same agent. The agent must have policies to manage the resulting complexity. Implicit policy includes the assumptions about communication between the client and agent. Explicit policy includes mechanisms to arbitrate between different clients, between operations of the same client, and to manage state owned by an client inside the agent.

Any easy way to look at the i2rs policies is that the policies answer who, what, and how.

Who: The first type of policies concern identity, secure roles, and security model for connecting. The same is true of any secured connect between two hosts where each host has an identity, a secured role in the communication and security model on who can connect.

What: The second set of policies look at secure model on what data can be examined, the scope of the read or writes, and the amount of resources an active i2rs agent can consume. A security model defines the barriers for the i2rs activity.

How: The third set of policies involve how the i2rs agent and i2rs client communication, and how they mitigate the natural contention of allowing an i2rs client to talk to multiple i2rs agents or an i2rs agent to communicate with multiple clients. For example, a single i2rs client connected to several i2rs agents (i2rs agent J and i2rs agent K) may learn of an interface overload (on i2rs agent J), and then want to reprioritize activities on i2rs agent K to find another data path. This is priority policy. In the same way, the two i2rs Clients (A and B) may try to install a RIB route on i2rs agent K. If there are overlapping actions, policy needs to determine who wins.

3.2.1. Identity

By definition, a client is associated with exactly one identity. An agent will store data that is owned by a particular client, based upon that client's identity. Since a client can communicate via multiple transport channels and no channel needs to be active for the agent to have associated state, the client's identity is used to identify the ownership of the data stored by the agent.

Similarly, by definition, an agent is associated with exactly one identity. A client may also store local state associated with a particular agent. The agent's identity can be used to identify ownership of the data stored by the client.

The details of what constitutes an identity can be dependent upon the specifics of the I2RS protocol and selected security mechanisms. However, there are some critical considerations for identity that do impose constraints.

An identity is not tied to a single communication channel. A client may use multiple IP addresses; an identity should not be tied to a specific IP address. If the client or agent is associated with a system that may be mobile, that should be considered in its identification. Finally, the syntax and semantics for identifiers used for a client and for an agent may be different.

3.2.2. Security Role

In the context of an agent, each client will have a security role. The client's identity and associated security role will have to be verified via an acceptable security mechanism. A variety of such mechanisms are anticipated to meet different security and operational

objectives. Example mechanisms might include a role assertion from the client to the agent that the agent can cryptographically verify or having the agent to use an already trusted protocol to verify the client's security role and identity.

An agent must know the scope and resources associated with each particular security role. This information may vary across different agents even in the same network or it may be consistent across different agents in the same network. The latter can be enforced by having a client that is authorized to influence the meta-data model of security roles on the relevant set of agents.

A security role also defines what precedences (See Section 3.2.8) a commissioner can use.

3.2.3. Security Model

As described above, roles identify the scope and resources allowed to an I2RS Client. The policy model therefore needs to include these roles. The question of the bindings of identities to roles, and the selection of identities are protocol specific matters outside the scope of this document.

The policy model for roles needs to address these two dimensions. It needs to create the roles themselves. This should allow for use of techniques like inheritance, presumably with some rules on how role definitions can augment or restrict the inherited definitions.

The security model also needs to define, by reference to the policy model itself, the scope of the role. The question of defining the resources of a role is for further study. The role definition needs to indicate what types and instances of data can be observed and what information about those instances entities with that role can observe. The security model also needs to define which data items can be modified, and what modifications (ranges, specified values, or other assertions that must be met) are permitted.

3.2.4. Scope

Scope is specified as part of a security role. A security role may be defined and managed in an external repository, centralized within an administration. The security role definitions must be accessible to an agent.

In the context of an interaction between a client and an agent, the effective scope is restricted to the intersection of the scopes of the two entities.

What information a particular client is authorized to read is known as the clien's read scope. A read scope includes the ability to see that particular data exists and to read the same data. The read scope can have its constraints specified in terms of specific portions of data models.

Similarly, what information a client can write (add/modify/delete) may be contrained. This is known as its write scope. The write scope is specific in both the parts of the data models and in the set and range of data that can be written. For example, a client might be able to write static routes in the RIB data-model for prefixes in 10.0/16.

While the client's behavior and functionality is specifically out-of-scope, it is useful to describe the same scope concepts for an agent operating in the context of a client.

An agent's read scope is the set of data that the agent can read or have access to. An agent would generally learn such data because the client has sent that data to the agent in an operation.

An agent's write scope is the set and range of data that the agent is allowed to provide to the client and that will be accepted by the client. For instance, client B may accept next-hop change notifications for prefix 10.0/16 from agent 1 but not from agent 2.

3.2.5. Resources

When a client sends operations to an agent, those operations can consume resources. Therefore, it is important that the agent have policy to limit the resources available to a particular client. This is based on the client's identity and security role. Such resource policy specifications need to be provided in a data-model that can be modified by appropriately authorized clients or local configuration.

Examples of such resource constraints include:

Number of installed operations owned,

Number of operations that haven't reached their start-time, and

Number of event notifications registered for.

As discussed in Section 3.2.7, a client can specify priorities for the operations it sends.

If compute resources are considered, it is not the intent to try and determine the computation associated with particular operations.

Instead, the constraint could be on percentage of the I2RS agent's compute-time given to a client every pre-defined period. This could provide a mechanism for fair sharing of compute resources between clients.

3.2.6. Connectivity

A client does not need to maintain an active communication channel with an agent. Therefore, an agent may need to open a communication channel to the client to communicate previously requested information. The lack of an active communication channel does not imply that the associated client is non-functional. When communication is required, the agent or client can open a new communication channel.

State held by an agent that is owned by a client should not be removed or cleaned up when a client is no longer communicating - even if the agent cannot successfully open a new communication channel to the client.

3.2.7. Priority

The motivating example for priority is when a single client is sending operations to accomplish multiple tasks. For example, one task might be long-term and another task might deal with unexpected requests that are more important. In this case, the client may wish to provide a hint to the relevant agents as to which operations should be done first.

Communication from a client can come across multiple channels, so simply specifying that operations be done in order is not sufficient. Additionally, all operations may not be immediately carried out, due to varying start-times or other constraints. With these factors and this motivating example, it is useful to introduce the concept of prioritization for operations sent from the same client.

By introducing the concept of priority for operations, a client can accomplish multiple uncorrelated tasks that affect the same agent with the specified prioritization.

A default priority can be specified for each particular communication channel. In addition, an I2RS operation can specify a priority to use instead. Priorities between operations from different clients need not be compared.

The priority can be used by an agent to determine which operation from a client to execute next.

3.2.8. Precedence

A mechanism is needed for the agent to determine what state to install when there are overlapping install operations. An install operation may overlap with locally-installed configuration state or with a previous install operation that was requested by a client. The mechanism to resolve this is termed "precedence". No simple mechanism can fully handle indirect interactions; considering such interactions is out-of-scope. Indirect interactions must be considered when different clients are given their tasks.

Precedence is a TLV; there is a precedence type and a precedence value that can vary based upon the precedence type. The different precedence types are ordered with regard to another so that, for instance, a precedence type of "Simple Integer" is preferred to precedence type of "mouse-type". If more than one operation has the same precedence type, then the precedence values are compared based upon the rules for the associated type. If multiple clients have equivalent precedence (based both on type and compared values), then preference is given to the newer operation that is being written. This tie-breaking policy is equivalent to that used by CLI or NetConf, where the new command or RPC gets to do its add/modify/delete operation. The different precedence types are ordered with regard to each other; the lowest precedence type will be preferred. If there is a tie for precedence type, then the precedence values will be compared and the preferred will be selected based on the precedence type's policy.

Option A: Type 100 ("Simple Integer") Value 10.

Option B: Type 200 (mouse-type) Value "cheese"

Option C: Type 100 ("Simple Integer") Value 10

Option D: Type 100 ("Simple Integer") Value 8

If Operation A arrived first and installed state, then when the I2RS agent decides whether to install Operation B, Operation B would be rejected or stored because A's type 100 is better than B's type 200. If Operation C were to arrive, however, then Operation C would be installed and Operation A preempted because A and C have the same type and value, so tie-breaking is done to prefer the new arrival. If Operation D were to arrive with A installed, then D would be rejected or stored because A and D share the same type but D's value of 8 is less than A's value of 10.

Given that clients are dynamically sending write operations and the associated arrival times can vary based on anything from program

state to network conditions, predictability is much better provided by using precedence instead of operation arrival time or start time many operations may be immediate start).

Each write operation has a precedence associated with it. This precedence may come from the default associated with the clientw, with the specific communication channel, or with the specific operation. The range of possible precedences that can be used is known based on the client's security role. The determination of the precedence associated with any operation is a policy decision at the agent, but may utilize any or all of the information described above.

When a write operation is executed, the agent first determines if there is overlapping existing I2RS-installed state. If not, the agent must determine if it overlaps existing local-configuration state. Local-configuration state will also have a precedence associated with it so that the agent can make an appropriate decision.

A client can specify whether a write operation should be store-if-not-best. This allows a client to determine what happens when a write operation doesn't win the precedence comparison. If store-if-not-best is specified, then the write operation succeeds and the associated installed state is stored but not actively installed by the agent. If store-if-not-best is not specified, then the install operation will fail.

The store-if-not-best flag is stored with the installed operation's precedence. If the agent determines that an installed operation must be preempted, then the agent consults the store-if-not-best flag. If store-if-not-best is specified, then the agent stores the preempted operation and does not notify the associated client. If store-if-not-best is not specified, then the agent notifies the associated client of the preemption and removes the previously installed state.

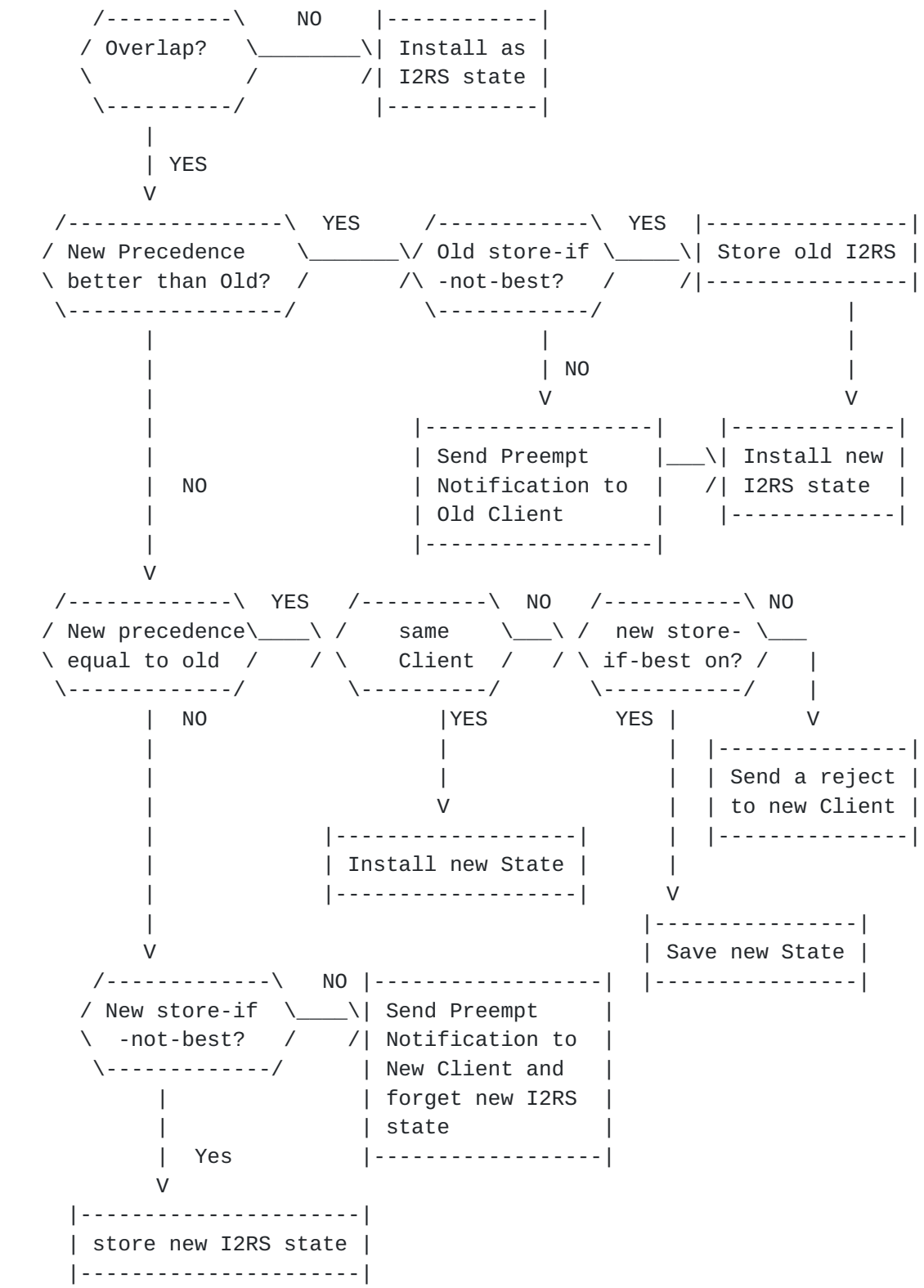

Figure 2: Precedence Decision-Making

If the overlapping new operation has a precedence that is better than

the existing state, then the agent should preempt the existing state and act according to the existing state's store-if-not-best flag. If that store-if-not-best flag is set, the agent will store the old state and install the new state. If the store-if-not-best flag is clear, the agent will send a preemption notification to the old client, install the new I2RS state, and forget the old.

If the overlapping existing state has the same precedence and the same client associated, then the agent completes the write operation; otherwise, the agent must reject or store the write operation, based on the store-if-not-best flag.

If the new overlapping operation has a precedence that is worse than the existing state, then the agent must reject or store the write operation, based on the state of the new store-if-not-best flag. If the store-if-not-best flag is set, then then the agent will store the new I2RS state. If the store-if-not-best flag is clear, then the the I2RS agent will send a preempt notification to the new client and forget the new I2RS state.

This decision process is illustrated in Figure 2.

When a delete operation is done, the stored state with the next best precedence should be selected and installed.

A consequence of the precedence policy mechanism is that a client must be able to handle its installed operations being preempted at any time, either explicitly or simply by having the active state changed. Such preemption can be minimized by appropriate separation of tasks, with their associated write operations, between the local systems of the clients and by knowledgeable local system configuration.

3.3. Policy between Agent and Local System

It is critical to understand and clearly specify how I2RS interacts with local configuration. The key questions are:

1. What happens when Local Configuration overlaps with I2RS installed state?

2. What happens when I2RS installed state is removed?

3. How is state recreated when a local system reboots?

A consequence of using I2RS is that the local system's state may not be synchronized with the local configuration. Since this is a change in understood behavior, any discrepancies should be clearly visible

to the operator with an associated explanation.

Logically, the local configuration is essentially modeled as a local client, with its own precedence, identity, and security role and immediate permanent write operations. The key differences are both that all relevant local configuration state need not be cached by the agent and that reboot imposes the need to process local configuration state before any other I2RS-installed state.

3.3.1. Local Configuration

The local system's local configuration may have overlapping write scope with that of one or more clients using an agent. Therefore, explicit and implicit policy interactions must be specified. The mechanism that I2RS provides for deciding between overlapping install operations is "precedence". This same mechanism can be used to decide between local configuration and an I2RS operation. Local configuration can specify the precedence to be used for the local system.

A precedence that causes the desired behavior can be specified as follows. (MAX is the highest precedence given to a client. MIN is the lowest precedence given to a client.)

MAX+1 Precedence: If the local configuration has a precedence higher than that given to any client, then state from the local configuration will always be installed. If any I2RS-installed state is therefore preempted, the agent will notify the associated client.

MIN-1 Precedence: If the local configuration has a precedence lower than that given to any client, then I2RS-installed state will always override local configuration. That this preemption has occurred should be reflected in how the local system displays its state.

Other Precedence: The local configuration can have higher precedence than that given to some clients, lower precedence than that given to other clients, and equal precedence to that given to other clients. Then some local configuration state may be preempted by I2RS-installed state while some I2RS-installed state can be preempted by local configuration.

Local-configuration wins all precedence ties.

Just as an agent must check to determine if a write operation overlaps with existing installed state, the process of committing local configuration must check to see if there is overlapping I2RS-

installed state.

What the process of committing local configuration is can vary by local system. Well known examples are when a return is sent to the CLI and when an explicit commit command is specified. How the proper checks for interaction between the agent and local configuration are done is a local system matter.

Similarly, when an agent checks to see if an write operation overlaps with existing installed state, the agent must determine if it overlaps with existing local configuration.

If the precedence associated with local configuration is changed, then it is retroactive. All local configuration state stored by the agent must be updated with the new precedence and installation decisions made for overlapping data. This change could be very disruptive.

3.3.2. Removal of I2RS-installed State

When a piece of local configuration is removed, the local system goes back to the appropriate system default. However, when an operation deletes some I2RS-installed state, the correct behavior is not to just go back to the system default. Instead, any stored state must be considered - whether that comes from local configuration or stored I2RS write operations that didn't have the highest precedence. If there is any stored state, then the highest precedence of the options is selected and installed. That existing overlapping state might come from the local-configuration.

If I2RS's implicit policy were to just go to the system default, then the local configuration and the local system state would not be synchronized and there would be no remaining I2RS-state to explain the discrepency. Since I2RS state can also be stored and not installed, the same mechanism can be used for stored I2RS install operations and for local configuration.

3.3.3. On Reboot

When the local system reboots, only persistent I2RS-installed state is preserved by the agent. The implicit policy for I2RS is that the local configuration is read and installed first. After the local system has its local configuration installed, the persistent I2RS write operations are executed to bring the system to the persistent state.

4. Policy in an I2RS Service

It is critical to consider how policy influences a service when defining the service and its associated data-model(s). There are several different aspects to consider.

How are scope and influence policy specified in the data model? What granularity levels are necessary for the particular service?

How does the implicit policy in the associated routing sub-system effect what I2RS can be allowed to influence?

Are the implicit policies of the associated routing sub-system captured in the semantic content of the information model, data model, and description?

What explicit policy communicated in the associated routing sub-system needs to be included in the data-model? What indirection and abstractions are needed?

4.1. Resource Reservation and Three-Phase Commit

Some agents and services may offer the ability to reserve resources required by operations before the operation start time. There are two aspects to how to support this.

First, if the agent can do time-aware resource reservation, then a write operation can specify "reserve-only" to prompt an acknowledgement or failure as to the ability of the agent to confirm the reservation. Then the client can either send an operation to commit the reservation, which causes the associated write operation, or to remove the reservation. A "reserve-only" operation will have its reservation expire at the end of its associated life-time.

Second, part of a service's data-model may be to request a reservation with a known start-time and duration. An example might be reserving a specific bandwidth on a path for an LSP between two devices. It is important to consider whether a particular service should offer a time-based reservation service as part of its data-model.

4.2. Defining I2RS Behavior Based on Implicit and Explicit Policy

The semantics in a data-model must respect and describe the implicit policy of the associated routing sub-system. This doesn't imply that the data-model components should instantiate it or allow reading or writing.

Policy Routing systems must deal with the verification, reading and installing of routes from sources such as EGP, IGP, and static routes. Policy routing may also control forwarding and the monitoring of data forwarding; and data resources. The explicit policy examples are given for the routing framework. It is assumed the reader can extend this framework to the data forwarding and data resource arena.

4.2.1. Example of Implicit Policy

The ISIS protocol specification uses implicit policy to set constraints on level 1 peers. Due to this fact, many ISIS implementations only let one level 1 ISIS peer associate with one Level 2 peer domain.

This policy is not encoded in any local configuration directly, but is rather included as an implicit policy. When local configuration policy is checked (prior to a configuration commit), this local policy is checked. If the configuration input from a CLI is in error, the input will be rejected, and the CLI will warn the user. Similarily programmic interfaces for the local configuration cause the implicit policy to be checked.

I2RS data models guide the client in an interoperable interaction with the reading and installation of data at a particular agent. The I2RS data models must contain both the implicit policy and the explicit policy. Although an agent may not report the I2RS implicit policy in the protocol, the client must know of the existence of the implicit policy.

This knowledge allows the client to know the implicit policy interactions on different systems in a heterogeneous network. For example, assume a situation where a client is talking to two agents - one on system A and one on system B. The routing process on system A has has different implicit rules for the ISIS Level 1 peer to Level 2 peer connection than the routing process on system B. Routing process A is built to allow one level 1 ISIS peer associated with 2 ISIS Level 2 peers. Routing process B upholds the standard implicit policy that 1 level ISIS peer can only be associated with 1 ISIS Level 2 peer. The client setting up the ISIS peering in a network containing system A and system B must know that System A will allow a level 1 peer to connect to 2 ISIS Level 2 peers. When the client's scope allows it to read data from system A and system B, it should not flag the difference in ISIS level 1 peer connections as a problem. Instead the client will need to determine if the use of the different configurations can cause a network problem.

4.2.2. Passing Explicit Policy

Routing systems' explicit policy controls protocols, associates/ deassociates interfaces, route verification policy, route forwarding policy, route aggregation policy, and route deaggregation policy. All of this policy can be found in the detailed configuration specification of a routing process. However, even via CLI, it is rarely possible to configure all the possible options. Other configuration mechanisms do not have public models for all the private router configuration. The developers of a routing system often have a complete policy model either in formal modeling languages or informal language.

Explicit policy contained in an I2RS data model is the detailed configuration model at the deepest level that an agent can access. This detailed configuration model may come from IETF Standards and/or the vendor specific configurations. The public data models must specify a vendor specific tree where the individual configuration is plugged into.

4.2.2.1. Explicit policy on Data Forwarding, Resources, and Policy passing

Forwarding policy has to do with the data flow may also be controlled by an agent. If so, the explicit policy must be placed in a data model along with the implicit policy.

Lastly, protocols have begun to pass explicit policy about passing policy. Examples of this type of policy are BGP ORFs, BGP Flowspecs, and ISIS policy passing. Clients must know the implicit policy and explicit policy this policy impacts, and the precedence between these policy. Due to the extensive use of BGP ORFs and the growing use in BGP Flowspecs policy, early data models for BGP should describe the implicit policy, explicit policy, policy precedence for the BGP ORFS and BGP FlowSpecs, and how they interacts with other BGP, route policy and preferences. This information should be placed inside an I2RS data model for an agent supporting these features.

These explicit models for BGP policy are not trivial, but these models exist today. Frequently, I2RS data models may be simply a casting of existing implicit policy and explicit policy into a common standard form so that programmic interfaces may interact with a routing element.

4.2.2.2. Example of Explicit Policy

There are two clear explicit policy pieces for ISIS. First is the peer level. Second is the policy of the external routes to be

redistributed into and out of ISIS.

5. Acknowledgements

The authors would like to thank Ross Callon, Adrian Farrel, David Meyer, David Ward, Rex Fernando, Russ White, Bruno Risjman, and Thomas Nadeau for their suggestions and review.

6. IANA Considerations

This document includes no request to IANA.

7. Security Considerations

This is empty boilerplate for now.

8. Informative References

[I-D.atlas-irs-problem-statement]
Atlas, A., Nadeau, T., and D. Ward, "Interface to the Routing System Problem Statement", draft-atlas-irs-problem-statement-00 (work in progress), July 2012.

[I-D.ward-irs-framework]
Atlas, A., Nadeau, T., and D. Ward, "Interface to the Routing System Framework", draft-ward-irs-framework-00 (work in progress), July 2012.

Authors' Addresses

Alia Atlas
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
USA

Email: akatlas@juniper.net

Susan Hares
Hickory Hill Consulting

Email: shares@ndzh.com

Joel Halpern
Ericsson

Email: Joel.Halpern@ericsson.com