RELOAD Status

draft-ietf-p2psip-base-02.txt
draft-ietf-p2psip-sip-01.txt

Eric Rescorla

P2PSIP WG

IETF 74 (SFO)

Changes Since MSP (01/00 drafts)

- Dynamic updates of configuration files
- Addition of new kinds without an RFC
- Removed REMOVE method
- Some work on reliability/flow control (unfinished)
- Removed Diagnostics text

Dynamic Configuration File Updating

- How do overlay configurations change?
  - Add new kinds
  - New permissions
  - New algorithms?
- Easy to update configuration file
  - But how do existing nodes get it?
  - Need some RELOAD mechanism for update

Basic Approach: Quasi-Flood

- Configuration documents have sequence #
  - Monotonically increasing
  - Carried in forwarding header
- If you receive a request with old SN
  - Reject with `Config_Too_Old`
  - Generate `Config_Update`
- If you receive a request with new SN
  - Generate a `Config_Too_New` error
  - The other node generates a `Config_Update`

Configuration Document Signing

- In -01 config documents fetched over HTTPS
  - Didn't need to be signed
- In -02 you can also get config from peers
  - Now we have a security issue
- All configuration documents are now signed
  - Using the public key of the config server (used for TLS)
- Format issues
  - Explicitly not using CMS or DSIG
  - We already have a RELOAD signing construct
    - Sign raw XML
    - Base-64 and insert
  - Kind of hacky but easier for implementors

Addition of New Kinds

- Want to define new kinds without an RFC
  - Suggestion from Vidya Narayanan
- Proposed approach
  - Define new kinds in config document
  - Required kinds must be listed there anyway
- Requires two changes
  - Allow kind definitions to include numeric kind-ids
    - means “this is defined here”
  - Require an explicit access control policy for each listed kind

Example Syntax

```
<kind id="2000">
  <data-model>array</data-model>
  <access-control>user-match</access-control>
  <max-count>22</max-count>
  <max-size>4</max-size>
</kind>
```

Defined Access Control Policies

- `USER-MATCH` -- user name must hash to resource-id
- `NODE-MATCH` -- node-id must match resource-id
- `USER-NODE-MATCH` -- For Dictionaries. `USER-MATCH` + Dictionary key == node-id
- `NODE-MULTIPLE` -- node-id + index must hash to resource-id
- `USER-MATCH-WITH-ANONYMOUS-CREATE` -- anyone can create, `USER-MATCH` for overwrite

Open issues with this approach (my interpretation of Vidya)

- This requires a centralized server
  - To generate and sign the configuration document
  - Might be possible to delegate this permission
    - E.g., some designated set of writers
    - Need to deal with write conflicts somehow
- Why not let the writer choose access control model?
  - Write(ResourceID, access_control_model, data)
  - Each writer gets separate space

RELOAD Storage Security Goals

- Data integrity
  - Ensure that data stored by A is really from A
- Access control
  - Prevent A from overwriting B's data
- Limit resource consumption
  - Contain the amount of data any user can store
  - Contain the amount of resources any peer needs to allocate

Distributed Quota

- For a network of P peers and U users and b-bit IDs
- An object of type O can be up to B bytes
- Any given user can store Os at L location
- Total storage per user is BL
  - Total storage in system is UBL
  - Average peer must store UBL/P
- What happens if we allow users to select security model on store?
  - They could store at every location in the overlay!
    - Up to BL * 2^b storage per user!
  - This is inconsistent with quota models

Removed REMOVE

- REMOVE turns out to be tricky
  - For instance: how long do you remember REMOVED values?
- Proposed resolution: Remove it
  - RELOAD already supports “nonexistent” values
    - Used to represent REMOVED objects, gaps in arrays, etc.
  - To remove an object, STORE a “nonexistent” over top
  - This makes all the semantics look like ordinary stored values

Transport/Reliability, etc.

- We'd really like to use TCP between nodes
  - Unfortunately we can't rely on this
  - Firewalls, NATs, etc.
- Only mature IETF NAT traversal technology (ICE) uses UDP
  - ICE TCP is far from done
  - Existing research on TCP traversal isn't that convincing
- Need to provide some reliable transport using UDP
- This just recaps existing WG decisions

Fragmentation

- Each hop can fragment
  - Each fragment has full forwarding header
  - Final destination reassembles
- Forwarding header must be < 1 MTU
  - Previously it could get pretty large
  - Removed route_log
  - Shrink via list
    - Entries can be just adjacency ids (16 bits)
    - Special format to support these

Congestion Control

- Can’t send a lot of data without cong. control
  - So we need something
- Basic concept
  - MUST NOT be more aggressive than TCP
  - Standardize feedback
  - Recommend some sending CC algorithms
    - Rely on feedback
  - Potential algorithms: stop and wait, AIMD, TFRC
    - But only requirement is the aggressiveness limit

PMTU Discovery

- DTLS does no PMTU discovery
  - Except for the handshake
  - Leaves this up to the application
- Should RELOAD do explicit discovery?
  - Use PING to do RFC 4821 discovery
  - Advantage: more efficient use of network
  - Disadvantage: adds a lot of latency
- Alternative 1: be conservative
  - Use 576/1280
- Alternative 2: “passive discovery”
  - Send packets at the “natural” size
  - Adjust PMTU estimate downward in response to loss

Queuing
(my interpretation of Bruce)

- You obviously need some kind of queue
  - This must be at least 5 messages deep
  - Must be no more than 500ms wait
- Can have a separate queue for your own data