Workgroup: Internet Engineering Task Force
Internet-Draft: draft-jgc-netconf-privcand-01
Published: 29 March 2023
Intended Status: Standards Track
Expires: 30 September 2023
Authors: JG. Cumming R. Wills
Nokia Cisco Systems

NETCONF Private Candidates

Abstract

This document provides a mechanism to extend the Network Configuration Protocol (NETCONF) to support multiple clients making configuration changes simultaneously and ensuring that they commit only those changes that they defined.

This document addresses two specific aspects: The interaction with a private candidate over the NETCONF protocol and the methods to identify and resolve conflicts between clients.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 30 September 2023.

Table of Contents

1. Introduction

NETCONF [RFC6241] provides a mechanism for one or more clients to make configuration changes to a device running as a NETCONF server. Each NETCONF client has the ability to make one or more configuration change to the servers shared candidate configuration.

As the name shared candidate suggests, all clients have access to the same candidate configuration. This means that multiple clients may make changes to the shared candidate prior to the configuration being committed. This behavior may be undesirable as one client may unwittingly commit the configuration changes made by another client.

NETCONF provides a way to mitigate this behavior by allowing clients to place a lock on the shared candidate. The placing of this lock means that no other client may make any changes until that lock is released. This behavior is, in many situations, also undesirable.

Many network devices already support private candidates configurations, where a user (machine or otherwise) is able to edit a personal copy of a devices configuration without blocking other users from doing so.

This document details the extensions to the NETCONF protocol in order to support the use of private candidates.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Definitions and terminology

2.1. Session specific datastore

A session specific datastore is a configuration datastore that, unlike the candidate and running configuration datastores which have only one per system, is bound to the specific NETCONF session.

2.2. Shared candidate configuration

The candidate configuration datastore defined in [RFC6241] is referenced as the shared candidate configuration in this document.

2.3. Private candidate configuration

A private candidate configuration is a session specific candidate configuration datastore.

The specific NETCONF session (and user) that created the private candidate configuration is the only session (user) that has access to it over NETCONF. Devices may expose this to other users through other interfaces but this is out of scope for this document.

The private candidate configuration contains a full copy of the running configuration when it is created (in the same way as a branch does in a source control management system and in the same way as the candidate configuration datastore as defined in [RFC6241]). Any changes made to it, for example, through the use of operations such as <edit-config> and <edit-data>, are made in this private candidate configuration.

Obtaining this private candidate over NETCONF will display the entire configuration, including all changes made to it. Performing a <commit> operation will merge the changes from the private candidate into the running configuration (the same as a merge in source code management systems). A <discard-changes> operation will revert the private candidate to the branch's initial state.

All changes made to this private candidate configuration are held separately from any other candidate configuration changes, whether made by other users to the shared candidate or any other private candidate, and are not visible to or accessible by anyone else.

3. Limitations using the shared candidate configuration for multiple clients

The following sections describe some limitations and mitigation factors in more detail for the use of the shared candidate configuration during multi-client configuration over NETCONF.

3.1. Issues

3.1.1. Unintended deployment of alternate users configuration changes

Consider the following scenario:

1. Client 1 modifies item A in the shared candidate configuration
2. Client 2 then modifies item B in the shared candidate configuration
3. Client 2 then issues a <commit> RPC

In this situation, both client 1 and client 2 configurations will be committed by client 2. In a machine-to-machine environment client 2 may not have been aware of the change to item A and, if they had been aware, may have decided not to proceed.

3.2. Current mitigation strategies

3.2.1. Locking the shared candidate configuration datastore

In order to resolve unintended deployment of alternate users configuration changes as described above NETCONF provides the ability to lock a datastore in order to restrict other users from editing and committed changes.

This does resolve the specific issue above, however, it introduces another issue. Whilst one of the clients holds a lock, no other client may edit the configuration. This will result in the client failing and having to retry. Whilst this may be a desirable consequence when two clients are editing the same section of the configuration, where they are editing different sections this behavior may hold up valid operational activity.

Additionally, a lock placed on the shared candidate configuration must also lock the running configuration, otherwise changes committed directly into the running datastore may conflict.

3.2.2. Always use the running configuration datastore

The use of the running configuration datastore as the target for all configuration changes does not resolve any issues regarding blocking of system access in the case a lock is taken, nor does it provide a solution for multiple NETCONF clients as each configuration change is applied immediately and the client has no knowledge of the current configuration at the point in time that they commenced the editing activity nor at the point they commit the activity.

3.2.3. Fine-grained locking

[RFC5717] describes a partial lock mechanism that can be used on specific portions of the shared candidate datastore.

Partial locking does not solve the issues of staging a set of configuration changes such that only those changes get committed in a commit operation, nor does it solve the issue of multiple clients editing the same parts of the configuration at the same time.

Partial locking additionally requires that the client is aware of any interdependencies within the servers YANG models in order to lock all parts of the tree.

4. Private candidates solution

The use of NETCONF private candidates resolves the issues detailed earlier in this document.

NETCONF sessions are able to utilize the concept of private candidates in order to streamline network operations, particularly for machine-to-machine communication.

Using this approach clients may improve their performance and reduce the likelihood of blocking other clients from continuing with valid operational activities.

One or more private candidates may exist at any one time, however, a private candidate SHOULD:

*Be accessible by one client only

*Be visible by one client only

Additionally, the choice of using a shared candidate configuration datastore or a private candidate configuration datastore MUST be for the entire duration of the NETCONF session.

4.1. What is a private candidate

A private candidate is defined earlier in the definitions and terminology section of this document.

4.2. When is a private candidate created

A private candidate datastore is created when the first RPC that requires access to it is sent to the server. This could be, for example, an <edit-config>.

When the private candidate is created a copy of the running configuration is made and stored in it. This can be considered the same as creating a branch in a source code repository.

Closing a session that is operating using a private candidate will discard all changes in that session's private candidate and destroy the private candidate.

4.3. How to signal the use of private candidates

In order to utilise a private candidate configuration, the client must inform the server that it wishes to do this.

4.3.1. Server

The server MUST signal its support for private candidates. The server does this by advertising the :candidate capability as defined in [RFC6241] and a new :private-candidate capability:

urn:ietf:params:netconf:capability:private-candidate:1.0

If the server has not signalled these capabilities, or has signalled the support for :candidate but not :private-candidate, or :private-candidate but not :candidate the session MUST be terminated when a client attempts to interact with a private candidate (for example, by explicitly targeting the private-candidate NMDA datastore).

4.3.2. Client

Two approaches are available for the client to signal that it wants to use a private candidate:

4.3.2.1. Client capability declaration

When a NETCONF client connects with a server it sends a list of client capabilities including one of the :base NETCONF version capabilties.

In order to enable private candidate mode for the duration of the NETCONF client session the NETCONF client sends the following capability:

urn:ietf:params:netconf:capability:private-candidate:1.0

In order for the use of private candidates to be established both the NETCONF server and the NETCONF client MUST advertise this capability.

When a server receives the client capability its mode of operation will be set to private candidate mode for the duration of the NETCONF session.

All RPC requests that target the candidate configuration datastore will operate in exactly the same way as they would do when using the shared candidate configuration datastore, however, when the server receives a request to act upon the candidate configuration datastore it instead uses the session's private candidate configuration datastore.

Using this method, the use of private candidates can be made available to NMDA and non-NMDA capable servers.

No protocol extensions are required for the transitioning of candidates between the shared mode and the private mode and no extensions are required for any RPCs (including <lock>)

4.3.2.2. Private candidate datastore

The private candidate configuration datastore is exposed as its own datastore similar to other NMDA [RFC8342] capable datastores. This datastore is called private-candidate.

All NMDA operations that support candidate NMDA datastore SHOULD support the private-candidate datastore.

Any non-NMDA aware NETCONF operations that take a source or target (destination) may be extended to accept the new datastore.

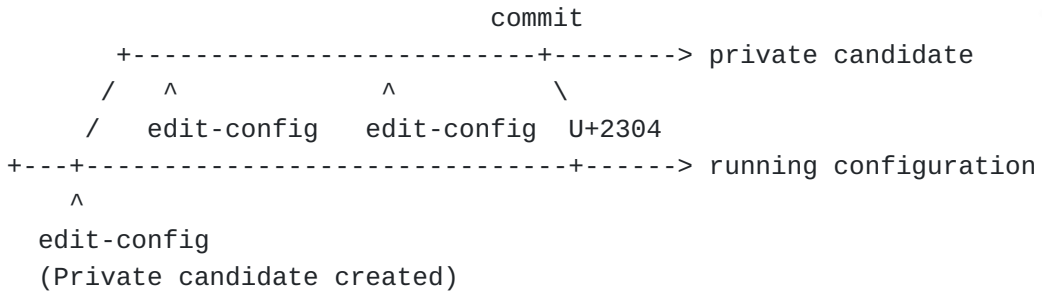
The ability for the NETCONF server to support private candidates is optional and SHOULD be signalled in NMDA supporting servers as a datastore in addition to the server capabilities described earlier in this document.

The first datastore referenced (either candidate or private-candidate) in any NETCONF operation will define which mode that NETCONF session will operate in for its duration. As an example, performing a <get-data> operation on the private-candidate datastore will switch the session into private candidate configuration mode and subsequent <edit-config> operations that reference the candidate configuration datastore will fail.

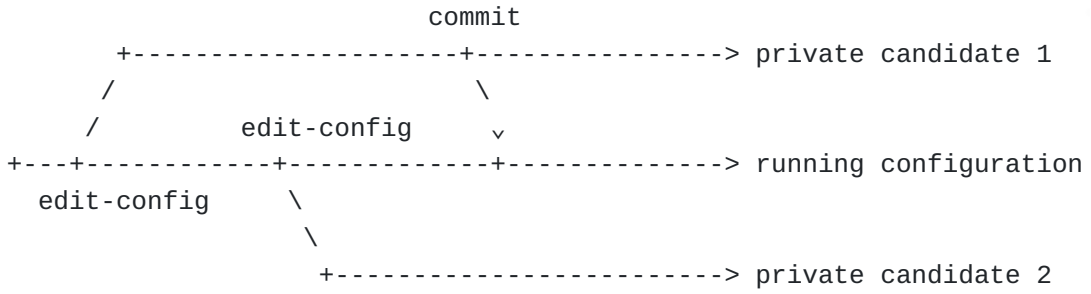
4.4. Interaction between running and private-candidate(s)

Multiple NETCONF operations may be performed on the private candidate in order to stage changes ready for a commit.

In the simplest example, a session may create a private candidate configuration, perform multiple NETCONF operations (such as <edit-config>) on it and then perform a <commit> operation to merge the private candidate configuration into the running configuration in line with semantics in [RFC6241].



More complex scenarios need to be considered, when multiple private candidate sessions are working on their own configuration (branches) and they make commits into the running configuration.



In this situation, if, how and when private candidate 2 is updated with the information that the running configuration has changed must be considered.

As described earlier, the client MUST be aware of changes to the running configuration so it can be assured that it is only committing its own modifications.

It is possible, during an update, for conflicts to occur and the detection and resolution of these is discussed later in this document.

Two modes of operation are provided. Both modes may be supported, however, the server SHOULD advertise which approach is being used in a capability.

4.4.1. Static branch mode: Independent private candidate branch

The private candidate is treated as a separate branch and changes made to the running configuration are not placed into the private candidate datastore except in one of the following situations:

* The client requests that the private candidate be refreshed using a new <update> operation

*<commit> is issued (which SHOULD automatically issue an <update> operation)

This approach is similar to the standard approach for source code management systems.

In this model of operation it is possible for the private candidate configuration to become significantly out of sync with the running configuration should the private candidate be open for a long time without an operation being sent that causes a resync (rebase in source code control terminology).

A <compare> operation may be performed against the initial starting point (head) of the private candidates branch or against the running configuration.

Conflict detection and resolution is discussed later in this document.

4.4.2. Continuous rebase mode: Continually updating private candidate

The private candidate is treated as a separate branch, however, when changes are made to the running configuration the update operation will automatically be run on all open private candidate branches.

This is equivalent to all currently open private candidate branches being rebased onto the running configuration every time a change is made to it by any session.

In this model of operation the following should be considered:

*Because the private candidate is automatically re-synchronized (rebased) with the running configuration each time a change is made in the running configuration, the NETCONF session is unaware that their private candidate configuration has changed unless they perform one of the get operations on the private candidate and analyse it for changes.

*A <compare> operation may be performed against the initial starting point (head) of the private candidates branch or against the running configuration but these will both report the same results as the starting point is continually reset.

*The output of the <compare> operation may not match the set of changes made to the session's private candidate but may include different output due to the changes in the running configuration made by other sessions.

*A conflict may occur in the automatic update process pushing changes from the running configuration into the private candidate.

Conflict detection and resolution is discussed later in this document.

4.5. Detecting and resolving conflicts

4.5.1. What is a conflict?

A conflict is when the intent of the NETCONF client may have been different had it had a different starting point. In configuration terms, a conflict occurs when the same set of nodes in a configuration being altered by one user are changed between the start of the configuration preparation (the first <edit-config>/<edit-data> operation) and the conclusion of this configuration session (terminated by a <commit> operation).

The situation where conflicts have the potential of occurring are when multiple configuration sessions are in progress and one session commits changes into the running configuration after the private candidate (branch) was created.

When this happens a conflict occurs if the nodes modified in the running configuration are the same nodes that are modified in the private candidate configuration.

Examples of conflicts include:

*An interface has been deleted in the running configuration that existed when the private candidate was created. A change to a child node of this specific interface is made in the private candidate using the default merge operation would, instead of changing the child node, both recreate the interface and then set the child node.

*A leaf has been modified in the running configuration from the value that it had when the private candidate was created. The private candidate configuration changes that leaf to another value.

4.5.2. Detecting and reporting conflicts

A conflict can occur when an <update> operation is triggered. This can occur in a number of ways:

*Manually triggered by the <update> NETCONF operation

* Automatically triggered by the NETCONF server running an <update> operation upon a <commit> being issued by the client in the private candidate session.

* Automatically triggered by the NETCONF server running an <update> operation upon a <commit> being issued by any other configuration session (or user). This occurs in continual rebase mode only.

When a conflict occurs the client MUST be given the opportunity to re-evaluate its intent based on the new information. The resolution of the conflict may be manual or automatic depending on the server and client decision (discussed later in this document).

When a conflict occurs, a <commit> or <update> operation MUST fail. It MUST inform the client of the conflict and SHOULD detail the location of the conflict(s).

In continuous rebase mode, it is possible for the automated <update> operation to fail. In this instance, the next NETCONF operation (of any type) MUST fail. It MUST inform the client of the conflict and SHOULD detail the location of the conflict(s).

The location of the conflict(s) should be reported as a list of xpaths and values.

4.5.3. Conflict resolution

When a conflict is detected, the client MUST be informed. The client then has a number of options available to resolve the conflict.

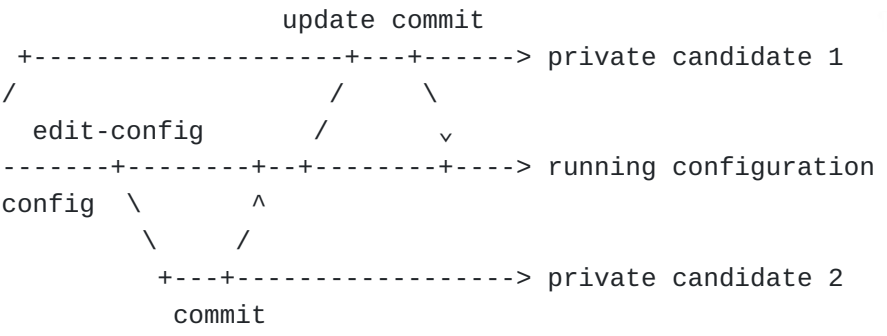
It is worth noting that in the case of continuous rebase mode automated <update> operations may be performed against multiple private candidate configurations at once.

The resolution method SHOULD be provided as an input to the <update> operation described later in this document. This input may be through a default selection, a specific input or a configuration element.

The following configuration data is used below to describe the behavior of each resolution method:

```
<configure>
  <interfaces>
    <interface>
      <name>intf_one</name>
      <description>Link to London<description>
    </interface>
    <interface>
      <name>intf_two</name>
      <description>Link to Tokyo<description>
    </interface>
  </interfaces>
</configure>
```

The following workflow diagram is used and the outcome is the same regardless of whether static branch mode or continuous rebase mode is being used. For the purpose of the examples below assume the update operation is manually provided by a client.



There are three defined resolution methods:

4.5.3.1. Ignore

When using the ignore resolution method items in the running configuration that are not in conflict with the private candidate configuration are merged from the running configuration into the private candidate configuration. Nodes that are in conflict are ignored and not merged. The outcome of this is that the private candidate configuration reflects changes in the running that were not being worked on and those that are being worked on in the private candidate remain in the private candidate. Issuing a <commit> operation at this point will overwrite the running configuration with the conflicted items from the private candidate configuration.

Example:

Session 1 edits the configuration by submitting the following

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name>intf_one</name>
            <description>Link to San Francisco<description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 2 then edits the configuration deleting the interface intf_one, updating the description on interface intf_two and committing the configuration to the running configuration datastore.

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name operation="delete">intf_one</name>
          </interface>
          <interface>
            <name>intf_two</name>
            <description>Link moved to Paris</description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 1 then sends an <update> NETCONF operation.

```
<rpc message-id="update"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <update>
    <resolution-mode>ignore</resolution-mode>
  </update>
</rpc>
```

The un-conflicting changes are merged and the conflicting ones are ignored (and not merged from the running into private candidate 1).

The resulting data in private candidate 1 is:

```
<configure>
  <interfaces>
    <interface>
      <name>intf_one</name>
      <description>Link to San Francisco<description>
    </interface>
    <interface>
      <name>intf_two</name>
      <description>Link moved to Paris<description>
    </interface>
  </interfaces>
</configure>
```

4.5.3.2. Overwrite

When using the overwrite resolution method items in the running configuration that are not in conflict with the private candidate configuration are merged from the running configuration into the private candidate configuration. Nodes that are in conflict are pushed from the running configuration into the private candidate configuration, overwriting any previous changes in the private candidate configuration. The outcome of this is that the private candidate configuration reflects the changes in the running configuration that were not being worked on as well as changing those being worked on in the private candidate to new values.

Example:

Session 1 edits the configuration by submitting the following

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name>intf_one</name>
            <description>Link to San Francisco<description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 2 then edits the configuration deleting the interface intf_one, updating the description on interface intf_two and committing the configuration to the running configuration datastore.

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name operation="delete">intf_one</name>
          </interface>
          <interface>
            <name>intf_two</name>
            <description>Link moved to Paris</description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 1 then sends an <update> NETCONF operation.

```
<rpc message-id="update"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <update>
    <resolution-mode>overwrite</resolution-mode>
  </update>
</rpc>
```

The un-conflicting changes are merged and the conflicting ones are pushed into the private candidate 1 overwriting the existing changes.

The resulting data in private candidate 1 is:

```
<configure>
  <interfaces>
    <interface>
      <name>intf_two</name>
      <description>Link moved to Paris<description>
    </interface>
  </interfaces>
</configure>
```

4.5.3.3. Revert-on-conflict

When using the revert-on-conflict resolution method items and update will fail to complete when any conflicting node is found. The session issuing the update will be informed of the failure.

No changes, whether conflicting or un-conflicting are merged into the private candidate configuration.

The owner of the private candidate session must then take deliberate and specific action to adjust the private candidate configuration to rectify the conflict. This may be by issuing further <edit-config> or <edit-data> operations, by issuing a <discard-changes> operation or by issuing an <update> operation with a different resolution method.

This resolution method is the default resolution method as it provides for the highest level of visibility and control to ensure operational stability.

This resolution method may not be selected by a system operating in continuous rebase mode when performing automatic <update> operations. Clients operating in continuous rebase mode may use this resolution mode in their <update> operation.

Example:

Session 1 edits the configuration by submitting the following

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name>intf_one</name>
            <description>Link to San Francisco<description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 2 then edits the configuration deleting the interface intf_one, updating the description on interface intf_two and committing the configuration to the running configuration datastore.

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name operation="delete">intf_one</name>
          </interface>
          <interface>
            <name>intf_two</name>
            <description>Link moved to Paris</description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 1 then sends an <update> NETCONF operation.

```
<rpc message-id="update"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <update>
    <resolution-mode>revert-on-conflict</resolution-mode>
  </update>
</rpc>
```

A conflict is detected and no merges/overwrite operations happen.

The resulting data in private candidate 1 is:

```
<configure>
  <interfaces>
    <interface>
      <name>intf_one</name>
      <description>Link to San Francisco<description>
    </interface>
    <interface>
      <name>intf_two</name>
      <description>Link to Tokyo<description>
    </interface>
  </interfaces>
</configure>
```

4.6. NETCONF operations

4.6.1. New NETCONF operations

4.6.1.1. <update>

The <update> operation is provided to allow NETCONF clients (or servers) to trigger a rebase of the private candidate configuration against the running configuration.

The <update> operation may be triggered manually by the client or automatically by the server.

The <update> operation MUST be triggered by a <commit> operation being executed in any candidate configuration on the device if the device is operating in continuous rebase mode.

The <update> operation SHOULD be triggered by a specific NETCONF session issuing a <commit> operation.

4.6.1.1.1. <resolution-mode> parameter

The <update> operation takes the <resolution-mode> parameter

The resolution modes are described earlier in this document and the accepted inputs are:

- *revert-on-conflict (default)
- *ignore
- *overwrite

4.6.2. Updated NETCONF operations

Specific NETCONF operations altered by this document are listed in this section. Any notable behavior with existing unaltered NETCONF operations is noted in the appendix.

4.6.2.1. <edit-config>

The <edit-config> operation is updated to accept private-candidate as valid input to the <target> field.

The use of <edit-config> will create a private candidate configuration if one does not already exist for that NETCONF session.

Sending an <edit-config> request to private-candidate after one has been sent to the shared candidate datastore in the same session will fail (and visa-versa).

Multiple <edit-config> requests may be sent to the private-candidate datastore in a single session.

4.6.2.2. <lock> and <unlock>

Performing a <lock> on the private-candidate datastore is a valid operation and will also lock the running configuration.

Taking a lock on this datastore will stop other session from committing any configuration changes, regardless of the datastore.

Other NETCONF sessions are still able to create a new private-candidate configurations.

Performing an <unlock> on the private-candidate datastore is a valid operation. This will also unlock the running configuration. Unlocking the private-candidate datastore allows other sessions to resume <commit> functions.

Changes in the private-candidate datastore are not lost when the lock is released.

Attempting to perform a <lock> or <unlock> on any other datastore while the private-candidate datastore is locked will fail. Attempting to perform a <lock> or <unlock> on any other sessions private-candidate datastore will also fail.

4.6.2.3. <compare>

Performing a <compare> [RFC9144] with the private-candidate datastore as either the <source> or <target> is a valid operation.

If <compare> is performed prior to a private candidate configuration being created, one will be created at that point.

The <compare> operation will be extended to allow the operation to reference the start of the private candidate's branch (head).

4.6.2.4. <get-config>

The <get-config> operation is updated to accept private-candidate as valid input to the <source> field.

The use of <get-config> will create a private candidate configuration if one does not already exist for that NETCONF session.

Sending an <get-config> request to private-candidate after one has been sent to the shared candidate datastore in the same session will fail (and visa-versa).

4.6.2.5. <get-data>

The <get-data> operation accepts the private-candidate as a valid datastore.

The use of <get-data> will create a private candidate configuration if one does not already exist for that NETCONF session.

Sending an <get-data> request to private-candidate after one has been sent to the shared candidate datastore in the same session will fail (and visa-versa).

4.6.2.6. <copy-config>

The <copy-config> operation is updated to accept private-candidate as a valid input to the <source> or <target> fields.

4.6.2.7. <delete-config>

The <delete-config> operation is updated to accept private-candidate as a valid input to the <target> field.

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

This document should not affect the security of the Internet.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC9144] Clemm, A., Qu, Y., Tantsura, J., and A. Bierman, "Comparison of Network Management Datastore Architecture (NMDA) Datastores", RFC 9144, DOI 10.17487/RFC9144, December 2021, <https://www.rfc-editor.org/info/rfc9144>.

[RFC5717] Lengyel, B. and M. Bjorklund, "Partial Lock Remote Procedure Call (RPC) for NETCONF", RFC 5717, DOI 10.17487/RFC5717, December 2009, <https://www.rfc-editor.org/info/rfc5717>.

7.2. Informative References

Appendix A. Behavior with unaltered NETCONF operations

A.1. <get>

The <get> operation does not accept a datastore value and therefore this document is not applicable to this operation. The use of the get operation will not create a private candidate configuration.

Contributors

The authors would like to thank Jan Lindblad, Jason Sterne and Rob Wilton for their contributions and reviews.

Authors' Addresses

James Cumming
Nokia

Email: james.cumming@nokia.com

Robert Wills
Cisco Systems

Email: rowills@cisco.com