NETCONF Efficiency Extensions

draft-bierman-netconf-efficiency-extensions-00
IETF 88, November 2013

Andy Bierman

Agenda

- Problems with NETCONF for constrained networks
- NETCONF-EX Solution Proposal
  - 3 new protocol capabilities
  - 4 new protocol operations
- Need for NETCONF-EX?

NETCONF Problem Summary for Constrained Networks

- Message sizes can be very large
- No standard caching mechanisms
- Only message encoding is XML
- Edit transactions can require a sequence of several protocol operations
- No support for bulk edits or patch-list edits
- Data retrieval could be easier to filter so unwanted data is not sent in the <rpc-reply>

<hello> Exchange Problems

- Server always sends a <hello> message with a complete capability list
  - This capability set can be large and likely to change infrequently
- A client could cache server capabilities if there were standard mechanisms to support it
- The server <hello> message could be optimized so an abbreviated version can be sent to reduce <hello> message size by 90%

Configuration Retrieval Problems

- A client is likely to retrieve the entire running configuration with a <get-config> operation before editing any data resources
  - <rpc-reply> will be large, likely among the largest messages sent by the server
- A client could cache server configurations if there were standard mechanisms to support it

Message Encoding Problems

- XML message encoding can be large
- The message encoding should not be coupled to the protocol
  - Encoding format should be extensible
- The client could request the desired message encoding it wants for the session
  - The message encoding could be negotiated in the <hello> exchange, then that encoding is used for all subsequent protocol messages

Datastore Editing Problems

- Multiple protocol operations (1 to 9+) are required to accomplish an edit transaction
  - 1 <lock> + 1 <unlock> for each datastore
    - candidate, running, startup == max 6 operations
  - 1 <edit-config> or <copy-config> for each edit
    - client can choose 1 or more edit steps
  - 1 <commit> to activate the edits
  - 1 <copy-config> to NV-save the edits
- If the session lost in the middle of the transaction, the client has to start over

Datastore Locking Problems

- Client lock procedure can be expensive to implement if multiple datastores need to be locked
- 2 clients attempting to lock multiple datastores at the same time can get stuck holding 1 lock and waiting for another
- A client will likely retry to get the lock if a lock-denied error-tag is returned, so it might want to ask the server to wait instead of returning an error right away

Confirmed Commit Problems

- Same operation <commit> is used to end a commit and make an unconfirmed commit
  - client 2 can end a confirmed-commit procedure started by client 1
- Confirmed commit only allowed if :candidate also supported
  - network-wide commit and rollback applies even if the :writable-running capability is supported instead

Retrieval Problems

- The <get> operation returns all data, not just operational data
- No standard extensible metadata retrieval
- No simple instance discovery mechanism
- No sub-tree depth limit control
- Need proper YANG filter specification

NETCONF-EX Solution

- **:capability-id** Capability
  - allows caching of server capability sets
- **:config-id** Capability
  - allows caching of server configurations
- **:encoding** Capability
  - allows message encoding negotiation
- **<edit2>** Operation
  - allows entire edit transaction in 1 message
- **<get2>** Operation
  - allows simplified and optimized retrieval filtering

:capability-id Capability

- Server maintains an entity tag for its active capability set, called the ”capability-id”
- :capability-id is advertised by both peers
  - Client advertises its cached capability-id, if any
  - Cerver advertises its current capability-id
  - Server waits slightly to receive the client <hello> first. If a match, then send an abbreviated <hello>, else a full <hello>
- Abreviated <hello> contains only the :capability-id and :config-id capabilities

:config-id Capability

- Server maintains an entity tag for the current running datastore, called the ”config-id”
- :config-id is advertised by the server
  - Client compares the config-id value to the value of its cached config-id, if any
  - If a match, then a <get-config> operation is not needed because the cached copy is current

:encoding Capability

- <hello> messages are always sent in XML
- If encoding negotiation fails, default is XML
- :encoding is advertised by both peers
  - Client advertises a priority-ordered list of media types desired for the session
  - Server advertises an unordered list of the media types it supports
  - Highest order client entry in common is used

<edit2> Operation 1/3

- Supports entire edit procedure in 1 request
  - **target**: datastore to edit (candidate or running datastore)
  - **target-resource**: XPath node-set of edit nodes (if :xpath supported)
  - **yang-patch**: ordered edit list on the target resource(s)
  - **test-only**: validate request and exit
  - **if-match**: entity-tag to match or cancel edit

<edit2> Operation 2/3

- Parameter list part 2
  - **with-locking**: edit with exclusive write access
  - **max-lock-wait**: max time to wait to clear locks
  - **activate-now**: <commit> now if :candidate supported
  - **nvstore-now**: <copy-config> now if :startup supported

<edit2> Operation 3/3

- Parameter list part 3
  - **confirmed**: start or extend a confirmed commit
  - **confirm-timeout**: time before revert running
  - **persist**: value required for followup persist-id

Confirmed Commit Operations

- <complete-commit>
  - Complete a confirmed commit procedure
- <revert-commit>
  - Cancel a confirmed commit procedure
- Cannot use existing operations:
  - Existing <commit> and <cancel-commit> rely on the :candidate capability

<get2> Operation 1/3

- Combine several filters and locking for optimized retrieval
  - **source**: datastore to read
  - **filter-spec**: extensible choice of content filters
  - **keys-only**: retrieve key leafs and ancestors
  - **depth**: return limited number of descendant nodes

<get2> Operation 2/3

- Parameter list part 2
  - **if-modified-since**: retrieve only if datastore changed
  - **full-delta**: retrieve sub-trees only if data resource changed
  - **with-defaults**: specify defaults retrieval mode
  - **with-metadata**: specify metadata to include

<get2> Operation 3/3

- Parameter list part 3
  - **with-locking**: read with exclusive write access
  - **max-lock-wait**: max time to wait to clear locks

Need for NETCONF-EX?

- NETCONF scope seems focused on a small number of large routers that are well-connected to stable high-speed networks
  - Not all deployment scenarios can assume stability, low latency, and unlimited bandwidth for network management
- The WG should make NETCONF appropriate for a larger set of use cases than just big router configuration