Contents

1 Configuring NETCONF

1.1 Introduction

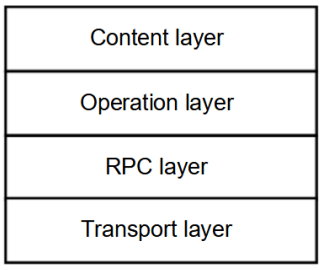
1.1.1 Overview

As network scale, complexity, and heterogeneity increase, it becomes more and more difficult for the traditional IP network management protocol Simple Network Management Protocol (SNMP) to manage the complicated network. In particular, SNMP cannot meet the requirements for configuration management and application system development of a large network. To overcome the shortcomings of SNMP, the Internet Engineering Task Force (IETF) formulates an extensible markup language (XML) based network management protocol – Network Configuration Protocol (NETCONF). NETCONF provides a programmable method to configure and manage network devices. This protocol configures parameters and obtains parameter values and statistics in XML packets and supports scalability, making network device configuration and management easy and efficient.

1.1.2 Structure of NETCONF

NETCONF supports the client/server (C/S) communication mode, in which a device runs the server of the protocol and a user runs the client of the protocol. NETCONF packets, including all configuration data and protocol messages, follow the XML format. With a hierarchical similarity to the ISO/OSI system, NETCONF consists of four layers from down to up: transport layer, remote procedure call (RPC) layer, operation layer, and content layer, as shown in Figure 1-1.

Figure 1-1Structure of NETCONF



1. Transport Layer

The transport layer provides secure transmission channels for NETCONF and supports security protocols such as Secure Shell (SSH), Simple Object Access Protocol (SOAP), and Blocks Extensible Exchange Protocol (BEEP). Now, SSH is preferred.

2. RPC Layer

The RPC layer provides a simple transmission-irrelevant mechanism, including stipulations on the elements of error reply messages. RPC defines three message types:

- Hello: The hello messages are used to complete capability set exchange when a session is created between the NETCONF client and server.
- RPC and RPC-Reply: RPC are request messages sent by the NETCONF client to the NETCONF server, and RPC-Reply are reply messages returned by the NETCONF server to the NETCONF client. The reply messages are returned only when the NETCONF server receives RPC request messages; the reply messages and the received RPC request messages must contain the same message ID.
- Notification: The notifications are sent by means of subscription.

3. Operation Layer

Based on the basic primitive operation sets applied in RPC, 9 types of basic operation methods are defined:

- Data acquisition: Get and Get-Config
- Configuration: Edit-Config, Copy-Config, and Delete-Config
- Protection for critical resource (for example, configuration file) concurrency: Lock and Unlock
- Session ending: Close-Session and Kill-Session

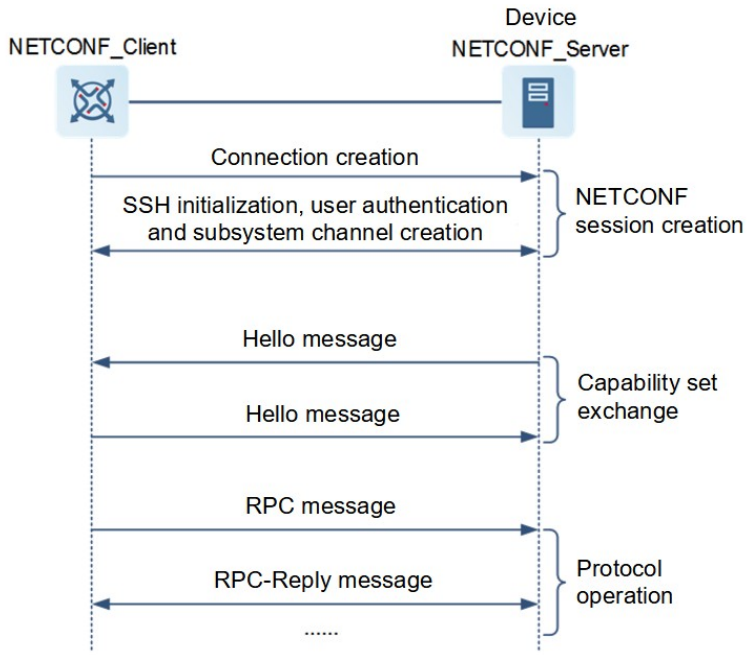
4. Content Layer

The content layer is a set of managed data objects. The content layer is the only layer that is not standardized and does not have a unified data model in NETCONF.

Note

Ruijie products use YANG for data modeling and XML for data transmission.

Figure 1-1NETCONF Session Message Interaction



1.1.3 NETCONF Session Interaction Procedure

Figure 1-1 shows the NETCONF session interaction procedure, which falls into three parts.

1. Session Creation

(1) The transport layer of NETCONF relies on SSH. After the NETCONF server on a device starts the NETCONF process, a monitoring port is created.

(2) The NETCONF server monitors port 830 and creates an SSH channel. The SSH channel is created after a series of transport algorithm negotiations (including key agreement, compression algorithm, hash algorithm, encryption algorithm, and signature algorithm) and user authentication.

(3) After a session is created on the transport layer, the NETCONF client can exchange with the NETCONF server through this session.

Note

According to NETCONF, the default SSH TCP port ID is 830, which is customized based on the particular situation.

2. Capability Set Exchange

After the session is created, the NETCONF server and client send hello messages mutually to provide respective realized capability sets and ignore capabilities that are not understandable or realizable. The basic capability (urn:ietf:params:netconf:base:1.1) must be supported by the NETCONF server and client. To be compatible with earlier versions of NETCONF, the NETCONF server and client must support the basic capability (urn:ietf:params:netconf:base:1.0) of the earlier versions of NETCONF The NETCONF server and client finally obtain the intersected capabilities to perform subsequent data operation and management.

Caution

- In addition to the exchange capabilities defined in NETCONF RFC, developers can design other capabilities based on the specifications and formats described in NETCONF RFC.
- Capability exchange messages sent by the NETCONF client to the NETCONF server do not contain the session ID node (<session-id>).

3. Protocol Operation

- Get: Obtains device status or configuration data.

The packets sent by the client follow the format below:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc message-id="xxx " xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get>
<filter type="subtree">
      Configuration data (or status data) filtering rule
</filter>
</get>
</rpc>
```

The replies returned by the server follow the format below:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc-reply message-id="xxx " xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
Obtained configuration data (or status data)
</data>
</rpc-reply>
```

If all subsets of status data on the device fail to match the filtering rule, an empty data node is returned as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc-reply message-id="Message ID "
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"/>
</rpc-reply>
```

- Get-config: Obtains configuration data based on the filtering node.

The packets sent by the client follow the format below:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc message-id="xxx" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-config>
<source>
<running/>
</source>
<filter type="subtree">
      Protocol-based filtering rule
</filter>
</get-config>
</rpc>
```

The replies returned by the server follow the format below:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc-reply message-id="xxx" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
Obtained configuration data
</data>
</rpc-reply>
```

If all subsets of configuration data on the device fail to match the filtering rule, an empty data node is returned as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc-reply message-id="xxx " xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"/>
</rpc-reply>
```

Note

The Get-config operation can obtain configuration data subsets through different subtree filtering rules but cannot obtain device status.

- Edit-config: Configures a device based on the model definitions and operation attributes.

  In the edit-config packet, there are five operation attributes which are given in the description of the operation attribute of the XML packet. These five attributes are as follows:

  - Merge: Merges the configuration data that includes this attribute in the Edit-config packets to the specified configuration file or database. If this configuration data does not exist, create this configuration data.
  - Replace: Replaces the configuration data node in the specified configuration file or database with the configuration data that includes this attribute in the Edit-config packets. If this configuration data does not exist, create this configuration data based on the delivered content.
  - Create: Creates configuration data that includes this attribute in the Edit-config packets in the specified configuration file or database. If the configuration data does not exist, deliver content to create this configuration data. If the configuration data exists, a reply, corresponding to the RPC-error packet, is returned and contains the value of error-tag "data-exists".
  - Delete: Deletes configuration data that includes this attribute in the Edit-config packets from the specified configuration file or database. If the configuration data does not exist, a reply, corresponding to the RPC-error packet, is returned and contains the value of error-tag "data-missing". If the configuration data exists, corresponding configuration is deleted.
  - Remove: Removes configuration data that includes this attribute in the Edit-config packets from the specified configuration file or database. If the configuration data does not exist, this operation is ignored and OK is returned. If the configuration data exists, corresponding configuration is removed.

  The packets sent by the client follow the format below:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc message-id="xxx " xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target> <running/> </target>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        Configuration data
</config>
</edit-config>
</rpc>
```

  The replies returned by the server follow the format below:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc-reply message-id="Message ID "
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
```

  The Edit-config packet includes the Error-option node. The values of the Error-option node are of the enumerated type, including:

  - **Continue-on-error**: During an Edit-config operation, if a configuration error occurs, the current error node is recorded and configuration continues. But an error message is returned finally (In the case of any configuration error, the final reply message is RPC-error).
  - **Stop-on-error**: This is the default value of the Error-option. During an Edit-config operation, if a first configuration error occurs, the Edit-config operation stops immediately. The data configured before this

error occurs takes effect.

The packets generally follow the format below:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc message-id="xxx" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target> <running/> </target>
Acton option when a configuration error occurs
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
       Configuration data
</config>
</edit-config>
</rpc>
```

Specification

Currently, Ruijie devices do not support the Replace operation. If this operation is delivered, it is processed as a Merge operation.

Note

- If the Edit-config packet does not include the error-option node, the default value of the error-option node is stop-on-error. Once a configuration error occurs on a node, the remaining configuration in the Edit-config packet stops immediately and an RPC-error is returned.
- If the Edit-config packet does not include the test-option node, the default value of the test-option node is **test-then-set**.
- If the Edit-config packet does not include the default-operation node, the default value of the default-operation node is **merge**.

- Copy-config: Copies the configuration file. For example, candidate configuration is copied to the configuration file, startup configuration is copied to the running configuration file, and running configuration is written into the startup configuration file. The copies can be successful if the target files support the write capability.

The packets sent by the client follow the format below:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc message-id="xxx"  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <copy-config>
    <target>
      <startup/>
    </target>
    <source>
      <running/>
    </source>
  </copy-config>
</rpc>
```

The replies returned by the server follow the format below:

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

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

- Delete-config: Deletes the configuration file. It is not allowed to delete the running file.

The packets sent by the client follow the format below:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc message-id="xxx " xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <delete-config>
   <target>
     <startup/>
   </target>
 </delete-config>
</rpc>
```

The replies returned by the server follow the format below:

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

Specification

Now, only the startup configuration can be synchronized to the running configuration file.

- Lock: Locks the configuration file. The configuration file can be accessed or modified through the current client, but cannot be accessed or modified by other clients or non-NETCONF clients (for example, SNMP or CLI). The packets sent by the client follow the format below:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc message-id="xxx " xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <lock>
   <target>
     <running/>
   </target>
 </lock>
</rpc>
```

The replies returned by the server follow the format below:

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

Note

According to RFC6241, the Lock operation is used to lock the configuration database (configuration files) and prevent multiple sources (for example, CLI, SNMP, and NETCONF sessions) from concurrently revising the configuration files. This avoids introducing other irrelevant configuration revisions. The device simplifies this operation, and can only prevent concurrent modifications (running configuration) from multiple NETCONF.

- Unlock: Unlocks the configuration database (configuration file, which indicates the running configuration file in a device). This operation is opposite to the Lock operation.

The packets sent by the client follow the format below:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc message-id="xxx "  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <unlock>
   <target>
     <running/>
   </target>
 </unlock>
</rpc>
```

The replies returned by the server follow the format below:

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

- Close-session: Closes the current session, including resource and lock release and disconnection.

The packets sent by the client follow the format below:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc message-id="xxx" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <close-session/>
</rpc>
```

The replies returned by the server follow the format below:

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

Note

Before the Close-session operation closes the current session, ensure that proceeding business has been completed and no new business request is received.

- Kill-session: Forcibly closes the current session, including resource and lock release and disconnection.

Caution

- Before using the Kill-session operation to forcibly close the current session, ensure that proceeding business has been stopped and unfinished business is rolled back to original state.
- The Kill-session operation cannot close the current session.

1.1.4 Authentication Mechanism

The NETCONF authentication mechanism is used to manage the NETCONF operations of specific users and the access to NETCONF resources, allowing the users to perform operations and access data nodes stipulated by NETCONF.

1. Access Permissions Supported by the NETCONF Authentication Mechanism

The NETCONF authentication function is not configured by default. Therefore, legal users have all NETCONF permissions. Access permissions supported by the NETCONF authentication mechanism are as follows:

- Create: Permits or denies new data nodes to be added.
- Read: Permits or denies data nodes to be read.
- Update: Permits or denies data nodes to be updated.
- Delete: Permits or denies data nodes to be deleted.
- Execute: Permits or denies all protocol operations.

2. Content Supported by the NETCONF Authentication Mechanism

The NETCONF authentication function supports RPC method authentication and data node authentication.

- RPC method authentication

  RPC method authentication of NETCONF is used to control permissions of the operation methods stipulated by NETCONF. Table 1-1 lists the RPC methods and default permissions.

Table 1-1RPC Methods and Default Permissions

RPC Method	Function	Default Permission (R: read; W: write; X: execute)
get	Obtains data.	X→R
get-config	Obtains configuration.	X→R
edit-config	Edits configuration.	X→W
copy-config	Replaces the target configuration with the source configuration.	X
delete-config	Deletes configuration.	X
validate	Performs syntax check.	X
commit	Validates the candidate configuration and runs the configuration.	X
cancel-commit	Cancels the confirmed-commit operation.	X
discard-changes	Discards candidate configuration that has not taken effect.	X
lock/unlock	Locks or unlocks configuration.	X

RPC Method	Function	Default Permission (R: read; W: write; X: execute)
close-session	Closes the current session.	X
kill-session	Closes other sessions.	X
get-schema	Obtains the YANG file.	X

Note

- All RPC methods require the execution permission by default. The Get, Get-config, and Edit-config methods require the check of read/write permission for the data node.
- The Close-session method is permitted by default and cannot be denied.
- The Delete-config and Kill-session methods are denied by default, and can be configured as Permitted. Other RPC methods are permitted by default.
- The Lock and Unlock methods must be paired. If the Lock permission is configured, the Unlock permission is available as well.

- Data node authentication

  Data node authentication of NETCONF is used to control permissions of NETCONF data nodes. It can control the permissions of all modules, single module, and specific data nodes.

  - If only the slash (/) is configured, the authentication of NETCONF can control the permissions of the data nodes of all modules.
  - If a module name is configured, for example, /Orion-snmp:snmp, the authentication of NETCONF can control the permissions of the data nodes of the single module.
  - In other configuration conditions, permission control is applied to the data nodes of specific modules.

1.1.5 Protocols and Standards

- RFC4741: NETCONF Configuration Protocol
- RFC4742: Using the NETCONF Configuration Protocol over Secure Shell (SSH)
- RFC4743: Using NETCONF over the Simple Object Access Protocol (SOAP)
- RFC4744: Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP)
- RFC5277: NETCONF Event Notifications
- RFC5381: Experience of Implementing NETCONF over SOAP
- RFC5539: NETCONF Over Transport Layer Security (TLS)
- RFC5717: Partial Lock RPC for NETCONF
- RFC6022: NETCONF Monitoring Schema
- RFC6241: Network Configuration Protocol
- RFC6242: Using the Network Configuration Protocol over Secure Shell
- RFC6243: With-defaults capability for NETCONF
- RFC6470: NETCONF Notification Events

- RFC6536: NETCONF Access Control Model (NACM)

Note

RFC4741 and RFC4742 have been replaced by RFC6241 and RFC6242 respectively.

1.2 Configuration Task Summary

NETCONF configuration includes the following tasks:

(1) Configuring Communication Between the NETCONF Server and Client

1.3 Configuring Communication Between the NETCONF Server and Client

1.3.1 Overview

After the NETCONF function is enabled on the NETCONF server, the NETCONF server can communicate with the NETCONF client to manage network devices.

1.3.2 Restrictions and Guidelines

- The **netconf yang multi-revision** command must be configured before the capability packet (Hello packet) of the NETCONF server is advertised.
- The **no netconf yang multi-revision** command must be configured before the capability packet (Hello packet) of the NETCONF server is advertised and one YANG module can advertise only its current latest version in the capability notification packet.
- In strict check mode, some XML packets that can be delivered in lightweight check mode may be intercepted.

1.3.3 Prerequisites

The NETCONF protocol is based on the SSH protocol. Before the NETCONF protocol is enabled, the SSH protocol must be configured. For the configuration procedure, see "Configuring SSH".

1.3.4 Procedure

(1) Enter the privileged EXEC mode.

enable

(2) Enter the global configuration mode.

configure terminal

(3) Configure the attribute parameters of the NETCONF server. The configuration tasks below are optional. Select tasks as required.

- (Optional) Enable the NETCONF service.

  netconf enable

  The NETCONF function is enabled by default.

- (Optional) Configure the maximum number of session connections supported by NETCONF.

netconf max-sessions *max-sessions-numbers*

The maximum number of session connections supported by NETCONF is **5** by default.

- (Optional) Configure the timeout time of the Edit-config operation in a NETCONF session.

  netconf timeout *timeout*

  The default timeout time of the Edit-config operation in a NETCONF session is **120** seconds.

- (Optional) Enable the YANG module multi-version notification function.

  netconf yang multi-revision

  The YANG module multi-version notification function of NETCONF is enabled by default.

- (Optional) Configure the NETCONF capabilities.

  netconf capability{ **candidate** | **rollback** | **validate** }

  The NETCONF-related capabilities are not configured by default.

- (Optional) Configure the NETCONF check mode.

  netconf calibration-mode { **lightweight** | **strict** }

  The default NETCONF check mode is lightweight check.

- (Optional) Enable the NETCONF logging function.

  netconf log { **capability** | **edit** | **get** | **session** }

  The NETCONF logging function is disabled by default.

- (Optional) Configure the port ID of the NETCONF server.

  netconf port *port-number*

  The default port ID of the NETCONF server is 830.

1.4 Monitoring

Run the **show** command to check the running status of a configured function to verify the configuration effect.

Table 1-1Monitoring

Command	Purpose
show netconf authorization user-name *name* { **rule-list** [**detail**] | **user-group** }	Displays a rule list that users are associated with, list details, and user group information.
show netconf session	Displays all session information of NETCONF.
show netconf statistics	Displays global statistics of NETCONF.
show netconf yang file	Displays all YANG files supported currently.
show netconf yang node-path	Displays all node paths supported currently.
show netconf yang tree	Displays all YANG model trees supported currently.

1.5 Configuration Examples

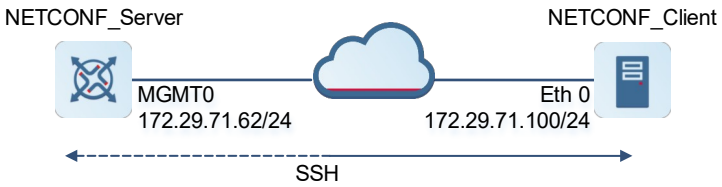
1.5.1 Configuring NETCONF

1. Requirements

NETCONF can be used as a network management tool to meet the high security and scalability requirements for network device management. NETCONF is based on SSH. As a security protocol of the application layer, SSH ensures security of NETCONF. As shown in Figure 1-1, the NETCONF network management software is used to manage and monitor network devices.

2. Topology

Figure 1-1NETCONF Topology



3. Notes

- Configure an IP address for the network management interface of the NETCONF server to ensure that the NETCONF server and client are mutually reachable via L3 routes.
- Enable the SSH function on the NETCONF server.
- Configure NETCONF related parameters on the NETCONF server to meet requirements of actual scenarios.
- Connect the NETCONF server to the NETCONF client through SSH to manage and monitor network devices.

4. Procedure

Configure an IP address for the network management interface of the NETCONF server.

```
Hostname> enable
Hostname# configure terminal
Hostname(config)# interface mgmt 0
Hostname(config-if-Mgmt 0)# ip address 172.29.71.62 255.255.255.0
Hostname(config-if-Mgmt 0)# gateway 172.29.71.1
```

Enable the SSH function and generate a local key. For more information about the rule of selecting a key type, see the Usage Guidelines in the **crypto key generate** command in "SSH Commands".

```
Hostname> enable
Hostname# configure terminal
Hostname(config)# enable service ssh-server
Hostname(config)# crypto key generate rsa
% You already have RSA keys.
% Do you really want to replace them? [yes/no]:y
```

```
Choose the size of the rsa key modulus in the range of 512 to 2048
and the size of the dsa key modulus in the range of 360 to 2048 for your
Signature Keys. Choosing a key modulus greater than 512 may take
a few minutes.
Choose the size of the ecc key modulus from (256, 384, 521)

How many bits in the modulus [1024]:2048
% Generating 2048 bit RSA1 keys ...[ok]
% Generating 2048 bit RSA keys ...[ok]
Hostname(config)#
```

Create an SSH user with the username **netconf** and set the password to netconf_1234.

```
Hostname(config)# username netconf privilege 15 password netconf_1234
```

Caution

To ensure security, you are advised to regularly modify the username and password.

Configure local user authentication of a VTY line.

```
Hostname(config)# line vty 0 35
Hostname(config-line)# login local
```

Enable the NETCONF service on the NETCONF server.

```
Hostname(config)# netconf enable
```

Set the maximum number of session connections supported by the NETCONF server to 6.

```
Hostname(config)# netconf max-sessions 6
```

Set the timeout time of the Edit-config operation in a NETCONF session to 100 seconds on the NETCONF server.

```
Hostname(config)# netconf timeout 100
```

Enable the YANG module multi-version notification function on the NETCONF server.

```
Hostname(config)# netconf yang multi-revision
```

Enable the candidate capability on the NETCONF server.

```
Hostname(config)# netconf capability candidate
```

Configure the check mode as lightweight check on the NETCONF server.

```
Hostname(config)# netconf calibration lightweight
```

Enable the syslog function related to the NETCONF sessions on the NETCONF server.

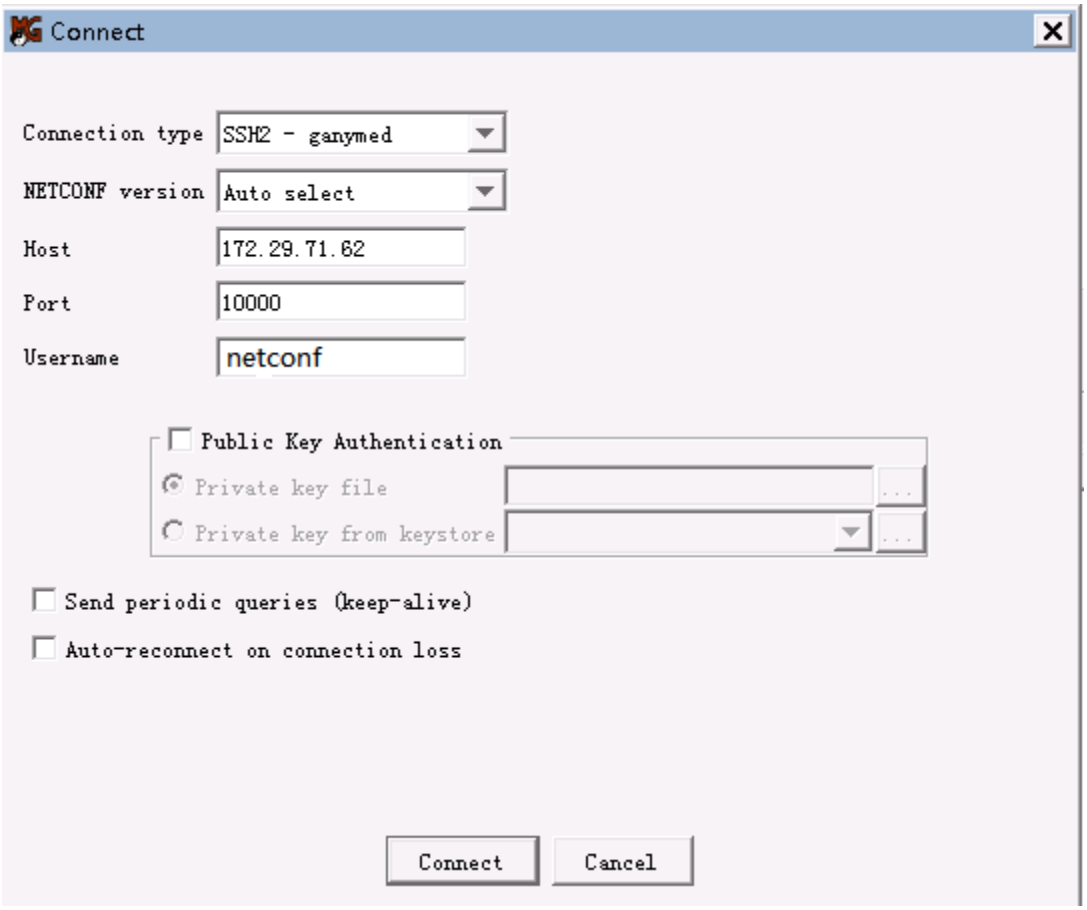
```
Hostname(config)# netconf log session
```

Set the port ID of the NETCONF service to 5000 on the NETCONF server.

```
Hostname(config)# netconf port 5000
```

Log in to the NETCONF server through the SSH protocol by using the NETCONF client software and manage and monitor network devices, as shown in Figure 1-1.

Figure 1-1Device Login Example Through the NETCONF Client Software



5. Verification

Run the **ping** command to check whether the NETCONF server and client are mutually reachable via L3 routes.

```
Hostname# ping oob 172.29.71.100
Sending 5, 100-byte ICMP Echoes to 172.29.71.100, timeout is 2 seconds:
  < press Ctrl+C to break >
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/3 ms.
Hostname#
```

Run the **show service** command to check whether the SSH service is enabled.

```
Hostname# show service
snmp-agent    : enabled
ssh-server    : enabled
telnet-server : enabled
```

Run the **show netconf session** command to check whether the NETCONF server is connected to the NETCONF client software through the SSH protocol.

```
Hostname# show netconf session
************************session information************************
```

```
                          Session count: 1
******************************************************************
Session ID              : 20
Session version         : 1.1
Session transport       : netconf-ssh
Session login IP        : 172.29.69.21
Session login time      : 2020-12-18T08:26:30Z
Session in rpcs         : 0
Session in bad rpcs     : 0
Session out rpc errors  : 0
Session out notification: 0
Session out rpcs        : 0
Session out send fail   : 0
Session get             : 0
Session get config      : 0
Session edit config     : 0
Session copy config     : 0
Session delete config   : 0
Session close session   : 0
Session unsupport       : 0
Session lock_or_unlock  : 0
==================================================================
```

6. Configuration Files

NETCONF_Server configuration file

```
hostname Hostname
!
username netconf privilege 15 password netconf_1234
!
netconf log session
netconf capability candidate
netconf port 5000
netconf max-sessions 6
netconf timeout 100
!
enable service ssh-server
!
interface Mgmt 0
 ip address 172.29.71.62 255.255.255.0
 gateway 172.29.71.1
!
line vty 0 35
 login local
!
end
```