Internet Engineering Task Force
Internet-Draft
Updates: 6241, 8342, 8526, 9144 (if approved)
Intended status: Standards Track
Expires: 6 May 2026

JG. Cumming
Nokia
R. Wills
Cisco Systems

2 November 2025

NETCONF and RESTCONF Private Candidate Datastores
draft-ietf-netconf-privcand-08

Abstract

This document provides a mechanism to extend the Network Configuration Protocol (NETCONF) and RESTCONF protocol 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 and RESTCONF protocols 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 6 May 2026.

Copyright Notice

Table of Contents

1. Introduction

NETCONF [RFC6241] and RESTCONF [RFC8040] both provide a mechanism for one or more clients to make configuration changes to a device running as a NETCONF/RESTCONF server. Each client has the ability to make one or more configuration changes to the server's 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 behaviour may be undesirable as one client may unwittingly commit the configuration changes made by another client.

NETCONF provides a way to mitigate this behaviour 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 behaviour is, in many situations, also undesirable.

Many network devices support candidate configurations within the CLI interface, where a user (machine or otherwise) is able to edit a self-contained copy of a device's configuration without blocking other users from doing so.

This document specifies the extensions to the NETCONF protocol in order to support the use of private candidates. It also describes how the RESTCONF protocol can be used on a system that implements 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.

1.2. Updates to RFC 6241 and RFC 8526

This document updates [RFC6241] to describe how the <edit-config>, <copy-config>, <get>, <get-config>, <commit>, <cancel-commit>, <delete-config>, <discard-changes>, <lock> and <unlock> operations work with a private candidate datastore. These updates are described in Section 3.8.2. This document also adds a new <update> operation, described in Section 3.8.1.

This document also updates [RFC8526] to describe how the <edit-data> and <get-data> operations work with a private candidate datastore. These updates are described in Section 3.8.2.

1.3. Updates to RFC 9144

This document updates [RFC9144] to augment the <compare> operation to describe how it works with the private candidate datastore. These updates are described in Section 3.8.2.7.

1.4. 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 or RESTCONF.

1.4.1. Issues

1.4.1.1. 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.

1.4.1.2. Mitigation strategies when using a shared candidate datastore

1.4.1.2.1. 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 behaviour may hold up valid operational activity.

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

Finally, this locking mechanism isn’t available to RESTCONF clients.

1.4.1.2.2. 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 and RESTCONF 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.

1.4.1.2.3. 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.

Finally, partial locking is not widely implemented.

2. Definitions and terminology

2.1. Session specific datastore

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

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.

When a private candidate is used by NETCONF, the specific 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.

When a private candidate is used by RESTCONF, the client that created the private candidate configuration is the only client that has access to it over RESTCONF.

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]). When the private candidate datastore is in use, any operations that target the <candidate> configuration datastore (for example, <edit-config> and <edit-data> operations) actually make changes to the private candidate datastore.

Obtaining this private candidate over NETCONF or RESTCONF 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). This merge is performed in two steps and is described further in Section 3.8.2.1. A <discard-changes> operation will revert the private candidate to the branch’s initial state or it’s state at the last <update> (whichever is most recent).

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 any other configuration session.

3. Private candidates solution

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

NETCONF sessions and RESTCONF clients are able to utilize private candidates 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.

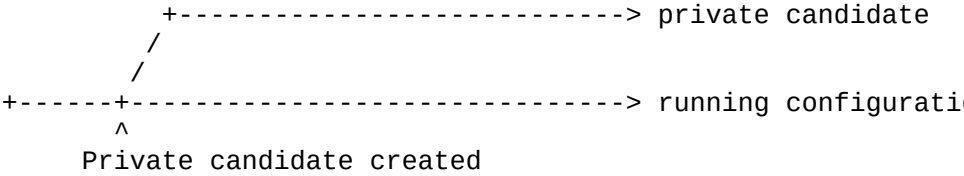
3.1. What is a private candidate

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

3.2. 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.



3.3. When is a private candidate destroyed

A private candidate is valid for the duration of the NETCONF session, or the duration of the RESTCONF request. Issuing a <commit> operation will not close the private candidate but will issue an implicit <update> operation resyncing changes from the running configuration. More details on this later in this document.

A NETCONF session that is operating using a private candidate will discard all uncommitted changes in that session's private candidate and destroy the private candidate if the session is closed through a deliberate user action or disconnected for any other reason (such as a loss of network connectivity).

3.4. When is a private candidate updated

An <update> operation performed on a private candidate triggers a rebase of the private candidate configuration datastore against the running configuration datastore. This rebase is conceptually described as replacing the candidate configuration datastore with the current running configuration datastore at that point in time and replaying the configuration changes from the candidate configuration datastore against it, identifying any conflicts as this process progresses.

A server may choose to automatically issue an update when any client updates the running configuration datastore. Triggering an update in this manner must be specifically chosen and signalled by a server using an optional parameter to the :private-candidate NETCONF capability (see Section 3.5).

A <commit> operation always issues an implicit <update>, therefore it is not necessary for a client to perform an <update> before a <commit>. The implicit <update> will fail if conflicts are found.

More details on the <update> operation are provided later in this document.

3.5. How to signal the use of private candidates

3.5.1. Server

The server MUST signal its support for private candidates. The server does this by advertising a new :private-candidate capability:

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

As support for the shared candidate configuration datastore is a pre-requisite for the private candidate functionlity, a server MUST also advertise the :candidate capability as defined in [RFC6241].

If a server chooses to execute an update operation automatically when any client (not just the local client) performs an update to the running configuration datastore it must advertise this behaviour using the following optional parameter to the capability: trigger=all-updates.

3.5.2. NETCONF client

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

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 using this approach both the NETCONF server and the NETCONF client MUST advertise this capability.

If a client sends the capability and the server does not support private candidates, the server MUST choose one of the following options:

* Ignore the capability and continue with the session without the use of private candidates (this ensures backwards compatibility with older servers).

* Close the session as the requested functionality cannot be provided.

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 or impact the shared candidate configuration datastore will implicitly target or impact the session's private candidate configuration datastore instead, and operate in exactly the same way.

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

3.5.3. RESTCONF client

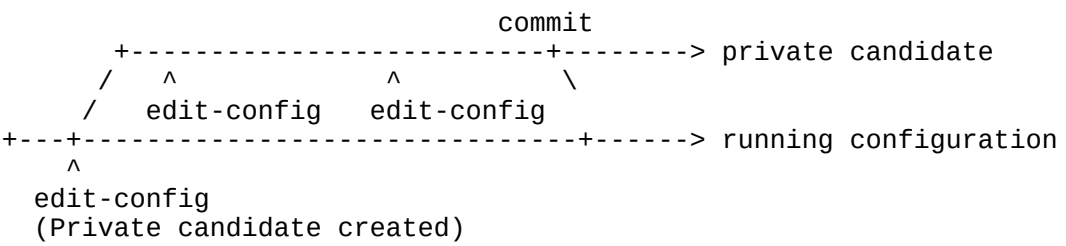
RESTCONF does not provide a mechanism for the client to advertise a capability. Therefore when a RESTCONF server advertises the :private-candidate capability, all client requests will use a private candidate when the client references the "{+restconf}/data" resource described in Section 3.3.1 of [RFC8040]. All edits are made to the client's private candidate, and the private candidate is automatically committed.

This ensures backwards compatibility with RESTCONF clients that are not aware of private candidates, because those clients will expect their changes to be committed immediately.

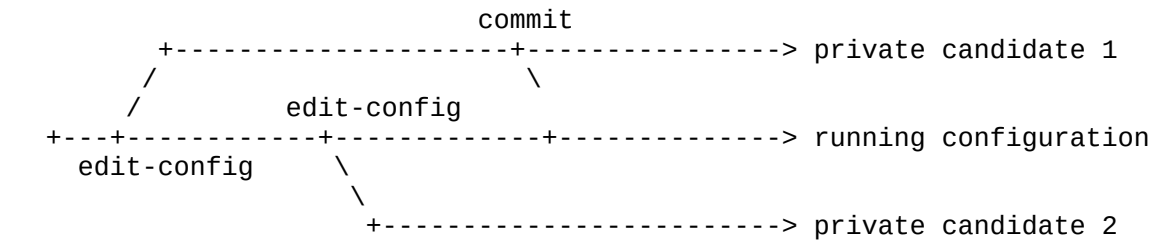
3.6. Interaction between running and private-candidate(s)

Multiple 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 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 its private candidate configuration that are different from the running configuration datastore 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.

A good way to understand the interaction between candidates is to consider them as branches such as you might find in a source code management system.

Each private candidate is treated as a separate branch and changes made to the running configuration are not placed into a 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 MUST automatically issue an <update> operation immediately prior to committing the configuration)

* An implementation chooses to perform an <update> operation after a change to the running configuration by any other client.

It is possible for a private candidate configuration to become significantly out of sync with the running configuration should the private candidate be open for a long time, however, most NETCONF configuration activities (between the first <edit-config>/<edit-data> and a <commit>) are short-lived.

An implementation may choose, optionally, to automatically perform an <update> operation on each private candidate configuration datastore after a change to the running configuration from another client. In this situation, the client will be unaware of these changes to its private candidate configuration datastore, and issuing a manual <update> operation will be a no-op.

A <compare> [RFC9144] operation MAY be performed against:

* The initial creation point of the private candidate’s branch
* Against the last update point of the private candidate’s branch
* Against the running configuration

3.7. Detecting and resolving conflicts

3.7.1. What is a conflict?

A conflict is when the intent of the 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 activity (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 for each node modified in the running configuration that is also modified in the private candidate configuration.

A node is considered modified if:

* There is a change of any value
* There is a change of existence (or otherwise) of any list entry
* There is a change to the order of any list items in a list configured as "ordered-by user"
* There is a change of existence (or otherwise) of a presence container

* There is a change of any component member of a leaf-list

* There is a change to the order of any items in a leaf-list configured as "ordered-by user"

* There is a change of existence (or otherwise) of a leaf

* There is a change to any YANG metadata associated with the node

A server MAY choose to add extra checks in addition to the list above.

If a server implements the transaction ID feature then this MAY be considered as part of detecting a conflict.

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.

3.7.2. 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 server running an <update> operation, such as when a <commit> operation is performed by the client in the private candidate session.

When a conflict is identified that node is internally marked by the server as "in conflict" in the private candidate. This "in conflict" status does not propagate back up the tree to the parent node(s). Each node in the ancestral tree is evaluated as in conflict or otherwise on its own merits. The "in conflict" marker remains until the conflict is resolved on that node.

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).

* A <commit> operation (that MUST trigger an automatic <update> operation immediately before) MUST fail irrespective of any system-wide default resolution-mode. It MUST inform the client of the conflict and SHOULD detail the location of the conflict(s).

* An <update> operation triggered by a client (excluding updates triggered by the <commit> operation) MUST fail unless the server has explicitly configured a system-wide resolution-mode of prefer-candidate or prefer-running (discussed later in this document).

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

Note: If a server implementation has chosen to automatically issue an <update> operation every time a change is made to the running configuration, the server will use the system-wide system resolution mode. If this resolution mode is prefer-candidate or prefer-running the conflicts will be resolved using those rules. If the resolution mode is set to revert-on-conflict the semantics are the same as the prefer-candidate method, however, all changes, whether in conflict or otherwise will be marked in the private candidate as "in-conflict". This means that any subsequent <commit> will fail until the client makes a conscious choice to resolve them.

3.7.3. Conflict resolution

Conflict resolution defines which configuration elements are retained when a conflict is resolved; those from the running configuration or those from the private candidate configuration.

When a conflict is detected in any client-triggered activity, the client MUST be informed. The client then has a number of options available to resolve the conflict:

* Edit the candidate configuration nodes marked as "in conflict". When a node marked as "in conflict" is subsequently edited the "in conflict" marker on that node is removed.

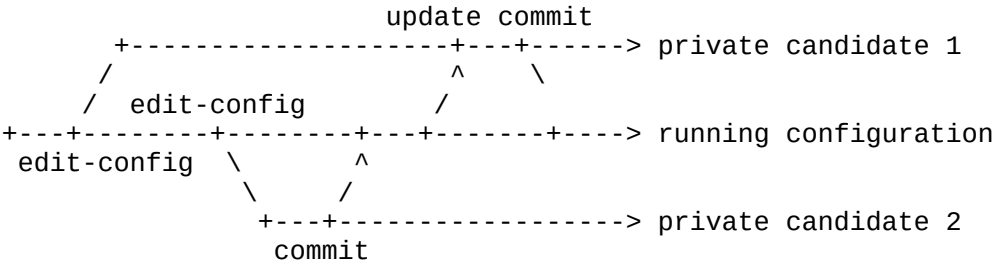
* Issue an <update> operation with a specific resolution-mode.

The <update> operation uses the resolution-mode specified in the request, or the system resolution mode in specific circumstances, to resolve the conflicts. The <update> operation is discussed later in this document.

The following configuration data is used below to illustrate the behaviour 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 example workflow is shown in this diagram and is used for the purpose of the examples below. In these examples the reader should assume that the <update> operation is manually provided by a client working in private candidate 1. The update operation triggered by a system upon commit is not shown.



There are three defined resolution methods:

3.7.3.1. Prefer-candidate

Reminder: The starting configuration and workflow used to illustrate this resolution method is detailed in Section 3.7.3.

When using the prefer-candidate 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 commits 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>prefer-candidate</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>
```

3.7.3.2. Prefer-running

Reminder: The starting configuration and workflow used to illustrate this resolution method is detailed in Section 3.7.3.

When using the prefer-running 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 commits 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>prefer-running</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>
```

3.7.3.3. Revert-on-conflict

Reminder: The starting configuration and workflow used to illustrate this resolution method is detailed in Section 3.7.3.

When using the revert-on-conflict resolution method, an 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 MUST be the default resolution method as it provides for the highest level of visibility and control to ensure operational stability.

This resolution method MUST be supported by a server.

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 commits 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, the update fails with an <rpc-error> 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>
```

3.7.4. System resolution mode and advertisement of this mode

The system resolution mode is revert-on-conflict. However, a system MAY choose to select a different system resolution mode.

The system resolution mode only takes effect if a server implementation performs an automatic update to all private candidate configurations following a commit to running.

The system resolution mode MUST be advertised in the :private-candidate capability by adding the default-resolution-mode parameter if the system default is anything other than revert-on-conflict. If the system default resolution mode is revert-on-conflict then advertising this in the :private-candidate capability is optional.

In this example, a server has configured a default system-wide resolution mode of prefer-running which MUST be signalled with the :private-candidate capability as follows:

```
urn:ietf:params:netconf:capability:private-candidate:1.0
              ?default-resolution-mode=prefer-running
```

3.7.5. Supported resolution modes

A server SHOULD support all three resolution modes. However, if the server does not support all three modes, the server MUST report the supported modes in the :private-candidate capability using the supported-resolution-modes, for example:

```
urn:ietf:params:netconf:capability:private-candidate:1.0
              ?supported-resolution-modes=revert-on-conflict,prefer-candidate
```

3.8. NETCONF operations

3.8.1. New NETCONF operations

3.8.1.1. <update>

The <update> operation triggers a rebase of the private candidate configuration against the running configuration. This rebase is conceptually described as replacing the candidate configuration with the current running configuration at that point in time and replaying the configuration changes from the candidate against it, identifing any conflicts as this process progresses.

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

The <update> operation is atomic. This means that the entire <update> operation succeeeds or the entire <update> operation MUST fail.

The <update> operation MUST be implicitly triggered by a specific NETCONF session issuing a <commit> operation when using private candidates. This implicit <update> operation always has a resolution mode of revert-on-conflict. The actual order of operations in the server MUST be to issue the implicit <update> operation first and then the <commit> operation.

A <commit> operation that fails the implicit <update> operation MUST fail. The client is then required to make a specific decision to rectify the issue prior to committing. This may be to edit the private candidate configuration or to issue a manual <update> operation with a specific resolution mode selected.

3.8.1.1.1. <resolution-mode> parameter

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

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

* revert-on-conflict (default)

* prefer-candidate

* prefer-running

3.8.2. Updated NETCONF operations

Specific NETCONF operations altered by this document are listed in this section.

3.8.2.1. <commit>

The behaviour of the <commit> operation is updated such that processing a <commit> MUST follow a two step process in order to copy the proposed private configuration into the running configuration. These steps are:

1. An implicit update operation MUST be performed by the server using the resolution-mode revert-on-conflict (described earlier in this document).

2. On successful completion of step 1, the private candidate configuration is copied into the running configuration replacing the current contents.

3.8.2.1.1. Interactions with commit confirmed operations and private candidates

Nothing in this document alters the behaviour of the <confirmed>, <persist> or <persist-id> parameters and these MUST work when using the a private candidate configuration if the :confirmed-commit capability is advertised.

When a private candidate is committed using the <confirmed/> parameter and the commit operation disconnects the client's session, the configuration in the running configuration is immediately reverted and the proposed client changes are discarded.

When a private candidate is committed using the <confirmed/> parameter and the commit operation does not disconnect the client's session, and subsequently, the commit operation is either cancelled using the <cancel-commit> operation or the timeout expires, the running configuration is reverted and the proposed client changes are returned to the client's private candidate.

If a private candidate is committed using the <confirmed/> parameter and the <persist> parameter is provided, and the client subsequently disconnects its session for any reason whilst the timer is running, upon cancellation using the <cancel-commit> operation or on the expiry of the timer, the running configuration will be reverted, and the proposed client changes are discarded.

3.8.2.2. <get-config>

Performing a <get-config> operation with the candidate configuration datastore as the source input parameter will instead act on the private candidate datastore, and return data from that datastore. If no private candidate configuration has been created, one will be created at this point.

3.8.2.3. <edit-config>

Performing an <edit-config> operation with the candidate configuration datastore as the target, the changes will be placed into the private candidate configuration datastore. If no private candidate configuration has been created, one will be created at this point.

3.8.2.4. <copy-config>

When performing a <copy-config> operation with the candidate configuration datastore as the source or target, the private candidate will be used. If no private candidate configuration has been created, one will be created at this point.

3.8.2.5. <get-data>

Performing a <get-data> operation with the candidate configuration datastore as the datastore, the configuration from the private candidate will be returned. If no private candidate configuration has been created, one will be created at this point.

3.8.2.6. <edit-data>

Performing an <edit-data> operation with the candidate configuration datastore as the datastore, the changes will be placed into the private candidate configuration datastore. If no private candidate configuration has been created, one will be created at this point.

3.8.2.7. <compare>

Performing a <compare> [RFC9144] operation with the candidate datastore, operating as a private candidate, 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 is extended by this document to allow the ability to compare the private candidate (at its current point in time) with the same private candidate at an earlier point in time, or with another datastore.

3.8.2.7.1. <reference-point> parameter

This document adds the optional <reference-point> node to the input of the <compare> operation that accepts the following values:

* last-update

* creation-point

Servers MAY support this functionality but are not required to by this document.

The last-update selection of <reference-point> will provide an output comparing the current private candidate configuration with the same private candidate configuration at the time it was last updated using the <update> NETCONF operation described in this document (whether automatically or manually triggered).

The creation-point selection of <reference-point> will provide an output comparing the current private candidate configuration datastore with the same private candidate configuration datastore at it was first created.

3.8.2.8. <lock>

The behaviour of the <lock> operation is updated such that locking the candidate configuration datastore will lock that session's private candidate configuration datastore only.

3.8.2.9. <unlock>

The behaviour of the <unlock> operation is updated such that unlocking the candidate configuration datastore will unlock that session's private candidate configuration datastore only.

3.8.2.10. <delete-config>

The behaviour of the <delete-config> operation is updated such that deleting the private candidate will destroy the private candidate for that session. A new one will subsequently be created on first access as described in Section 3.2.

3.8.2.11. <discard-changes>

The behaviour of the <discard-changes> operation is updated such that discarding the changes in a private candidate will reset it to the state it was when it was initially created, or to the state following the latest <update> operation, whichever is most recent. This state may not match the current running configuration.

To align the private candidate with the running configuration the <update> or <delete-config> operations may be used.

3.8.2.12. <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.

3.8.2.13. <cancel-commit>

The <cancel-commit> operation is unchanged. Any changes made to the running configuration are returned to the private candidate if it still exists. See Section 3.8.2.1.1 for further details.

4. IANA Considerations

This document requests the registration the the following NETCONF capabilities:

* urn:ietf:params:netconf:capability:private-candidate:1.0 (Version 1.0)

5. Security Considerations

This document should not affect the security of the Internet.

6. References

6.1. 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>.

[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>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8526] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "NETCONF Extensions to Support the Network Management Datastore Architecture", RFC 8526, DOI 10.17487/RFC8526, March 2019, <https://www.rfc-editor.org/info/rfc8526>.

6.2. Informative References

Appendix A. YANG modules

A.1. ietf-netconf-private-candidate@2024-09-12.yang

```
<CODE BEGINS> file "ietf-netconf-private-candidate@2025-10-30.yang"
module ietf-netconf-private-candidate {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-private-candidate";
  prefix pc;

  import ietf-nmda-compare {
    prefix cmp;
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/netconf/>
     WG List:  <netconf@ietf.org>

     Editor:   James Cumming
            <james.cumming@nokia.com>

     Editor:   Robert Wills
            <rowills@cisco.com>";
  description
    "NETCONF private candidate support.

     Copyright (c) <year> IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject to
     the license terms contained in, the Revised BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
```

```
        (https://trustee.ietf.org/license-info).

        This version of this YANG module is part of RFC XXXX
        (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
        for full legal notices.

        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 (RFC 2119) (RFC 8174) when, and only when,
        they appear in all capitals, as shown here.";

    revision 2025-10-30 {
      description
        "Introduce private candidate support";
      reference
        "draft-ietf-netconf-privcand:
         Netconf Private Candidates";
    }
    revision 2024-09-12 {
      description
        "Introduce private candidate support";
      reference
        "draft-ietf-netconf-privcand:
         Netconf Private Candidates";
    }

    feature private-candidate {
      description
        "NETCONF :private-candidate capability;
         If the server advertises the :private-candidate
         capability for a session, then this feature must
         also be enabled for that session.  Otherwise,
         this feature must not be enabled.";
      reference
        "draft-ietf-netconf-privcand";
    }

    rpc update {
      if-feature "private-candidate";
      description
        "Updates the private candidate from the running
         configuration.";
      reference
        "draft-ietf-netconf-privcand";
      input {
        leaf resolution-mode {
          type enumeration {
```

```
            enum revert-on-conflict {
              description
                "Reject update when any conflicting
                 node is found and revert the private
                 candidate configuration datastore to its
                 state prior to issuing the update.";
            }
            enum prefer-candidate {
              description
                "Resolve conflicted node by selecting
                 the private candidate configuration
                 datastore version.";
            }
            enum prefer-running {
              description
                "Resolve conflicted node by selecting
                 the running configuration datastore
                 version.";
            }
          }
          default "revert-on-conflict";
          description
            "Mode to resolve conflicts between running and
             private-candidate configurations.";
        }
      }
    }

    augment "/cmp:compare/cmp:input" {
      if-feature "private-candidate";
      description
        "Augment private candidate extensions into the NETCONF compare
         RPC";
      leaf reference-point {
        type enumeration {
          enum last-update {
            description
              "Compare using the point the private
               candidate configuration datastore was
               last updated using the update or commit
               RPCs.";
          }
          enum creation-point {
            description
              "Compare using the point the private
               candidate configuration datastore was
               initially created.";
          }
```

```
        }
        default "last-update";
        description
          "When this leaf is provided and the source or
           destination is the candidate datastore, operating
           in private candidate mode, the comparison will
           either occur between the last-update point of
           the private candidate or the creation-point of
           the private candidate.";
        reference
          "draft-ietf-netconf-privcand";
      }
    }
  }
  <CODE ENDS>
```

Acknowledgements

The authors would like to thank Andy Bierman, Jan Lindblad, Lori-Ann McGrath, Dylan Sadoun, Jason Sterne, Kent Watsen and Rob Wilton for their reviews and feedback.

Authors' Addresses

James Cumming
Nokia
Email: james.cumming@nokia.com

Robert Wills
Cisco Systems
Email: rowills@cisco.com