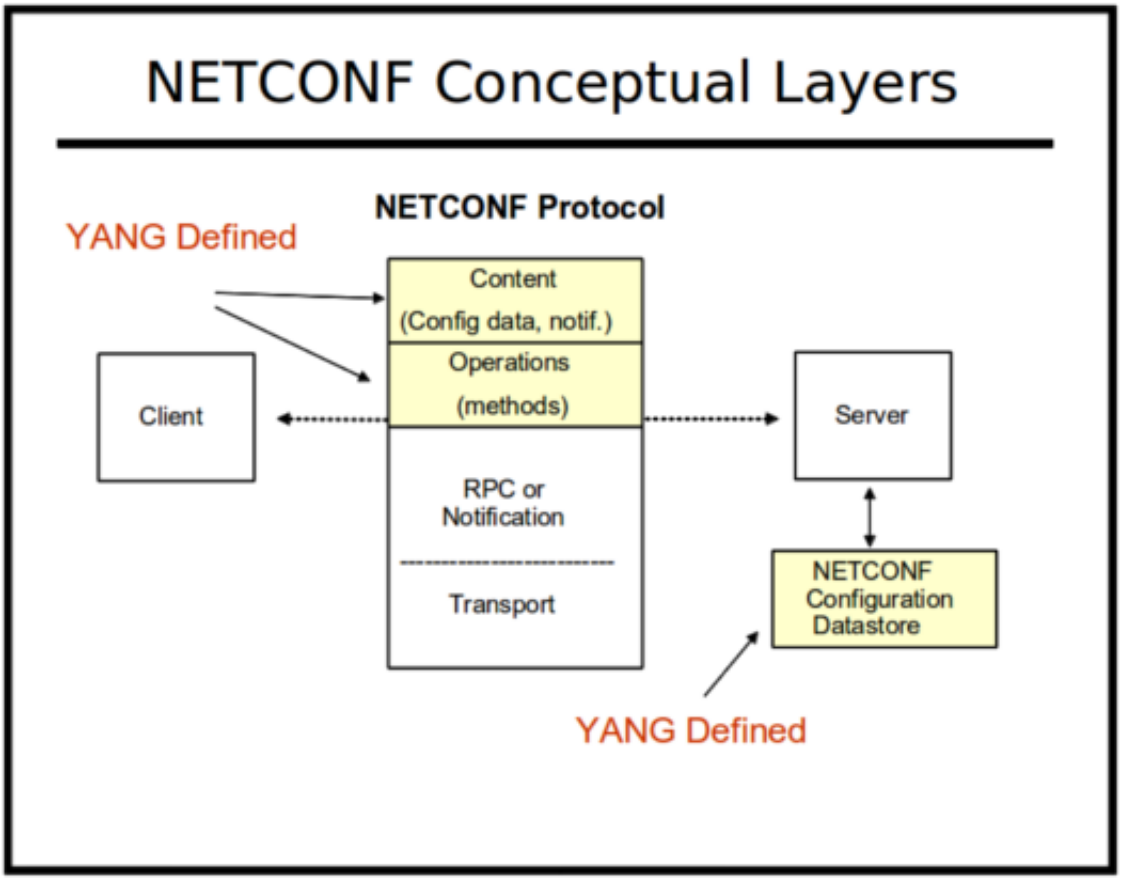
netconf central Show Module Report

Network Configuration Protocol



Summary

[Note: This is not official documentation for the NETCONF protocol. Refer to the actual RFC specifications for authoritative documentation.]

The Network Configuration Protocol (NETCONF) provides operators and application developers with a standard framework and a set of standard Remote Procedure Call (RPC) methods to manipulate the configuration of a network device.

It is designed to be a replacement for Command Line Interface (CLI) based programmatic interfaces, such as Perl + Expect over Secure Shell (SSH). The CLI is also used by humans, which increases the complexity and reduces the predictability of the API for real application usage.

NETCONF is usually transported over the SSH protocol, using the 'netconf' sub-system (similar to the 'sftp' sub-system). and in many ways it mimics the native proprietary CLI over SSH interface available in the device. However, it uses structured schema-driven data, and provides detailed structured error return information, which the CLI cannot provide.

The device configuration data, and the protocol itself, are encoded with the Extensible Markup Language (XML). Standard XML tools such as XML Path Language (XPath) are used to provide retrieval of a particular subset configuration data. All NETCONF messages are encoded in XML within XML

Namespaces. The protocol messages usually contain user data in a different namespace than the NETCONF protocol PDUs.

All NETCONF devices must allow the configuration data to be locked, edited, saved, and unlocked. In addition, all modifications to the configuration data must be saved across a reboot in non-volatile storage.

The protocol (and sometimes even the configuration data) is conceptually partitioned, based on a 'capability'. These capabilities are given unique identifiers and advertised by the server when the client starts a NETCONF session.

A capability can be thought of as an 'API contract' between the server and the client. It represents a set of functionality that cannot be diminished by other capabilities. They can be nested (i.e., one capability required in order to support another) but they are always additive.

There are a core set of operations that must always be supported by the server. To use any additional optional operations, the client should make sure that the server supports the capability associated with that operation.

Base Protocol

The **Network Configuration Protocol** is defined in RFC 4741, and was published as a Proposed Standard in December 2006, by the NETCONF Working Group within the Internet Engineering Task Force (IETF).

NETCONF is a session-based network management protocol, which uses XML-encoded remote procedure calls (RPCs) and configuration data to manage network devices.

The mandatory transport protocol for NETCONF is The Secure Shell Transport Layer Protocol (SSH), and the details for implementing this transport mapping are defined in RFC 4742. The default TCP port assigned for this mapping is 830. A NETCONF server implementation must listen for connections to the 'netconf' subsystem on this port.

A server may optionally support additional transport mappings. RFC 4743 defines mappings to the Simple Object Access Protocol (SOAP). Two different transport protocols are supported for SOAP. The Blocks Extensible Exchange Protocol (BEEP) mapping, called 'SOAP over BEEP', is defined in RFC 4227. The default TCP port for this mapping is 833. The Hypertext Transfer Protocol (HTTP) mapping is defined by BEEP. NETCONF servers must provide secure HTTP (HTTPS), by running HTTP over the Transport Layer Security Protocol (TLS). . The default TCP port for this mapping is 832.

The NETCONF protocol can also be run directly over BEEP. This mapping is defined in RFC 4744. The default TCP port for this mapping is 831.

The IANA port assignments for NETCONF can be summarized in **/etc/services** format as follows:

```
netconf-ssh     830/tcp    NETCONF-over-SSH              # [RFC4742]
netconf-beep    831/tcp    NETCONF-over-BEEP             # [RFC4744]
netconfsoaphttp 832/tcp    NETCONF-for-SOAP-over-HTTPS   # [RFC4743]
netconfsoapbeep 833/tcp    NETCONF-for-SOAP-over-BEEP    # [RFC4743]
netconf-tls     6513/tcp   NETCONF-over-TLS              # [RFC5539]
```

The XSD in RFC 4741 defining the NETCONF protocol has been converted to a YANG module, called ietf-netconf.yang.

NETCONF Sessions

All NETCONF operations are carried out within a session, which is tied to the transport layer connection. There is no standard security model for NETCONF yet, but it is assumed that a session represents a particular user with some set of access rights (assigned by an administrator). The NETCONF server is required to authenticate the entity requesting a session before processing any requests from the client.

NETCONF messages are encoded in XML, using the UTF-8 character set. For SSH, a special message termination sequence of 6 characters is used to provide message framing:

```
]]>]]>
```

Session Initiation For Clients

The client must initiate the connection and session establishment in NETCONF. The mandatory (and most implemented) transport is SSH, so a client must open an SSH2 connection to the **netconf sub-system** to reach the NETCONF server, as shown in the following command line example:

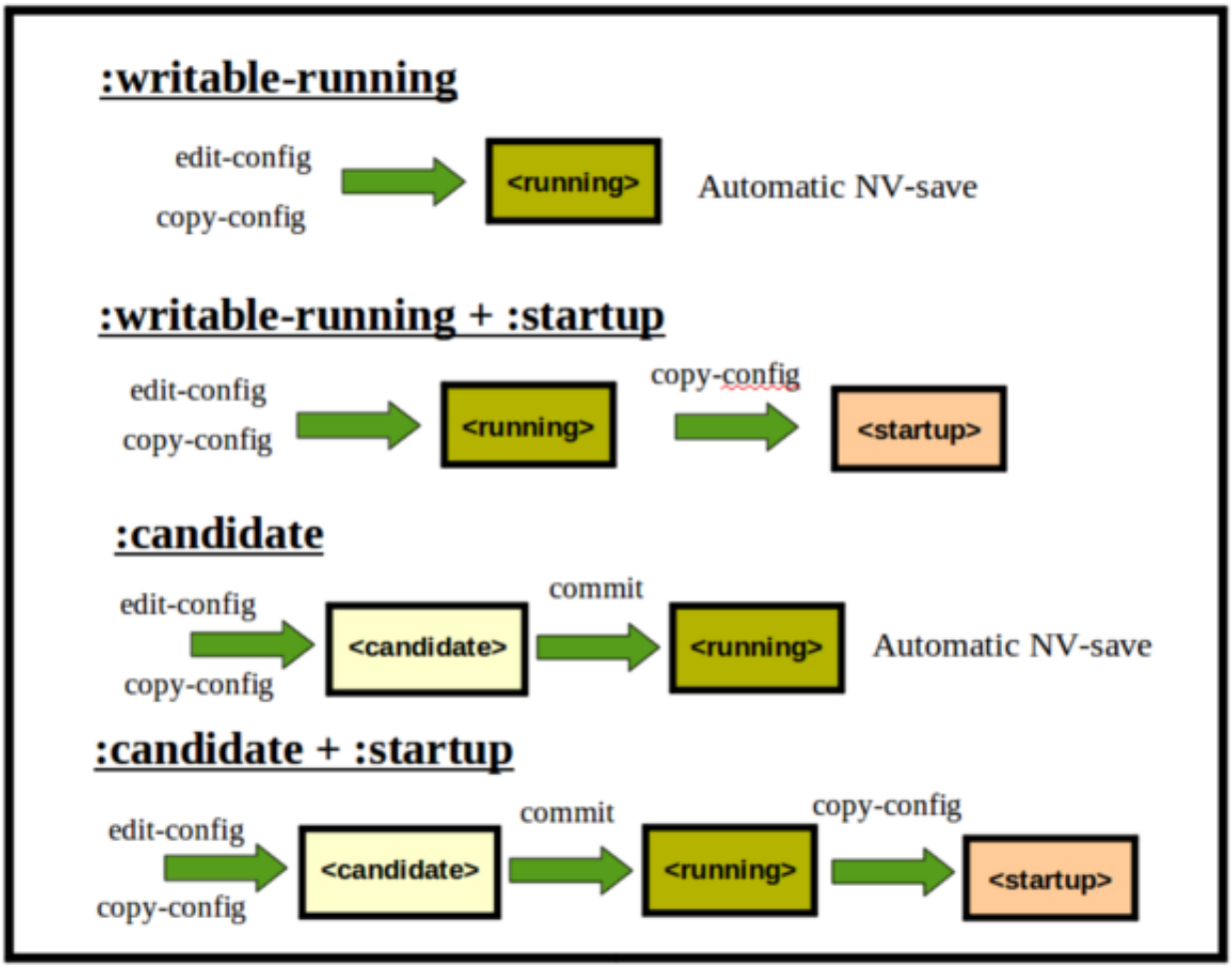
```
nms1> ssh -s -p830 server.example.com netconf
```

The server should sent its hello message right away, and the client should do the same. The following example shows the entire <hello> message that a client is required to send:

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
  </capabilities>
</hello>]]>]]>
```

At this point the server should be waiting for <rpc> requests to process. The server should sent an <rpc-reply> for each <rpc> request. The client can add as many XML attributes to the <rpc> element as desired, and the server will return all those attributes in the <rpc-reply> element. The **message-id** attribute is required by the protocol, although this is not really needed.

Configuration Databases



The NETCONF protocol contains several standard operations which operate on one or more conceptual configuration databases. For example, the <target> parameter for the <edit-config> operation specifies which database to edit.

Configuration databases are represented (in XML) within NETCONF operations as either an empty element identifying a standard database, or a <url> element identifying a non-standard (possibly offline) database.

There are 3 standard conceptual configuration databases

- **<running/>**

  This database represents the entire active configuration currently within the device. It is the only mandatory standard database.

  Unless the server supports the **:candidate** capability, the server must allow this database to be edited directly. Otherwise, the server is not required to support changing this database directly. If it does support it, then the **:writable-running** capability will be advertised by the server to indicate this support.

The <running/> database is also used to contain all the conceptual state information currently available on the device. This can be confusing, but operations such as <get>, which operate on the <running/> database can retrieve status information and statistics, in addition to configuration parameters. The <get-config> operation can be used instead of <get> to retrieve only the configuration data.

- **<candidate/>**

  This database is available if the **:candidate** capability is supported by the server. It is a global scratchpad database that is used to collect edits via 1 or more <edit-config> operations. A client can build up a set of changes which may or may not be validated by the server, until explicitly committed to the running configuration, all at once.

  Unlike the <running/> database, any changes made to the <candidate/> database do not take effect right away within the network device.

  When ready, the client can use the <commit> operation to activate the changes embodied in the <candidate/> database, and make them part of the <running/> configuration.

  After a successful <commit> operation, the <candidate/> database and the <running/> database have the same configuration contents. This special condition is important because a <lock> operation on the <candidate/> database cannot be granted otherwise. It does not matter which session makes any changes to the <candidate/> database. If it is different than the <running/> database, then it cannot be locked.

  Since this is a global database, the client should use the <discard-changes> operation to remove any unwanted changes, if the <commit> operation is not used. This will clean out the <candidate/> database without activating any changes that it may contain, and prevent the next client using this global database from making unintended changes. Care must be taken (e.g., use locks) to make sure multiple sessions do not make any database edits at the same time.

  Server platforms which support the **:candidate** capability usually do not also support the **:writable-running** capability, since mixing direct edits to <running/> would defeat the purpose of using this scratchpad database to collect and validate changes before applying them.

- **<startup/>**

  This database is available if the **:startup** capability is supported by the server. It represents the configuration to use upon the next reboot of the device.

  If present, then the server will not automatically save changes to the <running/> database in non-volatile storage. Instead, a <copy-config> operation is needed to overwrite the contents of the <startup/> database with the current configuration.

  If not present, then the server will automatically update its non-volatile storage any time the running configuration is modified. In either case, the server is required to maintain non-volatile storage of the running configuration, and be able to restore a running configuration after a reboot.

Protocol Operations

Once a NETCONF session is established, the client knows which capabilities the server supports. The client then can send RPC method requests and receive RPC replies from the server. The server's request queue is serialized, so requests will be processed in the order received.

Most operations are designed to select one or two specific configuration databases, but there are also two general operations for ending NETCONF sessions.

The following table summarizes the set of protocol operations, and shows which capabilities must be supported by the server (see next section) in order for a client to use the operation.

Operation	Usage	Description
close-session	:base	Terminate this session
commit	:base AND :candidate	Commit the contents of the <candidate/> configuration database to the <running/> configuration database
copy-config	:base	Copy a configuration database
create-subscription	:notification	Create a NETCONF notification subscription
delete-config	:base	Delete a configuration database
discard-changes	:base AND :candidate	Clear all changes from the <candidate/> configuration database and make it match the <running/> configuration database
edit-config	:base	Modify a configuration database
get	:base	Retrieve data from the running configuration database and/or device statistics
get-config	:base	Retrieve data from the running configuration database
kill-session	:base	Terminate another session
lock	:base	Lock a configuration database so only my session can write
unlock	:base	Unlock a configuration database so any session can write
validate	:base AND :validate	Validate the entire contents of a configuration database

Editing the Configuration

Before using the NETCONF edit operations, the client must determine which database to use as the target by examining the capabilities sent by the server during session establishment.

```
if ':candidate' capability supported:
   target = <candidate/>
else if ':writable-running' capability supported:
   target = <running/>
else if ':url' capability supported:
   target = <url>file://path/to/file</url>
else:
   target = None     #  Server is non-complaint
```

Once the target of the edit operation is determined, the client needs to determine the 'activate' operation that will be needed for the configuration changes to take effect.

```
if ':candidate' capability supported:
   if ':confirmed-commit' capability supported and desired:
      if default timeout of 600 seconds desired:
         activate_fn =  <commit>
                         <confirmed/>
                        </commit>
      else:
         activate_fn =  <commit>
                         <confirmed/>
                         <confirm-timeout>300</confirm-timeout>
                        </commit>
   else:
      activate_fn =  <commit/>
else
   activate_fn = None     #  <running/> or <url> target
```

After the 'target' and 'activate function' are determined, the client needs to determine how the activated configuration changes are saved in non-volatile storage.

```
if ':startup' capability supported:
   save_fn =  <copy-config>
                 <target><startup/></target>
                 <source><running/></source>
              </copy-config>
else
   save_fn = None     #  automatic NV-update
```

Candidate Configuration Example

A basic NETCONF edit transaction if the **candidate** database is the target can be described with the following set of RPC transactions:

1. lock <running/> database
2. lock <candidate/> database
3. edit <candidate/> database
4. commit <candidate/> database
5. unlock <candidate/> database
6. unlock <running/> database

Here is an XML example of this PDU sequence. Note that step 3 can be done multiple times, in arbitrary fashion, since none of the changes take effect until step 4.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <lock>
    <target><running/></target>
```

```
  </lock>
</rpc>

# server returns <ok/> status

<rpc message-id="102"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <lock>
    <target><candidate/></target>
  </lock>
</rpc>

# server returns <ok/> status

<rpc message-id="103"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/></target>
    <default-operation>none</default-operation>
    <test-option>test-then-set</test-option>
    <error-option>stop-on-error</error-option>
    <nc:config
        xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
        xmlns="uri-for-my-data-model-namespace">
      <some-existing-node>
         <my-new-node nc:operation="create">
           <my-new-leaf>7</my-new-leaf>
         </my-new-node>
      </some-existing-node>
    </nc:config>
  </edit-config>
</rpc>

# server returns <ok/> status

<rpc message-id="104"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <commit/>
</rpc>

# server returns <ok/> status

<rpc message-id="105"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <unlock>
    <target><candidate/></target>
  </unlock>
</rpc>

# server returns <ok/> status

<rpc message-id="106"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <unlock>
    <target><running/></target>
  </unlock>
</rpc>

# server returns <ok/> status
```

Running + Startup Configuration Example

A basic NETCONF edit transaction where the **<running/>** database is the target, and the changes need to be explicitly saved to the **<startup/>** database, can be described with the following set of RPC transactions:

1. lock <running/> database
2. lock <startup/> database
3. edit <running/> database
4. copy <running/> database to <startup/> database
5. unlock <startup/> database
6. unlock <running/> database

Here is an XML example of this PDU sequence. Note that step 3 must be done carefully, since the changes will take effect right away.

```
<rpc message-id="107"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <lock>
    <target><running/></target>
  </lock>
</rpc>

# server returns <ok/> status

<rpc message-id="108"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <lock>
    <target><startup/></target>
  </lock>
</rpc>

# server returns <ok/> status

<rpc message-id="109"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><running/></target>
    <default-operation>none</default-operation>
    <test-option>test-then-set</test-option>
    <error-option>rollback-on-error</error-option>
    <nc:config
        xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
        xmlns="uri-for-my-data-model-namespace">
      <some-existing-node>
         <my-new-node nc:operation="create">
           <my-new-leaf>7</my-new-leaf>
         </my-new-node>
      </some-existing-node>
    </nc:config>
  </edit-config>
</rpc>

# server returns <ok/> status

<rpc message-id="110"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
  <copy-config>
    <target><startup/></target>
    <source><running/></source>
  </copy-config>
</rpc>

# server returns <ok/> status

<rpc message-id="111"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <unlock>
    <target><startup/></target>
  </unlock>
</rpc>

# server returns <ok/> status

<rpc message-id="112"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <unlock>
    <target><running/></target>
  </unlock>
</rpc>

# server returns <ok/> status
```

Protocol Capabilities

Many features and mechanisms within the NETCONF protocol do not apply to every use case or every device. Optional mechanisms are given URI handles, which are sent by the server during the NETCONF <hello> message exchange, during session initialization.

For example, if a server allows the running configuration to be edited directly, then it will include the following <capability> element in its <hello> message:

```
<capability>
  urn:ietf:params:netconf:capability:writable-running:1.0
</capability>
```

Each capability is also given a human-readable name, which is used throughout the documentation, instead of the URI representation. By convention, a colon character is pre-pended to the name to indicate it is a capability identifier.

```
:writable-running
```

The NETCONF protocol itself has a capability URI assignment, which is used in the <hello> message exchange to ensure both peers are using the same version of the protocol. The client and server each send

the following capability URI during this exchange. For the client, this is the only capability sent. The server will also include <capability> elements for each optional capability supported, if any.

```
<capability>urn:ietf:params:netconf:base:1.0</capability>
```

Although the base protocol is not optional, and not really a capability, it is given the following name in this document, for consistency and future-proofing.

```
:base
```

Standard Capabilities

The following table summarizes the standard capabilities which a server may choose to support.

- **:candidate**

  The server supports the <candidate/> database. It will allow this special database to be locked, edited, saved, and unlocked. The server will also support the <discard-changes> and basic <commit> operations.

- **:confirmed-commit**

  For servers that support the **:candidate** capability, this additional capability will also be advertised if the server supports the 'confirmed commit' feature. **This special mode requires a server to send two <commit> RPC method requests instead of one, to save any changes to the <running/> database.** If the second request does not arrive within a specified time interval, the server will automatically revert the running configuration to the previous version.

- **:interleave**

  The server will accept <rpc> requests (besides <close-session> while notification delivery is active. The :notification capability must also be present if this capability is advertised.

- **:notification**

  The server supports the basic notification delivery mechanisms defined in RFC 5277, e.g., the <create-subscription> operation will be accepted by the server. Unless the :interleave capability is also supported, only the <close-session> operation must be supported by the server while notification delivery is active.

- **:partial-lock**

  The server supports the <partial-lock> and <partial-unlock> operations, defined in RFC 5717. This allows multiple independent clients to each write to a different part of the <running> configuration at the same time.

- **:rollback-on-error**

The server supports the 'rollback-on-error' value for the <error-option> parameter to the <edit-config> operation. If any error occurs during the requested edit operation, then the target database (usually the running configuration) will be left affected. **This provides an 'all-or-nothing' edit mode for a single <edit-config> request.**

- **:startup**

  The server supports the <startup/> database. It will allow the running configuration to be copied to this special database. It can also be locked, and unlocked, but a server is not required to allow it to be edited.

- **:url**

  The server supports the <url> parameter value form to specify protocol operation source and target parameters. The capability URI for this feature will indicate which schemes (e.g., file, https, sftp) that the server supports within a particular URL value. The 'file' scheme allows for editable local configuration databases. The other schemes allow for remote storage of configuration databases.

- **:validate**

  The server supports the <validate> operation. When this operation is requested on a target database, the server will perform some amount of parameter validation and referential integrity checking. Since the standard does not define exactly what must be validated by this operation, a client cannot really rely on it for anything useful.

  This operation is used to validate a complete database. There is no standard way to validate a single edit request against a target database, however a non-standard set-option for the <edit-config> operation called **test-only** has been defined for this purpose.

- **:writable-running**

  The server allows the client to change the running configuration directly. Either this capability or the **:candidate** capability will be supported by the server, but usually not both.

- **:xpath**

  The server fully supports the XPath 1.0 specification for filtered retrieval of configuration and other database contents. The 'type' attribute within the <filter> parameter for <get> and <get-config> operations may be set to 'xpath'. The 'select' attribute (which contains the XPath expression) will also be supported by the server.

  A server may support partial XPath retrieval filtering, but it cannot advertise the **:xpath** capability unless the entire XPath 1.0 specification is supported.

Notifications

NETCONF has a notification delivery mechanism, defined in RFC 5277.

The <create-subscription> operation is used to setup session-specific filtering and optional buffered

notification delivery parameters. These parameters cannot be modified after notification delivery has started on a particular session. The only way to stop live notification delivery is to terminate the session, usually with the <close-session> operation.

The NETCONF notification definitions are divided into 2 namespaces, one for the top-level <notification> element, and the other for some initial notification content, such as the <replayComplete> event.

These definitions have been converted to YANG, named nc-notifications.yang and notifications.yang.

Completed RFC Specifications

RFC 4741

Defines the NETCONF protocol RPC layer and operations layer.

Status: Proposed Standard RFC, mandatory-to-implement

NETCONF Configuration Protocol

RFC 4742

Defines the NETCONF-over-SSH transport mapping.

Status: Proposed Standard RFC, mandatory-to-implement

Using the NETCONF Configuration Protocol over Secure Shell (SSH)

RFC 4743

Defines the NETCONF-over-SOAP transport mapping.

Status: Proposed Standard RFC, optional-to-implement

Using NETCONF over the Simple Object Access Protocol (SOAP)

RFC 4744

Defines the NETCONF-over-BEEP transport mapping.

Status: Proposed Standard RFC, optional-to-implement

Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP)

RFC 5277

Defines the NETCONF notification delivery mechanisms.

Status: Proposed Standard RFC, optional-to-implement

NETCONF Event Notifications

RFC 5381

Documents some NETCONF-over-SOAP implementation experience.

Status: Informational RFC, nothing-to-implement

Experience of Implementing NETCONF over SOAP

RFC 5539

Defines the NETCONF-over-TLS transport mapping.

Status: Proposed Standard RFC, optional-to-implement

NETCONF Over Transport Layer Security (TLS)

RFC 5717

Defines a partial database locking mechanism (based on instance-identifier or XPath expressions) for the NETCONF protocol.

Status: Proposed Standard RFC, optional-to-implement

Partial Lock RPC for NETCONF

RFC 6022

Defines a server monitoring data model and schema retrieval mechanism for the NETCONF protocol.

Status: Proposed Standard RFC, optional-to-implement

NETCONF Monitoring Schema

RFC 6241

Defines a minor update to the NETCONF protocol.

Status: Proposed Standard RFC, mandatory-to-implement

Network Configuration Protocol

RFC 6242

Defines a minor update to the NETCONF protocol over Secure Shell document.

Status: Proposed Standard RFC, mandatory-to-implement

Using the Network Configuration Protocol over Secure Shell

RFC 6243

Defines a mechanism to control the filtering of leaf objects containing the 'default' value, during NETCONF retrieval operations.

Status: Proposed Standard RFC, optional-to-implement

With-defaults capability for NETCONF

RFC 6470

Defines some NETCONF notification events for the NETCONF notification stream.

Status: Proposed Standard RFC, optional-to-implement

NETCONF Notification Events

RFC 6536

Defines an access control model for all forms of NETCONF content.

Status: Proposed Standard RFC, optional-to-implement

NETCONF Access Control Model (NACM)

Standards Work in Progress

RFC4743 and RFC4744 to Historic status

This is a document to remove the SOAP and BEEP transport mappings from the NETCONF standards.

Intended Status: Informational RFC, nothing-to-implement

RFC4743 and RFC4744 to Historic status

NETCONF Over TLS (RFC 5539-bis)

The Network Configuration Protocol (NETCONF) provides mechanisms to install, manipulate, and delete the configuration of network devices. This document describes how to use the Transport Layer Security (TLS) protocol to secure NETCONF exchanges. This document obsoletes RFC 5539.

Intended Status: Proposed Standard RFC, optional-to-implement

NETCONF Over Transport Layer Security (TLS)

Additional NETCONF Resources

- NETCONF WG Charter
- NETCONF WG Wiki Page
- Errata for RFC 4742 (NETCONF Over SSH)
- IANA assignment for NETCONF namespace
- IANA assignment for NETCONF base namespace
- IANA assignment for NETCONF over SOAP URI
- IANA version of the NETCONF protocol XSD
- Tutorial Slides (circa 2004)
- NETCONF White Paper by David French (embeddedmind.com)

Send comments to web-admin@netconfcentral.org