Dynamic Resource Reservation Proposal

Name

Advanced Reservation Service

Repo Name

reservation

Description

This page is about the reservation module in OpendayLight using MD-SAL/Yang/RestConf The idea behind the reservation application is to be able to have end to end multi-layer provisioning. Assume the two end to be two ports in a device. The link provisioning could be done based on information such as latency, bandwidth, etc. Link reservation also may consider the period of time (duration), and taking into account the cost of reservation. In a cloud access scenario, one can assume that a user would benefit from reservation service to get access to cloud resources. In another scenario, a link can be established between two cloud datacenter for virtual machine mobility.

Dynamic vs On-Demand Resource Reservation

	Pro	Cons
on-demand	flexible, fits occasional and infrequent work load	expensive for long-term usage
reservation	cost-effective for long-term usage	long-term usage commitment, expensive for occasional workload

Brokerage Service

what is coming handy is a broker that reserves a large pool of resources/instances, which then enables the users to purchase them from the broker in an on-demand fashion. This model will reduce the actual cost of reserved resources due to the increased resource utilization. Moreover, the broker can perform time-multiplexing partial usage.

How To

The MD-SAL toaster architecture is the goto example, however the Ping example appears to be more practical and easier to understand. Overall steps to build the Reservation module:

1. Create a sub project into MD-SAL called model-reservation
2. Create the project pom file with MD-SAL as a parent
3. Define the Yang data model to define the API
4. Create a folder src/main/yang/reservation.yang and build the maven project
5. Create a project under the controller which will serve as the implementation or the "plugin"
6. ?? Use the Add/Remove Flow API via the plugin to add or remove flow
7. Call the restconf API to add and remove flow

Description

In recent years considerable interest has been shown in dynamic circuit services for research networks, indeed, services in Europe (BoD), US (OSCARS) and Japan (G-Lambda) are achieving an uptake of users. As such this project proposal is intended to provided and advanced reservation service for the OpenDaylight controller for end to end multipoint connectivity.

This OpenDaylight project intent is to provide an optional advanced reservation service to OpenDaylight's connectivity services which will allow end users to request resource allocation for a specific period of time based on desired connectivity attributes.

Yang Model

```
module reservation {
  namespace "urn:opendaylight:reservation";
  prefix reservation;
  import ietf-inet-types {prefix inet;}
  revision "2013-09-11" {
    description "Reservation module between 2 ports of a a given switch";
  }
  rpc reserve {
    description "This reserve message allows
                 an RA to check the feasibility of a connection reservation, or modification an existing
connection
                 reservation. Any resources associated with the reservation or modification operation will be
                 allocated and held until a reserveCommit message is received for the specific connectionId or a
                 reservation timeout occurs (whichever arrives first).";
    input {
      leaf connectionId {
        type ConnectionIdType;
        mandatory true;
      }
      leaf globalReservationId {
        type GlobalReservationIdType;
        mandatory false;
      }
      leaf description {
        type string;
        mandatory false;
      }
      leaf criteria {
        type ReservationRequestCriteriaType;
        mandatory true;
      }
    }
    output {
      leaf reserveResponse {
        type ReserveResponseType;
      }
    }
  }
  rpc reserveCommit {
    description "The reserveCommit message is sent from an RA to a PA when a reservation or modification to an
                 existing reservation is being committed. This reservation MUST currently reside in the Reserve
Held
                 state for this operation to be accepted. The reserveCommitACK indicates that the PA has accepted
                 the modify request for processing. A reserveCommitConfirmed or reserveCommitFailed message
                 will be sent asynchronously to the RA when reserve or modify processing has completed.";
     input {
       leaf connectionId {
         type ConnectionIdType;
         mandatory true;
       }
     }
     output {
       leaf acknowledgement {
         type GenericAcknowledgementType;
       }
     }
  }
  rpc reserveAbort {
    description "The reserveAbort message is sent from an RA to a PA when an initial reservation request, or
                 modification to an existing reservation is to be aborted, and the reservation state machine
returned
                 to the previous version of the reservation. The reserveAbortACK indicates that the PA has
accepted
                 the abort request for processing. A reserveAbortConfirmed message will be sent asynchronously to
                 the RA when the abort processing has completed. There is no associated Failed message for this
                 operation.";
    input {
      leaf connectionId {
        type ConnectionIdType;
        mandatory true;
      }
    }
    output {
      leaf acknowledgement {
        type GenericAcknowledgementType;
      }
```

```
    }
  }
  rpc provision {
    description "The provision message is sent from an RA to a PA when an existing reservation is to be
                 transitioned into a provisioned state. The provisionACK indicates that the PA has accepted the
                 provision request for processing. A provisionConfirmed or message will be sent asynchronously to
                 the RA when provision processing has completed. There is no associated Failed message for this
                 operation.";
    input {
      leaf connectionId {
        type ConnectionIdType;
        mandatory true;
       }
    }
    output {
      leaf acknowledgement {
        type GenericAcknowledgementType;
       }
    }
  }
  rpc release {
    description "The NSI CS release message allows an RA to transition a previously requested reservation into a
                 released state. A reservation in a released state will deactivate associated data plane
resources,
                 but the reservation is not affected.";
    input {
      leaf connectionId {
        type ConnectionIdType;
        mandatory true;
       }
    }
    output {
      leaf acknowledgement {
        type GenericAcknowledgementType;
      }
    }
  }
  rpc terminate {
           description "The NSI CS terminate message allows an RA to transition a previously requested
reservation into a
                Terminated state. A reservation in a Terminated state will release all of the associated
resources.";
    input {
      leaf connectionId {
        type ConnectionIdType;
        mandatory true;
      }
    }
    output {
      leaf acknowledgement {
      type GenericAcknowledgementType;
      }
    }
  }
  rpc querySummary {
    description "The querySummary message provides a mechanism for an RA to query the PA for a set of
                 connection service reservation instances between the RA-PA pair. This message can be used to
                 monitor the progress of a reservation."
    input {
      leaf connectionId {
        type ConnectionIdType;
        mandatory true;
      }
      leaf globalReservationId {
        type globalReservationIdType;
        mandatory true;
      }
    }
    output {
      leaf querySummaryACK{
        type GenericAcknowledgeType;
      }
    }
  }


  rpc queryRecursive {
    description "The queryRecursive message is sent from an RA to a PA to determine the status of existing
                 reservations. The queryRecursiveACK indicates that the PA has accepted the queryRecursive
                 request for processing. A queryRecursiveConfirmed or queryRecursiveFailed message will be
                 sent asynchronously to the RA when queryRecursive processing has completed.";
    input {
      leaf connectionId {
        type ConnectionIdType;
```

```
        mandatory true;
      }
      leaf globalReservationId {
        type globalReservationIdType;
        mandatory true;
      }
    }
    output {
      leaf queryRecursiveACK {
           type GenericAcknowledgeType;
      }
    }
  }

  rpc queryResultSync {
    description "The queryResultSync message provides a mechanism for an RA to query the PA for a list of
                 confirmed, failed, and error messages against a connectionId. An RA can recover lost result
                 messages using this operation, or a polling RA can use it to retrieve a list of result messages
for
                 operations issued."
    leaf connectionId {
      type ConnectionIdType;
      mandatory true;
    }
    leaf startResultId {
      type ResultIdType;
      mandatory true;
    }
    leaf endResultId {
      type ResultIdType;
      mandatory true;
    }
  }
  rpc queryResult {
    description "The queryResult message provides a mechanism for an RA to query the PA for a list of operation
                 result messages (confirmed, failed, and error) against a connectionId. An RA can recover lost
result
                 messages using this operation"
    input {
      leaf connectionId {
        type ConnectionIdType;
        mandatory true;
      }
      leaf startResultId {
        type ResultIdType;
        mandatory true;
      }
      leaf endResultId {
        type ResultIdType;
        mandatory true;
      }
    }
    output {
    }
  }
  rpc queryNotificationSync{
    description "The queryNotificationSync message is sent from an RA to a PA to
                 retrieve a list of notification messages associated with a connectionId
                 on the PA. Unlike the queryNotification operation, the
                 queryNotificationSync is synchronous and will block until the results of
                 the query operation have been collected."
    input {
      leaf connectionId {
        type ConnectionIdType;
        mandatory true;
      }
      leaf startNotificationId {
        type int;
        mandatory true;
      }
      leaf endNotificationId{
        type int;
        mandatory true;
      }
    }
    output {
    }
  }
  rpc queryNotification{
    description "The queryNotification message is sent from an RA to a PA to retrieve
                 a list of notification messages against an existing reservation residing
                 on the PA. The returned results will be a list of notifications for the
                 specified connectionId."
    input {
```

```
      leaf connectionId {
        type ConnectionIdType;
        mandatory true;
      }
      leaf startNotificationId {
        type int;
        mandatory true;
      }
      leaf endNotificationId {
        type int;
        mandatory true;
      }
    }
    output {
    }
  }
  rpc querySummarySync{
    description "The querySummarySync message is sent from an RA to a PA. Unlike
                 the querySummary operation, the querySummarySync is synchronous
                 and will block further message processing until the results of the query
                 operation have been collected."
    input {
      leaf connectionId {
        type ConnectionIdType;
        mandatory true;
      }
      leaf globalReservationId {
        type GlobalReservationIdType;
        mandatory true;
      }
    }
    output {
    }
  }

  typeDef ConnectionIdType {
    type string {
      length "0 .. 255";
    }
  }
  typeDef GlobalReservationIdType {
    type uri {
      length "0 .. 255";
    }
  }
  typeDef ReservationRequestCriteriaType {
    leaf attributes {
      leaf version {
        type int;
      }
      leaf schedule {
        type ScheduleType;
      }
      leaf serviceType {
        type string;
      }
    }
  }
}
```

Scope

The detailed scope of work is available :

- Create a Reservation and Scheduling Features for available resources in the network
- Provide Multi-layer E2E provisioning capabilities to these reservations
- Integrate scheduling both with physical and virtual environments
- Create TL1 and MTOSI soundbound plugins for ODL

Proposal Slides: Download

Proposal Meeting Minutes

Resources Committed (developers committed to working)

- Mathieu Lemay <mlemay@inocybe.com> IRC handle: mlemay
- Arash Eghtesadi <aeghtesadi@inocybe.com> IRC handle: arash
- Jake Shamash <jshamash@inocybe.com>
- Mathieu Legault <mlegault@inocybe.com>
- Paul Hudgins <phudgins@ciena.com> <--- TO BE CONFIRMED

Initial Committers

- Mathieu Lemay <mlemay@inocybe.com> IRC handle: mlemay
- Arash Eghtesadi <aeghtesadi@inocybe.com> IRC handle: arash

Vendor Neutral

This project will be rewritten from scratch but some legacy code might be reused. Such code will be properly treated and any branding and references will be removed from the code.

Meets Board Policy (including IPR)

References

https://wiki.opendaylight.org/view/Ping : Ping module using MD-SAL.

https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Toaster_Tutorial : MD-SAL Toaster example

http://tools.ietf.org/html/rfc6020 : RFC about the Yang data model

http://tools.ietf.org/html/draft-bierman-netconf-restconf-04 : RFC about the netconf / restconf

http://fredhsu.wordpress.com/2013/06/14/adding-flows-in-opendaylight-using-python-and-rest-api/ : Adding flow with Opendaylight REST API

Mathieu Lemay, Dynamic Resource Reservation Project Proposal for Opendaylight, April 10, 2014.

Wei Wang, Di Niu, Baochun Li, and Ben Liang, ‘‘Dynamic Cloud Resource Reservation via Cloud Brokerage,’’ in Proc. IEEE Int. Conf. Distributed Computing Systems (ICDCS), Philadelphia, Pennsylvania, July 2013.