Workgroup: NETCONF
Internet-Draft:
draft-ietf-netconf-transaction-id-00
Published: 9 March 2023
Intended Status: Standards Track
Expires: 10 September 2023
Authors: J. Lindblad
Cisco Systems

Transaction ID Mechanism for NETCONF

Abstract

NETCONF clients and servers often need to have a synchronized view of the server's configuration data stores. The volume of configuration data in a server may be very large, while data store changes typically are small when observed at typical client resynchronization intervals.

Rereading the entire data store and analyzing the response for changes is an inefficient mechanism for synchronization. This document specifies an extension to NETCONF that allows clients and servers to keep synchronized with a much smaller data exchange and without any need for servers to store information about the clients.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Network Configuration Working Group mailing list (netconf@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/netconf/.

Source for this draft and an issue tracker can be found at https://github.com/netconf-wg/transaction-id.

Status of This Memo

This Internet-Draft will expire on 10 September 2023.

Copyright Notice

Table of Contents

1. Introduction

When a NETCONF client wishes to initiate a new configuration transaction with a NETCONF server, a frequently occurring use case is for the client to find out if the configuration has changed since the client last communicated with the server. Such changes could occur for example if another NETCONF client has made changes, or another system or operator made changes through other means than NETCONF.

One way of detecting a change for a client would be to retrieve the entire configuration from the server, then compare the result with a previously stored copy at the client side. This approach is not popular with most NETCONF users, however, since it would often be very expensive in terms of communications and computation cost.

Furthermore, even if the configuration is reported to be unchanged, that will not guarantee that the configuration remains unchanged when a client sends a subsequent change request, a few moments later.

In order to simplify the task of tracking changes, a NETCONF server could implement a meta level transaction tag or timestamp for an entire configuration datastore or YANG subtree, and offer clients a way to read and compare this tag or timestamp. If the tag or timestamp is unchanged, clients can avoid performing expensive operations. Such tags and timestamps are referred to as a transaction id (txid) in this document.

Evidence of a transaction id feature being demanded by clients is that several server implementors have built proprietary and mutually

incompatible mechanisms for obtaining a transaction id from a NETCONF server.

RESTCONF, [RFC8040], defines a mechanism for detecting changes in configuration subtrees based on Entity-Tags (ETags) and Last-Modified txid values.

In conjunction with this, RESTCONF provides a way to make configuration changes conditional on the server confiuguration being untouched by others. This mechanism leverages [RFC7232] "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests".

This document defines similar functionality for NETCONF, [RFC6241], and ties this in with YANG-Push, [RFC8641].

2. Conventions and Definitions

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.

This document uses the terminology defined in [RFC6241], [RFC7950], [RFC8040], and [RFC8641].

In addition, this document defines the following terms:

Versioned node A node in the instantiated YANG data tree for which the server maintains a transaction id (txid) value.

3. NETCONF Txid Extension

This document describes a NETCONF extension which modifies the behavior of get-config, get-data, edit-config, edit-data, discard-changes, copy-config, delete-config and commit such that clients are able to conditionally retrieve and update the configuration in a NETCONF server.

For servers implementing YANG-Push, an extension for conveying txid updates as part of subscription updates is also defined.

Several low level mechanisms could be defined to fulfill the requirements for efficient client-server txid synchronization. This document defines two such mechanisms, the etag txid mechanism and the last-modified txid mechanism. Additional mechanisms could be added in future.

3.1. Use Cases

The common use cases for such mecahnisms are briefly discussed here.

Initial configuration retrieval When the client initially connects to a server, it may be interested to acquire a current view of (parts of) the server's configuration. In order to be able to efficiently detect changes later, it may also be interested to store meta level txid information for subtrees of the configuration.

Subsequent configuration retrieval When a client needs to reread (parts of) the server's configuration, it may be interested to leverage the txid meta data it has stored by requesting the server to prune the response so that it does not repeat configuration data that the client is already aware of.

Configuration update with txid return When a client issues a transaction towards a server, it may be interested to also learn the new txid meta data the server has stored for the updated parts of the configuration.

Conditional configuration change When a client issues a transaction towards a server, it may specify txid meta data for the transaction in order to allow the server to verify that the client is up to date with any changes in the parts of the configuration that it is concerned with. If the txid meta data in the server is different than the client expected, the server rejects the transaction with a specific error message.

Subscribe to configuration changes with txid return When a client subscribes to configuration change updates through YANG-Push, it may be interested to also learn the the updated txid meta data for the changed data trees.

3.2. General Txid Principles

All servers implementing a txid mechanism MUST maintain a txid meta data value for each configuration datastore supported by the server. Txid mechanism implementations MAY also maintain txid meta data values for nodes deeper in the YANG data tree. The nodes for which the server maintains txids are collectively referred to as the "versioned nodes".

The server returning txid values for the versioned nodes MUST ensure the txid values are changed every time there has been a configuration change at or below the node associated with the txid value. This means any update of a config true node will result in a new txid value for all ancestor versioned nodes, up to and including the datastore root itself.

This also means a server MUST update the txid value for any nodes that change as a result of a configuration change, regardless of source, even if the changed nodes are not explicitly part of the change payload. An example of this is dependent data under YANG [RFC7950] when- or choice-statements.

The server MUST NOT change the txid value of a versioned node unless the node itself or a child node of that node has been changed. The server MUST NOT change any txid values due to changes in config false data.

3.3. Initial Configuration Retrieval

When a NETCONF server receives a get-config or get-data request containing requests for txid values, it MUST return txid values for all versioned nodes below the point requested by the client in the reply.

The exact encoding varies by mechanism, but all txid mechanisms would have a special "txid-request" txid value (e.g. "?") which is guaranteed to never be used as a normal txid value. Clients MAY use this special txid value associated with one or more nodes in the data tree to indicate to the server that they are interested in txid values below that point of the data tree.

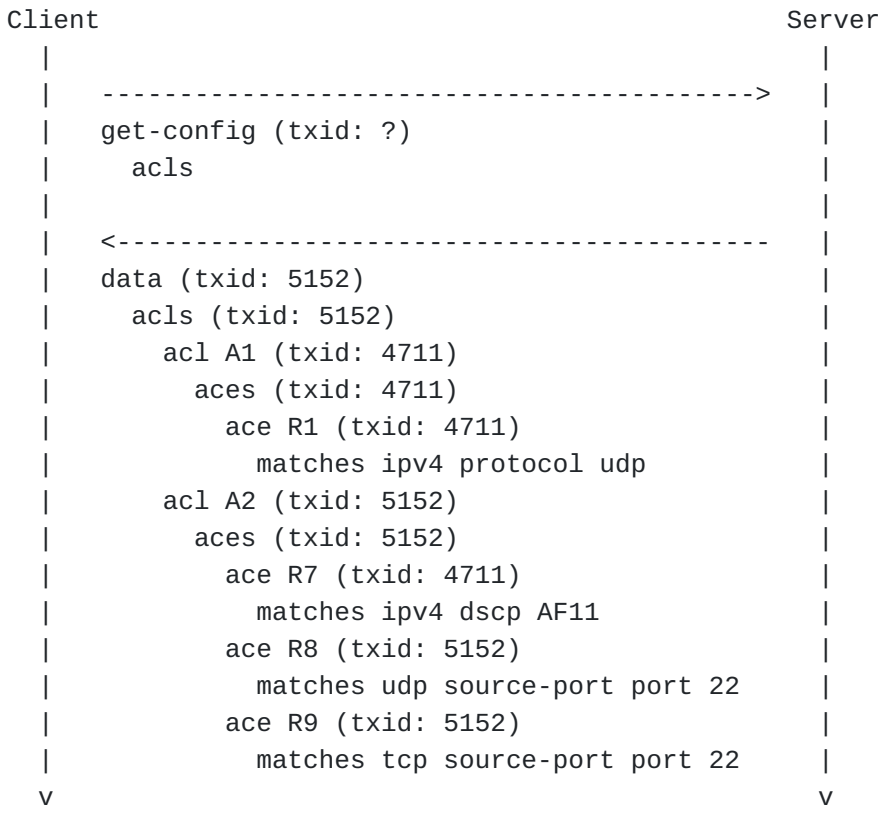

Figure 1: Initial Configuration Retrieval. The server returns the requested configuration, annotated with txid values. The most recent change seems to have been an update to the R8 and R9 source-port.

NOTE: In the call flow examples we are using a 4-digit, monotonously increasing integer as txid. This is convenient and enhances readability of the examples, but does not reflect a typical implementation. Servers may assign values randomly. In general, no information can be derived by observing that some txid value is numerically or lexicographically lower than another txid value. The only operation defined on a pair of txid values is testing them for equality.

3.4. Subsequent Configuration Retrieval

Clients MAY request the server to return txid values in the response by adding one or more txid values received previously in get-config or get-data requests.

When a NETCONF server receives a get-config or get-data request containing a node with a client specified txid value, there are several different cases:

* The node is not a versioned node, i.e. the server does not maintain a txid value for this node. In this case, the server MUST look up the closest ancestor that is a versioned node, and use the txid value of that node as the txid value of this node in the further handling below. The datastore root is always a versioned node.

* The client specified txid value is different than the server's txid value for this node. In this case the server MUST return the contents as it would otherwise have done, adding the txid values of all child versioned nodes to the response. In case the client has specified txid values for some child nodes, then these cases MUST be re-evaluated for those child nodes.

* The client specified txid value matches the server's txid value. In this case the server MUST return the node decorated with a special "txid-match" txid value (e.g. "=") to the matching node, pruning any value and child nodes. A server MUST NOT ever use the txid-match value (e.g. "=") as an actual txid value.

For list elements, pruning child nodes means that top-level key nodes MUST be included in the response, and other child nodes MUST NOT be included. For containers, child nodes MUST NOT be included.

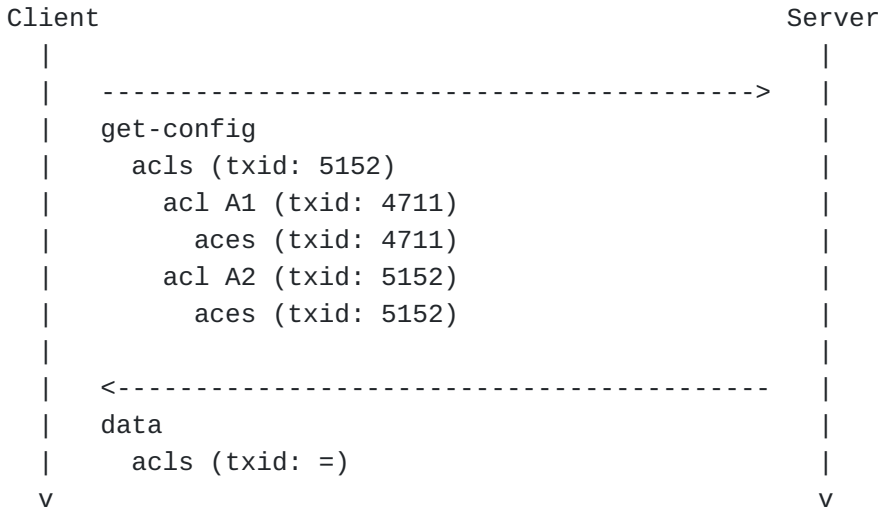

Figure 2: Response Pruning. Client sends get-config request with known txid values. Server prunes response where txid matches expectations.

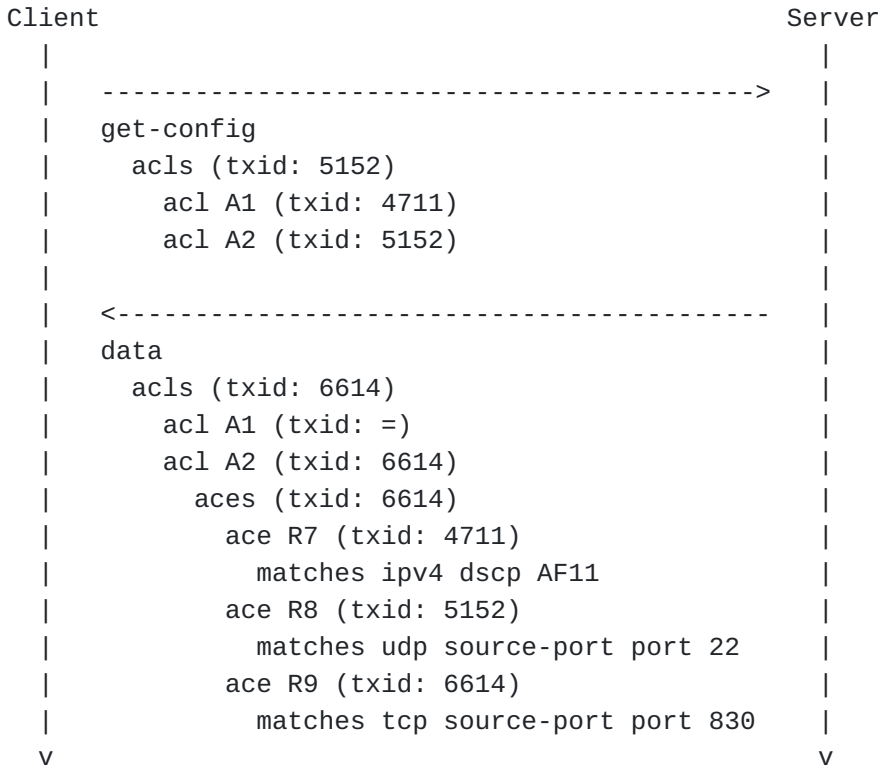

Figure 3: Out of band change detected. Client sends get-config request with known txid values. Server provides update where changes have happened. Specifically ace R8 is returned since ace R8 is a child of a node for which the request had a different txid than the server, and the client did not specify any matching txid for the ace R8 node.

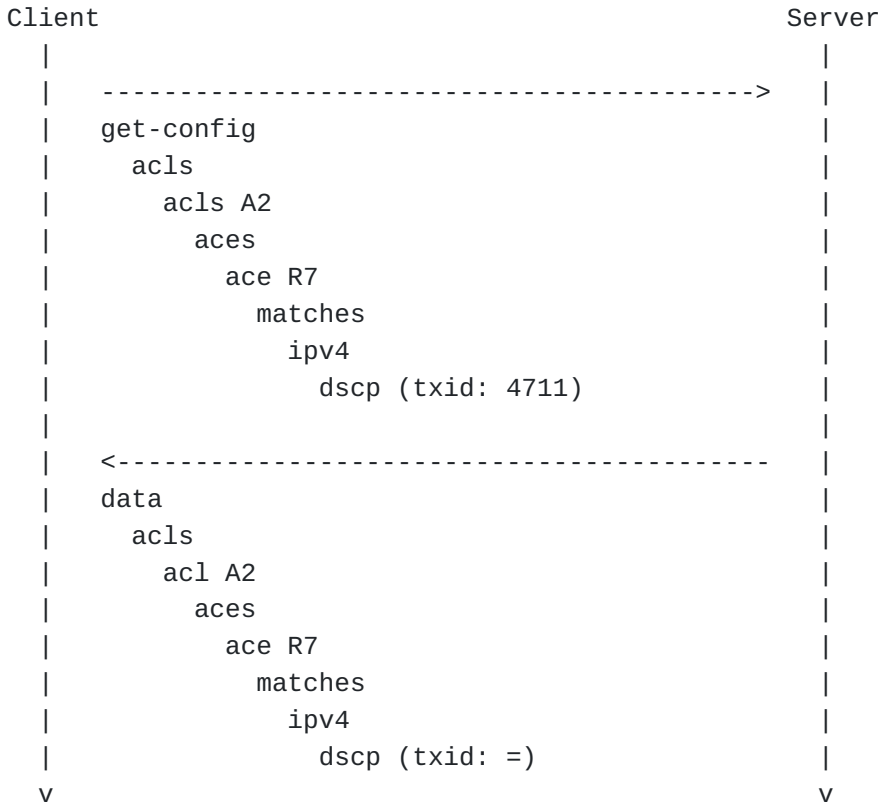

Figure 4: Versioned nodes. Server lookup of dscp txid gives 4711, as closest ancestor is ace R7 with txid 4711. Since the server's and client's txid match, the etag value is '=', and the leaf value is pruned.

3.5. Configuration Retrieval from the Candidate Datastore

When a client retrieves the configuration from the candidate datastore, some of the configuration nodes may hold the same data as the corresponding node in the running datastore. In such cases, the server MUST return the same txid value for nodes in the candidate datastore as in the running datastore.

If a node in the candidate datastore holds different data than in the running datastore, the server has a choice of what to return.

*The server MAY return a txid-unknown value (e.g. "!"). This may be convenient in servers that do not know a priori what txids will be used in a future, possible commit of the canidate.

*If the txid-unknown value is not returned, the server MUST return he txid value the node will have after commit, assuming the client makes no further changes of the candidate datastore.

See the example in Transactions toward the Candidate Datastore (Section 3.7).

3.6. Conditional Transactions

Conditional transactions are useful when a client is interested to make a configuration change, being sure that relevant parts of the server configuration have not changed since the client last inspected it.

By supplying the latest txid values known to the client in its change requests (edit-config etc.), it can request the server to reject the transaction in case any relevant changes have occurred at the server that the client is not yet aware of.

This allows a client to reliably compute and send confiuguration changes to a server without either acquiring a global datastore lock for a potentially extended period of time, or risk that a change from another client disrupts the intent in the time window between a read (get-config etc.) and write (edit-config etc.) operation.

Clients that are also interested to know the txid assigned to the modified versioned nodes in the model immediately in the response could set a flag in the rpc message to request the server to return the new txid with the ok message.

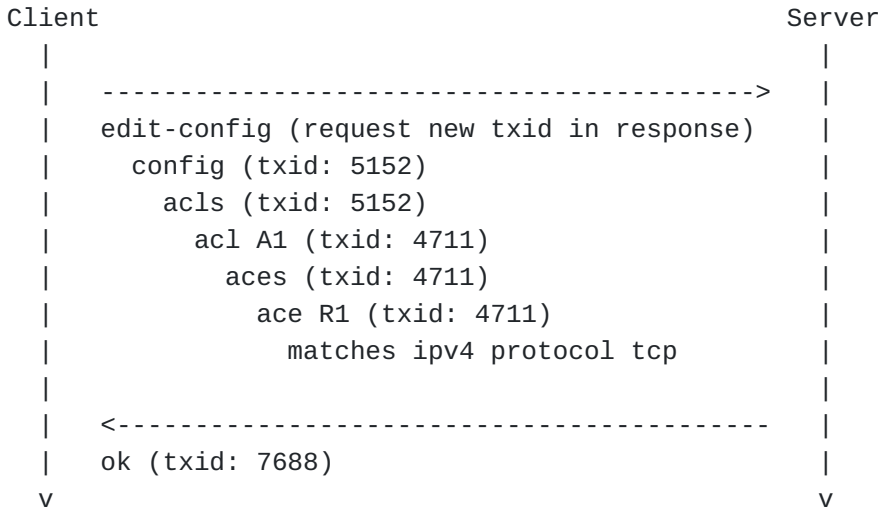

Figure 5: Conditional transaction towards the Running datastore successfully executed. As all the txid values specified by the client matched those on the server, the transaction was successfully executed.

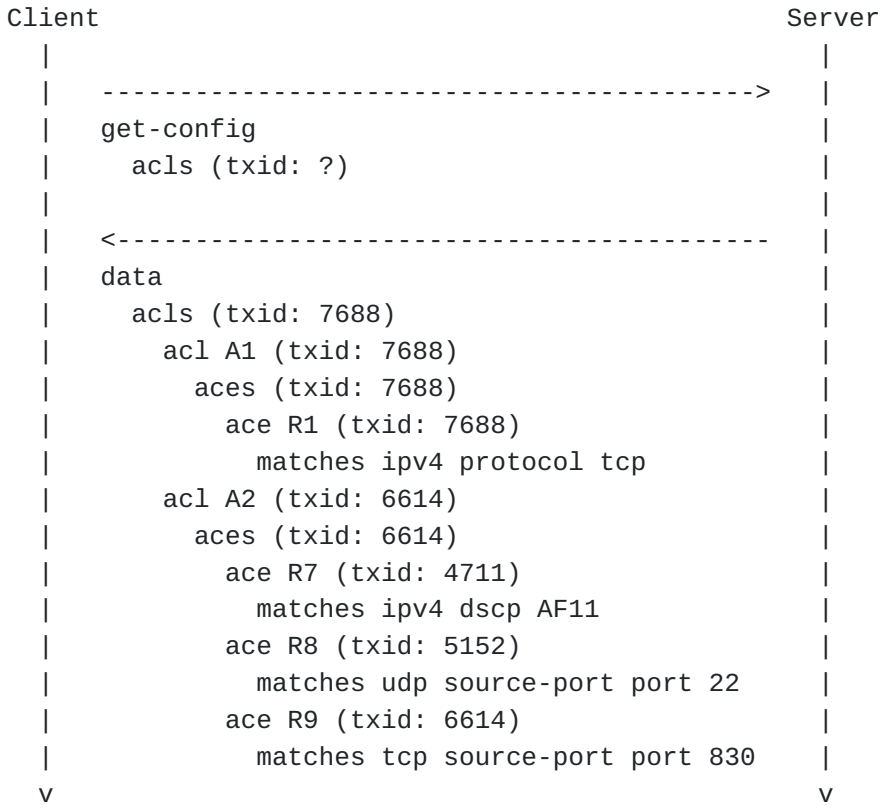

Figure 6: The txids are updated on all versioned nodes that were modified themselves or have a child node that was modified.

If the server rejects the transaction because one or more of the configuration txid value(s) differs from the client's expectation, the server MUST return at least one rpc-error with the following values:

```
error-tag:      operation-failed
error-type:     protocol
error-severity: error
```

Additionally, the error-info tag MUST contain an sx:structure containing relevant details about one of the mismatching txids. A server MAY send multiple rpc-errors when multiple txid mismatches are detected.

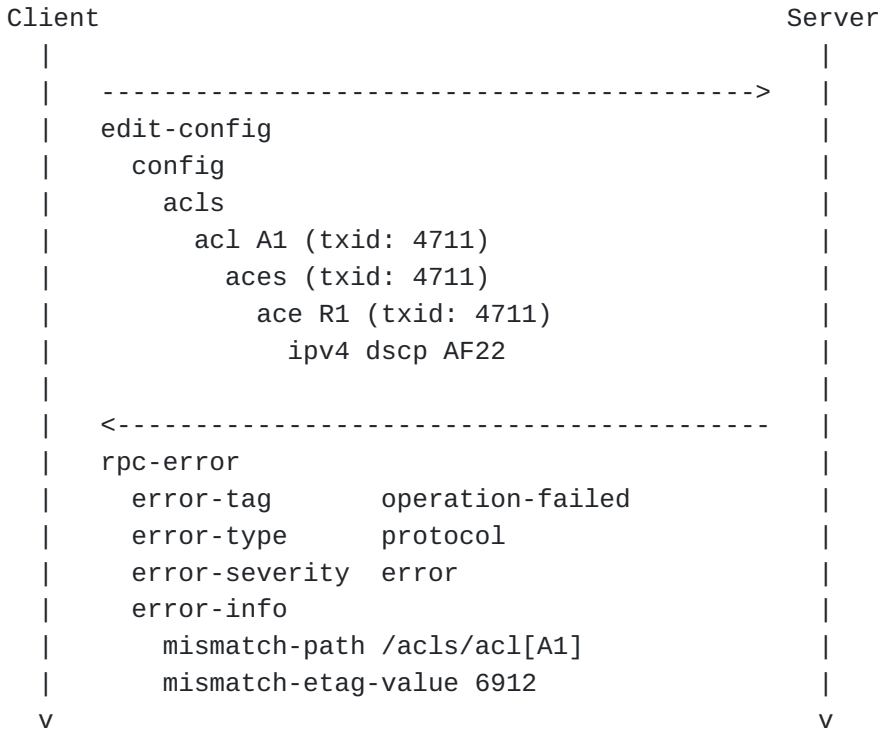

Figure 7: Conditional transaction that fails a txid check. The client wishes to ensure there has been no changes to the particular acl entry it edits, and therefore sends the txid it knows for this part of the configuration. Since the txid has changed (out of band), the server rejects the configuration change request and reports an error with details about where the mismatch was detected.

3.7. Transactions toward the Candidate Datastore

When working with the Candidate datastore, the txid validation happens at commit time, rather than at individual edit-config or edit-data operations. Clients add their txid attributes to the configuration payload the same way. In case a client specifies different txid values for the same element in successive edit-config or edit-data operations, the txid value specified last MUST be used by the server at commit time.

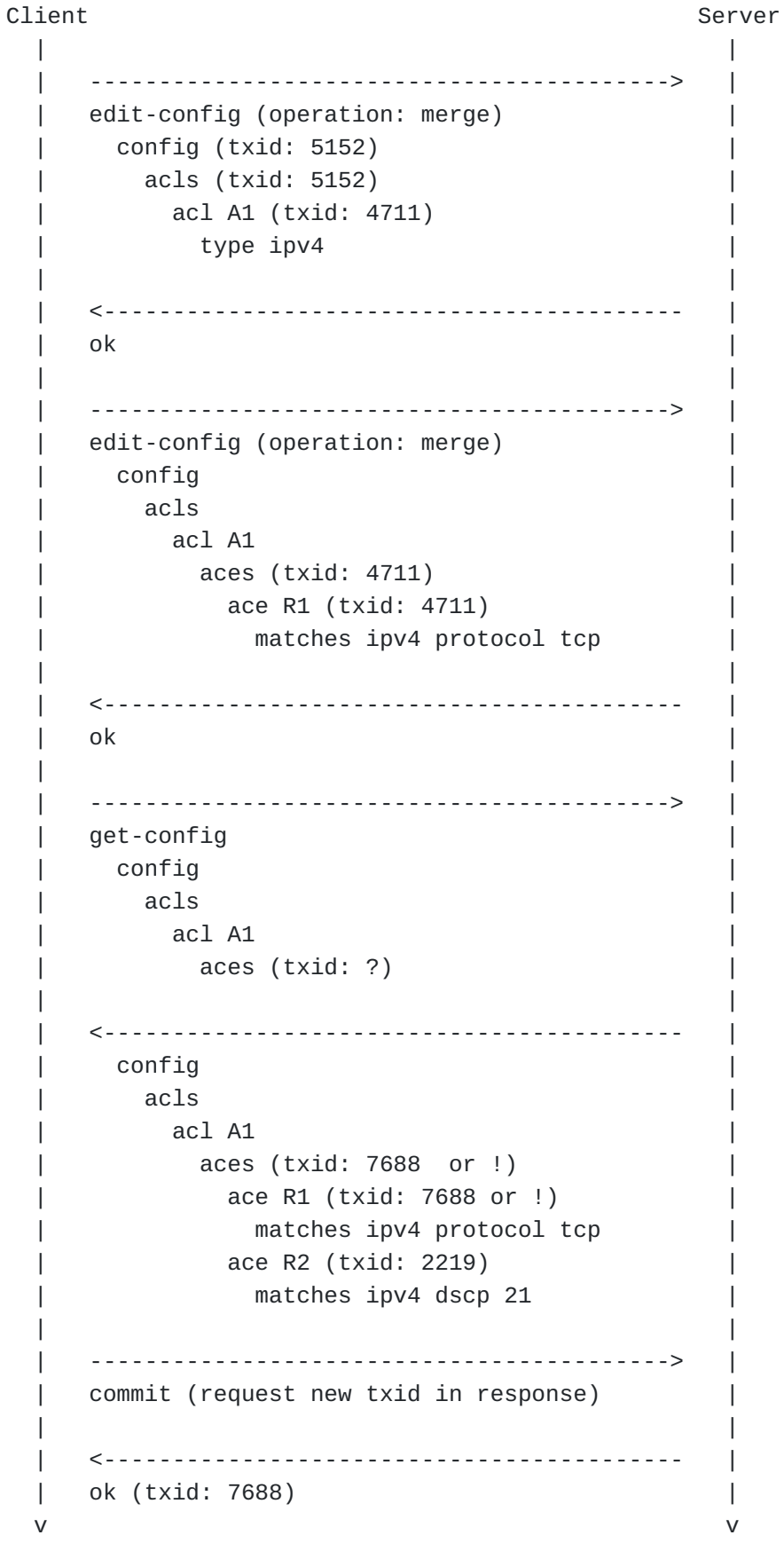
Client
Server
edit-config (operation: merge)
config (txid: 5152)
acls (txid: 5152)
acl A1 (txid: 4711)
type ipv4
ok
edit-config (operation: merge)
config
acls
acl A1
aces (txid: 4711)
ace R1 (txid: 4711)
matches ipv4 protocol tcp
ok
get-config
config
acls
acl A1
aces (txid: ?)
config
acls
acl A1
aces (txid: 7688 or !)
ace R1 (txid: 7688 or !)
matches ipv4 protocol tcp
ace R2 (txid: 2219)
matches ipv4 dscp 21
commit (request new txid in response)
ok (txid: 7688)

Figure 8: Conditional transaction towards the Candidate datastore successfully executed. As all the txid values specified by the client matched those on the server at the time of the commit, the transaction was successfully executed. If a client issues a get-config towards the candidate datastore, the server may choose to return the special txid-unknown value (e.g. "!") or the txid value that would be used if the candidate was committed without further changes (when that txid value is known in advance by the server).

3.8. Dependencies within Transactions

YANG modules that contain when-statements referencing remote parts of the model will cause the txid to change even in parts of the data tree that were not modified directly.

Let's say there is an energy-example.yang module that defines a mechanism for clients to request the server to measure the amount of energy that is consumed by a given access control rule. The energy-example module augments the access control module as follows:

```
module energy-example {
...

  container energy {
    leaf metering-enabled {
      type boolean;
      default false;
    }
  }

  augment /acl:acls/acl:acl {
    when /energy-example:energy/energy-example:metering-enabled;
    leaf energy-tracing {
      type boolean;
      default false;
    }
    leaf energy-consumption {
      config false;
      type uint64;
      units J;
    }
  }
}
```

This means there is a system wide switch leaf metering-enabled in energy-example which disables all energy measurements in the system when set to false, and that there is a boolean leaf energy-tracing that controls whether energy measurement is happening for each acl rule individually.

In this example, we have an initial configuration like this:

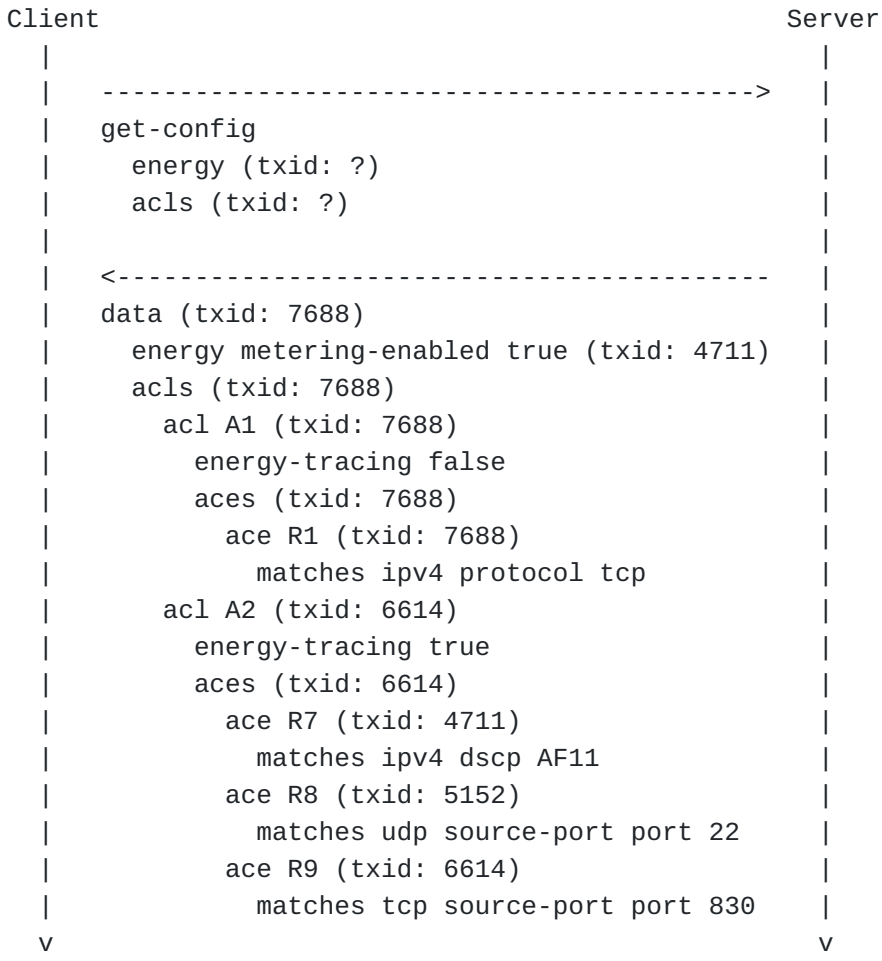

Figure 9: Initial configuration for the energy example. Note the energy metering-enabled leaf at the top and energy-tracing leafs under each acl.

At this point, a client updates metering-enabled to false. This causes the when-expression on energy-tracing to turn false, removing the leaf entirely. This counts as a configuration change, and the txid MUST be updated appropriately.

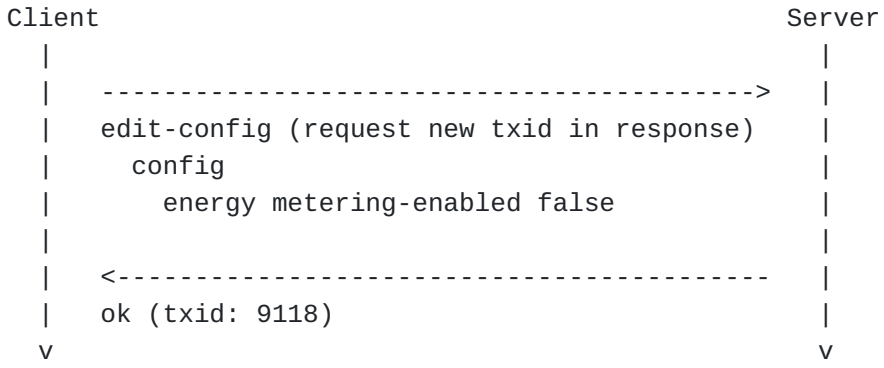

Figure 10: Transaction changing a single leaf. This leaf is the target of a when-statement, however, which means other leafs elsewhere may be indirectly modified by this change. Such indirect changes will also result in txid changes.

After the transaction above, the new configuration state has the energy-tracing leafs removed. Every such removal or (re)introduction of a node counts as a configuration change from a txid perspective, regardless of whether the change has any net configuration change effect in the server.

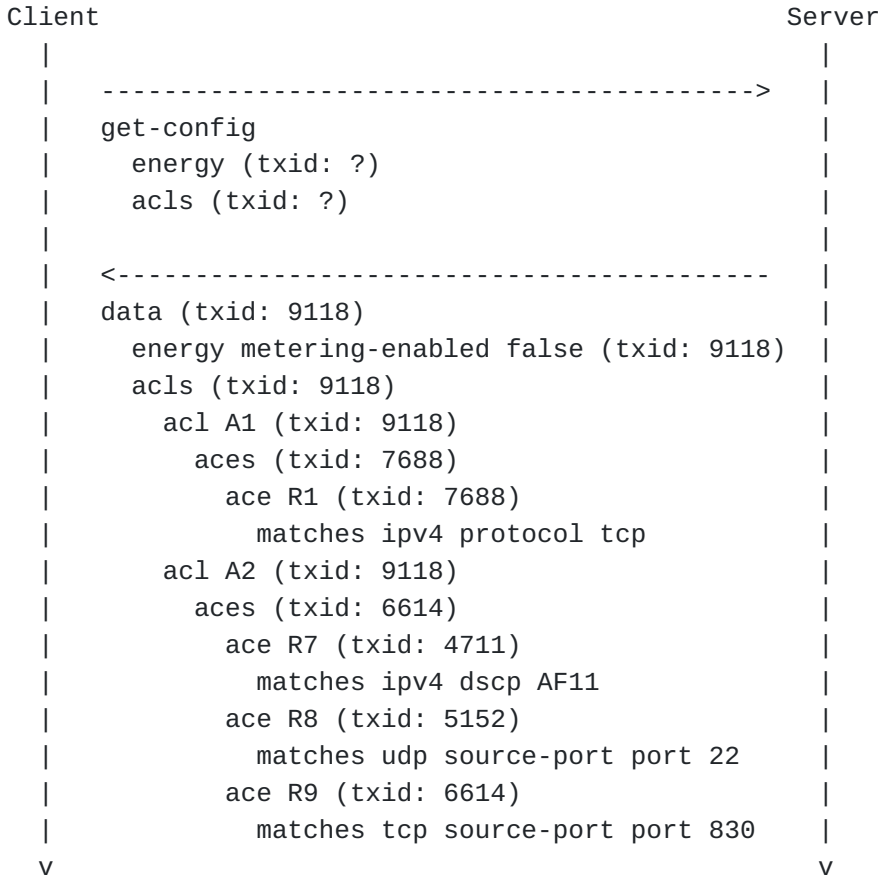

Figure 11: The txid for the energy subtree has changed since that was the target of the edit-config. The txids of the ACLs have also changed since the energy-tracing leafs are now removed by the now false when-expression. Both acl A1 and acl A2 have their txids updated, even though energy-tracing was already false for acl A1.

3.9. Other NETCONF Operations

discard-changes The discard-changes operation resets the candidate datastore to the contents of the running datastore. The server MUST ensure the txid values in the candidate datastore get the same txid values as in the running datastore when this operation runs.

copy-config The copy-config operation can be used to copy contents between datastores. The server MUST ensure the txid values are retained and changed as if the data being copied had been sent in through an edit-config operation.

delete-config The server MUST ensure the datastore txid value is changed, unless it was already empty.

commit At commit, with regards to the txid values, the server MUST treat the contents of the candidate datastore as if any txid value provided by the client when updating the candidate was provided in a single edit-config towards the running datastore. If the transaction is rejected due to txid value mismatch, an rpc-error as described in section Conditional Transactions (Section 3.6) MUST be sent.

3.10. YANG-Push Subscriptions

A client issuing a YANG-Push establish-subscription or modify-subscription request towards a server that supports both YANG-Push [RFC8641] and a txid mechanism MAY request that the server provides updated txid values in YANG-Push subscription updates.

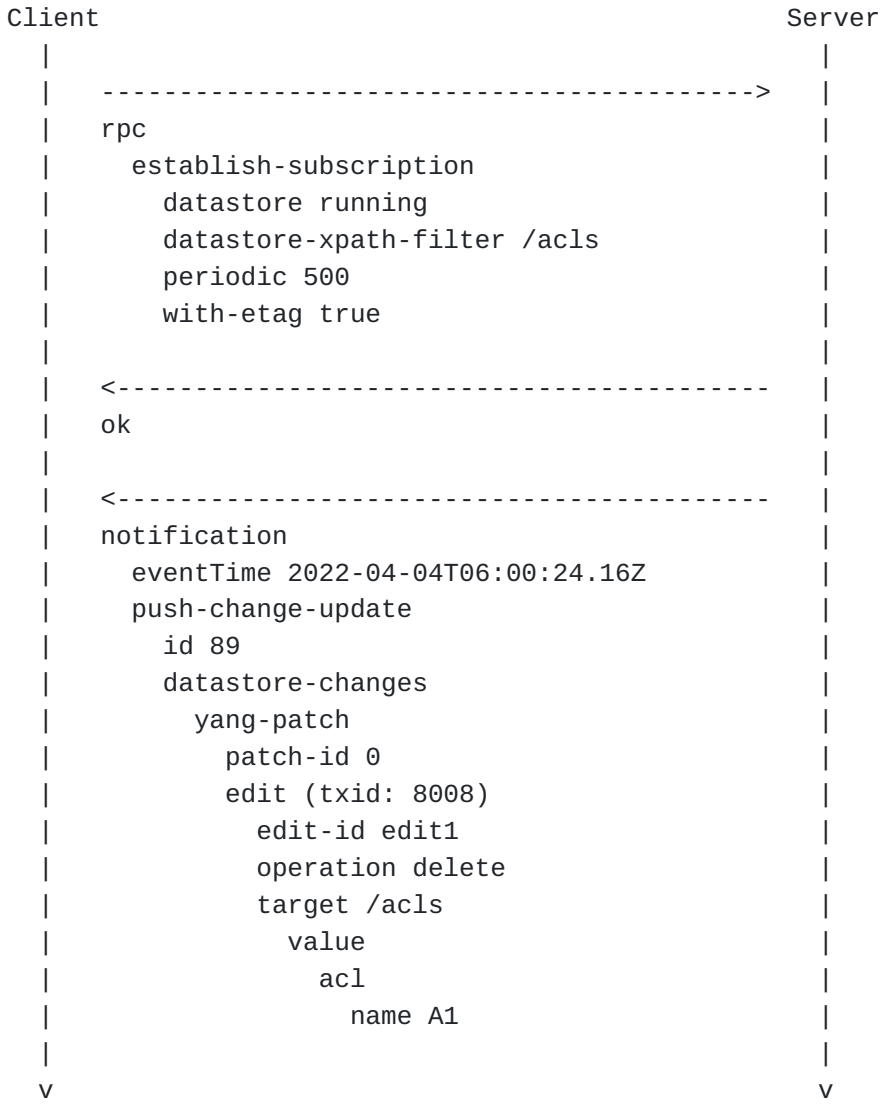

Figure 12: A client requests a YANG-Push subscription for a given path with txid values included. Later, when the server delivers a push-change-update notification, the txid is included.

4. Txid Mechanisms

This document defines two txid mechanisms:

*The etag attribute txid mechanism

*The last-modified attribute txid mechanism

Servers implementing this specification MUST support the etag attribute txid mechanism and MAY support the last-modified attribute txid mechanism.

Section NETCONF Txid Extension (Section 3) describes the logic that governs all txid mechanisms. This section describes the mapping from the generic logic to specific mechanism and encoding.

If a client uses more than one txid mechanism, such as both etag and last-modified in a particular message to a server, or patricular commit, the result is undefined.

4.1. The etag attribute txid mechanism

The etag txid mechanism described in this section is centered around a meta data XML attribute called "etag". The etag attribute is defined in the namespace "urn:ietf:params:xml:ns:netconf:txid:1.0". The etag attribute is added to XML elements in the NETCONF payload in order to indicate the txid value for the YANG node represented by the element.

NETCONF servers that support this extension MUST announce the capability "urn:ietf:params:netconf:capability:txid:etag:1.0".

The etag attribute values are opaque UTF-8 strings chosen freely, except that the etag string must not contain space, backslash or double quotes. The point of this restriction is to make it easy to reuse implementations that adhere to section 2.3.1 in [RFC7232]. The probability SHOULD be made very low that an etag value that has been used historically by a server is used again by that server if the configuration is different.

It is RECOMMENDED that the same etag txid values are used across all management interfaces (i.e. NETCONF, RESTCONF and any other the server might implement), if it implements more than one.

The detailed rules for when to update the etag value are described in section General Txid Principles (Section 3.2). These rules are chosen to be consistent with the ETag mechanism in RESTCONF, [RFC8040], specifically sections 3.4.1.2, 3.4.1.3 and 3.5.2.

4.2. The last-modified attribute txid mechanism

The last-modified txid mechanism described in this section is centered around a meta data XML attribute called "last-modified". The last-modified attribute is defined in the namespace "urn:ietf:params:xml:ns:netconf:txid:1.0". The last-modified attribute is added to XML elements in the NETCONF payload in order to indicate the txid value for the YANG node represented by the element.

NETCONF servers that support this extension MUST announce the capability "urn:ietf:params:netconf:capability:txid:last-modified:1.0".

The last-modified attribute values are yang:date-and-time values as defined in ietf-yang-types.yang, [RFC6991].

"2022-04-01T12:34:56.123456Z" is an example of what this time stamp format looks like. It is RECOMMENDED that the time stamps provided by the server to closely match the real world clock. Servers MUST ensure the timestamps provided are monotonously increasing for as long as the server's operation is maintained.

It is RECOMMENDED that server implementors choose the number of digits of precision used for the fractional second timestamps high enough so that there is no risk that multiple transactions on the server would get the same timestamp.

It is RECOMMENDED that the same last-modified txid values are used across all management interfaces (i.e. NETCONF and any other the server might implement), except RESTCONF.

RESTCONF, as defined in [RFC8040], is using a different format for the time stamps which is limited to one second resolution. Server implementors that support the Last-Modified txid mechanism over both RESTCONF and other management protocols are RECOMMENDED to use Last-Modified timestamps that match the point in time referenced over RESTCONF, with the fractional seconds part added.

The detailed rules for when to update the last-modified value are described in section General Txid Principles (Section 3.2). These rules are chosen to be consistent with the Last-Modified mechanism in RESTCONF, [RFC8040], specifically sections 3.4.1.1, 3.4.1.3 and 3.5.1.

4.3. Common features to both etag and last-modified txid mechanisms

Clients MAY add etag or last-modified attributes to zero or more individual elements in the get-config or get-data filter, in which case they pertain to the subtree(s) rooted at the element(s) with the attributes.

Clients MAY also add such attributes directly to the get-config or get-data tags (e.g. if there is no filter), in which case it pertains to the txid value of the datastore root.

Clients might wish to send a txid value that is guaranteed to never match a server constructed txid. With both the etag and last-modified txid mechanisms, such a txid-request value is "?".

Clients MAY add etag or last-modified attributes to the payload of edit-config or edit-data requests, in which case they indicate the client's txid value of that element.

Clients MAY request servers that also implement YANG-Push to return configuration change subsription updates with etag or last-modified txid attributes. The client requests this service by adding a with-etag or with-last-modified flag with the value 'true' to the subscription request or yang-push configuration. The server MUST then return such txids on the YANG Patch edit tag and to the child elements of the value tag. The txid attribute on the edit tag reflects the txid associated with the changes encoded in this edit section, as well as parent nodes. Later edit sections in the same push-update or push-change-update may still supercede the txid value for some or all of the nodes in the current edit section.

Servers returning txid values in get-config, edit-config, get-data, edit-data and commit operations MUST do so by adding etag and/or last-modified txid attributes to the data and ok tags. When servers prune output due to a matching txid value, the server MUST add a txid-match attribute to the pruned element, and MUST set the attribute value to "=", and MUST NOT send any element value.

Servers returning a txid mismatch error MUST return an rpc-error as defined in section Conditional Transactions (Section 3.6) with an error-info tag containing a txid-value-mismatch-error-info structure.

4.3.1. Candidate Datastore

When servers return txid values in get-config and get-data operations towards the candidate datastore, the txid values returned MUST adhere to the following rules:

* If the versioned node holds the same data as in the running datastore, the same txid value as the versioned node in running MUST be used.

* If the versioned node is different in the candidata store than in the running datastore, the server has a choice of what to return. The server MAY return the special "txid-unknown" value "!". If the txid-unknown value is not returned, the server MUST return the txid value the versioned node will have if the client decides to commit the candidate datastore without further updates.

4.3.2. Namespaces and Attribute Placement

The txid attributes are valid on the following NETCONF tags, where xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0", xmlns:ncds="urn:ietf:params:xml:ns:yang:ietf-netconf-nmda", xmlns:sn="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications", xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-patch" and xmlns:ypatch="urn:ietf:params:xml:ns:yang:ietf-yang-patch":

In client messages sent to a server:

```
  */nc:rpc/nc:get-config

  */nc:rpc/nc:get-config/nc:filter//*

  */nc:rpc/ncds:get-data

  */nc:rpc/ncds:get-data/ncds:subtree-filter//*

  */nc:rpc/ncds:get-data/ncds:xpath-filter//*

  */nc:rpc/nc:edit-config/nc:config

  */nc:rpc/nc:edit-config/nc:config//*

  */nc:rpc/ncds:edit-data/ncds:config

  */nc:rpc/ncds:edit-data/ncds:config//*
```

In server messages sent to a client:

```
  */nc:rpc-reply/nc:data

  */nc:rpc-reply/nc:data//*

  */nc:rpc-reply/ncds:data

  */nc:rpc-reply/ncds:data//*

  */nc:rpc-reply/nc:ok

  */yp:push-update/yp:datastore-contents/ypatch:yang-patch/
   ypatch:edit

  */yp:push-update/yp:datastore-contents/ypatch:yang-patch/
   ypatch:edit/ypatch:value//*

  */yp:push-change-update/yp:datastore-contents/ypatch:yang-patch/
   ypatch:edit

  */yp:push-change-update/yp:datastore-contents/ypatch:yang-patch/
   ypatch:edit/ypatch:value//*
```

5. Txid Mechanism Examples

5.1. Initial Configuration Response

5.1.1. With etag

NOTE: In the etag examples below, we have chosen to use a txid value consisting of "nc" followed by a monotonously increasing integer. This is convenient for the reader trying to make sense of the examples, but is not an implementation requirement. An etag would often be implemented as a "random" string of characters, with no comes-before/after relation defined.

To retrieve etag attributes across the entire NETCONF server configuration, a client might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <get-config txid:etag="?"/>
</rpc>
```

The server's reply might then be:

```
<rpc-reply message-id="1"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data txid:etag="nc5152">
    <acls xmlns=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list"
          txid:etag="nc5152">
      <acl txid:etag="nc4711">
        <name>A1</name>
        <aces txid:etag="nc4711">
          <ace txid:etag="nc4711">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>udp</protocol>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
      <acl txid:etag="nc5152">
        <name>A2</name>
        <aces txid:etag="nc5152">
          <ace txid:etag="nc4711">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>AF11</dscp>
              </ipv4>
            </matches>
          </ace>
          <ace txid:etag="nc5152">
            <name>R8</name>
            <matches>
              <udp>
                <source-port>
                  <port>22</port>
                </source-port>
              </udp>
            </matches>
          </ace>
          <ace txid:etag="nc5152">
            <name>R9</name>
            <matches>
              <tcp>
                <source-port>
                  <port>22</port>
                </source-port>
              </tcp>
```

```
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
    <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"
          txid:etag="nc3072">
      <groups txid:etag="nc3072">
        <group txid:etag="nc3072">
          <name>admin</name>
          <user-name>sakura</user-name>
          <user-name>joe</user-name>
        </group>
      </groups>
    </nacm>
  </data>
</rpc>
```

To retrieve etag attributes for a specific ACL using an xpath filter, a client might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="xpath"
      xmlns:acl=
        "urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      select="/acl:acls/acl:acl[acl:name='A1']"
      txid:etag="?"/>
  </get-config>
</rpc>
```

To retrieve etag attributes for "acls", but not for "nacm", a client might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="3"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <acls
        xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
        txid:etag="?"/>
      <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"/>
    </filter>
  </get-config>
</rpc>
```

If the server considers "acls", "acl", "aces" and "acl" to be versioned nodes, the server's response to the request above might look like:

```
<rpc-reply message-id="3"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls xmlns=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list"
          txid:etag="nc5152">
      <acl txid:etag="nc4711">
        <name>A1</name>
        <aces txid:etag="nc4711">
          <ace txid:etag="nc4711">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>udp</protocol>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
      <acl txid:etag="nc5152">
        <name>A2</name>
        <aces txid:etag="nc5152">
          <ace txid:etag="nc4711">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>AF11</dscp>
              </ipv4>
            </matches>
          </ace>
          <ace txid:etag="nc5152">
            <name>R8</name>
            <matches>
              <udp>
                <source-port>
                  <port>22</port>
                </source-port>
              </udp>
            </matches>
          </ace>
          <ace txid:etag="nc5152">
            <name>R9</name>
            <matches>
              <tcp>
                <source-port>
                  <port>22</port>
                </source-port>
              </tcp>
```

```
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
    <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"/>
      <groups>
        <group>
          <name>admin</name>
          <user-name>sakura</user-name>
          <user-name>joe</user-name>
        </group>
      </groups>
    </nacm>
  </data>
</rpc>
```

5.1.2. With last-modified

To retrieve last-modified attributes for "acls", but not for "nacm", a client might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="4"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <acls
        xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
        txid:last-modified="?"/>
      <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"/>
    </filter>
  </get-config>
</rpc>
```

If the server considers "acls", "acl", "aces" and "acl" to be versioned nodes, the server's response to the request above might look like:

```
<rpc-reply message-id="4"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      txid:last-modified="2022-04-01T12:34:56.789012Z">
      <acl txid:last-modified="2022-03-20T16:20:11.333444Z">
        <name>A1</name>
        <aces txid:last-modified="2022-03-20T16:20:11.333444Z">
          <ace txid:last-modified="2022-03-20T16:20:11.333444Z">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>udp</protocol>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
      <acl txid:last-modified="2022-04-01T12:34:56.789012Z">
        <name>A2</name>
        <aces txid:last-modified="2022-04-01T12:34:56.789012Z">
          <ace txid:last-modified="2022-03-20T16:20:11.333444Z">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>AF11</dscp>
              </ipv4>
            </matches>
          </ace>
          <ace txid:last-modified="2022-04-01T12:34:56.789012Z">
            <name>R8</name>
            <matches>
              <udp>
                <source-port>
                  <port>22</port>
                </source-port>
              </udp>
            </matches>
          </ace>
          <ace txid:last-modified="2022-04-01T12:34:56.789012Z">
            <name>R9</name>
            <matches>
              <tcp>
                <source-port>
                  <port>22</port>
                </source-port>
              </tcp>
```

```
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
    <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"/>
      <groups>
        <group>
          <name>admin</name>
          <user-name>sakura</user-name>
          <user-name>joe</user-name>
        </group>
      </groups>
    </nacm>
  </data>
</rpc>
```

5.2. Configuration Response Pruning

A NETCONF client that already knows some txid values MAY request that the configuration retrieval request is pruned with respect to the client's prior knowledge.

To retrieve only changes for "acls" that do not have the last known etag txid value, a client might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="6"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <acls
        xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
        txid:etag="nc5152">
        <acl txid:etag="nc4711">
          <name>A1</name>
          <aces txid:etag="nc4711"/>
        </acl>
        <acl txid:etag="nc5152">
          <name>A2</name>
          <aces txid:etag="nc5152"/>
        </acl>
    </filter>
  </get-config>
</rpc>
```

Assuming the NETCONF server configuration is the same as in the previous rpc-reply example, the server's response to request above might look like:

```
<rpc-reply message-id="6"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      txid:etag="="/>
  </data>
</rpc>
```

Or, if a configuration change has taken place under /acls since the client was last updated, the server's response may look like:

```
<rpc-reply message-id="6"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      txid:etag="nc6614">
      <acl txid:etag="=">
        <name>A1</name>
      </acl>
      <acl txid:etag="nc6614">
        <name>A2</name>
        <aces txid:etag="nc6614">
          <ace txid:etag="nc4711">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>AF11</dscp>
              </ipv4>
            </matches>
          </ace>
          <ace txid:etag="nc5152">
            <name>R8</name>
            <matches>
              <ipv4>
                <source-port>
                  <port>22</port>
                </source-port>
              </ipv4>
            </matches>
          </ace>
          <ace txid:etag="nc6614">
            <name>R9</name>
            <matches>
              <ipv4>
                <source-port>
                  <port>830</port>
                </source-port>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc>
```

In case the client provides a txid value for a non-versioned node, the server needs to treat the node as having the same txid value as the closest ancestor that does have a txid value.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="7"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <acls
        xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
        <acl>
          <name>A2</name>
          <aces>
            <ace>
              <name>R7</name>
              <matches>
                <ipv4>
                  <dscp txid:etag="nc4711"/>
                </ipv4>
              </matches>
            </ace>
          </aces>
        </acl>
      </acls>
    </filter>
  </get-config>
</rpc>
```

If a txid value is specified for a leaf, and the txid value matches, the leaf value is pruned.

```
<rpc-reply message-id="7"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
      <acl>
        <name>A2</name>
        <aces>
          <ace>
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp txid:etag="="/>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc-reply>
```

5.3. Configuration Change

A client that wishes to update the ace R1 protocol to tcp might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="8">
  <edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
               xmlns:ietf-netconf-txid=
                "urn:ietf:params:xml:ns:yang:ietf-netconf-txid">
    <target>
      <running/>
    </target>
    <test-option>test-then-set</test-option>
    <ietf-netconf-txid:with-etag>true<ietf-netconf-txid:with-etag>
    <config>
      <acls
        xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
        txid:etag="nc5152">
        <acl txid:etag="nc4711">
          <name>A1</name>
          <aces txid:etag="nc4711">
            <ace txid:etag="nc4711">
              <matches>
                <ipv4>
                  <protocol>tcp</protocol>
                </ipv4>
              </matches>
            </ace>
          </aces>
        </acl>
      </acls>
    </config>
  </edit-config>
</rpc>
```

The server would update the protocol leaf in the running datastore, and return an rpc-reply as follows:

```
<rpc-reply message-id="8"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <ok txid:etag="nc7688"/>
</rpc-reply>
```

A subsequent get-config request for "acls", with txid:etag="?" might then return:

```
<rpc-reply message-id="9"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      txid:etag="nc7688">
      <acl txid:etag="nc7688">
        <name>A1</name>
        <aces txid:etag="nc7688">
          <ace txid:etag="nc7688">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>tcp</protocol>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
      <acl txid:etag="nc6614">
        <name>A2</name>
        <aces txid:etag="nc6614">
          <ace txid:etag="nc4711">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>AF11</dscp>
              </ipv4>
            </matches>
          </ace>
          <ace txid:etag="nc5152">
            <name>R8</name>
            <matches>
              <udp>
                <source-port>
                  <port>22</port>
                </source-port>
              </udp>
            </matches>
          </ace>
          <ace txid:etag="nc6614">
            <name>R9</name>
            <matches>
              <tcp>
                <source-port>
                  <port>830</port>
                </source-port>
              </tcp>
```

```
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc>
```

In case the server at this point received a configuration change from another source, such as a CLI operator, removing ace R8 and R9 in acl A2, a subsequent get-config request for acls, with txid:etag="?" might then return:

```
<rpc-reply message-id="9"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      txid:etag="cli2222">
      <acl txid:etag="nc7688">
        <name>A1</name>
        <aces txid:etag="nc7688">
          <ace txid:etag="nc7688">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>tcp</protocol>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
      <acl txid:etag="cli2222">
        <name>A2</name>
        <aces txid:etag="cli2222">
          <ace txid:etag="nc4711">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>AF11</dscp>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc>
```

5.4. Conditional Configuration Change

If a client wishes to delete acl A1 if and only if its configuration has not been altered since this client last synchronized its configuration with the server, at which point it received the etag "nc7688" for acl A1, regardless of any possible changes to other acls, it might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="10"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0"
     xmlns:ietf-netconf-txid=
       "urn:ietf:params:xml:ns:yang:ietf-netconf-txid">
  <edit-config>
    <target>
      <runnign/>
    </target>
    <test-option>test-then-set</test-option>
    <ietf-netconf-txid:with-etag>true<ietf-netconf-txid:with-etag>
    <config>
      <acls xmlns=
          "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
        <acl nc:operation="delete"
             txid:etag="nc7688">
          <name>A1</name>
        </acl>
      </acls>
    </config>
  </edit-config>
</rpc>
```

If acl A1 now has the etag txid value "nc7688", as expected by the client, the transaction goes through, and the server responds something like:

```
<rpc-reply message-id="10"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <ok txid:etag="nc8008"/>
</rpc-reply>
```

A subsequent get-config request for acls, with txid:etag="?" might then return:

```
<rpc-reply message-id="11"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      txid:etag="nc8008">
      <acl txid:etag="cli2222">
        <name>A2</name>
        <aces txid:etag="cli2222">
          <ace txid:etag="nc4711">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>AF11</dscp>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc>
```

In case acl A1 did not have the expected etag txid value "nc7688", when the server processed this request, it rejects the transaction, and might send:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:acl=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list"
           xmlns:ietf-netconf-txid=
             "urn:ietf:params:xml:ns:yang:ietf-netconf-txid"
           message-id="11">
  <rpc-error>
    <error-type>protocol</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>error</error-severity>
    <error-info>
      <ietf-netconf-txid:txid-value-mismatch-error-info>
        <ietf-netconf-txid:mismatch-path>
          /acl:acls/acl:acl[acl:name="A1"]
        </ietf-netconf-txid:mismatch-path>
        <ietf-netconf-txid:mismatch-etag-value>
          cli6912
        </ietf-netconf-txid:mismatch-etag-value>
      </ietf-netconf-txid:txid-value-mismatch-error-info>
    </error-info>
  </rpc-error>
</rpc-reply>
```

5.5. Reading from the Candidate Datastore

Let's assume that a get-config towards the running datastore currently contains the following data and txid values:

```
<rpc-reply message-id="12"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      txid:etag="nc4711">
      <acl txid:etag="nc4711">
        <name>A1</name>
        <aces txid:etag="nc4711">
          <ace txid:etag="nc4711">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>udp</protocol>
              </ipv4>
            </matches>
          </ace>
          <ace txid:etag="nc2219">
            <name>R2</name>
            <matches>
              <ipv4>
                <dscp>21</dscp>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc-reply>
```

A client issues discard-changes (to make the candidate datastore equal to the running datastore), and issues an edit-config to change the R1 protocol from udp to tcp, and then executes a get-config with the txid-request attribute "?" set on the acl A1, the server might respond:

```
<rpc-reply message-id="13"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
      <acl txid:etag="!">
        <name>A1</name>
        <aces txid:etag="!">
          <ace txid:etag="!">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>tcp</protocol>
              </ipv4>
            </matches>
          </ace>
          <ace txid:etag="nc2219">
            <name>R2</name>
            <matches>
              <ipv4>
                <dscp>21</dscp>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc-reply>
```

Here, the txid-unknown value "!" is sent by the server. This particular server implementation does not know beforehand which txid value would be used for this versioned node after commit. It will be a value different from the current corresponding txid value in the running datastore.

In case the server is able to predict the txid value that would be used for the versioned node after commit, it could respond with that value instead. Let's say the server knows the txid would be "7688" if the candidate datastore was committed without further changes, then it would respond with that value in each place where the example shows "!" above.

5.6. Commit

The client MAY request that the new etag txid value is returned as an attribute on the ok response for a successful commit. The client requests this by adding with-etag to the commit operation.

For example, a client might send:

```
<rpc message-id="14"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    xmlns:ietf-netconf-txid=
      "urn:ietf:params:xml:ns:yang:ietf-netconf-txid"
  <commit>
    <ietf-netconf-txid:with-etag>true<ietf-netconf-txid:with-etag>
  </commit>
</rpc>
```

Assuming the server accepted the transaction, it might respond:

```
<rpc-reply message-id="15"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <ok txid:etag="nc8008"/>
</rpc-reply>
```

5.7. YANG-Push

A client MAY request that the updates for one or more YANG Push subscriptions are annotated with the txid values. The request might look like this:

```
<netconf:rpc message-id="16"
             xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
      xmlns=
        "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
      xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push"
      xmlns:ietf-netconf-txid-yp=
        "urn:ietf:params:xml:ns:yang:ietf-txid-yang-push">
    <yp:datastore
        xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:running
    </yp:datastore>
    <yp:datastore-xpath-filter
        xmlns:acl=
          "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
      /acl:acls
    </yp:datastore-xpath-filter>
    <yp:periodic>
      <yp:period>500</yp:period>
    </yp:periodic>
    <ietf-netconf-txid-yp:with-etag>
      true
    </ietf-netconf-txid-yp:with-etag>
  </establish-subscription>
</netconf:rpc>
```

In case a client wishes to modify a previous subscription request in order to no longer receive YANG Push subscription updates, the request might look like this:

```
<rpc message-id="17"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
      xmlns=
        "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
      xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push"
      xmlns:ietf-netconf-txid-yp=
        "urn:ietf:params:xml:ns:yang:ietf-txid-yang-push">
    <id>1011</id>
    <yp:datastore
        xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:running
    </yp:datastore>
    <ietf-netconf-txid-yp:with-etag>
      false
    </ietf-netconf-txid-yp:with-etag>
  </modify-subscription>
</rpc>
```

A server might send a subscription update like this:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2022-04-04T06:00:24.16Z</eventTime>
  <push-change-update
      xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <id>89</id>
    <datastore-changes>
      <yang-patch>
        <patch-id>0</patch-id>
        <edit txid:etag="nc8008">
          <edit-id>edit1</edit-id>
          <operation>delete</operation>
          <target xmlns:acl=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
            /acl:acls
          </target>
          <value>
            <acl xmlns=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
              <name>A1</name>
            </acl>
          </value>
        </edit>
      </yang-patch>
    </datastore-changes>
  </push-change-update>
</notification>
```

6. YANG Modules

6.1. Base module for txid in NETCONF

```
<CODE BEGINS>
module ietf-netconf-txid {
  yang-version 1.1;
  namespace
    'urn:ietf:params:xml:ns:yang:ietf-netconf-txid';
  prefix ietf-netconf-txid;

  import ietf-netconf {
    prefix nc;
  }

  import ietf-netconf-nmda {
    prefix ncds;
  }

  import ietf-yang-structure-ext {
    prefix sx;
  }

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:   <http://tools.ietf.org/wg/netconf/>
     WG List:  <netconf@ietf.org>

     Author:   Jan Lindblad
               <mailto:jlindbla@cisco.com>";

  description
    "NETCONF Transaction ID aware operations for NMDA.

     Copyright (c) 2022 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 Simplified 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 2023-03-01 {
  description
    "Initial revision";
  reference
    "RFC XXXX: Xxxxxxxxx";
}

typedef etag-t {
  type string {
    pattern ".* .*" {
      modifier invert-match;
    }
    pattern '.*".*' {
      modifier invert-match;
    }
    pattern ".*\\.*" {
      modifier invert-match;
    }
  }
  description
    "Unique Entity-tag txid value representing a specific
    transaction.  Could be any string that does not contain
    spaces, double quotes or backslash.  The txid values '?',
    '!' and '=' have special meaning.";
}

typedef last-modified-t {
  type union {
    type yang:date-and-time;
    type enumeration {
      enum ? {
        description "Txid value used by clients that is
          guaranteed not to match any txid on the server.";
      }
      enum ! {
        description "Txid value used by servers to indicate
          the node in the candidate datastore has changed
          relative the running datastore, but not yet received
          a new txid value on the server.";
      }
      enum = {
        description "Txid value used by servers to indicate
```

```
          that contents has been pruned due to txid match
          between client and server.";
      }
    }
  }
  description
    "Last-modified txid value representing a specific transaction.
     The txid values '?', '!' and '=' have special meaning.";
}

grouping txid-grouping {
  leaf with-etag {
    type boolean;
    description
      "Indicates whether the client requests the server to include
       a txid:etag txid attribute when the configuration has
       changed.";
  }
  leaf with-last-modified {
    type boolean;
    description
      "Indicates whether the client requests the server to include
       a txid:last-modified attribute when the configuration has
       changed.";
  }
  description
    "Grouping for txid mechanisms, to be augmented into
     rpcs that modify configuration data stores.";
}

augment /nc:edit-config/nc:input {
  uses txid-grouping;
  description
    "Injects the txid mechanisms into the
    edit-config operation";
}

augment /nc:commit/nc:input {
  uses txid-grouping;
  description
    "Injects the txid mechanisms into the
    commit operation";
}

augment /ncds:edit-data/ncds:input {
  uses txid-grouping;
  description
    "Injects the txid mechanisms into the
    edit-data operation";
```

```
  }

  sx:structure txid-value-mismatch-error-info {
    container txid-value-mismatch-error-info {
      description
         "This error is returned by a NETCONF server when a client
          sends a configuration change request, with the additonal
          condition that the server aborts the transaction if the
          server's configuration has changed from what the client
          expects, and the configuration is found not to actually
          not match the client's expectation.";
      leaf mismatch-path {
        type instance-identifier;
        description
          "Indicates the YANG path to the element with a mismatching
           etag txid value.";
      }
      leaf mismatch-etag-value {
        type etag-t;
        description
          "Indicates server's txid value of the etag
          attribute for one mismatching element.";
      }
      leaf mismatch-last-modified-value {
        type last-modified-t;
        description
          "Indicates server's txid value of the last-modified
          attribute for one mismatching element.";
      }
    }
  }
}

<CODE ENDS>
```

6.2. Additional support for txid in YANG-Push

```
<CODE BEGINS>
module ietf-netconf-txid-yang-push {
  yang-version 1.1;
  namespace
    'urn:ietf:params:xml:ns:yang:ietf-netconf-txid-yang-push';
  prefix ietf-netconf-txid-yp;

  import ietf-subscribed-notifications {
    prefix sn;
    reference
      "RFC 8639: Subscription to YANG Notifications";
  }

  import ietf-netconf-txid {
    prefix ietf-netconf-txid;
    reference
      "RFC XXXX: Xxxxxxxxx";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:   <http://tools.ietf.org/wg/netconf/>
     WG List:  <netconf@ietf.org>

     Author:   Jan Lindblad
               <mailto:jlindbla@cisco.com>";

  description
    "NETCONF Transaction ID aware operations for YANG Push.

     Copyright (c) 2022 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 Simplified 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 2022-04-01 {
    description
      "Initial revision";
    reference
      "RFC XXXX: Xxxxxxxxx";
  }

  augment "/sn:establish-subscription/sn:input" {
    description
      "This augmentation adds additional subscription parameters
        that apply specifically to datastore updates to RPC input.";
    uses ietf-netconf-txid:txid-grouping;
  }
  augment "/sn:modify-subscription/sn:input" {
    description
      "This augmentation adds additional subscription parameters
        specific to datastore updates.";
    uses ietf-netconf-txid:txid-grouping;
  }
  augment "/sn:subscriptions/sn:subscription" {
    description
      "This augmentation adds additional subscription parameters
        specific to datastore updates.";
    uses ietf-netconf-txid:txid-grouping;
  }
}

<CODE ENDS>
```

7. Security Considerations

7.1. NACM Access Control

NACM, [RFC8341], access control processing happens as usual, independently of any txid handling, if supported by the server and enabled by the NACM configuration.

It should be pointed out however, that when txid information is added to a reply, it may occasionally be possible for a client to deduce that a configuration change has happened in some part of the configuration to which it has no access rights.

For example, a client may notice that the root node txid has changed while none of the subtrees it has access to have changed, and thereby conclude that someone else has made a change to some part of the configuration that is not acessible by the client.

7.1.1. Hash-based Txid Algorithms

Servers that implement NACM and choose to implement a hash-based txid algorithm over the configuration may reveal to a client that the configuration of a subtree that the client has no access to is the same as it was at an earlier point in time.

For example, a client with partial access to the configuration might observe that the root node txid was 1234. After a few configuration changes by other parties, the client may again observe that the root node txid is 1234. It may then deduce that the configuration is the same as earlier, even in the parts of the configuration it has no access to.

In some use cases, this behavior may be considered a feature, since it allows a security client to verify that the configuration is the same as expected, without transmitting or storing the actual configuration.

7.2. Unchanged Configuration

It will also be possible for clients to deduce that a confiuration change has not happened during some period, by simply observing that the root node (or other subtree) txid remains unchanged. This is true regardless of NACM being deployed or choice of txid algorithm.

Again, there may be use cases where this behavior may be considered a feature, since it allows a security client to verify that the configuration is the same as expected, without transmitting or storing the actual configuration.

8. IANA Considerations

This document registers the following capability identifier URN in the 'Network Configuration Protocol (NETCONF) Capability URNs' registry:

urn:ietf:params:netconf:capability:txid:1.0

This document registers three XML namespace URNs in the 'IETF XML registry', following the format defined in [RFC3688].

URI: urn:ietf:params:xml:ns:netconf:txid:1.0

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-txid

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-txid-yang-push

Registrant Contact: The NETCONF WG of the IETF.

XML: N/A, the requested URIs are XML namespaces.

This document registers two module names in the 'YANG Module Names' registry, defined in [RFC6020].

name: ietf-netconf-txid

prefix: ietf-netconf-txid

namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-txid

RFC: XXXX

and

name: ietf-netconf-txid-yp

prefix: ietf-netconf-txid-yp

namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-txid-yang-push

RFC: XXXX

9. Changes

9.1. Major changes in -03 since -02

* Changed the logic around how txids are handled in the candidate datastore, both when reading (get-config, get-data) and writing (edit-config, edit-data). Introduced a special "txid-unknown" value "!".

*Changed the logic of copy-config to be similar to edit-config.

*Clarified how txid values interact with when-dependencies together with default values.

*Added content to security considerations.

*Added a high-level example for YANG-Push subscriptions with txid.

*Updated language about error-info sent at txid mismatch in an edit-config: error-info with mismatch details MUST be sent when mismatch detected, and that the server can choose one of the txid mismatch occurrences if there is more than one.

*Some rewording and minor additions for clarification, based on mailing list feedback.

*Divided RFC references into normative and informative.

*Corrected a logic error in the second figure (figure 6) in the "Conditional Transactions" section

9.2. Major changes in -02 since -01

*A last-modified txid mechanism has been added (back). This mechanism aligns well with the Last-Modified mechanism defined in RESTCONF [RFC8040], but is not a carbon copy.

*YANG Push functionality has been added. This allows YANG Push users to receive txid updates as part of the configuration updates. This functionality comes in a separate YANG module, to allow implementors to cleanly keep all this functionality out.

*Changed name of "versioned elements". They are now called "versioned nodes".

*Clarified txid behavior for transactions toward the Candidate datastore, and some not so common situations, such as when a client specifies a txid for a non-versioned node, and when there are when-statement dependencies across subtrees.

*Examples provided for the abstract mechanism level with simple message flow diagrams.

*More examples on protocol level, and with ietf-interfaces as example target module replaced with ietf-access-control to reduce confusion.

*Explicit list of XPaths to clearly state where etag or last-modified attributes may be added by clients and servers.

*Document introduction restructured to remove duplication between sections and to allow multiple (etag and last-modified) txid mechanisms.

*Moved the actual YANG module code into proper module files that are included in the source document. These modules can be compiled as proper modules without any extraction tools.

9.3. Major changes in -01 since -00

*Updated the text on numerous points in order to answer questions that appeared on the mailing list.

*Changed the document structure into a general transaction id part and one etag specific part.

*Renamed entag attribute to etag, prefix to txid, namespace to urn:ietf:params:xml:ns:yang:ietf-netconf-txid.

*Set capability string to urn:ietf:params:netconf:capability:txid:1.0

*Changed YANG module name, namespace and prefix to match names above.

*Harmonized/slightly adjusted etag value space with RFC 7232 and RFC 8040.

*Removed all text discussing etag values provided by the client (although this is still an interesting idea, if you ask the author)

*Clarified the etag attribute mechanism, especially when it comes to matching against non-versioned elements, its cascading upwards in the tree and secondary effects from when- and choice-statements.

*Added a mechanism for returning the server assigned etag value in get-config and get-data.

*Added section describing how the NETCONF discard-changes, copy-config, delete-config and commit operations work with respect to etags.

*Added IANA Considerations section.

*Removed all comments about open questions.

10. References

10.1. 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/rfc/rfc2119>.

[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/rfc/rfc6241>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/rfc/rfc6991>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/rfc/rfc7950>.

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

[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/rfc/rfc8174>.

[RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <https://www.rfc-editor.org/rfc/rfc8641>.

10.2. Informative References

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/rfc/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/rfc/rfc6020>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <https://www.rfc-editor.org/rfc/rfc7232>.

[RFC8341]
Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/rfc/rfc8341>.

Acknowledgments

The author wishes to thank Benoit Claise for making this work happen, and the following individuals, who all provided helpful comments: Per Andersson, Kent Watsen, Andy Bierman, Robert Wilton, Qiufang Ma, Jason Sterne and Robert Varga.

Author's Address

Jan Lindblad
Cisco Systems

Email: jlindbla@cisco.com