gRPC Applications and Configuration

gRPC operations

gRPC operations are a set of remote procedure calls that enable clients to interact with Cisco IOS XR devices for configuration and operational data retrieval.

- They support configuration retrieval and modification.
- They provide access to operational and model data.
- They allow CLI-based and structured output retrieval.

These operations are essential for automating and managing network devices programmatically using gRPC clients.

Manageability service gRPC operations

This table defines the manageability service gRPC operations for Cisco IOS XR.

gRPC Operation	Description
GetConfig	Retrieves the configuration from the router.
GetModels	Gets the supported Yang models on the router
MergeConfig	Merges the input config with the existing device configuration.
DeleteConfig	Deletes one or more subtrees or leaves of configuration.
ReplaceConfig	Replaces part of the existing configuration with the input configuration.
CommitReplace	Replaces all existing configuration with the new configuration provided.

gRPC Operation	Description
GetOper	Retrieves operational data.
CliConfig	Invokes the input CLI configuration.
ShowCmdTextOutput	Returns the output of a show command in the text form
ShowCmdJSONOutput	Returns the output of a show command in JSON form.

gRPC operation to Get configuration

The gRPC example shows how a gRPC GetConfig request works for feature.

The client initiates a message to get the current configuration of running on the router. The router responds with the current configuration.

gRPC request (Client to Router)	gRPC response (Router to Client)

rpc GetConfig
{
"Cisco-IOS-XR-cdp-cfg:cdp": [
"cdp": "running-configuration"
]
}
 	
rpc GetConfig
{
"Cisco-IOS-XR-ethernet-lldp-cfg:lldp": [
"lldp": "running-configuration"
]
}
	{
	"Cisco-IOS-XR-cdp-cfg:cdp": {
	"timer": 50,
	"enable": true,
	"log-adjacency": [
	null
	],
	"hold-time": 180,
	"advertise-v1-only": [
	null
	]
	}
	}
	
	{
	"Cisco-IOS-XR-ethernet-lldp-cfg:lldp": {
	"timer": 60,
	"enable": true,
	"reinit": 3,
	"holdtime": 150
	}
	}

gRPC authentication modes

A gRPC authentication mode is a security mechanism for gRPC communication that

- provides different methods to verify the identity of clients and servers,
- supports both metadata-based and certificate-based approaches for authentication, and
- enables compliance with varying security requirements through configurable settings such as TLS, Mutual TLS, and non-TLS options.

This section details the authentication modes supported by gRPC to secure communication and ensure authorized access to services.

gRPC supports multiple authentication modes to secure communication between clients and servers. These modes ensure that only authorized entities can access gRPC services such as gNOI, gRIBI, and P4RT. Upon receiving a gRPC request, the device authenticates the user and performs authorization checks.

The following table lists the authentication types and their configuration requirements:

Table 1: gRPC authentication modes and configuration requirements

Type	Authentication Method	Authorization Method	Configuration Requirement	Requirement From Client
Metadata with TLS	username, password	username	**grpc**	username, password, and CA
Metadata without TLS	username, password	username	**grpc no-tls**	username, password
Metadata with Mutual TLS	username, password	username	**grpc tls-mutual**	username, password, client certificate, client key, and CA
Certificate based Authentication	client certificate's common name field	username from client certificate's common name field	**grpc tls-mutual** and **grpc certificate authentication**	client certificate, client key, and CA

Certificate-based authentication

In Extensible Manageability Services (EMS) gRPC, certificates play a vital role in ensuring secure and authenticated communication. The EMS gRPC utilizes these certificates for authentication:

```
/misc/config/grpc/ems.pem
                   /misc/config/grpc/ems.key
                   /misc/config/grpc/ca.cert
```

Note For clients to use the certificates, ensure to copy the certificates from **/misc/config/grpc/**

Generation of certificates

These certificates are typically generated using a Certificate Authority (CA) by the device. The EMS certificates, including the server certificate (**ems.pem**), public key (**ems.key**), and CA certificate (**ca.cert**), are generated with specific parameters like the common name **ems.cisco.com** to uniquely identify the EMS server and placed in the **/misc/config/grpc/** location.

The default certificates that are generated by the server are Server-only TLS certificates and by using these certificates you can authenticate the identity of the server.

Usage of certificates

These certificates are used for enabling secure communication through Transport Layer Security (TLS) between gRPC clients and the EMS server. The client should use **ems.pem** and **ca.cert** to initiate the TLS authentication.

To update the certificates, ensure to copy the new certificates that have been generated earlier to the location and restart the server.

Custom certificates

If you want to use your own certificates for EMS gRPC communication, then you can follow a workflow to generate custom certificates with the required parameters and then configure the EMS server to use these custom certificates. This process involves replacing the default EMS certificates with the custom ones and ensuring that the gRPC clients also trust the custom CA certificate. For more information on how to customize the **common-name**, see *Certificate Common-Name For Dial-in Using gRPC Protocol*.

Configure authentication for gRPC services

This task explains how to configure different types of authentication for gRPC services, including TLS and AAA-based authentication.

Before you begin

Ensure that the router supports gRPC and that you have access to the CLI in configuration mode. TLS and AAA configurations must be available if required by the authentication method.

Procedure

Step 1 Configure your preferred authentication method:

- Configure authentication using metadata with TLS

```
Router#config
Router(config)#grpc
Router(config-grpc)#commit
```

- Configure authentication using metadata without TLS

```
Router#config
Router(config)#grpc
Router(config-grpc)#no-tls
Router(config-grpc)#commit
```

- Configure authentication using metadata with mutual TLS

```
Router#config
Router(config)#grpc
Router(config-grpc)#tls-mutual
Router(config-grpc)#commit
```

- Configure certificate-based authentication

```
Router(config)#grpc
Router(config-grpc)#tls-mutual
Router(config-grpc)#certificate-authentication
Router(config-grpc)#commit
```

Step 2 Verify the configuration.

Example:

```
Router# show grpc
Tue Jul 30 09:54:23.001 UTC

Server name                             : DEFAULT
Address family                          : dual
Port                                    : 57400

Service ports
```

```
  gNMI                                : none
  P4RT                                : none
  gRIBI                               : none

DSCP                                  : Default
TTL                                   : 64
VRF                                   : global-vrf
Server                                : enabled
TLS                                   : enabled
TLS mutual                            : disabled
Trustpoint                            : none
Certificate Authentication            : disabled
Certificate common name               : ems.cisco.com
TLS v1.0                              : enabled
Maximum requests                      : 128
Maximum requests per user             : 10
Maximum streams                       : 32
Maximum streams per user              : 32
Maximum concurrent streams            : 32
Memory limit (MB)                     : 1024
Keepalive time                        : 30
Keepalive timeout                     : 20
Keepalive enforcement minimum time    : 300



TLS cipher suites
  Default                             : none
  Default TLS1.3                      : aes_128_gcm_sha256
                                      : aes_256_gcm_sha384
                                      : chacha20_poly1305_sha256

  Enable                              : none
  Disable                             : none

  Operational enable                  : ecdhe-rsa-chacha20-poly1305
                                      : ecdhe-ecdsa-chacha20-poly1305
                                      : ecdhe-rsa-aes128-gcm-sha256
                                      : ecdhe-ecdsa-aes128-gcm-sha256
                                      : ecdhe-rsa-aes256-gcm-sha384
                                      : ecdhe-ecdsa-aes256-gcm-sha384
                                      : ecdhe-rsa-aes128-sha
                                      : ecdhe-ecdsa-aes128-sha
                                      : ecdhe-rsa-aes256-sha
                                      : ecdhe-ecdsa-aes256-sha
                                      : aes128-gcm-sha256
                                      : aes256-gcm-sha384
                                      : aes128-sha
                                      : aes256-sha
  Operational disable                 : none
Listen addresses                      : ANY
```

The gRPC service is configured with the selected authentication method and is ready to accept secure client connections.

What to do next

Verify the gRPC connection and monitor authentication logs to ensure proper access control.

gRPC servers with TLS version 1.3 support

gRPC servers with TLS version 1.3 support are network security solutions that

- provide end-to-end encrypted communication between clients and servers,
- use modern cryptographic protocols for stronger security and performance, and
- allow administrators to configure minimum and maximum TLS versions for compliance and interoperability.

Table 2: Feature History Table

Feature Name	Release Information	Description
gRPC Server TLS Version 1.3 Support	Release 24.4.1	You can now enhance the security of your network connections with stronger protection against vulnerabilities by enabling TLS 1.3 support over gRPC services. This update improves performance with faster connection times and reduced latency by reducing the number of round trips required to establish a connection and removing outdated ciphers. Additionally, it complies with internal security mandates, providing a more robust and future-proof solution for your network management needs. Previously, gRPC server supported TLS version 1.2. The feature introduces these changes: **CLI:** • **tls-min-version** • **tls-max-version**

Security benefits of TLS 1.3

The gRPC Remote Procedure Calls (gRPC) server Transport Layer Security (TLS) version 1.3 support is a security feature that:

- Provides end-to-end communications security over networks
- Prevents unauthorized access and eavesdropping
- Protects against tampering and message forgery

The TLS private key is encrypted before being stored on the disk. For more details on SSL or TLS version certificates, keys, and communication parameters, see Manage certificates using Certz.proto.

Guidelines and limitations for TLS configuration

TLS version configuration limitations

- Ensure that the `tls-min-version` value is not greater than the `tls-max-version` value.

- Starting in Release 2.4.4.1, the `tlsv1-disable` command is deprecated. Avoid using this command in new configurations.
- If you use the `tlsv1-disable` command, do not use the `tls-min-version` or `tls-max-version` commands.
- If you use the `tls-min-version` and `tls-max-version` commands, do not use the `tlsv1-disable` command.

Best practice for disabling TLS 1.0

To disable TLS version 1.0, set the `tlsv1-disable` command. Alternatively, you can set the `tls-min-version` to a value greater than 1.0.

Configure gRPC TLS version

Configuring gRPC TLS version enables you to control which TLS protocol versions are permitted for secure gRPC communication between the router and clients. This can be important for maintaining compatibility and achieving desired security standards.

Before you begin

- Verify that gRPC is enabled on the router.
- Determine which TLS versions (1.0, 1.1, 1.2, or 1.3) your environment and clients require.

Procedure

Step 1 Configure gRPC TLS minimum, maximum, or both versions.

Example:

- Configure gRPC TLS maximum version.

```
Router# config
                                Router(config)# grpc
                                Router(config-grpc)# tls-max-version 1.2
                                Router(config-grpc)# commit
```

tls-max-version can be 1.0, 1.1, 1.2, or 1.3. The default maximum version for TLS is 1.3.

Step 2 Verify the gRPC TLS minimum and maximum versions.

Example:

```
Router# show grpc
                        Thu Aug 29 00:49:24.428 UTC

                        Server name                             : DEFAULT
                        Address family                          : dual
                        Port                                    : 57400

                        Service ports
                        gNMI                                  : none
                        P4RT                                  : none
                        gRIBI                                 : none

                        DSCP                                    : Default
```

```
TTL                                     : 64
VRF                                     : global-vrf
Server                                  : disabled (Unknown)
TLS                                     : enabled
TLS mutual                              : disabled
Trustpoint                              : none
Certificate Authentication              : disabled
Certificate common name                 : ems.cisco.com
TLS v1.0                                : enabled
Maximum requests                        : 128
Maximum requests per user               : 10
Maximum streams                         : 32
Maximum streams per user                : 32
Maximum concurrent streams              : 32
Memory limit (MB)                       : 1024
Keepalive time                          : 30
Keepalive timeout                       : 20
Keepalive enforcement minimum time      : 300
TLS Minimum Version                     : TLS 1.0
    TLS Maximum Version                    : TLS 1.2

    TLS cipher suites
    Default                               : none
    Default TLS1.3                        : aes_128_gcm_sha256
    : aes_256_gcm_sha384
    : chacha20_poly1305_sha256

Enable                                  : none
Disable                                 : none

Operational enable                      : ecdhe-rsa-chacha20-poly1305
: ecdhe-ecdsa-chacha20-poly1305
: ecdhe-rsa-aes128-gcm-sha256
: ecdhe-ecdsa-aes128-gcm-sha256
: ecdhe-rsa-aes256-gcm-sha384
: ecdhe-ecdsa-aes256-gcm-sha384
: ecdhe-rsa-aes128-sha
: ecdhe-ecdsa-aes128-sha
: ecdhe-rsa-aes256-sha
: ecdhe-ecdsa-aes256-sha
: aes128-gcm-sha256
: aes256-gcm-sha384
: aes128-sha
: aes256-sha
Operational disable                     : none
Listen addresses                          : ANY
```

The TLS 1.3 cipher suites are not configurable, they are either fixed or static.

After completing this task, the router will use the specified TLS version for gRPC communication.

Example

For example, enabling only TLS 1.3 ensures that gRPC connections use the most secure protocol version supported by the router.

What to do next

After configuring the TLS version, verify the gRPC server status and test connectivity using a gRPC client to ensure compatibility.

SPIFFE ID-based authentication and authorization services for gRPC services

A SPIFFE ID (Secure Production Identity Framework for Everyone) based authentication and authorization service is a standardized framework that:

- enables secure identification and authorization of services communicating over gRPC,
- provides interoperability for authentication and access control across diverse and distributed environments, and
- leverages SPIFFE IDs and Verifiable Identity Documents (SVIDs) to enforce mutual TLS (mTLS) and authorization policies.

SPIFFE ID-Based authentication and authorization services for gRPC services uses SPIFFE IDs and SPIFFE Verifiable Identity Documents (SVIDs) to authenticate and authorize gRPC traffic. This is especially useful in distributed systems where workloads span multiple platforms.

- **Authentication:** Performed via mutual TLS (mTLS) using SVIDs
- **Authorization:** Based on mapping SPIFFE IDs to XR usernames
- **Identity format:** SVIDs can be encoded as X.509 certificates or JWTs
- **Integration:** Enables EMS and gRPC services to enforce access control

Workflow for SPIFFE ID-based authentication and authorization for gRPC services

Mapping initialization and configuration

1. The EMS starts searching for the *spiffe-user-map.json* file at the location `/misc/config/grpc/gnsi/credentialz/spiffe-user-map.json`.
2. If the file exists, it is parsed, and the mapping is stored globally in the `aaa/auth package`.
3. If the file does not exist or parsing is unsuccessful, the mapping will be empty.
4. The EMS registers with the configuration manager to receive updates for the `aaa` configuration.

Authentication and authorization Flow

1. When processing requests in the Authentication interceptor, the spiffe-user mapping API checks for the SPIFFE ID mapping.
2. If the mapping exists, the API responds with the corresponding username.
3. If the mapping does not exist but the `aaa` configuration exists, the API responds with the configured username.
4. If neither the mapping nor the `aaa` configuration is present, the API responds with an empty string.
5. Upon a client connecting to the server, the server interceptor extracts the SPIFFE ID from the client's certificate and uses the mapping stored in the `aaa/auth package` to find the corresponding username.
6. The username identifies it and then includes the metadata into the context.

7. gRPC services that require XR Authorization will later verify the access rights for the username identified in the previous step when handling the request.
8. If the mapping is unsuccessful, the request is passed to the relevant service, such as gNMI, which then decides whether to grant or deny access based on its authorization requirements.

Authenticate and authorize gRPC service requests using the SPIFFE standard

This task describes how to authenticate and authorize gRPC service requests using the SPIFFE standard by mapping SPIFFE IDs to usernames and evaluating authorization policies.

Before you begin

Before authenticating and authorizing gRPC service requests using the SPIFFE standard, ensure the following prerequisites are met:

- Enable mutual TLS authentication with the `tls-mutual` command.
- Enable certificate authentication with the `certificate-authentication` command to facilitate SPIFFE ID recognition. For more information, see Configure authentication for gRPC services, on page 4.
- Configure the gNSI Authz policy by setting the principal to the SPIFFE-ID for service-level authorization (gNSI AuthZ).

After establishing the connection, the gRPC server extracts the SPIFFE ID from the client's certificate.

To authenticate and authorize gRPC service requests using the SPIFFE standard, follow these steps:

Procedure

Step 1 Configure the username in the system.

Example:

```
Router#show running-config aaa
                        Thu Oct 12 11:43:15.771 UTC
                        username cisco
                        group root-lr
                        group cisco-support
                        password 7 104D000A061843595F
                        !
```

Step 2 Map the SPIFFE ID to a username using the **aaa map-to username** command. This command assigns a default username to any SPIFFE ID.

```
Router(config)#aaa map-to username cisco spiffe-id any
                            Router(config)#commit
```

Note
Each SPIFFE ID supports only one username.

Step 3 Evaluate the client's SPIFFE ID against the service-level authorization policy (gNSI AuthZ).

The gRPC service request is authenticated and authorized using the SPIFFE ID mapped to a system username and evaluated against the gNSI AuthZ policy.

Example

For example, after mapping the SPIFFE ID to the username `cisco`, the system uses this identity to authorize access based on the configured gNSI AuthZ policy.

What to do next

After completing this task:

- Monitor gRPC logs to verify successful authentication and authorization events using SPIFFE IDs.

Certificate common-name for dial-in using gRPC protocol

A certificate common-name for dial-in using gRPC protocol is a security configuration that:

- allows the router to generate certificates with a user-defined common-name,
- enables gRPC clients to verify the server identity using a matching hostname, and
- prevents certificate verification failures caused by fixed or mismatched common-names.

This feature enhances TLS authentication flexibility and supports secure, hostname-based validation for gRPC dial-in sessions.

Table 3: Feature History Table

Feature Name	Release Information	Description
Certificate common-name for dial-in using gRPC protocol	Release 24.1.1	You can now specify a **common-name** for the certificate generated by the router while using gRPC dial-in. Earlier, the **common-name** in the certificate was fixed as *ems.cisco.com* and was not configurable. Using a specified **common-name** avoids potential certification failures where you may specify a hostname different from the fixed common name to connect to the router. The feature introduces these changes: **CLI:** • **grpc certificate common-name** **YANG Data Model:** • New XPath for `Cisco-IOS-XR-um-grpc-cfg.yang` • New XPath for `Cisco-IOS-XR-man-ems-cfg` (see GitHub, YANG Data Models Navigator)

gRPC dial-in certificate common-name configuration

When using gRPC dial-in on Cisco IOS-XR routers, the `common-name` associated with the certificate generated by the router was previously fixed as *ems.cisco.com*, causing failures during certificate verification if a different hostname was used. From Cisco IOS XR 24.11, you can now specify the common-name in the certificate using the `grpc certificate common-name` command, allowing gRPC clients to more flexibly and securely verify the server's domain name.

Configure certificate common name for dial-in

Configure a common name to be used in EMSD certificates for gRPC dial-in.

Before you begin

Before you begin, ensure the following:

- The router is running with the correct OS image.

- gRPC is enabled and properly configured on the device.

Procedure

Step 1 Configure a common name.

Example:

```
Router#config
                        Router(config)#grpc
                        Router(config-grpc)#certificate common-name cisco.com
                        Router(config-grpc)#commit
```

Use the show command to verify the common name:

```
Router#show grpc
                        Certificate common name                : cisco.com
```

Note
For the above configuration to be successful, ensure to regenerate the certificate so that the new EMSD certificates include the configured common name.

To **regenerate** the self-signed certificate, perform the following steps.

Step 2 Remove the certificates: `/misc/config/grpc/ems.pem`, `/misc/config/grpc/ems.key`, and `/misc/config/grpc/ca.cert` from `/misc/config/grpc` file.

Example:

```
Router#run ls -ltr /misc/config/grpc/

                        total 16
                        drwx------. 2 root root 4096 Feb 14 09:17 dialout
                        -rw-rw-rw-. 1 root root 1505 Feb 14 10:58 ems.pem
                        -rw-------. 1 root root 1675 Feb 14 10:58 ems.key
                        -rw-r--r--. 1 root root 1505 Feb 14 10:58 ca.cert

Router#run rm -rf /misc/config/grpc/ems.pem /misc/config/grpc/ems.key

Router#run ls -ltr /misc/config/grpc/

                        total 8
                        drwx------. 2 root root 4096 Feb 14 09:17 dialout
                        -rw-r--r--. 1 root root 1505 Feb 14 10:58 ca.cert
```

Step 3 Restart gRPC server by toggling the TLS configuration.

Configure gRPC with non TLS and then re-configure with TLS.

Example:

```
Router#config
                        Router(config)#grpc
                        Router(config-grpc)#no-tls
                        Router(config-grpc)#commit

Router#run ls -ltr /misc/config/grpc/

                        total 8
                        drwx------. 2 root root 4096 Feb 14 09:17 dialout
                        -rw-r--r--. 1 root root 1505 Feb 14 10:58 ca.cert
```

```
Router#config
                        Router(config)#grpc
                        Router(config-grpc)#no no-tls
                        Router(config-grpc)#commit
Router#run ls -ltr /misc/config/grpc/

                        total 16
                        drwx------. 2 root root 4096 Feb 14 09:17 dialout
                        -rw-rw-rw-. 1 root root 1505 Feb 14 14:23 ems.pem
                        -rw-------. 1 root root 1675 Feb 14 14:23 ems.key
                        -rw-r--r--. 1 root root 1505 Feb 14 14:23 ca.cert
```

Copy the newly generated `/misc/config/grpc/ems.pem` certificate in this path (from the device) to the gRPC client.

The common name is successfully configured and reflected in the regenerated EMSD certificate used for gRPC dial-in.

Example

For example, after configuring `certificate common-name cisco.com` and regenerating the certificate, the output of `show grpc` displays: *Certificate common name : cisco.com*.

What to do next

After completing this task:

- Ensure the gRPC client trusts the new certificate and can establish a secure connection using the updated common name.

gRPC over UNIX domain sockets

gRPC over UNIX domain sockets is a method that allows establishing gRPC connections using local containers without the need for password rotations.

- Extends gRPC TCP-based connections to UNIX domain sockets for local communication.
- Eliminates the need for username/password authentication for local containers.
- Improves security and control using UNIX file permissions.

This method enhances inter-process communication and simplifies secure local access to gRPC services on Cisco routers.

Feature History Table

gRPC server initialization and service registration

When gRPC is configured on the router, the gRPC server starts and then registers services such as gNMI and gNOI. After all the gRPC server registrations are complete, the listening socket is opened to listen to incoming gRPC connection requests. Currently, a TCP listen socket is created with the IP address, VRF, or gRPC listening port.

UNIX domain sockets and dual socket listening for gNMI

With this feature, the gRPC server listens over UNIX domain sockets that must be accessible from within the container through a local connection by default. With the UNIX socket enabled, the server listens on both TCP and UNIX sockets. However, if the UNIX socket is disabled, the server listens only on the TCP socket. The socket file is located at the directory.

Configure gRPC over UNIX domain sockets

You can use local containers and scripts on the router to establish gRPC connections over UNIX domain sockets.

Before you begin

Ensure that the router supports gRPC and that you have access to the CLI in configuration mode.

Procedure

Step 1 Configure the gRPC server

Example:

```
Router(config)#grpc
Router(config-grpc)#local-connection
Router(config-grpc)#commit
```

To disable the UNIX socket use the **no** form of the command.

```
Router(config-grpc)#no local-connection
```

The gRPC server restarts after you enable or disable the UNIX socket. If you disable the socket, any active gRPC sessions are dropped and the gRPC data store is reset.

The scale of gRPC requests remains the same and is split between the TCP and Unix socket connections. The maximum session limit is 256. If you utilize the 256 sessions on Unix sockets, further connections on either TCP or UNIX sockets are rejected.

Step 2 Verify that the local-connection is successfully enabled

Example:

```
Router#show grpc status
Thu Nov 25 16:51:30.382 UTC
*************************show gRPC status**********************
----------------------------------------------------------------
transport : grpc
access-family : tcp4
TLS : enabled
trustpoint :
listening-port : 57400
local-connection : enabled
max-request-per-user : 10
max-request-total : 128
```

```
max-streams : 32
max-streams-per-user : 32
vrf-socket-ns-path : global-vrf
min-client-keepalive-interval : 300
_____________________________________________________________
```

A gRPC client must dial into the socket to send connection requests.

Here is an example of a Go client connecting to a UNIX socket.

```
const sockAddr =
  " /misc/app_host/ems/grpc.sock" // for ncs_5500
  " /var/lib/docker/ems/grpc.sock" // for cisco8000
...

func UnixConnect(addr string, t time.Duration) (net.Conn, error) {
  unix_addr, err := net.ResolveUnixAddr("unix", sockAddr)
  conn, err := net.DialUnix("unix", nil, unix_addr)
  return conn, err
}

func main() {
  ...
  opts = append(opts, grpc.WithTimeout(time.Second*time.Duration(*operTimeout)))
  opts = append(opts, grpc.WithDefaultCallOptions(grpc.MaxCallRecvMsgSize(math.MaxInt32)))
  ...
  opts = append(opts, grpc.WithDialer(UnixConnect))
  conn, err := grpc.Dial(sockAddr, opts...)
  ...
}
```

The gRPC server is configured to accept connections over UNIX domain sockets, and clients can connect using the specified socket path.

What to do next

Monitor the gRPC sessions and ensure that the session count does not exceed the maximum limit of 256.

gRPC network management interface

The gRPC Network Management Interface (gNMI) is a gRPC-based network management protocol used to modify, install or delete configuration from network devices. It is also used to view operational data, control and generate telemetry streams from a target device to a data collection system. It uses a single protocol to manage configurations and stream telemetry data from network devices.

- Supports configuration management and telemetry streaming.
- Uses gRPC as the transport protocol.
- Enables real-time data collection without prior sensor path configuration.

gNMI subscription model and transport protocol

The subscription in gNMI does not require prior sensor path configuration on the target device. Sensor paths are requested by the collector (such as pipeline), and the subscription mode can be specified for each path. gNMI uses gRPC as the transport protocol and the configuration is same as that of gRPC.

gNMI operations

gNMI (gRPC Network Management Interface) operations define how clients interact with network devices to retrieve or modify configuration and operational data. These operations are part of the gNMI specification and are supported in Cisco IOS XR.

The gNMI operations include:

- **Capabilities:** Retrieves metadata about the network device
- **Get:** Retrieves state, configuration, and operational data
- **Set:** Modifies or deletes configuration data
- **Subscribe:** Subscribes to real-time updates for specific data paths
- **Release support:** Most operations are supported from release 7.0.1; Subscribe is supported from release 24.2.1

The following table lists the gNMI operations and their support in Cisco IOS XR:

Table 4: gNMI operations and their support in Cisco IOS XR

gNMI Operation	Supported Release	Description	Additional Details
Capabilities	Release 7.0.1	Retrieves the metadata of the network device.	—
Get	Release 7.0.1	Retrieve state data, configuration, and operational information from a network device	—
Set	Release 7.0.1	You can modify the state of a network device such as router's configuration, replace router's entire configuration sections, or delete specific parts of the configuration using the **Set** operation.	—
Subscribe	Release 24.2.1	Subscribes to a stream of updates for specific paths within the device's data model.	Stream Telemetry Data for LLDP Statistics

Note The gNMI Get operation is not supported for Sysadmin YANG models.

gNMI wildcards in schema path

gNMI wildcard schema is a method that supports the use of wildcards to represent all elements within a given subtree in the schema.

The gNMI wildcards are used for telemetry subscriptions or gNMI `Get` requests. The path is encoded in a structured format consisting of elements such as the path name and keys, which are represented as string values regardless of their type within the schema.

gNMI Wildcard Search Types

The table shows the gNMI wildcard search types.

Table 5: Single and multi-level wildcards

Single-level wildcard	Multi-level wildcard

The name of a path element is specified as an asterisk (*). The sample shows a wildcard as the key name. This operation returns the description for all interfaces on a device.
`path {`
`  elem {`
`  name: "interfaces"`
`  }`
`  elem {`
`  name: "interface"`
`  key {`
`  key: "name"`
`  value: "*"`
`  }`
`  }`
`  elem {`
`  name: "config"`
`  }`
`  elem {`
`  name: "description"`
`  }`
`}`
	The name of the path element is specified as an ellipsis (…). The example shows a wildcard search that returns all fields with a description available under `/interfaces` path.
	`path {`
	`  elem {`
	`  name: "interfaces"`
	`  }`
	`  elem {`
	`  name: "..."`
	`  }`
	`  elem {`
	`  name: "description"`
	`  }`
	`}`

gNMI Get request path to a leaf

The table shows the gNMI Get request and response messages in the schema path to fetch the operational state of an interface.

Message Type	gNMI Get Request	gNMI Get Response
gNMI operation to fetch operational state of an interface	path: < origin: "Cisco-IOS-XR-pfi-im-cmd-oper" elem: < name: "interfaces" > elem: < name: "interface-xr" > elem: < name: "interface" key: < key: "interface-name" value: "\"GigabitEthernet0/0/0/0\"" > > elem: < name: "state" > > type: OPERATIONAL encoding: JSON_IETF	notification: < timestamp: 1597974202517298341 update: < path: < origin: "Cisco-IOS-XR-pfi-im-cmd-oper" elem: < name: "interfaces" > elem: < name: "interface-xr" > elem: < name: "interface" key: < key: "interface-name" value: "\"GigabitEthernet0/0/0/0\"" > > elem: < name: "state" > > val: < json_ietf_val: im-state-admin-down > > > error: < >

Message Type	gNMI Get Request	gNMI Get Response
gNMI operation without a key specified in the schema path	path: < origin: "Cisco-IOS-XR-pfi-im-cmd-oper" elem: < name: "interfaces" > elem: < name: "interface-xr" > elem: < name: "interface" > elem: < name: "state" > > type: OPERATIONAL encoding: JSON_IETF	

Message Type	gNMI Get Request	gNMI Get Response
		path: < origin: "Cisco-IOS-XR-pfi-im-cmd-oper" elem: < name: "interfaces" > elem: < name: "interface-xr" > elem: < name: "interface" > elem: < name: "state" > > type: OPERATIONAL encoding: JSON_IETF notification: < timestamp: 1597974202517298341 update: < path: < origin: "Cisco-IOS-XR-pfi-im-cmd-oper" elem: < name: "interfaces" > elem: < name: "interface-xr" > elem: < name: "interface" key: < key: "interface-name" value: "\"GigabitEthernet0/0/0/0\"" > > elem: < name: "state" > > val: < json_ietf_val: im-state-admin-down > > update: < path: < origin:

Message Type	gNMI Get Request	gNMI Get Response
		"Cisco-IOS-XR-pfi-im-cmd-oper" elem: < name: "interfaces" > elem: < name: "interface-xr" > elem: < name: "interface" key: < key: "interface-name" value: "\"GigabitEthernet0/0/0/1\"" > > elem: < name: "state" > > val: < json_ietf_val: im-state-admin-down > > update: < path: < origin: "Cisco-IOS-XR-pfi-im-cmd-oper" elem: < name: "interfaces" > elem: < name: "interface-xr" > elem: < name: "interface" key: < key: "interface-name" value: "\"GigabitEthernet0/0/0/2\"" > > elem: < name: "state" > > val: < json_ietf_val: im-state-admin-down >

Message Type	gNMI Get Request	gNMI Get Response
		> update: < path: < origin: "Cisco-IOS-XR-pfi-im-cmd-oper" elem: < name: "interfaces" > elem: < name: "interface-xr" > elem: < name: "interface" key: < key: "interface-name" value: "\"MgmtEth0/RP0/CPU0/0\"" > > elem: < name: "state" > > val: < json_ietf_val: im-state-admin-down > >

Message Type	gNMI Get Request	gNMI Get Response

gNMI operation with unique path to a CLI	path: <
	origin: "cli"
	elem: <
	name: "show version"
	>
	>
	type: ALL
	encoding: ASCII
		path: <
		origin: "cli"
		elem: <
		name: "show version"
		>
		>
		type: ALL
		...
		...
		[
		{
		"source": "unix:///var/run/test_env.sock",
		"timestamp": 1730123328800447525,
		"time": "2024-10-28T06:48:48.800447525-07:00",
		"updates": [
		{
		"Path": "show version",
		"values": {
		"show version": "_____
		show version
		_____
		Cisco IOS XR Software, Version 24.4.1.37I
		Copyright (c) 2013-2024 by Cisco Systems, Inc.
		Build Information:\n
		Built By : swtools
		Built On : Mon Oct 21 03:16:32 PDT 2024
		Built Host : iox-lnx-121\n Workspace : /auto/iox-lnx-121-sm2/prod/24.4.1.37I.SIT_IMAGE/ncs5500/ws
		Version : 24.4.1.37I\n
		Location : /opt/cisco/XR/packages/
		Label : 24.4.1.37I-EFT2LabOnly
		cisco NCS-5500 () processor
		System uptime is 3 days 22 hours 54 minutes\n\n\n"
		}
		}
		]
		}
		]

gNMI bundling of telemetry updates

The gNMI bundling of telemetry updates is a method that

- optimizes bandwidth, and
- bundles multiple gNMI `Update` messages for the same client and sends them together.

Table 6: Feature History Table

Feature Name	Release Information	Description
gNMI bundling size enhancement	Release 7.8.1	With gRPC Network Management Interface (gNMI) bundling, the router internally bundles multiple gNMI `Update` messages meant for the same client into a single gNMI `Notification` message and sends it to the client over the interface. You can now optimize the interface bandwidth utilization by accommodating more gNMI updates in a single notification message to the client. We have now increased the gNMI bundling size from 32768 to 65536 bytes, and enabled gNMI bundling size configuration through Cisco native data model. Prior releases allowed only a maximum bundling size of 32768 bytes, and you could configure only through CLI. The feature introduces new XPaths to the `Cisco-IOS-XR-telemetry-model-driven-cfg.yang` Cisco native data model to configure gNMI bundling size. To view the specification of gNMI bundling, see Github repository.

The router internally bundles multiple gNMI `Update` messages into a single gNMI `Notification` message within a gNMI `SubscribeResponse` message. This approach reduces the number of bytes sent over the gNMI interface. IOS-XR software release Release 7.8.1 supports gNMI bundling size up to 65536 bytes.

Bundling instances of the client

This table shows how the router handles bundling instances for the same or different clients.

Bundling instances of the same client	Bundling instances of the different client
The router bundles multiple instances of the same client. For example, a router bundles interfaces `MgmtEth0/RP0/CPU0/0`, `FourHundredGigE0/0/0/0`, `FourHundredGigE0/0/0/1`, and so on, of this path. • Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters	The router does not bundle messages of different clients into a single gNMI Notification message. For example: • Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters • Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/protocol

Data under the container of the client path cannot be split into different bundles.

Timestamp assignment in gNMI notification messages

The gNMI Notification message contains a timestamp indicating when an event occurred or a sample was taken. The bundling process assigns a single timestamp for all bundled `Update` values, which is the timestamp of the first message in the bundle.

Exceptions to gNMI bundling size enforcement

The ON-CHANGE subscription mode does not support gNMI bundling.

The router does not enforce the bundling size in these scenarios:

- At the end of (N-1) message processing, if the notification message size is less than the configured bundling size, the router allows one extra instance, which could result in exceeding the bundling size.
- Data of a single instance exceeding the bundling size.
- The XPath `network-instances/network-instance/afts` does not support bundling.

Configure gNMI bundling size

Use this task to enable gNMI bundling and configure the bundling size for all gNMI subscribe sessions.

gNMI bundling is disabled by default and the default bundling size is 32,768 bytes. gNMI bundling size ranges from 1024 to 65536 bytes. Prior to IOS-XR software release Release 7.8.1 the range was 1024 to 32768 bytes. You can enable gNMI bundling to all gNMI subscribe sessions and specify the bundling size.

Before you begin

Ensure that you are in configuration mode and have access to the telemetry model-driven configuration context.

Procedure

Step 1 Enable gNMI bundling and configure bundling size

Example:

```
Router# configure
Router(config)# telemetry model-driven
Router(config-model-driven)# gnmi
Router(config-gnmi)# bundling
Router(config-gnmi-bdl)# size 2000
Router(config-gnmi-bdl)# commit
```

Step 2 Verify the running configuration

Example:

```
Router# show running-config
telemetry model-driven
gnmi
bundling
size 2000
!
```

```
!
!
```

gNMI bundling is enabled and the bundling size is set as configured. The configuration is visible in the running configuration.

Example

This configuration is useful when optimizing telemetry data transmission by controlling the size of gNMI message bundles.

What to do next

Monitor the gNMI telemetry performance to ensure the configured bundling size meets operational requirements.

Replace router configurations at sub-tree level using gNMI

The gNMI replace operation at the sub-tree level is a configuration management capability that enables targeted updates to specific sections of a router's configuration hierarchy. This operation uses a SetRequest RPC message to replace existing configurations with new ones, offering a model-aware and efficient approach to configuration management.

The gNMI replace operation provides:

- **Granular scope**: Operates at the sub-tree level within the same YANG model, allowing precise updates.
- **Structured targeting**: Accepts a structured path (with elements and key values) to define the root of the replace operation.
- **Behavioral logic**:
  - Reverts omitted elements with default values to their defaults.
  - Deletes omitted elements without defaults, returning them to an unconfigured state.

Table 7: Feature History Table

Feature Name	Release Information	Description
Replace Router Configuration at Sub-tree Level Using gNMI	Release 7.8.1	Using the gNMI `SetRequest` message, you can replace the router's existing configuration with a new set of configurations at the subtree level within the same model. Earlier you could replace router configurations at the data tree root level. To view the specification of gNMI replace, see Github repository.

The replace operation either includes all the path elements which are defined under the root or only a few of them. If the omitted path elements are configured with default values, they are reverted to their default values

during the replace operation. If the omitted path elements are not configured with default values, they are deleted from the data tree during the replace operation and returned to their original unconfigured state.

gNMI replace operation in data tree schema

```
root +
     |
     + a --+
     |     |
     |     +-- b
     |     |
     |     +-- c
     |
     |
     + d --+
           +-- e
           |
           +-- f
```

In this data tree schema, b has a default value of true and c has no default value. Both b and c are set as False.

When a replace operation is performed with e and f as set, and all other elements are omitted, b is reverted to its default setting true, and c is deleted from the tree, and returned to its original unconfigured state.

gNMI replace example

This example shows the gNMI replace request and response messages.

gNMI Replace Request	gNMI Replace Response

Request Message:
replace: <
  path: <
  elem: <
  name: "system"
  >
  elem: <
  name: "config"
  >
  elem: <
  name: "hostname"
  >
  >
  val: <
  json_ietf_val: "\"testing123\""
  >
>
	Response Message:
	  path: <
	  elem: <
	  name: "system"
	  >
	  elem: <
	  name: "config"
	  >
	  elem: <
	  name: "hostname"
	  >
	  >
	  op: REPLACE
	>
	message: <
	>
	timestamp: 1662873319202107537

gNMI union replace operations

gNMI union replace operations are a category of configuration update methods that:

- allow full replacement of router configurations in a single operation,
- support merging of multiple schema types including OpenConfig, CLI, and native YANG, and
- ensure alignment between intended and actual router configurations using a unified SetRequest RPC message.

Supported schema types

The gNMI union replace operation is a method that allows you to update your router's entire configuration in one go, ensuring that the actual settings of your network operating system align with the intended setup. To directly replace the existing router settings, this operation enables the merging of different schemas including:

- native YANG models,
- command-line interface (CLI), and
- OpenConfig YANG models.

Feature History Table

Table 8: Feature History Table

Feature Name	Release Information	Description
gNMI Union Replace Operation	Release 24.2.11	You can now update your router's entire configuration in one go to ensure that the actual settings of your network operating system align with the intended setup. The update includes OpenConfig (OC), Native YANG (NY), and CLI configurations and is done using the gRPC Network Management Interface (gNMI). The update is possible with the gNMI union-replace operation in a `gNMI SetRequest` RPC message which supports mixing of the configuration schemas. The supported schema combinations are: • OpenConfig (OC) and CLI • OC and native YANG (NY) To view the specification of gNMI union-replace, see the Github repository.

Supported schema combinations

gNMI union-replace operation in a `gNMI SetRequest` RPC message supports these two schema combinations:

- OC and CLI
- OC and NY

gNMI union-replace operation guidelines and limitations

gNMI union-replace operation guidelines are a category of configuration rules that:

- ensure the target router's state is updated only after all changes are successfully accepted,
- define how path-values are replaced, deleted, or defaulted based on their presence in the request and schema, and
- restrict the use of delete, replace, and update operations in union_replace RPC messages.

Operational behavior and constraints

Using gNMI when a client sends the `gNMI SetRequest` RPC message with union-replace operations to a target router:

- The state of the target router must not be changed until all the changes have been accepted successfully.
- If a particular path-value is specified in the gNMI request, the value replaces the current value in the target router.
- If a particular path-value isn't specified in the gNMI request and the path doesn't have a default value in the corresponding schema, it's deleted.
- If a path-value isn't specified in the gNMI request and the path does have a default value, the default value is applied on the target router.
- A `gNMI SetRequest` RPC message containing union_replace operations must not contain delete, replace, and update operations.

Origin field handling

The origin field in the path message of a gNMI union-replace operation is set to one of these options:

- **openconfig**: Path and content are part of OC YANG models.
- **cisco_native**: Path and content are part of Cisco's network operating system YANG models.
- **cisco_cli**: This origin represents an ASCII text or CLI configuration defined as command-line interface (CLI) text.

If the origin field is unspecified, the origin value is set to OpenConfig.

gNMI union replace operation examples

gNMI union replace operation examples are a category of schema combination use cases that:

- demonstrate how union_replace operations are structured in gNMI SetRequest RPC messages,
- illustrate the use of different origin schemas such as CLI, OC, and native YANG, and
- serve as references for implementing configuration merging strategies across schema types.

Schema combination examples

The schema combination examples show the union_replace operation in the `gNMI SetRequest` RPC message:

- OC and CLI origin, on page 30
- OC and NY origin, on page 32

OC and CLI origin

OC and CLI origin configurations are a category of gNMI union_replace schema combinations that:

- allow merging of configuration data from both OC and CLI origins,
- replace the router's existing configuration with the merged result, and

- prioritize CLI configuration values when overlapping with OC values.

Example of union_replace with OC and CLI origin

gNMI union_replace operation in `gNMI SetRequest` RPC message with OC and CLI origin schema combination example is as follows:

```
union_replace: {
path: {
origin: "cisco_cli"
}
val: {
ascii_val: "hostname myhost"
}
}
union_replace: {
path: {
elem: {
name: "interfaces"
}
elem: {
name: "interface"
key: {
key: "name"
value: "FourHundredGigE0/0/0/0"
}
}
elem: {
name: "config"
}
elem: {
name: "description"
}
}
val: {
json_ietf_val: "\"true\""
}
}
```

Replacement sequence for the OC and CLI origin schema combination

The configurations from both the schemas are merged and the merged configuration replaces the router's existing configuration.

Note If the CLI and OC configuration values overlap, the CLI configuration takes higher precedence and overwrites the value set by OC.

Guidelines for OC and CLI origin

Ensure that you don't use a union-replace operation with an empty path under OC or CLI origins. Doing so removes all the content of the respective origin on the target router.

A union-replace operation with OC and CLI schema combination containing bootz configuration, the processing order of the configuration application on the target router is as follows: OC → CLI → bootz.

OC and NY origin

OC and NY origin configurations are a category of gNMI union_replace schema combinations that:

- merge configuration data from both OC and Cisco native YANG origins,
- replace the router's existing configuration with the merged result, and
- prioritize NY configuration values when overlapping with OC values.

Example of union_replace with OC and NY origin

A gNMI union_replace operation in the `gNMI SetRequest` RPC message with OC and NY origin schema combination example is as follows.

```
union_replace: {
path: {
origin: "cisco_native"
elem: {
name: "Cisco-IOS-XR-shellutil-cfg:host-names"
}
elem: {
name: "host-name"
}
}
val: {
json_ietf_val: "\"abc\""
}
}
union_replace: {
path: {
elem: {
name: "interfaces"
}
elem: {
name: "interface"
key: {
key: "name"
value: "FourHundredGigE0/0/0/0"
}
}
elem: {
name: "config"
}
elem: {
name: "description"
}
}
val: {
json_ietf_val: "\"true\""
}
}
```

Guidelines for OC and NY origin

The configurations from both the schemas are merged and the merged configuration replaces the router's existing configuration.

If the OC and NY schema configuration values overlap, the NY configuration takes higher precedence and overwrites the value set by OC.

If an OC and NY union-replace request explicitly sets configuration items that are overlapping, the RPC doesn't return INVALID_ARGUMENT.

RPC error scenarios

RPC error scenarios are a category of gNMI SetRequest validation conditions that:

- occur when one of the origins from the supported schema combinations is missing or when the union_replace operation lacks a specified path value for one of the origins,
- arise when union-replace operations for all three origins (“cisco_native”, “cisco_cli”, and “openconfig”) are present in the `gNMI SetRequest` RPC message, and
- result from a `gNMI SetRequest` RPC message with union_replace operations that contain delete, replace, or update operations.

These conditions must be avoided to ensure successful processing of the RPC message.

gNMI XPath-based authorizations

A gNMI XPath-based authorization is a process where, upon receiving a gNMI SetRequest message for a configuration change, the router applies an XPath-based pathz policy to determine the request's authorization. The pathz policy originates from a gNSI RPC within the router. The policy configurations can be established during the router's boot process or dynamically adjusted while the router is operational.

- Authorization is based on XPath rules defined per user or group.
- Policies can be loaded securely during boot or dynamically.
- Authorization decisions result in PERMIT or DENY outcomes.

The router securely receives the initial pathz policy either through Secure Zero Touch Provisioning (sZTP) or a secure bootstrapping protocol like bootz when booting up. The policy includes the user or group name and a list of rules defining XPaths and their associated access permissions. The policy is enforced before processing any gNMI requests.

Authorization by the gNSI pathz policy is granted or denied based on user or group credentials, permitting or declining the gNMI SetRequest accordingly.

Table 9: Feature History Table

Feature Name	Release Information	Description
gNMI XPath-based authorization	Release 24.2.11	We've introduced gNMI authorization through the gNSI pathz policy which is adding authorization of a user or a group to access a specified YANG XPath through gNMI. The policy configurations can be done on the router either when the router boots up or dynamically when the router is up and running. When a user or a group sends a `gNMI SetRequest` message using a certain XPath, the system validates the request against the permissions specified in the policies associated with that user or the group. To view the specification of gNSI for the OpenConfig XPath-based Authorization, see the Github repository. This feature introduces these changes: **CLI:** • show gnsi path authorization policy • show gnsi path authorization counters • show gnsi trace pathz • show gnsi path authorization statistics • show tech-support gnsi • clear gnsi path authorization counters

The router securely receives the initial pathz policy either through Secure Zero Touch Provisioning (sZTP) or a secure bootstrapping protocol like bootz when booting up. The policy includes the user or group name and a list of rules defining XPaths and their associated access permissions. The policy is enforced before processing any gNMI requests.

Authorization by the gNSI pathz policy is granted or denied based on user or group credentials, permitting or declining the `gNMI SetRequest` accordingly.

gNMI authorization using gNSI pathz policy

Starting from release 24.2.11, you can perform gNMI XPath-based authorization using gNSI pathz policies.

The gnsi-pathz YANG model defines these counters and timestamps for each configured rule READ, WRITE, PERMIT, and DENY:

- access-rejects: 64-bit
- last-access-reject: timestamp
- access-accepts: 64-bit
- last-access-accept: timestamp

The counters get incremented per accepted or rejected XPath (e.g., per gNMI request).

Define Authorization Policy for a gNSI Pathz

The authorization policy for gNSI Pathz consists of three components.

The table lists the gNSI authorization policy components.

Table 10: Authorization Policy Components

Authorization Policy Component	Details
Users	Individuals named in rules or group definitions.
Groups of users	A group of users in the administrative domain, such as operators or administrators. • The matching policy gives precedence to a specific user over a group. • Match rules enable authorization against either a user or a group, but not both simultaneously.
Policy rules	Each rule defines a single authorization policy. • Authorization (how the policy is defined) is performed for a specific user in a predefined group of users on a specific gNMI path and a specific access methodology (example: READ or WRITE). • The wildcard character (*): • Replaces the missing keys in keyed path elements. Absence of keys implies a wildcard by default. • Masks all the values entirely, it doesn't permit partial value masking (Example: /this/is/a/keyed[name=Ethernet1/*/3]/things is invalid).

How Authorization Policy Matching Rules Work

Policy Matching Rule	Description
Multiple rules	The authorization process evaluates the rule with the longest match when granting access, rather than defaulting to the first rule encountered.
A defined KEY and wildcard in a keyed path	The defined KEY in the keyed path is preferred over the wildcard. For example, the router prefers /a/b[key=FOO]/c/d over /a/b[key=*]/c/d due to its more precise key match.
A user-specific rule and a corresponding group rule for the same user	The rule that corresponds to a specific user is prioritized over the one that matches with a user's group.
Permission mode	A mode that matches with the request (READ or WRITE) is considered.

Policy Matching Rule	Description
DENY or PERMIT	DENY takes priority over PERMIT when other conditions are equal, and multiple matching rules are present.

Policy evaluation results with a single best match rule for the provided {user, path, or mode}. If multiple best matches emerge, an error is logged, and the evaluation fails.

If no matching rule is found, an implicit DENY is applied and detailed in a log entry.

The authorization evaluation process results in a PERMIT or DENY decision, along with the version of the policy and the identifier of the rule applied.

Scenario for Authorization Policy Rules

Rule	User	Group	Path	Action	Mode
1	Bob	—	/interfaces/interface[FourHundredGigE0/0/0/0]	PERMIT	READ
2	Bob	—	/interfaces/interface[FourHundredGigE0/0/0/0]	PERMIT	WRITE
3	Bob	—	/interfaces/interface[FourHundredGigE1/1/1/1]	DENY	WRITE
4	—	Admin	/interfaces/interface[*]	PERMIT	WRITE
5	Bob	—	/interfaces	PERMIT	READ
6	—	Admin	/interfaces/interface[FourHundredGigE0/0/0/0]	PERMIT	WRITE
7	Jim	—	/interfaces/interface[FourHundredGigE0/0/0/0]	DENY	WRITE

For user Bob, these authorization ruless apply:

- READ or WRITE (gNMI request) access to the XPath /interfaces/interface[FourHundredGigE0/0/0/0] is granted under rules 1 and 2.
- READ access to the XPath /interfaces/interface[FourHundredGigE1/1/1/1] is granted under rule 5 due to the longest match criterion, which specifies READ mode. WRITE access to this path is denied by rule 3.
- WRITE access to the XPath /interfaces/interface[FourHundredGigE2/2/2/2] is granted being a member of the Admins group as specified by rule 4. Without the Admin membership, access is denied by the default deny all rule.
- READ access to the XPath /interfaces/interface[FourHundredGigE2/2/2/2] is granted under rule 5, independent of group affiliation.

For user Jim, these authorization rules apply:

- Access to the XPath /interfaces/interface[FourHundredGigE0/0/0/0] is controlled by a policy that favors personal user permissions over group permissions. As a result, although the admins group is allowed access, Jim is individually denied access because the policy emphasizes user-specific rules.

gNSI Pathz authorization policy configuration

The gNSI pathz authorization policy configuration defines how access control policies are applied to gRPC services using the gNSI pathz mechanism. These policies determine which users or groups are permitted or denied access to specific RPCs or data paths.

- **Policy-based access control:** Enables fine-grained authorization for gRPC services based on user or group identity.
- **Flexible deployment:** Policies can be configured either during initial setup or dynamically through rotation.
- **gNSI integration:** Uses the gNSI pathz service to enforce authorization rules across the network infrastructure.

Configuration methods

To set a gNSI pathz authorization policy, you can perform either of these methods:

- Initial policy loading
- Policy rotation

Load gNSI Pathz policies at boot-time

gNSI Pathz policy boot-time loading is a configuration mechanism that enables routers to apply authorization policies automatically during system startup.

- Supports loading via Secure Zero Touch Provisioning (sZTP).
- Supports loading via bootstrapping configuration workflows.

This ensures that path authorization policies are enforced immediately after boot, improving security and automation in deployment workflows.

To load gNSI pathz policies at boot-time into the router, you can use either sZTP or bootstrapping.

For details on loading gNSI pathz policy through sZTP, refer to *Secure Zero Touch Provisioning* section of *Cisco IOS XR Setup and Upgrade Guide for Cisco 8000 Series Routers* guide.

Rotate, finalize, and get the gNSI Pathz policy

gNSI Pathz policy rotation overview

gNSI Pathz policy rotation is a mechanism that allows dynamic management of authorization policies on a running router using gRPC operations.

- Supports rotating (updating) a candidate policy instance for testing.
- Allows finalizing (committing) the candidate policy to become the active policy.
- Enables retrieving (getting) the current active or candidate policy for inspection.

This mechanism ensures secure and flexible policy updates without requiring router reloads.

When the router is up and running, you can rotate (update), finalize (commit), and get (read) the gNSI pathz policy using the gNSI pathz gRPC operations. To view the specification of gNSI pathz policy rotation, see the Github repository.

gNSI pathz supports these policy instances:

- Active policy—Used for authorizing gNMI requests.
- Potential or candidate policy—Used to test a policy before rotation.

Rules for authorization policy rotation

- The node holds on to the candidate policy indefinitely until either:
  - The candidate is committed or again rotated, or
  - The RPC session is closed (this event removes the candidate instance).
- A single policy rotation RPC can be active at any given time. Concurrent RPC requests for policy rotation is rejected with the gRPC error code `UNAVAILABLE`.
  - gNMI allows different encodings, including JSON. IOS XR applies the gNSI pathz policy based on each leaf of the flattened JSON model for authorizing the gNMI request.

OpenConfig metadata for configuration annotations

OpenConfig metadata for configuration annotations is a YANG extension mechanism that:

- Allows tagging of configuration elements with metadata such as status, support, or intent.
- Enables tooling and documentation systems to interpret and display annotations.
- Improves clarity and consistency across OpenConfig models.

This extension enhances the semantic richness of YANG models by enabling structured metadata tagging.

Table 11: Feature History Table

Feature Name	Release	Description
OpenConfig metadata for configuration annotations	Release 7.10.1	You can annotate the **OpenConfig-metadata** as part of the OpenConfig **edit-config** request to the Cisco IOS XR router and later fetch using the OpenConfig **get-config** request or delete through gNMI request only. The `Set` or `Get` operations can be performed through gNMI only; not through Netconf RPCs.

An OpenConfig Metadata feature allows you to set or delete **OpenConfig-metadata** at the **root** level node through gNMI requests only, and it can be read back while retrieving or verifying the device configuration. Netconf RPC requests are not supported.

Note The usage guidelines in this document provides the OpenConfig YANG support for a specific metadata annotation based on RFC7952 requirements for configuration commits only.

This solution is intended for the requirements of the **OpenConfig-metadata** annotation use case only and not intended to be changed for any other use beyond the scope of this document.

Here is an example for the item.

```
{
  "@": {
    "openconfig-metadata:config-metadata": "xyz" // xyz is base64 encoded string per RFC7951
 encoding rules
  }
  // Rest of configurations
}
```

The **OpenConfig-metadata** annotation is persistent across system restart. The latest **OpenConfig-metadata** annotation is preserved and it overwrites all the previous data. Also, the previous or old **OpenConfig-metadata** annotations cannot be retrieved with any operation (including configuration rollback). If the commit action fails, then the **OpenConfig-metadata** annotation is not updated. During startup failures resulting in removal of running configurations, the **OpenConfig-metadata** annotation at the time of last commit shall persist.

Example: Set request

This is a sample `Set` request for **OpenConfig-metadata**:

```
Request:
-------
update: {
  path: {
  }
  val: {
    json_ietf_val: "{\"openconfig-lldp:lldp\":{\"config\":{
\"enabled\":true,\"system-description\":\"test-replace\"}},
\"@\":{\"openconfig-metadata:protobuf-metadata\":
\"01234567890123456789012345678901234567890123456789
12345678901234567890123456789012345678901234567890123456789\"}}"
  }
}

Response:
--------

response: <
  path: <
  >
  op: UPDATE
>
message: <
>
timestamp: 1662150302538441219
```

Example: Get request

This is a sample `Get` request for **OpenConfig-metadata**:

```
Request:
-------
path:  {
```

```
  elem:  {
    name:  "@"
  }
  elem: {
    name:  "protobuf-metadata"
  }
}
type:  CONFIG
encoding:  JSON_IETF

Response:
--------

notification: <
  timestamp: 1662869232324390815
  update: <
    path: <
      origin: "openconfig"
      elem: <
        name: "@"
      >
      elem: <
        name: "protobuf-metadata"
      >
    >
    val: <
      json_ietf_val: "\"0123456789012345678901234567890
1234567890123456789012345678901234567890123456789012345678901234567890
1234567890123456789\""
    >
  >
>
error: <
>
```

Verification

The **OpenConfig-metadata** annotations are stored persistently in the router and are opaque (not visible) to the IOS XR routers. However, the **show** command displays the presence and size of the **OpenConfig-metadata** annotation.

This example displays the **show** command output:

```
Router#show cfgmgr commitdb
.
.
.
last-commit-metadata-len
[UINT32] 100000 (Ox186A0)
.
.
.
```

Note The **show** command displays only the presence and size of the **OpenConfig-metadata** annotation. If there is no **OpenConfig-metadata** annotation stored in the persistent database, then the output of the **show** command will not contain this entry.

Metrics of gNSI authorization rules

gNSI authorization rule metrics are operational diagnostics that provide visibility into the behavior and performance of path-based access control in IOS XR.

- They include statistics and counters for policy enforcement and access attempts.
- They provide trace data for debugging authorization flows.
- They support tech support outputs for comprehensive diagnostics.

These metrics help administrators monitor, troubleshoot, and validate gNSI-based access control configurations.

IOS-XR pathz supports these statistics, counters, diagnostics, and trace data commands for the gNSI authorization rules:

- Pathz Policy and Statistics
- Path Authorization Counters
- Traces for Pathz
- gNSI Tech Support

gNSI path authorization counters

gNSI path authorization counters are diagnostic statistics that help monitor access control decisions for gRPC-based network services.

- They display read and write access attempts for all or specific XPaths on a gRPC server.
- They include accept and reject counts for each path, along with timestamps of the last access attempts.
- They support optional filtering by server name and XPath for targeted inspection.

These counters are useful for auditing authorization behavior and verifying policy enforcement across network paths.

The gNSI path authorization counters show the counters for a given gRPC server-name for all XPaths, or the specified XPath. Providing the XPath and server-name is optional. To view the gNSI Path Authorization counters, use the **show gnsi path authorization counters** command.

- ```
  Router# show gnsi path authorization counters
                                  Mon Apr 1 08:05:46.297 UTC
                                  ----------------Pathz Counters Info--------------
                                  /system/config/hostname:
                                  Read Write
                                  Rejects : 0 0
                                  Last : N/A N/A
                                  Accepts : 0 3
                                  Last : N/A Mon, 01 Apr 2024 08:05:25 +0000
                                  Total path records received 1


  Router# show gnsi path authorization counters server-name 64.103.223.33
                                  Mon Apr 1 08:33:25.194 UTC
                                  ----------------Pathz Counters Info--------------
                                  /:
                                  Read Write
                                  Rejects : 0 2
                                  Last : N/A Mon, 01 Apr 2024 08:32:37 +0000
  ```
  ```

```
                                Accepts : 0 0
                                Last : N/A N/A
                                /system/config/hostname:
                                Read Write
                                Rejects : 0 6
                                Last : N/A Mon, 01 Apr 2024 08:32:36 +0000
                                Accepts : 0 0
                                Last : N/A N/A
                                Total path records received 2
                                Router#

Router# show gnsi path authorization counters path /system/config/hostname
                                Mon Apr 1 08:32:46.468 UTC
                                ----------------Pathz Counters Info--------------
                                /system/config/hostname:
                                Read Write
                                Rejects : 0 6
                                Last : N/A Mon, 01 Apr 2024 08:32:36 +0000
                                Accepts : 0 0
                                Last : N/A N/A
                                Total path records received 1
                                Router#
```

- To clear the gNSI path authorization counters, use the **clear gnsi path authorization counters** command.

```
Router# clear gnsi path authorization counters
                             Router#
```

gNSI Pathz policy and statistics

gNSI Pathz policy and statistics are operational tools used to inspect and monitor path-based authorization behavior in gRPC-enabled systems.

- They allow you to view the active and sandbox policies configured for gNSI path authorization.
- They provide detailed counters and error metrics related to policy evaluation, gNMI path access, and internal engine operations.
- They help in troubleshooting and validating policy enforcement through CLI-based inspection commands.

These tools are essential for administrators to verify policy deployment and monitor authorization activity in real time.

To display the configured gNSI policy and statistics, use these commands:

- **show gnsi path authorization policy** — Shows the running gNSI path authorization policy.
- **show gnsi path authorization statistics** — Shows gNSI path authorization statistics.

```
Router# show gnsi path authorization policy
          Mon Apr 1 04:29:37.905 UTC
          version:"1" created_on:1711946719670313 policy:{rules:{user:"cafyauto"
path:{origin:"openconfig" elem:{name:"system"} elem:{name:"config"} elem:{name:"hostname"}}
 action:ACTION_PERMIT mode:MODE_WRITE}}
          Router#


Router# show gnsi path authorization statistics
          Mon Apr 1 04:29:23.259 UTC
          ----------------Pathz Info--------------
```

```
Engine:

State:
Active Policy:
Version : 1
Created On (UTC) : Wed, 09 Dec 54251401 07:58:33 +0000
Sandbox Policy:
Version : N/A
Created On (UTC) : N/A
Policy Rotation in Progress: False

Stats:
Rotations in Progress Count: 0
Policy Rotations : 0
Policy Rotation Errors : 0
Policy Upload Requests : 0
Policy Upload Errors : 0
Policy Finalize : 0
Policy Finalize Errors : 0
Probe Requests : 0
Probe Errors : 0
Get Requests : 0
Get Errors : 0
Policy Unmarshall Errors : 0
Sandbox Policy Errors : 0

Counters:
No Policy Auth Requests : 0
gNMI Path Leaves : 0
gNMI Authorizations : 0
gNMI Set Path Permit : 0
gNMI Set Path Deny : 0
gNMI Get Path Permit : 0
gNMI Get Path Deny : 0

Errors:
Path To String : 0
Origin Type : 0
Bad Mode : 0
Bad Action : 0
JSON Flatten : 0
String To Path : 0
Join Paths : 0
Nil Path : 0
Nil SetRequest : 0
Empty User : 0
Probe Internal : 0
Path Counters:
Increment : 0
Find : 0
Clear : 0
Walk : 0
```

gNSI Pathz trace data

gNSI Pathz trace data is a diagnostic output that captures real-time authorization events and policy evaluations for gNSI path-based access control.

- Provides visibility into policy loading, sandboxing, and activation events.
- Logs authorization decisions including denied and permitted paths.
- Helps troubleshoot issues related to policy application and access control enforcement.

This trace data is essential for auditing and debugging gNSI path authorization behavior on IOS XR devices.

To trace the configured gNSI policy, use the **show gnsi trace pathz** command.

```
                    Router# show gnsi trace pathz all
                    Mon Apr 1 04:31:26.689 UTC
                    61 wrapping entries (21760 possible, 512 allocated, 0 filtered, 61
total)
                   Apr 1 04:07:09.681 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(178) 'Trying
 to load policy' '/mnt/rdsfs/ems/gnsi/pathz_policy.txt'
                    Apr 1 04:07:09.685 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(173) 'Set
Sandbox policy' '1(54251382-02-18 11:34:58 +0000 UTC)'
                    Apr 1 04:07:09.685 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(179) 'Set
Policy from' '/mnt/rdsfs/ems/gnsi/pathz_policy.txt'
                   Apr 1 04:07:09.685 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(249) 'Pathz
 Policy Clearing Counters' ' '
                  Apr 1 04:07:09.685 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code (79): 'Engine
 Initialized'
                    Apr 1 04:08:05.761 gnsi/pathz 0/RP0/CPU0 t11794 Pathz: Code(63)
'Pathz.Get()' '5.38.4.111:52126'
                    Apr 1 04:08:05.761 gnsi/pathz_err 0/RP0/CPU0 t11794 Pathz ERROR: Code
(65): 'Nil Policy'
                    Apr 1 04:08:05.788 gnsi/pathz 0/RP0/CPU0 t11480 Pathz: Code(63)
'Pathz.Get()' '5.38.4.111:52126'
                    Apr 1 04:08:05.788 gnsi/pathz 0/RP0/CPU0 t11480 Pathz: Code(176) 'Get'
 'POLICY_INSTANCE_ACTIVE 1(1711946094752098)'
                  Apr 1 04:08:05.791 gnsi/pathz_deny 0/RP0/CPU0 t11481 Pathz DENY: Code(235)
 'Upd/Rep Denied path' 'cafyauto@/system/config/hostname, 1,1711946094752098'
                  Apr 1 04:08:05.808 gnsi/pathz_deny 0/RP0/CPU0 t11383 Pathz DENY: Code(234)
 'Del Denied path' 'cafyauto@/system/config/hostname, 1,1711946094752098'
                  Apr 1 04:08:05.821 gnsi/pathz_deny 0/RP0/CPU0 t11480 Pathz DENY: Code(235)
 'Upd/Rep Denied path' 'cafyauto@/system/config/hostname, 1,1711946094752098'
                  Apr 1 04:08:07.348 gnsi/pathz_deny 0/RP0/CPU0 t11383 Pathz DENY: Code(235)
 'Upd/Rep Denied path' 'cafyauto@/lldp/config/enabled, 1,1711946094752098'
                    Apr 1 04:08:08.205 gnsi/pathz 0/RP0/CPU0 t11383 Pathz: Code(63)
'Pathz.Get()' '5.38.4.111:52126'
                    Apr 1 04:08:08.205 gnsi/pathz_err 0/RP0/CPU0 t11383 Pathz ERROR: Code
(65): 'Nil Policy'
                    Apr 1 04:08:08.221 gnsi/pathz 0/RP0/CPU0 t11480 Pathz: Code(63)
'Pathz.Get()' '5.38.4.111:52126'
                    Apr 1 04:08:08.221 gnsi/pathz 0/RP0/CPU0 t11480 Pathz: Code(176) 'Get'
 'POLICY_INSTANCE_ACTIVE 1(1711946094752098)'
                  Apr 1 04:08:08.238 gnsi/pathz_deny 0/RP0/CPU0 t11481 Pathz DENY: Code(235)
 'Upd/Rep Denied path' 'cafyauto@/system/config/hostname, 1,1711946094752098'
                  Apr 1 04:08:08.281 gnsi/pathz_deny 0/RP0/CPU0 t11480 Pathz DENY: Code(234)
 'Del Denied path' 'cafyauto@/system/config/hostname, 1,1711946094752098'
                    Router#
```

gNSI state details

The **show tech-support gnsi** command is used to collect diagnostic information related to the gRPC Network Security Interface (gNSI). This command helps in troubleshooting and analyzing the state of gNSI services on the router.

- **Purpose:** Gathers detailed technical data for gNSI diagnostics.
- **Output:** Saves the collected data in a compressed file on the router's storage.
- **Usage:** Useful for support and debugging purposes during gNSI-related issues.

Command usage

To collect diagnostic information of gNSI, use the following command:

```
                    Router# show tech-support gnsi
                    Mon Apr 1 06:55:51.482 UTC
                    ++ Show tech start time: 2024-Apr-01.065551.UTC ++
                    Mon Apr 1 06:55:52 UTC 2024 Waiting for gathering to complete
                    ...
                    Mon Apr 1 06:56:01 UTC 2024 Compressing show tech output
                    Show tech output available at Router#:
/harddisk:/showtech/showtech-mtb_sf2-gnsi-2024-Apr-01.065551.UTC.tgz
                    ++ Show tech end time: 2024-Apr-01.065601.UTC ++
```

The **show tech-support gnsi** command places the collected diagnostic information in a file, for example:

Router#: /harddisk:/showtech/showtech-mtb_sf2-gnsi-2024-Apr-01.065551.UTC.tgz

gRPC network operations interface

The gRPC Network Operations Interface (gNOI) is a set of gRPC-based microservices for executing operational commands on network devices. These services are used in conjunction with the gRPC Network Management Interface (gNMI) to manage both target and operational state across the network.

- **Modular services:** gNOI provides modular services for specific operational tasks such as rebooting, certificate management, and file operations.
- **RPC-driven operations:** Each gNOI service is implemented as a Remote Procedure Call (RPC), enabling precise and efficient execution of operational commands.
- **Integration with gNMI:** gNOI complements gNMI by handling operational state and actions, while gNMI manages configuration and telemetry.
- **Transport via gRPC:** gNOI uses gRPC as its transport protocol, ensuring secure, high-performance communication.
- **OpenConfig standard compliance:** gNOI services and messages are defined using OpenConfig protocol buffers (proto files), ensuring interoperability and vendor neutrality.

For more information about gNOI, see the GitHub repository.

gNOI RPCs

To send gNOI RPC requests, you need a client that implements the gNOI client interface for each RPC.

All messages within the gRPC service definition are defined as protocol buffer (.proto) files. gNOI OpenConfig proto files are located in the Github repository.

Table 12: Feature History Table

Feature Name	Release Information	Description
gNOI System Proto	Release 7.8.1	You can now avail the services of `CancelReboot` to terminate outstanding reboot request, and `KillProcess` RPCs to restart the process on device.

gNOI supports these remote procedure calls (RPCs).

Table 13: gNOI RPC Types, RPC Names, and Description

RPC Type	Purpose	RPC Name	Description
System	System RPCs enable system-level operations including software upgrades, device reboots, and network troubleshooting. For more details on the**system.proto**, see the Github repository.	Reboot	Reboots the target. The router supports these reboot options: • COLD = 1; Shutdown and restart OS and all hardware • POWERDOWN = 2; Halt and power down • HALT = 3; Halt • POWERUP = 7; Apply power
		RebootStatus	Returns the status of the target reboot.
		SetPackage	Places a software package including bootable images on the target device.
		Ping	Pings the target device and streams the results of the ping operation.
		Traceroute	Runs the traceroute command on the target device and streams the result. The default hop count is 30.
		Time	Returns the current time on the target device.
		SwitchControlProcessor	Switches from the current route processor to the specified route processor. If the target does not exist, the RPC returns an error message.
		CancelReboot	Cancels any pending reboot request.
		KillProcess	Stops an OS process and optionally restarts it.

RPC Type	Purpose	RPC Name	Description
File	File RPCs facilitate file-level operations, including reading file contents and metadata. For more details on the **file.proto**, see the Github repository.	Get	Reads and streams the contents of a file from the target device. The RPC streams the file as sequential messages with 64 KB of data.
		Remove	Removes the specified file from the target device. The RPC returns an error if the file does not exist or permission is denied to remove the file.
		Stat	Returns metadata about a file on the target device. **Note** gNOI File.Stat returns only the filename in the response, which can cause incorrect handling, especially during recursive processing, as the file might be mistakenly treated as a directory.
		Put	Streams data into a file on the target device.
		TransferToRemote	Transfers the contents of a file from the target device to a specified remote location. The response contains the hash of the transferred data. The RPC returns an error if the file does not exist, the file transfer fails or an error when reading the file. This is a blocking call until the file transfer is complete.

RPC Type	Purpose	RPC Name	Description
Certificate Management (Cert)	Certificate Management RPCs handle certificate operations on the target device. For more details on the **cert.proto**, see the Github repository.	Rotate	Replaces an existing certificate on the target device by creating a new CSR request and placing the new certificate on the target device. If the process fails, the target rolls back to the original certificate.
		Install	Installs a new certificate on the target by creating a new CSR request and placing the new certificate on the target based on the CSR.
		GetCertificates	Gets the certificates on the target.
		RevokeCertificates	Revokes specific certificates.
		CanGenerateCSR	Asks a target if the certificate can be generated.
		LoadCertificateAuthorityBundle	Loads a bundle of CA certificates on the target. This CA certificate bundle is used to verify the client certificate when mutual TLS is enabled.
Interface	Interface RPCs manage operations on the interfaces. Fore more details on the **interface.proto**, see the Github repository.	SetLoopbackMode	Sets the loopback mode on an interface.
		GetLoopbackMode	Gets the loopback mode on an interface.
		ClearInterfaceCounters	Resets the counters for the specified interface.
Layer2	Layer2 RPCs facilitate operations on the Link Layer Discovery Protocol (LLDP) for layer 2 neighbor discovery. For more details on the **layer2.proto**, see the Github repository.	ClearLLDPInterface	Clears all the LLDP adjacencies on the specified interface.

RPC Type	Purpose	RPC Name	Description
BGP	BGP RPCs manage operations for the Link Layer Discovery Protocol (LLDP) and layer 2 neighbor discovery For more details on the **bgp.proto**, see the Github repository.	ClearBGPNeighbor	Clears a BGP session.
Diagnostic (Diag)	Diagnostic RPCs execute diagnostic tests on the target device, utilizing unique IDs to manage each bit error rate test (BERT) operation. For more details on the **diag.proto**, see the Github repository.	StartBERT	Starts BERT on a pair of connected ports between devices in the network.
		StopBERT	Stops an already in-progress BERT on a set of ports.
		GetBERTResult	Gets the BERT results during the BERT or after the operation is complete.
MPLS	MPLS RPCs execute MPLS-related operations on the target device. For more details on the **mpls.proto**, see the Github repository.	MPLSPing	Checks basic connectivity using MPLS ping operation. See RFC 4379. In Cisco IOS XR Release 7.5.4, the RPC supports `ldp_fec` and `rsvpte_lsp_name` destination types. The destination types `fec129_pwe` and `rsvpte_lsp` are not supported.
		ClearLSP	Clears a single tunnel.
		ClearLSPCounters	Clears the MPLS counters for the specified Label Switched Path (LSP)

RPC Type	Purpose	RPC Name	Description
Operating System (OS)	The OS service offers an interface for installing the OS on a target device, with RPCs used to update the router software and upgrade the system. Concurrent installations on the same target are not permitted. For more details on the **os.proto**, use the Github repository.	Install	Transfers an OS package onto the target. **Note** Only Golden ISO installation is supported; RPM installation is not supported.
		Activate	Sets the requested OS version as the version that is used at the next reboot. If booting up the requested OS version fails, the system recovers by rolling back to the previously running OS package.
		Verify	Verifies the running OS version.

These examples shows the gNOI supported RPCs.

Table 14: gNOI RPC Name, Purpose, and Example

RPC Name	Purpose	Example

Get	Streams the contents of a file from the target.	RPC to 10.105.57.106:57900
		RPC start time: 20:58:27.513638
		----------File Get Request----------
		RPC start time: 20:58:27.513668
		remote_file: "harddisk:/giso_image_repo/test.log"
		
		----------File Get Response----------
		RPC end time: 20:58:27.518413
		contents: "GNOI \n\n"
		
		hash {
		method: MD5
		hash: "Ð\002\375ñ\237\322\024\341\370\361\9k\310\333\016\343"
		}

RPC Name	Purpose	Example
Remove	Remove the specified file from the target.	RPC to 10.105.57.106:57900 RPC start time: 21:07:57.089554 ----------File Remove Request---------- remote_file: "harddisk:/sample.txt" ----------File Remove Response---------- RPC end time: 21:09:27.796217 File removal harddisk:/sample.txt successful

Reboot	Reloads a requested target.	RPC to 10.105.57.106:57900
		RPC start time: 21:12:49.811536
		----------Reboot Request----------
		RPC start time: 21:12:49.811561
		method: COLD
		message: "Test Reboot"
		subcomponents {
		origin: "openconfig-platform"
		elem {
		name: "components"
		}
		elem {
		name: "component"
		key {
		key: "name"
		value: "0/RP0"
		}
		}
		elem {
		name: "state"
		}
		elem {
		name: "location"
		}
		}

Set Package	Places software package on the target.	RPC to 10.105.57.106:57900
		RPC start time: 21:12:49.811536
		----------Set Package Request----------
		RPC start time: 15:33:34.378745
		Sending SetPackage RPC
		package {
		filename:
		"harddisk:/giso_image_repo/<platform-version>-giso.iso"
		activate: true
		}
		method: MD5
		hash:
		"\314\207\354\217\270=\021\341y\355\240\274\003\034\334"
		RPC end time: 15:47:00.928361

RPC Name	Purpose	Example

Reboot Status	Returns the status of reboot for the target.	RPC to 10.105.57.106:57900
		RPC start time: 22:27:34.209473
		----------Reboot Status Request----------
		subcomponents {
		origin: "openconfig-platform"
		elem {
		name: "components"
		}
		elem {
		name: "component"
		key {
		key: "name"
		value: "0/RP0"
		}
		}
		elem {
		name: "state"
		}
		elem
		name: "location"
		}
		}
		
		RPC end time: 22:27:34.319618
		
		----------Reboot Status Response----------
		Active : False
		Wait : 0
		When : 0
		Reason : Test Reboot
		Count : 0
		CancelReboot RPC
		
		Cancels any outstanding reboot
		
		Request :
		CancelRebootRequest
		subcomponents {
		origin: "openconfig-platform"
		elem {
		name: "components"
		}
		elem {
		name: "component"
		key {
		key: "name"
		value: "0/RP0/CPU0"
		}
		}
		elem {
		name: "state"
		}
		elem {
		name: "location"
		}
		}

RPC Name	Purpose	Example
CancelRebootResponse	Cancels any outstanding reboot	`(rhel7-22.24.10) -bash-4.2$`

KillProcess	Kills the executing process. Either a PID or process name must be specified, and a termination signal must be specified.	`KillProcessRequest`
		`pid: 3451`
		`signal: SIGNAL_TERM`
		
		`KillProcessResponse`
		`-bash-4.2$`

gNOI packet link qualifications

The gRPC network operations interface (gNOI) packet link qualification is a link qualification service which provides a way to certify link quality between a generator and a reflector device.

- Provides a method to check link quality using test traffic between generator and reflector devices.
- Supports RPC-based diagnostics to assess packet transmission and reception metrics.
- Includes capabilities to fetch transmission rate and link capacity via gNSI RPCs.

Additional information such as supported roles, timing configurations, and RPC specifications are available through the gNOI protocol documentation.

Table 15: Feature History Table

Feature Name	Release Information	Feature Description
gNOI Packet Link Qualification	Release 24.2.11	You can now check and assess the reliability of the link speed and packet drops between the two network devices (generator and the reflector) by performing the gNOI packet-based link qualification service. This can be achieved by sending the packets from the generator to the reflector, and receiving the looped back packets from the reflector within a certain tolerance limit.

Packet link qualification overview

The gRPC network operations interface (gNOI) packet link qualification provides a method to check the link quality between a generator and a reflector device. The generator device generates test traffic and sends it out of the requested interface, maintaining counters of the sent, received, errored, and dropped packets. The reflector device loops back the traffic on the requested interface. The packet-based link qualification service verifies that the packets are sent and received on the requested interface. You can obtain the transmission rate and the link's capacity range for that interface from the gNSI Packet Link Qualification RPC messages: `Capabilities` and `Get`.

To view the packet link qualification specification, see the Github repository.

This table lists the packet link qualification RPCs.

Table 16: Packet link qualification (PLQ) RPCs

RPC	Description
Capabilities	Fetches the capabilities of the device as a link qualification service. The capabilities result includes: • The roles supported on the device (Packet generator, Physical Medium Dependent (PMD) loopback reflector) • Information on whether the NTP synchronization is supported or not • Information on whether the current device time is synchronized through NTP or not. • The Maximum number of results stored per interface
Create	Creates a set of link qualifications on the device. Each element in a `Create` message specifies these parameters: • A unique qualification ID • The interface on which to run the qualification • The endpoint type (the role of the device) • Role-specific configuration • Timing information in the form of either NTP-based or RPC-based timing For more information, see `Link Qualifications Based on Timing` table. **Note** Packet generator and PMD loopback roles are supported The packet injector and ASIC loopback roles are not supported.
Delete	Deletes a set of qualifications by their IDs. Stops all the running qualification tests listed and deletes their records from the device. The qualifications are automatically deleted from the device 24 hours either after successful completion or in the event of any error.
Get	Gets the status of each of the unique qualification IDs that you specify. For generator qualifications, it returns the number of packets sent, received, errored, dropped, and the expected and achieved rate in bytes per second. This data isn't present for reflector qualifications.
List	This RPC lists all the qualifications on the device.

Link qualifications based on timing

When you run the `Create RPC` (see table `Packet Link Qualification (PLQ) RPCs`), it creates a set of link qualifications based on either it's NTP-based or RPC-based timing.

For both NTP-based and RPC-based timings, the qualification start time must be set no earlier than the minimum setup duration from the current time, as specified in the `Capabilities` RPC (see table `Packet Link Qualification (PLQ) RPCs`) response message.

The table lists the NTP-based and RPC-based link qualification timing.

Table 17: Link qualification timing

NTP-based Timing	RPC-based Timing
NTP-based timing specifies: • Specific start time • Specific end time • Teardown time	RPC-based timing specifies: • Presync duration (duration from the current time to when the setup should start) • Setup duration • Qualification duration • Postsync duration (duration from the end of the qualification to when the teardown should start) • Teardown duration

gNOI Healthz

The gRPC Network Operations Interface (gNOI) Healthz is a gRPC service that focuses on the health check and monitoring of the network devices. It determines whether all the nodes of a network are fully functional, degraded, or must be replaced. The gNOI Healthz process involves:

- Waiting for the health status data from various subsystem components
- Inspecting and analyzing health status data to identify any unhealthy entities
- Collecting logs

Table 18: Feature History Table

Feature Name	Release Information	Feature Description
gNOI Healthz	Release 24.4.1	With gNOI Healthz, you can monitor and troubleshoot device health by collecting logs and conducting root-cause analysis on detected issues. This proactive approach enables early identification and resolution of system health problems, thereby reducing downtime and enhancing reliability. For the specification on gNOI.healthz, see the GitHub repository.

Health Monitoring with gNOI Healthz

gNOI Healthz, in conjunction with gNMI telemetry, monitors the health of network components.

When a component becomes HEALTHY or UNHEALTHY, a telemetry update is sent for that health event. For more details about the health event, see gNOI Healthz RPCs. When a system component changes its state to UNHEALTHY, the intended artifacts (debug logs, core file, and so on) are generated automatically at the time of the health event.

Router Health Status Updates Workflow

1. The client subscribes to the component's OpenConfig path with an ON_CHANGE request and waits for a health event to occur. When a health event is detected in the router for that component, the client receives a notification. The client monitors these parameters:
   - **status**: Health, Unhealthy, or Unknown
   - **last-unhealthy time**: Timestamp of last known healthy state
   - **unhealthy-count**: Number of times the particular component is reported unhealthy
2. When the router receives gNOI Healthz RPCs from gNOI client, it performs these actions and responds to the gNOI client.

Table 19: gNOI healthz RPCs

When the gNOI client sends...	The Router...
Get RPC	Retrieves the latest set of health statuses that are associated with a specific component and its subcomponents.
List RPC	Returns all events that are associated with a device.
Artifact RPC	Retrieves specific artifacts that are listed by the target system in the List() or Get() RPC.
Acknowledge RPC	Acknowledges a series of artifacts that are listed by the Acknowledge() RPC.
Check RPC	Performs intensive health checks that may impact the service, ensuring they are done intentionally to avoid disruptions.

Verify router health using gNOI RPCs

Monitor health status telemetry of a router using gNOI healthz RPC.

Procedure

Step 1 Monitor health state of the router.

Example:

```
Router# show health status
            SNo   Component name                          Health status
```

```
----- ---------------------------------------- --------------------
1     0_RP0_CPU0-appmgr                        healthy
2     0_RP0_CPU0-ownershipd                    healthy
```

Step 2 Monitor router health with gNOI **List RPC** by tracking all the events.

Example:

```
Router# /auto/appmgr/xrhealth/bin/gnoic -a ${MGMT_IP} --port 57400 --insecure -u
 cisco -p <password> healthz
            list --path "openconfig:/components/component[name=${OC_COMP}]"
            WARN[0000] "192.0.1.0" could not lookup hostname: lookup
198.51.100.0.in-addr.arpa. on
            171.70.168.183:53: no such host
            target "192.0.1.0:57400":

+----------------+----------------+-----------------------------------------+-------------+

            |Target Name       |          ID           |
Path                            |      Status           |
                                                                            |

+----------------+----------------+-----------------------------------------+-------------+

            | 192.0.1.0:57400 | 1721815320614225976 |
openconfig:components/component[name=0_RP0_CPU0-appmgr] | STATUS_UNHEALTHY |
            | 192.0.1.0:57400 | 1721815320614225976 |
openconfig:components/component[name=0_RP0_CPU0-appmgr] | STATUS_UNHEALTHY |
            | 192.0.1.0:57400 | 1721815321290718105 |
openconfig:components/component[name=0_RP0_CPU0-appmgr] | STATUS_HEALTHY   |

+----------------+----------------+-----------------------------------------+-------------+



+-----------------------------+-------------------------------------------------------------------------------+

            Created At                 |
     Artifact ID                                                 |

+---------------+---------------+-------------------------------------------------------------------------------+

            2024-07-24 10:02:00.614225976 +0000 UTC |
0_RP0_CPU0-appmgr-1721815320614225976-58c4d59caf2e8bd971715eea491048673bf1af290fade112ad0ece654e285568|

            2024-07-24 10:02:00.614225976 +0000 UTC |
0_RP0_CPU0-appmgr-1721815320614225976-85f9ab33eccf4e48373865f00d8fd24f4e8e4901b49b7809297694f7b57864ea|

            2024-07-24 10:02:01.290718105 +0000 UTC |
                                                                                                        |

+----------------+----------------+-------------------------------------------------------------------------------+
```

Step 3 Monitor router health with gNOI **Get RPC** for specific components.

Example:

```
Router# /auto/appmgr/xrhealth/bin/gnoic -a ${MGMT_IP} --port 57400 --insecure -u
 cisco -p <password>
          healthz get --path "openconfig:/components/component[name=${OC_COMP}]"


            WARN[0000] "192.0.1.0" could not lookup hostname: lookup
198.51.100.0.in-addr.arpa.
            on 171.70.168.183:53: no such host
            target "192.0.1.0:57400":
            path     : openconfig:components/component[name=0_RP0_CPU0-appmgr]
            status   : STATUS_HEALTHY
            id       : 1721815321290718105
            acked    : false
            created  : 2024-07-24 10:02:01.290718105 +0000 UTC
            expires  : 2024-07-31 10:02:01.000290718 +0000 UTC
            Router# cd /tmp/
            Router/tmp#
          Router/tmp# /auto/appmgr/xrhealth/bin/gnoic -a ${MGMT_IP} --port 57400
 --insecure -u cisco -p <password>
            healthz artifact --id 0_RP0_CPU0-appmgr-1721815320614225976-
            58c4d59caf2e8bd971715eea491048673bf1af290fade112ad0ece654e285568
            WARN[0000] "192.0.1.0" could not lookup hostname: lookup
198.51.100.0.in-addr.arpa.
            on 171.70.168.183:53: no such host
            INFO[0000] 192.0.1.0:57400: received file header for artifactID:
0_RP0_CPU0-appmgr-1721815320614225976-
            58c4d59caf2e8bd971715eea491048673bf1af290fade112ad0ece654e285568
            id:
"0_RP0_CPU0-appmgr-1721815320614225976-58c4d59caf2e8bd971715eea491048673bf1af290fade112ad0ece654e285568"

            file:  {
            name:  "procmgr_event_20240724100205.tar.gz"
            path:
"/harddisk:/eem_ac_logs/xrhealth/artifacts/procmgr_event_20240724100205.tar.gz"
            mimetype:  "application/gzip"
            size:  3825
            hash:  {
            method:  SHA256
            hash:
"\xf5\xa5\xfe]\xc1~Y\xbc-\xe4\xfcJ\xe9r\xb4\x8e\xd2\xe6\x0fvk\x90\xf52\r\xe6\xda\x94\x83\x80\xc9\xff"

            }
            }
            INFO[0000] received 3825 bytes for artifactID:

0_RP0_CPU0-appmgr-1721815320614225976-58c4d59caf2e8bd971715eea491048673bf1af290fade112ad0ece654e285568

            INFO[0000] 192.0.1.0:57400: received trailer for artifactID:
```

```
0_RP0_CPU0-appmgr-1721815320614225976-58c4d59caf2e8bd971715eea491048673bf1af290fade112ad0ece654e285568

            INFO[0000] 192.0.1.0:57400: received 3825 bytes in total
            INFO[0000] 192.0.1.0:57400: comparing file HASH
            INFO[0000] 192.0.1.0:57400: HASH OK
```

The router health status is successfully monitored using gNOI healthz RPCs including List, Get, and artifact retrieval.

Example

For example, use the gNOI healthz artifact RPC to retrieve logs related to a specific health event using the artifact ID.

What to do next

After completing this task:

- Review the health status and logs to identify and resolve any component issues.

gRIBI

The gRPC Routing Information Base Interface (gRIBI) is a gRPC service that allows an external client to programmatically manage the routes in the Routing Information Base (RIB) of the router.

- **Traffic engineering:** Enables route control independent of traditional routing protocols.
- **External client support:** Clients can be local or remote and interact with the router via gRIBI RPCs.
- **OpenConfig integration:** Uses OpenConfig AFT YANG models and protobufs for route management.
- **Telemetry support:** Supports Event-driven Telemetry (EDT) for route monitoring.

Using he external client application, programatically you can add, edit, or remove the routing entries in the routing table. The client can be local to the router or hosted externally in the network management station.

gRIBI and OpenConfig protobuf files

All messages within the gRPC service definition are defined as protocol buffer (.proto) files. gRIBI OpenConfig proto file gribi.proto is located in the Github repository.

gRIBI's use of OpenConfig AFT for RIB management

The OpenConfig Abstract Forwarding Table (openconfig-aft.yang) data model defines a common abstraction of the RIB information and describes the forwarding entries installed on a network element. The AFT definitions are auto-generated from the OpenConfig AFT YANG schema. The protocol buffer (protobuf) representation of the OpenConfig AFT schema is available as the gribi_aft.proto file in the Github repository. gRIBI leverages this data model and the proto file to manage the RIB entries. This data model supports streaming Event-driven telemetry (EDT) data to check the installed routes in the Forwarding Information Base (FIB).

Routing preference hierarchy in gRIBI

The routes configured using static configuration have the highest preference, followed by routes configured using gRIBI, and then those configured using other protocols such as BGP or ISIS.

gRIBI RPCs

gRIBI supports these remote procedure calls (RPCs) to manage the routes in the RIB:

- **Modify Operation:** Provides a bidirectional streaming RPC that is used to issue modifications to the AFT in the form of a `ModifyRequest` message. The network element responds asynchronously with a `ModifyResponse` message based on each request.

  Messages:

  Supports route modifications on IPv4Entry, next hop group (NHG), next hop (NH) key objects. The traffic engineering controller ensures that specific ordering of gRIBI transactions is met—NH and NHG entries are sent before IPv4Entries. The NHGs and NHs are sent in a single `ModifyRequest` as repeated AFTOperation messages. The controller expects that the NHG or NH transactions are acknowledged before programming the corresponding IPv4Entry transactions. In the next hop entries, `decapsulate_header`, `encapsulate_header`, `interface_ref`, `ip_address`, `ip_in_ip` and `network_instance` attributes are supported.

Note IP forwarding, encapsulation and decapsulation operations are supported. MPLS operations are not supported.

SessionParameters:

- For client redundancy, only `SINGLE_PRIMARY` is supported. The primary client is designated based on the client with the highest election ID. Each AFTOperation carries an election ID. The server processes the AFTOperation if the election ID is the last advertised ID and is the highest ID on the server. If the election ID is less than the current election ID, the ID is ignored. If the election ID is equal, the client sending the message is accepted as a new master.

  For client persistence, only the `PRESERVE` option is supported, wherein the network device preserves the routes programmed by the gRIBI server's RIB, the system RIB, and the system FIB when the primary client disconnects.

  When gRIBI restarts, the configuration in the gRIBI server’s cache are sent to the router to reprogram the RIB. The election ID is reset to 0 upon restart.

The table shows the request and response messages exchanged between the client and server for the `Modify` RPCs.

Table 20: Request and Response Messages of Client-Server Exchange for Modify RPCs

Operation	Request (Client to Server)	Response (Server to Client)
Session setup	`Message ModifyResponse { SessionParameters }`	`Message ModifyResponse{ SessionParametersResult }`
Election ID	`Message Modify Request { election_id (int128) }`	`Message Modify Request { election_id (int128) }`

Operation	Request (Client to Server)	Response (Server to Client)
AFTOperation	`Message ModifyResponse { repeated AFTOperation operation }`	`Message ModifyResponse{ repeated AFTResult result }`

- **Get Operation**: Retrieves the content of the installed AFTs from the gRIBI daemon. The client requests for information using a `GetRequest` message, and the server responds with the set of currently installed entries via the `GetResponse` message. Once all entries have been sent, the server closes the RPC.

  Supports all operations defined in the `gribi.proto` file.

Note IPv4Entry.metdata is supported only in `Get` RPC, and not in AFT telemetry.

The table shows the request and response messages exchanged between the client and server for the `Get` RPCs.

Table 21: Request and Response Messages of Client-Server Exchange for Get RPCs

Operation	Request (Client to Server)	Response (Server to Client)
Get entries	`Message GetRequest { network_instance name [string] all AFTType aft }`	`Message GetResponse { repeated AFTEntry entry }`

- **Flush Operation**: Removes all AFT entries that are currently installed on the server using gRIBI RPCs. The client sends a `FlushRequest` message to remove all the entries, and the server responds with a `FlushResponse` message after the operation is complete.

  Supports all operations defined in the `gribi.proto` file.

  The table shows the request and response exchanged between the client and server for the `Flush` RPCs.

Table 22: Request and Response Messages of Client-Server Exchange for Flush RPCs

Operation	Request (Client to Server)	Response (Server to Client)
Flush entries	`Message FlushRequest { election id [uint 128] override [bool] network_instance name [string] all }`	`Message FlushResponse { timestamp [int64] }`

gRIBI configuration to modify routing entries

Configure gRIBI protocol to directly interact with the routers' RIB using RPCs. The gRIBI client sends messages to the RIB to add a route, delete a route, register next hop and next hop groups to manage the routes.

Before you begin

Ensure that you have configured the gRIBI client application.

Procedure

Step 1 Enable gRPC protocol on the router.

Example:

```
Router#configure
                        Router(config)#grpc
```

Step 2 Configure the port number and address family.

Example:

```
Router(config-grpc)#port 57345
                        Router(config-grpc)#address-family ipv4
```

The port number ranges from 57344 to 57999. The default port is 57400. If a port number is unavailable, an error is displayed.

Step 3 Verify that gRPC is enabled on the router.

Example:

```
Router#show grpc
                        Thu Feb 2 22:03:17.004 UTC

                        Address family              : dual
                        Port                        : 57777
                        DSCP                        : Default
                        TTL                         : 64
                        VRF                         : global-vrf
                        Server                      : enabled
                        TLS                         : enabled
                        TLS mutual                  : disabled
                        Trustpoint                  : none
                        Certificate Authentication  : disabled
                        TLS v1.0                    : enabled
                        Maximum requests            : 128
                        Maximum requests per user   : 10
                        Maximum streams             : 32
                        Maximum streams per user    : 32

                        TLS cipher suites
                        Default                   : none
                        Enable                    : none
                        Disable                   : none

                        Operational enable        : ecdhe-rsa-chacha20-poly1305
                        : ecdhe-ecdsa-chacha20-poly1305
                        : ecdhe-rsa-aes128-gcm-sha256
                        : ecdhe-ecdsa-aes128-gcm-sha256
                        : ecdhe-rsa-aes256-gcm-sha384
                        : ecdhe-ecdsa-aes256-gcm-sha384
                        : ecdhe-rsa-aes128-sha
                        : ecdhe-ecdsa-aes128-sha
                        : ecdhe-rsa-aes256-sha
                        : ecdhe-ecdsa-aes256-sha
                        : aes128-gcm-sha256
                        : aes256-gcm-sha384
```

```
                        : aes128-sha
                        : aes256-sha
                        Operational disable       : none

                        Listen address suites
                        Listen to Address         : ANY
```

Step 4 Manage the routing entries using gRIBI RPCs. In this example, you use the `Modify` RPC to add a next hop entry with IP address 192.0.2.0.

a) Configure the next hop parameters for the AFT message.

Example:

```
NextHop {
                                ip_address 192.0.2.0;
                                InterfaceRef {
                                interface [string]
                                subinterface [uint]
                                }
                                IPnIP {
                                dst_ip [string]
                                src_ip [string]
                                }
                                }
```

b) Set the next hop entry in the AFTOperation. In this example, you add the next hop IP address.

Example:

```
Message AFTOperation {
                                id
                                network_instance
                                Operation op
                                ADD
                                entry
                                Afts.NextHopKey next_hop
                                }
```

c) Initiate the `ModifyRequest` RPC using the AFTOperation message.

Example:

```
Message ModifyRequest {
                                repeated AFTOperation operation
                                }
```

The NHs are sent in a single `ModifyRequest` RPC as repeated AFTOperation messages.

d) View that the request is acknowledged in the gRIBI client.

Example:

```
gRIBIClient sent Modify message operation:{id:1 network_instance:"DEFAULT"
                                op:ADD next_hop:{index:1000
next_hop:{ip_address:{value:"192.0.2.0"}}} election_id:{low:3}}
```

Step 5 Verify the configuration performed using gRIBI RPC.

Example:

In this example, you verify the next hop IP address that you sent to the server through the `Modify` RPC is configured successfully.

```
Router#show gribi aft next-hop-groups
                    Thu Feb  2 00:34:08.104 UTC
                    100, Backup NHG: 1111
```

```
  [100, 2]: 192.0.2.40
  [200, 2]: 192.0.2.42
  [1111, 100]: (vrf REPAIR) (!)
1000
  [1100, 30]: 192.0.2.10
  [1200, 10]: 192.0.2.14
  [1000, 60]: 192.0.2.0
1111
  [1111, 100]: (vrf REPAIR)
2000
  [2000, 50]: 192.0.2.18
  [2100, 50]: 192.0.2.22
3000
  [3000, 10]: 192.0.2.26
4000
  [4000, 10]: Decapsulate IPv4(vrf DEFAULT)
```

After completing this task, the gRIBI service is configured and operational. You can verify its status using CLI commands.

Example

For example, after enabling the gRIBI service and configuring the port, you can use the 'show gribi' command to verify the service is running and listening on the expected port.

What to do next

After configuration, monitor the gRIBI service periodically to ensure it remains active and reachable. Adjust port settings or restart the service if issues are detected.

P4Runtime

P4Runtime is a protocol and framework that provides a control plane API for

- programming, and
- managing network devices.

A P4Runtime to manage traffic operations is a control plane specification to manage the data plane elements of a device. It defines the navigation and management of packets through data plane blocks using P4Runtime APIs. These data plane blocks can be managed to perform a set of traffic operations between the P4Runtime controller and the router:

- Send or receive packets using PacketOut and PacketIn I/O messages—`StreamMessageRequest`, `StreamMessageResponse` and `StreamError` messages.
- Elect the primary controller using the `MasterArbitrationUpdate` message.
- Read and write forwarding table entries, protocol headers, counters, and other P4 entities.

For more information about how controllers can connect to the router and program P4-defined functionalities, see P4RT specification.

Configure P4RT to manage packets

Configure P4RT to send or receive packets between one or more controllers and the router.

Before you begin

Ensure that the device supports P4RT and that you have administrative access to the CLI.

Procedure

Step 1 Enable P4Runtime.

Example:

```
Router#config
Router(config)#grpc
Router(config-grpc)#p4rt
Router(config-grpc-p4rt)#commit
```

Step 2 Assign a unique P4 numeric identifier to the required physical port on the router. The controller uses this port ID as an alias to identify the interface through which the packets are sent or received with ingress or egress metadata.

Example:

```
Router(config-grpc-p4rt)#interface HundredGigE0/0/0/24 port-id 3
Router(config-grpc-p4rt)#interface HundredGigE0/0/0/25 port-id 6
Router(config-grpc-p4rt)#interface HundredGigE0/0/0/26 port-id 7
```

The `port-id` is a unique 32-bit identifier. The range is 1 to 4294967039.

Step 3 Assign a unique P4 device identifier to each Network Processing Unit (NPU) in the system.

Example:

```
Router(config-grpc-p4rt)#location 0/0/CPU0 npu-id 0 device-id 1000000
Router(config-grpc-p4rt)#location 0/0/CPU0 npu-id 1 device-id 1000001
Router(config-grpc-p4rt)#location 0/1/CPU0 npu-id 0 device-id 1000002
Router(config-grpc-p4rt)#location 0/1/CPU0 npu-id 1 device-id 1000011
Router(config-grpc-p4rt)#commit
Router(config-grpc-p4rt)#end
```

The `device-id` is a unique 64-bit identifier. The range is 1 to 18446744073709551615. The `npu-id` represents a NPU identifier within a line card and the value ranges from 0 to 7.

The controller or the P4Runtime agent, which can be external or internal to the router, can use the `port-id` and `device-id` to inject packets and request to send certain packet types. For example, P4Runtime supports the ability to configure Access Control Lists (ACLs) in order to redirect packets with TTL value 1 to the controller. When the router receives a packet with that TTL value, the packet is sent to the controller with the details such as packet received from `device-id x`, `port-id y` and the packet is being sent to `port-id z`.

For more information about programming the router using P4Runtime, see P4RT specification.

P4RT is successfully configured and operational on the device.

What to do next

Verify P4RT telemetry and monitor the session status to ensure continued operation.

gNSI Acctz loggings

The gNSI accounting (Acctz) protocol collects and transfers AAA accounting records from a router to a remote collection service over a gRPC transport connection. It enhances visibility into user activity and service usage for improved network performance and security.

- **Protocol:** gNSI Acctz over gRPC
- **Purpose:** Logs CLI and gRPC service activity for AAA accounting
- **Transport:** gRPC-based streaming to remote collectors
- **Monitoring:** Real-time and historical record tracking

Table 23: Feature History Table

Feature Name	Release Information	Feature Description
gNSI Acctz Logging	Release 24.3.1	You can now log and monitor AAA (Authentication, Authorization, and Accounting) accounting of gRPC operations and CLI accounting data through gNSI Acctz for effective management of network for better performance and resource utilization. You can also configure the number of gNSI accounting records that can be streamed. Previously, you could monitor the AAA accounting data through syslog only. The feature introduces these changes: **CLI:** • • To view the specification of gNSI Accounting (Acctz) RPCs and messages, see the Github repository.

Starting from IOS XR software release 24.3.1, you can log gRPC AAA accounting data through gNSI accounting (Acctz). The gNSI Acctz data is logged, stored in accounting records, and send to gNSI client for monitoring purposes. These gNSI Acctz accounting records contain

- users' login or logout times,
- network access resources such as interface IP and port, and
- duration of each session.

The gNSI Acctz logging can be done using the RecordSubscribe() gRPC request to a router. For more information on the RecordSubscribe() RPC, see the GitHub repository.

gNSI Acctz Logging Stream Capacity

The gNSI Acctz logs are recorded in a queue, maintaining a history of the 10 most recent records. When the accounting queue is full and no gNSI Acctz collectors are connected, the stream drops the records. Besides the 10 records stored for streaming, up to 512 additional records are stored during processing. As new records arrive, the data stream continues until the gNSI session ends or an error occurs, such as a client disconnection

due to network issues or the server going down. If the server's output buffer remains full for an extended period, new records are dropped until the collector starts receiving them.

When the queue reaches its full capacity, the system automatically replaces the oldest records with the newest ones. The router then transmits this logged information through gNSI to gNSI client for real-time monitoring purposes. You can configure the queue size using the **grpc aaa accounting queue-size** command.

Supported Records for gNSI Acctz Logging

gNSI Acctz logging system supports Command and gRPC service records.

Table 24: CLI and gRPC Accounting Records

Command Services Accounting Records	gRPC Services Accounting Records
The command accounting records are generated for the commands executed in CLI mode and sent to gNSI Acctz collectors. The details logged include: • **Session Info:** remote/local IP addresses, remote/local ports, and channel ID. • **Authentication details**: Identity, privilege level, authentication status (PERMIT/DENY), and the cause of denial (if applicable). • **Command** and **Command status**: authentication status (PERMIT/DENY). • **Timestamp:** The time when the event was generated.	The gRPC accounting records are generated for the RPCs executed by gRPC services and sent to gNSI Acctz collectors. The details logged include: • **Session Info**: remote/local IP addresses, remote/local ports, and channel ID. • **Authentication details**: Identity and privilege level. • **RPC Service Request**: Service type, RPC name, payload, and configuration metadata. • **gRPC Service Status**: PERMIT/DENY. • **Timestamp:** The time at which the event was generated.

Default Behavior and Verification of gNSI Acctz Logging

By default, gNSI Acctz records are logged when the configuration is enabled. You can verify the gNSI Acctz using **show gnsi state**, **show gnsi acctz statistics**, and **show aaa accounting statistics** commands.

Configure gNSI Acctz logging

Monitor AAA information through gNSI Acctz logs.

Procedure

Step 1 Monitor gNSI state in the router.

Example:

```
Router# show gnsi state
            Wed Jun 26 09:26:39.035 UTC
            ----------------GNSI state--------------
            Global:
            Main Thread cerrno              : Success
            Acctz Thread cerrno             : Success
            State                           : Active
```

```
RDSFS State                     : Active
```

Step 2 Obtain gRPC port number.

Example:

```
show grpc
Tue Aug 13 14:21:50.995 IST

Server name                             : DEFAULT
Address family                          : dual
Port                                    : 57400

Service ports
gNMI                                  : none
P4RT                                  : none
gRIBI                                 : none

DSCP                                    : Default
TTL                                     : 64
VRF                                     :
Server                                  : enabled
TLS                                     : disabled
TLS mutual                              : disabled
Trustpoint                              : none
Certificate Authentication              : disabled
Certificate common name                 : ems.cisco.com
TLS v1.0                                : disabled
Maximum requests                        : 128
Maximum requests per user               : 10
Maximum streams                         : 32
Maximum streams per user                : 32
Maximum concurrent streams              : 32
Memory limit (MB)                       : 1024
Keepalive time                          : 30
Keepalive timeout                       : 20
Keepalive enforcement minimum time      : 300

TLS cipher suites
Default                               : none
Default TLS1.3                        : aes_128_gcm_sha256
: aes_256_gcm_sha384
: chacha20_poly1305_sha256

Enable                                : none
Disable                               : none

Operational enable                    : none
Operational disable                   : none
Listen addresses                        : ANY
```

Step 3 Configure gNSI queue size.

Example:

```
Router# configure
        Router(config)# grpc aaa accounting queue-size 30
        Router(config)# end
```

Step 4 Monitor gNSI Acctz statistics in the router.

Example:

```
Router# show gnsi acctz statistics
        Tue Aug 13 05:57:24.210 UTC
```

```
SentToAAA Queue:
Grpc services:
GNMI:    4998 sent, 0 dropped
GNOI:    0 sent, 0 dropped
GNSI:    2 sent, 0 dropped
GRIBI:   0 sent, 0 dropped
P4RT:    0 sent, 0 dropped
UNSPECIFIED: 0 sent, 0 dropped
Stats:
Total Sent: 5000
Total Drops: 0

Streams:
Grpc services:
GNMI:    4996 sent, 2 dropped
GNOI:    0 sent, 0 dropped
GNSI:    1 sent, 0 dropped
GRIBI:   0 sent, 0 dropped
P4RT:    0 sent, 0 dropped
UNSPECIFIED: 0 sent, 0 dropped
Stats:
Total Sent: 4997
Total Drops: 2
Cmd services:
CLI:     3 sent, 0 dropped
Stats:
Total Sent: 3
Total Drops: 0
Router#
```

Step 5 Provide port and IP address to the Acctz gNSI client.

Example:

```
acctz_collector -server_addr 192.0.2.111:57400 -username <user name> -password <passwod> -dieafter
600
```

```
--------------- gSNI Remote Collector ---------------
2024/08/25 22:59:13    Connecting to gNSI Server.
2024/08/25 22:59:13    gNSI Server connected.
2024/08/25 22:59:13    Started new acctz client.
2024/08/25 22:59:13    Initiate Acctz RecordSubscribe with server .
2024/08/25 22:59:13    Stream started
2024/08/25 22:59:13 Waiting for response from server.
```

Step 6 Verify the accounting record from the router.

Example:

gNSI Acctz RPC RecordSubscribe() response to the Acctz gRPC client

```
session_info:
{
local_address:"192.0.2.111"
local_port:57400
remote_address:"192.0.2.1"
remote_port:44374
ip_proto:6
user:
{
identity:"lab"
}
}
```

```
            timestamp:
            {
            seconds:1718971022 nanos:105825300
            }
            grpc_service:
            {
            service_type:GRPC_SERVICE_TYPE_GNSI
            rpc_name:"/gnsi.acctz.v1.AcctzStream/RecordSubscribe" payload_istruncated:true
            authz:
            {
            status:AUTHZ_STATUS_PERMIT
            }
            }
```

AAA Accounting Statistics

```
Router# show aaa accounting statistics
            Sat Aug 17 17:10:43.055 UTC
            Successfully logged events:
            Total events: 0
            XR CLI: 0
            XR SHELL: 0
            GRPC:
            GNMI: 0
            GNSI: 2
            GNOI: 0
            GRIBI: 0
            P4RT: 0
            SLAPI: 0
            NETCONF: 0
            SysAdmin:
            CLI: 0
            SHELL: 0
            Host:
            SHELL: 0

            Errors:
            Invalid requests: 0

            Max. records in buffer: 100
            Total records in buffer: 0
            Router#
```

After completing this task, gNSI Acctz logging is enabled and operational. The router streams accounting records to the Acctz gNSI client, and you can verify successful transmission and logging using CLI and gRPC responses.

Example

For example, after configuring the queue size and starting the Acctz client, the router streams accounting records such as user login sessions. You can verify this using the **show gnsi acctz statistics** and **show aaa accounting statistics** commands.

What to do next

After completing the configuration, monitor the gNSI Acctz logs periodically to ensure accounting records are being received without drops.

- Adjust the queue size if you observe dropped records in the statistics.
- Ensure the Acctz client remains connected and responsive.

Data logging with gNSI AcctzStream service

The data logging with gNSI AcctzStream is a service that allows clients to receive a continuous stream of accounting records from a target network device to track the changes made on that device.

- Replaces the existing bi-directional data streaming service, **Acctz**, with the new server-streaming service, **AcctzStream**.
- Ensures effective network optimization and resource utilization.
- Allows configuration of maximum memory allocated for cached accounting history records using the **grpc aaa accounting history-memomy** command.

Provides CLI enhancements and integration with gNSI AcctzStream RPCs and messages.

gNSI AcctzStream service

Starting from Cisco IOS XR Release 24.4.1, the **gNSI AcctzStream** server-streaming service is used to collect and transfer accounting records from a router to a remote collection service over a gRPC transport connection, similar to the deprecated **gNSI Acctz** protocol.

The collectors request for logs using the RecordSubscribe() gRPC from the gNSI AcctzSteam service running on the router. The logs are sent to the collectors through the RecordResponse() gRPC.

This feature has introduced the new **grpc aaa accounting history-memomy** command used to configure the maximum memory allocated for cached accounting history records. Use this command with the **grpc aaa accounting queue-size** configuration to effectively limit the EMSD memory used by cached accounting history records.

Configure gNSI AcctzStream logging

Monitor AAA information through gNSI AcctzStream logs.

Procedure

Step 1 Monitor gNSI state in the router.

Example:

```
Router# show gnsi state
            Thu Sep 12 12:06:44.035 UTC
            ----------------GNSI state--------------
            Global:
            Main Thread cerrno              : Success
            Acctz Thread cerrno             : Success
            State                           : Active
            RDSFS State                     : Active
```

Step 2 Obtain gRPC port number.

Example:

```
show grpc
            Thu Sep 12 13:23:06.022 UTC

            Server name                                : DEFAULT
            Address family                             : dual
            Port                                       : 57400

            Service ports
            gNMI                                     : none
            P4RT                                     : none
            gRIBI                                    : none

            DSCP                                       : Default
            TTL                                        : 64
            VRF                                        :
            Server                                     : enabled
            TLS                                        : disabled
            TLS mutual                                 : disabled
            Trustpoint                                 : none
            Certificate Authentication                 : disabled
            Certificate common name                    : ems.cisco.com
            TLS v1.0                                   : disabled
            Maximum requests                           : 128
            Maximum requests per user                  : 10
            Maximum streams                            : 32
            Maximum streams per user                   : 32
            Maximum concurrent streams                 : 32
            Memory limit (MB)                          : 1024
            Keepalive time                             : 30
            Keepalive timeout                          : 20
            Keepalive enforcement minimum time         : 300

            TLS cipher suites
            Default                                  : none
            Default TLS1.3                           : aes_128_gcm_sha256
            : aes_256_gcm_sha384
            : chacha20_poly1305_sha256

            Enable                                   : none
            Disable                                  : none

            Operational enable                       : none
            Operational disable                      : none
            Listen addresses                           : ANY
```

Step 3 Configure gNSI history memory.

Example:

```
Router# configure
            Router(config)# grpc aaa accounting history-memory 20
            Router(config)# end
```

Step 4 Configure gNSI queue size.

Example:

```
Router# configure
            Router(config)# grpc aaa accounting queue-size 30
            Router(config)# end
```

Step 5 Monitor gNSI Acctz statistics in the router.

Example:

```
Router# show gnsi acctz statistics
            Thu Sep 12 13:56:18.043 UTC
            SentToAAA Queue:
            Grpc services:
            GNMI:    4998 sent, 0 dropped
            GNOI:    0 sent, 0 dropped
            GNSI:    2 sent, 0 dropped
            GRIBI:   0 sent, 0 dropped
            P4RT:    0 sent, 0 dropped
            UNSPECIFIED: 0 sent, 0 dropped
            Stats:
            Total Sent: 5000
            Total Drops: 0

            Streams:
            Grpc services:
            GNMI:    4996 sent, 2 dropped
            GNOI:    0 sent, 0 dropped
            GNSI:    1 sent, 0 dropped
            GRIBI:   0 sent, 0 dropped
            P4RT:    0 sent, 0 dropped
            UNSPECIFIED: 0 sent, 0 dropped
            Stats:
            Total Sent: 4997
            Total Drops: 2
            Cmd services:
            CLI:     3 sent, 0 dropped
            Stats:
            Total Sent: 3
            Total Drops: 0
            Router#
```

Step 6 Provide port and IP address to the Acctz gNSI client.

Example:

```
acctz_collector -server_addr 192.0.2.111:57400 -username <user name> -password <passwod> -dieafter
600

            --------------- gSNI Remote Collector ---------------
            2024/08/25 22:59:13    Connecting to gNSI Server.
            2024/08/25 22:59:13    gNSI Server connected.
            2024/08/25 22:59:13    Started new acctz client.
            2024/08/25 22:59:13    Initiate Acctz RecordSubscribe with server .
            2024/08/25 22:59:13    Stream started
            2024/08/25 22:59:13 Waiting for response from server.
```

Step 7 Verify the accounting record from the router.

Example:

gNSI AcctzStream RPC RecordSubscribe() response to the Acctz gRPC client

```
            session_info:
            {
            local_address:"192.0.2.111"
            local_port:57400
            remote_address:"192.0.2.1"
            remote_port:44374
            ip_proto:6
            user:
            {
```

```
identity:"lab"
}
}
timestamp:
{
seconds:1718971022 nanos:105825300
}
grpc_service:
{
service_type:GRPC_SERVICE_TYPE_GNSI
rpc_name:"/gnsi.acctz.v1.AcctzStream/RecordSubscribe" payload_istruncated:true
authz:
{
status:AUTHZ_STATUS_PERMIT
}
}
```

AAA Accounting Statistics

```
Router# show gnsi accounting statistics
            Acctz History Buffer:
            Total record: 200
            Total history truncation: 10

            Cmd service records:
            Shell:   0
            Cli:     0
            Netconf: 0
            Grpc service records:
            GNMI:    0
            GNOI:    0
            GNSI:    0
            GRIBI:   0
            P4RT:    0
            History Snapshot:
            Max Memory size: 200 MB
            Memory used: 8 MB
            Max number of records: 100
            Total number of records present: 16

            gRPC Accounting Queue:
            Grpc services:
            GNMI:    0 sent, 0 dropped, 0 truncated
            GNOI:    0 sent, 0 dropped, 0 truncated
            GNSI:    0 sent, 0 dropped, 0 truncated
            GRIBI:   0 sent, 0 dropped, 0 truncated
            P4RT:    0 sent, 0 dropped, 0 truncated
            Stats:
            Queue buffer size: 100 MB
            Queue buffer used: 0 MB
            Queue size: 1
            Queue used: 0
            Queue enqueue: 0
            Queue dequeue: 0
            Queue drops: 0
            Queue max time: 0 usec
            Queue min time: 0 usec
            Queue avg time: 0 usec
            Errors:
            Queue init failure: 0
            Queue update failure: 0
            Queue dequeue failure: 0
            Queue invalid parameters: 0
            Queue memory limit: 0
            Queue size limit: 0
```

```
SendtoAAA Accounting Queue:
Grpc services:
GNMI:    0 sent, 0 dropped, 0 truncated
GNOI:    0 sent, 0 dropped, 0 truncated
GNSI:    0 sent, 0 dropped, 0 truncated
GRIBI:   0 sent, 0 dropped, 0 truncated
P4RT:    0 sent, 0 dropped, 0 truncated
Stats:
Queue buffer size: 100 MB
Queue buffer used: 0 MB
Queue size: 1
Queue used: 0
Queue enqueue: 0
Queue dequeue: 0
Queue drops: 0
Queue max time: 0 usec
Queue min time: 0 usec
Queue avg time: 0 usec
Errors:
Queue init failure: 0
Queue update failure: 0
Queue dequeue failure: 0
Queue invalid parameters: 0
Queue memory limit: 0
Queue size limit: 0
Cmd Accounting Queue:
Cmd services:
Shell:   0 sent, 0 dropped, 0 truncated
Cli:     0 sent, 0 dropped, 0 truncated
Netconf: 0 sent, 0 dropped, 0 truncated
Stats:
Queue buffer size: 100 MB
Queue buffer used: 0 MB
Queue size: 1
Queue used: 0
Queue enqueue: 0
Queue dequeue: 0
Queue drops: 0
Queue max time: 0 usec
Queue min time: 0 usec
Queue avg time: 0 usec
Errors:
Queue init failure: 0
Queue update failure: 0
Queue dequeue failure: 0
Queue invalid parameters: 0
Queue memory limit: 0
Queue size limit: 0
```

After completing this task, the router streams accounting records to the Acctz gNSI client. You can verify successful transmission and logging using CLI and gRPC responses.

Example

For example, after configuring the queue size and starting the Acctz client, the router streams accounting records such as user login sessions. You can verify this using the 'show gnsi acctz statistics' and 'show aaa accounting statistics' commands.

What to do next

After completing the configuration, monitor the gNSI Acctz logs periodically to ensure accounting records are being received without drops.

- Adjust the queue size if you observe dropped records in the statistics.
- Ensure the Acctz client remains connected and responsive.

gRPC network packet sampling interface

The gNPSI (gRPC Network Packet Sampling Interface) protocol is a network protocol that

- replaces UDP transport for flow data
- uses gRPC for reliable and secure transmission, and
- improves flow data reliability and security.

Table 25: Feature History Table

Challenges with traditional methods

Traditional UDP-based flow-data transport has several challenges:

- UDP transport can cause packet loss during network stress.
- UDP channels lack encryption and authentication, which can cause security risks.
- Discovery using VIP affects a large area if issues occur.
- The dial-out approach does not work when security policies require the collector to initiate connections.
- Encryption and proxy deployment are complex with UDP and multiple wire protocols.

Improved flow export with the gNPSI streaming model

Flow data was exported using traditional UDP-based mechanisms before the introduction of the gRPC Network Packet Sampling Interface (gNPSI) feature. These mechanisms can lose packets under network stress and do not provide built-in encryption or authentication.

This complexity makes secure deployment more difficult, especially when you use proxies, and reduces flexibility because dial-out models conflict with environments where collectors need to initiate connections.

With gNPSI, you can carry flow data over encrypted and authenticated gRPC sessions. This approach provides better reliability and security, lets your collectors initiate connections, and helps you deploy secure solutions with proxies more easily.

The new workflow uses a controller-driven streaming model. In this model, EMSd coordinates with your linecard export processes to give you more control, resiliency, and visibility for flow data delivery.

Note Only version 0.1.0 of the gNPSI protocol is supported.

Benefits of gNPSI

gNPSI provides these benefits:

- Reduces packet loss during network stress.
- Encrypts and authenticates the data channel.
- Limits the impact area during failures or changes.
- Supports secure, collector-initiated connections.
- Simplifies the deployment of encrypted and proxy based solutions.

Usage guidelines for gNPSI

To use gNPSI effectively, follow these best practices:

- Apply the NetFlow or sFlow configuration with explicit gNPSI settings on the router.
- Use the correct export process (nfsvr).
- Ensure that controllers connect with the gNPSI protocol to start streaming.
- Monitor your controller connections for streaming status.
- Use supported show commands to view export and drop statistics.

Restrictions for gNPSI

gNPSI has these limitations:

- You can use only version 0.1.0 of the gNPSI protocol.
- Use the trace commands and the show commands. You cannot use the debug commands.
- You can use only the published OpenConfig gNPSI telemetry model.

How gNPSI data and control flow work

Summary

Use the gNPSI data and control flow process to collect, manage, and stream network flow data from routers to controllers efficiently. Configure routers for NetFlow or sFlow using gNPSI. Initiate export and streaming operations.

Establish communication between linecards and the Route Processor (RP), and manage packet flow through the EMSd service to external controllers. You can start and stop streaming based on controller connections and system events. This approach provides reliable and controlled data delivery for your network monitoring and analysis tasks.

Workflow

1. The gNPSI protocol processes flow data using these steps:

- The router applies NetFlow or sFlow configuration with gNPSI settings.
- The export process (nfsvr) starts on the linecards.
- The export process establishes a TCP connection to the EMSd process on the RP using a new backplane VLAN.
- Sampling begins, but packets are dropped until a gNPSI request is received.
- When EMSd on the RP receives a gNPSI streaming request from a controller, it instructs the NetFlow process on the linecards to start streaming.
- The NetFlow process streams packets to EMSd.
- EMSd encapsulates the packets in gNPSI gRPC messages and streams them to all connected controllers.
- If all controllers disconnect, EMSd tells the NetFlow process to stop streaming.
- After a RP failover or EMSd restart, the NetFlow process re-establishes the session.

2. Use gNPSI to manage code and control flow in these ways:
   - EMSd creates new external gNPSI gRPC Service code to encapsulate flow packets.
   - EMSd creates new internal gNPSI TCP Service code to listen for flow packets from linecards.
   - NetFlow or sFlow configuration triggers the appropriate export and control processes.
   - EMSd and NetFlow processes coordinate the start and stop of packet streaming based on controller connections.

Configure gNPSI

Collect and export traffic data using NetFlow or sFlow with gNPSI. After enabling the required router services, apply your flow-export settings, confirm that the export process is active, and verify that EMSd and controller connections are operating correctly.

gNPSI allows you to monitor and export traffic on routers with NetFlow and sFlow. You can manage configurations by using CLI or YANG models. EMSd and controller integration support efficient streaming and telemetry.

Before you begin

- Decide whether to use the CLI or YANG models for configuration.
- Confirm EMSd process is running and that the appropriate port is available for NetFlow or sFlow packets.

Follow these steps to configure gNPSI:

Procedure

Step 1 Configure NetFlow or sFlow with gNPSI settings by using the CLI or supported YANG models.

a) Apply your required gRPC configuration.

Example:

```
Add/edit any other grpc config required
```

b) Define the flow exporter, sampler, and monitor maps.

Example:

```
flow exporter-map EXP-1
version sflow v5
export protocol gnpsi
exit
sampler-map Sample
random 1 out-of 8000
exit
flow monitor-map fmm
record sflow
exporter EXP-1
```

c) Apply the monitor and sampler to the intended interface.

Example:

```
int FourHundredGigE0/0/0/0
            ! link towards peer where netflow is enabled
            no shutdown
            ipv4 address 192.0.2.1/24
            ipv6 address 2001:DB8::20:0:0:1/120
            flow datalink monitor fmm sampler Sample ingress
```

Refer to the OpenConfig gNPSI (OC-gNPSI) model if telemetry support is needed.

Step 2 Apply and activate the configuration on the router.

Step 3 Verify that the export process starts successfully on the router linecards.

Step 4 Ensure the EMSd process is running and listens for incoming export packets.

Step 5 Establish a controller connection to the router to initiate streaming if required by your architecture.

Step 6 Use available show commands on the RP and linecards to monitor export statistics and packet drop counts.

- **show flow gnpsi session**
- **show flow gnpsi statistics**
- **show gnpsi connections**
- **show gnpsi sessions**
- **show gnpsi stats**

Step 7 Extend trace in EMSd and NetFlow or sFlow processes as needed for monitoring.

---

gNPSI is configured and operational. The router exports monitored traffic data using NetFlow or sFlow. You can monitor export and drop statistics with show commands. Telemetry is enabled based on your configuration. EMSd and controller processes run as required.

gRIBI default route resolution without recirculation

A gRIBI default route resolution without recirculation is a routing mechanism that

- maintains default route behavior within the same VRF
- eliminates bandwidth impact by removing the recycle process, and
- uses controller-driven programming instead of configuration changes.

Table 26: Feature History Table

Benefits of gRIBI default route resolution without recirculation

These are the benefits of gRIBI default route resolution without recirculation:

- Prevents bandwidth reduction that previously resulted from the recycle process.
- Maintains the original behavior of default routing in your network.
- Enables control by moving from static configuration to controller-driven programming.
- Requires only minimal additional memory for an action type attribute.

Usage guidelines for gRIBI default route resolution without recirculation

- Use controller-driven programming to manage default routes.
- Use controller-driven programming to remove bandwidth impact instead of configuration-based changes.

Restrictions for gRIBI default route resolution without recirculation

- This feature is not supported on Q100 hardware due to lack of VRF redirect support.
- The Platform Independent (PI) Forwarding Information Base (FIB) applies the special LOOKUP action only when the LOOKUP next hop is the primary path.
- The PI FIB does not apply the special LOOKUP action if the LOOKUP next hop is a backup path, even after a switchover.

How gRIBI default route resolution without recirculation works

When default route lookups cross VRF boundaries, you may experience reduced bandwidth and degraded performance due to recirculation. You can use static route or VRF fallback configurations to look up default routes without recirculation, but these options do not provide flexibility at runtime.

Summary

gRIBI default route resolution enhances routing performance. It eliminates the need for packet recirculation when default route lookups cross VRF boundaries. This process uses a controller-driven architecture and runtime flags to achieve bandwidth-efficient default routing.

This process involves these key components:

- The initial VRF: This VRF contains the packet’s prefix and is the initial point of route lookup.

- The user network uses default routes that previously required a recirculation step, leading to bandwidth degradation.
- Controller (gRIBI): Manages routing behavior through events and runtime programming.
- PI FIB handles prefix and next-hop group (NHG) creation or update events. It also programs flags and manages LOOKUP actions.
- PD FIB: Processes static and controller-driven configurations to manage VRF resolution and redirect objects.
- SHLDI (Shared Local Device Interface) facilitates the transfer of action types, attributes, and VRF information to PD FIB.
- TEP (Tunnel End Point) delivers redirect VRF information and is involved in LOOKUP actions.
- RIB (Routing Information Base) initiates prefix and NHG events for processing via the Service Layer Application Programming Interface (SL-API).

Workflow

The process includes these stages:

1. Prefix lookup and default route evaluation:
    - The system checks the incoming packet's prefix in the initial VRF.
        - If no route exists for the prefix, the default route is used, which triggers a lookup into another VRF.
2. User requirement identification:
    - The user setup uses default routes that previously required a recycle step and that suffered bandwidth degradation.
    - The user's objective is to maintain default routing efficiency and ensure the solution is controller-driven.
3. Controller-driven architecture deployment:
    - The design employs controller-driven programming using gRIBI to manage routing behavior.
    - Runtime flags and actions replace the previous static configuration approach.
4. PI FIB event processing:
    - PI FIB receives prefix and NHG creation/update events from RIB via the SL API and the gRIBI controller.
    - PI FIB sends flags for prefix and NHG events to PD FIB.
    - The system sets an explicit DEFAULT flag for default routes.
    - If an NHG is marked with a LOOKUP action, the LOOKUP next hop becomes the primary path.
5. PD FIB handling for VRF resolution:

- In static configurations, PD FIB identifies parent objects with a table resolution flag, saves the VRF table value, and creates or resolves VRF redirect objects in hardware.
- In controller-driven flows, PD FIB obtains equivalent information via SHLDI and TEP.
- If SHLDI uses a TEP with a LOOKUP action, PD FIB replicates the behavior of the static configuration in hardware.

6. Runtime action and memory impact:
   - The action type variable is stored in `SHLDI FIB HAL` attribute and uses only 8 bits, resulting in minimal memory impact.
7. Detailed controller-driven route programming:
   - PD FIB receives the SHLDI object instead of LWLDI.
   - After PI FIB programs SHLDI, PD FIB stores the action type. It then retrieves redirect VRF information from the next-hop TEP.
   - During route programming, PD FIB accesses the action type and redirect VRF from SHLDI only if the route is a default route and the SHLDI action is LOOKUP.
   - During SDK operation, PD FIB programs the default route to use the redirect VRF, avoiding the recycle process.

Configure gRIBI default route resolution without recirculation

Set up gRIBI to resolve default routes without using recirculation.

This task shows you how to configure gRIBI to resolve default routes without using recycle. The configuration works with supported platforms and programs both routes and next-hop groups (NHG).

Before you begin

- Verify that your platform supports VRF redirect.
- Make sure your system does not use Q100 hardware.
- Make sure you can access the gRIBI controller and SDK.

Follow these steps to configure gRIBI default route resolution without recirculation:

Procedure

Step 1 Program a default route using the gRIBI controller.

Example:

```
! Enable gRIBI on the device
  router static
   gribi
    admin-state enable
    mode all-primary
    persistence true
```

```
   fib-ack true
  exit
 !

 ! Configure gRIBI controller session
 grpc
  service-layer
   gribi
    controller GRIBI-CTRL
     address 203.0.113.10 port 57400
     admin-state enable
    exit
   exit
  exit
 !

 ! (Optional) Ensure interface is ready for next-hop reachability
 interface HundredGigE0/0/0/1
  ipv4 address 192.0.2.254 255.255.255.0
  no shutdown
 exit
 !
```

This default route should point to a primary NHG.

Step 2 Configure the next-hop within this primary NHG to include a LOOKUP action.

Example:

```
grpc
   service-layer
    gribi
     admin-state enable
     mode all-primary
     fib-ack true
    exit
   exit
  !

  ! Optional: ensure local interface for recursion/lookup
  interface HundredGigE0/0/0/0
   ipv4 address 192.0.2.254 255.255.255.0
   no shutdown
  exit
  !
```

This LOOKUP action should specify the target VRF where the default route resolution should occur.

Step 3 Validate the configuration to ensure that the default route resolves to the specified redirect VRF without packet recirculation.

Example:

```
Router#show fib vrf Blue 203.0.113.0/24 detail
FIB entry for VRF: Blue
Prefix: 203.0.113.0/24
Route State: Active
Route Type: Connected
Flags: 0x0

Nexthop(s):
• Nexthop 1:
 • IP Address: 203.0.113.1
 • Interface: GigabitEthernet0/0/0/0
 • Afi: IPv4
 • Weight: 0
 • Flags: Direct, Hardware-Programmed
```

```
• Nexthop 2:
 • IP Address: 198.51.100.1
 • Interface: GigabitEthernet0/0/0/1
 • Afi: IPv4
 • Weight: 0
 • Flags: Direct, Hardware-Programmed


Load-split: 1/1
Packets Forwarded: 0
Bytes Forwarded: 0
Updated: 00:03:17 ago
```

You configure gRIBI default routing to resolve default routes without recirculation by using redirect VRF on compatible hardware and software.

What to do next

- Confirm that the VRF redirect is functioning and that no recirculation path is used.
- Review any logs or system reports to ensure that configuration changes have taken effect.