NorthBound Infrastructure

Proposal on Centralised Management Daemon (MGMTD)

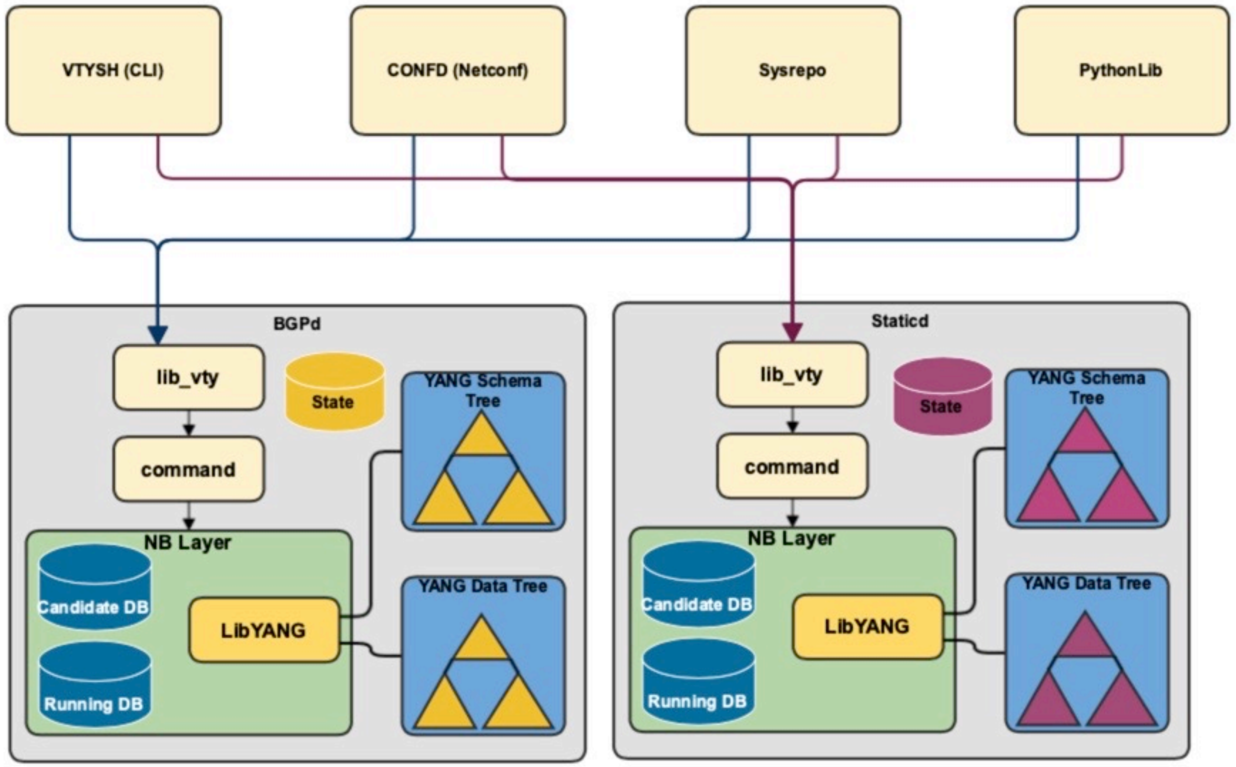
Pushpasis Sarkar <spushpasis@vmware.com>

Current NorthBound(NB) Infrastructure

Problems with it

- Performance impact on individual daemons
  - Loading huge configuration data.
  - Retrieving huge operational data.
- Direct IPC between CLI client and FRR daemons
  - Too many internal IPC channels with other forms of UI
- Running DB maintained inside individual daemons
  - Collection of ‘show running-configuration’ over multiple IPC channels.
  - Possible ordering issues with transaction-based configuration commits leading to inconsistent state.
  - Burden of YANG data parsing and validations on backend daemons.

Current Architecture



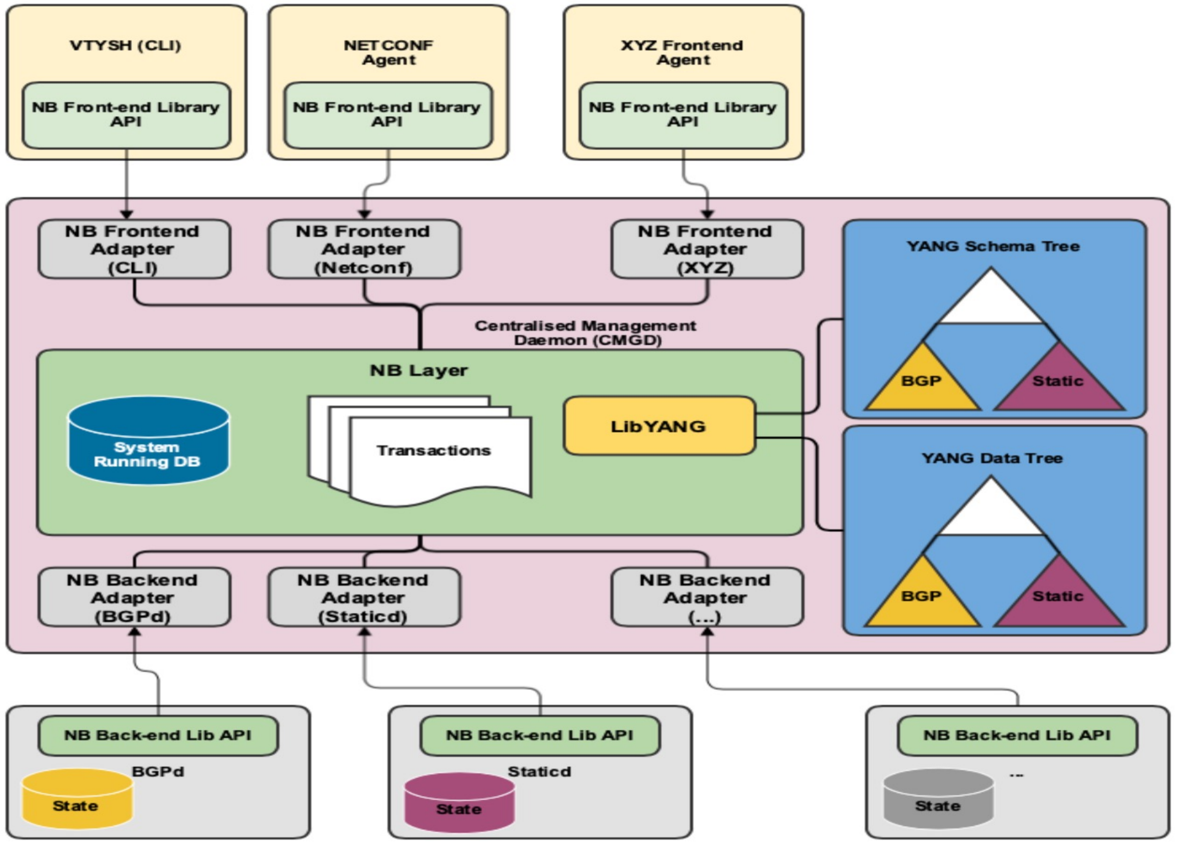
- Running DB and YANG trees maintained across all back-end daemons.
  - Harder to maintain consistency of configuration state across the system.
- Bulk of UI processing borne by back-end daemons.
  - Keeping away from attending to other regular businesses.
  - Performance impact while
    - Commit large configuration or
    - Retrieving large operational dataset(s).
- Full-Mesh' of IPC connections between UI Front-end(s) and Back-end daemons.

Objectives of this Proposal

- Consolidation and management of Running DB by a single entity
  - Better control over configuration validation, commit and rollback.
  - Faster collection of ‘show running-config’.
  - Remove burden of YANG data parsing and validations (model-wise) away from Backend daemons.
- Improve performance and avoid CPU hogging of Backend daemons while:
  - Loading huge configuration changeset(s).
  - Retrieving huge operational dataset(s).
- Avoid too many IPC channels between UI client and FRR daemons
  - Move from ‘Full Mesh’ to ‘Hub-n-Spoke’

FRR Management Daemon(MGMTD)

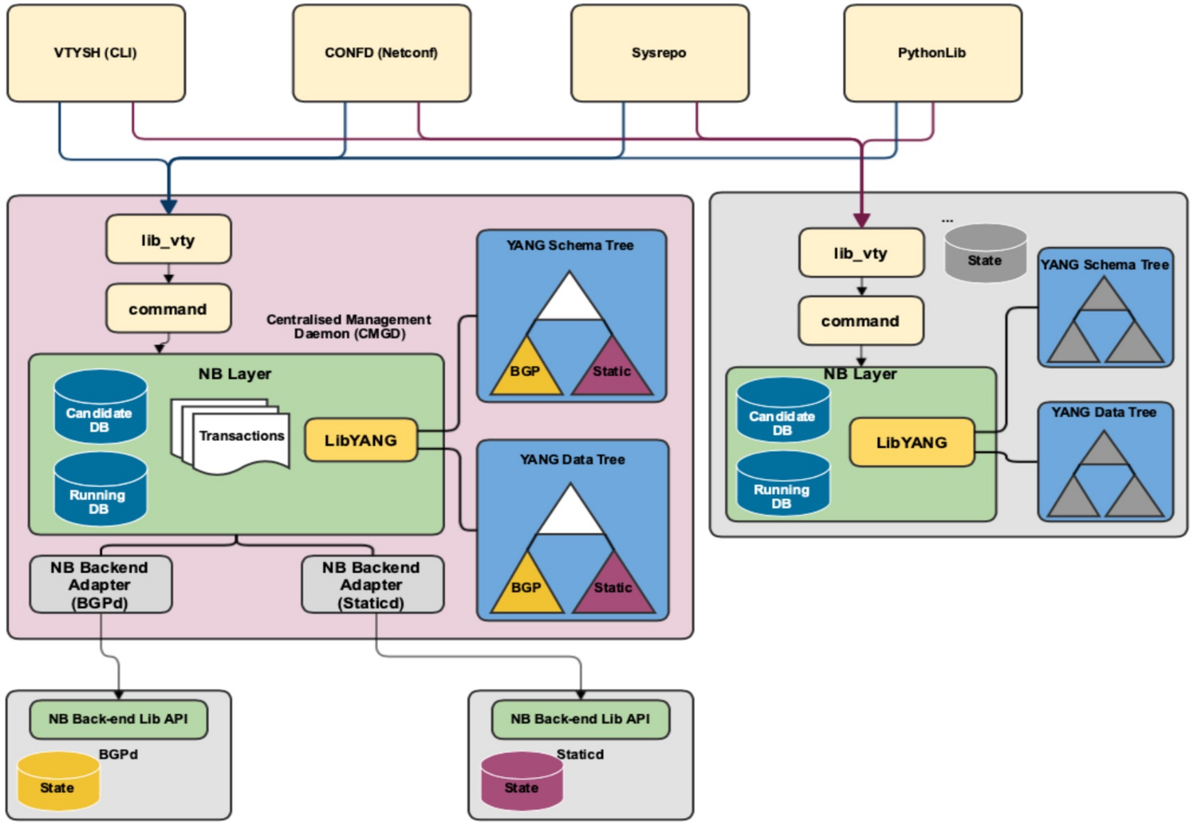
Final Proposed Architecture



- New MGMTD daemon between UI front-end and Backend daemons.
- Hub-n-Spoke instead of ‘Full-Mesh’
- System-wide Running DB and all the YANG trees maintained by a single entity
- Single point to manage and authenticate all UI transactions.
- Minimal UI processing on the backend daemons.

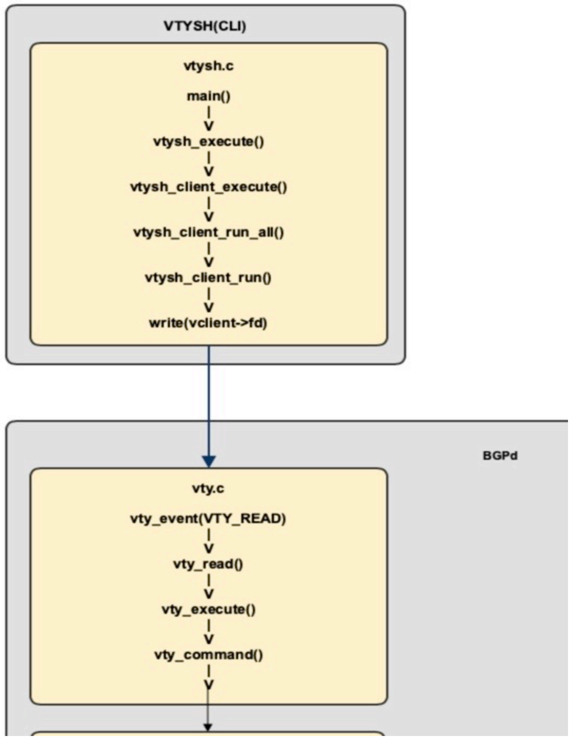
FRR Management Daemon(MGMTD)

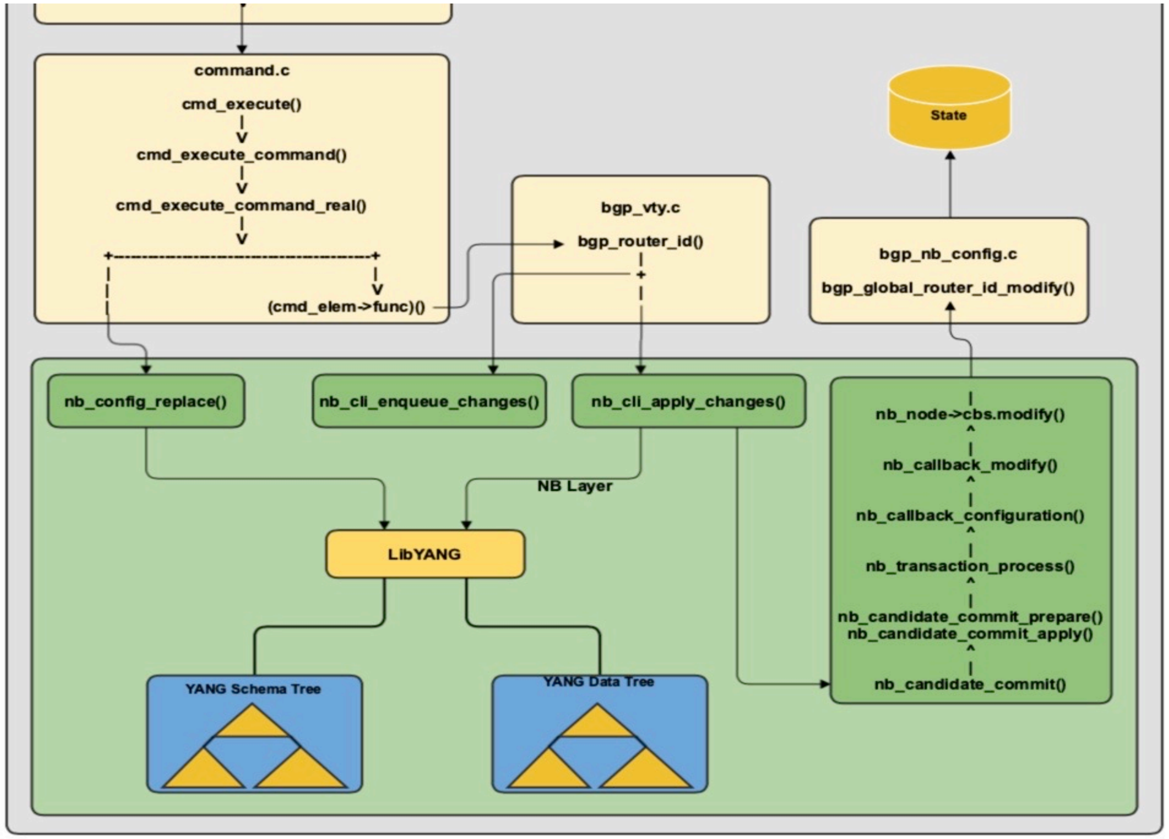
Interim Proposed Architecture



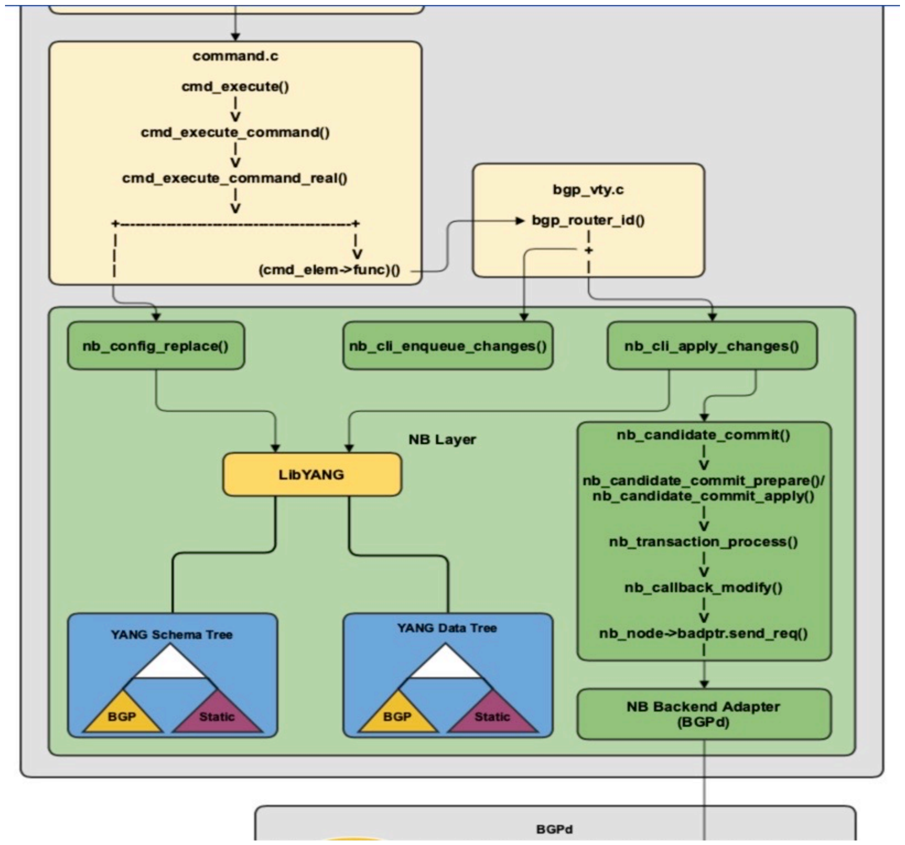
- New and old infrastructure co-exist together.
- Minimal impact.
- Upgrade backend daemons one by one and move to new infrastructure.
- Remove old infrastructure after all backend daemons has been moved to new infrastructure.
- Revamp existing or implement new front-ends using the new infrastructure.

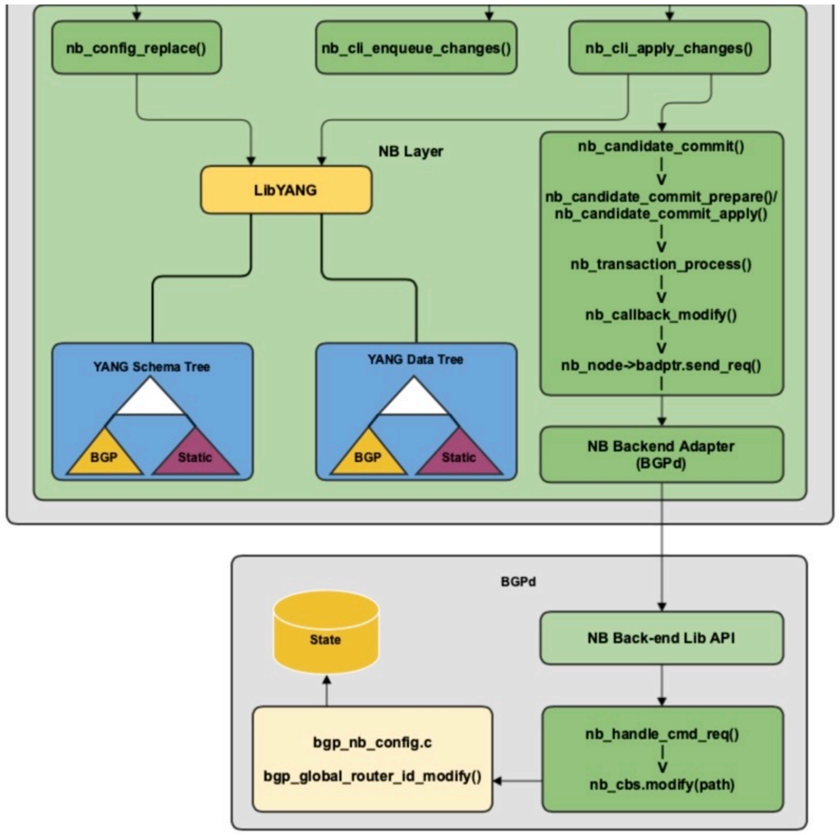
Current CLI Processing Flow





Proposed CLI Processing Flow





Datastore Model

Datastore Model

NETCONF RFC8342

```
+-------------+                 +-----------+
| <candidate> |                 | <startup> |
|  (ct, rw)   |<---+       +--->| (ct, rw)  |
+-------------+    |       |    +-----------+
       |           |       |           |
       |         +-----------+         |
       +-------->| <running> |<--------+
                 | (ct, rw)  |
                 +-----------+
                       |
                       |        // configuration transformations,
                       |        // e.g., removal of nodes marked as
                       |        // "inactive", expansion of
                       |        // templates
                       v
                 +------------+
                 | <intended> | // subject to validation
                 | (ct, ro)   |
                 +------------+
                       |        // changes applied, subject to
                       |        // local factors, e.g., missing
                       |        // resources, delays
                       |
  dynamic              |   +-------- learned configuration
  configuration        |   +-------- system configuration
  datastores -----+    |   +-------- default configuration
                  |    |   |
                  v    v   v
               +---------------+
               | <operational> | <-- system state
               | (ct + cf, ro) |
               +---------------+

ct = config true; cf = config false
rw = read-write; ro = read-only
boxes denote named datastores
```

Datastores Requirements

Types of Datastores

- Config Datastores/DBs
  - NETCONF RFC6536
    - Running DB
      - Global – SUPPORT: YES
    - Candidate DB
      - Global – SUPPORT: YES
      - Per-Session –Not needed
    - Startup DB
      - Global – SUPPORT: YES
    - Intended DB -- Global
      - Should it be Supported?
  - Others
    - Multiple copies for Rollback???
- Operational Datastores/DB
  - Both Config(config:true) and Operational(config:false) data.
  - WRITE operations NOT allowed.
  - READ always fetched from back-end.

Front-End Interface

Front-End Clients

- CLI Terminal / VTYSH
- NETCONF (RFC6241) Agent
- RESTCONF (RFC8040) Agent
- GRPC (grpc.io) Agent

Front-End Clients

NETCONF Requirements (RFC6241/RFC6536/RFC8526)

- Datastores/DBs
  - Running DB <Global>
  - Candidate DB <Global>
  - Startup DB <Global>
  - Intended DB <Global> ???
- Features
  - Writable Running
  - Candidate Configuration
  - Rollback-on-error
  - Validate
  - Distinct Startup DB
  - Operational Default

- Operations
  - Base (RFC6241)
    - <lock>
    - <unlock>
    - <get-config>
    - <edit-config>
    - <delete-config>
    - <copy-config>
    - <commit>
    - <get>
  - Others (RFC6241)
    - <validate> – dry-run for ‘candidate’
    - <commit> without <commit-confirmed>
    - <cancel-commit> – NOT Supported
  - RFC 8526
    - <get-data>
    - <edit-data>

Front-End Clients

NETCONF Capabilities

Capability	Requirement	Support
Writable-Running Capability [RFC6241 section8.2]	• Allow direct write operations like <edit-config> and <copy-config> on ‘Running’ datastore as target.	YES
Candidate Configuration Capability [RFC6241 section 8.3]	• Supports a candidate configuration datastore. • Requires supporting <commit> and <discard-changes> operation along with other regular operations.	YES
Confirmed Commit Capability [RFC6241 section 8.4]	• Support <commit> operation with <confirmed-commit>, <commit-timeout> and <persist> parameters. • Support <cancel-commit> operation.	NO
Rollback-on-Error Capability [RFC6241 section 8.5]	• Support the "rollback-on-error" value in the <error-option> parameter to the <edit-config> operation	YES

Front-End Clients

NETCONF Capabilities (contd.)

Capability	Requirement	Support
Validate Capability [RFC6241 section 8.6]	• Support <validate> operation. • Support <test-option> parameter for <edit-config> operation.	YES
Distinct Startup Capability [RFC6241 section 8.7]	• Support separate running and startup configuration datastores	YES
URL Capability [RFC6241 section 8.8]	• Ability to accept the <url> element in <source> and <target> parameters.	TBD
XPath Capability [RFC6241 section 8.9]	• Supports the use of XPath expressions in the <filter> element.	YES
Yang Library Capability [RFC8526 section 2]	• Advertise support for YANG library 1.1	YES

Front-End Clients

NETCONF Capabilities (contd.)

Capability	Requirement	Support
“with-operational-defaults” Capability [RFC8526 section 3.1.1.2]	• Support <with-defaults> parameter with <get-data> operation.	YES

Front-End Clients

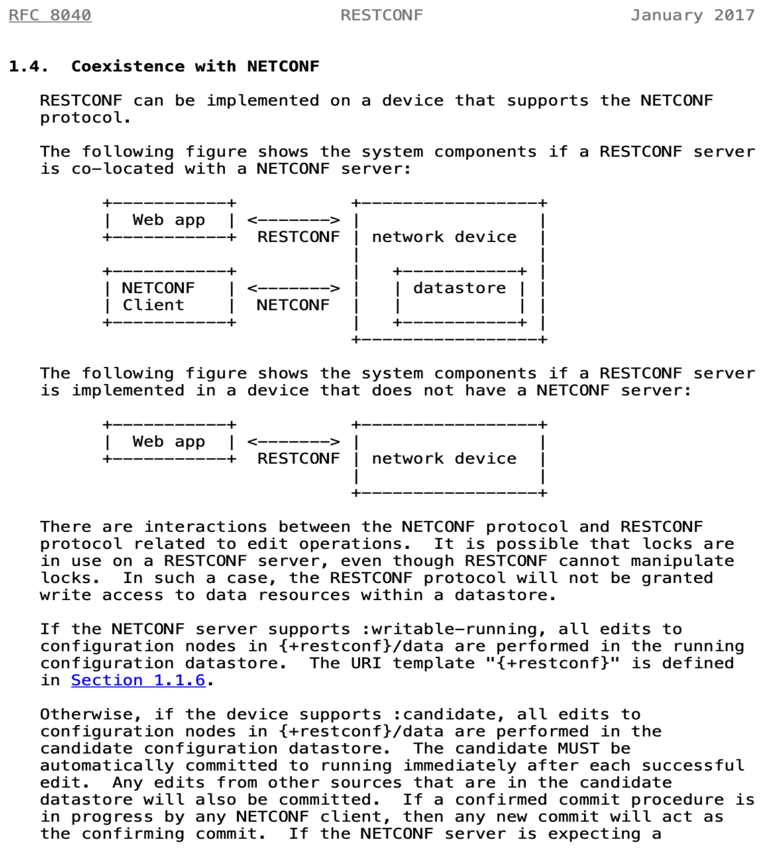
RESTCONF Requirements (RFC8040)

- Datastores/DBs
  - Running DB <Global>
  - Candidate DB <Global>

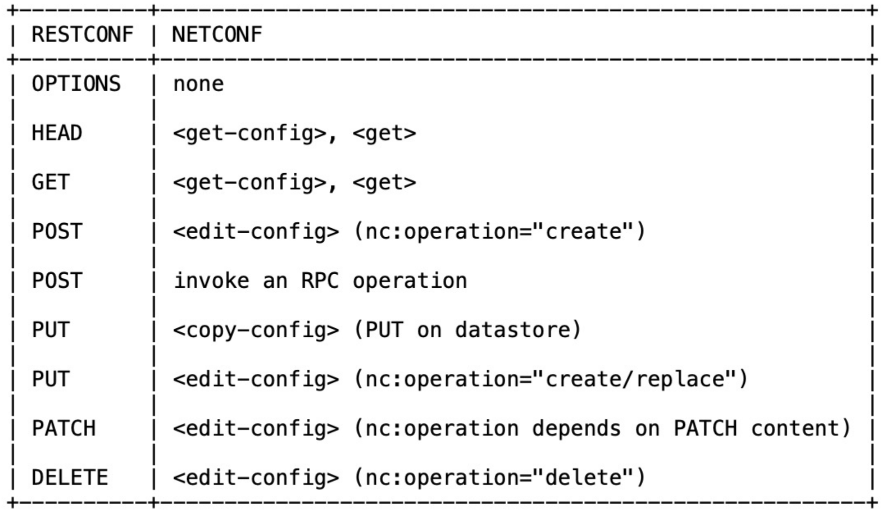
- Operations
  - HEAD
    - Get key fields only.
  - GET
  - POST
    - Create -- Config
    - Invoke -- RPC
  - PUT
  - PATCH
  - DELETE
  - QUERY

Front-End Clients

RESTCONF Requirements (RFC8040)



```
RFC 8040                       RESTCONF                   January 2017

1.4.  Coexistence with NETCONF

   RESTCONF can be implemented on a device that supports the NETCONF
   protocol.

   The following figure shows the system components if a RESTCONF server
   is co-located with a NETCONF server:

        +-----------+           +-----------------+
        |  Web app  | <-------> |                 |
        +-----------+  RESTCONF | network device  |
                                |                 |
        +-----------+           |  +-----------+  |
        | NETCONF   | <-------> |  | datastore |  |
        | Client    |  NETCONF  |  |           |  |
        +-----------+           |  +-----------+  |
                                +-----------------+

   The following figure shows the system components if a RESTCONF server
   is implemented in a device that does not have a NETCONF server:

        +-----------+           +-----------------+
        |  Web app  | <-------> |                 |
        +-----------+  RESTCONF | network device  |
                                |                 |
                                +-----------------+

   There are interactions between the NETCONF protocol and RESTCONF
   protocol related to edit operations.  It is possible that locks are
   in use on a RESTCONF server, even though RESTCONF cannot manipulate
   locks.  In such a case, the RESTCONF protocol will not be granted
   write access to data resources within a datastore.

   If the NETCONF server supports :writable-running, all edits to
   configuration nodes in {+restconf}/data are performed in the running
   configuration datastore.  The URI template "{+restconf}" is defined
   in Section 1.1.6.

   Otherwise, if the device supports :candidate, all edits to
   configuration nodes in {+restconf}/data are performed in the
   candidate configuration datastore.  The candidate MUST be
   automatically committed to running immediately after each successful
   edit.  Any edits from other sources that are in the candidate
   datastore will also be committed.  If a confirmed commit procedure is
   in progress by any NETCONF client, then any new commit will act as
   the confirming commit.  If the NETCONF server is expecting a
```



RESTCONF	NETCONF
OPTIONS	none
HEAD	<get-config>, <get>
GET	<get-config>, <get>
POST	<edit-config> (nc:operation="create")
POST	invoke an RPC operation
PUT	<copy-config> (PUT on datastore)
PUT	<edit-config> (nc:operation="create/replace")
PATCH	<edit-config> (nc:operation depends on PATCH content)
DELETE	<edit-config> (nc:operation="delete")

Front-End Interface

MGMTD Front-End Adapter Message-API

- INIT_SESSION_REQ<Client-Connection-Id>
- INIT_SESSION_REPLY<Client-Connection-Id, Session-Id>
- LOCK_DB_REQ <Session-Id, Database-Id>
- LOCK_DB_REPLY <Session-Id, Database-Id>
- UNLOCK_DB_REQ <Session-Id, Database-Id>
- UNLOCK_DB_REPLY <Session-Id, Database-Id>
- GET_CONFIG_REQ <Session-Id, Database-Id, Base-Yang-Xpath>
- GET_CONFIG_REPLY <Session-Id, Database-Id, Base-Yang-Xpath, Yang-Data-Set>
- SET_CONFIG_REQ <Session-Id, Database-Id, Base-Yang-Xpath, Delete>
- SET_CONFIG_REPLY <Session-Id, Database-id, Base-Yang-Xpath, Status>

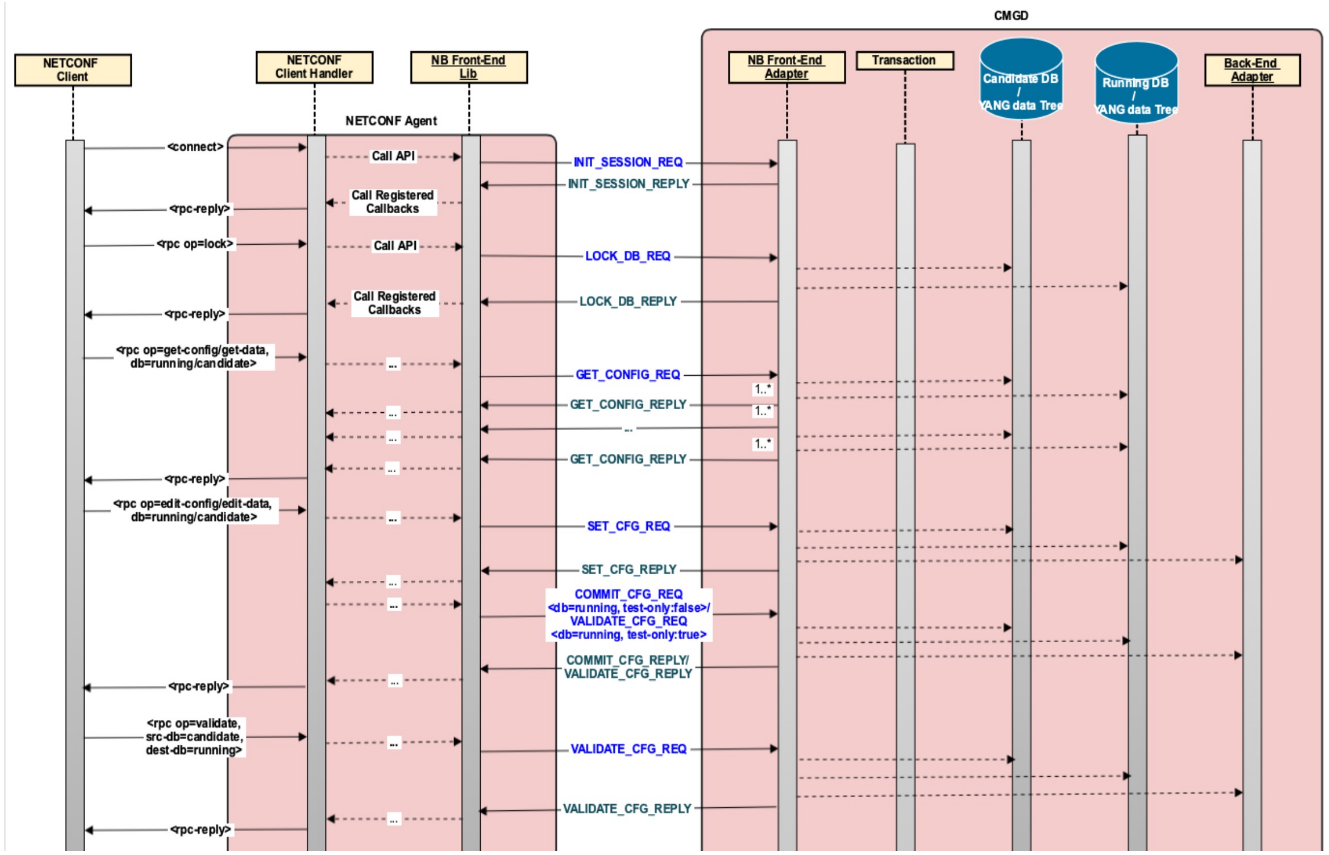
Front-End Interface

MGMTD Front-End Adapter Message-API

- VALIDATE_CONFIG_REQ <Session-Id, Database-Id>
- VALIDATE_CONFIG_REPLY <Session-Id, Database-id>
- COMMIT_CONFIG_REQ <Session-Id, Source-Db-Id, Dest-Db-Id>
- COMMIT_CONFIG_REPLY <Session-Id, Source-Db-id, Dest-Db-Id, Status>
- GET_DATA_REQ <Session-Id, Database-Id, Base-Yang-Xpath>
- GET_DATA_REPLY <Session-Id, Database-id, Base-Yang-Xpath, Yang-Data-Set>
- REGISTER_NOTIFY_REQ <Session-Id, Database-Id, Base-Yang-Xpath>
- DATA_NOTIFY_REQ <Database-Id, Base-Yang-Xpath, Yang-Data-Set>
- CLOSE_SESSION_REQ<Session-Id>

Front-End Interface

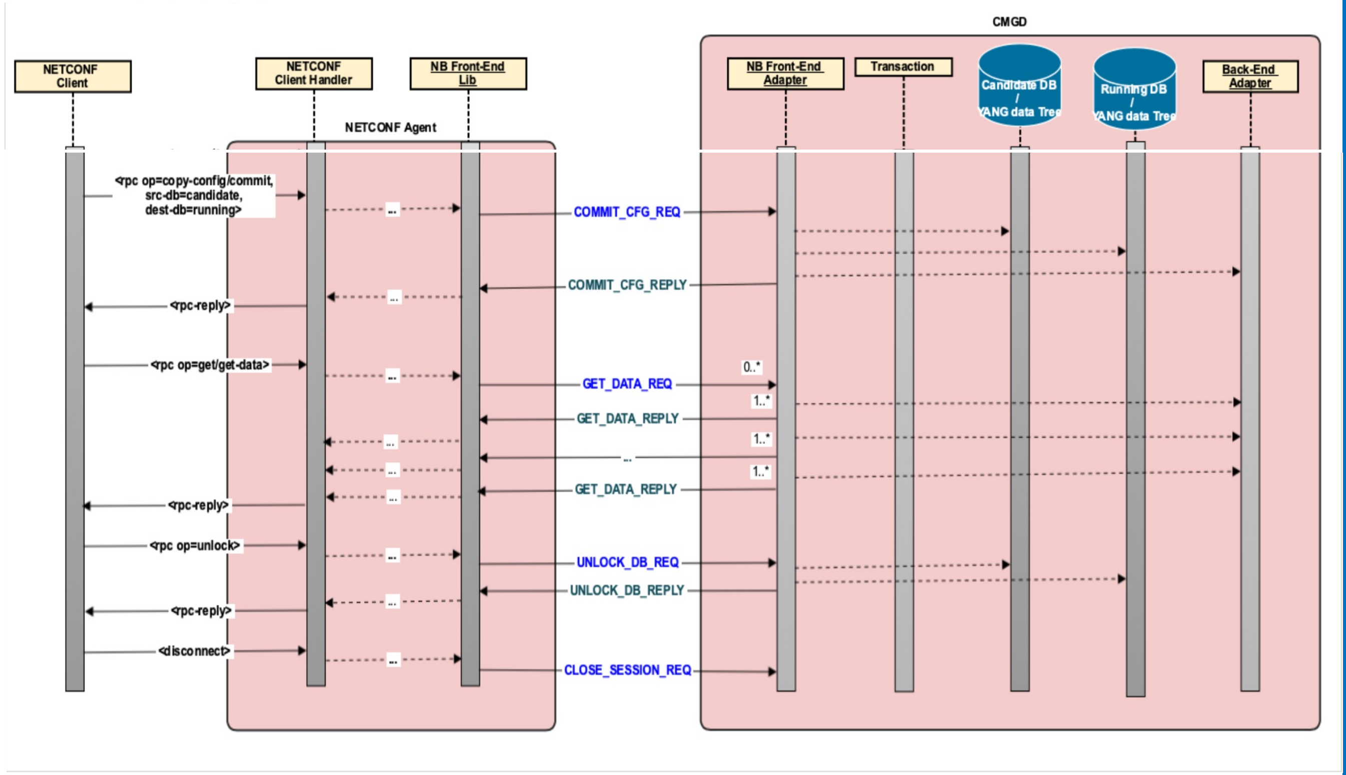
NETCONF Client



- <lock>
  - LOCK_DB_REQ/REPLY
- <get-config/get-data> for 'running' or 'candidate' DB.
  - GET_CONFIG_REQ/REPLY
- <edit-config>/<edit-data>
  - SET_CONFIG_REQ/REPLY
  - VALIDATE_CONFIG_REQ/REPLY for 'running' DB and 'test-only:true'.
  - COMMIT_CONFIG_REQ/REPLY for 'running' DB and 'test-only:false'.
- <validate>
  - VALIDATE_CONFIG_REQ/REPLY

Front-End Interface

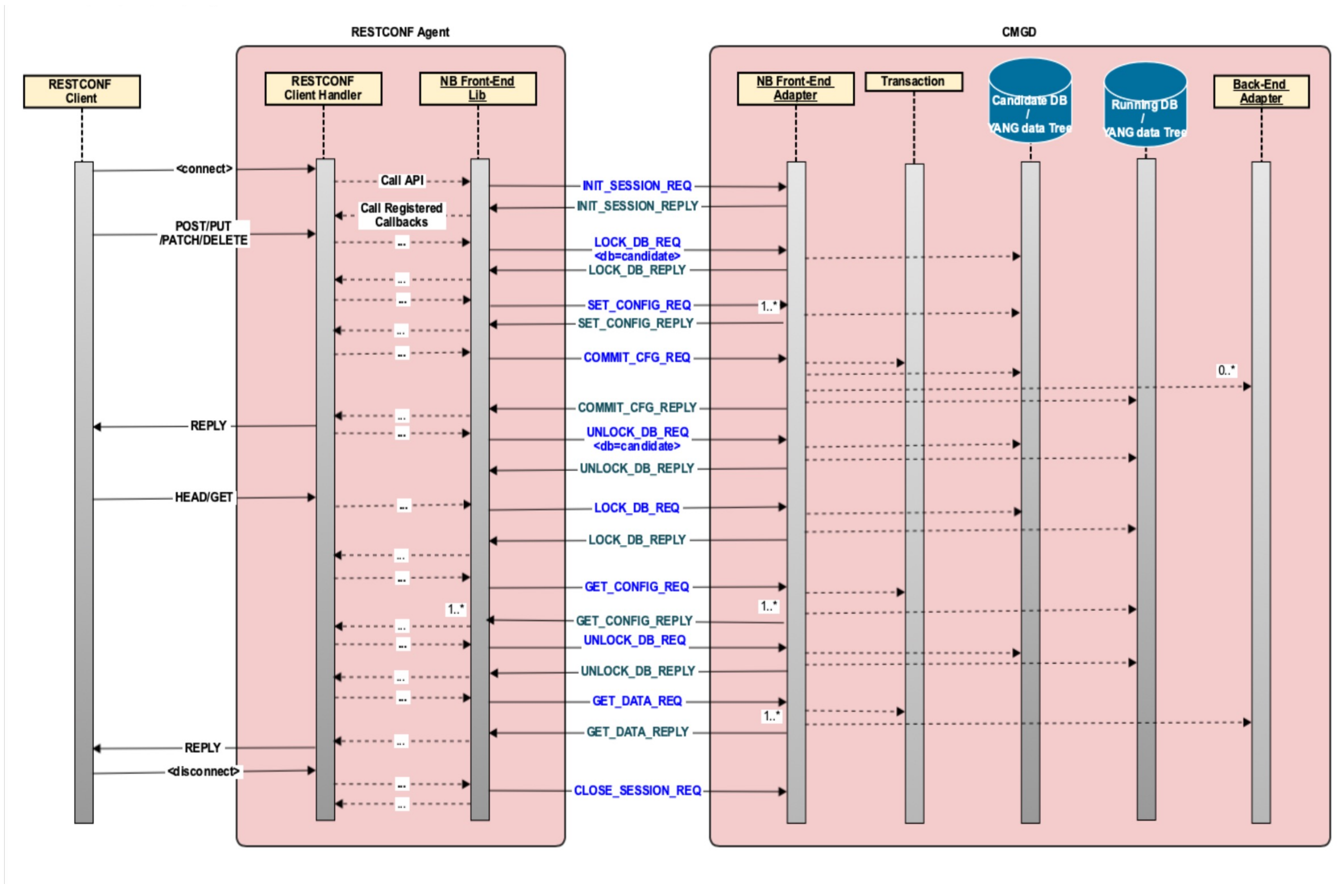
NETCONF Client (contd.)



- <copy-config>/<commit>
  - COMMIT_CONFIG_REQ/REPLY.
- <get-data>/<get> for 'operational' DB.
  - GET_DATA_REQ/REPLY
- <unlock>
  - UNLOCK_DB_REQ/REPLY

Front-End Interface

RESTCONF Client



- POST/PUT/PATCH/DELETE
  - LOCK_DB_REQ/REPLY
  - SET_CONFIG_REQ/REPLY
  - COMMIT_CONFIG_REQ/REPLY
  - UNLOCK_DB_REQ/REPLY
- HEAD/GET
  - LOCK_DB_REQ/REPLY
  - GET_CONFIG_REQ/REPLY
  - UNLOCK_DB_REQ/REPLY
  - GET_DATA_REQ/REPLY

Back-End Interface

Backend-End Interface

MGMTD Back-End Adapter Message-API

- Back-End Registration
  - CLIENT_SUBSCRIBE_REQ <Req-Id, Base-Yang-Xpath, Filter-Type>
  - CLIENT_SUBSCRIBE_REPLY <Req-Id, Status>
- Transaction Management
  - CREATE_TRXN_REQ <Trxn-Id, Create> with ‘Create’=False for Delete request
  - CREATE_TRXN_REPLY <Trxn-Id, Create, Status>
- Notifications
  - DATA_NOTIFY_REQ <Xpath, DataContents>

Backend-End Interface

MGMTD Back-End Adapter Message-API

- Configuration
  - **CREATE_CFGDATA_REQ** <Trxn-Id, Req-Id, Batch-Id, ConfigDataContents>
  - **CREATE_CFGDATA_ ERROR** <Trxn-Id, Req-Id, Batch-Id, Status>
  - **VALIDATE_CFGDATA_REQ** <Trxn-Id, Req-Id, Batch-Id>
  - **VALIDATE_CFGDATA_REPLY** <Trxn-Id, Batch-Id, Status, ErrorInfo>
  - **APPLY_CFGDATA_REQ** <Trxn-Id, Batch-Id>
  - **APPLY_CFGDATA_REPLY** <Trxn-Id, Batch-Id, Status, ErrorInfo>
- Retrieving Opertional Data
  - **GET_OPERDATA_REQ** <Trxn-Id, Req-Id, Base-Yang-Xpath, Filter-Type>
  - **GET_OPERDATA_REPLY** <Trxn-Id, Req-Id, OperDataContents>
- Invoke RPC/Action
  - **ACTION_REQ** < Trxn-Id, Req-Id, Yang-Xpath>
  - **ACTION_REPLY** < Trxn-Id, Req-Id, Status, ErrorIfAny>

Transaction Management

Transaction Management

Low-level Design details

Requirements

- Only one configuration ‘commit’ allowed at any point in time.
- But one configuration ‘commit’ and several ‘show’ commands from multiple sessions MAY be allowed.

Assumptions

- Multiple users may enter config mode and enter config commands.
  - But only the transaction for which ‘commit’ is received first will be put ‘In-Progress’.
  - All ‘commits’ from that point will be responded back with failure till the current ‘In-progress’ commit is finished and the ‘Running DB’ is updated.

Transaction Management

Handling Multiple User Sessions: OPTION1

User 1

```
$ config terminal
# vrf red
# Ip route 1.1.1.1/32 ens192
# show config
        vrf red
        Ip router 1.1.1.1/32 ens192
# commit

        <commit successful>
#
```

User 2

```
$
$ config terminal
# vrf red
# Ip route 1.1.1.1/32 ens224
# show config
        vrf red
        Ip router 1.1.1.1/32 ens224
# commit
< error – commit already in-progress >
# do show running-config vrf red
        vrf red
        Ip router 1.1.1.1/32 ens192
# commit

        <commit successful>
#
```

User 3

```
$ show running-config vrf red
        <blank>
$ show running-config vrf red
        <blank>
$ show running-config vrf red
        <blank>
$ show running-config vrf red
        <blank>
$ show running-config vrf red
        <blank>
$ show running-config vrf red
        vrf red
        Ip router 1.1.1.1/32 ens192
$
$ show running-config vrf red
        vrf red
        Ip router 1.1.1.1/32 ens224
```

Transaction Management

Handling Multiple User Sessions: OPTION2 <Preferred so far>

User 1

```
$ config terminal
# vrf red
# Ip route 1.1.1.1/32 ens192
# show config
        vrf red
        Ip router 1.1.1.1/32 ens192
# commit

        <commit successful>
# end
$
```

User 2

```
$
$ config terminal
   < error – config session already in-progress >
$ config terminal
   < error – config session already in-progress >
$ config terminal
   < error – config session already in-progress >
$ config terminal
   < error – config session already in-progress >
…
$ config terminal
# vrf red
# ip route 1.1.1.1/32 ens224
#commit
        <commit successful>
#
```

User 3

```
$ show running-config vrf red
        <blank>
$ show running-config vrf red
        <blank>
$ show running-config vrf red
        <blank>
$ show running-config vrf red
        <blank>
$ show running-config vrf red
        <blank>
$ show running-config vrf red
        vrf red
        Ip router 1.1.1.1/32 ens192
$
$ show running-config vrf red
        vrf red
        Ip router 1.1.1.1/32 ens224
```

Transaction Management

Types of Transactions

CONFIG – Configuration Transactions

- Write (and possibly read) transactions.
- Initiated with a ‘commit’ command.
- Only one allowed at any point in time.
- Takes a write-lock on the System-wide Running DB
  - Only when merging the candidate DB with running DB -- ???.

SHOW – Show Oper/Config Transactions

- Read-only transactions.
- Initiated with a ‘show’ command.
- Multiple parallel ‘show’ transaction may be allowed.
- Takes a read-lock on the System-wide Running DB.
  - Cannot proceed while ‘merging of the candidate DB with running DB’ from a parallel CONFIG transaction.

Processing Configurations

Processing Configuration

Sub-Phases and Requirements

- **PHASE-1: Converting Text Commands to Yang Xpaths** – VTYSH only
  - See later slides for discussion.
- **PHASE-2: Validating the XPATH and corresponding value against Yang Schema**
  - Again executed on application daemon.
  - Needs access to Yang Schema tree and Data tree (Running DB).
- **PHASE-3: Validating the corresponding value against current configuration**
  - Again executed through 'create/modify/destroy' callbacks on application daemon.
  - Needs access to current configuration in Yang Data tree (Running DB).
- **PHASE-4: Applying the corresponding value on the backend internal state**
  - Again executed through 'create/modify/destroy' callbacks on application daemon.
  - Needs to access backend internal state.
  - Must not need access to Yang Data tree.

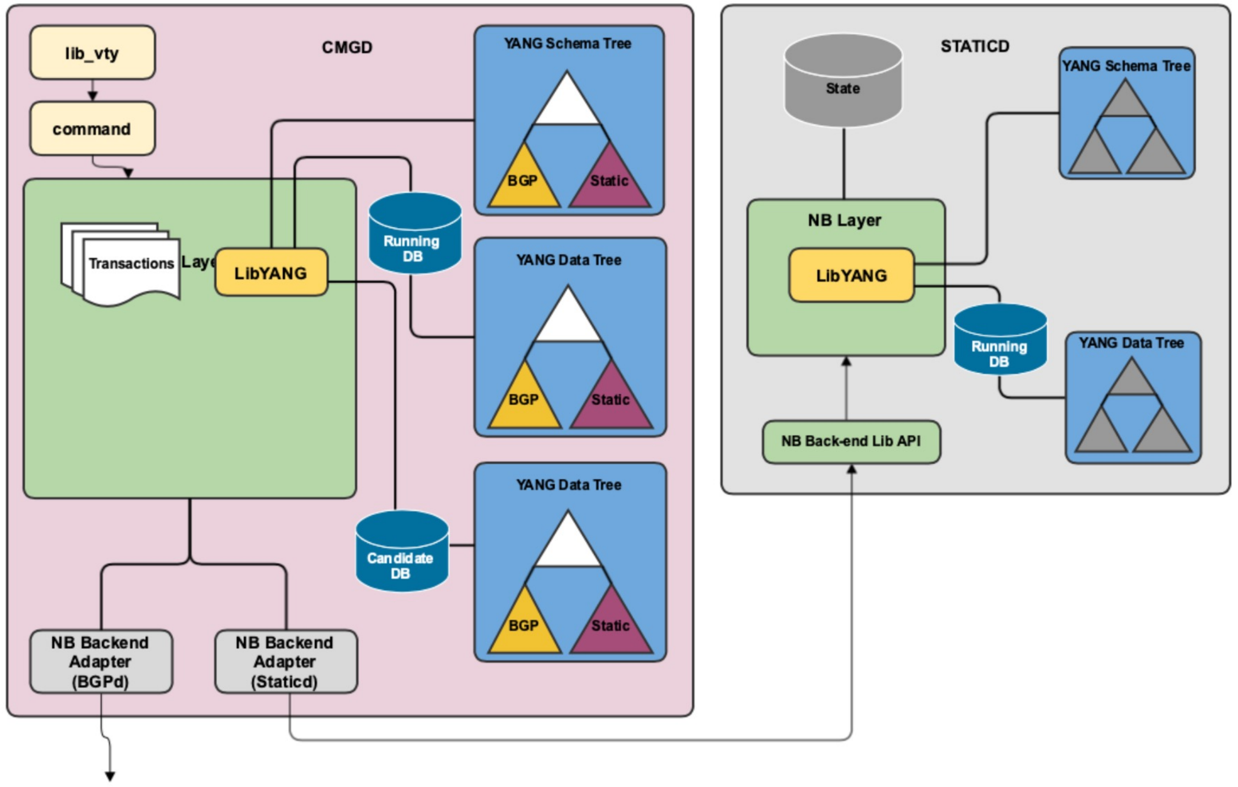
Processing Configuration

Proposal 1 (Initial Approach)

- **PHASE-1: Converting Text Commands to Yang Xpaths** – VTYSH only
  - See later slides for discussion.
- **PHASE-2: Validating the XPATH and corresponding value against Yang Schema**
  - Anyhow will have to be moved to MGMTD daemon.
  - Access to Yang Schema tree and Data tree (Running DB) can be provided on MGMTD itself.
- **PHASE-3: Validating the corresponding value against current configuration**
  - Triggered through Backend client library using message exchange between MGMTD and backend.
  - Executed through 'create/modify/destroy' callbacks on backend daemon.
  - Validation against current configuration
    - Maintain a duplicate Yang Data sub-tree (Running DB) on backend daemon.
    - Or, maintain a shadow copy of configuration on internal storage. Most daemons do that.
- **PHASE-4: Applying the corresponding value on the backend internal state**
  - Executed through 'create/modify/destroy' callbacks on backend daemon.
  - Needs to access backend internal state.
  - Must not need access to Yang Data tree.
  - Triggered through Backend client library using message exchange between MGMTD and backend.

Sharing Management Data

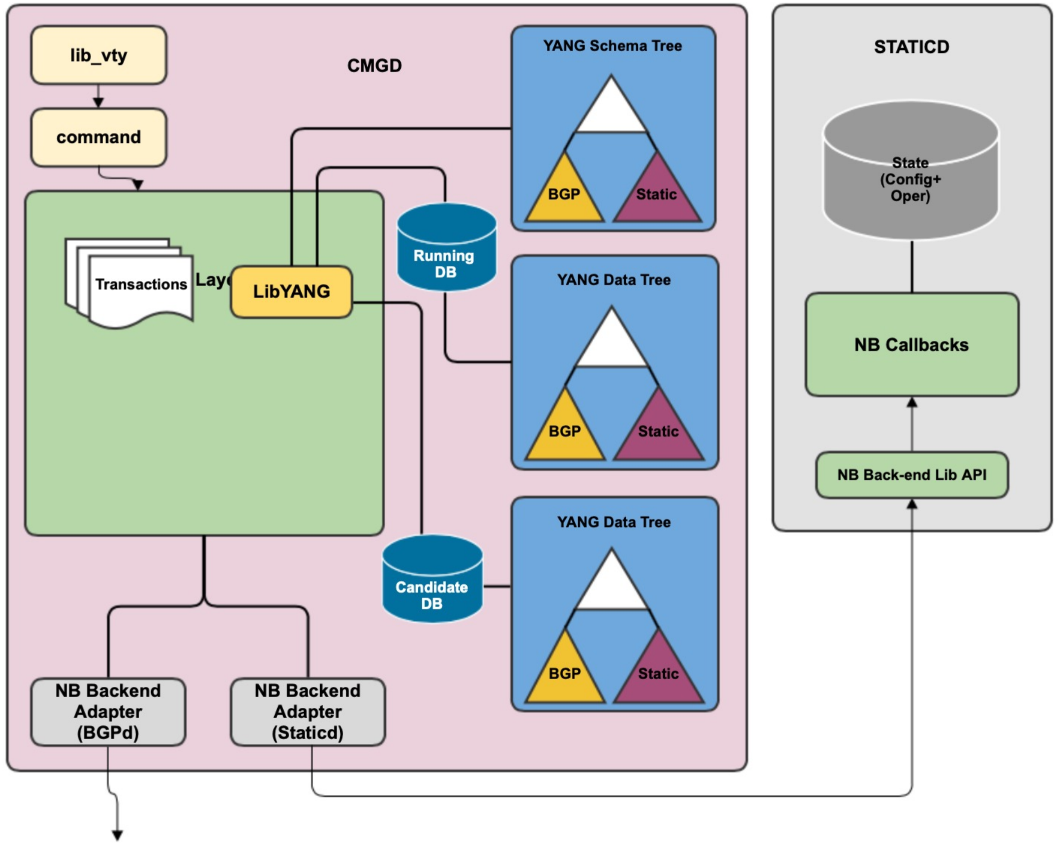
Proposal 1: Option1 (Initial Approach)



- Option-1
  - MGMTD daemon maintain
    - Yang Schema Tree
    - Yang Data Tree
      - Running Db
      - Candidate Db
  - Backend daemon maintain
    - Duplicate Yang Schema Tree
    - Duplicate Yang Data subtree copy
    - Internal state.
  - MGMTD loads modules written by backend to map text commands to corresponding XPath.
    - But they MUST NOT have dependency on internal state of the backend daemon.
  - MGMTD and Backend daemon exchange Xpath and data in native format (using Protobufs)
    - For Validation (In proposal 1 only)
    - For Applying config.

Sharing Management Data

Proposal-1 Option2 and Proposal-2 (Final goal)



- Option-2
  - MGMTD daemon maintain
    - Yang Schema Tree
    - Yang Data Tree
      - Running Db
      - Candidate Db
  - Backend daemon maintain
    - Internal state only.
  - MGMTD loads modules written by backend to map text commands to corresponding XPath.
    - But they MUST NOT have dependency on internal state of the backend daemon.
  - MGMTD and Backend daemon exchange Xpath and data in native format (using Protobufs)
    - For Validation (In proposal 1 only)
    - For Applying config.

Processing Configuration

Proposal 2 (Incremental on-need basis)

- **PHASE-1: Converting Text Commands to Yang Xpaths** – VTYSH only
  - See later slides for discussion.
- **PHASE-2: Validating the XPATH and corresponding value against Yang Schema**
  - Anyhow will have to be **moved to MGMTD daemon**.
  - Access to Yang Schema tree and Data tree (Running DB) can be provided on MGMTD itself.
- **PHASE-3: Validating the corresponding value against current configuration**
  - Executed through 'create/modify/destroy' **callbacks on (and hence to be loaded on) MGMTD daemon**.
  - **Validation against** current configuration in **Yang Data tree (Running DB) on MGMTD daemon only**.
  - **No message exchange required** between MGMTD and backend.
- **PHASE-4: Applying the corresponding value on the backend internal state**
  - Executed through 'create/modify/destroy' callbacks on backend daemon.
  - Needs to access backend internal state.
  - **Must not access Yang Data tree**.
  - Triggered through Backend client library using **message exchange between MGMTD and backend**.

Processing Show Commands

Processing Show Commands

Sub-Phases and Requirements

- **PHASE-1: Converting Text Commands to Yang Xpaths** – VTYSH only
  - Currently implemented and executed on backend daemon.
- **PHASE-2: Validating the XPATHs against Yang Schema**
  - Again executed on backend daemon.
  - Needs access to Yang Schema tree and Data tree (Running DB).
- **PHASE-3: Fetching the corresponding value from the backend internal state**
  - Again executed backend daemon.
  - MUST need access to backend internal state.

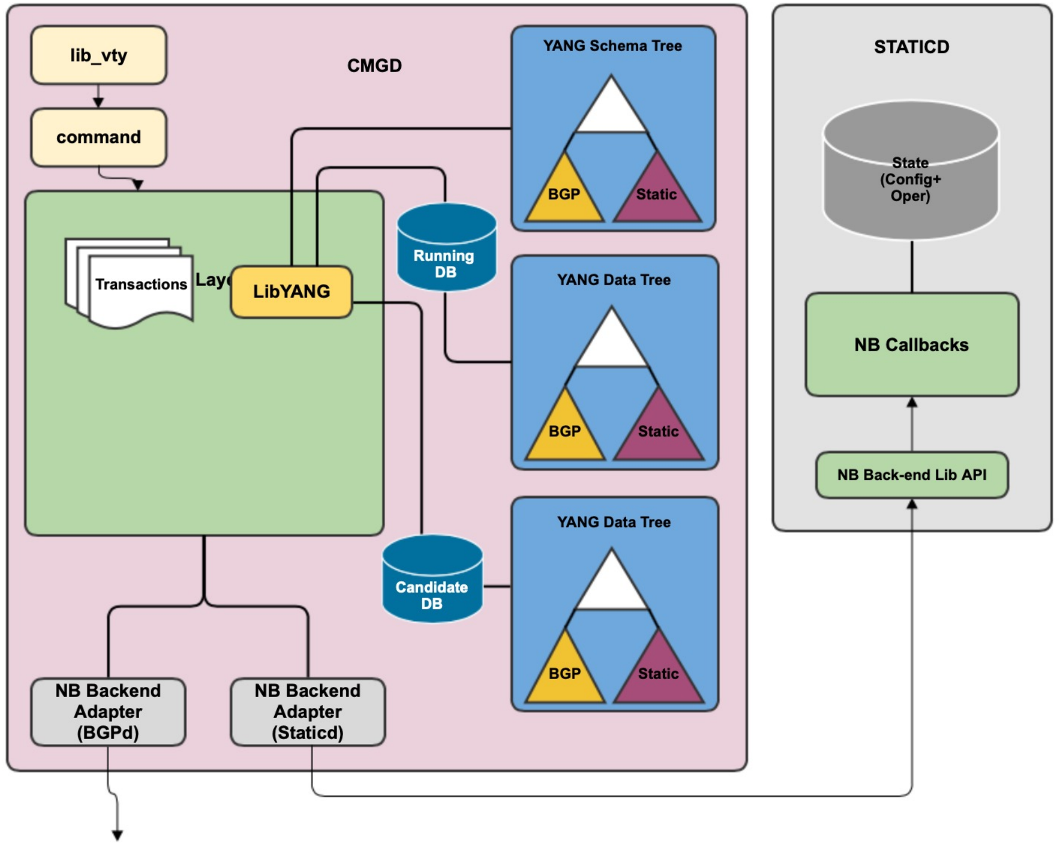
Processing Show Commands

Proposal

- **Step-1: Converting Text Commands to Yang Xpaths** – VTYSH only
  - Has to be somehow **moved to MGMTD daemon**.
- **Step-2: Validating the XPATH and corresponding value against Yang Schema**
  - Anyhow will have to be **moved to MGMTD daemon**.
  - Access to Yang Schema tree and Data tree (Running DB) can be provided on MGMTD itself.
- **Step-3: Fetching the corresponding value from the backend internal state**
  - Executed through 'get-data/get-elem' callbacks on backend daemon.
  - Needs to access backend internal state.
  - **Must not access Yang Data tree**.
  - Triggered through Backend client library using **message exchange between MGMTD and backend**.

Sharing Management Data

Proposal-2



- MGMTD daemon maintain
  - Yang Schema Tree
  - Yang Data Tree
    - Running Db
    - Candidate Db
- Backend daemon maintain
  - Internal state only needed.
- MGMTD loads modules written by backend to map text commands to corresponding XPath.
  - But they MUST NOT have dependency on internal state of the backend daemon.
- MGMTD and Backend daemon exchange Xpath and data in native format (using Protobufs)
  - For fetching value from Internal state.

Parsing YANG Data

Parsing and passing Yang data

Current method

- Yang data stored in Application Daemon
- Parsed from string to native format on receiving from VTYSH
- Stored as native binary format in Yang Data Tree (lyd_term_node::value)
- Converted back to string while replying back to VTYSH.

Parsing and passing Yang data

New MGMTD Architecture

- Yang data stored in MGMTD Daemon
- Proposal 1
  - Parsed from string to native data format on Frontend client.
  - Passed as native data format on Frontend interface to MGMTD.
  - Stored as native data format in Yang Data Tree on MGMTD.
  - Passed to and back from application daemon as native data format over Backend interface.
  - Passed as native data format on Frontend interface back to Frontend client.
  - Converted back to string on Frontend client.

Parsing and passing Yang data

New MGMTD Architecture

- Yang data stored in MGMTD Daemon
- Proposal 2 (community preferred approach)
  - Passed as string on Frontend interface to MGMTD.
  - Parsed from string to native data format on MGMTD.
  - Stored as native data format in Yang Data Tree on MGMTD.
  - Passed to and back from application daemon as native data format over Backend interface.
  - Converted back to string on MGMTD to send for replying to Frontend client.
  - Passed as string format on Frontend interface back to Frontend client.

Applying configurations on Backend

Applying Config on Backend

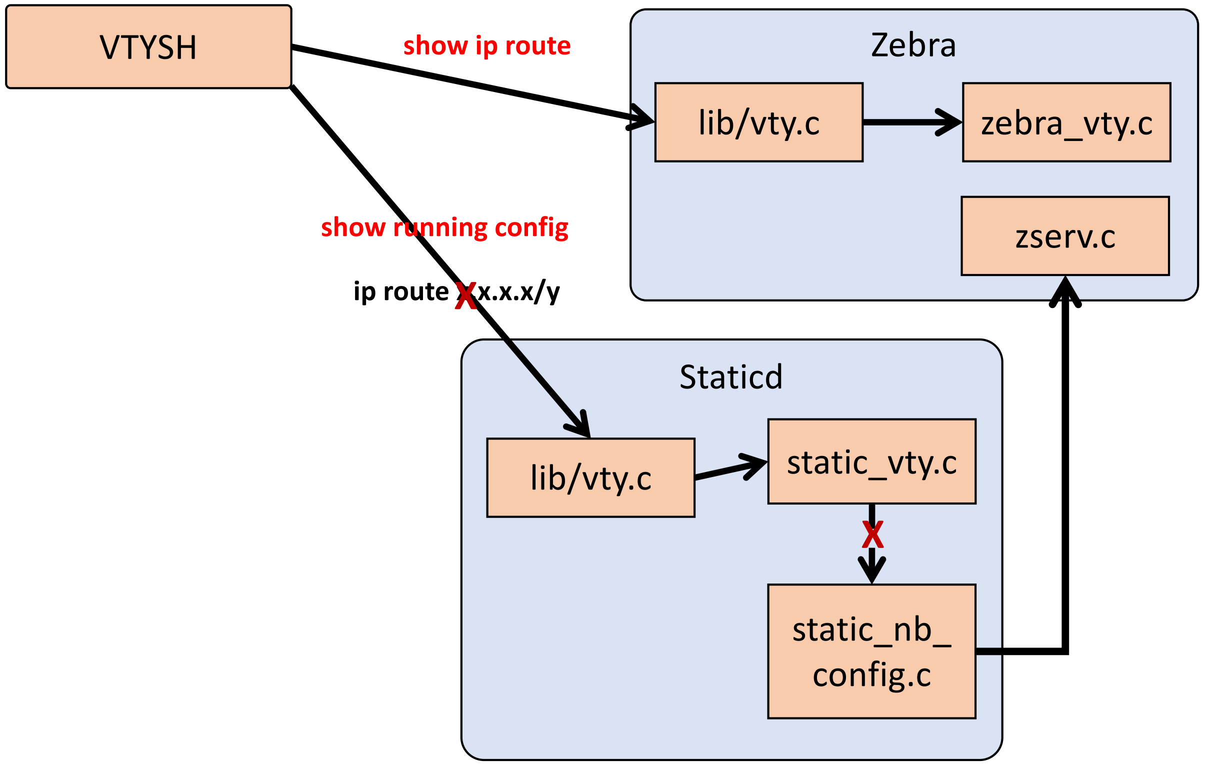
Offline processing of CFR_APPLY_REQ

- Config is pre-validated on MGMTD.
  - No need to wait for APPLY on Backend for Commit to complete.
- CFG_APPLY_REQ is queued for processing later.
  - Pros
    - Commit can return much earlier.
    - Configuration apply can be spread over time to let other job on the same thread.
    - Or can be packed into a much lesser batches than what was possible with inline processing.
  - Cons
    - If and only if apply fails, the backend becomes out-of-sync with Config on MGMTD.
    - How to recover?
      - Perhaps restart the backend daemon and re-download config on restart. ???

Handling VTYSH commands

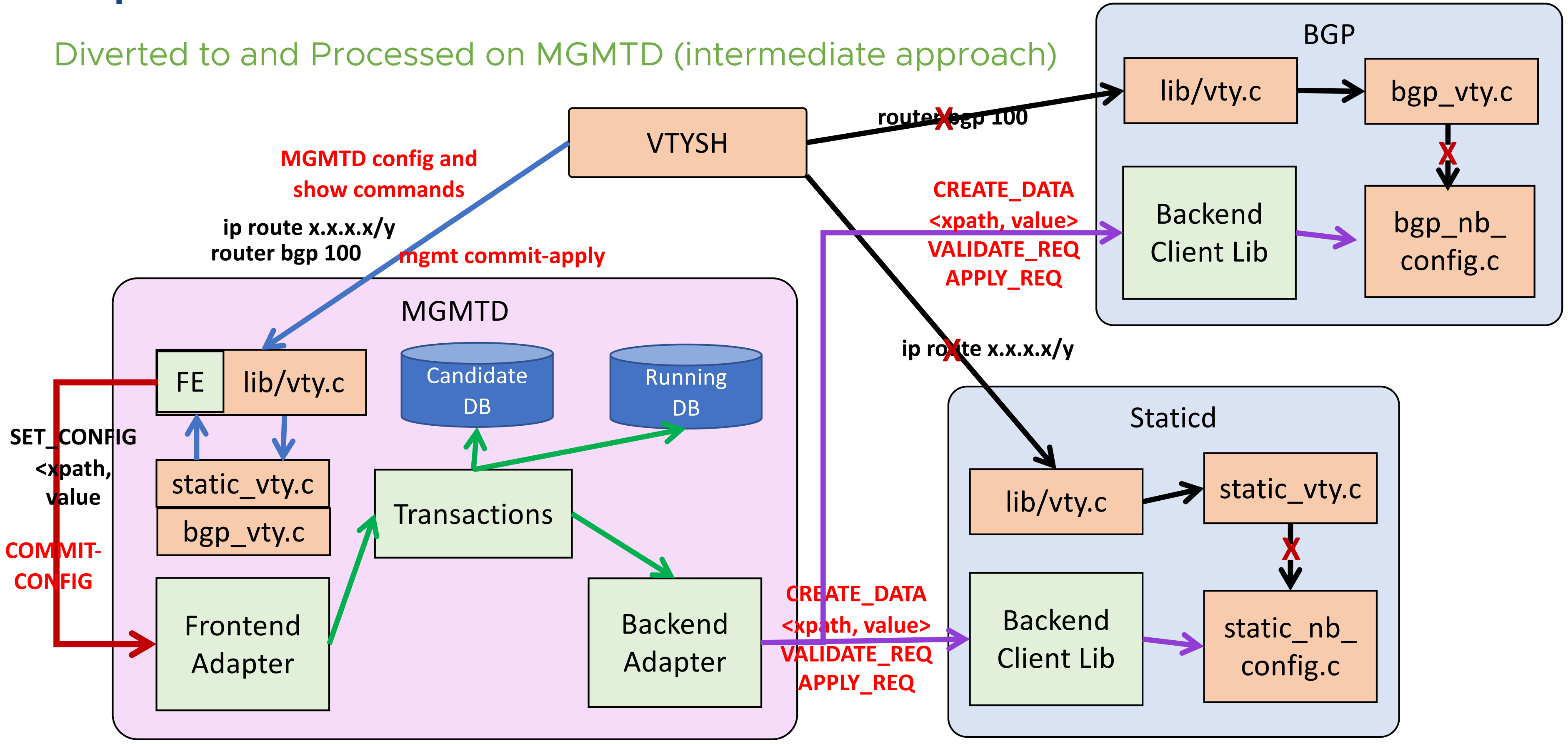
Current Method

Current Config Commands Processing



Option 1

Diverted to and Processed on MGMTD (intermediate approach)



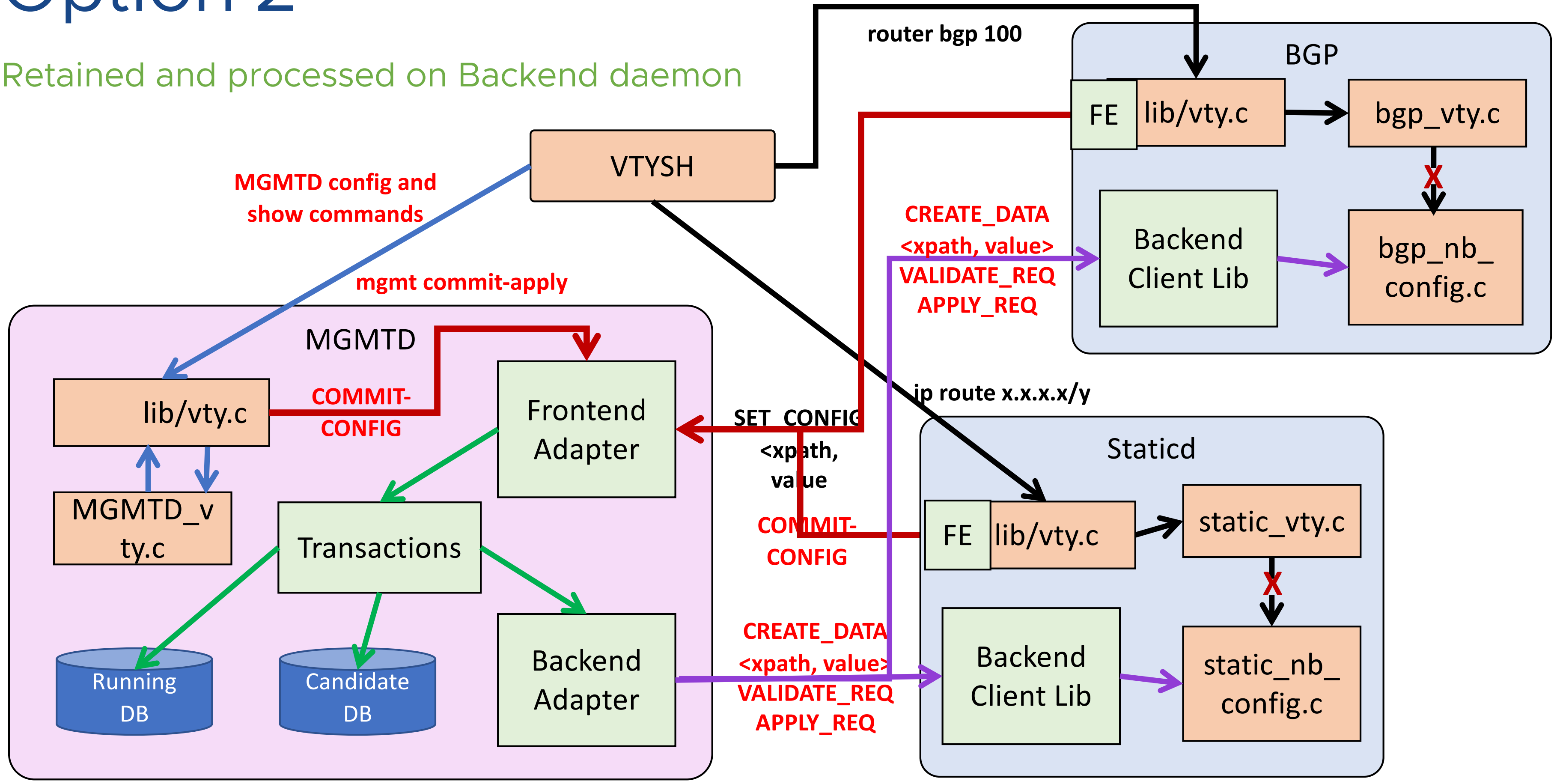
Option 1

Diverted to and Processed on MGMTD (interim approach)

- Command handlers lib-ified and loaded on MGMTD
  - Cannot access any internal state of the Back-end application daemon.
  - Passes set of Xpaths, values and operations to MGMTD Front-end connection.
- Pros
  - Single connection from VTYSH to MGMTD (and not multiple backend daemons).
  - Conversion from string commands to Xpaths/Value/Operation is no more a computational burden on the back-end application daemon.
- Cons
  - Auto-completion cannot be done as is today.
    - Needs to fetch data from MGMTD (running-config or operational).
    - And then present auto-completion options.

Option 2

Retained and processed on Backend daemon



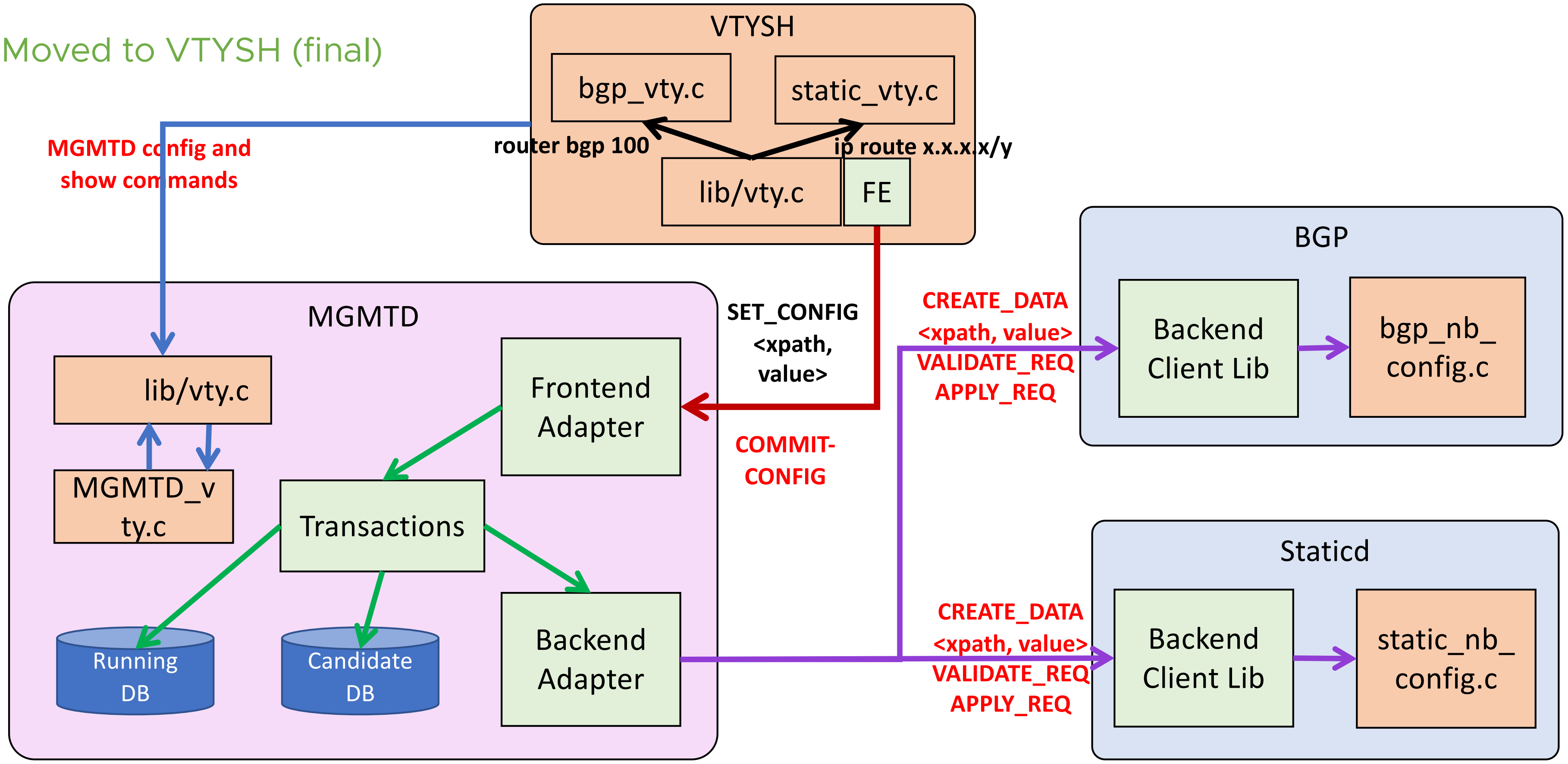
Option 2

Retained and processed on Backend daemon (not feasible)

- Command handlers running on Back-end daemons as is today
  - Can access any internal state of the Back-end application daemon.
  - Passes set of Xpaths, values and operations to MGMTD Front-end connection.
- Pros
  - Auto-completion can be done as is today.
- Cons
  - Conversion from string commands to Xpaths/Value/Operation is still a burden on the back-end application daemon.
  - VTYSH continues to maintain connections to all individual back-end application daemons.
  - MGMTD currently does not allow more than one front-end client to edit candidate DB simultaneously. NOT FEASIBLE.

Option 3

Moved to VTYSH (final)



Option 3

Moved to VTYSH

- Command handlers lib-ified and loaded on VTYSH
  - Cannot access any internal state of the Back-end application daemon.
  - Passes set of Xpaths, values and operations to MGMTD Front-end connection.
- Pros
  - Single connection from VTYSH to MGMTD (and not multiple backend daemons).
  - Conversion from string commands to Xpaths/Value/Operation is no more a computational burden on the back-end application daemon.
  - MGMTD does not need to deal with Text-to-YANG conversion at all.
- Cons
  - Auto-completion cannot be done as is today.
    - Needs to fetch data from MGMTD (running-config or operational).
    - And then present auto-completion options.

Parsing YANG Xpath

Parsing and passing Yang Xpath

Problem Statement

- Currently
  - Yang data and schema trees stored in Application Daemon.
  - Xpath in string format parsed using Yang Schema and Data trees.
    - The same is used to retrieve Xpath key-values within NB callback functions (e.g. vrf-name, interface-name, address-families etc).
    - Sometime involves looking up nodes in the Yang data tree.
- New MGMTD Infrastructure
  - Yang data and schema trees stored in MGMTD.
  - Xpath parsing is possible on MGMTD.
  - How to provide Xpath key-values to NB callbacks on application daemon
    - Without needing to look up nodes in the Yang data tree.

Parsing and passing Yang Xpath

Possible Solution

- Solution 1 (Interim approach)
  - Always maintain a Yang schema and data tree on application daemons too.
  - Have NB callbacks keep looking up data nodes as today.
- Solution 2 (Final goal)
  - Parse Xpath into tokens (of tags and key-values).
    - May not have equivalent support from libyang.
  - Pass it to NB callbacks for use (instead of looking up data nodes in Yang data tree).

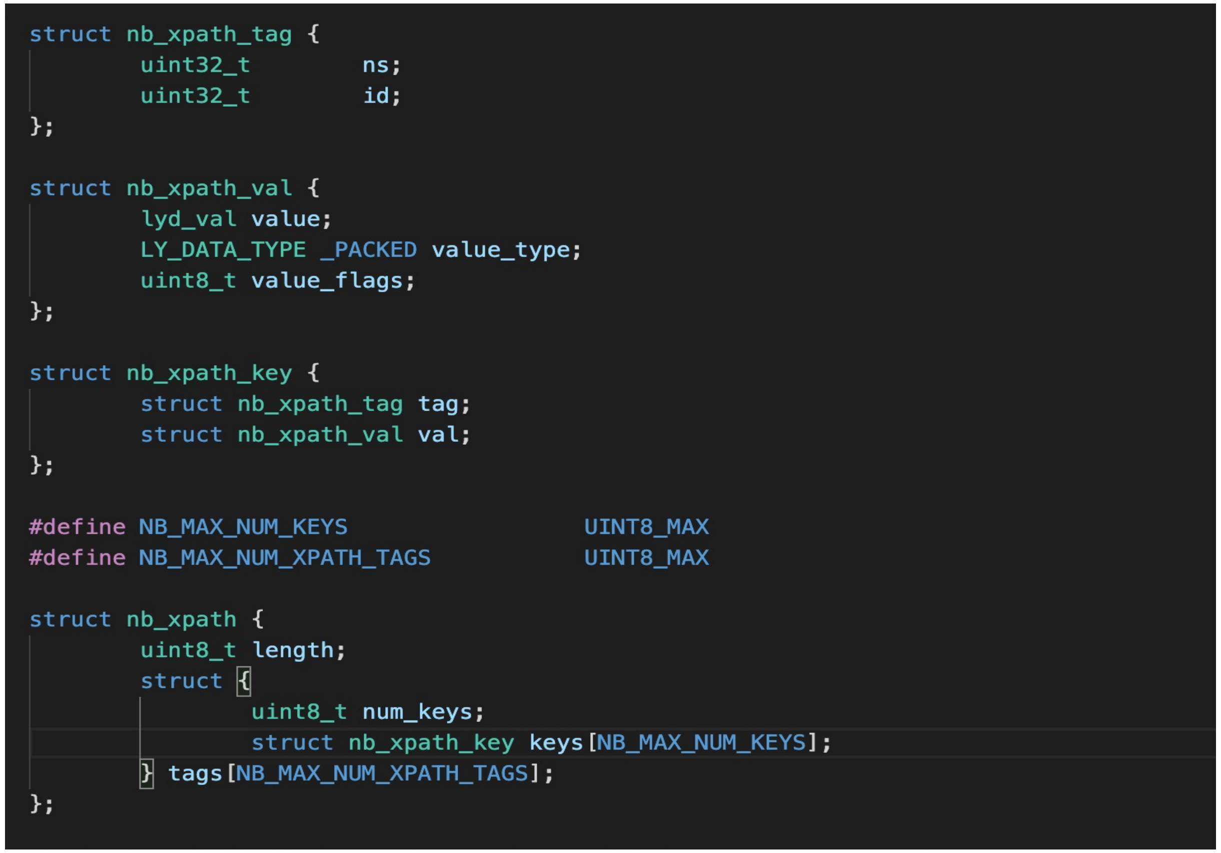
Parsing YANG XPath

XPATH: /frr-routing:routing/control-plane-protocols/control-plane-protocol{frr-rt:type='frr-bgp:bgp', name='bgp-100', frr-vrf:vrf='default'}/frr-bgp:bgp/global/frr-rt:router-id

length = 7	num_keys	keys[0]	keys[1]	keys[2]	...	keys[255]
tags[0]	1	frr-routing | routing				
tags[1]	1	frr-routing | control-plane-protocols				
tags[2]	1	frr-routing | control-plane-protocol				
tags[3]	3	frr-routing | **type =** **"bgp"**	frr-routing | **name =** **"bgp-100"**	frr-routing | **Vrf** = **"default"**		
tags[4]	1	frr-bgp | bgp				
tags[5]	1	Frr-bgp | global				
tags[6]	1	frr-bgp | **router-id** = **"1.1.1.1"**				
...						
Tags[255]						

- <Notes TBA>

Parsing YANG XPath


```
struct nb_xpath_tag {
        uint32_t        ns;
        uint32_t        id;
};

struct nb_xpath_val {
        lyd_val value;
        LY_DATA_TYPE _PACKED value_type;
        uint8_t value_flags;
};

struct nb_xpath_key {
        struct nb_xpath_tag tag;
        struct nb_xpath_val val;
};

#define NB_MAX_NUM_KEYS                 UINT8_MAX
#define NB_MAX_NUM_XPATH_TAGS           UINT8_MAX

struct nb_xpath {
        uint8_t length;
        struct {
                uint8_t num_keys;
                struct nb_xpath_key keys[NB_MAX_NUM_KEYS];
        } tags[NB_MAX_NUM_XPATH_TAGS];
};
```

- <Notes TBA>

Parsing YANG XPath

	num_keys	keys[0]	keys[1]	keys[2]	…	keys[255]

tags[0]	1	.tag = {
		  .ns = frr-routing,
		  .id = **routing**
		}
		.val = {
		  .xml_tag =
		    "routing"
		}				

…						

tags[3]	3	.tag = {
		  .ns = frr-rt,
		  .id = **type**
		}
		.val = {
		  .identityref =
		    ”bgp"
		}
			.tag = {
			  .ns = frr-rt,
			  .id = **name**
			}
			.val = {
			  .string =
			    ”bgp-100"
			}
				.tag = {
				  .ns = frr-vrf,
				  .id = **vrf**
				}
				.val = {
				  .string = **”default”**
				}		

…						

tags[6]	1	tag = {
		  .ns = frr-rt,
		  .id = **router-id**
		}
		.val = {
		  .ipv4_addr =
		    ”1.1.1.1"
		}				

Static Mapping

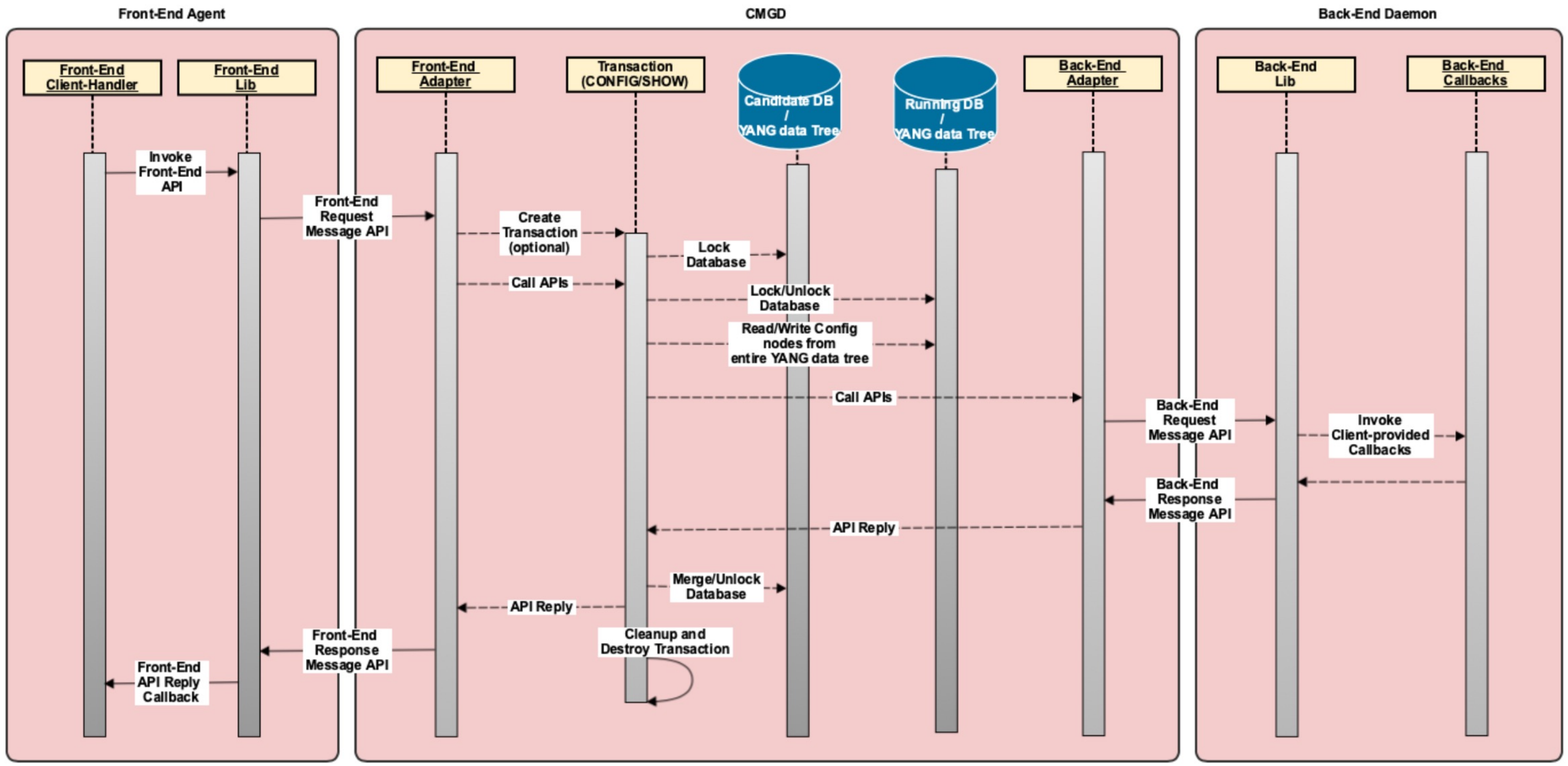
XPATH:
/frr-routing:routing/control-plane-protocols/control-plane-protocol{frr-rt:type='frr-bgp:bgp', name='bgp-100', frr-vrf:vrf='default'}/frr-bgp:bgp/global/frr-rt:router-id

Xpath Regexp	Backend Adapter Name
“/frr-routing:routing/control-plane-protocols/control-plane-protocol{frr-rt:type=“frr-bgp”', name=‘*’, frr-vrf:vrf=‘*’}/*”	“BGPd”
“/frr-system:system/hostname”	“ISISd”, “OSPd”, “Systemd”
“/frr-system:system/*”	“Systemd”
tags[3]	3
tags[4]	1
tags[5]	1
tags[6]	1
…	
Tags[255]	

- <Notes TBA>

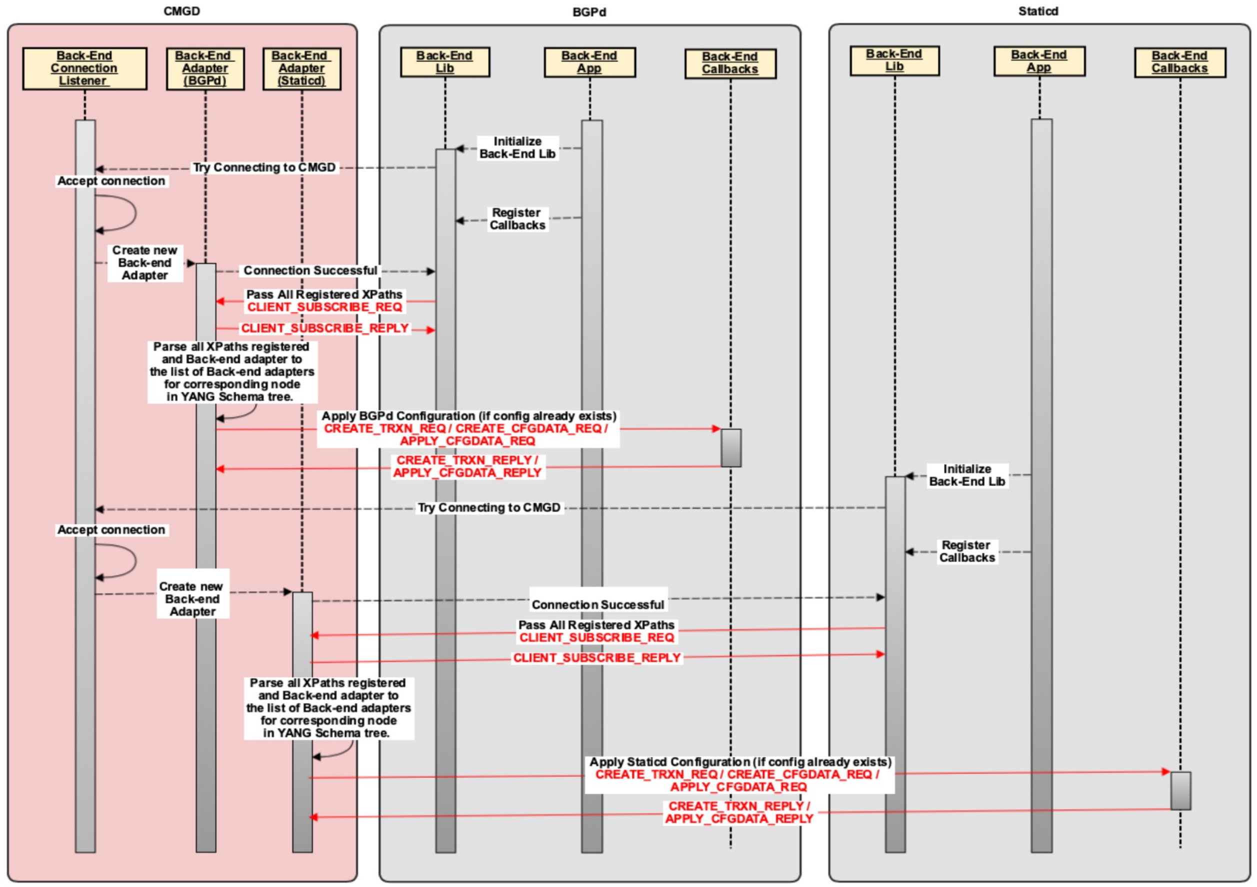
End-to-End Processing

High-level End-to-End Processing Flow


Front-End Agent
CMGD
Back-End Daemon
Front-End Client-Handler
Front-End Lib
Front-End Adapter
Transaction (CONFIG/SHOW)
Candidate DB / YANG data Tree
Running DB / YANG data Tree
Back-End Adapter
Back-End Lib
Back-End Callbacks
Invoke Front-End API
Front-End Request Message API
Create Transaction (optional)
Lock Database
Call APIs
Lock/Unlock Database
Read/Write Config nodes from entire YANG data tree
Call APIs
Back-End Request Message API
Invoke Client-provided Callbacks
Back-End Response Message API
API Reply
Merge/Unlock Database
API Reply
Front-End Response Message API
Cleanup and Destroy Transaction
Front-End API Reply Callback

Back-end Initialization and Registration

(Re)Connecting to MGMTD

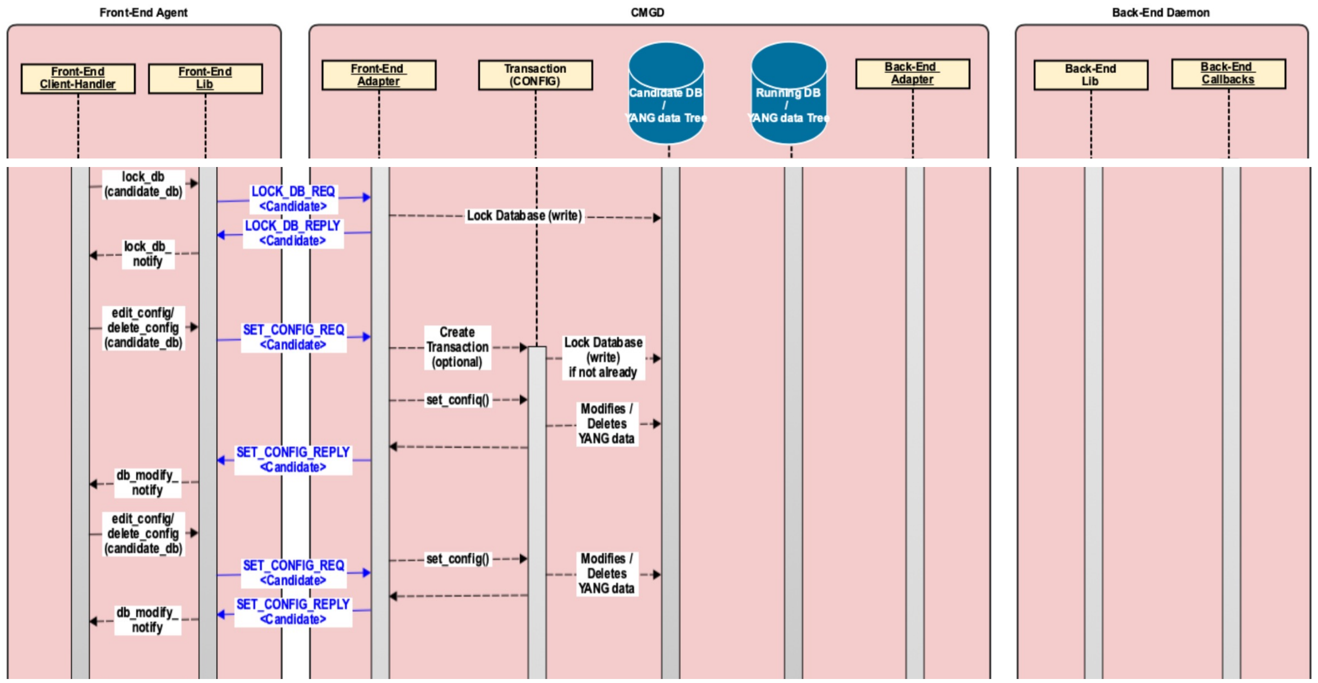


CMGD
BGPd
Staticd
Back-End Connection Listener
Back-End Adapter (BGPd)
Back-End Adapter (Staticd)
Back-End Lib
Back-End App
Back-End Callbacks
Initialize Back-End Lib
Try Connecting to CMGD
Accept connection
Register Callbacks
Create new Back-end Adapter
Connection Successful
Pass All Registered XPaths
CLIENT_SUBSCRIBE_REQ
CLIENT_SUBSCRIBE_REPLY
Parse all XPaths registered and Back-end adapter to the list of Back-end adapters for corresponding node in YANG Schema tree.
Apply BGPd Configuration (if config already exists)
CREATE_TRXN_REQ / CREATE_CFGDATA_REQ / APPLY_CFGDATA_REQ
CREATE_TRXN_REPLY / APPLY_CFGDATA_REPLY
Apply Staticd Configuration (if config already exists)

- Back-end daemon re/init
  - Initializes NB back-end library
    - Provides unique details identifying itself uniquely.
    - Triggers connection initiation towards MGMTD.
  - Registers required callbacks for the YANG models it is interested in.
    - Registers a series of callback handlers for individual/group of configuration and operational data items.
    - NB back-end lib passes the same to MGMTD.
- MGMTD
  - Creates new Back-end adapter.
    - Associates it with relevant parts of the YANG data tree.
  - Passes any relevant configuration already present.

Processing Configuration using Transactions

Step-1: Locking Candidate Database and Editing Configuration on Candidate DB

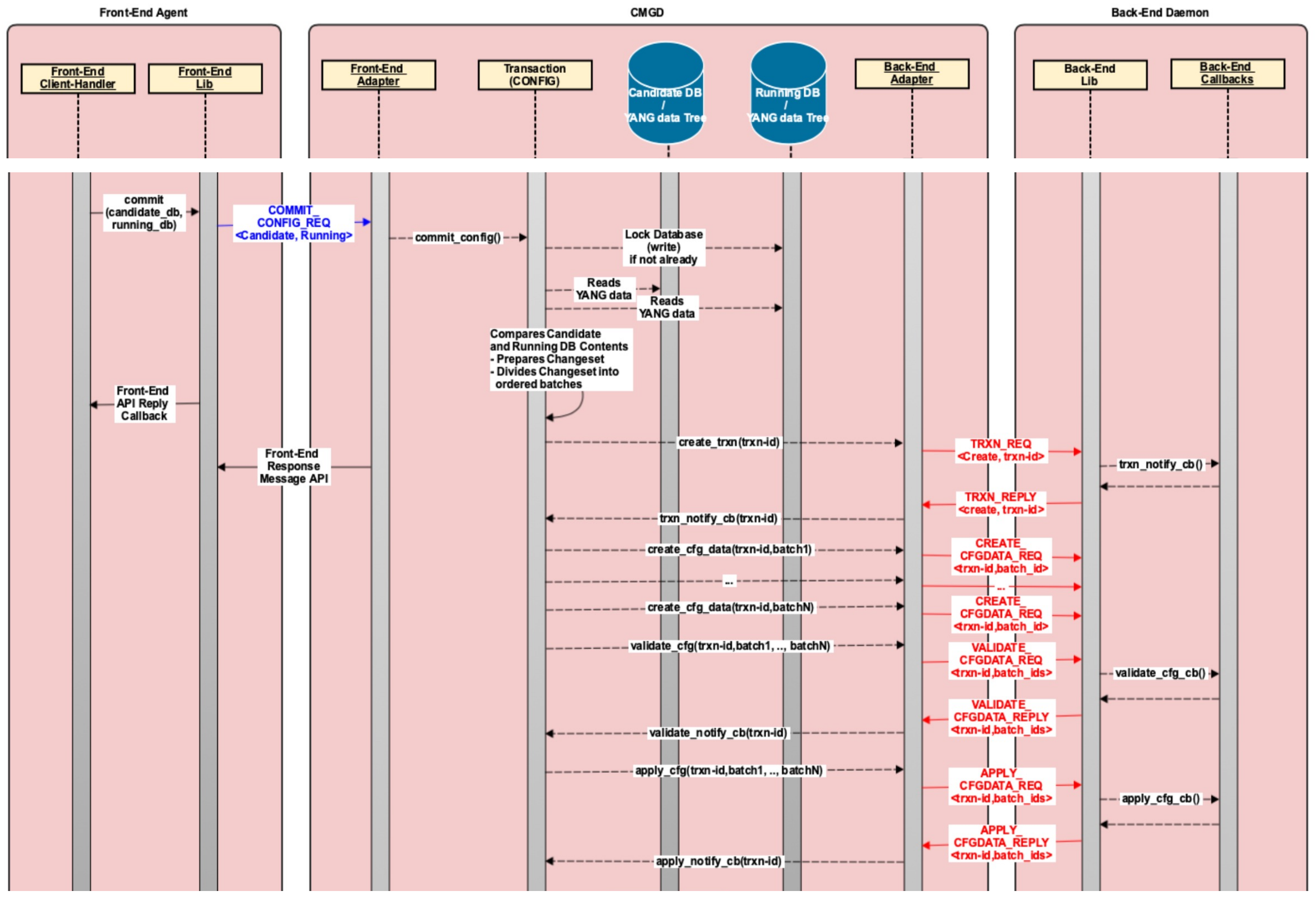


Front-End Agent
CMGD
Back-End Daemon
Front-End Client-Handler
Front-End Lib
Front-End Adapter
Transaction (CONFIG)
Candidate DB / YANG data Tree
Running DB / YANG data Tree
Back-End Adapter
Back-End Lib
Back-End Callbacks
lock_db (candidate_db)
LOCK_DB_REQ <Candidate>
Lock Database (write)
LOCK_DB_REPLY <Candidate>
lock_db_notify
edit_config/ delete_config (candidate_db)
SET_CONFIG_REQ <Candidate>
Create Transaction (optional)
Lock Database (write) if not already
set_config()
Modifies / Deletes YANG data
SET_CONFIG_REPLY <Candidate>
db_modify_notify
edit_config/ delete_config (candidate_db)
SET_CONFIG_REQ <Candidate>
set_config()
Modifies / Deletes YANG data
SET_CONFIG_REPLY <Candidate>
db_modify_notify

- MGMTD receives LOCK_DEB_REQ for candidate DB.
  - If no other session has taken a write-lock on Candidate-Db
    - Take write-lock on Candidate-Db.
    - Sends LOCK_DB_REPLY indicating success.
  - Else
    - Sends LOCK_DB_REPLY indicating Failure.
- Client on receiving LOCK_DB_REPLY with success
  - Sends one (or more) SET_CONFIG_REQ
- MGMTD on receiving SET_CONFIG_REQ
  - If this is first for this session, Creates a CONFIG transaction if and only if there are no CONFIG transactions in progress.
  - Modifies the contents of the Candidate DB with data from SET_CONFIG_REQ.

Processing Configuration using Transactions

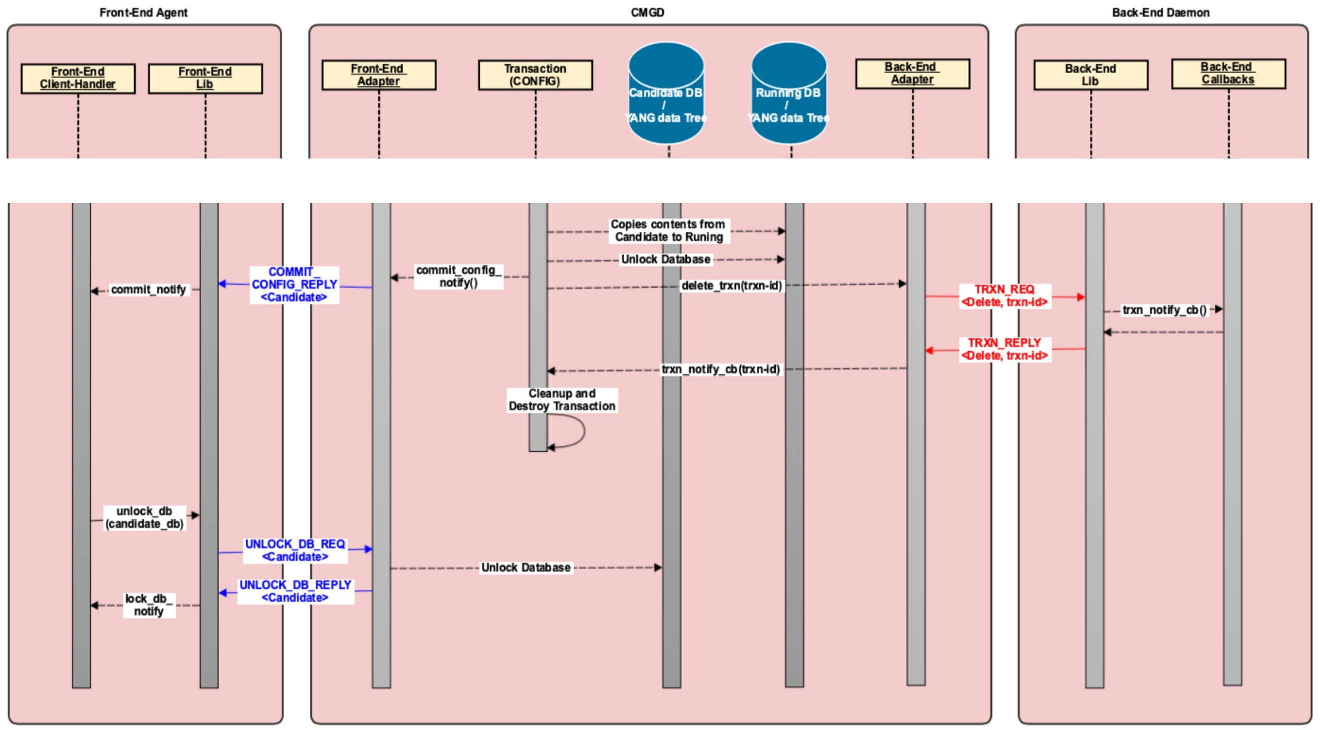
Step-2: Commit Configuration from Candidate to Running DB


Front-End Agent
CMGD
Back-End Daemon
Front-End Client-Handler
Front-End Lib
Front-End Adapter
Transaction (CONFIG)
Candidate DB / YANG data Tree
Running DB / YANG data Tree
Back-End Adapter
Back-End Lib
Back-End Callbacks
commit (candidate_db, running_db)
COMMIT_CONFIG_REQ <Candidate, Running>
commit_config()
Lock Database (write) if not already
Reads YANG data
Reads YANG data
Compares Candidate and Running DB Contents - Prepares Changeset - Divides Changeset into ordered batches
Front-End API Reply Callback
Front-End Response Message API
create_trxn(trxn-id)
TRXN_REQ <Create, trxn-id>
trxn_notify_cb()
TRXN_REPLY <create, trxn-id>
trxn_notify_cb(trxn-id)
create_cfg_data(trxn-id,batch1)
CREATE_CFGDATA_REQ <trxn-id,batch_id>
...
create_cfg_data(trxn-id,batchN)
CREATE_CFGDATA_REQ <trxn-id,batch_id>
validate_cfg(trxn-id,batch1, .., batchN)
VALIDATE_CFGDATA_REQ <trxn-id,batch_ids>
validate_cfg_cb()
VALIDATE_CFGDATA_REPLY <trxn-id,batch_ids>
validate_notify_cb(trxn-id)
apply_cfg(trxn-id,batch1, .., batchN)
APPLY_CFGDATA_REQ <trxn-id,batch_ids>
apply_cfg_cb()
APPLY_CFGDATA_REPLY <trxn-id,batch_ids>
apply_notify_cb(trxn-id)

- MGMTD receives commit request.
  - Creates transaction for entire processing
    - Examines the changeset and prepares.
      - Candidate DB andl ist of YANG tree nodes being changed.
      - List of back-end adapters associated.
      - Ordered batches of config nodes with associated back-end adapter(s)
    - Sends indication to each back-end daemon for creating local transaction context.
    - Sends each batch of config items to corresponding back-end daemons for verification/validation.
    - On successful validation from all
      - Pushes the same batches of config items to the relevant back-end daemon(s) for final apply.
      - Replies back to Front-end adapter with success
      - Merges candidate DB to running DB.
      - Cleans up and deletes the transaction itself.

Processing Configuration using Transactions

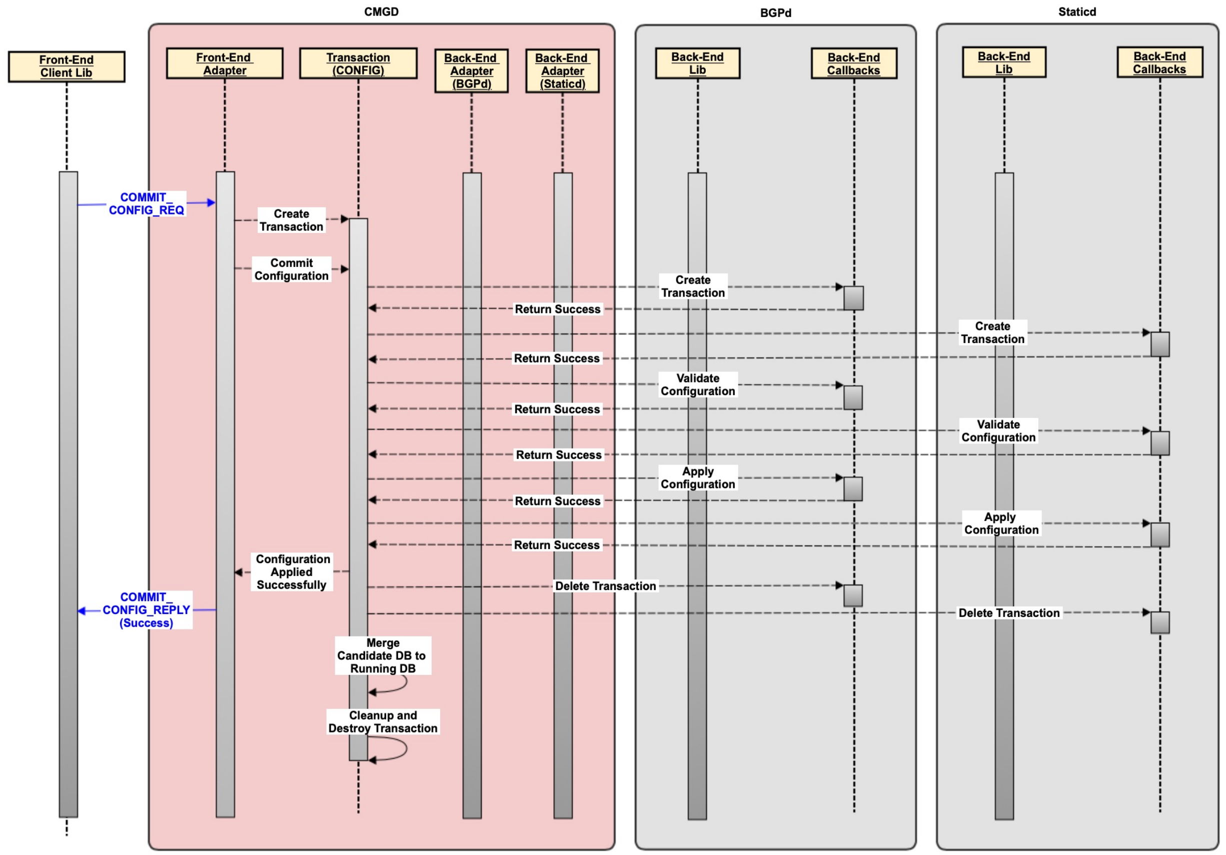
Step-3: Cleaning up commit and Unlocking the Candidate DB



- Merges candidate DB to running DB.
- Replies back to Front-end client with COMMIT_CONFIG_REPLY indicating success.
- Cleans up and deletes the transaction itself.

Processing Configuration using Transactions

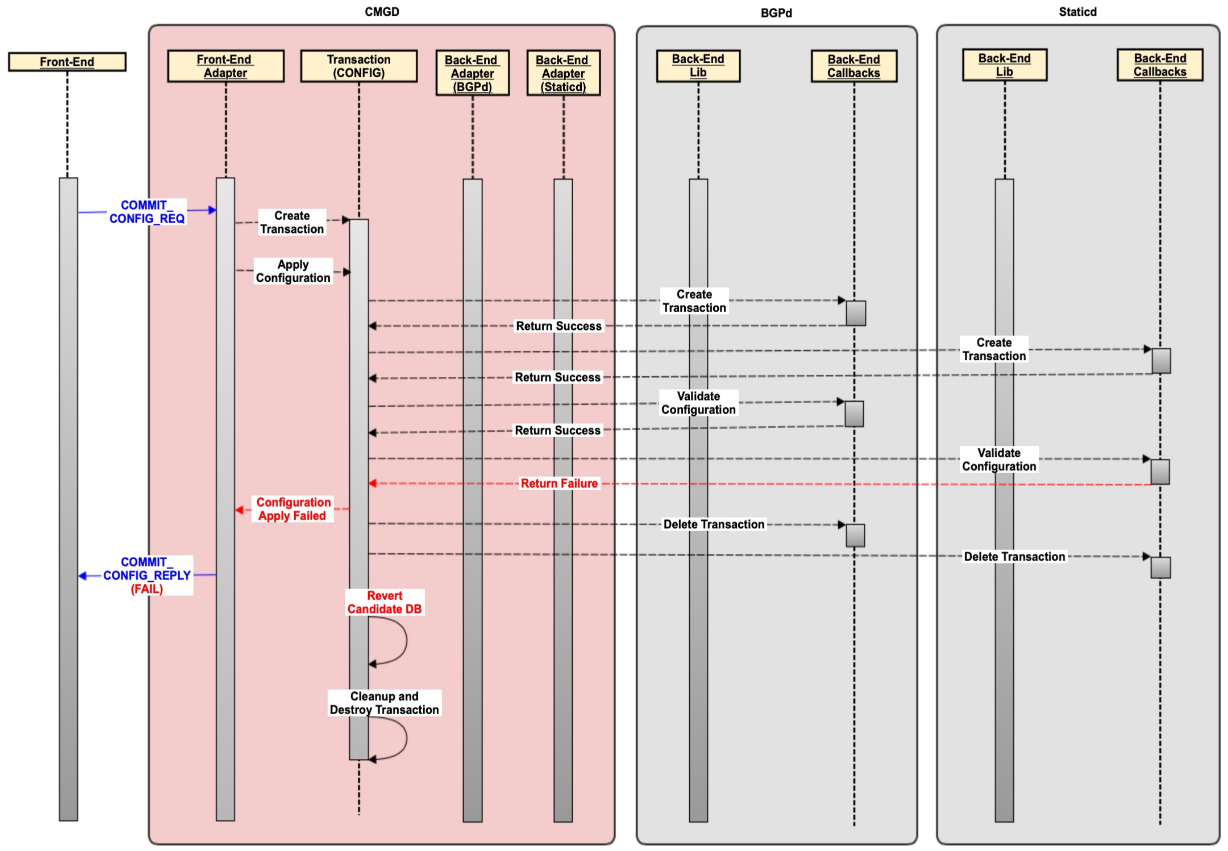
Handling Valid Configurations across multiple Backend Daemons



- MGMTD receives commit request.
  - Creates transaction for entire processing
    - Examines the changeset and prepares.
      - Candidate DB andl ist of YANG tree nodes being changed.
      - List of back-end adapters associated.
      - Ordered batches of config nodes with associated back-end adapter(s)
    - Sends indication to each back-end daemon for creating local transaction context.
    - Sends each batch of config items to corresponding back-end daemons for verification/validation.
    - On successful validation from all
      - Pushes the same batches of config items to the relevant back-end daemon(s) for final apply.
      - Replies back to Front-end adapter with success
      - Merges candidate DB to running DB.
      - Cleans up and deletes the transaction itself.

Processing Configuration using Transactions

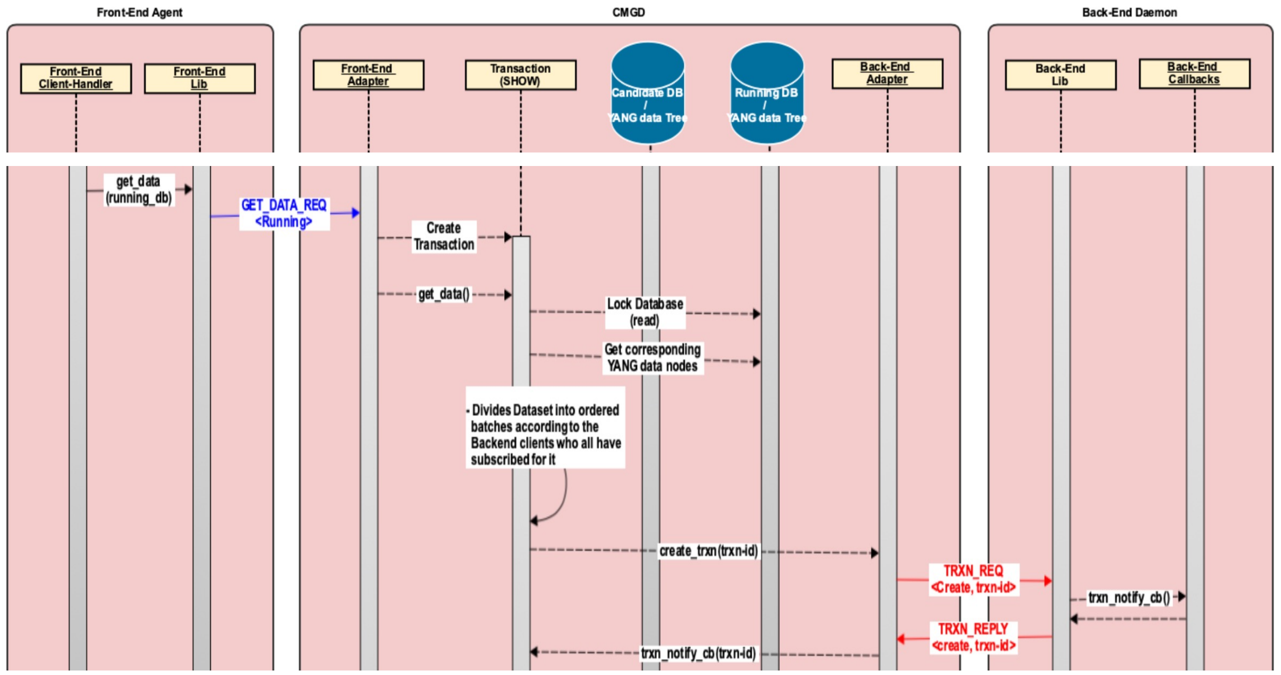
Handling Invalid Configurations across multiple Backend Daemons



- MGMTD receives commit request.
  - Creates transaction thread for entire processing
    - Examines the changeset and prepares.
      - Candidate DB andl ist of YANG tree nodes being changed.
      - List of back-end adapters associated.
      - Ordered batches of config nodes with associated back-end adapter(s)
    - Sends indication to each back-end daemon for creating local transaction context.
    - Sends each batch of config items to corresponding back-end daemons for verification/validation.
    - On first validation failure from any
      - Replies back to Front-end adapter with error.
      - Deletes candidate DB.
      - Cleans up and deletes the transaction thread itself.

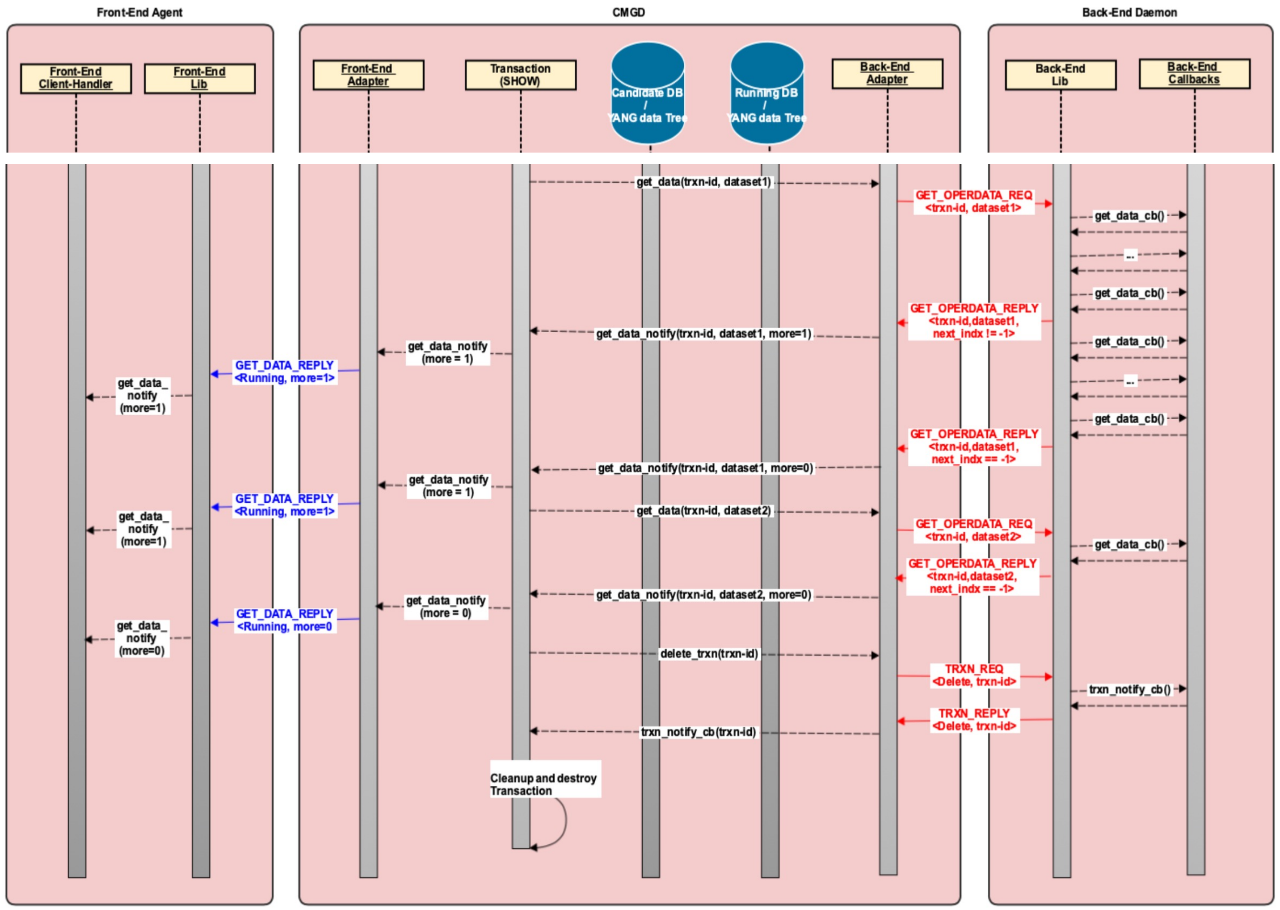
Retrieve Operational Data

Retrieve Lists, Containers and Leaf members

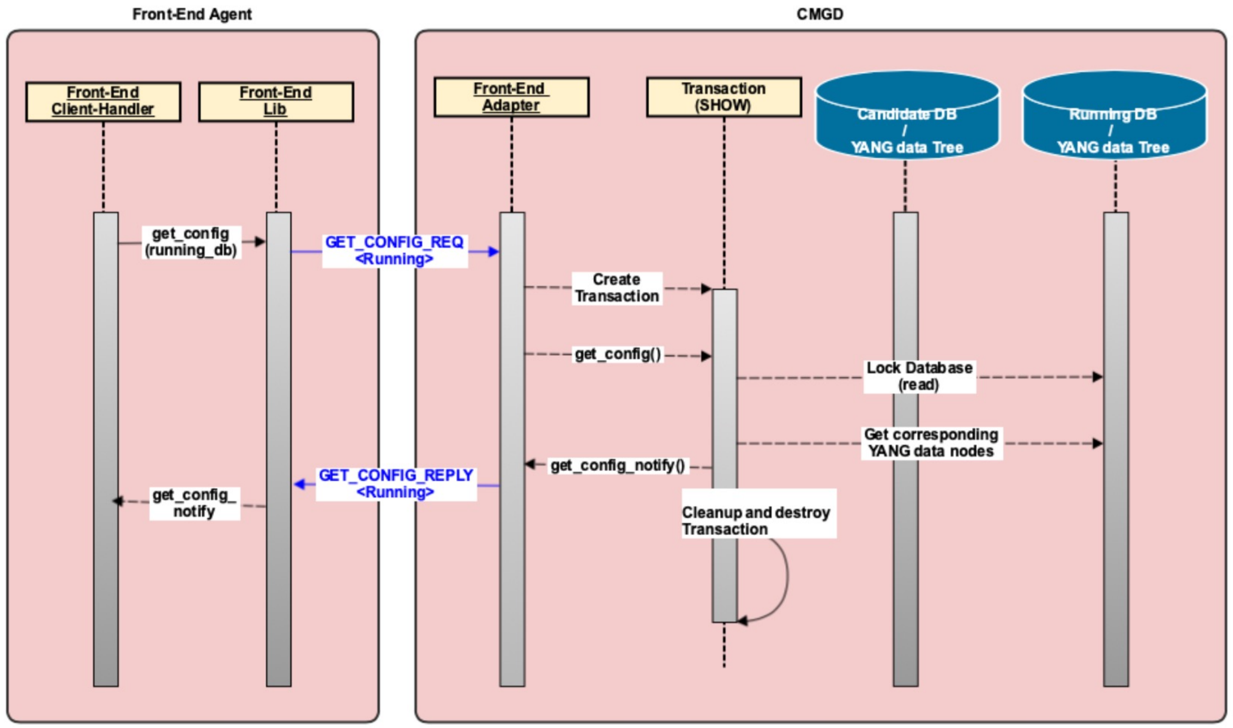


Retrieve Operational Data

Retrieve Lists, Containers and Leaf members

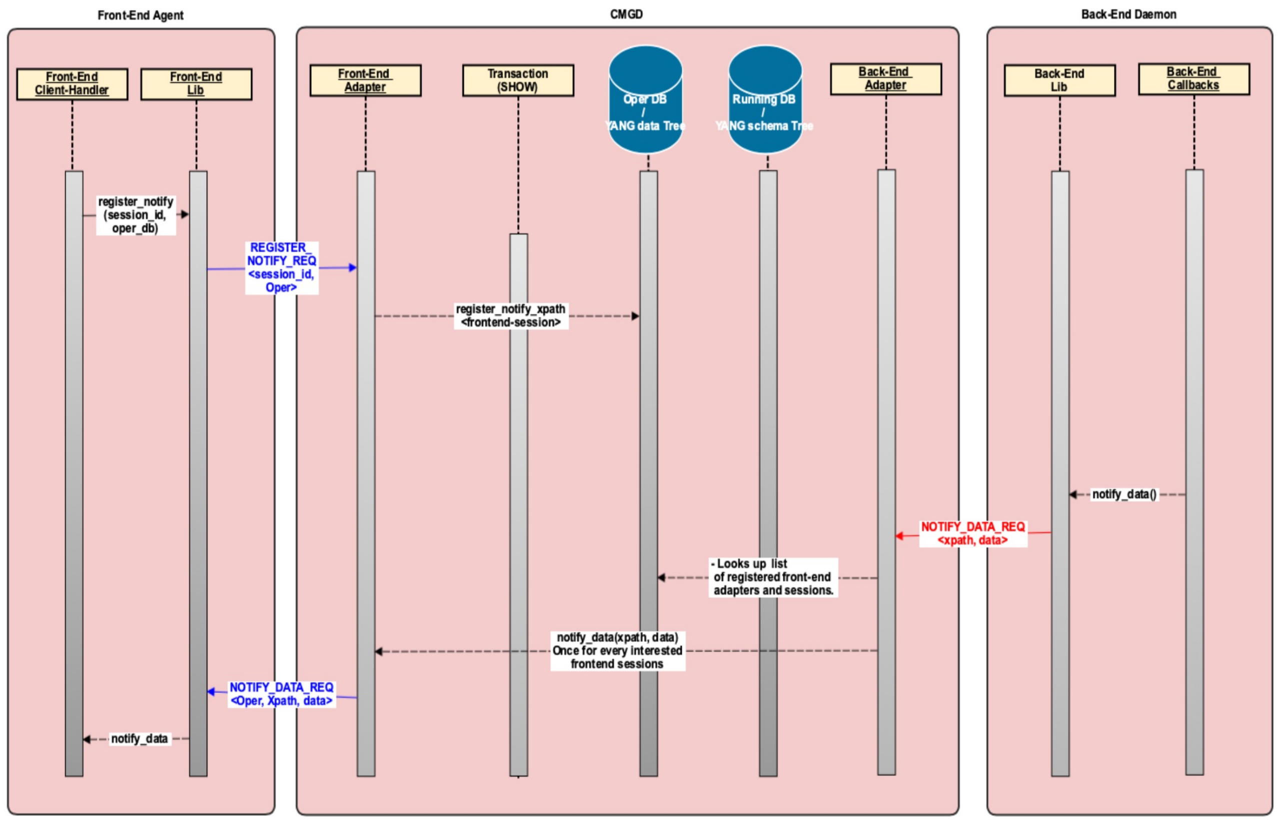

Front-End Agent
CMGD
Back-End Daemon
Front-End Client-Handler
Front-End Lib
Front-End Adapter
Transaction (SHOW)
Candidate DB / YANG data Tree
Running DB / YANG data Tree
Back-End Adapter
Back-End Lib
Back-End Callbacks
get_data(trxn-id, dataset1)
GET_OPERDATA_REQ <trxn-id, dataset1>
get_data_cb()
...
get_data_cb()
GET_OPERDATA_REPLY <trxn-id,dataset1, next_indx != -1>
get_data_notify(trxn-id, dataset1, more=1)
get_data_cb()
get_data_notify (more = 1)
GET_DATA_REPLY <Running, more=1>
get_data_notify (more=1)
...
get_data_cb()
GET_OPERDATA_REPLY <trxn-id,dataset1, next_indx == -1>
get_data_notify(trxn-id, dataset1, more=0)
get_data_notify (more = 1)
GET_DATA_REPLY <Running, more=1>
get_data_notify (more=1)
get_data(trxn-id, dataset2)
GET_OPERDATA_REQ <trxn-id, dataset2>
get_data_cb()
GET_OPERDATA_REPLY <trxn-id,dataset2, next_indx == -1>
get_data_notify(trxn-id, dataset2, more=0)
get_data_notify (more = 0)
GET_DATA_REPLY <Running, more=0>
get_data_notify (more=0)
delete_trxn(trxn-id)
TRXN_REQ <Delete, trxn-id>
trxn_notify_cb()
TRXN_REPLY <Delete, trxn-id>
trxn_notify_cb(trxn-id)
Cleanup and destroy Transaction

Retrieving ‘show running-config’



- Running DB maintained on MGMTD.
- Configurations already verified and applied earlier on back-end daemons.
- No need to involve back-end process for ‘show running config’.

Notifying data from Backend


Front-End Agent
CMGD
Back-End Daemon
Front-End Client-Handler
Front-End Lib
Front-End Adapter
Transaction (SHOW)
Oper DB / YANG data Tree
Running DB / YANG schema Tree
Back-End Adapter
Back-End Lib
Back-End Callbacks
register_notify (session_id, oper_db)
REGISTER_ NOTIFY_REQ <session_id, Oper>
register_notify_xpath <frontend-session>
notify_data()
NOTIFY_DATA_REQ <xpath, data>
- Looks up list of registered front-end adapters and sessions.
notify_data(xpath, data) Once for every interested frontend sessions
NOTIFY_DATA_REQ <Oper, Xpath, data>
notify_data

- Front-end client sessions express interest in specific Yang data-items.
  - Frontend client library sends NOTIFY_DATA_REQ with corresponding details
- Front-end adapter on MGMTD registers the corresponding session against the Yang data node for future reference.
- Later data notification arrives on backend adapter
- Looks up all registered sessions.
  - Sends a DATA_NOTIFY_REQ for each of the registered sessions.

Open Items and Issues

Items

- **Support for namespace**
- **Registration of callbacks at back-end. Possibly from multiple back-end daemons for the same configuration item with priority-based ordering.**
- **Delivery of callbacks registered from multiple backends for the same configuration item**
- Avoid YANG parsing on back-end process.
- Provide all the keys of YANG XPath to back-end callback handlers.

Issues

- Re-use of existing NB callback handlers.
- Avoid duplicating YANG data tree across MGMTD and back-end daemons.

Next Items

Low-level Designs

- **Support for namespace**
- **Registration of callbacks at back-end. Possibly from multiple back-end daemons for the same configuration item with priority-based ordering.**
- **Delivery of callbacks registered from multiple backends for the same configuration item**
- Avoid YANG parsing on back-end process.
- Provide all the keys of YANG XPath to back-end callback handlers.

Issues

- Re-use of existing NB callback handlers.
- Avoid duplicating YANG data tree across MGMTD and back-end daemons.

References

Community Documents

- **Northbound Architecture Wiki: https://github.com/opensourcerouting/frr/wiki/Architecture**
- **Requirements for Centralised Management: https://github.com/FRRouting/frr/wiki/FRR-Centralized-Management-Requirements**

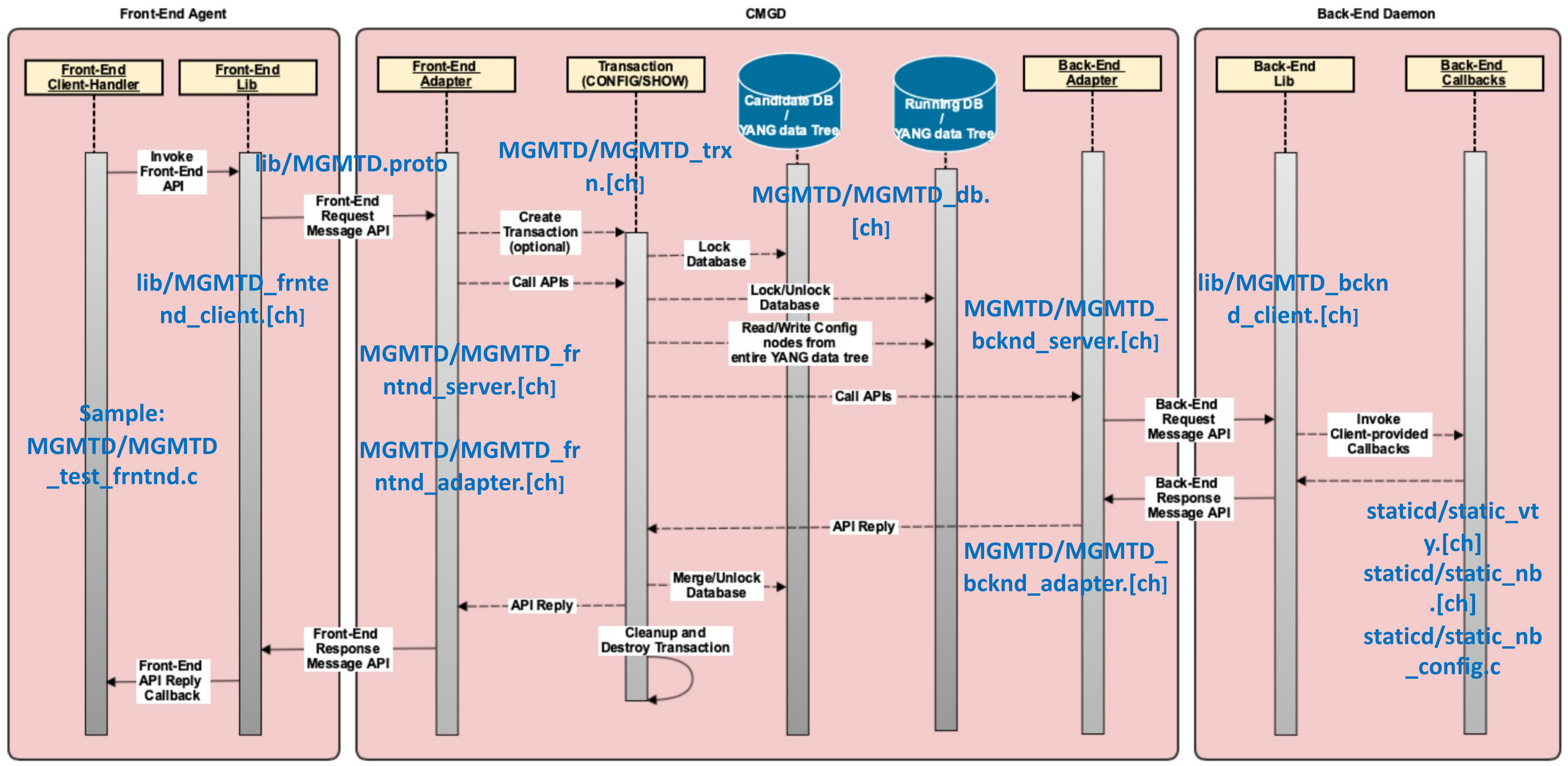
Standards

- RFC 6421 – NETCONF Protocol : https://datatracker.ietf.org/doc/html/rfc6241
- RFC 6536 – NETCONF Access Control Model : https://datatracker.ietf.org/doc/html/rfc6536
- RFC 7954 – YANG Specifications : https://datatracker.ietf.org/doc/html/rfc7954
- RFC 8342 – NETCONF Datastore Architecture : https://datatracker.ietf.org/doc/html/rfc8342
- RFC 8526 – NETCONF extensions for Datastores : https://datatracker.ietf.org/doc/html/rfc8526
- RFC 8040 – RESTCONF Protocol : https://datatracker.ietf.org/doc/html/rfc8040

THANK YOU!!!

Backup Slides

Code Organization



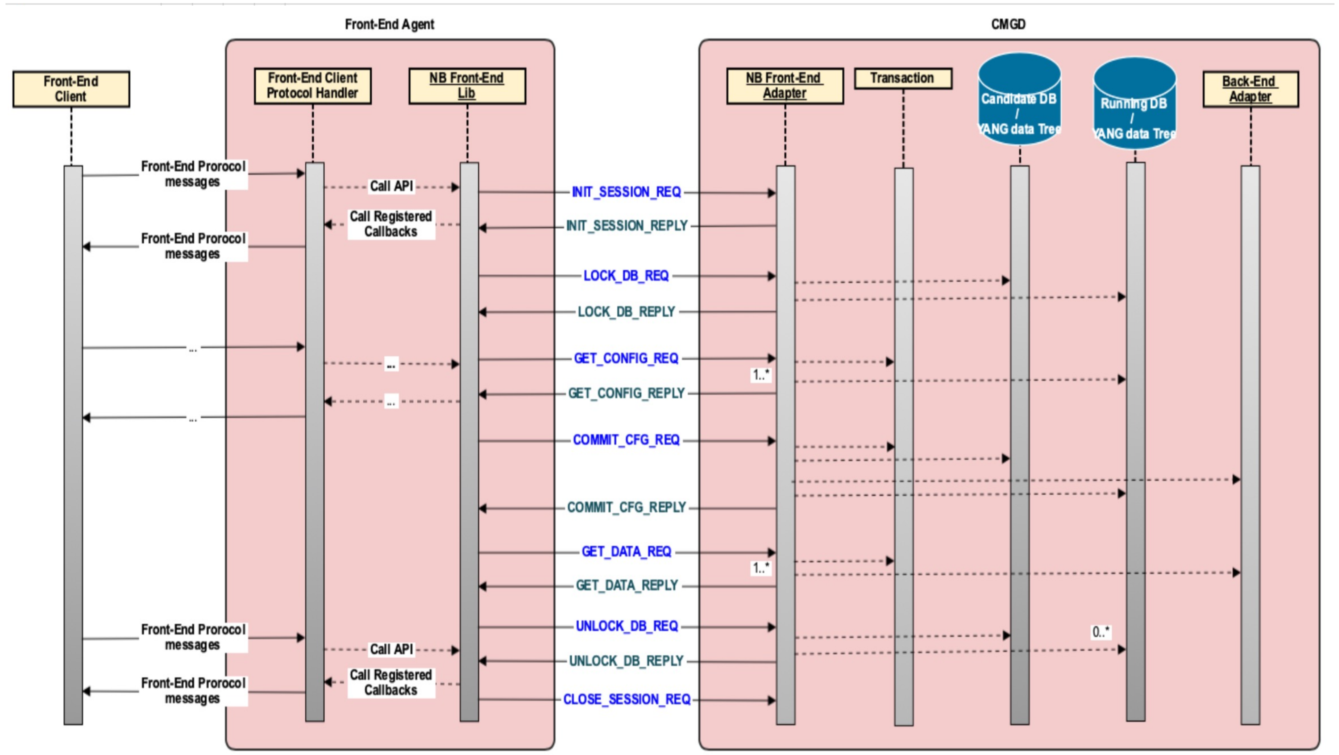
Front-End Agent
CMGD
Back-End Daemon
Front-End Client-Handler
Front-End Lib
Front-End Adapter
Transaction (CONFIG/SHOW)
Candidate DB / YANG data Tree
Running DB / YANG data Tree
Back-End Adapter
Back-End Lib
Back-End Callbacks
Invoke Front-End API
Front-End Request Message API
Create Transaction (optional)
Lock Database
Call APIs
Lock/Unlock Database
Read/Write Config nodes from entire YANG data tree
Call APIs
Back-End Request Message API
Invoke Client-provided Callbacks
Back-End Response Message API
API Reply
Merge/Unlock Database
API Reply
Front-End Response Message API
Cleanup and Destroy Transaction
Front-End API Reply Callback
lib/MGMTD.proto
MGMTD/MGMTD_trxn.[ch]
MGMTD/MGMTD_db.[ch]
lib/MGMTD_frntend_client.[ch]
MGMTD/MGMTD_bcknd_server.[ch]
lib/MGMTD_bcknd_client.[ch]
MGMTD/MGMTD_frntnd_server.[ch]
Sample: MGMTD/MGMTD_test_frntnd.c
MGMTD/MGMTD_frntnd_adapter.[ch]
MGMTD/MGMTD_bcknd_adapter.[ch]
staticd/static_vty.[ch]
staticd/static_nb.[ch]
staticd/static_nb_config.c

Front-End Interface

GRPC Client

- <TBA>

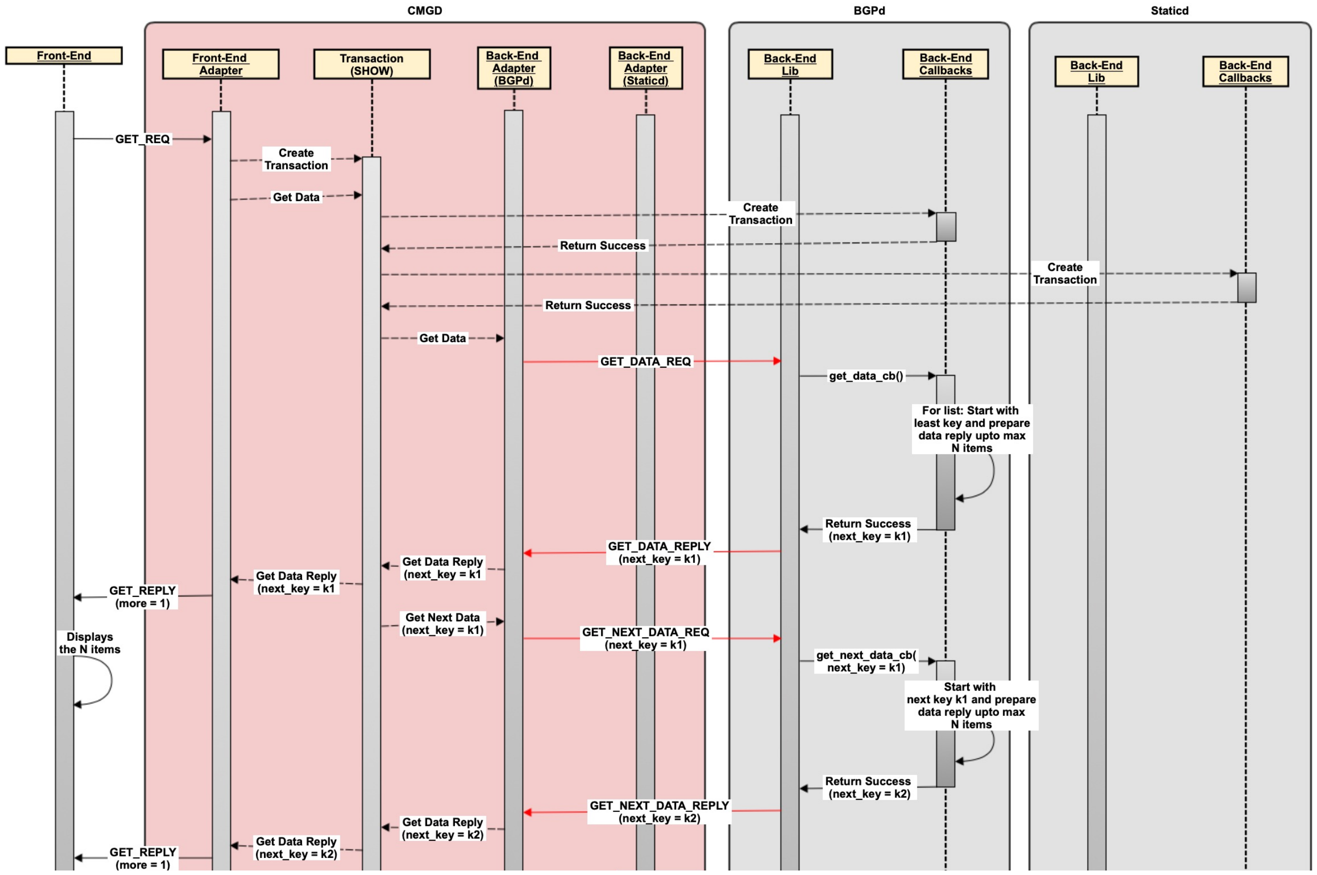
Front-End Interface



- INIT_SESSION_REQ
- INIT_SESSION_REPLY
- LOCK_DB_REQ
- LOCK_DB_REPLY
- UNLOCK_DB_REQ
- UNLOCK_DB_REPLY
- GET_CONFIG_REQ
- GET_CONFIG_REPLY
- SET_CONFIG_REQ
- SET_CONFIG_REPLY
- COMMIT_CFG_REQ
- COMMIT_CFG_REPLY
- GET_DATA_REQ
- GET_DATA_REPLY
- CLOSE_SESSION_REQ

Retrieve Operational Data

Retrieving Large List elements

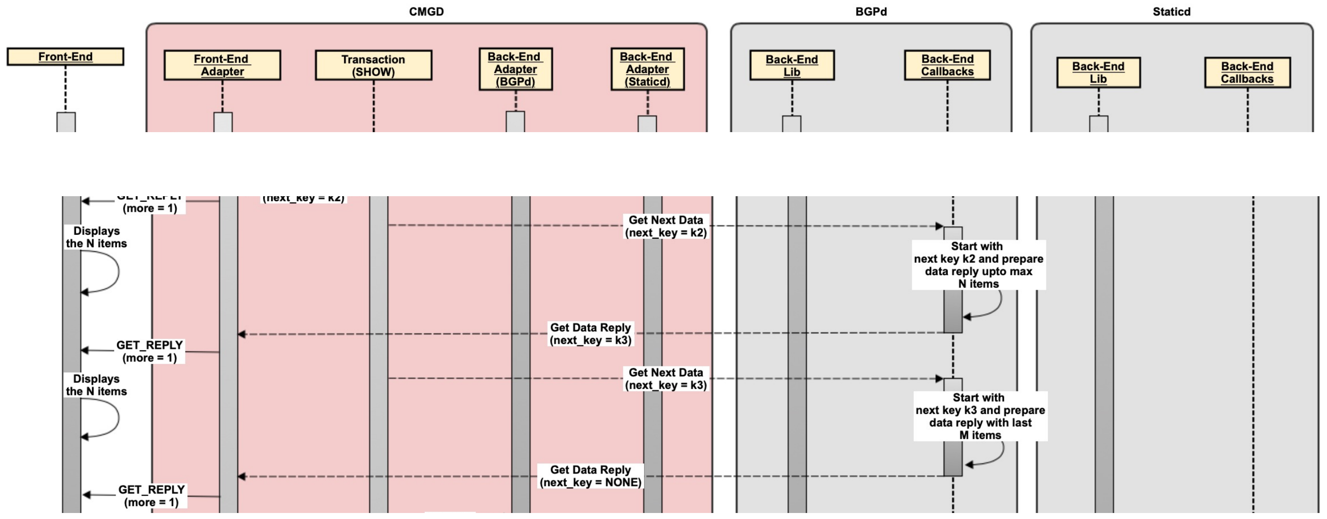


CMGD
BGPd
Staticd
Front-End
Front-End Adapter
Transaction (SHOW)
Back-End Adapter (BGPd)
Back-End Adapter (Staticd)
Back-End Lib
Back-End Callbacks
Back-End Lib
Back-End Callbacks
GET_REQ
Create Transaction
Get Data
Create Transaction
Return Success
Create Transaction
Return Success
Get Data
GET_DATA_REQ
get_data_cb()
For list: Start with least key and prepare data reply upto max N items
Return Success (next_key = k1)
GET_DATA_REPLY (next_key = k1)
Get Data Reply (next_key = k1
Get Data Reply (next_key = k1
GET_REPLY (more = 1)
Displays the N items
Get Next Data (next_key = k1)
GET_NEXT_DATA_REQ (next_key = k1)
get_next_data_cb(next_key = k1)
Start with next key k1 and prepare data reply upto max N items
Return Success (next_key = k2)
GET_NEXT_DATA_REPLY (next_key = k2)
Get Data Reply (next_key = k2)
Get Data Reply (next_key = k2)
GET_REPLY (more = 1)

- MGMTD receives GET_DATA request.
  - Examines the XPath and prepares.
    - List of YANG tree.
    - List of back-end adapters associated.
  - For large 'list' node
    - MGMTD sends a single Get-Data requests to back-end daemon.
    - Back-end prepares and send upto 'N' items in reply.
      - Indicates the key value for the next item.
    - MGMTD sends data received from back-end to Front-end via adapter.
      - Indicates more reply GET-REPLY to be expected.
    - MGMTD sends nexr follow upGET-Next-Data request withnext key value received in last reply.
    - Back-end prepares and send next 'N' items starting at next key value.
      - Indicates the key value for the next item ('None' if there's no more).

Retrieve Operational Data

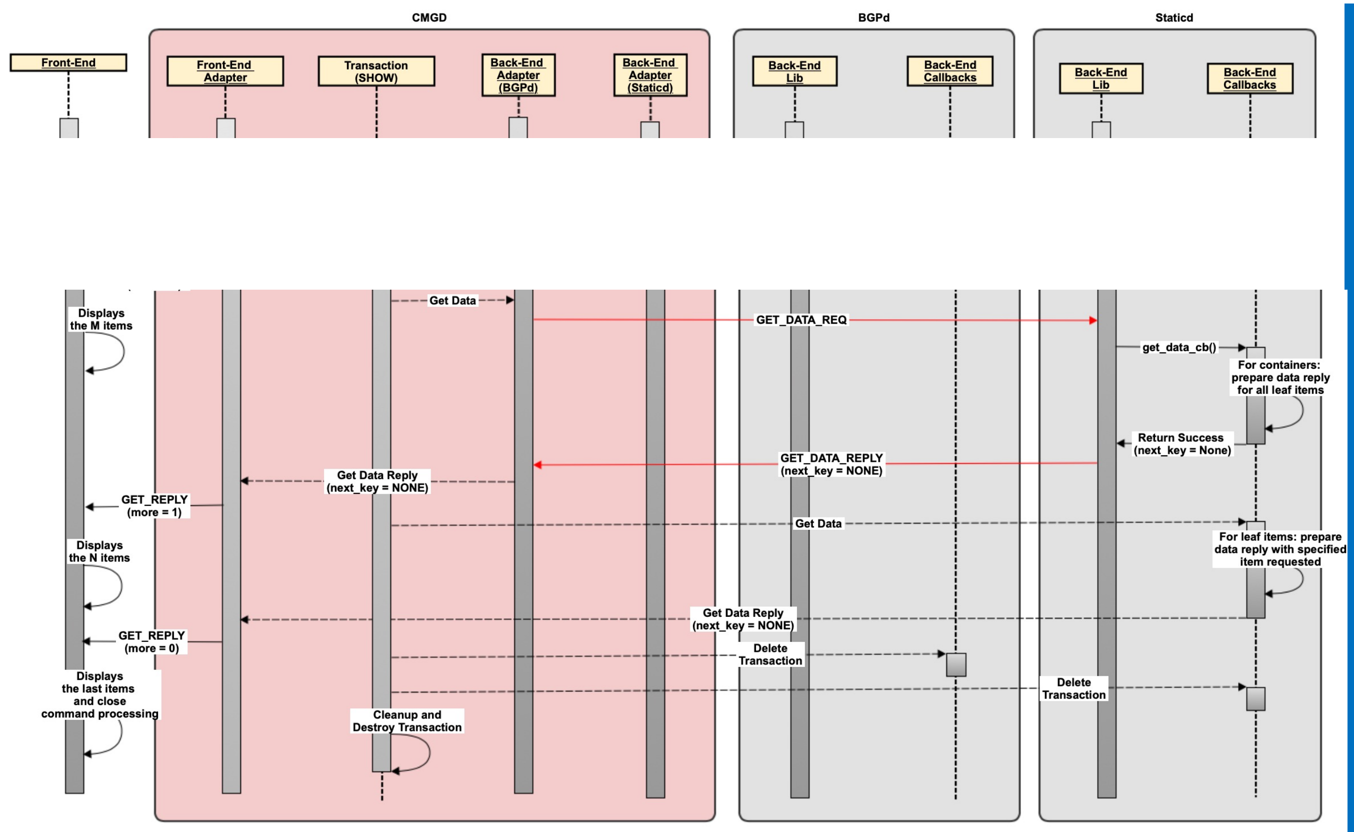
Retrieving Large List elements (contd.)


CMGD
BGPd
Staticd
Front-End
Front-End Adapter
Transaction (SHOW)
Back-End Adapter (BGPd)
Back-End Adapter (Staticd)
Back-End Lib
Back-End Callbacks
Back-End Lib
Back-End Callbacks
GET_REPLY (more = 1)
(next_key = k2)
Displays the N items
Get Next Data (next_key = k2)
Start with next key k2 and prepare data reply upto max N items
Get Data Reply (next_key = k3)
GET_REPLY (more = 1)
Displays the N items
Get Next Data (next_key = k3)
Start with next key k3 and prepare data reply with last M items
Get Data Reply (next_key = NONE)
GET_REPLY (more = 1)

- For large ‘list’ node (contd.)
  - …
  - MGMTD sends multiple follow up GET-Next-Data requests to back-end with appropriate next-key value.
  - Back-end prepares and send next ‘N’ items in reply.
    - Indicates the key value for the next item (‘None’ if there’s no more).

Retrieve Operational Data

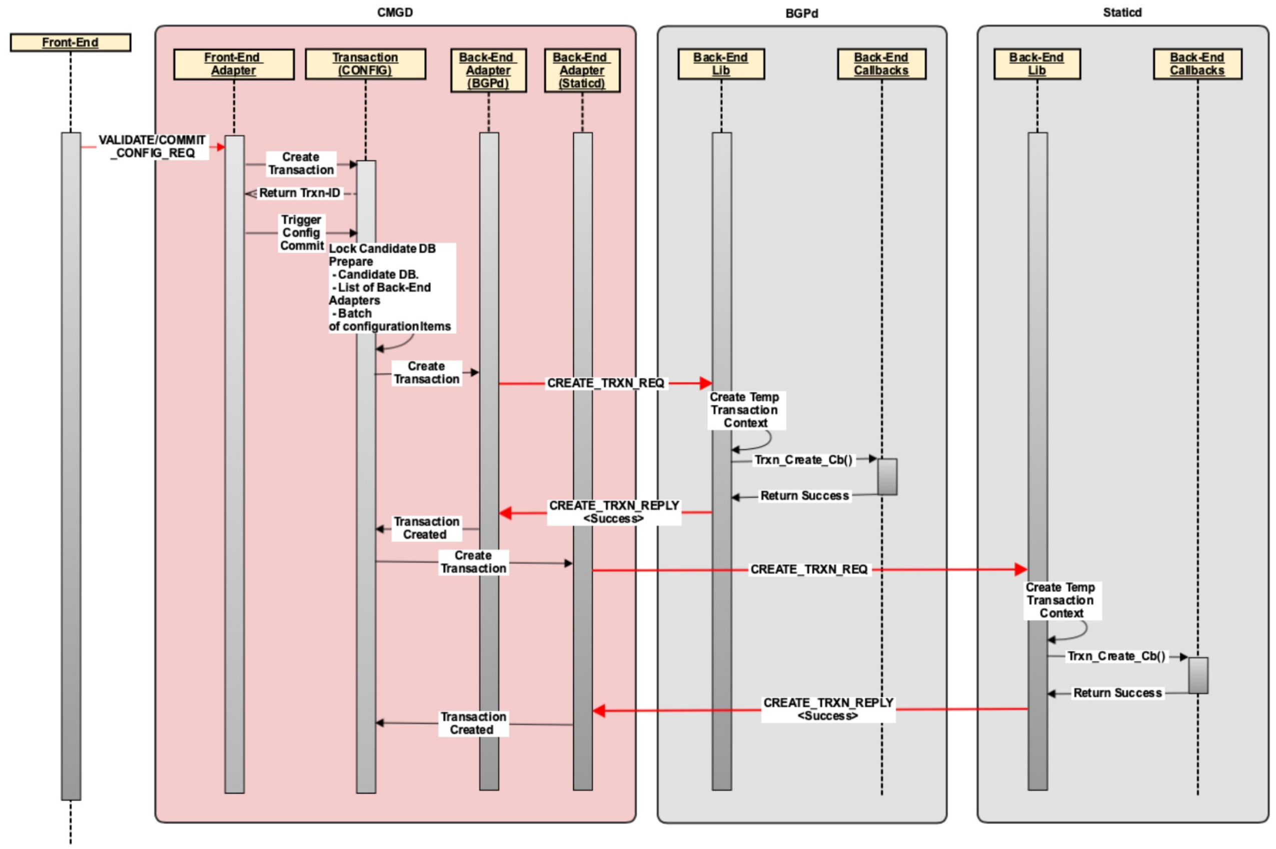
Retrieve Containers and Leaf members


CMGD
BGPd
Staticd
Front-End
Front-End Adapter
Transaction (SHOW)
Back-End Adapter (BGPd)
Back-End Adapter (Staticd)
Back-End Lib
Back-End Callbacks
Back-End Lib
Back-End Callbacks
Get Data
GET_DATA_REQ
Displays the M items
get_data_cb()
For containers: prepare data reply for all leaf items
Return Success (next_key = None)
GET_DATA_REPLY (next_key = NONE)
Get Data Reply (next_key = NONE)
GET_REPLY (more = 1)
Get Data
Displays the N items
For leaf items: prepare data reply with specified item requested
Get Data Reply (next_key = NONE)
GET_REPLY (more = 0)
Delete Transaction
Delete Transaction
Displays the last items and close command processing
Cleanup and Destroy Transaction

- For 'container' node
  - MGMTD sends a single Get-Data requests to back-end daemon.
    - May even combine more than one node in a single request.
  - Back-end prepares and sends all 'leaf' items under a single container in a single reply.
  - MGMTD sends data received from back-end to Front-end via adapter.
- For 'leaf' node
  - MGMTD sends a single Get-Data requests to back-end daemon.
  - Back-end prepares and sends the single 'leaf' in a single reply.
  - MGMTD sends data received from back-end to Front-end via adapter.
- If all data nodes has been sent, MGMTD sends the last GET-REPLY to Front-end.
  - with indication that no more GET-Replies are to be expected.

Transactions

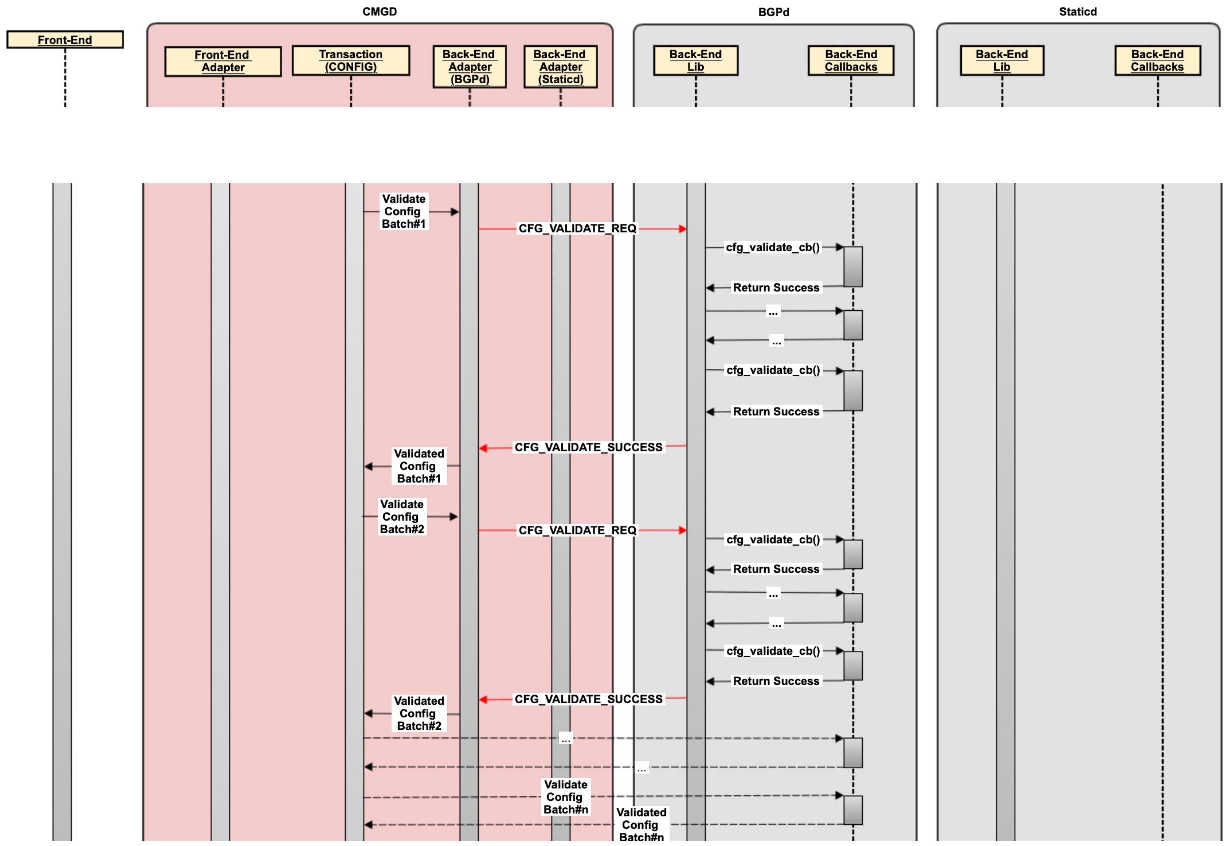
CONFIG Transactions -- Creating New Transaction



- <Notes TBA>

Transactions

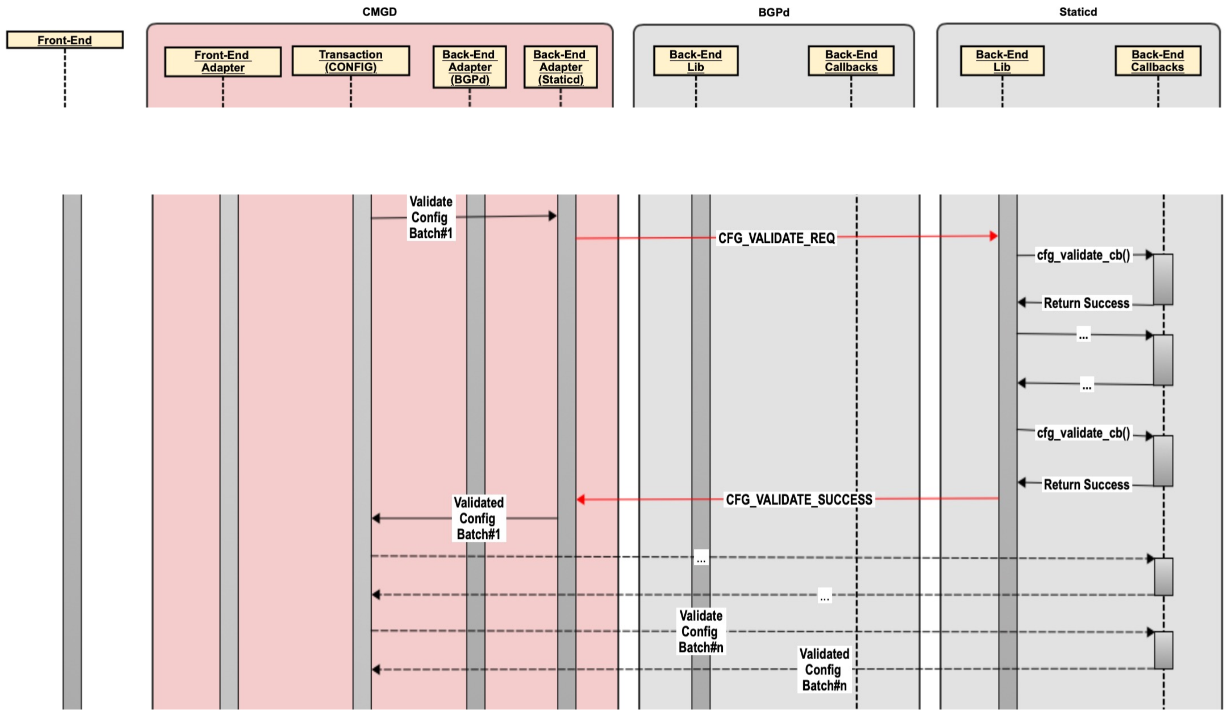
CONFIG Transactions -- Validating Valid Configurations



- <Notes TBA>

Transactions

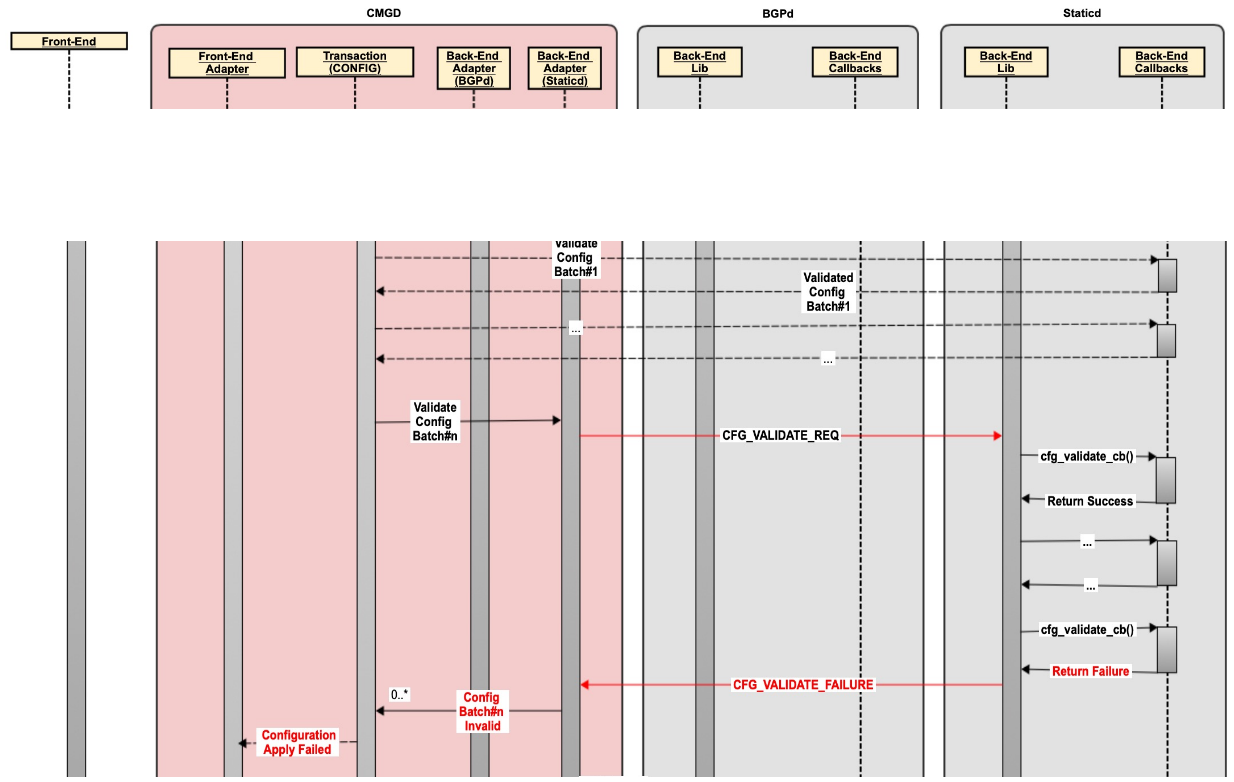
CONFIG Transactions -- Validating Valid Configurations (contd)



- <Notes TBA>

Transactions

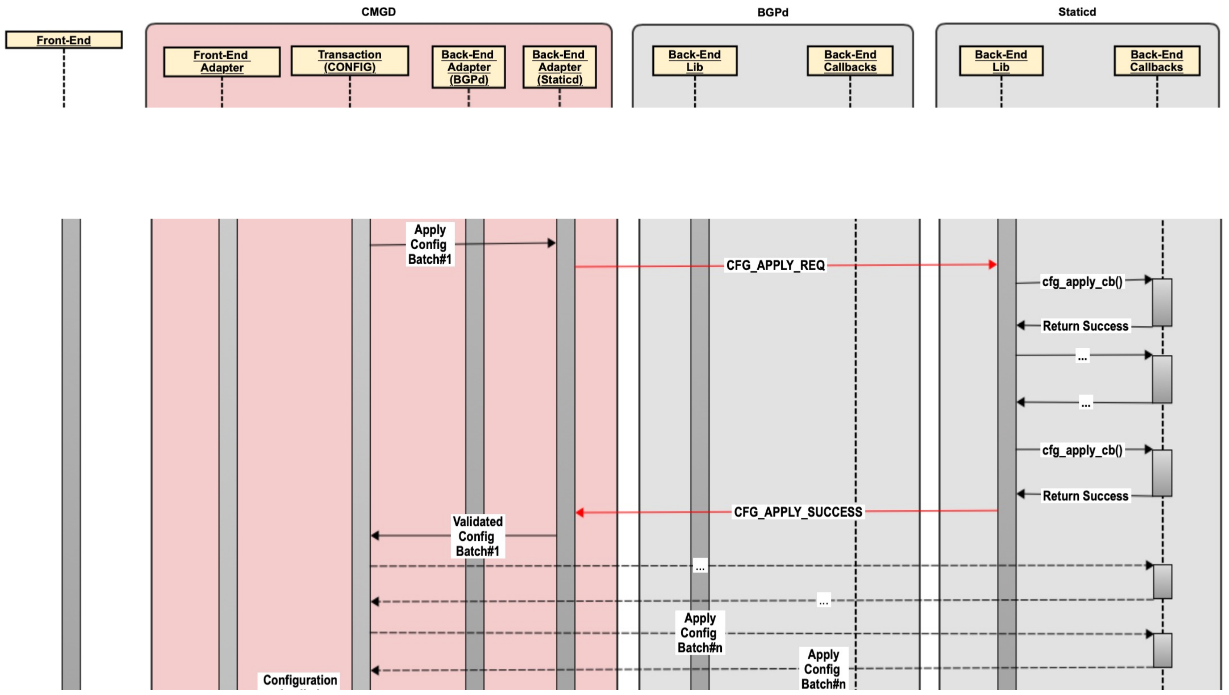
CONFIG Transactions -- Validating Invalid Configurations



- <Notes TBA>

Transactions

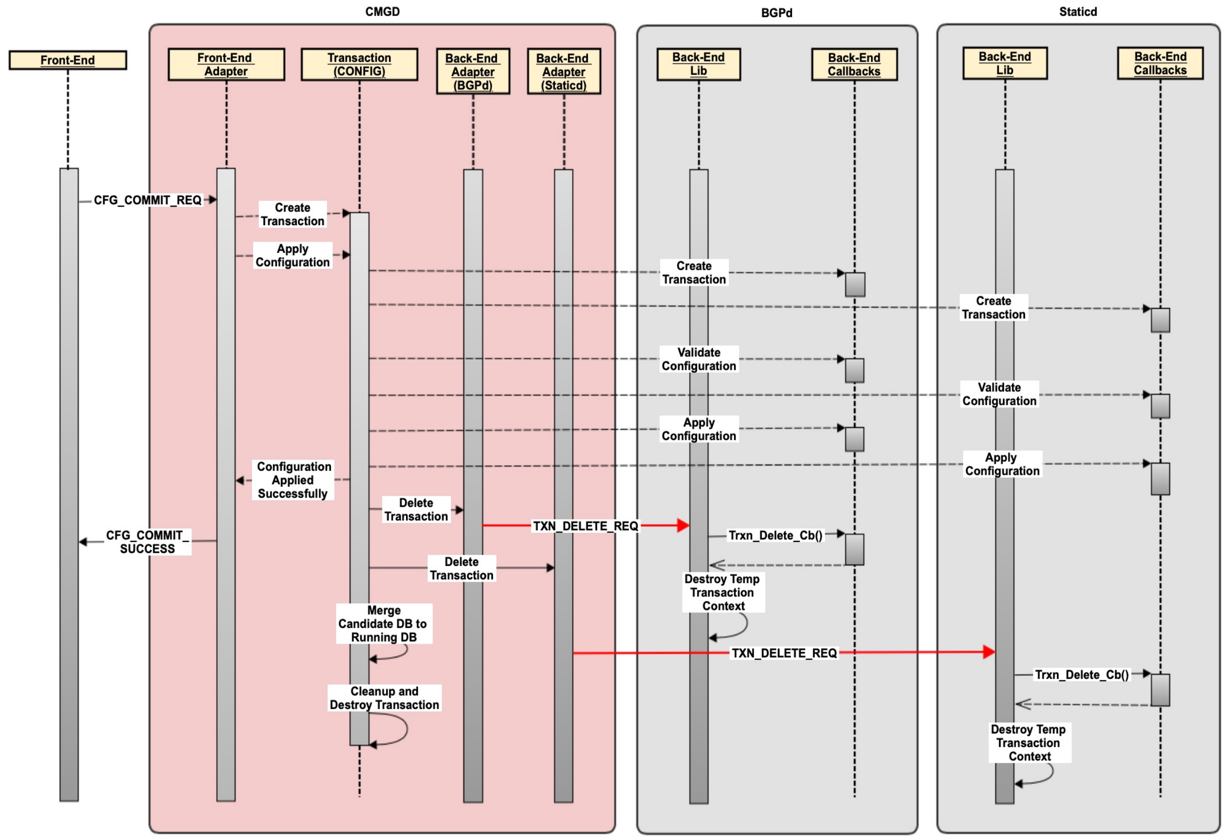
CONFIG Transactions -- Applying Validated Configurations (contd.)



- <Notes TBA>

Transactions

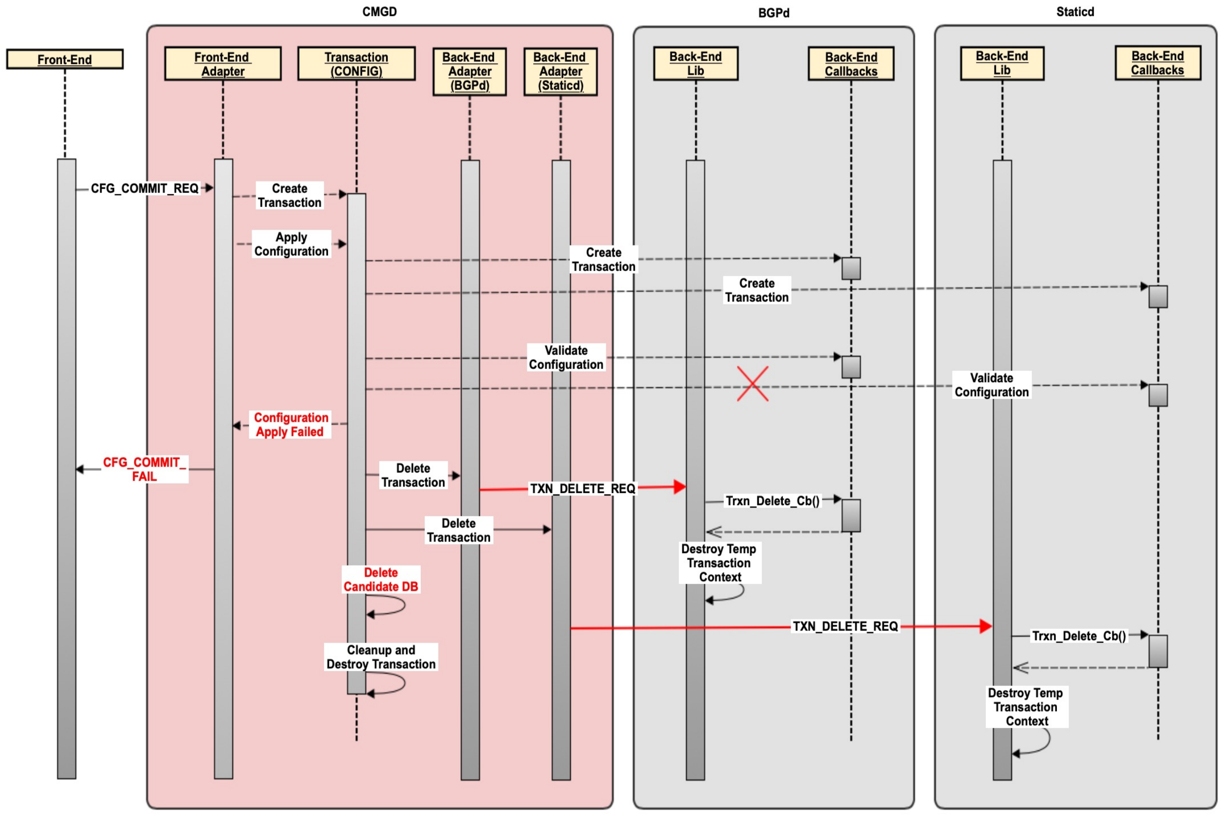
CONFIG Transactions -- Wrapping Up Successful Commit



- <Notes TBA>

Transactions

CONFIG Transactions -- Wrapping Up Rejected Commit



- <Notes TBA>

Processing VTYSH Commands

Processing VTYSH Commands

Sub-Phases and Requirements

- **PHASE-1: Converting Text Commands to Yang Xpaths** – VTYSH only
  - Currently implemented as DEFPY_YANG() routines and executed on backend daemon.
  - Calls nb_cli_enqueue_changes() followed by nb_cli_apply_changes()
- **PHASE-2: Validating the XPATH and corresponding value against Yang Schema**
  - Again executed on backend daemon.
  - Needs access to Yang Schema tree and Data tree (Running DB).
- **PHASE-3: Validating the corresponding value against current configuration**
  - Again executed through 'create/modify/destroy' callbacks on backend daemon.
  - Needs access to current configuration in Yang Data tree (Running DB).
- **PHASE-4: Applying the corresponding value on the backend internal state**
  - Again executed through 'create/modify/destroy' callbacks on backend daemon.
  - Needs to access backend internal state.
  - Must not need access to Yang Data tree.

Processing Configuration

Proposal 1

- **PHASE-1: Converting Text Commands to Yang Xpaths** – VTYSH only
  - Has to be somehow **moved to MGMTD daemon** for Phase-2.
- **PHASE-2: Validating the XPATH and corresponding value against Yang Schema**
  - Anyhow will have to be **moved to MGMTD daemon**.
  - Access to Yang Schema tree and Data tree (Running DB) can be provided on MGMTD itself.
- **PHASE-3: Validating the corresponding value against current configuration**
  - Triggered through Backend client library using **message exchange between MGMTD and backend**.
  - Executed through 'create/modify/destroy' **callbacks on backend daemon**.
  - Validation against current configuration
    - Maintain a **duplicate Yang Data sub-tree** (Running DB) on backend daemon.
    - Or, maintain a **shadow copy of configuration on internal storage**. Most daemons do that.
- **PHASE-4: Applying the corresponding value on the backend internal state**
  - Executed through 'create/modify/destroy' callbacks on backend daemon.
  - Needs to access backend internal state.
  - Must not need access to Yang Data tree.
  - Triggered through Backend client library using **message exchange between MGMTD and backend**.

Processing Configuration

Proposal 2

- **PHASE-1: Converting Text Commands to Yang Xpaths** – VTYSH only
  - Has to be somehow **moved to MGMTD daemon** for Phase-2.
- **PHASE-2: Validating the XPATH and corresponding value against Yang Schema**
  - Anyhow will have to be **moved to MGMTD daemon**.
  - Access to Yang Schema tree and Data tree (Running DB) can be provided on MGMTD itself.
- **PHASE-3: Validating the corresponding value against current configuration**
  - Executed through 'create/modify/destroy' **callbacks on (and hence to be loaded on) MGMTD daemon**.
  - **Validation against** current configuration in **Yang Data trree (Running DB) on MGMTD daemon only**.
  - **No message exchange required** between MGMTD and backend.
- **PHASE-4: Applying the corresponding value on the backend internal state**
  - Executed through 'create/modify/destroy' callbacks on backend daemon.
  - Needs to access backend internal state.
  - **Must not access Yang Data tree**.
  - Triggered through Backend client library using **message exchange between MGMTD and backend**.

Next Items

Low-level Designs

- **Support for namespace**
- **Registration of callbacks at back-end. Possibly from multiple back-end daemons for the same configuration item with priority-based ordering.**
- **Delivery of callbacks registered from multiple backends for the same configuration item**
- Avoid YANG parsing on back-end process.
- Provide all the keys of YANG XPath to back-end callback handlers.

Issues

- Re-use of existing NB callback handlers.
- Avoid duplicating YANG data tree across MGMTD and back-end daemons.

References

- **Northbound Architecture Wiki: https://github.com/opensourcerouting/frr/wiki/Architecture**
- **Requirements for Centralised Management: https://github.com/FRRouting/frr/wiki/FRR-Centralized-Management-Requirements**