Product Backlog

OpenFlow Plugin:Backlog:FlowCookieLifecycle

Flow cookie usage

Cookie value is ignored by switch and can be used by controller to identify flow. Switch identifies flow by tableId, priority and match.

- flow statistics (flow statistic response contains flow cookie)
- flow management (in dataStore there is id, name and cookie)

Flow management

Flow can be added, removed or updated by:

- changing data in config context of dataStore (= managed by **FRM**)
- invoking corresponding action programmatically (= managed by application)
- invoking corresponding action via restconf (= managed by application)

Flows managed by application should be ignored by FRM.

FRM management

FRM is invoked through dataChange notification after new flow is written into config context of dataStore. So in case the device is available, FRM does following:

- check the mapping in order to find flow cookie (sent to device previously)
- if not found we have **new flow**
    - ignore cookie value in config if any
    - generate unique cookie for the flow
    - store generated cookie in mapping <flow.id, cookie>
    - invoke flow service provider rpc (OFPlugin) using generated cookie
- if found we have **existing flow** - update (or remove)
    - invoke corresponding rpc

Mapping will be stored in operational context (under table/cookieMapping)

Flow cookie mapping

Mapping is needed while creating, updating and deleting of flow - having flowId the cookie on device can be found.

In case of statistics there is flow cookie from device available and corresponding flowId could be found.

OpenFlow Plugin:Backlog:Overload Protection and Event Dampening

Overload Protection

Rationale and Usecase Scenarios

Typically on OpenFlow interface, there is likelihood of heavy packet-ins spikes due to scenarios like links going down, switches going down (thereby, potentially invalidating the pre-computed path-based flow rules) etc. Combining this with ongoing traffic across the topology the spike of packet-in rate punted towards controller could be very high and steep.

There could be other scenarios which can trigger spikes of packet-ins apart from above example.

So, it becomes essential to protect the controller from becoming unstable due to these spikes. One way would be to spread the load. But that is a discussion related to clustering. But, it would still be prudential to have a overload-protection as do all communications server systems do.

There are multiple means of realizing overload-protection

Solution Approaches

Some options which could be considered are (there could be other options, but following ones are relatively simpler)

Option 1: when OpenFlow messages are constructed from raw wire bytes, timestamp the objects with "creation-time". At plugin layer, compare current-time with the creation time of the message. This would indicate that how long message has "waited" in individual queues. By setting a higher and a lower threshold for wait times in queue, we can decide to process / drop the packet-in

Option 2: monitor the resource loading - CPU utilization, memory etc and based on a higher and lower watermarks, sense the load. Action part could be to drop packet-ins when load is between lower and higher thresholds

Event Dampening

Rationale and Usecase Scenarios

This has more to do with the events which occur very frequently due to instability in switch/network layer. For example, flapping of switch links is one such case.

If multiple applications start reacting to the link-up down events at a very high rate, the possibility of system becoming unstable is very high. So, selective Openflow (or any events from southbound interfaces) must be dampened before escalating the same to the applications layer.

Solution Approaches

One solution approach is discussed here (this example explains the edge-device-side implementation - but, in principle, this can also be useful on controller) - http://www.cisco.com/c/en/us/td/docs/ios/12_0s/feature/guide/s_ipevdp.html

Of course, side-effects of event-dampening has to be analyzed further. But, this can be a useful feature on the controller side.

OpenFlow Plugin:Bundles Usage

- Please head over to official documentation for Bundle Usage

OpenFlow Plugin:OF13+ Single Layer Serialization

Currently, OpenFlowPlugin supports 2 types of serialization, and that is **Multi Layer Serialization** and **Single Layer Serialization**.

IMPORTANT! Single Layer Serialization is supported only on OpenFlow 1.3 and greater and only with Lithium design. In other case it will fall back to Multi Layer Serialization. Also, when Single Layer Serialization is turned on, RPC statistics compatibility services are not working (they are deprecated anyway, and should not be used when we have direct statistics). So if you want to use statistics compatibility services, make sure that Single Layer Serialization is turned off.

Multi Layer Serialization

Multi Layer Serialization first converts OpenFlowPlugin models to OpenFlowJava models and then serializing them to bytes and then sends them to device, and vice-versa when messages are coming back. Because of this, it is adding unnecessary overhead during this process, but is less error-prone than Single Layer Serialization, because it depends on OpenFlowJava models what are modelled to match OpenFlow specification as close as possible.

Single Layer Serialization

Single Layer Serialization is directly serializing OpenFlowPlugin models to bytes and sends them to device. This implementation should be faster than Multi Layer Serialization, but as it is serializing OpenFlowPlugin models to bytes, without depending on OpenFlow specification models, it is more error-prone.

Enabling/Disabling Single Layer Serialization

Single Layer Serialization can be turned on and off by changing leaf **use-single-layer-serialization** in openflow-provider-config.yang or openflowplugin.cfg to **true** or **false**.

Enabled

openflow-provider-config.yang

```
leaf use-single-layer-serialization {
    type boolean;
    default "true";
}
```

openflowplugin.cfg

```
use-single-layer-serialization=true
```

Disabled

openflow-provider-config.yang

```
leaf use-single-layer-serialization {
    type boolean;
    default "false";
}
```

openflowplugin.cfg

```
use-single-layer-serialization=false
```

Plugin initialization with/without Single Layer Serialization

blocked URL

Sending messages with/without Single Layer Serialization

blocked URL

Receiving messages with/without Single Layer Serialization

blocked URL

OpenDaylight OpenFlow Plugin:Southbound Manager

Southbound Manager

- Provides simpler API for applications to use OFPlugin RPC
- Facilities include Flow provisioning, Group provisioning, Flow <-> group dependency chaining and Group <-> group dependency chaining

Single Layer

OpenDaylight OpenFlow Plugin:Avoid Translations

The main goal is to improve performance in Openflow-plugin (OFP) in order to get better times on installing flows on devices. Main idea is to use flows in form as in Openflow-java (OFJ) and avoid using translators to spare time in heavy load.

Openflowplugin

OFP now is working as layer between generic type of flow format and standard openflow specification. This kind of design is highly modular and downstream project are able to use generic type of flows which are translated into device form. Any device can by provided with own translator and therefore able to work without any change on front-end.

Openflowjava

OFJ is able to convert flow specified by openflow specification. This kind of flow is OFJ able to send into device as byte representation message to the device.

Boron and previous version of OFP

Here you can see the basic architecture of processing all models in OFP.

blocked URL

As you can see there is a long way from creating a ODL overall known flow model to device bytes representation of flow. Therefore can possibly be a performance issue on heavy load.

WARNING: All proposal changes in this document will make change on actual project design and in two layer approach. With this changes will lowest model accessible to upper layer or OFJ basic architecture will be drastically changed.

First change proposal

First change proposal to avoid first flow translation is use direct usage of flows model as defined in openflow specification.

NOTE: This should have impact on all downstream projects in this solution they need to rewrite all models.

blocked URL

Second change proposal

Second change proposal to avoid using translator is in making changes in OFJ instead of changing actually the model using. This approach is better in aspect not to change the actual models. Using same models has no impact to downstream projects.

NOTE: Need to cooperate with OFJ committers and project leader.

blocked URL

OpenFlow Plugin:Backlog:Statistics collection performance improvements

Current Status

Brief of the current statistics collection module (Statistics-Manager) is present

Current implementation sends all statistics collection request to all the connected node at the same time. This results in flooding of response from all the connected nodes and causing CPU spikes as well as stressing MD-SAL operational data store.

Ideas for improvement

Current approach is not scalable and has performance issues. To reduce statistics collection resource consumption and make it more scalable

- Dynamic stats polling interval based on the number of connected nodes and number of entries in the respective tables (Flow tables, Group, Meter]
- For each connected node spread out stats polling throughout the stats collection interval
- Only start stats collection after the previous collection of the stat has finished (i.e. do not be timer-driven)
- Different periodic interval for each statistics type
  - Flow statistics is crucial compared to aggregate flow stats
    - Keep the initial periodic interval High for low priority statistics
    - and make it adaptive according to amount of data received in response

File:Statistics Manager Optimization Ideas.ppt

OpenFlow Plugin:Hitless resync

Synchronization using Bundles

This document should elaborate synchronization flows, ports and group using onf extension 230 Openflow Bundles.

How the synchronization is working now

Synchronization and flow modifications can come from two sources. One is configuration datastore using FRM/FRS and second is direct using of RPCs by application or user.

blocked URL
Synchronization without using bundles

All modifications from all sources come through "Sal" service. To using bundles we need to use a new service defined in extensions **SalBundleService**

Bundles framework

Bundles service need to be an extender of sal services such as flow, port and group services.

blocked URL
Bundles framework

Bundles sync problem

Problem with bundles synchronization is that we need to synchronize all input sources. If using resync with budles which first command were delete all flows just *to be sure all remaining flow on device are update* then all incomming flow mods during the resycn phase will be lost. The bundle framework need to be consider and reprocess all incomming mods and if there is synchronization in progress using another bundle to store them. If sychronization budle is finish then we can store second budle.

blocked URL
Bundle synchronization issue

Opening and closing bundles in framework

After device is connected framework need to be able know if device bundles or not. But there is only one service for all devices it can be configured only **on/off** per cluster node not per device. If switched off there would not be any framework loaded. If it is on framework still need to be checking if the device is capable of bundles and the and only then using budles service.

blocked URL

Opening bundles with framework
blocked URL
Closing bundles

OpenFlow Plugin:PoC For Single Model Layer

- Since this part is in a bigger context hence please follow up with PoC For Single Model Layer

OpenFlow Plugin:Split connection layer

Main idea

Main idea to split connection handler and core of the plugin. Now it is working as one layer and always when is connection drop (worse if connection is flapping) plugin does destroy all context which was created on a long way. If connection layer and core were divided to two separate layer plugin can create all context once and on disconnect just hold the information and stop working until reconnect. On reconnect it just use the information already created and start working. There are many cases need to be considered. But all connection problems were solved in small lightweight part of code and it should not affect the rest of the plugin which leads to quicker connection stabilization. It is possible even change connection for the device if needed.

Graphical representation of the idea

blocked URL

Class structure proposal

Class changes / adds

- Connection Handler
- Connection Holder
- Connection Status

Each separate layer need one handler to handle all event can occur on the layer. Connection handler need to register and hold all informations about connections. Each connection is stored in connection holder and holder has a status of connected,disconnected and if connected if connection is stable or unstable (flapping each amount of time)

- Context Handler
- Context (Chain) Holder
- Context Status

Context handler is starter for context chain holder and registrar of all of them. Each context chain holder need to have a status if created, working, sleeping or on disconnect/destroy. Context holder need to be a listener for connection handler events

Connection Handler events

- Created connection
- Dropped connection
- Changed status of connection

Reaction on event

If created connection the context handler need to check if context chain holder exist and if not the create it, otherwise put it into working state. If dropped connection and context handler exists and is in created state it should be changed to sleeping status. After some amount of time (but not if connection is just unstable) change context chain holder to destroy state and destroy it.

Class changes diagram

blocked URL

OpenFlow Plugin:Connection handler

Connection handler

Connection handler is implementation of previous idea od Split connection layer. In the first concept the idea was to split connection and core contexts handling in separate packages. After testing and considerations was the implementation to complex to handle it in one release. All changes were made in openflowplugin core. Where the creating, handling and managing the connections per switch is processed correct and predictable in cluster.

About connection we speak in terms of "after handshake and version negotiations".

Before connection handler

Each part of OFP is handled by manager and all information about this part and per switch connection is held in context. So there are RPC, Statistics and Device managers and their contexts. (Beryllium still have also Role manager and contexts).
After switch connect handling passes over to the device manager which was creating context then to the next statistics manager which was creating context for it self and last to the rpc manager who passed the handling to the device manager again to finish the creation. Store SLAVE state to the device could happen an issue and it causes to fail proper device connect. If that's the issue plugin needed to close the device connection and close all the contexts. OFP let the all the async tasks fail and after that and only then the plugin was able to accept a new connection from the same device.

blocked URL

blocked URL

Connection handler changes

All the complication with the core contexts handling were solved by creating ContextChainHolder and all the contexts are encapsulated with ContextChain. After connection the main process is passed to ContextChainHolder. Holder check if there is already ContextChain created for this connection and if it is ok.

- If there is a good connection for the same device, new connection is stored as auxiliary connection (But not used as auxiliary connection yet)
- If there isn't any connection create a ContextChain a start up all the asynchronous tasks
- If there is a connection for his device but the old connection is in disconnection state then refuse the new connection (in this case were testing to replace of this connection, but it leads to many issues. It is not meaning to replace a connection to the SLAVE state device and for the MASTERED device we needed to wait to close all other tasks before we could replace the connection anyway and the quicker way is create a new ContextsChain

blocked URL

blocked URL

When controller becomes owner ?

In the cluster environment the plugin doesn't know nothing about other instances or controllers so we need to rely on services given by md-sal. In previous releases we used only singleton services. With connection handler plugin mix-up singleton service and EOS. *Another possible solution were using akka, already used by md-sal on offset 0*
After singleton service becoming master for device, plugin starts lots of task to achieve a goal. Some of the tasks are asynchronous like initial statistics gathering, feature gathering, initial submit to the transaction - creating switch in DS, flow registry filling. Before Connection handler plugin was not able control the task properly each manager run its own tasks and if anything went wrong with the running threads plugin disconnected the device. Other tasks but didn't know about it and the whole process could take more time to end up. (timers etc.)
Connection handler will run the task also asynchronously but don't start working as master before all tasks are done on success. Then and only then cluster node can start to working. Plugin do disconnect if anything is on failure but the controller doesn't start work as master yet. And it doesn't lead to stop the process in the middle of work.

Benefits from Connection handler

Stability

This implementation provides stability in cluster on the start up mastership. Using singleton and EOS provides stability to proper delete device from DS.

Performance

The performance gain from this implementation isn't huge and it is hard measurable. It speeded slightly up the process of device connection if the device is flapping. And on the startup plugin doesn't care about switch status on the connect so no more threads with try to set slave state on device. Unless the controller don't setup ownership for certain time, then plugin set all remaining devices a role SLAVE which is AFTER the master role set on devices.

Enhancement (not finish yet)

Connection handler provides the plugin to create **"Mastership Service"** This service provides for plugins application (even can be used by downstream projects) instead of using singleton service or EOS or DTCL one service. This service has to events. One event is when controller becomes master for the switch and second is if something went wrong (disconnect, slave state ...).

OpenFlow Plugin Overload Protection

(With inputs from Michal Rehak)
Overload protection in the OF Plugin works in the following way:

1. The ConnectionConductor is the source from where all incoming messages are pushed to queues for async processing. It is the part of the OFPlugin closest to OFJava, and has on*Message methods (listeners to incoming messages).The ConnectionConductorImpl pushes messages to the QueueKeeper. Every ConnectionConductor has a local instance of the QueueKeeper.

The QueueKeeper has two queues:-- Unordered (for packetIn messages)-- Ordered (for other messages)Both are limited and blocking.2. If a particular queue is full, the messages pushed to it will be dropped. Upon a successful push, the harvester is pinged to rouse it from hibernation.3. A QueueZipper wraps the two queues, and provides the poll method. This poll method rotates regularly through the underlying queues. If the currently polled queue is empty, it polls the next queue. (See QueueKeeperFairImpl).4. Each QueueKeeper gets registered by the QueueKeeperHarvester. The Harvester runs upon one thread; iterates through all the registered QueueKeepers; and polls them. The polled message is then queued into the QueueProcessor.If all the registered queueKeepers are empty, the harverster hibernates.5. At the QueueProcessor are several threads translating messages from OFJava-API models to MD-SAL models (preserving order). The QueueProcessor uses two threadPools:-- One threadPool to process the queue items-- Another threadPool (containing one thread) to publish messages to the MD-SALA queue gets filled for different reasons:-- The MD-SAL is overloaded-- A node is flooding or generally something slowed down the processing pipeline.

If the queue in the QueueProcessor is full, it blocks the harvester. If the harvester is blocked, the queues in the QueueKeeper will not be emptied.
Note: The current implementation of the feature offers no checking of the memory or cpu load to actively throttle messages.

blocked URL

Effects of overload protection

- When a node floods the controller, it will not block messages from other nodes.
- The processing of messages is fair. Floody node messages are neither prioritized nor do they infest queues outside the ConnectionConductor.
- Memory is not used up on the controller side as messages gets dropped immediately upon an unsuccessful push to the local queue.
- The functionality cannot create back pressure at the netty level as pressure affects the echo message, and might cause a connection close action on the switch side.

OpenFlow Plugin:Bundles extension support

- Please head over to Bundles extension support documentation.

OpenFlow Plugin:Implementation For Single Model Layer

The main goal is to improve performance in Openflow-plugin (OFP) in order to get better times on installing flows on devices. Main idea is to use flows in form as in Openflow-java (OFJ) and avoid using translators to spare time in heavy load.

Openflowplugin

OFP now is working as layer between generic type of flow format and standard openflow specification. This kind of design is highly modular and downstream project are able to use generic type of flows which are translated into device form. Any device can by provided with own translator and therefore able to work without any change on front-end.

Openflowjava

OFJ is able to convert flow specified by openflow specification. This kind of flow is OFJ able to send into device as byte representation message to the device.

Boron and previous version of OFP

Here you can see the basic architecture of processing all models in OFP.

blocked URL

As you can see there is a long way from creating a ODL overall known flow model to device bytes representation of flow. Therefore can possibly be a performance issue on heavy load.

WARNING: All proposal changes in this document will make change on actual project design and in two layer approach. With this changes will lowest model accessible to upper layer or OFJ basic architecture will be drastically changed.

First change proposal

First change proposal to avoid first flow translation is use direct usage of flows model as defined in openflow specification.

NOTE: This should have impact on all downstream projects in this solution they need to rewrite all models.

blocked URL

Second change proposal

Second change proposal to avoid using translator is in making changes in OFJ instead of changing actually the model using. This approach is better in aspect not to change the actual models. Using same models has no impact to downstream projects.

NOTE: Need to cooperate with OFJ committers and project leader.

blocked URL