Abstract

This document describes a YANG data model for Multi-Protocol Label Switching (MPLS) Label Distribution Protocol (LDP) and Multipoint LDP (mLDP).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 10, 2015.
1. Introduction

The Network Configuration Protocol (NETCONF) [RFC6241] is a network management protocol that defines mechanisms to manage network devices. YANG [RFC6020] is a modular language that represents data structures in an XML tree format, and is used as a data modeling language for the NETCONF.

This document introduces a YANG data model for MPLS Label Distribution Protocol (LDP) [RFC5036] and Multipoint LDP (mLDP) [RFC6388]. For LDP, it also covers LDP IPv6 [I-D.ietf-mpls-ldp-ipv6] and LDP capabilities [RFC5561].
The data model is defined for following constructs that are used for managing the protocol:

- Configuration
- Operational State
- Executables (Actions)
- Notifications

Given mLDP tight coupling with LDP, mLDP model is defined under LDP tree. The document is organized to first define the data model for the configuration, operational state, actions and notifications of LDP, followed by mLDP.

2. Specification of Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

In this document, the word "IP" is used to refer to both IPv4 and IPv6, unless otherwise explicitly stated. For example, "IP address family" means and be read as "IPv4 and/or IPv6 address family"

3. LDP YANG Model

3.1. Overview

The LDP/mLDP Yang model is defined under "ietf-mpls-ldp" module and augments "routing-protocol" list in ietf-routing module [I-D.ietf-netmod-routing-cfg] with LDP and mLDP specific parameters. [Ed note: This model will be aligned with MPLS as and when a base tree for MPLS is defined and available].

There are four containers in LDP module as follows:

- Read-Write parameters (for configuration)
- Read-only parameters (for operational state)
- Notifications (for events)
- RPCs (for executing commands to perform some action)

Before going into data model details, it is important to take note of the following points:
This module aims to address only the core LDP/mLDP parameters as per RFC specification, as well as some widely used and deployed non-RFC features. Any vendor specific feature should be defined in a vendor-specific augmentation of this model.

Multi-topology LDP [RFC7307] and Multi-topology mLDP [I-D.iwijnand-mpls-mldp-multi-topology] are beyond the scope of this document.

This module does not cover any applications running on top of LDP and mLDP, nor does it cover any OAM procedures for LDP and mLDP.

Current revision defines protocol-centric model as compared to vrf-centric model. The vrf-centric model will be specified in a later revision. [Ed note: This specification will be aligned as and when a decision to use protocol vs vrf centric model is made at RTG group level]

This model assumes platform-wide label space (i.e. label space Id of zero).

In this model, an "instance" under LDP tree refers to a VRF instance.

This model currently supports two address-families, namely "ipv4" and "ipv6".

The label and neighbor policies and filters are defined using a prefix-list. The prefix-list is referenced from routing-policy model as defined in [I-D.shaikh-rtgw-policy-model].

The use of grouping (templates) for bundling and grouping the configuration items is not employed in current model, and is a subject for consideration in future.

A graphical representation of LDP YANG data model is presented in Figure 2, Figure 4, Figure 5, and Figure 6. Whereas, the actual model definition in YANG is captured in Section 5.

3.2. Configuration

This specification defines the configuration parameters for base LDP as defined in [RFC5036] and LDP IPv6 [I-D.ietf-mpls-ldp-ipv6]. Moreover, it incorporates provisions to enable LDP capabilities [RFC5561], and defines some of the most significant and commonly used capabilities such as Typed Wildcard FEC [RFC5918], End-of-LIB [RFC5919], and LDP Upstream Label Assignment [RFC6389].
This specification currently supports only protocol-centric configuration. We plan to specify VRF-centric configuration in a later next revision. In this protocol-centric model, the LDP configuration is applied within the standard routing-instance, with instance list helping to reference the routing instance (VRF) where LDP is enabled and activated.

module: ietf-mpls-ldp
  +--rw routing
    +--rw routing-instance [name]
    +--rw routing-protocols
      +--rw routing-protocol [name]
        +--rw mpls-ldp
          .
          .
          +--rw instance* [name]
            .

Figure 1

Given the configuration hierarchy, the model allows inheritance such that an item in a child tree is able to derive value from a similar or related item in one of the parent. For instance, hello holdtime can be configured either globally or per-VRF or per-VRF-interface, thus allowing inheritance as well flexibility to override with a different value on any child level.

Following is a simplified graphical representation of the data model for LDP configuration.

module: ietf-mpls-ldp
augment /rt:routing/rt:routing-instance/rt:routing-protocols/
  rt:routing-protocol:
    +--rw mpls-ldp
      +--rw graceful-restart
        |  +--rw enable?  boolean
        |  +--rw helper-enable?  boolean
        |     {graceful-restart-helper-mode}?
        |  +--rw reconnect-time?  uint16
        |  +--rw recovery-time?  uint16
        |  +--rw forwarding-holdtime?  uint16
      +--rw igp-synchronization-delay?  uint16
      +--rw nonstop-routing?  boolean
      +--rw discovery
++--rw interfaces
| ++--rw hello-holdtime?  uint16
| ++--rw hello-interval?  uint16
++--rw targeted
| ++--rw hello-holdtime?  uint16
| ++--rw hello-interval?  uint16
| ++--rw hello-accept {policy-extended-discovery-config}?
| | ++--rw enable?  boolean
++--rw neighbors
| ++--rw md5-password?  string
| ++--rw session-ka-holdtime?  uint16
| ++--rw session-ka-interval?  uint16
++--rw instance* [name]
| ++--rw name  union
| ++--rw admin-down?  boolean {admin-down-config}?
| ++--rw lsr-id?  union
++--rw capability
| | ++--rw end-of-lib {capability-end-of-lib-config}?
| | | ++--rw enable?  boolean
| | ++--rw typed-wildcard-fec  {capability-typed-wildcard-fec-config}?
| | | ++--rw enable?  boolean
| | ++--rw upstream-label-assignment  {capability-upstream-label-assignment-config}?
| | | ++--rw enable?  boolean
++--rw graceful-restart
| | ++--rw enable?  boolean
| | ++--rw helper-enable?  boolean {graceful-restart-helper-mode}?
| | ++--rw reconnect-time?  uint16
| | ++--rw recovery-time?  uint16
| | ++--rw forwarding-holdtime?  uint16
+--rw igp-synchronization-delay?  uint16
++--rw address-family
| ++--rw ipv4
| | ++--rw enable?  boolean
| | ++--rw label-policy
| | | ++--rw independent-mode
| | | | | ++--rw assign {policy-label-assignment-config}?
| | | | | | | ++--rw (prefix-option)?
| | | | | | | | ++: (prefix-list)
| | | | | | | | | ++--rw prefix-list?  prefix-list-ref
| | | | | | | | | | ++: (host-routes-only)
| | | | | | | | | | | ++--rw host-routes-only?  boolean
| | | | | | | | | | | | ++--rw advertise
| | | | | | | | | | | | | ++--rw explicit-null!
| | | | | | | | | | | | | | ++--rw prefix-list?  prefix-list-ref
| | | | | | | | | | | | | | | ++--rw prefix-list?  prefix-list-ref
++-rw accept
   ++-rw prefix-list?  prefix-list-ref
++-rw ordered-mode {policy-ordered-label-config}?
   ++-rw egress-lsr
      |  ++-rw prefix-list?  prefix-list-ref
   ++-rw advertise
      |  ++-rw prefix-list?  prefix-list-ref
   ++-rw accept
      |  ++-rw prefix-list?  prefix-list-ref
   ++-rw transport-address?  inet:ipv4-address
++-rw ipv6
++-rw enable?  boolean
++-rw label-policy
   ++-rw independent-mode
      |  ++-rw assign {policy-label-assignment-config}?
         |  ++-:(prefix-option)?
         |     |  ++-:(prefix-list)
         |  ++-:(host-routes-only)
      |  ++-rw host-routes-only?  boolean
++-rw advertise
   |  ++-rw explicit-null!
   |  |  ++-rw prefix-list?  prefix-list-ref
   |  ++-rw prefix-list?  prefix-list-ref
   ++-rw accept
      |  ++-rw prefix-list?  prefix-list-ref
++-rw ordered-mode {policy-ordered-label-config}?
   ++-rw egress-lsr
   ++-rw advertise
   ++-rw accept
   ++-rw prefix-list?  prefix-list-ref
++-rw transport-address?  inet:ipv6-address
++-rw discovery
   ++-rw interfaces
      |  ++-rw hello-holdtime?  uint16
      |  ++-rw hello-interval?  uint16
      |  ++-rw interface* [interface]
         |  ++-rw interface  if:interface-ref
         |  ++-rw hello-holdtime?  uint16
         |  ++-rw hello-interval?  uint16
         |  ++-rw igp-synchronization-delay?  uint16
            {per-interface-timer-config}?
      |  ++-rw address-family
         |  ++-rw ipv4
            |  ++-rw transport-address?  union
            |  ++-rw enable?  boolean
++-rw targeted
   +-rw hello-holdtime?  uint16
   +-rw hello-interval?  uint16
   +-rw hello-accept {policy-extended-discovery-config}?
      +-rw enable?        boolean
      +-rw peer-list?     peer-list-ref
   +++-rw address-family
      +-rw ipv4
      |  +-rw target* [address]
      |     +-rw address     inet:ipv4-address
      |     +-rw enable?     boolean
      +-rw ipv6
      |  +-rw target* [address]
      |     +-rw address     inet:ipv6-address
      |     +-rw enable?     boolean
   ++-rw neighbors
      +-rw md5-password?   string
      +-rw session-ka-holdtime?  uint16
      +-rw session-ka-interval?  uint16
      +-rw session-downstream-on-demand {session-downstream-on-demand-config}?
         +-rw enable?     boolean
         +-rw peer-list?  peer-list-ref
      ++-rw session-protection {session-protection}?
         +-rw enable?     boolean
         +-rw duration?   union
         +-rw peer-list?  peer-list-ref
      ++-rw neighbor* [lsr-id]
         +-rw lsr-id       union
         +-rw admin-down?  boolean
         +-rw md5-password? string
         +-rw graceful-restart
            +-rw enable?     boolean
            +-rw reconnect-time?  uint16
            +-rw recovery-time?  uint16
         +-rw session-ka-holdtime?  uint16
         +-rw session-ka-interval?  uint16
         +-rw session-protection {session-protection}?
            +-rw enable?     boolean
            +-rw duration?   union
         ++-rw address-family
            +-rw ipv4
               +-rw label-policy
                  +-rw advertise
                  |  +-rw prefix-list? prefix-list-ref
3.2.1. Configuration Hierarchy

The LDP configuration container is logically divided into following high level config areas:

1. Global parameters
2. Per-VRF parameters
   o Global parameters
   o Per-address-family parameters
   o Hello Discovery parameters
     - interfaces
       - Per-interface:
         Per-interface Global
         Per-interface per-address-family
     - targeted
     - Per-target
   o Neighbor parameters
     - Global
     - Per-neighbor
       Per-neighbor per-address-family

Following subsections briefly explain these configuration areas.

3.2.1.1. Global parameters

These are the parameters whose scope apply globally or apply to all VRF instances. The example of a global configuration is LDP non-stop-routing feature. Typically, most of the parameters configurable at global level can be configured at per-VRF level or under other subtree under per-VRF.
3.2.1.2. Per-VRF parameters

These are the parameters whose scope apply within the context of a given VRF instance and are configured under "instance [name]". The majority of the LDP configuration falls under this category and is divided further into sub categories as follows.

3.2.1.2.1. Per-VRF global parameters

There are configuration items that are available directly under a VRF instance and do not fall under any other sub tree. Example of such a parameter is LDP lsr-id which is typically configured per VRF.

3.2.1.2.2. Per-VRF Per-Address-Family parameters

Any LDP configuration parameter related to IP address family (AF) whose scope is VRF wide is configured under this tree. The examples of per-AF parameters include enabling the AF, prefix-list based label policies, and LDP transport address.

3.2.1.2.3. Per-VRF Hello Discovery parameters

This container is used to hold LDP configuration related to Hello and discovery process for both basic (link) and extended (targeted) discovery.

The "interfaces" is a container to configure parameters related to VRF interfaces. There are parameters that apply to all interfaces (such as hello timers), as well as parameters that can be configured per-interface. Hence, an interface list is defined under "interfaces" container. The model defines parameters to configure per-interface non address-family related items, as well as per-interface per-AF items. The example of former is interface hello timers, and example of later is enabling hellos for a given AF under an interface.

The "targeted" container under a VRF instance allows to configure LDP targeted discovery related parameters. Within this container, the "target" list provides a mean to configure multiple target addresses to perform extended discovery to a specific destination target, as well as to fine tune parameters per-target.

3.2.1.2.4. Per-VRF Neighbor parameters

This container is used to hold LDP configuration related to LDP neighbors (i.e. peers) under a VRF instance. This container allows to configure parameters that either apply on all VRF neighbors or a subset (peer-list) of VRF neighbors. The example of such parameters
include authentication password, session KA timers etc. Moreover, the model also allows per-neighbor parameter tuning by specifying a "neighbor" list under the "neighbors" container. A neighbor is uniquely identified using its LSR Id and hence lsr-id is the key for neighbor list.

Like per-interface parameters, some per-neighbor parameters are AF-agnostic (i.e. either non AF related or apply to both IP address families), and some are that belong to an AF. The example of former is per-neighbor password configuration, whereas the example of later is prefix-list based label policies (inbound and outbound) that apply to a given neighbor.

3.3. Operational State

Operational state of LDP can be queried and obtained from this read-only container "mpls-ldp" which is augmented from "routing-state" of a routing-protocol (/rt:routing-state/rt:routing-instance/rt:routing-protocols/rt:routing-protocol).

Following is a simplified graphical representation of the data model for LDP operational state.

module: ietf-mpls-ldp
augment /rt:routing-state/rt:routing-instance/rt:routing-protocols/rt:routing-protocol:
  +--ro mpls-ldp
  .

[Ed note: TODO (later revision)]

Figure 4

3.4. Notifications

This model defines a list of notifications to inform client of important events detected during the protocol operation. These events include events related to changes in the operational state of an LDP neighbor, hello adjacency, etc.

Following is a simplified graphical representation of the data model for LDP notifications.
module: ietf-mpls-ldp

notifications:
  +++ n mpls-ldp-neighbor-event
    |  +++ ro event-type?          oper-status-event-type
    |  +++ ro routing-instance-ref? rt:routing-instance-ref
    |  +++ ro ldp-protocol-name?   leafref
    |  +++ ro ldp-vrf-instance?    leafref
    |  +++ ro neighbor-ref?        leafref
  +++ n mpls-ldp-adjacency-event
    |  +++ ro event-type?          oper-status-event-type
    |  +++ ro routing-instance-ref? rt:routing-instance-ref
    |  +++ ro ldp-protocol-name?   leafref
    |  +++ ro ldp-vrf-instance?    leafref
    |  +++ ro (adjacency-type)?    
      |  |  +++ ro targeted
      |  |    |  +++ ro target-address?     inet:ip-address
      |  |  +++ ro link
      |  |    |  +++ ro next-hop-interface? if:interface-ref
      |  |    |  +++ ro next-hop-address?    inet:ip-address

Figure 5

3.5. Actions

This model defines a list of rpcs that allow performing an action or executing a command on the protocol. For example, it allows to clear (reset) LDP neighbors, hello-adjacencies, and statistics. The model makes an effort to provide different level of control so that a user is able to either clear all, or clear all of a given type, or clear a specific entity.

Following is a simplified graphical representation of the data model for LDP actions.
module: ietf-mpls-ldp
rpcs:
  +++-x mpls-ldp-clear-neighbor
    +++-ro input
      +++-ro routing-instance-ref? rt:routing-instance-ref
      +++-ro ldp-protocol-name? leafref
      +++-ro ldp-vrf-instance? leafref
      +++-ro lsr-id? union
  +++-x mpls-ldp-clear-adjacency
    +++-ro input
      +++-ro routing-instance-ref? rt:routing-instance-ref
      +++-ro ldp-protocol-name? leafref
      +++-ro ldp-vrf-instance? leafref
      +++-ro adjacency
        +++-ro (adjacency-type)?
        +++:(targeted)
        |  +++-ro targeted!
        |    +++-ro target-address? inet:ip-address
        +++:(link)
        |  +++-ro link!
        |    +++-ro next-hop-interface? if:interface-ref
        |    +++-ro next-hop-address? inet:ip-address
  +++-x mpls-ldp-clear-neighbor-statistics
    +++-ro input
      +++-ro routing-instance-ref? rt:routing-instance-ref
      +++-ro ldp-protocol-name? leafref
      +++-ro ldp-vrf-instance? leafref
      +++-ro lsr-id? union

4. mLDP YANG Model

[Ed note: TODO (later revision)]

5. YANG Specification

Following are actual YANG definition for LDP and mLDP constructs defined earlier in the document.

module ietf-mpls-ldp {
    namespace "urn:ietf:params:xml:ns:yang:ietf-mpls-ldp";
    // replace with IANA namespace when assigned
    prefix ldp;

    import ietf-inet-types {


prefix "inet";
}

import ietf-yang-types {
  prefix "yang";
}

import ietf-interfaces {
  prefix "if";
}

import ietf-ip {
  prefix "ip";
}

import ietf-routing {
  prefix "rt";
}

import routing-policy {
  prefix "rpl";
}

organization "TBD";
contact "TBD";
description "";

revision 2015-03-08 {
  description "Initial revision.";
  reference "";
}

/ *
 * Features
 */
feature admin-down-config {
  description "This feature indicates that the system allows to configure administrative down on a VRF instance and a neighbor.";
}

feature capability-end-of-lib-config {
  description "This feature indicates that the system allows to configure LDP end-of-lib capability.";
}
feature capability-typed-wildcard-fec-config {
  description
    "This feature indicates that the system allows to configure
    LDP typed-wildcard-fec capability.";
}

feature capability-upstream-label-assignment-config {
  description
    "This feature indicates that the system allows to configure
    LDP upstream label assignment capability.";
}

feature global-session-authentication {
  description
    "This feature indicates that the system allows to configure
    authentication at global level.";
}

feature graceful-restart-helper-mode {
  description
    "This feature indicates that the system supports graceful
    restart helper mode.";
}

feature per-interface-timer-config {
  description
    "This feature indicates that the system allows to configure
    interface hello timers at the per-interface level.";
}

feature per-neighbor-graceful-restart-config {
  description
    "This feature indicates that the system allows to configure
    graceful restart at the per-neighbor level.";
}

feature per-neighbor-session-attributes-config {
  description
    "This feature indicates that the system allows to configure
    session attributes at the per-neighbor level.";
}

feature policy-extended-discovery-config {
  description
    "This feature indicates that the system allows to configure
    policies to control the acceptance of extended neighbor
discovery hello messages.;

}

feature policy-label-assignment-config {
  description
    "This feature indicates that the system allows to configure policies to assign labels according to certain prefixes.";
}

feature policy-ordered-label-config {
  description
    "This feature indicates that the system allows to configure ordered label policies.";
}

feature session-downstream-on-demand-config {
  description
    "This feature indicates that the system allows to configure session downstream-on-demand";
}

feature session-protection {
  description
    "This feature indicates that the system supports session protection";
}

/*
 * Typedefs
 */
typedef peer-list-ref {
  type leafref {
    path "/rpl:routing-policy/rpl:defined-sets/rpl:neighbor-set/
       +"rpl:neighbor-set-name";
  }
  description
    "A type for a reference to a prefix list.";
}

typedef prefix-list-ref {
  type leafref {
    path "/rpl:routing-policy/rpl:defined-sets/rpl:prefix-set/
       +"rpl:prefix-set-name";
  }
  description
    "A type for a reference to a prefix list.";
typedef oper-status-event-type {
  type enumeration {
    enum up {
      value 1;
      description
        "Operational status changed to up.";
    }
    enum down {
      value 2;
      description
        "Operational status changed to down.";
    }
  }
  description "Operational status event type for notifications.";
}

identity mpls-ldp {
  base "rt:routing-protocol";
  description "LDP";
}

grouping ldp-instance-ref {
  description
    "An absolute reference to an LDP instance.";
  leaf routing-instance-ref {
    type rt:routing-instance-ref;
    description
      "Reference to the routing instance.";
  }
  leaf ldp-protocol-name {
    type leafref {
      path "/rt:routing/rt:routing-instance"
      + "[rt:name = current()]/../routing-instance-ref/"
      + "rt:routing-protocols/rt:routing-protocol/rt:name";
    }
    description
      "Reference to an LDP protocol name.";
  }
  leaf ldp-vrf-instance {
    type leafref {

path "/rt:routing/rt:routing-instance"
 + "[rt:name = current()//..routing-instance-ref]"/
 + "rt:routing-protocols/rt:routing-protocol"
 + "[rt:name = current()//ldp-protocol-name]/mpls-ldp/"
 + "instance/name";
}
description
"Reference to an LDP instance."
}
} // ldp-instance-ref

grouping ldp-neighbor-ref {

description
"An absolute reference to an LDP neighbor."
uses ldp-instance-ref;
leaf neighbor-ref {

type leafref {

path "/rt:routing/rt:routing-instance"
 + "[rt:name = current()//..routing-instance-ref]"/
 + "rt:routing-protocols/rt:routing-protocol"
 + "[rt:name = current()//ldp-protocol-name]/mpls-ldp/"
 + "instance"
 + "[name = current()//ldp-vrf-instance]/neighbors/
 + "neighbor/lsr-id";
}
description
"Reference to an LDP neighbor."
}
} // ldp-neighbor-ref

grouping ldp-adjacency-ref {

description
"An absolute reference to an LDP adjacency."
uses ldp-instance-ref;
choice adjacency-type {

description
"Interface or targeted adjacency."

case targeted {

container targeted {

description "Targeted adjacency."

leaf target-address {

type inet:ip-address;

description
"The target address."
}
} // targeted
}
case link {


container link {
  description "Link adjacency.";
  leaf next-hop-interface {
    type if:interface-ref;
    description "Interface connecting to next-hop.";
  }
  leaf next-hop-address {
    type inet:ip-address;
    must ".../interface" {
      description "Applicable when interface is specified.";
    }
    description "IP address of next-hop.";
  }
} // link
}
} // ldp-adjacency-ref

grouping basic-discovery-timers {
  description "Basic discovery timer attributes.";
  leaf hello-holdtime {
    type uint16 {
      range 15..3600;
    }
    units seconds;
    default 15;
    description "The time interval for which a LDP link Hello adjacency is maintained in the absence of link Hello messages from the LDP neighbor";
  }
  leaf hello-interval {
    type uint16 {
      range 5..1200;
    }
    units seconds;
    default 5;
    description "The interval between consecutive LDP link Hello messages used in basic LDP discovery";
  }
} // basic-discovery-timers

grouping extended-discovery-timers {
description "Extended discovery timer attributes."
leaf hello-holdtime {
  type uint16 {
    range 15..3600;
  }
  units seconds;
  default 45;
  description "The time interval for which LDP targeted Hello adjacency is maintained in the absence of targeted Hello messages from an LDP neighbor.";
}
leaf hello-interval {
  type uint16 {
    range 5..3600;
  }
  units seconds;
  default 15;
  description "The interval between consecutive LDP targeted Hello messages used in extended LDP discovery.";
}
} // extended-discovery-timers

grouping discovery-attributes-container {
  description "Discovery configuration attributes.";
  container discovery {
    description "Neighbor discovery attributes.";
    container interfaces {
      description "Basic discovery attributes.";
      uses basic-discovery-timers;
    }
  }
  container targeted {
    description "Extended discovery attributes.";
    uses extended-discovery-timers;
    container hello-accept {
      if-feature policy-extended-discovery-config;
      description "Extended discovery acceptance policies.";
      leaf enable {
type boolean;
  description
    "'true' to accept; 'false' to deny."
} // hello-accept
} // targeted
} // discovery
} // discovery-attributes

grouping graceful-restart-attributes {
  description
    "Graceful restart configuration attributes.";
  container graceful-restart {
    description
      "Attributes for graceful restart.";
    leaf enable {
      type boolean;
      description
        "Enable or disable graceful restart.";
    }
    leaf helper-enable {
      if-feature graceful-restart-helper-mode;
      type boolean;
      description
        "Enable or disable graceful restart helper mode.";
    }
    leaf reconnect-time {
      type uint16 {
        range 10..1800;
      }
      units seconds;
      description
        "Specifies the time interval that the remote LDP peer
must wait for the local LDP peer to reconnect after the
remote peer detects the LDP communication failure.";
    }
    leaf recovery-time {
      type uint16 {
        range 30..3600;
      }
      units seconds;
      description
        "";
    }
    leaf forwarding-holdtime {
      type uint16 {
        range 30..3600;
      }
    }
} // graceful-restart-attributes


grouping graceful-restart-attributes-per-neighbor {
  description "Per neighbor graceful restart configuration attributes.";
  container graceful-restart {
    description "Attributes for graceful restart.";
    leaf enable {
      type boolean;
      description "Enable or disable graceful restart.";
    }
    leaf reconnect-time {
      type uint16 {
        range 10..1800;
      }
      units seconds;
      description "Specifies the time interval that the remote LDP peer must wait for the local LDP peer to reconnect after the remote peer detects the LDP communication failure.";
    }
    leaf recovery-time {
      type uint16 {
        range 30..3600;
      }
      units seconds;
      description "";
    }
  }
}
// graceful-restart-attributes-per-neighbor

grouping neighbor-attributes {
  description "Neighbor configuration attributes.";

  leaf session-ka-holdtime {
    type uint16 {
      range 45..3600;
    }
    units seconds;
    description "";
  }
} // graceful-restart-attributes-per-neighbor

"The time interval after which an inactive LDP session terminates and the corresponding TCP session closes. Inactivity is defined as not receiving LDP packets from the neighbor."

leaf session-ka-interval {
  type uint16 {
    range 15..1200;
  }
  units seconds;
  description
    "The interval between successive transmissions of keepalive packets. Keepalive packets are only sent in the absence of other LDP packets transmitted over the LDP session.";
}

// neighbor-attributes

grouping session-protection-per-vrf {
  description "Session protection attributes.";
  container session-protection {
    if-feature session-protection;
    description
      "Session protection attributes.";
    leaf enable {
      type boolean;
      description
        "'true' if session protection is enabled.";
    }
    leaf duration {
      type union {
        type uint32;
        type enumeration {
          enum "infinite" {
            description "The duration is infinite.";
          }
        }
      }
    }
    units seconds;
    description
      "Session protection duration.";
  }
  leaf peer-list {
    type peer-list-ref;
    description
      "The name of a peer ACL.";
  }
} // session-protection
} // session-protection-per-vrf
grouping session-protection-per-neighbor {
    description "Session protection attributes.";
    container session-protection {
        if-feature session-protection;
        description "Session protection attributes.";
        leaf enable {
            type boolean;
            description "'true' if session protection is enabled.";
        }
        leaf duration {
            type union {
                type uint32;
                type enumeration {
                    enum "infinite" {
                        description "The duration is infinite.";
                    }
                }
            }
            units seconds;
            description "Session protection duration.";
        }
    } // session-protection
} // session-protection-per-neighbor

grouping neighbor-authentication {
    description "Neighbor authentication attributes.";
    leaf md5-password {
        type string {
            length "1..80";
        }
        description "Assigns an encrypted MD5 password to an LDP neighbor";
    } // md5-password
} // neighbor-authentication

grouping neighbor-attributes-container {
    description "Container of neighbor configuration attributes.";
    container neighbors {
        description "Container of neighbor configuration attributes.";
        uses neighbor-authentication {
            if-feature global-session-authentication;
        }
    }
}
uses neighbor-attributes;
} // neighbor-attributes-container

grouping instance-attributes {
  description "Configuration attributes at instance level.";
  uses graceful-restart-attributes;

  leaf igp-synchronization-delay {
    type uint16 {
      range 3..60;
    }
    units seconds;
    description "Sets the interval that the LDP waits before notifying the Interior Gateway Protocol (IGP) that label exchange is completed so that IGP can start advertising the normal metric for the link.";
  }
} // instance-attributes

grouping global-attributes {
  description "Configuration attributes at global level.";
  uses instance-attributes;

  leaf nonstop-routing {
    type boolean;
    default false;
    description "Enables Nonstop Routing (NSR)";
  }
} // global-attributes

grouping policy-attributes {
  description "LDP policy attributes.";
  container label-policy {
    description "Label policy attributes.";
    container independent-mode {
      description "Independent label policy attributes.";
      container assign {
        if-feature policy-label-assignment-config;
        description "Label assignment policies";
      }
    }
  }
}
choice prefix-option {
  description
  "Use either prefix-list or host-routes-only."
  case prefix-list {
    leaf prefix-list {
      type prefix-list-ref;
      description
      "Assign labels according to certain prefixes.";
    }
  }
  case host-routes-only {
    leaf host-routes-only {
      type boolean;
      description
      "'true' to apply host routes only.";
    }
  }
} // prefix-option

container advertise {
  description
  "Label advertising policies."
  container explicit-null {
    presence "Present to enable explicit null."
    description
    "Enables an egress router to advertise an
     explicit null label (value 0) in place of an
     implicit null label (value 3) to the
     penultimate hop router."
    leaf prefix-list {
      type prefix-list-ref;
      description
      "Prefix list name. Applies the filters in the
       specified prefix list to label
       advertisements. If the prefix list is not specified, explicit
       null label advertisement is enabled for all
       directly connected prefixes.";
    }
  }
  leaf prefix-list {
    type prefix-list-ref;
    description
    "Applies the prefix list to outgoing label
     advertisements.";
  }
}
description
"Label advertisement acceptance policies."
leaf prefix-list {
type prefix-list-ref;
description
"Applies the prefix list to incoming label advertisements.";
}
} // independent-mode
container ordered-mode {
if-feature policy-ordered-label-config;
description
"Ordered label policy attributes."
container egress-lsr {
description
"Egress LSR label assignment policies";
leaf prefix-list {
type prefix-list-ref;
description
"Assign labels according to certain prefixes.";
}
}
container advertise {
description
"Label advertising policies."
leaf prefix-list {
type prefix-list-ref;
description
"Applies the prefix list to outgoing label advertisements.";
}
}
container accept {
description
"Label advertisement acceptance policies."
leaf prefix-list {
type prefix-list-ref;
description
"Applies the prefix list to incoming label advertisements.";
}
} // ordered-mode
} // label-policy
} // policy-attributes

grouping neighbor-af-policy-attributes {

grouping extended-discovery-policy-attributes {
    description
        "LDP policy to control the acceptance of extended neighbor
discovery hello messages.";
    container hello-accept {
        if-feature policy-extended-discovery-config;
        description
            "Extended discovery acceptance policies.";

        leaf enable {
            type boolean;
            description
                "'true' to accept; 'false' to deny.";
        }
        leaf peer-list {
            type peer-list-ref;
            description
                "The name of a peer ACL.";
        }
    } // hello-accept
}

// neighbor-af-policy-attributes
augment "/rt:routing/rt:routing-instance/rt:routing-protocols/
+ "rt:routing-protocol" {
  when "rt:type = 'ldp:mpls-ldp'" {
    description
    "This augment is only valid for a protocol instance
    of LDP."
  }
  description "LDP augmentation."
}

container mpls-ldp {
  description
  "LDP."
  uses global-attributes;
  uses discovery-attributes-container;
  uses neighbor-attributes-container;

  list instance {
    key "name";
    description
    "Per-vrf global params."
    leaf name {
      type union {
        type enumeration {
          enum default {
            description "Special 'default' VRF instance."
          }
        }
        type string;
      }
      description
      "VRF instance name."
    }
    leaf admin-down {
      if-feature admin-down-config;
      type boolean;
      default false;
      description
      "'true' to disable the instance."
    }
  }
} // extended-discovery-policy-attributes

/*
  * Configuration data nodes
  */
leaf lsr-id {
  type union {
    type yang:dotted-quad;
    type uint32;
  }
  description "Router ID."
}

container capability {
  description "Configure capability.";
  container end-of-lib {
    if-feature capability-end-of-lib-config;
    description
    "Configure upstream label assignment capability.";
    leaf enable {
      type boolean;
      description
      "Enable end-of-lib capability.";
    }
  }
  container typed-wildcard-fec {
    if-feature capability-typed-wildcard-fec-config;
    description
    "Configure typed-wildcard-fec capability.";
    leaf enable {
      type boolean;
      description
      "Enable typed-wildcard-fec capability.";
    }
  }
  container upstream-label-assignment {
    if-feature capability-upstream-label-assignment-config;
    description
    "Configure upstream label assignment capability.";
    leaf enable {
      type boolean;
      description
      "Enable upstream label assignment.";
    }
  }
}

container instance-attributes {

  container address-family {
    description
    "Per-vrf per-af params.";
    container ipv4 {

      // capability

    }
  }

description
"IPv4 address family.";
leaf enable {
  type boolean;
  description
  "'true' to enable IPv4 address family.";
} uses policy-attributes;
leaf transport-address {
  type inet:ipv4-address;
  description
  "The transport address advertised in LDP Hello messages.";
}
} // ipv4
container ipv6 {
  description
  "IPv6 address family.";
  leaf enable {
    type boolean;
    description
    "'true' to enable IPv6 address family.";
  } uses policy-attributes;
  leaf transport-address {
    type inet:ipv6-address;
    description
    "The transport address advertised in LDP Hello messages.";
  }
} // ipv6

container discovery {
  description
  "Neighbor discovery configuration.";

  container interfaces {
    description
    "A list of interfaces for basic discovery.";
    uses basic-discovery-timers;

    list interface {
      key "interface";
      description
      "List of LDP interfaces.";
      leaf interface {
        type if:interface-ref;
      }
    }
  }
}
description
"Interface.";
}

uses basic-discovery-timers {
  if-feature per-interface-timer-config;
}

leaf igp-synchronization-delay {
  if-feature per-interface-timer-config;
  type uint16 {
    range 3..60;
  }
  units seconds;
  description
  "Sets the interval that the LDP waits before notifying the Interior Gateway Protocol (IGP) that label exchange is completed so that IGP can start advertising the normal metric for the link.";
}

container address-family {
  description
  "Per-vrf per-af params.";
  container ipv4 {
    must "/if:interfaces/if:interface" + "[name = current()//interface]/ip:ipv4" {
      description
      "Only if IPv4 is enabled on the interface.";
    }
  }
  description
  "IPv4 address family.";
  leaf transport-address {
    type union {
      type enumeration {
        enum "use-interface-address" {
          description
          "Use interface address as the transport address.";
        }
      }
      type inet:ipv4-address;
    }
  }
  description
  "IP address to be advertised as the LDP transport address.";
}

leaf enable {
  type boolean;
  description
  "Feature";
}
"Enable IPv4 address family on the interface."
}
}
container ipv6 {
  must "/if:interfaces/if:interface"
  + "[name = current()//..//..//interface]/ip:ipv6" {
    description
    "Only if IPv6 is enabled on the interface."
  }
  description
  "IPv6 address family.
leaf transport-address {
  type union {
    type enumeration {
      enum "use-interface-address" {
        description
        "Use interface address as the transport address."
      }
    }
    type inet:ipv4-address;
  }
  description
  "IP address to be advertised as the LDP transport address."
}
leaf enable {
  type boolean;
  description
  "Enable IPv6 address family on the interface."
}
} // ipv6
} // address-family
} // list interface
} // interfaces

container targeted {
  description
  "A list of targeted neighbors for extended discovery."
  uses extended-discovery-timers;
  uses extended-discovery-policy-attributes;
  container address-family {
    description
    "Per-vrf per-af params."
    container ipv4 {
      description

"IPv4 address family.";
list target {
  key "address";
  description
  "Targeted discovery params.";
  leaf address {
    type inet:ipv4-address;
    description
    "Configures a remote LDP neighbor and enables extended LDP discovery of the specified neighbor.";
  }
  leaf enable {
    type boolean;
    description
    "Enable the target.";
  }
}
} // ipv4
container ipv6 {
  description
  "IPv6 address family.";
  list target {
    key "address";
    description
    "Targeted discovery params.";
    leaf address {
      type inet:ipv6-address;
      description
      "Configures a remote LDP neighbor and enables extended LDP discovery of the specified neighbor.";
    }
    leaf enable {
      type boolean;
      description
      "Enable the target.";
    }
  }
} // ipv6
} // address-family
} // targeted
} // discovery

container neighbors {
  description
"Neighbors configuration attributes."

uses neighbor-authentication {
  if-feature global-session-authentication;
}
uses neighbor-attributes;

container session-downstream-on-demand {
  if-feature session-downstream-on-demand-config;
  description
    "Session downstream-on-demand attributes.";
  leaf enable {
    type boolean;
    description
    "'true' if session downstream-on-demand is enabled.";
  }
  leaf peer-list {
    type peer-list-ref;
    description
    "The name of a peer ACL.";
  }
}
uses session-protection-per-vrf;

list neighbor {
  key "lsr-id";
  description
  "List of neighbors.";
  leaf lsr-id {
    type union {
      type yang:dotted-quad;
      type uint32;
    }
    description "LSR ID.";
  }
  leaf admin-down {
    type boolean;
    default false;
    description
    "'true' to disable the neighbor.";
  }
}
uses neighbor-authentication;
uses graceful-restart-attributes-per-neighbor {
if-feature per-neighbor-graceful-restart-config;
}

uses neighbor-attributes {
    if-feature per-neighbor-session-attributes-config;
}
uses session-protection-per-neighbor;

container address-family {
    description "Per-vrf per-af params.";
    container ipv4 {
        description "IPv4 address family.";
        uses neighbor-af-policy-attributes;
    }
    container ipv6 {
        description "IPv6 address family.";
        uses neighbor-af-policy-attributes;
    } // ipv6
} // address-family
} // list neighbor
} // instance
} // container mpls-ldp
}

/*
 * Operational state data nodes
 */
augment "/rt:routing-state/rt:routing-instance/"
+ "rt:routing-protocols/rt:routing-protocol" {
    when "rt:type = 'ldp:mpls-ldp'"
    description "This augment is only valid for a protocol instance
    of type 'ldp'.";
}

description "LDP state."
container mpls-ldp {
    description "LDP"
}
}
/*
* RPCs
*/

crpc mpls-ldp-clear-neighbor {
  description
      "Clears the session to the neighbor.";
  input {
    uses ldp-instance-ref {
      description
        "VRF instance name. If this is not provided
        then all instances are cleared.";
    }
    leaf lsr-id {
      type union {
        type yang:dotted-quad;
        type uint32;
      } 
      description
        "LSR ID of neighbor to be cleared. If this is not provided
        then all neighbors are cleared";
    }
  }
}

crpc mpls-ldp-clear-adjacency {
  description
      "Clears the hello adjacency";
  input {
    uses ldp-instance-ref {
      description
        "VRF instance name. If this is not provided
        then all instances are cleared.";
    }
    container adjacency {
      description
        "Link adjacency or targettted adjacency. If this is not
        provided then all hello adjacencies are cleared";
      choice adjacency-type {
        description "Adjacency type.";
        case targeted {
          container targeted {
            presence "Present to clear targeted adjacencies.";
            description
              "Clear targeted adjacencies.";
            leaf target-address {
              type inet:ip-address;
              description
                "The target address. If this is not provided then
                all targeted adjacencies are cleared";
            } 
          } 
        } 
      } 
    } 
  }
}
{  
  } // targeted
  
  case link {  
    container link {  
      presence "Present to clear link adjacencies.";
      description "Clear link adjacencies.";
      leaf next-hop-interface {  
        type if:interface-ref;
        description "Interface connecting to next-hop. If this is not provided then all link adjacencies are cleared.";
      }
      leaf next-hop-address {  
        type inet:ip-address;
        must ".../interface" {  
          description "Applicable when interface is specified.";
        }
        description "IP address of next-hop. If this is not provided then adjacencies to all next-hops on the given interface are cleared.";
      } // next-hop-address
    } // link
  } // link

  rpc mpls-ldp-clear-neighbor-statistics {  
    description "Clears protocol statistics (e.g. sent and received counters).";
    input {  
      uses ldp-instance-ref {  
        description "VRF instance name. If this is not provided then all instances are cleared.";
      }
      leaf lsr-id {  
        type union {  
          type yang:dotted-quad;
          type uint32;
        }
        description "";
      }
    }
  }

"LSR ID of neighbor whose statistic are to be cleared. If this is not provided then all neighbors statistics are cleared";

/* Notifications */

notification mpls-ldp-neighbor-event {
    description "Notification event for a change of LDP neighbor operational status.";
    leaf event-type {
        type oper-status-event-type;
        description "Event type.";
    }
    uses ldp-neighbor-ref;
}

notification mpls-ldp-adjacency-event {
    description "Notification event for a change of LDP adjacency operational status.";
    leaf event-type {
        type oper-status-event-type;
        description "Event type.";
    }
    uses ldp-adjacency-ref;
}

6. Security Considerations

The configuration, state, action and notification data defined in this document are designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The NETCONF access control model [RFC6536] provides means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.
LDP is a MPLS protocol that is used to establish MPLS transport LSPs. So it is critical to ensure security of the protocol to avoid disruption of the services that depend on these transport LSPs.

There are a number of data nodes defined in the LDP and mLDP YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations.

The security concerns listed above are, however, no different than faced by other routing protocols. Hence, this draft does not change any underlying security issues inherent in [I-D.ietf-netmod-routing-cfg]

7. IANA Considerations

None.

8. Acknowledgments

The authors would like to acknowledge Eddie Chami, Mannan Venkatesan, and Jeff Tantsura for their useful comments.

9. References

9.1. Normative References

[I-D.ietf-mpls-ldp-ipv6]
Asati, R., Pignataro, C., Raza, K., Manral, V., and R. Papneja, "Updates to LDP for IPv6", draft-ietf-mpls-ldp-ipv6-17 (work in progress), February 2015.

[I-D.ietf-netmod-routing-cfg]

[I-D.shaikh-rtgwg-policy-model]

9.2. Informative References

[I-D.iwijnand-mpls-mldp-multi-topology]


Authors’ Addresses

Kamran Raza
Cisco Systems, Inc.
Email: skraza@cisco.com

Reshad Rahman
Cisco Systems, Inc.
Email: rrahman@cisco.com

Xufeng Liu
Ericsson
Email: xufeng.liu@ericsson.com

Santosh Esale
Juniper Networks
Email: sesale@juniper.net

Xia Chen
Huawei Technologies
Email: jescia.chenxia@huawei.com

Himanshu Shah
Ciena
Email: hshah@ciena.com

Stephane Litkowski
Orange
Email: stephane.litkowski@orange.com

Rajiv Asati
Cisco Systems, Inc.
Email: rajiva@cisco.com