Network Configuration Protocol

Last Updated: December 20, 2011

The Network Configuration Protocol (NETCONF) defines a simple mechanism through which a network device can be managed, configuration data information can be retrieved, and new configuration data can be uploaded and manipulated. NETCONF uses Extensible Markup Language (XML)-based data encoding for the configuration data and protocol messages.

You can use the NETCONF over SSHv2 feature to perform network configurations via the Cisco command-line interface (CLI) over an encrypted transport. The NETCONF Network Manager, which is the NETCONF client, must use Secure Shell Version 2 (SSHv2) as the network transport to the NETCONF server. Multiple NETCONF clients can connect to the NETCONF server.

You can use the NETCONF over BEEP feature to send notifications of any configuration change over NETCONF. A notification is an event indicating that a configuration change has happened. The change can be a new configuration, deleted configuration, or changed configuration. The notifications are sent at the end of a successful configuration operation as one message showing the set of changes, rather than individual messages for each line in the configuration that is changed.

Blocks Extensible Exchange Protocol (BEEP) can use the Simple Authentication and Security Layer (SASL) profile to provide simple and direct mapping to the existing security model. Alternatively, NETCONF over BEEP can use the transport layer security (TLS) to provide a strong encryption mechanism with either server authentication or server and client-side authentication.

Finding Feature Information

Your software release may not support all the features documented in this module. For the latest feature information and caveats, see the release notes for your platform and software release. To find information about the features documented in this module, and to see a list of the releases in which each feature is supported, see the Feature Information Table at the end of this document.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Prerequisites for NETCONF

- NETCONF over SSHv2 requires that a vty line be available for each NETCONF session as specified in the **netconf max-session**command.
- NETCONF over BEEP listeners require SASL to be configured.

Restrictions for NETCONF

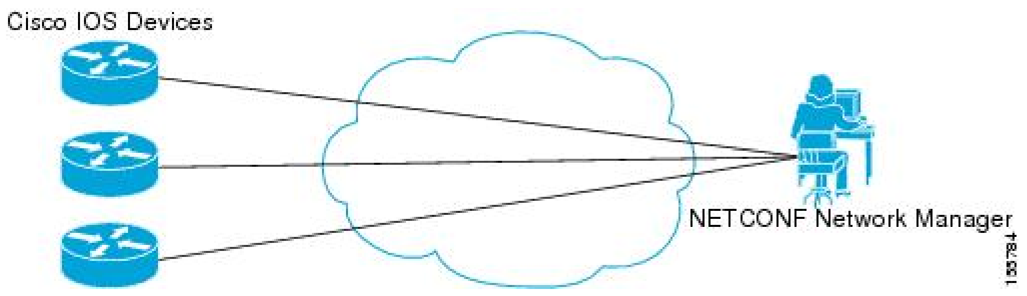
- NETCONF SSHv2 supports a maximum of 16 concurrent sessions.
- Only SSH version 2 is supported.
- You must be running a crypto image in order to configure BEEP using TLS.

Information About NETCONF

NETCONF over SSHv2

To run the NETCONF over SSHv2 feature, the client (a Cisco device running Cisco IOS XE software) establishes an SSH transport connection with the server (a NETCONF Network Manager.) Figure 1 shows a basic NETCONF over SSHv2 network configuration. The client and server exchange keys for security and password encryption. The user ID and password of the SSHv2 session running NETCONF are used for authorization and authentication purposes. The user privilege level is enforced and the client session may not have full access to the NETCONF operations if the privilege level is not high enough. If authentication, authorization, and accounting (AAA) is configured, the AAA service is used as if a user had established an SSH session directly to the device. Using the existing security configuration makes the transition to NETCONF almost seamless. Once the client has been successfully authenticated, the client invokes the SSH connection protocol and the SSH session is established. After the SSH session is established, the user or application invokes NETCONF as an SSH subsystem called “netconf.”

Figure 1 NETCONF over SSHv2



Secure Shell Version 2

SSHv2 runs on top of a reliable transport layer and provides strong authentication and encryption capabilities. SSHv2 provides a means to securely access and securely execute commands on another computer over a network.

NETCONF does not support SSH version 1. The configuration for the SSH Version 2 server is similar to the configuration for SSH version 1. Use the **ip ssh version** command to specify which version of SSH that you want to configure. If you do not configure this command, SSH by default runs in compatibility mode; that is, both SSH version 1 and SSH version 2 connections are honored.

Note SSH version 1 is a protocol that has never been defined in a standard. If you do not want your router to fall back to the undefined protocol (version 1), you should use the **ip ssh version** command and specify version 2.

Use the **ip ssh rsa keypair-name** command to enable an SSH connection using Rivest, Shamir, and Adelman (RSA) keys that you have configured. If you configure the **ip ssh rsa keypair-name** command with a key-pair name, SSH is enabled if the key pair exists, or SSH will be enabled if the key pair is generated later. If you use this command to enable SSH, you do not need to configure a hostname and a domain name.

NETCONF over BEEP

The NETCONF over BEEP feature allows you to enable BEEP as the transport protocol to use during NETCONF sessions. Using NETCONF over BEEP, you can configure either the NETCONF server or the NETCONF client to initiate a connection, thus supporting large networks of intermittently connected devices, and those devices that must reverse the management connection where there are firewalls and Network Address Translators (NATs).

BEEP is a generic application protocol framework for connection-oriented, asynchronous interactions. It is intended to provide the features that traditionally have been duplicated in various protocol implementations. BEEP typically runs on top of TCP and allows the exchange of messages. Unlike HTTP and similar protocols, either end of the connection can send a message at any time. BEEP also includes facilities for encryption and authentication and is highly extensible.

The BEEP protocol contains a framing mechanism that permits simultaneous and independent exchanges of messages between peers. These messages are usually structured using XML. All exchanges occur in the context of a binding to a well-defined aspect of the application, such as transport security, user authentication, or data exchange. This binding forms a channel; each channel has an associated profile that defines the syntax and semantics of the messages exchanged.

The BEEP session is mapped onto the NETCONF service. When a session is established, each BEEP peer advertises the profiles it supports. During the creation of a channel, the client (the BEEP initiator) supplies one or more proposed profiles for that channel. If the server (the BEEP listener) creates the channel, it selects one of the profiles and sends it in a reply. The server may also indicate that none of the profiles are acceptable, and decline creation of the channel.

BEEP allows multiple data exchange channels to be simultaneously in use.

Although BEEP is a peer-to-peer protocol, each peer is labelled according to the role it is performing at a given time. When a BEEP session is established, the peer that awaits new connections is the BEEP listener. The other peer, which establishes a connection to the listener, is the BEEP initiator. The BEEP peer that starts an exchange is the client, and the other BEEP peer is the server. Typically, a BEEP peer that acts in the server role also performs in the listening role. However, because BEEP is a peer-to-peer protocol, the BEEP peer that acts in the server role is not required to also perform in the listening role.

Simple Authentication and Security Layer

The SASL is an Internet standard method for adding authentication support to connection-based protocols. SASL can be used between a security appliance and a Lightweight Directory Access Protocol (LDAP) server to secure user authentication.

Transport Layer Security

The TLS is an application-level protocol that provides for secure communication between a client and server by allowing mutual authentication, the use of hash for integrity, and encryption for privacy. TLS relies upon certificates, public keys, and private keys.

Certificates are similar to digital ID cards. They prove the identity of the server to clients. Each certificate includes the name of the authority that issued it, the name of the entity to which the certificate was issued, the entity’s public key, and time stamps that indicate the certificate’s expiration date.

Public and private keys are the ciphers used to encrypt and decrypt information. Although the public key is shared, the private key is never given out. Each public-private key pair works together. Data encrypted with the public key can be decrypted only with the private key.

Access Lists

You can optionally configure access lists for use with NETCONF over SSHv2 sessions. An access list is a sequential collection of permit and deny conditions that apply to IP addresses. The Cisco IOS XE software tests addresses against the conditions in an access list one by one. The first match determines whether the software accepts or rejects the address. Because the software stops testing conditions after the first match, the order of the conditions is critical. If no conditions match, the software rejects the address.

The two main tasks involved in using access lists are as follows:

1. Creating an access list by specifying an access list number or name and access conditions.
2. Applying the access list to interfaces or terminal lines.

For more information about configuring access lists, see “Traffic Filtering, Firewalls, and Virus Detection” section of the Cisco IOS XE Security Configuration Guide.

NETCONF Notifications

NETCONF sends notifications of any configuration change over NETCONF. A notification is an event indicating that a configuration change has occurred. The change can be a new configuration, deleted configuration, or changed configuration. The notifications are sent at the end of a successful configuration operation as one message that shows the set of changes rather than showing individual messages for each line that is changed in the configuration.

How to Configure NETCONF

This section contains the following tasks:

- Enabling NETCONF over BEEP, page 12
- Configuring the NETCONF Network Manager Application, page 16
- Delivering NETCONF Payloads, page 17
- Formatting NETCONF Notifications, page 19
- Monitoring and Maintaining NETCONF Sessions, page 22

Enabling SSH Version 2 Using a Hostname and Domain Name

Perform this task to configure your router for SSH version 2 using a hostname and domain name. You may also configure SSH version 2 by using the RSA key pair configuration (see Enabling SSH Version 2 Using RSA Key Pairs, page 6).

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **hostname** *hostname*
4. **ip domain-name** *name*
5. **crypto key generate rsa**
6. **ip ssh** [**timeout** *seconds* | **authentication-retries** *integer*]
7. **ip ssh version 2**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**hostname** *hostname* **Example:** `Router(config)# hostname host1`	Configures a hostname for your router.

	Command or Action	Purpose
Step 4	**ip domain-name** *name* **Example:** `Router(config)# ip domain-name domain1.com`	Configures a domain name for your router.
Step 5	**crypto key generate rsa** **Example:** `Router(config)# crypto key generate rsa`	Enables the SSH server for local and remote authentication.
Step 6	**ip ssh** [**timeout** *seconds* | **authentication-retries** *integer*] **Example:** `Router(config)# ip ssh timeout 120`	(Optional) Configures SSH control variables on your router.
Step 7	**ip ssh version 2** **Example:** `Router(config)# ip ssh version 2`	Specifies the version of SSH to be run on your router.

Enabling SSH Version 2 Using RSA Key Pairs

Perform this task to enable SSH version 2 without configuring a hostname or domain name. SSH version 2 will be enabled if the key pair that you configure already exists or if it is generated later. You may also configure SSH version 2 by using the hostname and domain name configuration. (See Enabling SSH Version 2 Using a Hostname and Domain Name, page 5.)

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **ip ssh rsa keypair-name** *keypair-name*
4. **crypto key generate rsa usage-keys label** *key-label* **modulus** *modulus-size*
5. **ip ssh** [**timeout** *seconds* | **authentication-retries** *integer*]
6. **ip ssh version 2**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**ip ssh rsa keypair-name** *keypair-name* **Example:** `Router(config)# ip ssh rsa keypair-name sshkeys`	Specifies which RSA keypair to use for SSH usage. **Note** A Cisco IOS XE router can have many RSA key pairs.
Step 4	**crypto key generate rsa usage-keys label** *key-label* **modulus** *modulus-size* **Example:** `Router(config)# crypto key generate rsa usage-keys label sshkeys modulus 768`	Enables the SSH server for local and remote authentication on the router. For SSH version 2, the modulus size must be at least 768 bits. **Note** To delete the RSA key pair, use the **crypto key zeroize rsa** command. After you have deleted the RSA command, you automatically disable the SSH server.
Step 5	**ip ssh** [**timeout** *seconds* | **authentication-retries** *integer*] **Example:** `Router(config)# ip ssh time-out 120`	Configures SSH control variables on your router.
Step 6	**ip ssh version 2** **Example:** `Router(config)# ip ssh version 2`	Specifies the version of SSH to be run on a router.

Starting an Encrypted Session with a Remote Device

Perform this task to start an encrypted session with a remote networking device. (You do not have to enable your router. SSH can be run in disabled mode.)

From any UNIX or UNIX-like device, the following command is typically used to form an SSH session:

```
ssh -2 user@router.example.com netconf
```

SUMMARY STEPS

1. Do one of the following:
   - **ssh** [**-v** {**1** | **2**}] [**-c** {**3des**| **aes128-cbc** | **aes192-cbc**| **aes256-cbc**}] [**-m**{**hmac-md5** | **hmac-md5-96** | **hmac-sha1** | **hmac-sha1-96**}] [**l** *userid*] [**-o numberofpasswordprompts** *n*] [**-p** *port-num*] {*ip-addr* | *hostname*} [*command*]

DETAILED STEPS

	Command or Action	Purpose

Step 1	Do one of the following:
	• **ssh** [**-v** {**1** | **2**}] [**-c** {**3des**| **aes128-cbc** | **aes192-cbc**| **aes256-cbc**}] [**-m**{**hmac-md5** | **hmac-md5-96** | **hmac-sha1** | **hmac-sha1-96**}] [**l** *userid*] [**-o numberofpasswordprompts** *n*] [**-p** *port-num*] {*ip-addr* | *hostname*} [*command*]
	
	Example:
	`Router# ssh -v 2 -c aes256-cbc -m hmac-sha1-96 -l user2 10.76.82.24`
	
	Example:
	`Router#`
	`ssh -v 2 -c aes256-cbc -m hmac-sha1-96 user2@10.76.82.24`
		Starts an encrypted session with a remote networking device.
		
		The first example adheres to the SSH version 2 conventions. A more natural and common way to start a session is by linking the username with the hostname. For example, the second configuration example provides an end result that is identical to that of the first example.

- Troubleshooting Tips, page 8
- What to Do Next, page 8

Troubleshooting Tips

The **ip ssh version** command can be used for troubleshooting your SSH configuration. By changing versions, you can determine which SSH version has a problem.

What to Do Next

For more information about the **ssh** command, see the Cisco IOS Security Command Reference.

Verifying the Status of the Secure Shell Connection

Perform this task to display the status of the SSH connection on your router.

SUMMARY STEPS

1. **enable**
2. **show ssh**
3. **show ip ssh**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**show ssh** **Example:** `Router# show ssh`	Displays the status of SSH server connections.
Step 3	**show ip ssh** **Example:** `Router# show ip ssh`	Displays the version and configuration data for SSH.

- Examples, page 9

Examples

The following output from the **show ssh**command displays status about SSH version 2 connections:

```
Router# show ssh
Connection Version Mode Encryption  Hmac                State        Username
1          2.0     IN   aes128-cbc  hmac-md5     Session started       lab
1          2.0     OUT  aes128-cbc  hmac-md5     Session started       lab
%No SSHv1 server connections running.
```

The following output from the **show ip ssh** command displays the version of SSH that is enabled, the authentication timeout values, and the number of authentication retries:

```
Router# show ip ssh
SSH Enabled - version 2.0
Authentication timeout: 120 secs; Authentication retries: 3
```

Enabling NETCONF over SSHv2

Perform this task to enable NETCONF over SSHv2.

- SSHv2 must be enabled.

Note There must be at least as many vty lines configured as there are concurrent NETCONF sessions.

Note

- A minimum of four concurrent NETCONF sessions must be configured.
- A maximum of 16 concurrent NETCONF sessions can be configured.
- NETCONF does not support SSHv1.

>

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **netconf ssh** [**acl** *access-list-number*]
4. **netconf lock-time** *seconds*
5. **netconf max-sessions** *session*

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**netconf ssh** [**acl** *access-list-number*] **Example:** `Router(config)# netconf ssh acl 1`	Enables NETCONF over SSHv2. Optionally, you can configure an access control list for this NETCONF session.
Step 4	**netconf lock-time** *seconds* **Example:** `Router(config)# netconf lock-time 60`	(Optional) Specifies the maximum time a NETCONF configuration lock is in place without an intermediate operation. The valid range is 1 to 300 seconds. The default value is 10 seconds.

	Command or Action	Purpose
Step 5	**netconf max-sessions** *session* **Example:** `Router(config)# netconf max-sessions 5`	(Optional) Specifies the maximum number of concurrent NETCONF sessions allowed.

Configuring an SASL Profile

To enable NETCONF over BEEP using SASL, you must first configure an SASL profile, which specifies which users are allowed access into the router. Perform this task to configure an SASL profile.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **sasl profile** *profile-name*
4. **mechanism di gest-md5**
5. **server** *user-name* **password** *password*

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**sasl profile** *profile-name* **Example:** `Router(config)# sasl profile beep`	Configures an SASL profile and enters SASL profile configuration mode.

	Command or Action	Purpose
Step 4	**mechanism di gest-md5** **Example:** `Router(config-SASL-profile)# mechanism digest-md5`	Configures the SASL profile mechanism.
Step 5	**server** *user-name* **password** *password* **Example:** `Router(config-SASL-profile)# server user1 password password1`	Configures an SASL server.

Enabling NETCONF over BEEP

Perform this task to enable NETCONF over BEEP.

- There must be at least as many vty lines configured as there are concurrent NETCONF sessions.
- If you configure NETCONF over BEEP using SASL, you must first configure an SASL profile.

Note

- A minimum of four concurrent NETCONF sessions must be configured.
- A maximum of 16 concurrent NETCONF sessions can be configured.

>

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **crypto key generate rsa general-keys**
4. **crypto pki trustpoint** *name*
5. **enrollment url** *url*
6. **subject-name** *name*
7. **revocation-check** *method1* [*method2*[*method3*]]
8. **exit**
9. **crypto pki authenticate** *name*
10. **crypto pki enroll** *name*
11. **netconf lock-time** *seconds*
12. **line vty** line-number [*ending-line-number*]
13. **netconf max-sessions** *session*
14. **netconf beep initiator** {*hostname* | *ip-address*} *port-number* **user** *sasl-user* **password** *sasl-password*[**encrypt** *trustpoint*] [**reconnect-time** *seconds*]
15. **netconf beep listener** [*port-number*] [**acl** *access-list-number*] [**sasl** *sasl-profile*] [**encrypt** *trustpoint*]

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**crypto key generate rsa general-keys** **Example:** `Router(config)# crypto key generate rsa general-keys`	Generates RSA key pairs and specifies that the general-purpose key pair should be generated. Perform this step only once.

	Command or Action	Purpose
Step 4	**crypto pki trustpoint** *name* **Example:** `Router(config)# crypto pki trustpoint my_trustpoint`	Declares the trustpoint that your router should use and enters ca-trustpoint configuration mode.
Step 5	**enrollment url** *url* **Example:** `Router(ca-trustpoint)# enrollment url http://10.2.3.3:80`	Specifies the enrollment parameters of a certification authority (CA).
Step 6	**subject-name** *name* **Example:** `Router(ca-trustpoint)# subject-name CN=dns_name_of_host.com`	Specifies the subject name in the certificate request. **Note** The subject name should be the Domain Name System (DNS) name of the device.
Step 7	**revocation-check** *method1* [*method2*[*method3*]] **Example:** `Router(ca-trustpoint)# revocation-check none`	Checks the revocation status of a certificate.
Step 8	**exit** **Example:** `Router(ca-trustpoint)# exit`	Exits ca-trustpoint configuration mode and returns to global configuration mode.
Step 9	**crypto pki authenticate** *name* **Example:** `Router(config)# crypto pki authenticate my_trustpoint`	Authenticates the certification authority (by getting the certificate of the CA).
Step 10	**crypto pki enroll** *name* **Example:** `Router(config)# crypto pki enroll my_trustpoint`	Obtains the certificate or certificates for your router from CA.

	Command or Action	Purpose
Step 11	**netconf lock-time** *seconds* **Example:** `Router(config)# netconf lock-time 60`	(Optional) Specifies the maximum time a NETCONF configuration lock is in place without an intermediate operation. The valid value range for the seconds argument is 1 to 300 seconds. The default value is 10 seconds.
Step 12	**line vty** line-number [*ending-line-number*] **Example:** `Router(config)# line vty 0 15`	Identifies a specific virtual terminal line for remote console access. You must configure the same number of vty lines as maximum NETCONF sessions.
Step 13	**netconf max-sessions** *session* **Example:** `Router(config)# netconf max-sessions 16`	(Optional) Specifies the maximum number of concurrent NETCONF sessions allowed.
Step 14	**netconf beep initiator** {*hostname* | *ip-address*} *port-number* **user** *sasl-user* **password** *sasl-password*[**encrypt** *trustpoint*] [**reconnect-time** *seconds*] **Example:** `Router(config)# netconf beep initiator host1 23 user user1 password password1 encrypt 23 reconnect-time 60`	(Optional) Specifies BEEP as the transport protocol for NETCONF sessions and configures a peer as the BEEP initiator. **Note** Perform this step to configure a NETCONF BEEP initiator session. You can also optionally configure a BEEP listener session.
Step 15	**netconf beep listener** [*port-number*] [**acl** *access-list-number*] [**sasl** *sasl-profile*] [**encrypt** *trustpoint*] **Example:** `Router(config)# netconf beep listener 26 acl 101 sasl profile1 encrypt 25`	(Optional) Specifies BEEP as the transport protocol for NETCONF and configures a peer as the BEEP listener. **Note** Perform this step to configure a NETCONF BEEP listener session. You can also optionally configure a BEEP initiator session.

Configuring the NETCONF Network Manager Application

SUMMARY STEPS

1. Use the following CLI string to configure the NETCONF Network Manager application to invoke NETCONF as an SSH subsystem:
2. As soon as the NETCONF session is established, indicate the server capabilities by sending an XML document containing a <hello>:
3. Use the following XML string to enable the NETCONF network manager application to send and receive NETCONF notifications:
4. Use the following XML string to stop the NETCONF network manager application from sending or receiving NETCONF notifications:

DETAILED STEPS

Step 1 Use the following CLI string to configure the NETCONF Network Manager application to invoke NETCONF as an SSH subsystem:

Example:

```
Unix Side: ssh-2 -s companyname@10.1.1.1 netconf
```

Step 2 As soon as the NETCONF session is established, indicate the server capabilities by sending an XML document containing a <hello>:

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
    <hello>
      <capabilities>
        <capability>
            urn:ietf:params:xml:ns:netconf:base:1.0
          </capability>
          <capability>
            urn:ietf:params:ns:netconf:capability:startup:1.0
          </capability>
       </capabilities>
    <session-id>4<session-id>
</hello>
]]>]]>
```

The client also responds by sending an XML document containing a <hello>:

Example:

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

Note Although the example shows the server sending a <hello> message followed by the client's message, both sides send the message as soon as the NETCONF subsystem is initialized, perhaps simultaneously.

Tip All NETCONF requests must end with]]>]]> which denotes an end to the request. Until the]]>]]> sequence is sent, the device will not process the request.

See Configuring NETCONF over SSHv2 Example, page 24 for a specific example.

Step 3 Use the following XML string to enable the NETCONF network manager application to send and receive NETCONF notifications:

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rpc message-id="9.0"><notification-on/>
</rpc>]]>]]>
```

Step 4 Use the following XML string to stop the NETCONF network manager application from sending or receiving NETCONF notifications:

Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rpc message-id="9.13"><notification-off/>
</rpc>]]>]]>
```

Delivering NETCONF Payloads

Use the following XML to deliver the NETCONF payload to the Network Manager application:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.cisco.com/cpi_10/schema"
elementFormDefault="qualified" attributeFormDefault="unqualified" xmlns="http://
www.cisco.com/cpi_10/schema" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <!--The following elements define the cisco extensions for the content of the filter
element in a <get-config> request. They allow the client to specify the format of the
response and to select subsets of the entire configuration to be included.-->
   <xs:element name="config-format-text-block">
      <xs:annotation>
         <xs:documentation>If this element appears in the filter, then the cllient is
requesting that the response data be sent in config command block format.</
xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="text-filter-spec" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="config-format-text-cmd">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="text-filter-spec"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="config-format-xml">
      <xs:annotation>
         <xs:documentation>When this element appears in the filter of a get-config
```

```
request, the results are to be returned in E-DI XML format. The content of this element
is treated as a filter.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="xs:anyType"/>
         </xs:complexContent>
      </xs:complexType>
   </xs:element>
   <!--These elements are used in the filter of a <get> to specify operational data to
return.-->
   <xs:element name="oper-data-format-text-block">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="show" type="xs:string" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="oper-data-format-xml">
      <xs:complexType>
         <xs:sequence>
            <xs:any/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <!--When confing-format-text format is specified, the following describes the content
of the data element in the response-->
   <xs:element name="cli-config-data">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="cmd" type="xs:string" maxOccurs="unbounded">
               <xs:annotation>
                  <xs:documentation>Content is a command. May be multiple lines.</
xs:documentation>
               </xs:annotation>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="cli-config-data-block" type="xs:string">
      <xs:annotation>
         <xs:documentation>The content of this element is the device configuration as it
would be sent to a terminal session. It contains embedded newline characters that must be
preserved as they represent the boundaries between the individual command lines</
xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:element name="text-filter-spec">
      <xs:annotation>
         <xs:documentation>If this element is included in the config-format-text element,
then the content is treated as if the string was appended to the "show running-config"
command line.</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:element name="cli-oper-data-block">
      <xs:complexType>
         <xs:annotation>
            <xs:documentation> This element is included in the response to get operation.
Content of this element is the operational data in text format.</xs:documentation>
         </xs:annotation>
         <xs:sequence>
            <xs:element name="item" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="show"/>
                     <xs:element name="response"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:schema>
```

Formatting NETCONF Notifications

The NETCONF Network Manager application uses .xsd schema files to describe the format of the XML NETCONF notification messages being sent between a NETCONF Network Manager application and a router running NETCONF over SSHv2 or BEEP. These files can be displayed in a browser or a schema reading tool. You can use these schema to validate that the XML is correct. These schema describe the format, not the content, of the data being exchanged.

NETCONF uses the <edit-config> function to load all of a specified configuration to a specified target configuration. When this new configuration is entered, the target configuration is not replaced. The target configuration is changed according to the data and requested operations of the requesting source.

The following are schemas for the NETCONF <edit-config> function in CLI, CLI block, and XML format.

NETCONF <edit-config> Request: CLI Format

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <edit-config>
      <target>
         <running/>
      </target>
      <config>
         <cli-config-data>
<cmd>hostname test</cmd>
            <cmd>interface fastEthernet0/1</cmd>
            <cmd>ip address 192.168.1.1 255.255.255.0</cmd>
</cli-config-data>
      </config>
   </edit-config>
</rpc>]]>]]>
```

NETCONF <edit-config> Response: CLI Format

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:netconf:base:1.0">
   <ok/>
</rpc-reply>]]>]]>
```

NETCONF <edit-config> Request: CLI-Block Format

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id="netconf.mini.edit.3">
   <edit-config>
      <target>
         <running/>
      </target>
      <config>
         <cli-config-data-block>
            hostname bob
            interface fastEthernet0/1
            ip address 192.168.1.1 255.255.255.0
         </cli-config-data-block>
      </config>
   </edit-config>
</rpc>]]>]]>
```

NETCONF <edit-config> Response: CLI-Block Format

```
<?xml version="1.0" encoding=\"UTF-8\"?>
<rpc-reply message-id="netconf.mini.edit.3" xmlns="urn:ietf:params:netconf:base:1.0">
```

```
   <ok/>
</rpc-reply>]]>]]>
```

The following are schemas for the NETCONF <get-config> function in CLI and CLI-block format.

NETCONF <get-config> Request: CLI Format

```
<?xml version="1.0" encoding=\"UTF-8\"?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-config>
      <source>
         <running/>
      </source>
      <filter>
         <config-format-text-cmd>
            <text-filter-spec> | inc interface </text-filter-spec>
         </config-format-text-cmd>
</filter>
   </get-config>
</rpc>]]>]]>
```

NETCONF <get-config> Response: CLI Format

```
<?xml version="1.0" encoding=\"UTF-8\"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <data>
      <cli-config-data>
         <cmd>interface FastEthernet0/1</cmd>
         <cmd>interface FastEthernet0/2</cmd>
      </cli-config-data>
   </data>
</rpc-reply>]]>]]>
```

NETCONF <get-config> Request: CLI-Block Format

```
<?xml version="1.0" encoding=\"UTF-8\"?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-config>
      <source>
         <running/>
      </source>
      <filter>
         <config-format-text-block>
            <text-filter-spec> | inc interface </text-filter-spec>
         </config-format-text-block>
      </filter>
   </get-config>
</rpc>]]>]]>
```

NETCONF <get-config> Response: CLI-Block Format

```
<?xml version="1.0" encoding=\"UTF-8\"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <data>
      <cli-config-data-block>
         interface FastEthernet0/1
         interface FastEthernet0/2
      </cli-config-data-block>
   </data>
</rpc-reply>]]>]]>
```

NETCONF uses the <get> function to retrieve configuration and device-state information. The NETCONF <get> format is the equivalent of a Cisco IOS **show** command. The <filter> parameter specifies the portion of the system configuration and device-state data to retrieve. If the <filter> parameter is empty, nothing is returned.

The following are schemas for the <get> function in CLI and CLI-block format.

NETCONF <get> Request: CLI Format

```
<?xml version="1.0" encoding=\"UTF-8\"?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get>
       <filter>
          <config-format-text-cmd>
             <text-filter-spec> | include interface </text-filter-spec>
          </config-format-text-cmd>
          <oper-data-format-text-block>
             <show>interfaces</show>
             <show>arp</show>
          </oper-data-format-text-block>
       </filter>
    </get>
 </rpc>]]>]]>
```

NETCONF <get> Response: CLI Format

```
<?xml version="1.0" encoding=\"UTF-8\"?>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <data>
       <cli-config-data>
<cmd>interface Loopback0</cmd>
<cmd>interface GigabitEthernet0/1</cmd>
<cmd>interface GigabitEthernet0/2</cmd>
</cli-config-data>
<cli-oper-data-block>
          <item>
             <show>interfaces</show>
             <response>
                <!-- output of "show interfaces" ------>
             </response>
          <show>arp</show>
          <item>
             <show>arp</show>
             <response>
                <!-- output of "show arp" ------>
             </response>
          </item>
       </cli-oper-data-block>
    </data>
</rpc-reply>]]>]]>
```

NETCONF <get> Request: CLI-Block Format

```
<?xml version="1.0" encoding=\"UTF-8\"?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get>
       <filter>
          <config-format-text-block>
             <text-filter-spec> | include interface </text-filter-spec>
          </config-format-text-block>
          <oper-data-format-text-block>
             <show>interfaces</show>
             <show>arp</show>
          </oper-data-format-text-block>
       </filter>
    </get>
 </rpc>]]>]]>
```

NETCONF <get> Response: CLI-Block Format

```
<?xml version="1.0" encoding=\"UTF-8\"?>
```

```
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <data>
       <cli-config-data-block>
interface Loopback0
interface GigabitEthernet0/1
interface GigabitEthernet0/2
       </cli-config-data-block>
       <cli-oper-data-block>
          <item>
             <show>interfaces</show>
             <response>
                <!-- output of "show interfaces" ------>
             </response>
          <show>arp</show>
          <item>
             <show>arp</show>
             <response>
                <!-- output of "show arp" ------>
             </response>
          </item>
       </cli-oper-data-block>
    </data>
</rpc-reply>]]>]]>
```

Monitoring and Maintaining NETCONF Sessions

Perform this task to monitor and maintain NETCONF sessions.

Note

- A minimum of four concurrent NETCONF sessions must be configured.
- A maximum of 16 concurrent NETCONF sessions can be configured.
- NETCONF does not support SSHv1.

>

SUMMARY STEPS

1. **enable**
2. **show netconf** {**counters** | **session**| **schema**}
3. **debug netconf** {**all** | **error**}
4. **clear netconf** {**counters** | **sessions**}

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.

	Command or Action	Purpose

Step 2	**show netconf** {**counters** | **session**| **schema**}
	
	Example:
	`Router# show netconf counters`
		Displays NETCONF information.

Step 3	**debug netconf** {**all** | **error**}
	
	Example:
	`Router# debug netconf error`
		Enables debugging of NETCONF sessions.

Step 4	**clear netconf** {**counters** | **sessions**}
	
	Example:
	`Router# clear netconf sessions`
		Clears NETCONF statistics counters and NETCONF sessions, and frees associated resources and locks.

Configuration Examples for NETCONF

This section provides the following configuration examples:

- Enabling SSHv2 Using a Hostname and Domain Name Example, page 23
- Enabling Secure Shell Version 2 Using RSA Keys Example, page 24
- Starting an Encrypted Session with a Remote Device Example, page 24
- Configuring NETCONF over SSHv2 Example, page 24
- Configuring NETCONF over BEEP Example, page 24
- Configuring NETCONF Network Manager Application Example, page 25
- Monitoring NETCONF Sessions Example, page 25

Enabling SSHv2 Using a Hostname and Domain Name Example

The following example shows how to configure SSHv2 using a hostname and a domain name:

```
configure terminal

hostname host1

ip domain-name domain1.com

crypto key generate rsa

ip ssh timeout 120

ip ssh version 2
```

Enabling Secure Shell Version 2 Using RSA Keys Example

The following example shows how to configure SSHv2 using RSA keys:

```
configure terminal

ip ssh rsa keypair-name sshkeys

crypto key generate rsa usage-keys label sshkeys modulus 768
ip ssh timeout 120
ip ssh version 2
```

Starting an Encrypted Session with a Remote Device Example

The following example shows how to start an encrypted SSH session with a remote networking device from any UNIX or UNIX-like device:

```
ssh -2 user@router.example.com
```

Configuring NETCONF over SSHv2 Example

The following example shows how to configure NETCONF over SSHv2:

```
configure terminal
netconf ssh acl 1
netconf lock-time 60
netconf max-sessions 5
```

Configuring NETCONF over BEEP Example

The following example shows how to configure NETCONF over BEEP:

```
Router# configure terminal
Router(config)# crypto key generate rsa general-keys

Router(ca-trustpoint)# crypto pki trustpoint my_trustpoint

Router(ca-trustpoint)# enrollment url http://10.2.3.3:80
Router(ca-trustpoint)# subject-name CN=dns_name_of_host.com
Router(ca-trustpoint)# revocation-check none
Router(ca-trustpoint)# crypto pki authenticate my_trustpoint

Router(ca-trustpoint)# crypto pki enroll my_trustpoint

Router(ca-trustpoint)# line vty 0 15

Router(ca-trustpoint)# exit
Router(config)# netconf lock-time 60

Router(config)# netconf max-sessions 16

Router(config)# netconf beep initiator host1 23 user my_user password my_password encrypt
my_trustpoint reconnect-time 60

Router(config)# netconf beep listener 23 sasl user1 encrypt my_trustpoint
```

Configuring NETCONF Network Manager Application Example

The following example shows how to configure the NETCONF Network Manager application to invoke NETCONF as an SSH subsystem:

```
Unix Side: ssh-2 -s companyname@10.1.1.1 netconf
```

As soon as the NETCONF session is established, indicate the server capabilities by sending an XML document containing a <hello>:

```
<?xml version="1.0" encoding="UTF-8"?>
    <hello>
      <capabilities>
        <capability>
            urn:ietf:params:xml:ns:netconf:base:1.0
          </capability>
          <capability>
            urn:ietf:params:ns:netconf:capability:startup:1.0
          </capability>
       </capabilities>
    <session-id>4<session-id>
</hello>]]>]]>
```

The client also responds by sending an XML document containing a <hello>:

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

Use the following XML string to enable the NETCONF network manager application to send and receive NETCONF notifications:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rpc message-id="9.0"><notification-on/>
</rpc>]]>]]>
```

Use the following XML string to stop the NETCONF network manager application from sending or receiving NETCONF notifications:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rpc message-id="9.13"><notification-off/>
</rpc>]]>]]>
```

Monitoring NETCONF Sessions Example

The following is sample output from the **show netconf counters** command:

```
Router# show netconf counters
NETCONF Counters
Connection Attempts:0: rejected:0 no-hello:0 success:0
Transactions
        total:0, success:0, errors:0
detailed errors:
        in-use 0        invalid-value 0         too-big 0
        missing-attribute 0     bad-attribute 0         unknown-attribute 0
        missing-element 0       bad-element 0   unknown-element 0
        unknown-namespace 0     access-denied 0         lock-denied 0
        resource-denied 0       rollback-failed 0       data-exists 0
        data-missing 0  operation-not-supported 0       operation-failed 0
        partial-operation 0
```

The following is sample output from the **show netconf session** command:

```
Router# show netconf session
(Current | max) sessions:   3 | 4
Operations received: 100               Operation errors: 99
Connection Requests: 5                 Authentication errors: 2   Connection Failures: 0
ACL dropped : 30
Notifications  Sent: 20
```

The output of the **show netconf schema** command describes the element structure for a NETCONF request and the resulting reply. This schema can be used to construct proper NETCONF requests and parse the resulting replies. The nodes in the schema are defined in RFC 4741. The following is sample output from the **show netconf schema**command:

```
Router# show netconf schema
New Name Space 'urn:ietf:params:xml:ns:netconf:base:1.0'
<VirtualRootTag> [0, 1] required
  <rpc-reply> [0, 1] required
    <ok> [0, 1] required
    <data> [0, 1] required
    <rpc-error> [0, 1] required
      <error-type> [0, 1] required
      <error-tag> [0, 1] required
      <error-severity> [0, 1] required
      <error-app-tag> [0, 1] required
      <error-path> [0, 1] required
      <error-message> [0, 1] required
      <error-info> [0, 1] required
        <bad-attribute> [0, 1] required
        <bad-element> [0, 1] required
        <ok-element> [0, 1] required
        <err-element> [0, 1] required
        <noop-element> [0, 1] required
        <bad-namespace> [0, 1] required
        <session-id> [0, 1] required
  <hello> [0, 1] required
    <capabilities> 1 required
      <capability> 1+ required
  <rpc> [0, 1] required
    <close-session> [0, 1] required
    <commit> [0, 1] required
      <confirmed> [0, 1] required
      <confirm-timeout> [0, 1] required
    <copy-config> [0, 1] required
      <source> 1 required
        <config> [0, 1] required
          <cli-config-data> [0, 1] required
            <cmd> 1+ required
          <cli-config-data-block> [0, 1] required
          <xml-config-data> [0, 1] required
            <Device-Configuration> [0, 1] required
              <> any subtree is allowed
        <candidate> [0, 1] required
        <running> [0, 1] required
        <startup> [0, 1] required
        <url> [0, 1] required
      <target> 1 required
        <candidate> [0, 1] required
        <running> [0, 1] required
        <startup> [0, 1] required
        <url> [0, 1] required
    <delete-config> [0, 1] required
      <target> 1 required
        <candidate> [0, 1] required
        <running> [0, 1] required
        <startup> [0, 1] required
        <url> [0, 1] required
    <discard-changes> [0, 1] required
```

```
<edit-config> [0, 1] required
  <target> 1 required
    <candidate> [0, 1] required
    <running> [0, 1] required
    <startup> [0, 1] required
    <url> [0, 1] required
  <default-operation> [0, 1] required
  <test-option> [0, 1] required
  <error-option> [0, 1] required
  <config> 1 required
    <cli-config-data> [0, 1] required
      <cmd> 1+ required
    <cli-config-data-block> [0, 1] required
    <xml-config-data> [0, 1] required
      <Device-Configuration> [0, 1] required
        <> any subtree is allowed
<get> [0, 1] required
  <filter> [0, 1] required
    <config-format-text-cmd> [0, 1] required
      <text-filter-spec> [0, 1] required
    <config-format-text-block> [0, 1] required
      <text-filter-spec> [0, 1] required
    <config-format-xml> [0, 1] required
    <oper-data-format-text-block> [0, 1] required
      <show> 1+ required
    <oper-data-format-xml> [0, 1] required
      <show> 1+ required
<get-config> [0, 1] required
  <source> 1 required
    <config> [0, 1] required
      <cli-config-data> [0, 1] required
        <cmd> 1+ required
      <cli-config-data-block> [0, 1] required
      <xml-config-data> [0, 1] required
        <Device-Configuration> [0, 1] required
          <> any subtree is allowed
    <candidate> [0, 1] required
    <running> [0, 1] required
    <startup> [0, 1] required
    <url> [0, 1] required
  <filter> [0, 1] required
    <config-format-text-cmd> [0, 1] required
      <text-filter-spec> [0, 1] required
    <config-format-text-block> [0, 1] required
      <text-filter-spec> [0, 1] required
    <config-format-xml> [0, 1] required
<kill-session> [0, 1] required
  <session-id> [0, 1] required
<lock> [0, 1] required
  <target> 1 required
    <candidate> [0, 1] required
    <running> [0, 1] required
    <startup> [0, 1] required
    <url> [0, 1] required
<unlock> [0, 1] required
  <target> 1 required
    <candidate> [0, 1] required
    <running> [0, 1] required
    <startup> [0, 1] required
    <url> [0, 1] required
<validate> [0, 1] required
  <source> 1 required
    <config> [0, 1] required
      <cli-config-data> [0, 1] required
        <cmd> 1+ required
      <cli-config-data-block> [0, 1] required
      <xml-config-data> [0, 1] required
        <Device-Configuration> [0, 1] required
          <> any subtree is allowed
    <candidate> [0, 1] required
    <running> [0, 1] required
    <startup> [0, 1] required
    <url> [0, 1] required
```

```
<notification-on> [0, 1] required
<notification-off> [0, 1] required
```

Additional References

The following sections provide references related to NETCONF.

Related Documents

Related Topic	Document Title
IP access lists	“Traffic Filtering, Firewalls, and Virus Detection” module in the *Cisco IOS XE Security Configuration Guide*
Secure Shell and Secure Shell Version 2	“Configuring Secure Shell” and “Configuring Secure Shell Version 2 Support” modules in the *Cisco IOS XE Security Configuration Guide*
NETCONF commands: complete command syntax, command mode, command history, defaults, usage guidelines, and examples	*Cisco IOS Network Management Command Reference*
IP access lists commands: complete command syntax, command mode, command history, defaults, usage guidelines, and examples. Security commands: complete command syntax, command mode, command history, defaults, usage guidelines, and examples	*Cisco IOS Security Command Reference*

Standards

Standard	Title
None	--

MIBs

MIB	MIBs Link
• No new or modified MIBs are supported by this feature and support for existing MIBs has not been modified by this feature.	To locate and download MIBs for selected platforms, Cisco IOS XE releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

RFCs

RFC	Title
RFC 2222	*Simple Authentication and Security Layer (SASL)*
RFC 2246	*The TLS Protocol Version 1.0*
RFC 3080	*The Blocks Extensible Exchange Protocol Core*
RFC 4251	*The Secure Shell (SSH) Protocol Architecture*
RFC 4252	*The Secure Shell (SSH) Authentication Protocol*
RFC 4741	NETCONF Configuration Protocol
RFC 4742	Using the NETCONF Configuration Protocol over Secure SHell (SSH)
RFC 4744	Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP)

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/techsupport

Feature Information for NETCONF

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 1 Feature Information for NETCONF

Feature Name	Releases	Feature Information
NETCONF over SSHv2	Cisco IOS XE Release 2.1	The NETCONF over SSHv2 feature enables you to perform network configurations via the Cisco command-line interface (CLI) over an encrypted transport. The NETCONF protocol defines a simple mechanism through which a network device can be managed, configuration data information can be retrieved, and new configuration data can be uploaded and manipulated. NETCONF uses an Extensible Markup Language (XML)-based data encoding for the configuration data and protocol messages. The following commands were introduced or modified by this feature: **clear netconf**, **debug netconf**, **netconf lock-time**, **netconf max-sessions**, **netconf ssh**, **show netconf**.
NETCONF Access for Configuration over BEEP	Cisco IOS XE Release 2.1	The NETCONF over BEEP feature allows you to enable either the NETCONF server or the NETCONF client to initiate a connection, thus supporting large networks of intermittently connected devices and those devices that must reverse the management connection where there are firewalls and network address translators (NATs). The following commands were introduced or modified by this feature: **netconf beep initiator**, **netconf beep listener**.

Glossary

BEEP --Blocks Extensible Exchange Protocol. A generic application protocol framework for connection-oriented, asynchronous interactions.

NETCONF --Network Configuration Protocol. A protocol that defines a simple mechanism through which a network device can be managed, configuration data information can be retrieved, and new configuration data can be uploaded and manipulated.

SASL --Simple Authentication and Security Layer. An Internet standard method for adding authentication support to connection-based protocols. SASL can be used between a security appliance and an Lightweight Directory Access Protocol (LDAP) server to secure user authentication.

SSHv2 --Secure Shell Version 2. SSH runs on top of a reliable transport layer and provides strong authentication and encryption capabilities. SSHv2 provides a means to securely access and securely execute commands on another computer over a network.

TLS --Transport Layer Security. An application-level protocol that provides for secure communication between a client and server by allowing mutual authentication, the use of hash for integrity, and encryption for privacy. TLS relies upon certificates, public keys, and private keys.

XML --Extensible Markup Language. A standard maintained by the World Wide Web Consortium (W3C) that defines a syntax that lets you create markup languages to specify information structures. Information structures define the type of information (for example, subscriber name or address), not how the information looks (bold, italic, and so on). External processes can manipulate these information structures and publish them in a variety of formats. XML allows you to define your own customized markup language.