NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 2 February 2026

J. Lindblad
All For Eco

1 August 2025

Transaction ID Mechanism for NETCONF
draft-ietf-netconf-transaction-id-10

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 inefficient for synchronization. This document specifies a NETCONF extension 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 2 February 2026.

Table of Contents

1. Introduction

When a NETCONF client [RFC6241] 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 that 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 (e.g., local configuration).

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 may 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.

Note that several server implementors have built proprietary and mutually incompatible mechanisms for obtaining a transaction id from a NETCONF server. This document solves the interoperability issue.

RESTCONF, [RFC8040], defines a mechanism for detecting changes in configuration subtrees based on Entity-Tags (ETags) and Last-Modified headers. An example is depicted in Appendix B.2.2 of [RFC8040]

In conjunction with this, RESTCONF provides a way to make configuration changes conditional on the server configuration being untouched by others. This mechanism leverages conditional requests per Section 13 of [RFC9110].

This document defines similar mechanism for NETCONF, [RFC6241], for config true data. It also ties this in with YANG-Push, [RFC8641], and "Comparison of Network Management Datastore Architecture (NMDA) Datastores", [RFC9144]. 'Config false' data (operational data, state, and statistics) is left out of scope from this document.

This document does not change the RESTCONF protocol in any way, and is carefully written to allow implementations to share much of the code between NETCONF and RESTCONF. Note that the NETCONF txid mechanism described in this document uses XML attributes, but the RESTCONF mechanism relies on HTTP Headers instead, and use none of the XML attributes described in this document, nor JSON Metadata (see [RFC7952]).

1.1. How to Read this Document

At the heart of this document, in chapter Txid Mechanisms (Section 4), there are two transaction-id handling mechanisms defined, the "Etag" and "Last-Modified" Transaction-id mechanisms.

The common and general principles for all transaction-id mechanisms are defined in the chapter before that, NETCONF Txid Extension (Section 3). Since the two Transaction-id mechanisms defined in this document have a lot in common, and the future might bring additional such mechanisms, this arrangement keeps the repetition to a minimum. By necessity, this chapter is a bit abstract. The details of how the principles are expressed in a specific Transaction-id mechanism follows in the Txid Mechanisms (Section 4) chapter.

Next after the central chapter with the definitions of the Transaction-id handling mechanisms, there is an extensive chapter with usage examples. This chapter is called Txid Mechanism Examples (Section 5).

Towards the end, there is also a chapter with YANG Modules (Section 6). These are necessary for a correct implementation, but reading them will not provide much for the understanding of this document. The mechanisms defined in this document are largely on the NETCONF protocol level, and most aspects cannot be described by YANG modules.

The examples found throughout this document are referencing acls, aces, dscp and many other related names defined in YANG modules. Interested readers can find definitions of the relevant YANG structures in [RFC8519]. For the purposes of understanding this document, going there is entirely optional.

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], [RFC7952], [RFC8040], [RFC8641], and [RFC9144].

In addition, this document defines the following terms:

C-txid: Client side transaction-id, i.e., a txid value maintained or provided by a NETCONF client.

Etag: One protocol mechanism that conforms to the definitions in the NETCONF Txid Extension (Section 3) section in this document. Also the name of the XML attribute that this mechanism uses in the NETCONF stream, and the message header used in RESTCONF.

Last-Modified: Another protocol mechanism that conforms to the definitions in the NETCONF Txid Extension (Section 3) section in this document. Also the name of the XML attribute that this mechanism uses in the NETCONF stream, and the message header used in RESTCONF.

S-txid: Server side transaction-id, i.e., a txid value maintained or sent by a NETCONF server.

Transaction-id Mechanism: A protocol implementation that fulfills the principles described in the first part, NETCONF Txid Extension (Section 3), of this document. See also Etag and Last-Modified.

Txid: Abbreviation of Transaction-id. A transaction-id is an UTF-8 string of characters. The specific format depends on the protocol mechanism used (e.g. Etag or Last-Modified).

Txid History: Temporally ordered list of txid values used by the server. Allows the server to determine if a given txid occurred more recently than another txid.

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> operations such that clients are able to conditionally retrieve and update the configuration in a NETCONF server.

For servers implementing YANG-Push [RFC8641], an extension for conveying txid updates as part of subscription updates is also defined. A similar extension is also defined for servers implementing "Comparison of NMDA Datastores" [RFC9144].

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 (Section 4.1) and the 'last-modified txid' mechanism (Section 4.2). However, additional txid mechanisms may be defined in the future. Such mechanisms have to adhere to the principles defined in Section 3.2.

This document is divided into a two main parts; the first part discusses the txid mechanism in an abstract, protocol-neutral way. The second part, Txid Mechanisms (Section 4), then adds the protocol layer, and provides concrete encoding examples.

3.1. Sample Use Cases

The common use cases for txid mechanisms are briefly discussed in this section.

Initial configuration retrieval: When a 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 retrieve again (parts of) the server's configuration, it may be interested to leverage the txid metadata 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 metadata that 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 metadata 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 metadata 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 updated txid metadata for the changed data trees, and recognize the YANG-Push echo of its own changes.

Compare datastores: When a client compares datastores, it may be interested to get the latest txid values of the nodes being compared.

This chapter will also provide some details about how to handle the (or a) candidate datastore, dependencies within a transaction, and txid handling in a few other NETCONF operations (e.g. copy-config).

3.2. General Txid Principles

All servers implementing a txid mechanism MUST maintain a top level server side txid (s-txid) metadata value for each configuration datastore supported by the server. Txid mechanism implementations MAY also maintain txid metadata 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".

Server implementations MAY use the YANG extension statement ietf-netconf-txid:versioned-node to inform potential clients about which YANG nodes the server maintains a txid value for. Another way to discover (a partial) set of Versioned Nodes is for a client to request the current configuration with txids. The returned configuration will then have the Versioned Nodes decorated with their txid values.

Regardless of whether a server declares the Versioned Nodes or not, the set of Versioned Nodes in the server’s YANG tree MUST remain constant, except at system redefining events, such as software upgrades or entitlement (a.k.a. "license") installations or removals. If a Versioned Node was allowed to change status to a non-Versioned Node (or vice versa), the client would no longer be able to reason about the change. The effective txid of some nodes would sometimes seem to change even when no configuration change had taken place.

The server returning txid values for the Versioned Nodes MUST ensure that 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, and their ancestors, 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.

A 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, or any kind of metadata that the server may maintain for YANG data tree nodes.

3.3. Initial Configuration Retrieval

When a NETCONF server receives a <get-config> or <get-data> request (Section 3.1.1 of [RFC8526]) containing requests for txid values, and assuming no authorization or validation error is encountered, it MUST, in the reply, return txid values for all Versioned Nodes below the point requested by the client.

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.

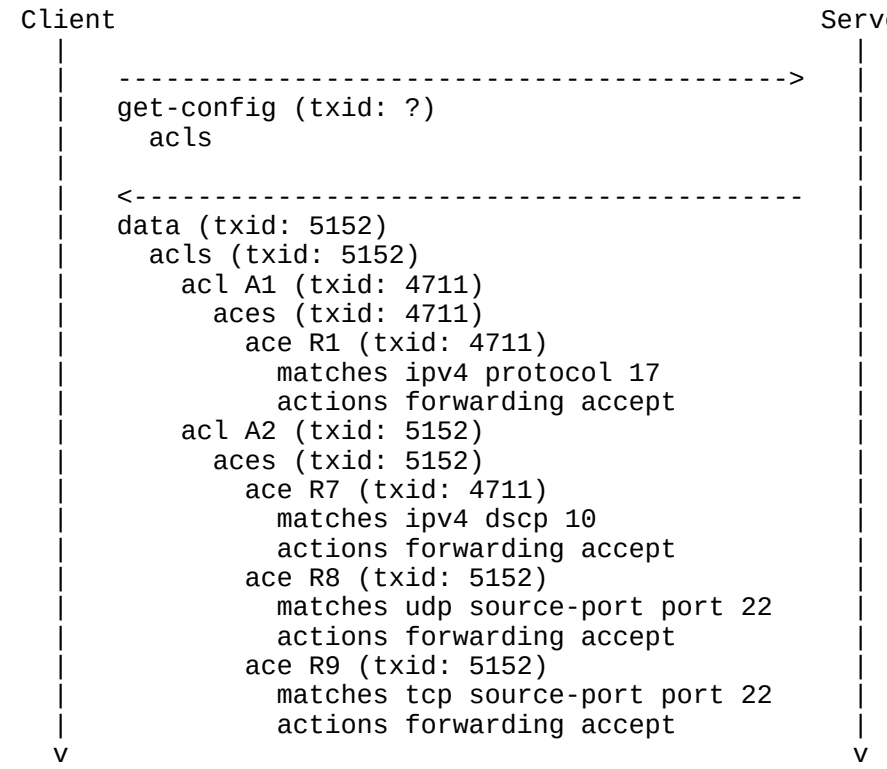


Figure 1: Initial Configuration Retrieval. The client annotated the get-config request itself with the txid request value, which makes the server return all txid values in the entire datastore, that also fall within the requested subtree filter. The most recent change seems to have been an update to ace R8 and R9.

The call flow examples in this document use a 4-digit, strictly increasing integer as txid. The same txid value is also used for all changed nodes in a given transaction. These conventions of the examples are convenient and enhances readability of the examples, but do not necessarily reflect a typical implementation.

Txid values are opaque strings that uniquely identify a particular configuration state. Servers are expected to know which txid values it has used in the recent past, and in which order they were assigned to configuration change transactions. This information is known as the server’s Txid History.

How many historical txid values to track is up to each server implementor to decide, and a server MAY decide not to store any historical txid values at all. The more txid values in the server’s

Txid History, the more efficient the client synchronization may be, as described in the coming sections. Servers may expose a configuration parameter to control the history depth. Such control depends on the local server capabilities. Refer to Section 3.6.2 for more considerations about history size.

Some server implementors may decide to use a strictly increasing integer as the txid value or a timestamp. Doing so obviously makes it very easy for the server to determine the sequence of historical transaction ids.

Some server implementors may decide to use a completely different txid value sequence, to the point that the sequence may appear completely random to outside observers.

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. Txid values sent by a client are referred to as c-txid.

When a client sends a c-txid value of a node that matches the server's s-txid value for that Versioned Node, or matches a more recent s-txid value in the server's Txid History, the server prunes (i.e., does not return) that subtree from the response. Since the client already knows the txid for that part of the data tree, or a txid that occurred more recently, it is obviously already up to date with that part of the configuration. Sending it again would be a waste of time and energy.

Table 1 describes in detail how the client side (c-txid) and server side txid (s-txid) values are determined and compared when the server processes each data tree reply node from a get-config or get-data request.

Servers MUST process each of the config true nodes as follows:

Case	Condition	Behavior
1. NO CLIENT TXID	In its request, the client did not specify a c-txid value for the current node, nor any ancestor of this node.	In this case, the server MUST return the current node according to the normal NETCONF specifications. The rules below do not apply to the current node. Any child

			nodes MUST also be evaluated with respect to these rules.
2. CLIENT ANCESTOR TXID	The client did not specify a c-txid value for the current node, but did specify a c-txid value for one or more ancestors of this node.	In this case, the current node MUST inherit the c-txid value of the closest ancestor node in the client's request that has a c-txid value. Processing of the current node continues according to the rules below.	
3. SERVER ANCESTOR TXID	The node is not a Versioned Node, i.e. the server does not maintain a s-txid value for this node.	In this case, the current node MUST, for the purposes of these rules, temporarily inherit the server's s-txid value of the closest ancestor that is a Versioned Node (has a server side s-txid value). The datastore root is always a Versioned Node. Processing of the current node continues according to the rules below.	
4. CLIENT TXID UP TO DATE	The client specified c-txid for the current node value is "up to date", i.e. it matches the server's s-txid value, or matches a s-txid value from the server's Txid History that is more recent than the server's s-txid value for this node.	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.	
5. CLIENT TXID OUT OF DATE	The specified c-txid is "outdated" or "unknown" to the server, i.e. it does not match the server's s-txid value for this node, nor does the client c-txid value	In this case the server MUST return the current node according to the normal NETCONF specifications. If the current node is a Versioned Node, it MUST be	

	match any s-txid value in the server's Txid History that is more recent than the server's s-txid value for this node.	decorated with the s-txid value. Any child nodes MUST also be evaluated with respect to these rules.	

Table 1: The Txid rules for response pruning.

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.

3.4.1. When there is No Change

Here follows a couple of examples of how the rules above are applied. See the example above (Figure 1) for the most recent server configuration state that the client is aware of, before this happens:

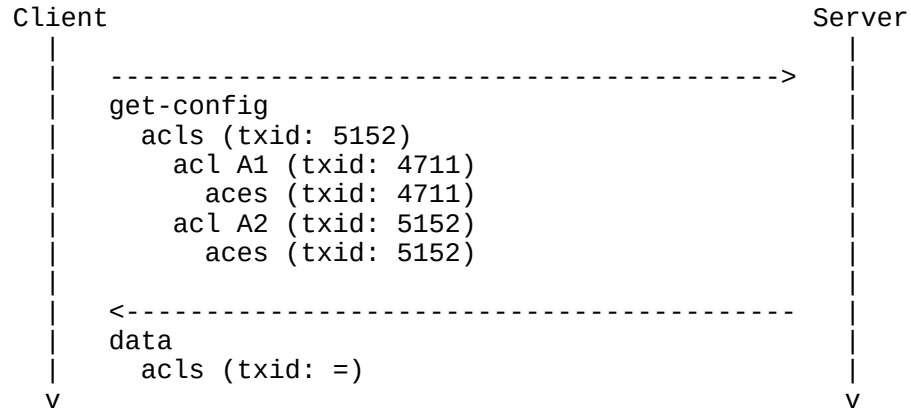


Figure 2: Response Pruning. Client sends get-config request with known txid values. Server prunes response where the c-txid matches expectations. In this case, the server had no changes, and pruned the response at the earliest point offered by the client.

In this case, the server's txid-based pruning saved a substantial amount of information that is already known by the client to be sent to and processed by the client.

3.4.2. When there is an Out-Of-Band (OOB) Change

In the following example someone has made a change to the configuration on the server. This server has chosen to implement a Txid History with up to 5 entries. The 5 most recently used s-txid values on this example server are currently: 4711, 5152, 5550, 6614, 7770 (most recent). Then a client sends this request:

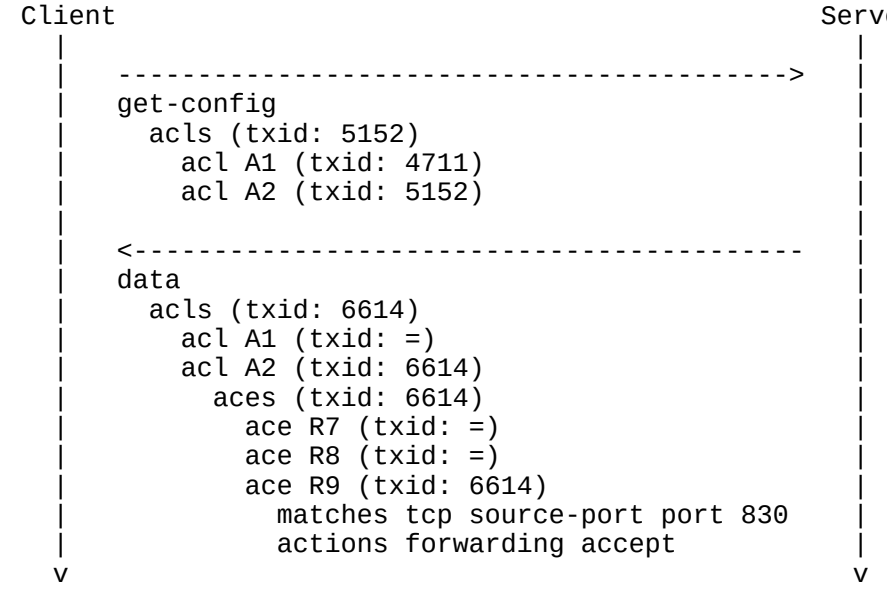


Figure 3: Out of band change detected. Client sends get-config request with known txid values. Server provides updates only where changes have happened. (Txid 7770 does not appear in this subtree, so that transaction must relate to some changes elsewhere.)

In the example depicted in Figure 3, the server returns the acls container because the client supplied c-txid value (5152) differs from the s-txid value held by the server (6614), and 5152 is less recent in the server's Txid History than 6614. The client is apparently unaware of the latest config developments in this part of the server config tree.

The server prunes list entry acl A1 is because it has the same s-txid value as the c-txid supplied by the client (4711). The server returns the list entry acl A2 because 5152 (specified by the client) is less recent than 6614 (held by the server).

The container aces under acl A2 is returned because 5152 is less recent than 6614. The server prunes ace R7 because the c-txid for this node is 5152 (from acl A2), and 5152 is more recent than the closest ancestor Versioned Node (with txid 4711).

The server also prunes acl R8 because the server and client txids exactly match (5152). Finally, acl R9 is returned because of its less recent c-txid value given by the client (5152, on the closest ancestor acl A2) than the s-txid held on the server (6614).

3.4.3. When a Txid value is Inherited from an Ancestor Node

In the example shown in Figure 4, the client specifies the c-txid for a node that the server does not maintain a s-txid for, i.e., it is not a Versioned Node.

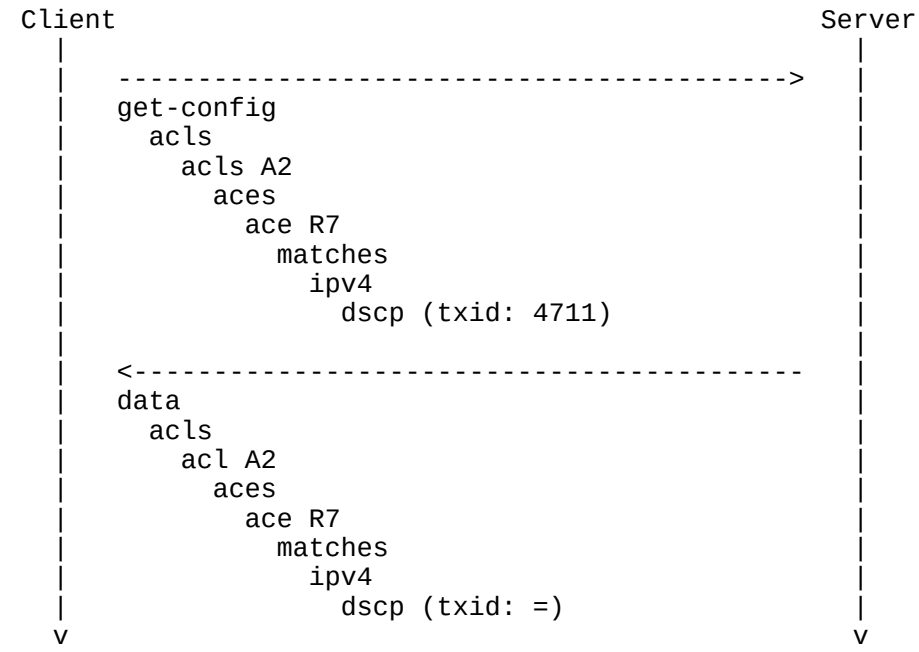


Figure 4: Versioned Nodes. Server lookup of dscp txid gives 4711, as closest ancestor is ace R7 with txid 4711. Since the server's and client's txid match, the txid value is '=', and the leaf value is pruned.

Here, the server looks up the closest ancestor node that is a Versioned Node. This particular server has chosen to keep a s-txid for the list entry ace R7, but not for any of its children. Thus the server finds the server side s-txid value to be 4711 (from ace R7), which matches the client's c-txid value of 4711.

Servers MUST NOT use the special txid values, txid-match, txid-request, txid-unknown (e.g., "=", "?", or "!") as actual txid values.

3.5. Candidate Datastore Configuration Retrieval

When a client retrieves the configuration from the (or a) 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 s-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 candidate.

* If the txid-unknown value is not returned, the server MUST return the s-txid value the node will have after commit, assuming the client makes no further changes of the candidate datastore. If a client makes further changes in the candidate datastore, the s-txid value MAY change again, i.e. the server is not required to stick with the s-txid value just returned.

See the example in Candidate Datastore Transactions (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 c-txid values known to the client in its change requests (<edit-config>, for example), 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 configuration 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>, for example) and write (<edit-config>, for example) operation.

Clients that are also interested to know the s-txid assigned to the root Versioned Node in the model immediately in the response could set a flag in the <rpc> element to request the server to return the new s-txid with the <ok> element.

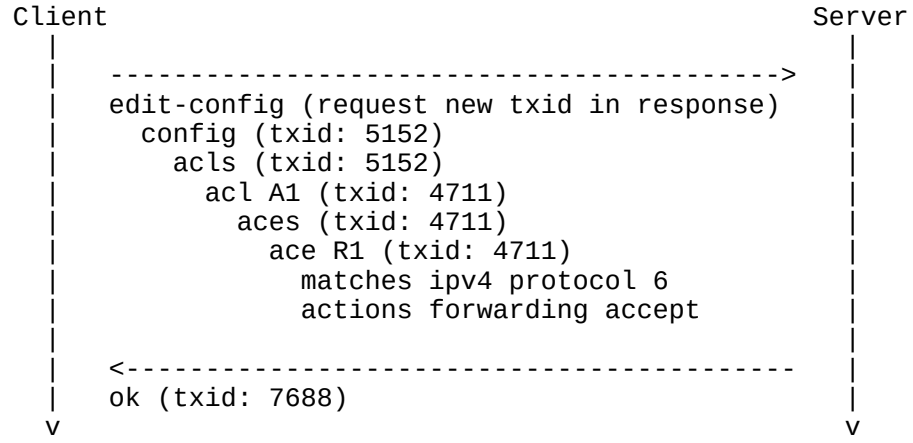


Figure 5: Conditional transaction towards the Running datastore successfully executed. As all the txid values specified by the client matched those on the server, the transaction was successfully executed.

After the above edit-config, the client might issue a get-config to observe the change. It would look like this:

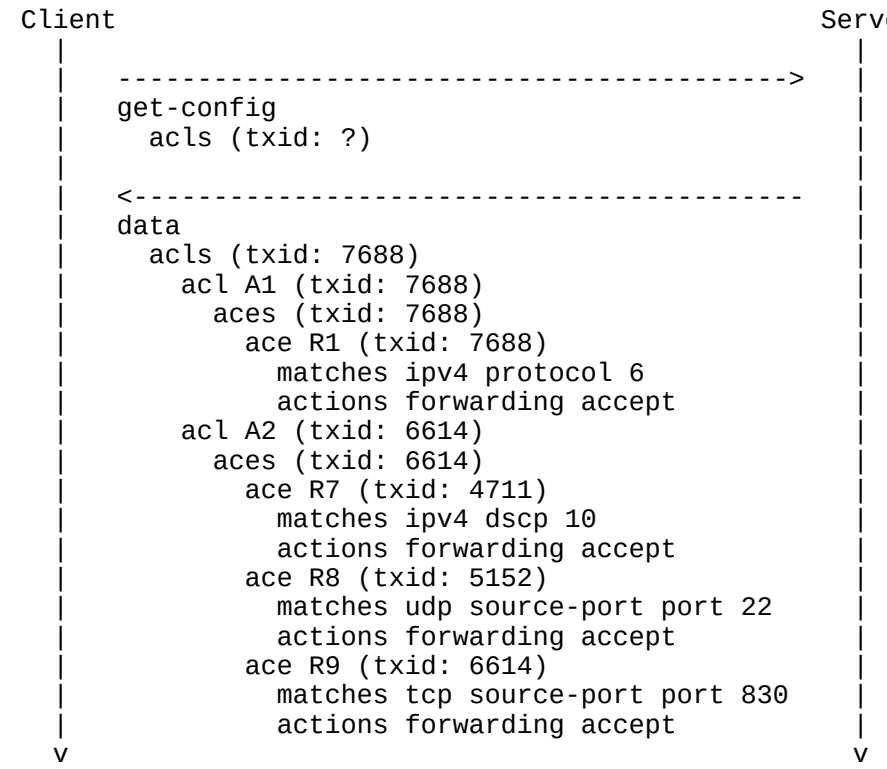


Figure 6: The txids are updated on all Versioned Nodes that were modified themselves or have a child node that was modified.

When a client sends in a c-txid value of a node, the server MUST consider it a match if the server's s-txid value is identical to the client, or if the server's value is found earlier in the server's Txid History than the value supplied by the client.

3.6.1. Error Response on Out-of-Band Changes

If the server rejects the transaction because one or more of the configuration s-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 [RFC8791] containing relevant details about one of the mismatching txids. A server MAY send multiple rpc-errors when multiple txid mismatches are detected.

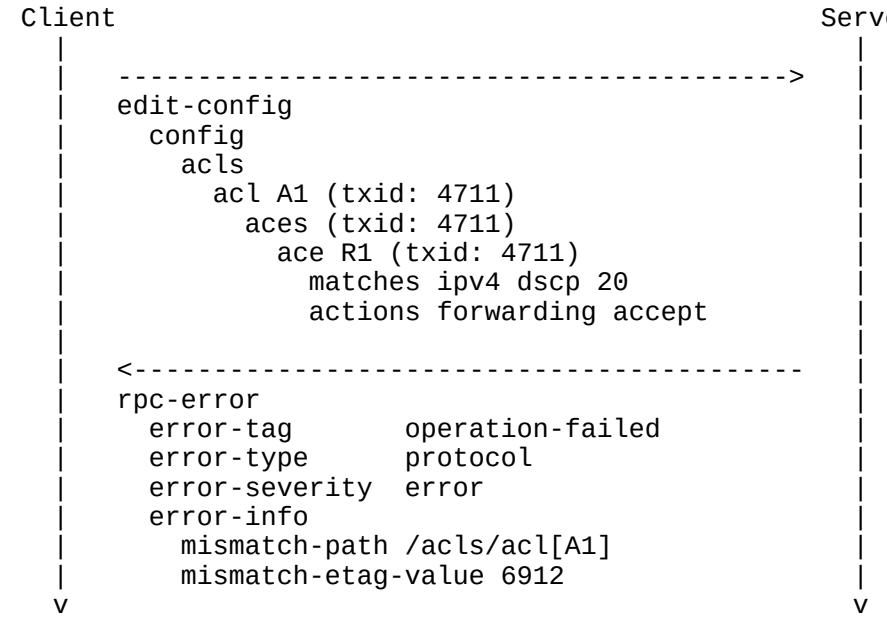


Figure 7: Conditional transaction that fails a txid check. The client wishes to ensure there has been no changes to the particular acl entry it edits, and therefore sends the c-txid it knows for this part of the configuration. Since the s-txid has changed (out of band), the server rejects the configuration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

3.6.2. Txid History Size Consideration

It may be tempting for a client implementor to send a single c-txid value for the tree being edited. In many cases, that would certainly work just fine. This is a way for the client to request the server to go ahead with the change as long as there has not been any changes more recent in the subtree below the c-txid provided.

Here the client is sending the same change as in the example above (Figure 5), but with only a single c-txid value that reflects the latest txid the client is aware of anywhere in the configuration.

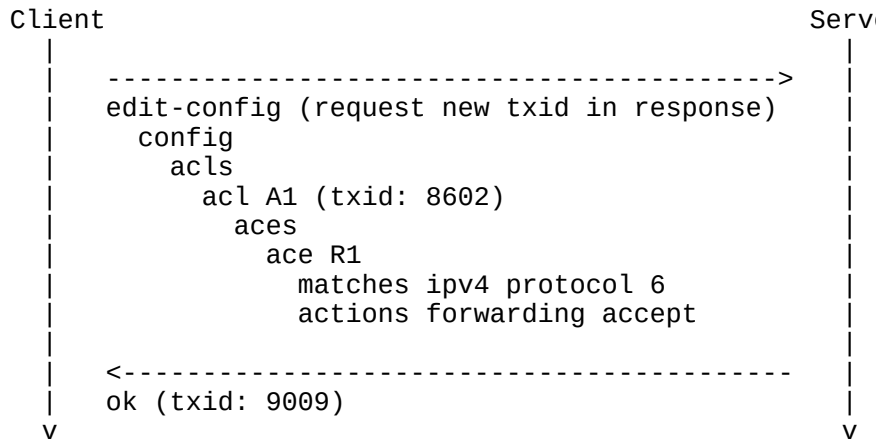


Figure 8: Conditional transaction towards the Running datastore successfully executed. As all the c-txid values specified by the client were the same or more recent in the server's Txid History, so the transaction was successfully executed.

This approach works well in the example above because the c-txid value 8602 is inherited down in the child nodes, from acl A1 to aces, ace R1, and onwards. The server compares the c-txid value 8602 with the s-txid value in the data tree. The server finds that the values do not match (e.g., s-txid 7688 for ace R1 is not equal to c-txid 8602), but finds that 8602 is a more recent txid than 7688 by looking in the server's Txid History, and therefore accepts the transaction.

Clients relying on the server's Txid History being long enough, could see their changes rejected if some of the s-txid have fallen out of the server's Txid History (e.g., if the txid 7688 happened so long ago that the it is no longer in the server's Txid History). Some servers may have a Txid History size of zero. A client specifying a single c-txid value for a change like the one above towards such a server would not be able to get the transaction accepted.

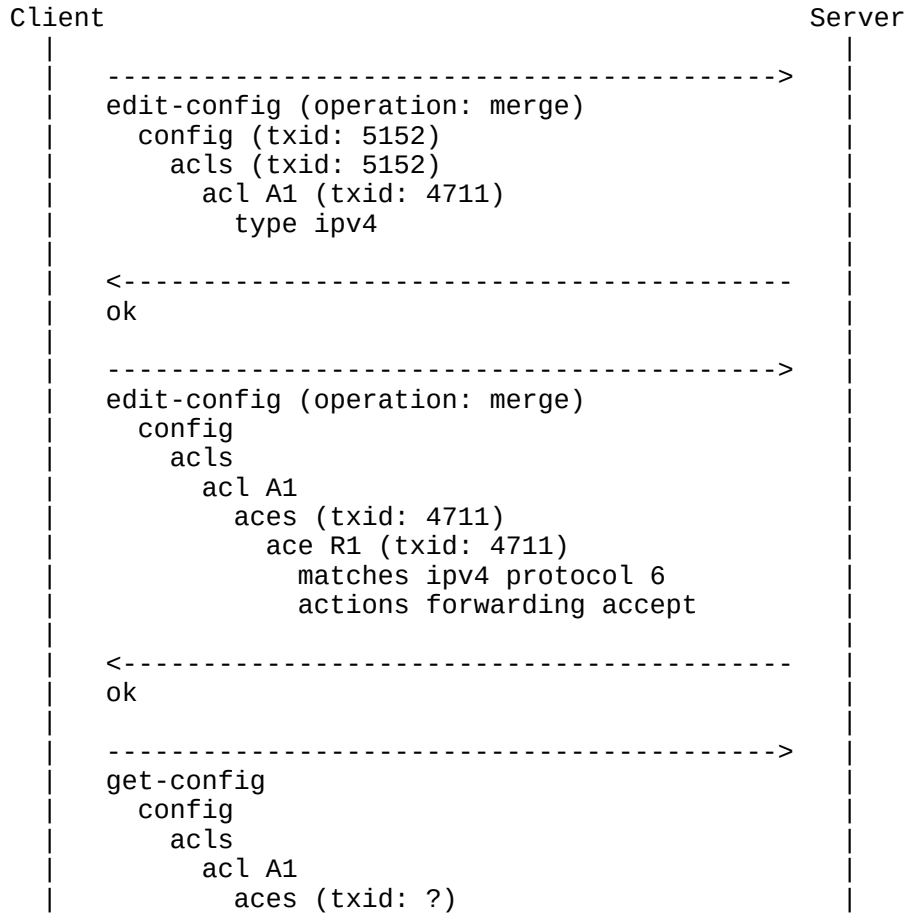
Choosing a Txid History size greater than zero in a server is an optimization allowing clients to be less explicit, saving both on communications and processing. Servers implementing a Txid Mechanism using txid values with a natural order (e.g. strictly increasing integers or timestamps) may be able to implement an infinite history very easily. Other schemes might need to store recently used txids in a database.

It is RECOMMENDED that server implementors that implement Txid History at all choose a Txid History size that is at least large enough to hold twice as many txids as this type of server normally

experiences between the typical connection interval by clients, and not less than 100. In a server near the core of a network, the number of transactions would often be high, but the connection interval by clients typically short. Servers closer to the edge might see fewer transactions, but also be visited by clients less often.

3.7. Candidate Datastore Transactions

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



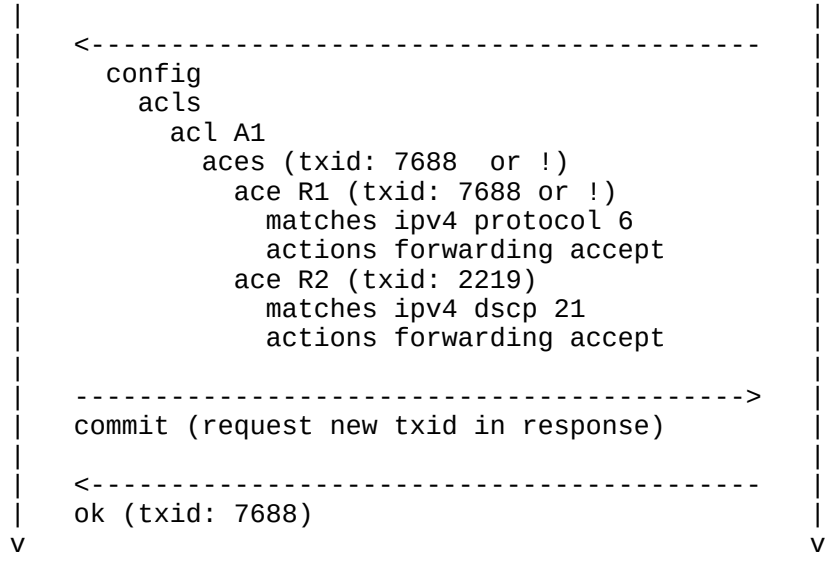


Figure 9: Conditional transaction towards the Candidate datastore successfully executed. As all the c-txid values specified by the client matched those on the server at the time of the commit, the transaction was successfully executed. If a client issues a get-config towards the candidate datastore, the server may choose to return the special txid-unknown value (e.g., "!") or the s-txid value that would be used if the candidate was committed without further changes (when that s-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 s-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:

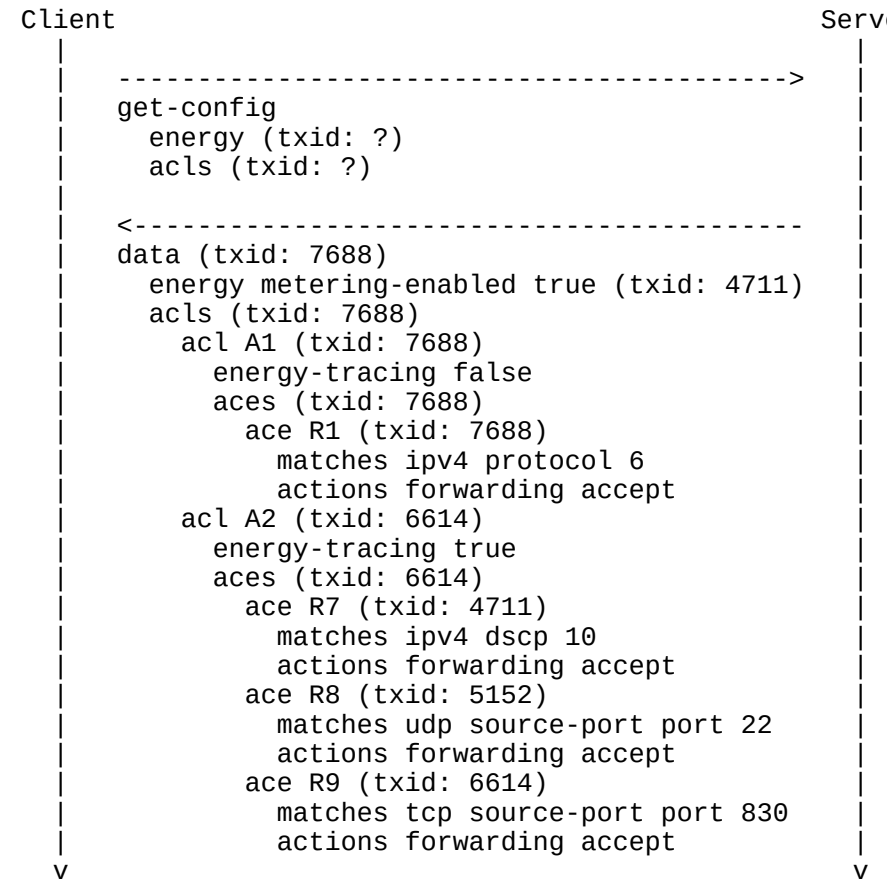


Figure 10: Initial configuration for the energy example. Note the energy metering-enabled leaf at the top and energy-tracing leafs under each acl.

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 s-txid must be updated appropriately.

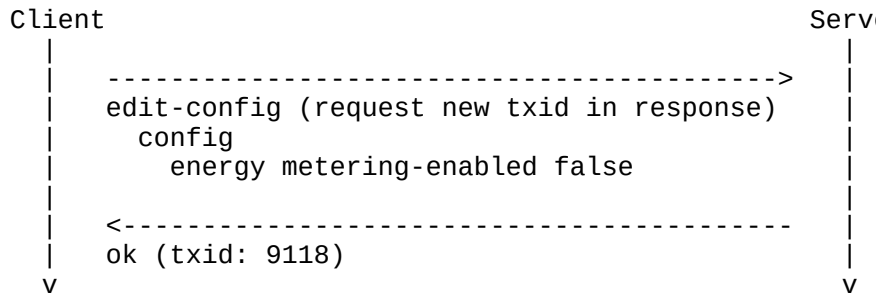


Figure 11: Transaction changing a single leaf. This leaf is the target of a when-statement, however, which means other leafs elsewhere may be indirectly modified by this change. Such indirect changes will also result in s-txid changes.

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.

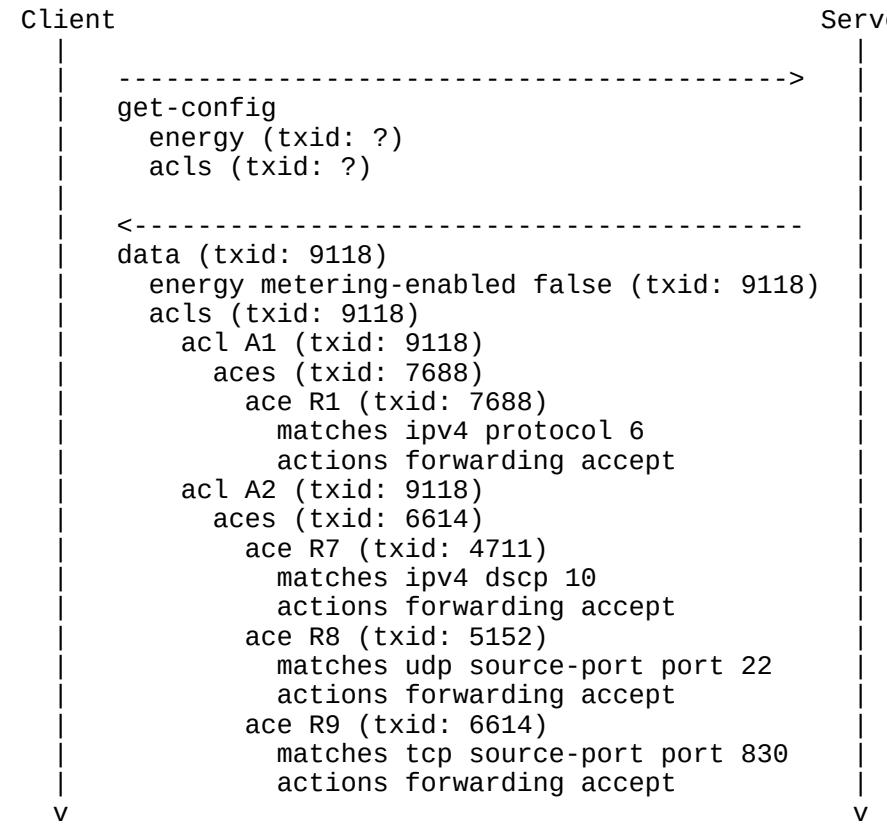


Figure 12: The txid for the energy subtree has changed since that was the target of the edit-config. The txids of the ACLs have also changed since the energy-tracing leafs are now removed by the now false when- expression. Both acl A1 and acl A2 have their txids updated, even though energy-tracing was 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 or configuring a YANG-Push subscription towards a server that supports ietf-netconf-txid-yang-push.yang MAY request that the server provides updated txid values in YANG-Push on-change subscription updates.

This functionality pertains only to on-change updates. This RPC may also be invoked over RESTCONF or other protocols, and might therefore be encoded in JSON.

To request txid values (e.g. etag), the client adds a flag in the request (e.g., with-etag). The server then returns the txid (e.g., etag) value in the yang-patch payload (e.g., as etag-value).

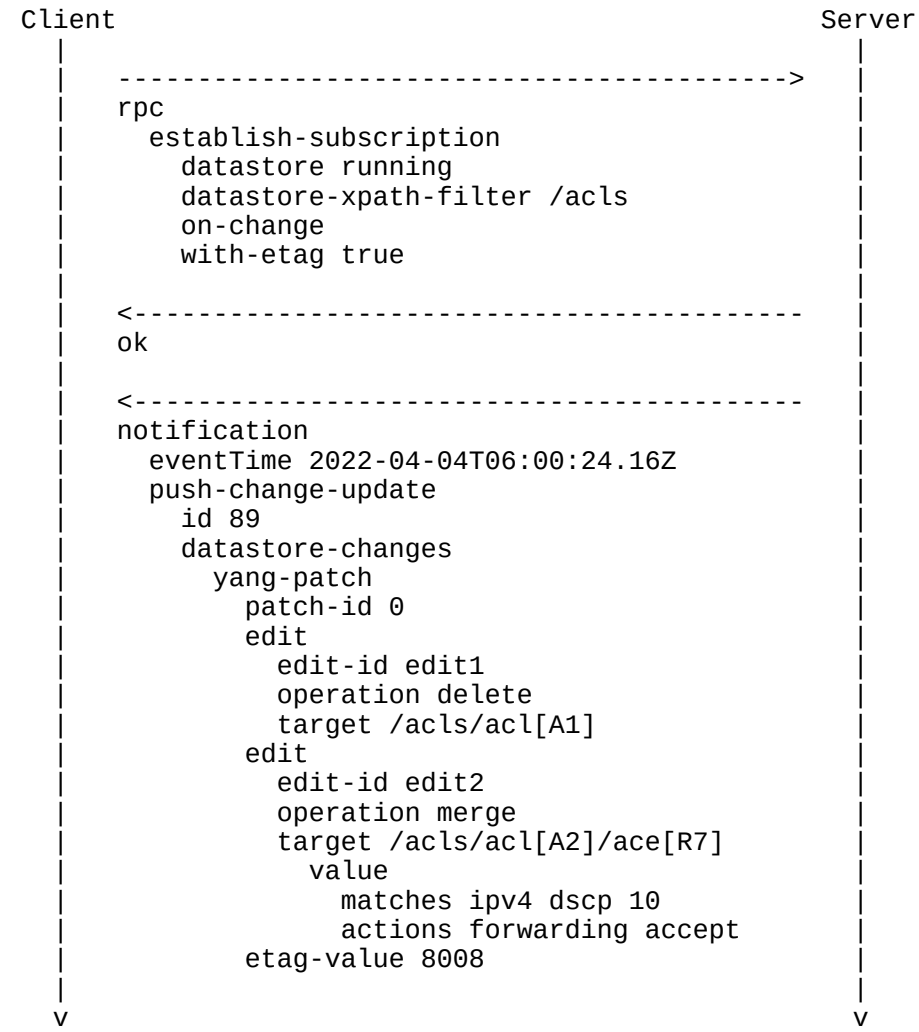


Figure 13: A client requests a YANG-Push subscription for a given path with txid value included. When the server delivers a push-change-update notification, the txid value pertaining to the entire patch is included.

3.11. Comparing YANG Datastores

A client issuing an NMDA Datastore compare request towards a server that supports ietf-netconf-txid-nmda-compare.yang MAY request that the server provides updated txid values in the compare reply. Besides NETCONF, this RPC may also be invoked over RESTCONF or other protocols, and might therefore be encoded in JSON.

To request txid values (e.g. etag), the client adds a flag in the request (e.g. with-etag). The server then returns the txid (e.g. etag) value in the yang-patch payload (e.g. as etag-value).

The txid value returned by the server MUST be the txid value pertaining to the target node in the source or target datastores that is the most recent. If one of the datastores being compared is not a configuration datastore, the txid in the configuration datastore MUST be used. If none of the datastores being compared are a configuration datastore, then txid values MUST NOT be returned at all.

The txid to return is the one that pertains to the target node, or in the case of delete, the closest surviving ancestor of the target node.

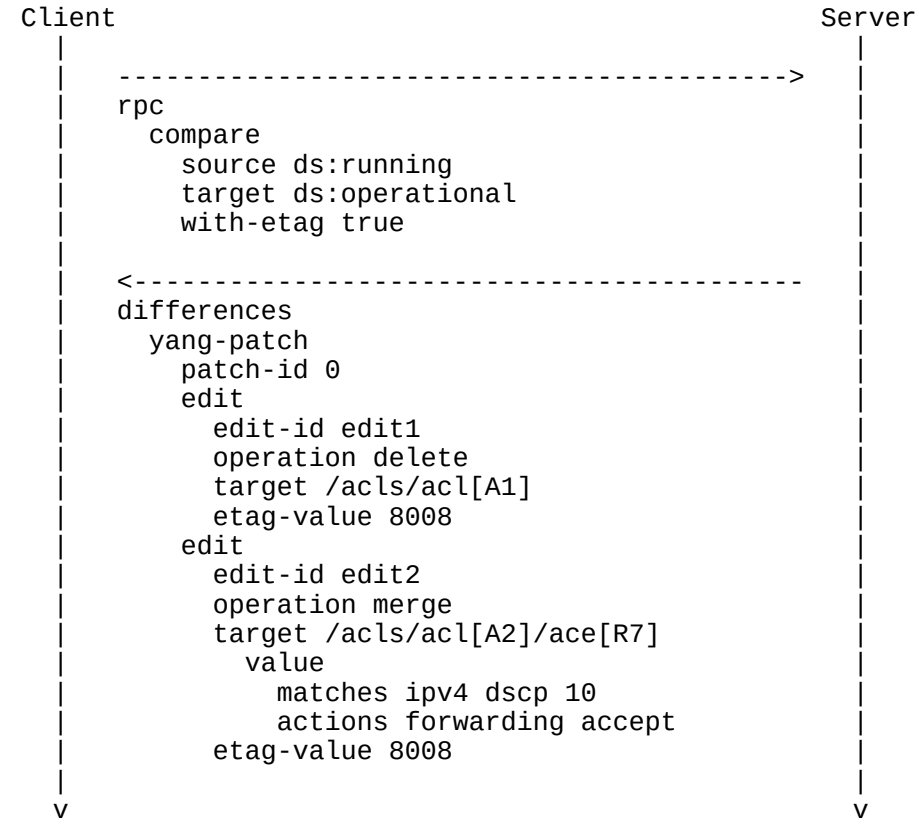


Figure 14: A client requests a NMDA Datastore compare for a given path with txid values included. When the server delivers the reply, the txid is included for each edit.

4. Txid Mechanisms

This document defines two txid mechanisms:

* The etag attribute txid mechanism (Section 4.1)

* The last-modified attribute txid mechanism (Section 4.2)

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 particular 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 strings chosen freely. They MUST consist of ASCII printable characters (VCHAR), except that the etag string MUST NOT contain space, backslash or double quotes. The point of these restrictions is to make it easy to reuse implementations that adhere to section 8.8.3.1 in [RFC9110]. 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. It is RECOMMENDED that the etag txid has an encoding specific suffix, especially when it is not encoded in XML. E.g. a response encoded in JSON might append "+json" at the end of the etag value. This is in line with the language in [RFC9110] and traditions in the HTTP world at large.

The detailed rules for when to update the etag value are described in Section 3.2. These rules are chosen to be consistent with the ETag mechanism in RESTCONF, specifically Sections 3.4.1.2, 3.4.1.3 and 3.5.2 of [RFC8040].

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 feature last-modified defined in ietf-netconf-txid.yang.

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. Servers MUST ensure the timestamps provided are strictly increasing for as long as the server’s operation is maintained.

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 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 subscription 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 supersede 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 candidate 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" [RFC4741] [RFC6241], xmlns:ncds="urn:ietf:params:xml:ns:yang:ietf-netconf-nmda" [RFC8526], xmlns:sn="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications" [RFC8639], xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push" [RFC8641] [RFC8072]:

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/yp:yang-patch/ yp:edit

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

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

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

5. Txid Mechanism Examples

5.1. Initial Configuration Response

5.1.1. With etag

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="?">
    <source>
      <running/>
    </source>
  </get-config>
</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="fd6a52d9-5152-411c-a117-b99d3b723c93">
    <acls xmlns=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list"
          txid:etag="fd6a52d9-5152-411c-a117-b99d3b723c93">
      <acl txid:etag="2c4b50e4-4711-49f8-a2b2-2e20aebe120f">
        <name>A1</name>
        <aces txid:etag="2c4b50e4-4711-49f8-a2b2-2e20aebe120f">
          <ace txid:etag="2c4b50e4-4711-49f8-a2b2-2e20aebe120f">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>17</protocol>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              </forwarding>
            </actions>
          </ace>
        </aces>
      </acl>
      ...
    </acls>
    ...
  </data>
</rpc-reply>
```

It is up to the server implementor to decide on the format of the etag txid value. In the example above, the server used "random" UUIDv4 values [RFC9562]. This is one valid implementation choice.

For the etag txid examples below, we have chosen to use an etag txid value consisting of "nc" (or "cli" in some cases) followed by a strictly increasing integer. This is another valid implementation choice. This format is convenient for the reader trying to make sense of the examples, but is not an implementation requirement.

Clients have to be prepared to receive etag txid values in different formats.

Repeating the example above, but now with a server returning more human readable etag txid values, the server’s reply might 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>17</protocol>
                 </ipv4>
               </matches>
               <actions>
                 <forwarding xmlns:acl=
                 "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                   acl:accept
                 </forwarding>
               </actions>
             </ace>
           </aces>
         </acl>
         <acl txid:etag="nc5152">
           <name>A2</name>
           <aces txid:etag="nc5152">
             <ace txid:etag="nc4711">
               <name>R7</name>
               <matches>
                 <ipv4>
                   <dscp>10</dscp>
                 </ipv4>
               </matches>
               <actions>
                 <forwarding xmlns:acl=
                 "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                   acl:accept
                 </forwarding>
               </actions>
             </ace>
             <ace txid:etag="nc5152">
               <name>R8</name>
               <matches>
                 <udp>
                   <source-port>
```

```
                    <port>22</port>
                  </source-port>
                </udp>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                </forwarding>
              </actions>
            </ace>
            <ace txid:etag="nc5152">
              <name>R9</name>
              <matches>
                <tcp>
                  <source-port>
                    <port>22</port>
                  </source-port>
                </tcp>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                </forwarding>
              </actions>
            </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-reply>
```

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>17</protocol>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              </forwarding>
            </actions>
          </ace>
        </aces>
      </acl>
      <acl txid:etag="nc5152">
        <name>A2</name>
        <aces txid:etag="nc5152">
          <ace txid:etag="nc4711">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>10</dscp>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              </forwarding>
            </actions>
          </ace>
          <ace txid:etag="nc5152">
            <name>R8</name>
            <matches>
              <udp>
                <source-port>
                  <port>22</port>
                </source-port>
              </udp>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              </forwarding>
            </actions>
          </ace>
          <ace txid:etag="nc5152">
```

```
              <name>R9</name>
              <matches>
                <tcp>
                  <source-port>
                    <port>22</port>
                  </source-port>
                </tcp>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                </forwarding>
              </actions>
            </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-reply>
```

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>17</protocol>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              </forwarding>
            </actions>
          </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>10</dscp>
            </ipv4>
          </matches>
          <actions>
            <forwarding xmlns:acl=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
              acl:accept
            </forwarding>
          </actions>
        </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>
          <actions>
            <forwarding xmlns:acl=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
              acl:accept
            </forwarding>
          </actions>
        </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>
          <actions>
            <forwarding xmlns:acl=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
              acl:accept
            </forwarding>
          </actions>
        </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-reply>
```

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>
      </acls>
    </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-reply>
```

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>10</dscp>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              </forwarding>
            </actions>
          </ace>
          <ace txid:etag="nc5152">
            <name>R8</name>
            <matches>
              <ipv4>
                <source-port>
                  <port>22</port>
                </source-port>
              </ipv4>
```

```
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                </forwarding>
              </actions>
            </ace>
            <ace txid:etag="nc6614">
              <name>R9</name>
              <matches>
                <ipv4>
                  <source-port>
                    <port>830</port>
                  </source-port>
                </ipv4>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                </forwarding>
              </actions>
            </ace>
          </aces>
        </acl>
      </acls>
    </data>
  </rpc-reply>
```

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 (i.e. is identical to the server's txid value, or found earlier in the server's Txid History), 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:txid="urn:ietf:params:xml:ns:netconf:txid: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>6</protocol>
                   </ipv4>
                 </matches>
                 <actions>
                   <forwarding xmlns:acl=
                   "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                      acl:accept
                   </forwarding>
                 </actions>
               </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>6</protocol>
                 </ipv4>
               </matches>
               <actions>
                 <forwarding xmlns:acl=
                 "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                   acl:accept
                 </forwarding>
               </actions>
             </ace>
           </aces>
         </acl>
         <acl txid:etag="nc6614">
           <name>A2</name>
           <aces txid:etag="nc6614">
             <ace txid:etag="nc4711">
               <name>R7</name>
               <matches>
                 <ipv4>
                   <dscp>10</dscp>
                 </ipv4>
               </matches>
               <actions>
                 <forwarding xmlns:acl=
                 "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                   acl:accept
                 </forwarding>
               </actions>
             </ace>
             <ace txid:etag="nc5152">
               <name>R8</name>
               <matches>
                 <udp>
                   <source-port>
```

```
                    <port>22</port>
                  </source-port>
                </udp>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                </forwarding>
              </actions>
            </ace>
            <ace txid:etag="nc6614">
              <name>R9</name>
              <matches>
                <tcp>
                  <source-port>
                    <port>830</port>
                  </source-port>
                </tcp>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                </forwarding>
              </actions>
            </ace>
          </aces>
        </acl>
      </acls>
    </data>
  </rpc-reply>
```

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>6</protocol>
                </ipv4>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                </forwarding>
              </actions>
            </ace>
          </aces>
        </acl>
        <acl txid:etag="cli2222">
          <name>A2</name>
          <aces txid:etag="cli2222">
            <ace txid:etag="nc4711">
              <name>R7</name>
              <matches>
                <ipv4>
                  <dscp>10</dscp>
                </ipv4>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                </forwarding>
              </actions>
            </ace>
          </aces>
        </acl>
      </acls>
    </data>
  </rpc-reply>
```

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>
      <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">
        <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>10</dscp>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              </forwarding>
            </actions>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc-reply>
```

In case acl A1 did not have the expected etag txid value "nc7688" when the server processed this request, nor was the client’s txid value found later in the server’s Txid History, then the server 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>17</protocol>
                 </ipv4>
               </matches>
               <actions>
                 <forwarding xmlns:acl=
                 "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                   acl:accept
                 </forwarding>
               </actions>
             </ace>
             <ace txid:etag="nc2219">
               <name>R2</name>
               <matches>
                 <ipv4>
                   <dscp>21</dscp>
                 </ipv4>
               </matches>
               <actions>
                 <forwarding xmlns:acl=
                 "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                   acl:accept
                 </forwarding>
               </actions>
             </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 (17) to tcp (6), 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>6</protocol>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              </forwarding>
            </actions>
          </ace>
          <ace txid:etag="nc2219">
            <name>R2</name>
            <matches>
              <ipv4>
                <dscp>21</dscp>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              </forwarding>
            </actions>
          </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="14"
    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:on-change/>
       <ietf-netconf-txid-yp:with-etag>
         true
       </ietf-netconf-txid-yp:with-etag>
     </establish-subscription>
   </netconf:rpc>
```

A server might send a subscription update like this:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0"
  xmlns:ietf-netconf-txid-yp=
    "urn:ietf:params:xml:ns:yang:ietf-netconf-txid-yang-push">
  <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>
          <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>
        <ietf-netconf-txid-yp:etag-value>
          nc8008
        </ietf-netconf-txid-yp:etag-value>
      </yang-patch>
    </datastore-changes>
  </push-change-update>
</notification>
```

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>
```

5.8. NMDA Compare

The following example is taken from section 5 of [RFC9144]. It compares the difference between the operational and intended datastores for a subtree under "interfaces".

In this version of the example, the client requests that txid values, in this case etag-values, are annotated to the result.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <compare xmlns="urn:ietf:params:xml:ns:yang:ietf-nmda-compare"
      xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores"
      xmlns:ietf-netconf-txid-nmda-compare=
        "urn:ietf:params:xml:ns:yang:ietf-netconf-txid-nmda-compare">
    <source>ds:operational</source>
    <target>ds:intended</target>
    <report-origin/>
    <ietf-netconf-txid-nmda-compare:with-etag>
      true
    </ietf-netconf-txid-nmda-compare:with-etag>
    <xpath-filter
        xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      /if:interfaces
    </xpath-filter>
  </compare>
</rpc>
```

RPC reply when a difference is detected:

```
   <rpc-reply
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
       message-id="101">
     <differences
       xmlns="urn:ietf:params:xml:ns:yang:ietf-nmda-compare"
       xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
       xmlns:ietf-netconf-txid-nmda-compare=
         "urn:ietf:params:xml:ns:yang:ietf-netconf-txid-nmda-compare"
       xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces">
       <yang-patch>
         <patch-id>interface status</patch-id>
         <comment>
           diff between operational (source) and intended (target),
           with txid values taken from intended.
         </comment>
         <edit>
           <edit-id>1</edit-id>
           <operation>replace</operation>
           <target>/ietf-interfaces:interface=eth0/enabled</target>
           <value>
             <if:enabled>false</if:enabled>
           </value>
           <source-value>
             <if:enabled or:origin="or:learned">true</if:enabled>
           </source-value>
           <ietf-netconf-txid-nmda-compare:etag-value>
             4004
           </ietf-netconf-txid-nmda-compare:etag-value>
         </edit>
         <edit>
           <edit-id>2</edit-id>
           <operation>create</operation>
           <target>/ietf-interfaces:interface=eth0/description</target>
           <value>
             <if:description>ip interface</if:description>
           </value>
           <ietf-netconf-txid-nmda-compare:etag-value>
             8008
           </ietf-netconf-txid-nmda-compare:etag-value>
         </edit>
       </yang-patch>
     </differences>
   </rpc-reply>
```

The same response in RESTCONF (using JSON format):

```
   HTTP/1.1 200 OK
   Date: Thu, 24 Jan 2019 20:56:30 GMT
   Server: example-server
   Content-Type: application/yang-data+json

   { "ietf-nmda-compare:output" : {
       "differences" : {
         "ietf-yang-patch:yang-patch" : {
           "patch-id" : "interface status",
           "comment" : "diff between intended (source) and operational",
           "edit" : [
             {
               "edit-id" : "1",
               "operation" : "replace",
               "target" : "/ietf-interfaces:interface=eth0/enabled",
               "value" : {
                 "ietf-interfaces:interface/enabled" : "false"
               },
               "source-value" : {
                 "ietf-interfaces:interface/enabled" : "true",
                 "@ietf-interfaces:interface/enabled" : {
                   "ietf-origin:origin" : "ietf-origin:learned"
                 }
               },
               "ietf-netconf-txid-nmda-compare:etag-value": "4004"
             },
             {
               "edit-id" : "2",
               "operation" : "create",
               "target" : "/ietf-interfaces:interface=eth0/description",
               "value" : {
                 "ietf-interface:interface/description" : "ip interface"
               },
               "ietf-netconf-txid-nmda-compare:etag-value": "8008"
             }
           ]
         }
       }
     }
   }
```

6. YANG Modules

6.1. Base module for txid in NETCONF

```
<CODE BEGINS> file "ietf-netconf-txid@2025-08-01.yang"
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;
    reference
      "RFC 6241: Network Configuration Protocol (NETCONF)";
  }
  import ietf-netconf-nmda {
    prefix ncds;
    reference
      "RFC 8342: Network Management Datastore Architecture (NMDA)";
  }
  import ietf-yang-structure-ext {
    prefix sx;
    reference
      "RFC 8791: YANG Data Structure Extensions.";
  }
  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }

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

     Author:   Jan Lindblad
               <mailto:jan.lindblad+ietf@for.eco>";
  description
    "NETCONF Transaction ID aware operations for NMDA.

     Copyright (c) 2025 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 2025-08-01 {
      description
        "Initial revision";
      reference
        "RFC XXXX: Transaction ID Mechanism for NETCONF";
    }

    feature last-modified {
      description
        "Servers implementing this module MUST support the
         etag txid mechanism.  Servers MAY also support the
         last-modified txid mechanism.  Support is shown by announcing
         this feature.";
    }

    extension versioned-node {
      description
        "This statement is used by servers to declare that a
         the server is maintaining a Txid for the YANG node with this
         statement.  Which YANG nodes are versioned nodes may be useful
         information for clients (especially during development).

         Servers are not required to use this statement to declare
         which nodes are versioned nodes.

         Example of use:

         container interfaces {
         ietf-netconf-txid:versioned-node;
         ...
         }
        ";
    }

    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:

         '?' This txid value is used by clients and is
            guaranteed not to match any txid on the server.

         '!' This txid value used by servers to indicate
            the node in the candidate datastore has changed
            relative to the running datastore, but not yet received
            a new txid value on the server.

         '=' This txid value used by servers to indicate
            that contents has been pruned due to txid match
            between client and server.
        ";
    }

    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 to 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 {
        if-feature "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.";
    }

    grouping txid-value-grouping {
      leaf etag-value {
        type etag-t;
        description
          "Indicates server's txid value for a YANG node.";
      }
      leaf last-modified-value {
        if-feature "last-modified";
        type last-modified-t;
        description
          "Indicates server's txid value for a YANG node.";
      }
      description
        "Grouping for txid mechanisms, to be augmented into
         output of RPCs that return txid metadata for 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 additional
         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 {
        if-feature "last-modified";
        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> file "ietf-netconf-txid-yang-push@2025-08-01.yang"
  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-yang-push {
      prefix yp;
      reference
        "RFC 8641: Subscriptions to YANG Datastores";
    }
    import ietf-netconf-txid {
      prefix ietf-netconf-txid;
      reference
        "RFC XXXX: Transaction ID Mechanism for NETCONF";
    }

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

       Author:   Jan Lindblad
                 <mailto:jan.lindblad+ietf@for.eco>";
    description
      "NETCONF Transaction ID aware operations for YANG Push.

       Copyright (c) 2025 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 2025-08-01 {
      description
        "Initial revision";
      reference
        "RFC XXXX: Transaction ID Mechanism for NETCONF";
    }

    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;
    }

    augment "/yp:push-change-update/yp:datastore-changes/"
          + "yp:yang-patch" {
      description
```

```
        "This augmentation makes it possible for servers to return
         txid-values.";
      uses ietf-netconf-txid:txid-value-grouping;
    }
  }
  <CODE ENDS>
```

6.3. Additional support for txid in NMDA Compare

```
  <CODE BEGINS> file "ietf-netconf-txid-nmda-compare@2025-08-01.yang"
  module ietf-netconf-txid-nmda-compare {
    yang-version 1.1;
    namespace
      "urn:ietf:params:xml:ns:yang:ietf-netconf-txid-nmda-compare";
    prefix ietf-netconf-txid-nmda-compare;

    import ietf-nmda-compare {
      prefix cmp;
      reference
        "RFC 9144: Comparison of Network Management Datastore
         Architecture (NMDA) Datastores";
    }
    import ietf-netconf-txid {
      prefix ietf-netconf-txid;
      reference
        "RFC XXXX: Transaction ID Mechanism for NETCONF";
    }

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

       Author:   Jan Lindblad
                 <mailto:jan.lindblad+ietf@for.eco>";
    description
      "NETCONF Transaction ID aware operations for NMDA Compare.

       Copyright (c) 2025 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 2025-08-01 {
      description
        "Initial revision";
      reference
        "RFC XXXX: Transaction ID Mechanism for NETCONF";
    }

    augment "/cmp:compare/cmp:input" {
      description
        "This augmentation makes it possible for clients to request
         txids to be returned.";
      uses ietf-netconf-txid:txid-grouping;
    }

    augment "/cmp:compare/cmp:output/cmp:compare-response/"
          + "cmp:differences/cmp:differences/cmp:yang-patch/cmp:edit" {
      description
        "This augmentation makes it possible for servers to return
         txid-values.";
      container most-recent {
        description
          "The txid value returned by the server MUST be the
           txid value pertaining to the target node in the source or
           target datastores that is the most recent.";
        uses ietf-netconf-txid:txid-value-grouping;
      }
    }
  }
  <CODE ENDS>
```

7. Security Considerations

The YANG modules specified in this document define YANG types, groupings, structures and additional RPC parameters for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

In the YANG modules published with this document, there is no configuration, state data, new RPCs, or notifications. This document defines additional XML attributes and headers, however, that merit consideration from a security perspective.

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 accessible 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 configuration 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

8.1. NETCONF Capability URN

This document requests IANA to register the following capability identifier URN in the ’Network Configuration Protocol (NETCONF) Capability URNs’ registry:

RFC Ed.: replace XXXX with actual RFC number and remove this note.

Capability: :txid
Capability Identifier: urn:ietf:params:netconf:capability:txid:1.0
Reference: RFC XXXX

8.2. IETF XML Registry

This document request IANA to register four XML namespace URIs in the "ns" subregistry within the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:netconf:txid:1.0
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-txid
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-txid-yang-push
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-txid-nmda-compare
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

8.3. YANG Module Names

This document requests IANA to register three module names in the "YANG Module Names" subregistry [RFC6020] within the "YANG Parameters" registry.

RFC Ed.: replace XXXX with actual RFC number in this document as well as in the following YANG modules, and remove this note: ietf-netconf-txid-nmda-compare.yang: "RFC XXXX: Transaction ID Mechanism for NETCONF"; ietf-netconf-txid-nmda-compare.yang: This version of this YANG module is part of RFC XXXX ietf-netconf-txid-nmda-compare.yang: (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself ietf-netconf-txid-nmda-compare.yang: "RFC XXXX: Transaction ID Mechanism for NETCONF"; ietf-netconf-txid-yang-push.yang: "RFC XXXX: Transaction ID Mechanism for NETCONF"; ietf-netconf-txid-yang-push.yang: This version of this YANG module is part of RFC XXXX ietf-netconf-txid-yang-push.yang: (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself ietf-netconf-txid-yang-push.yang: "RFC XXXX: Transaction ID Mechanism for NETCONF"; ietf-netconf-txid.yang: This version of this YANG module is part of RFC XXXX ietf-netconf-txid.yang: (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself ietf-netconf-txid.yang: "RFC XXXX: Transaction ID Mechanism for NETCONF";

```
name: ietf-netconf-txid
prefix: ietf-netconf-txid
namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-txid
maintained by IANA? N
RFC: XXXX

name: ietf-netconf-txid-yang-push
prefix: ietf-netconf-txid-yp
namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-txid-yang-push
maintained by IANA? N
RFC: XXXX

name: ietf-netconf-txid-nmda-compare
prefix: ietf-netconf-txid-nmda-compare
namespace:
  urn:ietf:params:xml:ns:yang:ietf-netconf-txid-nmda-compare
maintained by IANA? N
RFC: XXXX
```

9. Changes (to be deleted by RFC Editor)

9.1. Major changes in -10 since -09

Changes based on shepherd review.

* Updated module revision dates, copyright year and author's email address.

9.2. Major changes in -09 since -08

Changes based on shepherd review.

* Updated references to RFC 7232 (Etag, etc.) to the corresponding sections of RFC 9110, which obsoletes RFC 7232. Added a normative reference to RFC 9562 (UUIDs, etc.).

* Made sure all YANG imports have references to the corresponding RFC document. Updated the copyright year to 2025 in a few modules.

* Broke out NETCONF example messages into separate files, and added build logic to perform XML validation on them. Corrected a number of example message errors.

* Changed some document IETF metadata values (abbrev, area, author's affiliation).

* Corrected IANA registration parameters for ietf-netconf-txid-yang-push

* Spelling and formatting updates.

9.3. Major changes in -08 since -07

* Added brief motivation to why the server’s set of Versioned Nodes must not change unless the server at a discontinuity point (software upgrades, etc.)

* Added a few lines to the beginning of chapter 3 to better describe the contents later in that chapter.

* Added some guidance regarding the recommended Txid History size.

* Mention that examples are based on the RFC8519 YANG module for Network Access Control Lists.

9.4. Major changes in -07 since -06

* Changed "monotonically increasing" to "strictly increasing" in multiple locations. Removed recommendation about timestamps in the last-modified txid mechanism being similar to wall clock time.

* Removed two clumsily formulated sentences stating that clients MUST NOT infer temporal order from txid values. The remaining wording states that some servers use sequences of txid values that may appear random to outside observers.

* Added brief explanation that entitlements are sometimes also known as "licenses".

* Added introductory section on "How to Read this Document"

* Added an example to highlight that the etag txid values can have different formats, and do not need to consist of strictly increasing integers, as in most of the examples.

* Changed WG URLs in YANG modules to new datatracker format, e.g. https://datatracker.ietf.org/wg/netconf/

9.5. Major changes in -06 since -05

* Many language, style, spelling and formatting improvements thanks to reviews by Reshad Rahman and Med Boucadair

* Clarified Txid History Size Consideration example

9.6. Major changes in -05 since -04

* Corrected namespace for reference to elements in ietf-yang-push

9.7. Major changes in -04 since -03

* Updated security considerations.

* Added several normative RFC references.

9.8. Major changes in -03 since -02

* Updated language slightly regarding format of etag values, and some recommendations for implementors that support etags in multiple management protocols (NETCONF, RESTCONF, ...) and encodings (XML, JSON, ...).

* Added missing normative RFC references.

* Corrected the YANG-push namespace reference.

9.9. Major changes in -02 since -01

* Added optional to implement Txid History concept in order to make the algorithm both more efficient and less verbose. Servers may still choose a Txid History size of zero, which makes the server behavior the same as in earlier versions of this document. Implementations that use txids consisting of a monotonically increasing integer or timestamp will be able to determine the sequence of transactions in the history directly, making this trivially simple to implement.

* Added extension statement versioned-node, which servers may use to declare which YANG tree nodes are Versioned Nodes. This is entirely optional, however, but possibly useful to client developers.

* Renamed YANG feature ietf-netconf-txid:txid-last-modified to ietf-netconf-txid:last-modified in order to reduce redundant mentions of "txid".

9.10. Major changes in -01 since -00

* Changed YANG-push txid mechanism to use a simple leaf rather than an attribute to convey txid information. This is preferable since YANG-push content may be requested using other protocols than NETCONF and other encodings than XML. By removing the need for XML attributes in this context, the mechanism becomes significantly more portable.

* Added a section and YANG module augmenting the RFC9144 NMDA datastore compare operation to allow request and reply with txid information. This too is done with augments of plain leafs for maximum portability.

* Added note clarifying that the txid attributes used in the XML encoding are never used in JSON (since RESTCONF uses HTTP headers instead).

* Added note clarifying that pruning happens when client and server txids _match_, since the server sending information to the client only makes sense when the information on the client is out of date.

* Added note clarifying that this entire document is about config true data only.

* Rephrased slightly when referring to the candidate datastore to keep making sense in the event that private candidate datastores become a reality in the future.

* Added a note early on to more clearly lay out the structure of this document, with a first part about the generic mechanism part, and a second part about the two specific txid mechanisms.

* Corrected acl data model examples to conform to their YANG module.

9.11. Major changes in draft-ietf-netconf-transaction-id-00 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

9.12. 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.

9.13. 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>.

[RFC4741] Enns, R., Ed., "NETCONF Configuration Protocol", RFC 4741, DOI 10.17487/RFC4741, December 2006, <https://www.rfc-editor.org/rfc/rfc4741>.

[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>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/rfc/rfc6242>.

[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>.

[RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <https://www.rfc-editor.org/rfc/rfc8072>.

[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>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[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/rfc/rfc8526>.

[RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <https://www.rfc-editor.org/rfc/rfc8639>.

[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>.

[RFC8791] Bierman, A., Bj旦rklund, M., and K. Watsen, "YANG Data Structure Extensions", RFC 8791, DOI 10.17487/RFC8791, June 2020, <https://www.rfc-editor.org/rfc/rfc8791>.

[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/rfc/rfc9144>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/rfc/rfc9562>.

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>.

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <https://www.rfc-editor.org/rfc/rfc7952>.

[RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <https://www.rfc-editor.org/rfc/rfc8519>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

Acknowledgments

The author wishes to thank Benoît Claise for making this work happen, and the following individuals, who all provided helpful comments and reviews: Per Andersson, James Cumming, Kent Watsen, Andy Bierman, Robert Wilton, Qiufang Ma, Jason Sterne, Robert Varga, Reshad Rahman, Med Boucadair and Bing Liu.

Author’s Address

Jan Lindblad
All For Eco
Email: jan.lindblad+ietf@for.eco