Nokia Service Router Linux
7220 Interconnect Router
7250 Interconnect Router

Release 26.3

P4Runtime Guide

3HE 22260 AAAA TQZZA
Edition: 01
March 2026

Nokia is committed to diversity and inclusion. We are continuously reviewing our customer documentation and consulting with standards bodies to ensure that terminology is inclusive and aligned with the industry. Our future customer documentation will be updated accordingly.

Table of contents

1 About this guide

This document describes Nokia Service Router Linux (SR Linux) support for P4Runtime and includes procedures for configuring SR Linux to operate with P4Runtime clients.

Note:
This manual covers the current release and may also contain some content to be released in later maintenance loads. See the *SR Linux Software Release Notes* for information about features supported in each load.

Configuration and command outputs shown in this guide are examples only; actual displays may differ depending on supported functionality and user configuration.

1.1 Precautionary and information messages

The following are information symbols used in the documentation.

DANGER: Danger warns that the described activity or situation may result in serious personal injury or death. An electric shock hazard could exist. Before you begin work on this equipment, be aware of hazards involving electrical circuitry, be familiar with networking environments, and implement accident prevention procedures.

WARNING: Warning indicates that the described activity or situation may, or will, cause equipment damage, serious performance problems, or loss of data.

Caution: Caution indicates that the described activity or situation may reduce your component or system performance.

Note: Note provides additional operational information.

Tip: Tip provides suggestions for use or best practices.

1.2 Conventions

The Nokia SR Linux documentation uses the following command conventions:

- **Bold** type indicates a command that the user must enter.
- Input and output examples are displayed in `Courier` text.
- A vertical bar (|) indicates a mutually exclusive argument.
- Square brackets ([ ]) indicate optional elements.
- Braces ({ }) indicate a required choice. When braces are contained within square brackets, they indicate a required choice within an optional element.

- *Italic* type indicates a variable.

The following table outlines platform grouping conventions used in the SR Linux documentation suite.

Note: Some platforms in the 7250 IXR support mixed systems. For more information about mixed system support, see "Chassis types" in the *Configuration Basics Guide*.

Table 1: Platform grouping legend

Platform group	Description
7215 IXS	7215 IXS-A1
7220 IXR	All 7220 IXR platforms
7220 IXR-D*x*	7220 IXR-D1, 7220 IXR-D2, 7220 IXR-D2L, 7220 IXR-D3, 7220 IXR-D3L, 7220 IXR-D4, 7220 IXR-D5
7220 IXR-H*x*	7220 IXR-H2, 7220 IXR-H3, 7220 IXR-H4, 7220 IXR-H4-32D, 7220 IXR-H5-32D, 7220 IXR-H5-64D, 7220 IXR-H5-64O
7250 IXR[1]	7250 IXR platforms
7250 IXR Gen 2	7250 IXR-6, 7250 IXR-10
7250 IXR Gen 2c+	7250 IXR-6e with IMM2, 7250 IXR-10e with IMM2, 7250 IXR-X1b, 7250 IXR-X3b
7250 IXR Gen 3	7250 IXR-6e with IMM3, 7250 IXR-10e with IMM3, 7250 IXR-18e, 7250 IXR-X4
7250 IXR-6e/10e (mixed system)	7250 IXR-6e (mixed system)[2], 7250 IXR-10e (mixed system)[2]
7730 SXR	7730 SXR-1-32D, 7730 SXR-1d-32D, 7730 SXR-1x-44S

1 References to the 7250 IXR platform group may be appended with (including mixed systems) or (excluding mixed systems) to indicate mixed system support.

2 References to this platform as part of 7250 IXR (mixed system) indicate mixed system support of 7250 IXR Gen 2c+ (IMM2) and 7250 IXR Gen 3 (IMM3). That is, the 7250 IXR-6e and 7250 IXR-10e can hold and support both IMM2 and IMM3 at the same time.

2 What's new

There have been no updates in this document since it was last released.

3 Overview

Programming Protocol-Independent Packet Processors (P4) is an open-source language for programming the data plane on networking devices. P4Runtime is an API for controlling the data plane on devices defined in a P4 program. The P4 language and P4Runtime specification are maintained at p4.org.

The SR Linux eXtensible Data Path (XDP) is not programmed in P4. However, SR Linux is packaged with a fixed P4 program named `p4info`, installed in `/etc/opt/srlinux/p4rt/p4info.pb.txt` and also available on GitHub, that provides support for marking packets for trapping to a P4Runtime client via PacketIn messages, and transmitting packets from the P4Runtime client to an interface on the device via PacketOut messages. The following fields can be used to mark frames for extraction:

- VLAN ID
- Ethertype
- TTL

This could for example be used to redirect traceroute packets with TTL=0, TTL=1, or TTL=2 to a P4runtime client, so they can be enriched with information that is not visible to the device for the following ACL rules:

- TTL=0, IPv4 (ethertype 0x0800)
- TTL=1, IPv4 (ethertype 0x0800)
- TTL=2, IPv4 (ethertype 0x0800)
- TTL=0, IPv6 (ethertype 0x86DD)
- TTL=1, IPv6 (ethertype 0x86DD)
- TTL=2, IPv6 (ethertype 0x86DD)

This use case is applicable only to transit traffic. In the SR Linux implementation, packets destined for a local node address (a local interface IP, where the interface can be Ethernet, LAG, or loopback) are never sent to the P4Runtime client:

- Any packet that has a destination IP that is a local interface IP is not subject to P4 pipeline processing.
- Any packet that has a remote destination IP (not belonging to a local node) follows the P4 logic (if there is a matching rule), including packets with TTL=1.

Another use case is to use a free ethertype to allow the P4Runtime client to transmit and receive packets on all internal links on all devices in a network as a means of topology discovery.

To accommodate these use cases, SR Linux runs a `grpc_server` service named `p4rt` that provides the interface between P4Runtime clients and SR Linux.

3.1 SR Linux p4rt service

SR Linux supports packet input/output to P4Runtime clients through a `grpc_server` service named `p4rt`. This `grpc_server` process exposes instances of P4Runtime RPCs that P4Runtime clients can connect to, with mandatory arbitration to elect a single P4Runtime client as the primary (see P4Runtime client arbitration).

Instead of running multiple processes, SR Linux runs a single process with multiple sockets.

See Configuring SR Linux for P4Runtime for information about configuring the `p4rt grpc_server` service.

4 Supported P4Runtime RPCs

P4 allows an operator to define a set of tables that use match criteria to match to a set of defined actions. The P4 program is pushed to a supporting device that recreates the fast path pipeline based on the new tables. A P4RT client can interact with the pipeline using P4RT RPCs. The SR Linux implementation does not support the creation of new tables, but instead uses a fixed P4 program that exposes tables to allow programming of ACL rules to trap traffic toward a controller.

To support the use cases described in Overview, SR Linux supports the following P4Runtime RPCs:

- `StreamChannel` RPC
  Allows for session management, client arbitration, and packet injection/extraction. See StreamChannel RPC for information about how SR Linux handles `StreamChannel` RPC messages.
- `Write` RPC
  Injects rules to select packets to extract to the slow path.
- `Read` RPC
  Provides uniformity with the `Write` RPC; that is, it gives the controller the ability to query which rules have been programmed.
- `SetForwardingPipelineConfig` RPC
  Used for pushing a P4 program from a P4Runtime controller to a target SR Linux device. The P4 program must be the same program that is distributed with SR Linux.
- `GetForwardingPipelineConfig` RPC
  Used for reading out the P4 forwarding pipeline configuration.
- `Capabilities` RPC
  Allows a P4Runtime client to discover the capabilities of a target device, including the P4Runtime API version implemented by the SR Linux `p4rt` service.

Note: No programming of the SR Linux data path is performed using P4 (outside of enabling of packet extraction for frames matching ACL match criteria via the Write RPC), nor is the data path modeled using P4.

4.1 StreamChannel RPC

The P4Runtime StreamChannel RPC is used for session management, arbitration, and packet I/O. It has the following definition:

```
rpc StreamChannel(stream StreamMessageRequest) returns (stream StreamMessageResponse)
```

The StreamChannel RPC has two top-level messages, `StreamMessageRequest` and `StreamMessageResponse`.

The following is an example of the `StreamMessageRequest` message.

```
message StreamMessageRequest {
    oneof update {
        MasterArbitrationUpdate arbitration = 1;
```

```
        PacketOut packet = 2;
        DigestListAck digest_ack = 3;
        .google.protobuf.Any other = 4;
    }
}
```

The SR Linux `p4rt` service uses fields in the `StreamMessageRequest` messages as follows:

- `MasterArbitrationUpdate`
  P4Runtime clients send `StreamMessageRequest` messages to the `p4rt` service on the SR Linux device to perform arbitration via `MasterArbitrationUpdate` messages. See P4Runtime client arbitration.
- `PacketOut`
  The P4Runtime client selected as primary transmits packets via `PacketOut` messages. See PacketIn and PacketOut messages.
- The `DigestListAck` and `.google.proto.Any` messages within the `digest_ack` and `other` fields are not supported by SR Linux.

The following is an example of the `StreamMessageResponse` message:

```
message StreamMessageResponse {
    oneof update {
        MasterArbitrationUpdate arbitration = 1;
        PacketIn packet = 2;
        DigestList digest = 3;
        IdleTimeoutNotification idle_timeout_notification = 4;
        .google.protobuf.Any other = 5;
        // Used by the server to asynchronously report errors which occur when
        // processing StreamMessageRequest messages.
        StreamError error = 6;
    }
}
```

The SR Linux `p4rt` service uses fields in the `StreamMessageResponse` messages as follows:

- `MasterArbitrationUpdate`
  The `p4rt` service sends a `StreamMessageResponse` message to a P4Runtime client to respond to an arbitration request via a `MasterArbitrationUpdate` message. See P4Runtime client arbitration
- `PacketIn`
  The `p4rt` service transmits packets to the primary P4Runtime client via `PacketIn` messages. See PacketIn and PacketOut messages.
- `StreamError`
  The `p4rt` service transmits `StreamError` message within the `error` field based on any errors occurring. See Stream Error Reporting.
- The `DigestList`, `IdleTimeoutNotification`, `.google.proto.Any` messages within the `digest`, `idle_timeout_notification`, and `other` fields are not are not supported by SR Linux.

4.1.1 P4Runtime client arbitration

P4Runtime client arbitration refers to the process by which a single P4Runtime controller becomes the "primary", with any other controllers serving as backups. Only one P4Runtime controller can be the primary. In SR Linux, P4Runtime arbitration works as described in the P4 Runtime specification: the

primary is elected based on the `election_id` within the `MasterArbitrationUpdate` message in the `StreamChannel` RPC.

The `MasterArbitrationUpdate` message is defined as follows:

```
message MasterArbitrationUpdate {
    uint64 device_id = 1;
    // The role for which the primary client is being arbitrated. For use-cases
    // where multiple roles are not needed, the controller can leave this unset,
    // implying default role and full pipeline access.
    Role role = 2;
    // The stream RPC with the highest election_id is the primary. The 'primary'
    // controller instance populates this with its latest election_id. Switch
    // populates with the highest election ID it has received from all connected
    // controllers.
    Uint128 election_id = 3;
    // Switch populates this with OK for the client that is the primary, and
    // with an error status for all other connected clients (at every primary
    // client change). The controller does not populate this field.
    .google.rpc.Status status = 4;
}

message Role {
    // Uniquely identifies this role.
    string name = 3;
    // Describes the role configuration, i.e. what operations, P4 entities,
    // behaviors, etc. are in the scope of a given role. If config is not set
    // (default case), it implies all P4 objects and control behaviors are in
    // scope, i.e. full pipeline access. The format of this message is
    // out-of-scope of P4Runtime.
    .google.protobuf.Any config = 2;
}
```

Only the primary controller is allowed to send `PacketOut` messages and receive `PacketIn` messages for a specific ASIC.

In the event a client with the highest `election_id` for a given `device_id` disconnects, all clients with active sessions to the same `device_id` are sent an `Advisory` message indicating that the primary has gone down. On receiving this message clients increment their `election_id` to be greater than the `election_id` of the previous master. Until this is done no primary will exist for the `device_id`; there is no automatic re-selection of a primary with the next highest `election_id`. Once a new election has occurred, this results in `PacketIn` messages being forwarded to the new master, and the `p4rt` service only accepting `PacketOut` from the new master.

4.1.2 PacketIn and PacketOut messages

`PacketIn` messages are sent by the `p4rt` service to the P4Runtime client within `StreamMessage Response` messages, and `PacketOut` messages are sent by the P4Runtime client to the `p4rt` service within `StreamMessageRequest` messages.

An ACL pushed to the device via the `Write` RPC is used to mark/extract packets for `PacketIn` handling by the `p4rt` service.

`PacketIn` and `PacketOut` messages are defined as follows:

```
// Packet sent from the controller to the switch.
message PacketOut {
    bytes payload = 1;
```

```
    // This will be based on P4 header annotated as
    // @controller_header("packet_out").
    // At most one P4 header can have this annotation.
    repeated PacketMetadata metadata = 2;
}

// Packet sent from the switch to the controller.
message PacketIn {
    bytes payload = 1;
    // This will be based on P4 header annotated as
    // @controller_header("packet_in").
    // At most one P4 header can have this annotation.
    repeated PacketMetadata metadata = 2;
}

message PacketMetadata {
    // This refers to Metadata.id coming from P4Info ControllerPacketMetadata.
    uint32 metadata_id = 1;
    bytes value = 2;
}
```

The `PacketIn` and `PacketOut` messages require that the interface a packet is received on or transmitted out of be uniquely identified. To do this, unique identifiers are configured for SR Linux interfaces. See Identifying interfaces to P4Runtime.

5 Configuring SR Linux for P4Runtime

To configure SR Linux for P4Runtime, you perform the following configuration tasks:

- Configure a TLS profile to secure communication with P4Runtime clients. See the *SR Linux Configuration Basics Guide* for information about configuring TLS profiles.
- Configure interface identifiers. To allow P4Runtime clients to reference specific interfaces in `PacketIn` and `PacketOut` messages, you configure per-interface identifiers consisting of a port ID and device ID. See Identifying interfaces to P4Runtime.
- Configure settings for the P4Runtime server, including idle timeout, session limits, and connection rate limiting. See Configuring P4Runtime server settings.
- Configure UNIX-socket specific settings for the P4Runtime server. See Configuring the P4Runtime server for UNIX sockets.
- Configure service authorization for the `p4rt` interface type. See the *SR Linux Configuration Basics Guide*.

5.1 Identifying interfaces to P4Runtime

The `PacketIn` and `PacketOut` messages within the P4Runtime `StreamChannel` RPC require that the interface a packet is received on or transmitted out of be uniquely identified. To do this, you configure a unique per-interface identifier, which is a tuple consisting of the following:

- A chassis-unique port identifier (known as the `interface_id`). This identifier can be manually configured, or if it is not, the system `ifIndex` for the interface is used by default.
- A chassis-unique device identifier that indicates the specific line card and ASIC with which the port is associated (known as the `device_id`)

For example, to identify interface `ethernet 1/1` to P4Runtime, you can configure the `interface_id` for the `ethernet 1/1` port, and configure a `device_id` identifying the line card and ASIC associated with the `ethernet 1/1` port.

The P4Runtime client uses a lookup table consisting of the `device_id,interface_id` tuple → interface-name (as specified by the device) to translate where packets are to be sent to, or populate where a packet was received.

5.1.1 Configuring a port identifier for P4Runtime

Procedure

The `interface_id` part of the `device_id,interface_id` tuple uniquely identifies a port in the SR Linux chassis to a P4Runtime client. You can configure the value for `interface_id`. If you do not configure a value for `interface_id`, the port's `ifIndex` value is used by default.

Example

```
--{ candidate shared default }--[  ]--
# info with-context interface ethernet-1/1 p4rt
    interface ethernet-1/1 {
        p4rt {
            id 2002
        }
    }
```

5.1.2 Configuring a device identifier for P4RT

Procedure

The `device_id` identifies a specific line card and ASIC in the chassis. P4Runtime uses the combination of the `device_id` and `interface_id` to identify the specific interface that a packet was received on (in `PacketIn` messages). Note that for identifying the interface that a packet is to be sent via (in `PacketOut` messages), only the `interface_id` is used.

There is no default `device_id` for a line card / ASIC; you must configure the `device_id` value to be used by P4Runtime.

Example

```
--{ candidate shared default }--[  ]--
# info with-context platform linecard 1 forwarding-complex 0
    platform {
        linecard 1 {
            forwarding-complex 0 {
                p4rt {
                    id 10001
                }
            }
        }
    }
```

5.2 Configuring P4Runtime server settings

Procedure

You can configure settings for the specified P4Runtime server. These settings apply to the specified network-instance where the P4Runtime server is enabled, and to UNIX sockets if enabled. You can configure the following:

- Whether to administratively enable the P4Runtime server
- Limit the number of connection attempts per minute by P4Runtime clients
- Limit the number of P4Runtime RPC connections that can be active at one time
- Idle-timeout in seconds for P4Runtime clients
- The network instance on which to administratively enable the P4Runtime server
- The gRPC service to enable; in this case, **p4rt**
- The port the P4Runtime server listens to for the network-instance. By default, this is TCP port 57400.

- IP addresses the P4Runtime server listens on within the network-instance
- TLS profile to secure communication between P4 Runtime clients and SR Linux for the network-instance
- Whether username/password authentication is used for each P4Runtime RPC request
- gRPC trace options (**[common | grpc | request | response]**)
- The YANG models to use when the origin field is not present in requests (default: **native**)

Example

```
--{ candidate shared default }--[  ]--
# info with-context system grpc-server mgmt-test
    system {
        grpc-server mgmt-test {
            admin-state enable
            timeout 14400
            rate-limit 120
            session-limit 40
            metadata-authentication true
            tls-profile tls-profile-1
            network-instance mgmt
            port 9559
            services [
                p4rt
            ]
            source-address [
                192.168.0.1
            ]
        }
    }
```

5.3 Configuring the P4Runtime server for UNIX sockets

Procedure

You can configure whether to administratively enable the P4Runtime server for UNIX sockets.

Example

The following example configures settings for the P4Runtime server for UNIX sockets:

```
--{ candidate shared default }--[  ]--
# info with-context system grpc-server mgmt-test
    system {
        grpc-server mgmt-test {
            admin-state enable
            metadata-authentication true
            tls-profile test-tls
            unix-socket {
                admin-state enable
            }
        }
    }
```

5.4 Configuring a chassis ID for the SR Linux device

Procedure

P4Runtime controllers sometimes model the chassis as a set of IDs, with the chassis having an ID, and each forwarding class having its own ID.

If necessary, you can assign a chassis ID to the SR Linux device. The chassis ID is not used by SR Linux for any purpose; it is intended for use by other controllers that may require a chassis ID on the device.

Example

The following example configures a chassis ID for the SR Linux device:

```
--{ candidate shared default }--[  ]--
# info with-context platform chassis id
    platform {
        chassis {
            id 2403019611
        }
    }
```

5.5 Disconnecting P4Runtime clients

Procedure

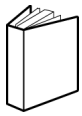
You can use a **tools** command to manually disconnect P4Runtime clients from the server.

To do this, get the identifier for the P4Runtime client from the **info from state system grpc-server** *<name>* command, then enter the following command to disconnect the client:

Example

```
-{ running }--[  ]--
# tools system grpc-server mgmt-test client 4053 disconnect
```

Customer document and product support



Customer documentation

Customer documentation welcome page

Technical support

Product support portal

Documentation feedback

Customer documentation feedback