BroadView™ Instrumentation Agent Specification

BroadView-SP100-R

Corporate Headquarters: San Jose, CA

April 26, 2017

For a comprehensive list of changes to this document, see the Revision History.

Table of Contents

Section 1: Introduction 6

Overview 6

Supported Features by Silicon 7

Section 2: Architecture 8

Architecture Blocks 9

Northbound Plug-in Block 9

Reference Implementation 9

Infrastructure Block 9

Reference Implementation 10

Southbound Plug-in Block 10

Reference Implementation 10

Instrumentation Features Block 10

Execution in Linux 11

Using the Agent with a NOS 12

SB Plug-ins 12

NB Plug-ins 12

Section 3: Communication 13

REST 13

JSON-RPC Messaging 13

Request Response Messages 13

Time Stamps 14

Version 14

Multi-ASIC Platforms 15

Error Reporting 16

Section 4: Generic Agent Management 17

Overview 17

Heartbeat Messaging 17

Switch Properties 18

Configurability 19

API 20

Overview 20

Configuration API 20

configure-system-feature 20

cancel-request 21

Status/Reporting API 21

get-port-queue-map 21

get-system-feature 30
get-switch-properties 30
Section 5: Buffer Statistics and Tracking 32
Overview 32
Buffer Utilization Statistics and Thresholds 32
Configurability 33
Complete and Incremental Reports 34
Queues 34
Deciphering Reports 35
Example 1 35
Example 2 35
Example 3 36
Example 4 36
API 37
Overview 37
Configuration/Clear API 37
configure-bst-feature 37
configure-bst-tracking 39
configure-bst-thresholds 40
clear-bst-statistics 42
clear-bst-thresholds 42
clear-bst-congestion-drop-counters 42
Status/Reporting API 43
get-bst-feature 43
get-bst-tracking 43
get-bst-thresholds 44
get-bst-report 46
get-bst-congestion-drop-counters 49
Trigger Reports 53
Section 6: Packet Trace 56
Overview 56
Configurability 56
Packet Format 57
Live Traffic Triggered Trace-Profile 58
Timestamps 59
Deciphering Trace-Profile 59
API 62
Overview 62

Configuration/Clear API ... 62

configure-packet-trace-feature ... 62

cancel-packet-trace-profile ... 63

cancel-packet-trace-lag-resolution ... 63

cancel-packet-trace-ecmp-resolution ... 64

Status/Reporting API ... 64

get-packet-trace-feature ... 64

get-packet-trace-lag-resolution ... 65

get-packet-trace-ecmp-resolution ... 68

get-packet-trace-profile ... 71

Section 7: Black Hole Detection ... 77

Overview ... 77

Black Hole Ports ... 77

Reporting Black Hole Events ... 78

Configurability ... 78

API ... 79

Overview ... 79

Configuration API ... 79

black-hole-detection-enable ... 79

configure-black-hole ... 80

cancel-black-hole ... 81

Status/Reporting API ... 82

get-black-hole-detection-enable ... 82

get-black-hole ... 82

get-sflow-sampling-status ... 83

Notifications ... 84

get-black-hole-event-report ... 84

Section 8: Reference Applications ... 86

Section 9: License ... 87

Section 10: Source Code ... 88

Section 11: Documentation ... 89

Appendix A: JSON Payload Revision History ... 90

Section 12: Revision History ... 92

Section 1: Introduction

Overview

With the proliferation of Software-Defined Networking (SDN), multi-tenant networks, and server Virtualization—aided by cloud deployments for applications and storage—network complexity is growing exponentially. Troubleshooting such networks has become an increasingly daunting task. Network operators need increased visibility into the network and deeper telemetry data in order to remain in control of the network and to ensure optimal network resource utilization. BroadView™ Instrumentation software provides this critical visibility and telemetry information.

BroadView Instrumentation Agent unlocks the potential of Broadcom® switch silicon by augmenting CPU functionality to deliver advanced network analytics. The Agent source code is Apache licensed, sports a modular design, offers REST API for interaction, and is easily adaptable into customer solutions.

The BroadView Instrumentation Agent communicates with the underlying Broadcom switch silicon. It collects various telemetry and visibility information, runs algorithms on the data, packages the information appropriately, and provides it to registered clients. Similarly, the Agent configures the silicon based on the configuration requests from the client.

The Agent communicates with clients using REST-style communication, with the data exchange in JSON-RPC (2.0) format. The Agent supports both the pull model of operation (the client requests data and obtains it) as well as the push model of operation (the Agent sends periodic reports, asynchronously).

The Agent works with a wide range of Broadcom switch silicon, and the features offered by the Agent operate in accordance with the underlying silicon.

In a deployment scenario, the BroadView Agent works in conjunction with the Network Operating System (NOS). The Agent can work completely integrated into the NOS, or as an independent application on the switch. Likewise, the telemetry information can be used by the NOS itself and/or by clients interacting with the Agent via the REST API.

This document describes the Agent architecture and each of the supported BroadView features. Each BroadView feature description is followed by a listing of REST APIs supported by the feature.

Supported Features by Silicon

Table 1 lists the BroadView Instrumentation Agent features and identifies which Broadcom devices support each feature.

Table 1: Supported Features by Silicon

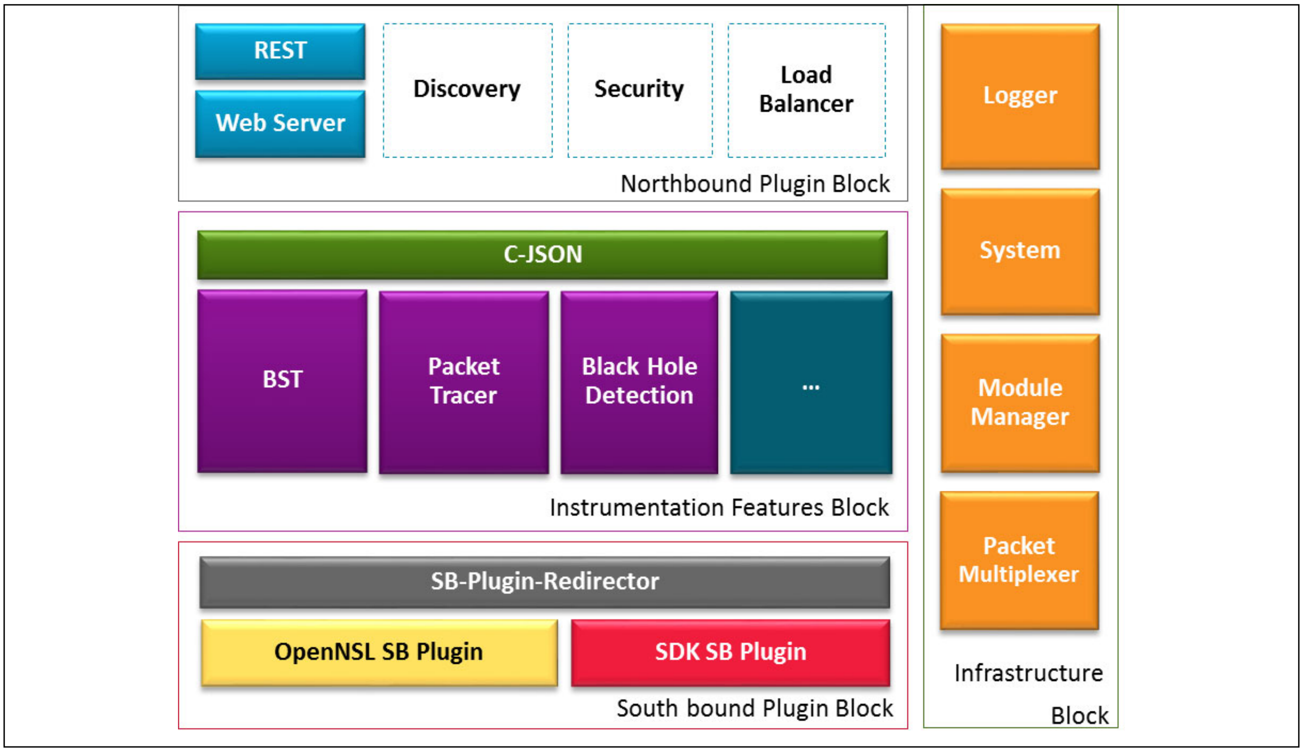
Feature	*Supported Silicon*
Buffer Statistics and Tracking (BST)	BCM56850 BCM56960
Packet Trace (PT)	BCM56960
Black Hole Detection (BHD)[a]	BCM56850 BCM56960 BCM88375

[a] BDH sFlow sampling is supported only on the BCM56960.

Section 2: Architecture

The BroadView Instrumentation Agent has a modular architecture and consists of a set of components that serve different purposes in the architecture. Some core components implement the telemetry and analytics algorithms, while others function as infrastructure components. Some of the components are meant for porting and/or customization, and the reference implementations of those components are for illustrative purposes only. For example, by default, the Agent software does not support any authentication or security for the communication with the collector. However, the security and web server modules can be easily enhanced or added to provide additional functionality. Subsequent pages list various customizations that are possible.

Figure 1: BroadView Agent Architecture



Architecture Blocks

The components that form the Agent software are divided into the following four blocks described below:

- "Northbound Plug-in Block"
- "Infrastructure Block"
- "Southbound Plug-in Block"
- "Instrumentation Features Block"

Northbound Plug-in Block

The Northbound (NB) Plug-in block is responsible for setting up and managing the communication with the applications/controller(s)/collector(s), referred to as collectors, that interface with the Agent. The distribution includes a reference application that exercises the NB API of the reference Agent.

The Agent uses REST-style communication to communicate with the collector, with the data exchange in the JSON-RPC format. The REST and web server components handle these interactions with the collectors. The REST component extracts the incoming JSON-RPC request, deciphers the API being invoked by the collector, and passes on the JSON message to the appropriate Instrumentation Features block component (using the Module Manager component services).

The Discovery component is responsible for detecting the presence of one or more collectors on the network and establishing communication between the Agent and the device. It also maintains a heartbeat channel with the collector for detecting any loss of communication.

The Security component provides encryption support for the communication between the Agent and the collector. It also provides the necessary authentication for the discovered collectors.

The Load Balancer component provides active and standby collectors. It keeps track of communication errors with the designated active collector. In the event of persistent failures, this component switches the standby collector to the active state and establishes communication with it.

Reference Implementation

The reference implementation of the Discovery module reads the collector IP addresses and communication port numbers from a predefined configuration file.

The Security module is not provided as part of the reference implementation. This means that the Agent allows unencrypted communication between the collector and the Agent. Likewise, the collectors are considered genuine, and no attempt is made to validate the authenticity of the collector.

The Load Balancer module is not provided as part of the reference implementation. Any errors during either the pull or during the push operations are logged, and communication is reattempted.

Infrastructure Block

The Infrastructure block provides the necessary infrastructure and utilities for the other modules in the Agent.

The Logging component provides logging functionality to the Agent system. It provides API functions that can be used for logging various events and data. Different levels of logging are supported. The Module Manager component provides for registration of various Instrumentation components with the Agent. Each of the Instrumentation components must register with the Module Manager to provide the details of various APIs it supports, the feature identification, supported silicon, and the callback functions that process the API. When a REST API is invoked by the collector, the Module Manager component assists the REST component in invoking the appropriate handler for the API, based on the registration information.

The Instrumentation components are typically C code linked with the Agent. The components register the supported REST API with the Module Manager at run time during the initialization process.

The System component is in charge of setting up the Agent based on the configuration and starting/stopping various other components as needed. The System component also provides the timer API, which can be used by the other components for periodic data collection by the hardware. Note that the periodic data collection may be disabled/enabled by the collector on a system-wide basis or selectively on a per-component basis.

Reference Implementation

The reference implementation of the Logging component uses syslog. It can easily be enhanced to use a simple SQLite3 database.

Southbound Plug-in Block

The Southbound block features different plug-in modules that conform to the SB-API specification. Each of the plug-ins registers with the SB-Plug-in Redirector component, with the list of features as well as the silicon that are supported by the plug-in. When any BroadView Instrumentation component attempts to communicate with the silicon, it invokes the corresponding API of the SB-Plug-in Redirector component, which in turn uses the registration information to invoke the corresponding plug-in component. The purpose of the SB-Plug-in is to allow functions to be written that are specific to the mechanism that is available to obtain the instrumentation information from the silicon on any specific system. For example, System 1 could allow the use of an SDK API while System 2 may have its own API (such as the OpenNSL API).

The SB-Plug-in modules are typically shared object libraries linked with the Agent. The plug-in registration is done by invoking the SB-Plug-in Redirector component's registration API.

Reference Implementation

The reference implementation consists of an SDK SB Plug-in and OpenNSL SB Plug-in.

Instrumentation Features Block

The Instrumentation components are registered statically with the Agent. Upon startup, the Agent initializes each of the registered components as part of the global initialization sequence. The components are required to register the supported silicon, features, and the REST API with the Module Manager component. Such registration enables the Module Manager to understand the set of REST APIs that are supported by the entire Agent. Any feature configuration REST API is also included in the registration.

The web server thread invokes the API handler function with the JSON message buffer.

The supported data formats for the API (commands) as well as the responses are documented in XML format. A tool is run on these XML files, which can generate C code that can parse the JSON buffer and convert the incoming data to an appropriate C structure. This generated code is used as the implementation for the handler functions. After the C structure is derived, the control is handed over to the component thread. The web server thread acknowledges the REST API to the collector.

The component, under its own thread context now, makes the appropriate function calls to process the request and invoke the appropriate SB-Plug-in API calls. It is also likely that the component may not be required to make any SB-Plug-in calls for fulfilling the REST API since the data may be collected by the periodic thread already. When all the required data is available, the data is converted into JSON. Using a web server API function, the data is sent back to the collector.

Execution in Linux

Various components of the Agent are compiled and linked into a single Linux-executable[1]. This user-space executable (as a single process) runs multiple threads within, based on the needs.

Typically, each of the Instrumentation feature modules start and maintain a thread. The API invoked by the collector is to be executed in the corresponding feature/module thread context. In addition, there is a system-wide timer thread meant for the periodic data collection. The other modules register their data-gathering functions with the system data-collection thread and suggest the periodicity with which the functions need to be invoked. The registered callbacks are invoked in the data-collection thread context, and care must be taken by the callbacks to ensure data integrity.

The components in the Agent use a POSIX API for any required inter-thread communication.

No kernel space operations are performed within the Agent process. Multiple Agent instances running in a single Linux environment is not supported.

[1] Linux Operating System is assumed.

Using the Agent with a NOS

This section describes high-level porting considerations when adapting the Agent to a NOS.

SB Plug-ins

A Southbound plug-in provides a means of interaction with the underlying silicon to the rest of the Agent software. The reference Agent has an SB plug-in that uses the SDK /OpenNSL API calls directly inside the implementation. For various reasons, this may not be desirable when the Agent is used in conjunction with a NOS. It may be preferable to route the silicon accesses via the NOS.

If the access to the silicon is to be routed via the NOS, an appropriate SB plug-in must be developed.

NB Plug-ins

The default web server provided by the Agent does not handle the security or the discovery/load-sharing aspects. If this is not an ideal implementation, use one of the following alternatives:

- Replace the default web server with another web server component, or integrate the Agent into the NOS-supported web server.
- Use the NOS web server to handle the incoming REST requests. After authenticating the request, the NOS web server hands the request over to the Agent web server.

Section 3: Communication

REST

The BroadView Agent supports a REST-based API and uses JSON-RPC messaging as the payload over REST. The name of the API becomes the method in the JSON-RPC message, and any associated parameters for the command form the **params** in the JSON-RPC message.

All the commands are to be sent as HTTP 1.1 POST operations, with the JSON content as the HTTP message body.

By default, the web server provided as part of the reference Agent uses the following URL scheme to provide the access to the API:

http://<ipaddr>:8080/broadview/<feature>/<api-name>

For example, the `configure-bst-feature` API is accessible via the following URL:

http://<ipaddr>:8080/broadview/bst/configure-bst-feature

For the Generic Agent Management API, the URL scheme does not have the feature name embedded in the URL. For example, the `configure-system-feature` API is accessible via the following URL:

http://<ipaddr>:8080/broadview/configure-system-feature

JSON-RPC Messaging

Request Response Messages

The API supported by BroadView Agent uses JSON-RPC messaging as payload over HTTP. There are three forms of JSON-RPC messages that are supported by the Agent:

- **Request** (from the collector and received/processed by the Agent), sometimes also called as a **Command**.
- **Response** (from the Agent and received/processed by the collector in response to a previous request).
- **Notification** (from the Agent and received/processed by the collector, asynchronously).

An example JSON-RPC **request** message sample is provided below for illustrative purposes.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-feature",
    "params": {},
    "asic-id": "1",
    "id": 1232
}
```

A sample JSON-RPC **response** message is provided below for illustrative purposes.

```
{
    "jsonrpc": "2.0",
    "asic-id": "1",
    "version": "1",
    "result": {
        "bst-enable": 1,
        "bst-trackers": {
            "device" : 0,
            "egress-unicast-queue" : 1
        }
    },
    "id": 1232
}
```

A notification message is similar to a request message, with the exception that it does not have an “id” field. More details are available at http://www.jsonrpc.org/specification.

When an Agent receives a request from the collector, the **method** field specifies the request that is sent. Any data associated with the request is part of the **params** field.

Time Stamps

The messages sent by the Agent may contain the timestamp indicating the time at which the data is read from the ASIC. The timestamp is reported in the YYYY-MM-DD - HH:MM:SS format.

A sample JSON report snippet containing a timestamp is provided below.

```
    ...
    "method": "get-bst-report",
    "asic-id": "2",
    "time-stamp": "2014-10-18 - 00:15:04 ",
    "report": [{
     ...
```

Version

The response message from the Agent contains a field called *version*. This field contains an integer indicating the specific JSON payload version being used by the Agent. The version number is incremented for every set of changes introduced in the JSON payload. The client may use this version number information to parse/decipher the response messages from the Agent.

Appendix A: “JSON Payload Revision History” lists the changes introduced in the JSON payload with various versions.

Multi-ASIC Platforms

The Agent supports platforms with more than one switch ASIC device. The configuration command/reporting is on a per-ASIC basis. The configuration commands, responses, and notification reports carry the ASIC identifier as part of the message.

All parts of the REST API supported by the BroadView Instrumentation Agent take `asic-id` as a parameter via the JSON requests. Similarly, all the responses and notification messages from the Agent include the `asic-id` as a parameter in the response JSON. This is illustrated below.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-feature",
    "params": "all",
    "asic-id": "1",
    "id": 1232
}

    ...
    "method": "get-bst-report",
    "asic-id": "2",
    "time-stamp": "2014-10-18 - 00:15:04 ",
    "report": [{
     ...
```

For a single ASIC platform, the ASIC identifier (`asic-id`) part of the configuration command for response messages and notification reports is ignored by the Agent, and the corresponding value is set to 0.

This document describes the REST API for all supported features in detail, including specific parameters for the request/response messages. These descriptions do not include the `asic-id` parameter to avoid repetitive duplication.

Error Reporting

The Agent uses the standard HTTP error reporting mechanism for reporting any errors. The error codes are listed in Table 2.

Table 2: Error Codes

HTTP Status Code	***Description***
HTTP 200	Indicates that the request successfully executed on the Agent. If the request is for status/data, the response accompanies this status.[a]
HTTP 400	Indicates a JSON error in the request message. The JSON message is badly formatted and contains errors.
HTTP 404	Indicates an error. The method requested is not supported.
HTTP 500	Indicates an error. There was an internal error on the Agent that caused this.

[a] An HTTP 200 status indicates successful execution only at the transport and agent level, but not necessarily at the application/feature level. In most cases, they are the same. However, in some corner cases, they may be different. Where applicable, the JSON response messages indicate application-level execution status, and the client is required to parse the response.

Each of the errors listed above (HTTP error codes 4xx and 5xx) have a JSONRPC payload in the response packet to give additional error details in the JSON payload. An example of an HTTP 404 error is provided below.

```
HTTP/1.1 404 Not Found
Server: BroadViewAgent (Unix) (Linux)
Content-Type: text/json

{
       "jsonrpc": "2.0",
       "version": "1",
       "error": {
              "code": -32601,
              "message": "Method not found"
       },
       "id": 1
}
```

Section 4: Generic Agent Management

Overview

The BroadView Instrumentation Agent offers an API that helps clients and SDN controllers discover switches running the Agent and determine the switch capabilities, including both ASIC capabilities and the features supported by the Agent.

Heartbeat Messaging

Heartbeat messages are keep-alive messages sent by the Agent to the client. The heartbeat messaging allows clients to auto detect the switches running the Agent and to configure the Agent, without requiring manual intervention. The heartbeat message is in the form of REST/JSON notification messages.

Upon configuration, the Agent sends periodic heartbeat messages to a client. The messages contain the Agent and ASIC capabilities. By default, the heartbeat[2] messages are sent every five seconds. Upon discovering the Agent, the client can start configuring the Agent and use the BroadView capabilities for monitoring the switch/ network. Optionally, the Agent can reduce the frequency with which the heartbeat messages are sent to the client.

The heartbeat mechanism allows the registration of a specific switch with BroadView Agent capability but does not provide information about any connectivity of the switch to other switches.

[2] Sometimes the heartbeat messages prior to Client configuration of the Agent are called registration/discovery messages. The content of the message is the same, independent of the name.

Switch Properties

The switch properties are a set of parameters describing the Agent capabilities as well as the switch information. This set of parameters packed inside a JSON message becomes the heartbeat message. It can also be retrieved on-demand via the `get-switch-properties` API.

The parameters are described in Table 3.

Table 3: Switch Properties Parameters

Parameter	*Type*	*Description*
number-of-asics	Integer	Number of ASICs on the switch.
asic-info	Multi-parameter	Describes each of the ASICs onboard. More details are below in Table 4.
supported-features	Array of strings	A list of strings indicating the features supported by the Agent.
network-os	String	The NOS currently used on the switch.
uid	String	Unique identifier for this switch. This unique ID is the key for the SDN controller to map the switch to the nodes existing in their discovery database.
agent-ip	String	IP address of the switch where the Agent is being run.
agent-port	String	TCP port number of the switch, at which the Agent is listening.
agent-sw-version	String	Software version number for the Agent.

The `asic-info` parameter is a multi-parameter, and each of the sub-parameters is described in Table 4. The `asic-info` parameter is encoded as a positional-parameter list in the JSON message.

Table 4: asic-info Subparameters

Position	*Parameter*	*Type*	*Description*
1	asic-id	String	Identifier for an ASIC on the switch. This ID must be used in all subsequent interactions with the Agent to refer to this ASIC.
2	chip-id	String	Indicates the part number of the silicon.
3	num-ports	Integer	Number of ports available on the switch, managed by this ASIC.

A sample heartbeat message is provided below.

```
{
  "jsonrpc": "2.0",
  "method": "get-switch-properties",
  "version": "2",
  "time-stamp": "2015-10-18 - 00:15:04",
  "result": {
    "number-of-asics": 1,
    "asic-info": [
      [
        "1",
        "BCM56850",
        78
      ]
    ],
    "supported-features": [
      "BST"
    ],
    "network-os": "openNSL",
    "uid": "0000d80bb99bbbbb",
    "agent-ip": "192.168.1.2",
    "agent-port": "8080",
    "agent-sw-version": "3.0.0.1"
  },
  "id": 10
}
```

Configurability

Configurability is provided under the Generic Agent Management category.

The Agent can be configured to enable/disable heartbeat messaging to the client, as well as the interval at which the registration messages are to be sent.

API

Overview

This section lists various APIs supported for the Generic Agent Management. A description of each API is in subsequent sections.

Table 5: APIs for Generic Agent Management

API	*Type*	*Description*
“configure-system-feature”	Configuration	Configure global Agent management functionality.
“get-port-queue-map”	Status/Reporting	Obtain the hardware-queue to user-queue mapping.
“get-system-feature”	Status/Reporting	Retrieve Agent management configuration.
“get-switch-properties”	Status/Reporting	Retrieve the switch capabilities.
“cancel-request”[a]	Configuration	Stop a previously initiated periodic reporting command.

[a] This API is not specific to any single feature, and may be used to cancel any previously initiated periodic reporting command.

Configuration API

configure-system-feature

The `configure-system-feature` API sets up the core Agent functionality on the switch. The parameters associated with this command are described in Table 6.

Table 6: configure-system-feature Parameters

Parameter	*Type*	*Description*
heartbeat-enable	Boolean	When enabled, the Agent asynchronously sends the registration and heartbeat message to the collector. By default, the sending discovery message is turned on.
msg-interval	Integer (range 1 to 600)	Determines the interval in seconds with which the registration and heartbeat messages are sent to the collector. The default value is 5 seconds.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "configure-system-feature",
    "params": {
        " heartbeat-enable": 1,
        "msg-interval": 10
      },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

cancel-request

The cancel-request API is a generic API, and it cancels a previously configured periodic reporting request, such as get-bst-congestion-drop-counters on the Agent. The parameter associated with this command is described in Table 7.

Table 7: cancel-request API

Parameter	*Type*	*Description*
request-id	Integer	The request-id parameter is the ID of the request made already by the client. If this parameter value is zero (0), then the agent cancels all previously initiated periodic reporting requests.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "cancel-request",
    "asic-id": "1",
    "params": {
        "request-id": 7
      },
    "id": 4
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

Status/Reporting API

get-port-queue-map

The `get-port-queue-map` command is used to retrieve the hardware queue to user-assigned queue mapping for every port. There are no parameters for this command.

A sample JSON-RPC request to get the port to queue mapping is shown below.

```
{
    "jsonrpc": "2.0",
       "method": "get-port-queue-map",
       "asic-id": "1",
       "params": { },
        "id": 1
}
```

The response parameters are as shown in Table 8. These parameters are provided for both unicast queues and multicast queues for each of the ports.

Table 8: get-port-queue-map Response Parameters

Parameter	Type	Description
hw-queue	Integer	The queue number as supported by the hardware
user-queue	Integer	The queue number for the specified port, for which the hw-queue is assigned to.

A sample port-queue map for the above request is as follows:

```
{
  "jsonrpc": "2.0",
  "method": "get-port-queue-map",
  "asic-id": "1",
  "version": "4",
  "result": [
    {
      "port": "1",
      "unicast-queue-map": [
        {
          "hw-queue": 0,
          "user-queue": 0
        },
        {
          "hw-queue": 1,
          "user-queue": 1
        },
        {
          "hw-queue": 2,
          "user-queue": 2
        },
        {
          "hw-queue": 3,
          "user-queue": 3
        },
        {
          "hw-queue": 4,
          "user-queue": 4
        },
        {
          "hw-queue": 5,
          "user-queue": 5
        },
        {
          "hw-queue": 6,
          "user-queue": 6
        },
        {
          "hw-queue": 7,
          "user-queue": 7
        }
      ],
      "multicast-queue-map": [
        {
          "hw-queue": 0,
          "user-queue": 0
        },
        {
          "hw-queue": 1,
          "user-queue": 1
        },
        {
          "hw-queue": 2,
          "user-queue": 2
        },
        {
          "hw-queue": 3,
          "user-queue": 3
        },
        {
          "hw-queue": 4,
          "user-queue": 4
        },
        {
          "hw-queue": 5,
          "user-queue": 5
        },
        {
          "hw-queue": 6,
          "user-queue": 6
        },
        {
          "hw-queue": 7,
          "user-queue": 7
        }
      ]
    },
    {
      "port": "2",
      "unicast-queue-map": [
        {
          "hw-queue": 8,
          "user-queue": 0
        },
        {
          "hw-queue": 9,
          "user-queue": 1
        },
        {
          "hw-queue": 10,
          "user-queue": 2
        },
        {
          "hw-queue": 11,
          "user-queue": 3
        },
        {
          "hw-queue": 12,
          "user-queue": 4
        },
        {
          "hw-queue": 13,
          "user-queue": 5
        },
        {
          "hw-queue": 14,
          "user-queue": 6
        },
        {
          "hw-queue": 15,
          "user-queue": 7
        }
      ],
      "multicast-queue-map": [
        {
          "hw-queue": 8,
          "user-queue": 0
        },
        {
          "hw-queue": 9,
          "user-queue": 1
        },
        {
          "hw-queue": 10,
          "user-queue": 2
        },
        {
          "hw-queue": 11,
          "user-queue": 3
        },
        {
          "hw-queue": 12,
          "user-queue": 4
        },
        {
          "hw-queue": 13,
          "user-queue": 5
        },
        {
          "hw-queue": 14,
          "user-queue": 6
        },
        {
          "hw-queue": 15,
          "user-queue": 7
        }
      ]
    },
    {
      "port": "3",
      "unicast-queue-map": [
        {
          "hw-queue": 16,
          "user-queue": 0
        },
        {
          "hw-queue": 17,
          "user-queue": 1
        },
        {
          "hw-queue": 18,
          "user-queue": 2
        },
        {
          "hw-queue": 19,
          "user-queue": 3
        },
        {
          "hw-queue": 20,
          "user-queue": 4
        },
        {
          "hw-queue": 21,
          "user-queue": 5
        },
        {
          "hw-queue": 22,
          "user-queue": 6
        },
        {
          "hw-queue": 23,
          "user-queue": 7
        }
      ],
      "multicast-queue-map": [
        {
          "hw-queue": 16,
          "user-queue": 0
        },
        {
          "hw-queue": 17,
          "user-queue": 1
        },
        {
          "hw-queue": 18,
          "user-queue": 2
        },
        {
          "hw-queue": 19,
          "user-queue": 3
        },
        {
          "hw-queue": 20,
          "user-queue": 4
        },
        {
          "hw-queue": 21,
          "user-queue": 5
        },
        {
          "hw-queue": 22,
          "user-queue": 6
        },
        {
          "hw-queue": 23,
          "user-queue": 7
        }
      ]
```

```
    },
    {
      "port": "4",
      "unicast-queue-map": [
        {
          "hw-queue": 24,
          "user-queue": 0
        },
        {
          "hw-queue": 25,
          "user-queue": 1
        },
        {
          "hw-queue": 26,
          "user-queue": 2
        },
        {
          "hw-queue": 27,
          "user-queue": 3
        },
        {
          "hw-queue": 28,
          "user-queue": 4
        },
        {
          "hw-queue": 29,
          "user-queue": 5
        },
        {
          "hw-queue": 30,
          "user-queue": 6
        },
        {
          "hw-queue": 31,
          "user-queue": 7
        }
      ],
      "multicast-queue-map": [
        {
          "hw-queue": 24,
          "user-queue": 0
        },
        {
          "hw-queue": 25,
          "user-queue": 1
        },
        {
          "hw-queue": 26,
          "user-queue": 2
        },
        {
          "hw-queue": 27,
          "user-queue": 3
        },
        {
          "hw-queue": 28,
          "user-queue": 4
        },
        {
          "hw-queue": 29,
          "user-queue": 5
        },
        {
          "hw-queue": 30,
          "user-queue": 6
        },
        {
          "hw-queue": 31,
          "user-queue": 7
        }
      ]
    },
    {
      "port": "5",
      "unicast-queue-map": [
        {
          "hw-queue": 32,
          "user-queue": 0
        },
        {
          "hw-queue": 33,
          "user-queue": 1
        },
        {
          "hw-queue": 34,
          "user-queue": 2
        },
        {
          "hw-queue": 35,
          "user-queue": 3
        },
        {
          "hw-queue": 36,
          "user-queue": 4
        },
        {
          "hw-queue": 37,
          "user-queue": 5
        },
        {
          "hw-queue": 38,
          "user-queue": 6
        },
        {
          "hw-queue": 39,
          "user-queue": 7
        }
      ],
      "multicast-queue-map": [
        {
          "hw-queue": 32,
          "user-queue": 0
        },
        {
          "hw-queue": 33,
          "user-queue": 1
        },
        {
          "hw-queue": 34,
          "user-queue": 2
        },
        {
          "hw-queue": 35,
          "user-queue": 3
        },
        {
          "hw-queue": 36,
          "user-queue": 4
        },
        {
          "hw-queue": 37,
          "user-queue": 5
        },
        {
          "hw-queue": 38,
          "user-queue": 6
        },
        {
          "hw-queue": 39,
          "user-queue": 7
        }
      ]
    },
    {
      "port": "6",
      "unicast-queue-map": [
        {
          "hw-queue": 40,
          "user-queue": 0
        },
        {
          "hw-queue": 41,
          "user-queue": 1
        },
        {
          "hw-queue": 42,
          "user-queue": 2
        },
        {
          "hw-queue": 43,
          "user-queue": 3
        },
        {
          "hw-queue": 44,
          "user-queue": 4
        },
        {
          "hw-queue": 45,
          "user-queue": 5
        },
        {
          "hw-queue": 46,
          "user-queue": 6
        },
        {
          "hw-queue": 47,
          "user-queue": 7
        }
      ],
      "multicast-queue-map": [
        {
          "hw-queue": 40,
          "user-queue": 0
        },
        {
          "hw-queue": 41,
          "user-queue": 1
        },
        {
          "hw-queue": 42,
          "user-queue": 2
        },
        {
          "hw-queue": 43,
          "user-queue": 3
        },
        {
          "hw-queue": 44,
          "user-queue": 4
        },
        {
          "hw-queue": 45,
          "user-queue": 5
        },
        {
          "hw-queue": 46,
          "user-queue": 6
        },
        {
          "hw-queue": 47,
          "user-queue": 7
        }
      ]
    }
    ],
    "id": 1
}
```

The Agent returns the requested map immediately.

The Agent acknowledges the command and returns an appropriate error code if needed.

get-system-feature

The `get-system-feature` API retrieves the current Agent configuration. There is no parameter associated with this API. The parameters that form the response message are the same as described in the "configure-system-feature" API.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-system-feature",
    "params": { },
    "id": 1
}
```

An associated sample response is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-system-feature",
    "version": "2",
    "result": {
        " heartbeat-enable": 1,
        "msg-interval": 5
    },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

get-switch-properties

The `get-switch-properties` API retrieves the switch properties as described in "Switch Properties". There are no parameters associated with this command.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-switch-properties",
    "asic-id": "1",
    "params": { },
    "id": 1
}
```

An associated sample response is shown below.

```
{
  "jsonrpc": "2.0",
  "method": "get-switch-properties",
  "version": "2",
  "time-stamp": "2015-10-18 - 00:15:04",
  "result": {
    "number-of-asics": 1,
    "asic-info": [
      [
        "1",
        "BCM56850",
        78
      ]
    ],
    "supported-features": [
      "BST"
    ],
    "network-os": "openNSL",
    "uid": "0000d80bb99bbbbb",
    "agent-ip": "192.168.1.2",
    "agent-port": "8080",
    "agent-sw-version": "3.0.0.1"
  },
  "id": 10
}
```

Section 5: Buffer Statistics and Tracking

Overview

The Buffer Statistics and Tracking (BST) feature provides a means to retrieve how each of the on-chip buffers are being utilized under different traffic scenarios. A collection of all such buffer-counts for various on-chip buffers is the BST report. This BST report is sent from the Agent to the collector either on-demand or periodically.

BST tracks either the current values or peak values of the buffer utilization. Also, BST can selectively track certain buffer categories.

BST allows thresholds to be set for some of the buffers and can send a notification to the client when the buffer usage exceeds the configured threshold.

BST allows clients to monitor for top ports suffering congestion or monitor the number of packets dropped due to congestion, on a per-port, per-queue-type (broadcast or multicast), or per-port-per-queue basis.

Buffer Utilization Statistics and Thresholds

Broadcom® switch ASICs track buffer utilization for various on-chip buffers. The buffer utilization statistics are presented under the corresponding categories, called realms. To access (statistics of a) a buffer, the realm parameters must be provided. These parameters act as indices for the given buffer. For example, to access a buffer in the `egress-uc-queue` realm, the parameter is the corresponding queue ID. Similarly, to access a buffer in the `ingress-port-priority-group` realm, the parameters are the port ID and the priority-group ID. The `device` realm has no indices. No realm has more than two indices.

Each buffer, given its parameters, may have one or more statistic. For example, the `ingress-port-service-pool` realm buffer, for a given port and service pool ID, offers a single statistic: `um-share-buffer-count`. The `egress-port-service-pool` buffer, for a given port and service pool ID, offers four statistics: `uc-share-buffer-count`, `um-share-buffer-count`, `mc-share-buffer-count`, and `mc-share-queue-entries`. Each buffer statistic has an associated threshold for configuration. For example, for the `uc-share-buffer-count`, there is an associated `uc-share-threshold`.

Table 9 provides the list of realms and the associated indices for each realm. It also lists all available statistics and thresholds for each realm.

Table 9: Buffer Utilization Statistics and Thresholds

Realm	*Index #1*	*Index #2*	*Statistic*	*Threshold*
device	–	–	data	threshold
ingress-port-priority-group	port	priority-group	um-share-buffer-count, um-headroom-buffer-count	um-share-threshold, um-headroom-threshold

Table 9: Buffer Utilization Statistics and Thresholds (Cont.)

Realm	***Index #1***	***Index #2***	***Statistic***	***Threshold***
ingress-port-service-pool	port	service-pool	um-share-buffer-count	um-share-threshold
ingress-service-pool	service-pool	–	um-share-buffer-count	um-share-threshold
egress-port-service-pool	port	service-pool	uc-share-buffer-count, um-share-buffer-count, mc-share-buffer-count, mc-share-queue-entries	uc-share-threshold, um-share-threshold, mc-share-threshold, mc-share-queue-entries-threshold
egress-service-pool	service-pool	–	um-share-buffer-count, mc-share-buffer-count, mc-share-queue-entries	um-share-threshold, mc-share-threshold, mc-share-queue-entries-threshold
egress-uc-queue	queue	–	uc-buffer-count	uc-threshold
egress-uc-queue-group	queue-group	–	uc-buffer-count	uc-threshold
egress-mc-queue	queue	–	mc-buffer-count, mc-queue-entries	mc-threshold, mc-queue-entries-threshold
egress-cpu-queue	queue	–	cpu-buffer-count	cpu-threshold
egress-rqe-queue	queue	–	rqe-buffer-count	rqe-threshold

Configurability

The following configurability is provided by the BST feature:

- The BST feature can be enabled or disabled on the Agent.
- The BST feature can be configured to auto-accumulate the buffer statistics periodically and send the BST report to the collector at configurable intervals (as notification messages).
- The BST feature can provide buffer statistics (BST report) on demand.
- Buffer statistics collection (in the switch and in the Agent) of various subsets of buffer statistics (such as device-level, various ingress-groups, etc.) can be selectively disabled or enabled.
- The ASIC can be set up to gather the statistics of current values or peak values for the statistics.
- The Agent can be configured to report the buffer statistics in units of bytes, or in the units of cells, or in terms of a percentage of buffer usage (in terms of allocated levels).
- Various thresholds can be set up for the statistics in the ASIC, and the current values can be retrieved.
- The Agent can be set up to asynchronously send the buffer statistics report, called the *trigger report***,** when the buffer usage exceeds the configured threshold.

- The BST feature can provide the number of packets dropped within the ASIC because of the congestion, for top ports suffering congestion, at a per-port, per-queue-type (broadcast or multicast), or per-port-per-queue level.

Any changes made to the feature configuration come into effect starting with the next reporting period.

Complete and Incremental Reports

Keeping in mind the extensive data that is likely to be collected and communicated, the following constraints are imposed on the BST reports.

- All of the on-demand BST reports are complete in nature.
- All of the asynchronous BST reports are incremental in nature, unless configured otherwise.
  - The BST feature compares the statistics collected in the current cycle to that of the previous cycle and marks the changed statistics in its native representation.
  - The JSON packing from the native representation is aware of the marking and ignores those statistics that have not changed since the last cycle.
  - Any statistics that are not supported by the underlying ASIC or the SDK are explicitly encoded with special value in the JSON message. These statistics are not included in subsequent asynchronous reports (being incremental).

Queues

The ASIC allows various on-chip queues to be assigned to individual ports. For example, a queue (say numbered 453) can be assigned as queue 4 to the port 45. Some of the ASIC allow any number of queues to be assigned to a port, where as some has a fixed assignment. This assignment is user driven. Thus, the queues are identified in two forms- the asic-specified-queue-number, and the combination of user-assigned-queue-number and the port number.

The asic-specified-queue-number is referred to as `hw-queue`, or simply `queue` in this document and in the JSON messaging. The user assigned-queue-number is referred to as `user-queue`.

The API provided by the BST feature, which have the queue information specified in response or request, allow both the formats. More specifically,

- The API requests needing queue to be specified, can accept *either of the formats*.
- The response/report messages contains the queue information in *both the formats*.

Deciphering Reports

While reporting statistics for a given realm, the statistics names are not included in the report. A JSON positional parameter method is used. In addition, one (and not more than one) of the indices is included as the first element in the array. This scheme is used to reduce the JSON message size for report messages. This scheme is applicable only for BST reports and threshold reports. For configuration messages to the Agent, all parameters are named JSON parameters.

This method is illustrated below with a few examples.

Example 1

Consider the following report snippet:

```
{
            "realm": "ingress-service-pool",
            "data": [[1, 3240], [2, 3660]]
}
```

The `ingress-service-pool` has a single index (`service pool`), and it is mentioned in the JSON array. See Table 9: “Buffer Utilization Statistics and Thresholds,” on page 32.

This report indicates the following.

- The realm is the `ingress-service-pool`.
- The service pool with ID 1 has the `uc-share-buffer-count` value of 3240.
- The service pool with ID 2 has the `uc-share-buffer-count` value of 3660.

Example 2

Consider the following report snippet:

```
{
            "realm": "ingress-port-priority-group",
            "data": [{
                    "port": "2",
                    "data": [[5, 45500, 44450]]
                }, {
                    "port": "3",
                    "data": [[5, 6700, 250], [7, 12667, 13456]]
                }]
}
```

The `ingress-port-priority-group` has two indices: `port` and the `priority-group`. See Table 9: “Buffer Utilization Statistics and Thresholds,” on page 32. The top-level index (`port`) is mentioned explicitly, and the second index (`priority-group`) is included as the first element in the JSON array.

This report indicates the following:

- The realm is the `ingress-port-priority-group`.
- The port 2 report has the following entries:
  - `priority-group` id 5 has

        - The `um-share-buffer-count` value is 45500.
        - The `um-headroom-buffer-count` value is 44450.
- The port 3 report has the following entries:
    - `priority-group` id 5 has
        - The `um-share-buffer-count` value is 6700.
        - The `um-headroom-buffer-count` value is 250.
    - `priority-group` id 7 has
        - The `um-share-buffer-count` value is 12267.
        - The `um-headroom-buffer-count` value is 13456.

Example 3

Consider the following report snippet:

```
{
            "realm": "egress-mc-queue",
            "data": [[1, "1", 1, 34, 89], [10, "2", 1, 1244, 0]]
}
```

The `egress-mc-queue` has a single index (queue). It is included as the first element in the JSON array. The second element is *port*, to which queue is assigned. The third element is the *user-queue*.

This report indicates the following:

- The realm is the `egress-mc-queue.`
- The Queue #1 (port #1 and user queue #1) has the following entries:
    - The `mc-buffer-count` value is 34.
    - The `mc-queue-entries` value is 89.
- The Queue #10 (port #2 and user queue #1) has the following entries:
    - The `mc-buffer-count` value is 1244.
    - The `mc-queue-entries` value is 0.

Example 4

Consider the following report snippet:

```
{
            "realm": "egress-uc-queue",
            "data": [[1, "1", 1, 34]]
}
```

The `egress-uc-queue` has a single index (queue). It is included as the first element in the JSON array. The second element is *port* to which queue is assigned. The third element is the *user-queue*.

This report indicates the following:

- The realm is the `egress-uc-queue.`
- The Queue #1 (port #1 and user queue #1) has the following entry
    - The `uc-buffer-count` value is 34.

The *port* is included as a second parameter for the buffer usage report and threshold reports for the `egress-uc-queue` and `egress-mc-queue` realms. Similarly the *user-queue* is included as the third parameter.

API

Overview

This section lists APIs supported by the BST feature. A description of each API is in subsequent sections.

Table 10: Buffer Statistics and Tracking APIs

API	***Type***	***Description***
"configure-bst-feature"	Configuration/Clear	Configure global options for BST and reporting.
"configure-bst-tracking"	Configuration/Clear	Setup tracking parameters.
"configure-bst-thresholds"	Configuration/Clear	Configure threshold/watermarks for specific buffers.
"clear-bst-statistics"	Configuration/Clear	Clear all statistics.
"clear-bst-thresholds"	Configuration/Clear	Clear all configured thresholds.
"get-bst-feature"	Status/Reporting	Retrieve current BST configuration.
"get-bst-tracking"	Status/Reporting	Retrieve current tracking configuration.
"get-bst-thresholds"	Status/Reporting	Retrieve current threshold configuration.
"get-bst-report"	Status/Reporting	Obtain a snapshot/incremental buffer usage report.
"Trigger Reports"	Notification	Asynchronous report indicating a breach of a configured threshold.
"get-bst-congestion-drop-counters"	Status/Reporting	Obtain number of packets dropped due to congestion on a per-port/per-queue basis, immediately or periodically.
"clear-bst-congestion-drop-counters"	Configuration/Clear	Clear the congestion drop counters.
"cancel-request"[a]	Configuration/Clear	Stop a previously initiated periodic reporting command.

[a] This API is not specific to the BST feature and may be used to cancel any previously initiated periodic reporting command.

Configuration/Clear API

configure-bst-feature

The configure-bst-feature API configures the BST functionality on the Agent[3]. The parameters associated with this command are described in Table 11.

[3] But not on the ASIC. See the next command.

Table 11: configure-bst-feature Parameters

Parameter	***Type***	***Description***
bst-enable	Boolean	Determines whether the BST feature should be active or not, on the Agent. By default, the BST feature is inactive on the Agent.
send-async-reports	Boolean	When enabled, the BST feature asynchronously collects the buffer statistics and sends the BST reports to the collector. By default, the asynchronous reporting is turned off.
collection-interval	Integer (range 0 to 600)	Determines the periodicity with which the BST reports are sent to the collector. The Agent may also use this value to read the statistics from the ASIC. The units for this parameter are seconds. The default value is 60 seconds.
stats-in-percentage	Boolean	When enabled, the Agent reports the buffer usage statistics as a percentage. When `stats-in-percentage` is enabled, the parameter `stat-units-in-cells`[a] is ignored while reporting the statistics. This field is applicable for statistics and threshold reporting. By default, the value is set to false. The percentage value in the BST/trigger report is an approximation of buffer utilization, not an exact value.
stat-units-in-cells	Boolean	Determines whether the Agent reports the buffer statistics in the units of bytes or in the units of cells. Set to *false*, by default. The Agent reports the usage in bytes.
trigger-rate-limit	Integer (range 1 to 30)	Determines the maximum number of trigger reports for the configured interval (as specified by the `trigger-rate-limit-interval` parameter below) to be sent to the collector. The default value is 1.
trigger-rate-limit-interval	Integer (range 1 to 60)	Determines the interval during which the number of trigger-reports is rate limited (`trigger-rate-limit`) for sending to the collector. The default value is 1 second.
send-snapshot-on-trigger	Boolean	Determines whether the Agent should send a complete buffer statistics report for all configured realms to the collector, when a threshold is breached. If set to *true*, trigger-report contains buffer statistics for all configured realms. If set to *false*, trigger-report contains buffer statistics only for the statistic/ counter for which the trigger was raised. Default value is *true*.
async-full-reports	Boolean	When enabled, the BST feature asynchronously sends full BST reports to the collector. By default, the asynchronous full reporting is turned off.

[a] Cells are buffers inside the ASIC that are used to hold parts of a packet. Refer to the hardware documentation for details.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "configure-bst-feature",
    "asic-id": "1",
    "params": {
        "bst-enable": 1,
        "send-async-reports": 1,
        "collection-interval": 300,
        "stat-units-in-cells": 0,
        "trigger-rate-limit": 5,
        "trigger-rate-limit-interval": 2,
        "send-snapshot-on-trigger": 1,
        "async-full-reports": 1,
        "stats-in-percentage": 0
      },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

configure-bst-tracking

The configure-bst-tracking command sets up the BST trackers and the tracking-mode on the ASIC. The parameters associated with the command are described in

Table 12: configure-bst-tracking Parameters

Parameter	***Type***	***Description***
track-peak-stats	Boolean	Determines whether ASIC tracks the current buffer usage count or the peak buffer usage. Set to *false*, by default. The ASIC tracks current buffer count.
track-ingress-port-priority-group	Boolean	When enabled, the ASIC actively tracks ingress per-port and per-priority group buffers. By default, this tracking is turned on.
track-ingress-port-service-pool	Boolean	When enabled, the ASIC actively tracks ingress per-port and per-service pool buffers. By default, this tracking is turned on.
track-ingress-service-pool	Boolean	When enabled, the ASIC actively tracks ingress per-service pool buffers. By default, this tracking is turned on.
track-egress-port-service-pool	Boolean	When enabled, the ASIC actively tracks egress per-port and per-service pool buffers. By default, this tracking is turned on.
track-egress-service-pool	Boolean	When enabled, the ASIC actively tracks egress per-service pool buffers. By default, this tracking is turned on.
track-egress-uc-queue	Boolean	When enabled, the ASIC actively tracks egress per unicast queue buffers. By default, this tracking is turned on.

Table 12: configure-bst-tracking Parameters (Cont.)

Parameter	*Type*	*Description*
track-egress-uc-queue-group	Boolean	When enabled, the ASIC actively tracks egress per unicast queue group buffers. By default, this tracking is turned on.
track-egress-mc-queue	Boolean	When enabled, the ASIC actively tracks egress per multicast queue buffers. By default, this tracking is turned on.
track-egress-cpu-queue	Boolean	When enabled, the ASIC actively tracks egress per CPU queue buffers. By default, this tracking is turned on.
track-egress-rqe-queue	Boolean	When enabled, the ASIC actively tracks egress per RQE queue buffers. By default, this tracking is turned on.
track-device	Boolean	When enabled, the ASIC actively tracks per-device (ASIC) buffers. By default, this tracking is turned on.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "configure-bst-tracking",
    "asic-id": "1",
    "params": {
        "track-peak-stats" : 1,
        "track-ingress-port-priority-group" : 1,
        "track-ingress-port-service-pool" : 1,
        "track-ingress-service-pool" : 1,
        "track-egress-port-service-pool" : 1,
        "track-egress-service-pool" : 1,
        "track-egress-uc-queue" : 1,
        "track-egress-uc-queue-group" : 1,
        "track-egress-mc-queue" : 1,
        "track-egress-cpu-queue" : 1,
        "track-egress-rqe-queue" : 1,
        "track-device" : 1
    },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

configure-bst-thresholds

The `configure-bst-thresholds` command sets up the BST thresholds for various realms in the ASIC.

When a threshold is configured, and the associated buffer usage exceeds the threshold value, then an asynchronous notification is sent by the Agent.

The parameters associated with this command depend on the realm for which the thresholds are being set up. The list of realms and associated parameters/thresholds for each realm is provided in Buffer Utilization Statistics and Thresholds.

At any point in time, the valid set of thresholds depends on the MMU configuration of the Broadcom ASIC. Consult the Broadcom ASIC documentation for supported combinations.

Threshold configurations are in terms of number of bytes, number of cells, or percentage of allocated buffer size. The actual unit is determined by the Agent based on its current configuration for reporting the statistics set via `configure-bst-feature` command. For example, when the Agent is configured to report statistics in terms of percentage via `configure-bst-feature` command, then the Agent accepts threshold configuration in terms of percentage.

Sample JSON-RPC requests are shown below.

```
{
    "jsonrpc": "2.0",
    "method": "configure-bst-thresholds",
    "asic-id": "1",
    "params": {
        "realm": "egress-rqe-queue",
        "queue": 1,
        "rqe-threshold": 15156
    },
    "id": 1
}

{
    "jsonrpc": "2.0",
    "method": "configure-bst-thresholds",
    "asic-id": "1",
    "params": {
        "realm": "egress-port-service-pool",
        "port": "1",
        "service-pool": 3,
        "uc-share-threshold": 15156,
        "um-share-threshold": 15156,
        "mc-share-threshold": 15156,
        "mc-share-queue-entries-threshold": 15156
    },
    "id": 1
}

{
    "jsonrpc": "2.0",
    "method": "configure-bst-thresholds",
    "asic-id": "1",
    "params": {
        "realm": "egress-mc-queue",
        "port": "105",
        "user-queue": 0,
        "mc-threshold": 100
    },
    "id": 5
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

clear-bst-statistics

The `clear-bst-statistics` command clears all the BST statistics on the Agent[4]. This API does not require any parameters.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "clear-bst-statistics",
    "asic-id": "1",
    "params": {
     },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

clear-bst-thresholds

The `clear-bst-thresholds` command clears all configured BST thresholds on the Agent. This API does not require any parameters.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "clear-bst-thresholds",
    "asic-id": "1",
    "params": {
    },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

Instead of clearing all thresholds, clearing a specific threshold can be achieved by setting the corresponding threshold to 0 by using the `configure-bst-thresholds` command.

clear-bst-congestion-drop-counters

The `clear-bst-congestion-drop-counters` command clears all the Congestion Drop counters on the Silicon. This API does not require any parameters.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "clear-bst-congestion-drop-counters",
    "asic-id": "1",
    "params": {
     },
    "id": 1
```

[4] A side effect of this command is that the following asynchronous BST report will be complete, including all realms.

```
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

Status/Reporting API

get-bst-feature

The `get-bst-feature` command is used to retrieve the current configuration of the BST functionality on the Agent.   This API does not require any parameters.

For the response, the Agent returns the same parameters as sent via the `configure-bst-feature` command.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-feature",
    "asic-id": "1",
    "params": { },
    "id": 1
}
```

An associated sample response is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-feature",
    "asic-id": "1",
    "version": "1",
    "result": {
        "bst-enable": 1,
        "send-async-reports": 0,
        "collection-interval": 200,
        "stats-in-percentage": 0,
        "stat-units-in-cells": 0,
        "trigger-rate-limit": 0,
        "trigger-rate-limit-interval": 0,
        "send-snapshot-on-trigger": 0
    },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

get-bst-tracking

The `get-bst-tracking` command is used to retrieve the current BST tracking configuration of the Broadcom ASIC. This API does not require any parameters.

For the response, the Agent returns the same parameters as those sent via the `configure-bst-tracking` command.

This command is valid even when the feature is disabled on the Agent.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-tracking",
    "asic-id": "1",
    "params": { },
    "id": 1
}
```

An associated sample response is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-tracking",
    "asic-id": "1",
    "version": "1",
    "result": {
        "track-peak-stats" : 1,
        "track-ingress-port-priority-group" : 1,
        "track-ingress-port-service-pool" : 1,
        "track-ingress-service-pool" : 1,
        "track-egress-port-service-pool" : 1,
        "track-egress-service-pool" : 1,
        "track-egress-uc-queue" : 1,
        "track-egress-uc-queue-group" : 1,
        "track-egress-mc-queue" : 1,
        "track-egress-cpu-queue" : 1,
        "track-egress-rqe-queue" : 1,
        "track-device" : 1     },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

get-bst-thresholds

The `get-bst-thresholds` command is used to retrieve the currently configured BST thresholds from the Agent. The collector may choose to request a selective report (see parameters below).

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-thresholds",
    "asic-id": "1",
    "params": {
        "include-ingress-port-priority-group" : 1,
        "include-ingress-port-service-pool" : 1,
        "include-ingress-service-pool" : 1,
        "include-egress-port-service-pool" : 1,
        "include-egress-service-pool" : 1,
        "include-egress-uc-queue" : 1,
        "include-egress-uc-queue-group" : 1,
        "include-egress-mc-queue" : 1,
        "include-egress-cpu-queue" : 1,
        "include-egress-rqe-queue" : 1,
        "include-device" : 1
    },
    "id": 1
}
```

The Agent always returns a complete report for the requested thresholds.

The thresholds in the ASIC are set by the Agent only as part of the `configure-bst-thresholds` command. Unlike statistics, the threshold configuration does not change on its own. For this reason, it is not necessary to retrieve a periodic threshold report from the Agent.

Threshold reporting is in terms of bytes, cells, or percentage of maximum allocated buffer size. The actual reporting unit is determined by the Agent based on its current configuration for reporting the statistics via the `configure-bst-feature` command. For example, when the Agent is configured to report statistics in terms of percentage via the `configure-bst-feature` command, the Agent reports thresholds in terms of percentage.

The `method` object of the JSON message in the BST report is set to `get-bst-thresholds` for the response message.

A sample BST threshold report indicating various realms (categories) and the associated data is shown below. It is provided for illustrative purposes only and does not include all port/queue/pool data in it. Rather, the JSON array is used to indicate the multiplicity while providing data for a single element.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-thresholds",
    "asic-id": "20",
    "version": "1",
    "time-stamp": "2014-11-14 - 00:15:04 ",
    "report": [{
            "realm": "device",
            "data": 46
        }, {
            "realm": "ingress-port-priority-group",
            "data": [{
                    "port": "2",
                    "data": [[5, 45500, 44450]]
                }, {
                    "port": "3",
                    "data": [[5, 45500, 44450]]
                }]
        }, {
            "realm": "ingress-port-service-pool",
            "data": [{
                    "port": "2",
                    "data": [[5, 324]]
                }, {
                    "port": "3",
                    "data": [[6, 366]]
                }]
        }, {
            "realm": "ingress-service-pool",
            "data": [[1, 3240], [2, 3660]]
        }, {
            "realm": "egress-cpu-queue",
            "data": [[3, 4566, 0]]
        }, {
            "realm": "egress-mc-queue",
            "data": [[1, "1", 1, 34, 89], [2, "4", 2, 1244, 0], [3, "5", 1, 0, 3]]
        }, {
            "realm": "egress-port-service-pool",
            "data": [{
                    "port": "2",
                    "data": [[5, 0, 324, 0]]
                }, {
                    "port": "3",
                    "data": [[6, 0, 366, 0]]
                }]
        }, {
            "realm": "egress-rqe-queue",
            "data": [[2, 3333, 4444], [5, 25, 45]]
        }, {
            "realm": "egress-service-pool",
            "data": [[2, 0, 0, 3240], [3, 3660, 0, 0]]
        }, {
            "realm": "egress-uc-queue",
            "data": [[6, "1", 0, 1111]]
        }, {
            "realm": "egress-uc-queue-group",
            "data": [[6, 2222]]
        }]
}
```

get-bst-report

The `get-bst-report` command is used to retrieve the current BST report from the Agent. The collector may choose to request for selective report (see parameters in Table 13).

Table 13: get-bst-report Parameters

Parameter	*Type*	*Description*
include-ingress-port-priority-group	Boolean	When set, the Agent includes the ingress per-port per-priority group buffer statistics into the report.
include-ingress-port-service-pool	Boolean	When set, the Agent includes the ingress per-port per-service pool buffer statistics into the report.
include-ingress-service-pool	Boolean	When set, the Agent includes the ingress per-service pool buffer statistics into the report.
include-egress-port-service-pool	Boolean	When set, the Agent includes the egress per-port per-service pool buffer statistics into the report.
include-egress-service-pool	Boolean	When set, the Agent includes the egress per-service pool buffer statistics into the report.
include-egress-uc-queue	Boolean	When set, the Agent includes the egress per-unicast queue buffer statistics into the report.
include-egress-uc-queue-group	Boolean	When set, the Agent includes the egress per-unicast queue group buffer statistics into the report.
include-egress-mc-queue	Boolean	When set, the Agent includes the egress per-multicast queue buffer statistics into the report.
include-egress-cpu-queue	Boolean	When set, the Agent includes the egress per-CPU queue buffer statistics into the report.
include-egress-rqe-queue	Boolean	When set, the Agent includes the ingress per-RQE queue buffer statistics into the report.

Table 13: get-bst-report Parameters (Cont.)

Parameter	Type	Description
include-device	Boolean	When set, the Agent includes the ingress device level buffer statistics into the report.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-report",
    "asic-id": "1",
    "params": {
        "include-ingress-port-priority-group" : 1,
        "include-ingress-port-service-pool" : 1,
        "include-ingress-service-pool" : 1,
        "include-egress-port-service-pool" : 1,
        "include-egress-service-pool" : 1,
        "include-egress-uc-queue" : 1,
        "include-egress-uc-queue-group" : 1,
        "include-egress-mc-queue" : 1,
        "include-egress-cpu-queue" : 1,
        "include-egress-rqe-queue" : 1,
        "include-device" : 1
    },
    "id": 1
}
```

The Agent always returns a complete report for the requested buffer statistics.

It must be noted that the Agent returns the BST report following the command. However, the asynchronous reporting cycle may be reset.

The `method` object of the JSON message in the BST report is set to `get-bst-report` for both the asynchronous notifications as well as the response message.

The parameters associated with this command depend on the realm for which the buffer usage (count) is being reported. The list of realms and associated parameters / buffer-count for each realm is provided in "Buffer Utilization Statistics and Thresholds".

It may be noted that at any point in time, the valid set of buffer counts (usage) depends on the MMU configuration of the Broadcom ASIC. Refer to the Broadcom ASIC documentation for supported configurations.

A sample BST report indicating various realms (categories) and the associated statistics is shown below. It is provided for illustrative purposes only and does not include all the realm/port/queue/pool data in it. Rather, the JSON array is used to indicate the multiplicity while providing data for a single element.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-report",
    "asic-id": "20",
    "version": "1",
    "time-stamp": "2014-11-18 - 00:15:04 ",
    "report": [{
            "realm": "device",
            "data": 46
```

```
        }, {
            "realm": "ingress-port-priority-group",
            "data": [{
                    "port": "2",
                    "data": [[5, 45500, 44450]]
                }, {
                    "port": "3",
                    "data": [[5, 45500, 44450]]
                }]
        }, {
            "realm": "ingress-port-service-pool",
            "data": [{
                    "port": "2",
                    "data": [[5, 324]]
                }, {
                    "port": "3",
                    "data": [[6, 366]]
                }]
        }, {
            "realm": "ingress-service-pool",
            "data": [[1, 3240], [2, 3660]]
        }, {
            "realm": "egress-cpu-queue",
            "data": [[3, 4566, 0]]
        }, {
            "realm": "egress-mc-queue",
            "data": [[1, "1", 1, 34, 89], [2, "4", 2, 1244, 0], [3, "5", 2, 0, 3]]
        }, {
            "realm": "egress-port-service-pool",
            "data": [{
                    "port": "2",
                    "data": [[5, 0, 324, 0]]
                }, {
                    "port": "3",
                    "data": [[6, 0, 366, 0]]
                }]
        }, {
            "realm": "egress-rqe-queue",
            "data": [[2, 3333, 4444], [5, 25, 45]]
        }, {
            "realm": "egress-service-pool",
            "data": [[2, 0, 0, 3240], [3, 3660, 0, 0]]
        }, {
            "realm": "egress-uc-queue",
            "data": [[6, "1", 2, 1111]]
        }, {
            "realm": "egress-uc-queue-group",
            "data": [[6, 2222]]
        }]
}
```

get-bst-congestion-drop-counters

The `get-bst-congestion-drop-counters` command is used to retrieve the congestion drop counters periodically or immediately from the Agent. There are different sets of drop counters that can be retrieved by the collector. Based on the type of the set, the number of parameters passed to the command varies. The basic parameters are provided in Table 14.

Table 14: get-bst-congestion-drop-counters Parameters

Parameter	*Type*	*Description*
request-type	String	Indicates the specific set of drop counters being requested. The following sets are supported: • top-drops. Ports suffering maximum congestion in the switch and the associated drop counters. • top-port-queue-drops. Top port-queue level drop-counters in the switch. • port-drops. Per-port total drop counters. • port-queue-drops. Port-queue level drop-counters.
collection-interval	Integer (range 0 to 3600)	Determines the period with which the congestion drop counters are collected from the ASIC and reported to the client. The units for this parameter are seconds. The default value is 0 (zero), which indicates an immediate response.

Table 15 shows drop report parameters for the top congestion ports/port-queues report.

Table 15: Top Congestion Ports/Port-Queues Drop Report Parameters

Parameter	*Type*	*Description*
count	Integer (range 0 to 64)	Number of ports required in the report. The ports are placed in a sorted list, with the port suffering maximum congestion at the top. Out of this list, the count number of ports and their drop-counters are reported back to the client.
queue-type	String	This represents the queue type filter for the report. Possible values are: • ucast. Indicates unicast queues. • mcast. Indicates multicast queues. • all. Indicates all supported queue types in the ASIC.

Drop report parameters for (all/specific) congestion ports/port-queues report are listed in Table 16.

Table 16: All/Specific Congestion Ports/Port-Queues Drop Report Parameters

Parameter	*Type*	*Description*
port-list	Array	Comma separated list of ports for which congestion drop counter report is requested. A special keyword *all* is allowed to get congestion drop counters for all ports.

Table 16: All/Specific Congestion Ports/Port-Queues Drop Report Parameters (Cont.)

Parameter	***Type***	***Description***
queue-type	String	This represents the queue type filter for the report. Possible values are: • ucast. Indicates unicast queues. • mcast. Indicates multicast queues. • all. Indicates all supported queue types in the ASIC.
queue-list	Array	An array of queue numbers to be considered for the drop report.

A sample JSON-RPC request to get the top eight congestion drop counters per port is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-congestion-drop-counters",
    "asic-id": "1",
    "params": {
        "request-type" : "top-drops",
        "request-params": {
         "count":8
         },
         "collection-interval": 30
    },
    "id": 1
}
```

A sample BST congestion drop report for the above request is as follows:

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-congestion-drop-counters",
    "asic-id": "1",
    "version": "3",
    "time-stamp": "2016-1-1 - 00:15:04 ",
    "report": [{
            "report-type": "top-drops",
            "data": [{
                    "port": "2",
                    "data": 8500
                }, {
                    "port": "8",
                    "data": 7550
                }, {
                    "port": "6",
                    "data": 6500
                }, {
                    "port": "1",
                    "data": 5550
                }, {
                    "port": "7",
                    "data": 4500
                }, {
                    "port": "10",
                    "data": 3550
                }, {
                    "port": "20",
                    "data": 2500
                }, {
                    "port": "19",
                    "data": 1550
                }
            ]
       }],
    "id": 1
}
```

A sample JSON-RPC request to get the top eight congestion drop counters per port-queue is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-congestion-drop-counters",
    "asic-id": "1",
    "params": {
        "request-type" : "top-port-queue-drops",
        "request-params": {
        "count":8,
        "queue-type": "all"
         },
         "collection-interval": 30
    },
    "id": 1
}
```

A sample BST congestion drop report for the above request is as follows:

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-congestion-drop-counters",
    "asic-id": "1",
    "version": "3",
    "time-stamp": "2016-1-1 - 00:15:04 ",
    "report": [{
            "report-type": "top-port-queue-drops",
            "data": [{
                    "port": "2",
                    "queue-type": "ucast",
                    "data": [[1,8500],[4,8400]]
                }, {
                    "port": "8",
                    "queue-type": "mcast",
                    "data": [[4,7550]]
                }, {
                    "port": "6",
                    "queue-type": "ucast",
                    "data": [[2,6500],[1,5400]]
                }, {
                    "port": "2",
                    "queue-type": "mcast",
                    "data": [[3,4500],[1,4400],[5,200]]
                }
            ]
       }],
    "id": 1
}
```

A sample JSON-RPC request to get all/specific port congestion drop counter information is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-congestion-drop-counters",
    "asic-id": "1",
    "params": {
        "request-type" : "port-drops",
        "request-params": {
        "port-list": ["1", "2", "3", "4"]
         },
         "collection-interval": 30
    },
    "id": 1
}
```

A sample BST congestion drop report for the above request is as follows:

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-congestion-drop-counters",
    "asic-id": "1",
    "version": "3",
    "time-stamp": "2016-1-1 - 00:15:04 ",
    "report": [{
            "report-type": "port-drops",
            "data": [{
                    "port": "1",
                    "data": 8500
                }, {
                    "port": "2",
                    "data": 27550
                }, {
                    "port": "3",
                    "data": 16500
                }, {
                    "port": "4",
                    "data": 5550
                }
            ]
       }],
    "id": 1
}
```

A sample JSON-RPC request to get all port-queue/ specific port-queue congestion drop counter information is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-congestion-drop-counters",
    "asic-id": "1",
    "params": {
        "request-type" : "port-queue-drops",
        "request-params": {
             "port-list": ["1", "2"],
             "queue-type": "all",
             "queue-list": [ 1, 2, 3]
         },
         "collection-interval": 30
    },
    "id": 1
}
```

A sample BST congestion drop report for the above request is as follows:

```
{
    "jsonrpc": "2.0",
    "method": "get-bst-congestion-drop-counters",
    "asic-id": "1",
    "version": "3",
    "time-stamp": "2016-1-1 - 00:15:04 ",
    "report": [{
            "report-type": "port-queue-drops",
            "data": [{
                    "port": "1",
                    "queue-type": "ucast",
                    "data": [[1,8500],[2,8400], [3,156000]]
                }, {
                    "port": "1",
                    "queue-type": "mcast",
                    "data": [[1,600],[2,400], [3,1000]]
                }, {
                    "port": "2",
                    "queue-type": "ucast",
                    "data": [[1,500],[2,800], [3,56000]]
                }, {
                    "port": "2",
                    "queue-type": "mcast",
                    "data": [[1,2600],[2,3400], [3,21000]]
                }]
       }],
    "id": 1
}
```

The Agent returns the requested drop counter report immediately. When the `collection-interval` is non-zero, the Agent continues to send requested drop counter reports at the configured interval asynchronously as notifications.

The Agent can be requested to serve multiple periodic reports.

The Agent acknowledges the command and returns an appropriate error code if needed.

Trigger Reports

When any of the configured threshold (via the `configure-bst-thresholds`) API is triggered, that is, the buffer usage crosses the configured threshold, an asynchronous notification is sent by the Agent. This notification is identical to an asynchronous BST report with the following two exceptions:

- The `method` object has the value of `trigger-report`.
- Additional fields are included to specify the actual counter causing the threshold breach.

A partial sample report is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "trigger-report",
    "asic-id": "20",
    "version": "1",
    "time-stamp": "2014-11-18 - 00:13:08 ",
    "realm": "ingress-port-priority-group",
    "counter": "um-share-buffer-count",
    "port": "2",
    "priority-group": "5",
    "report": [{
            "realm": "device",
            "data": 46
        }, {
            "realm": "ingress-port-priority-group",
            "data": [{
                    "port": "2",
                    "data": [[5, 45500, 44450]]
                }, {
                    "port": "3",
                    "data": [[5, 45500, 44450]]
                }]
        }, {
            "realm": "ingress-port-service-pool",
            "data": [{
                    "port": "2",
                    "data": [[5, 324]]
                }, {
                    "port": "3",
                    "data": [[6, 366]]
                }]
        }, {
            "realm": "ingress-service-pool",
            "data": [[1, 3240], [2, 3660]]
        }, {
            "realm": "egress-cpu-queue",
            "data": [[3, 4566, 0]]
        }, {
            "realm": "egress-mc-queue",
            "data": [[1, "1", 1, 34, 89], [2, "4", 2, 1244, 0], [3, "5", 1, 0, 3]]
        }, {
            "realm": "egress-port-service-pool",
            "data": [{
                    "port": "2",
                    "data": [[5, 0, 324, 0]]
                }, {
                    "port": "3",
                    "data": [[6, 0, 366, 0]]
                }]
        }, {
            "realm": "egress-rqe-queue",
            "data": [[2, 3333, 4444], [5, 25, 45]]
        }, {
            "realm": "egress-service-pool",
            "data": [[2, 0, 0, 3240], [3, 3660, 0, 0]]
        }, {
            "realm": "egress-uc-queue",
            "data": [[6, "1", 1, 1111]]
        }, {
            "realm": "egress-uc-queue-group",
            "data": [[6, 2222]]
        }]
}
```

Note: When a trigger situation is encountered (that is, any of the configured thresholds are breached), the ASIC freezes all the BST counters until the feature is re-enabled/re-configured. The Agent re-enables the BST on the ASIC upon a trigger notification and sends the trigger report to the client. The number of triggers in a given duration can be rate-limited via the `configure-bst-feature` command.

Section 6: Packet Trace

Overview

When a packet enters a switch, it is processed and forwarded out on one or more destination ports. Typically, the results of intermediate packet processing steps are not visible. The Packet Trace feature provides detailed information on these intermediate processes, like egress hashing information for LAG and ECMP. This detailed information is called a *trace-profile*.

The Packet Trace feature allows the client to inject a packet into the ingress packet processing pipeline that is then processed as if it were received on one of the front panel ports. The Packet Trace feature then logs the internal forwarding states. This can be useful in the diagnosis of unexpected errors or as an offline debugging tool.

The Packet Trace feature also provides an option to the client to match incoming live traffic packets against a client-specified match criterion and provide a *trace-profile* for such packets. In this case, the client can specify the packet match criterion instead of providing a packet to be injected.

In both the scenarios, the packet processing information—more specifically the egress port information for the packet—is provided back to the client.

Configurability

The following configurability is provided by the Packet Trace feature.

- The Packet Trace feature can be enabled or disabled on the Agent.
- The Packet Trace feature can be configured to collect trace-profile periodically for a client supplied packet and send trace-profile to the client at configurable interval (as notification reports).
- The Packet Trace feature can provide a trace-profile for a client-supplied packet on demand.
- The Packet Trace feature can be configured to drop or forward the packet injected by requester after the trace-profile is collected by the Agent.
- The Packet Trace feature can be configured to collect a trace-profile for all live traffic packets on the wire, matching user-supplied criteria and send a trace-profile to the client (as notification reports).

Packet Format

The BroadView Agent expects the packet to be in a PCAP format, which is a binary format. However, JSON has no data type to represent the binary data. This means that the collector must encode the PCAP in base64 format. The Agent decodes the PCAP information, which is in a base64 format, as binary and strips the PCAP global header and PCAP packet header before injecting the packet.

Each PCAP file can have multiple packets, but the BroadView Agent supports only one packet in a given base64-encoded PCAP data.

After decoding, the Agent checks the first four bytes for the magic number[5]. If the first four bytes is not the magic number, the input file data is ignored. The Agent also checks for the major and minor numbers. The supported major number is 2, and the supported minor number is 4. If the magic number and major and minor numbers are valid, then the Agent extracts the packet by removing the PCAP file header and the PCAP packet header.

Following is an example of the decoded packet (after base64 decoding) on the Agent.

```
a1 b2 c3 d4 0 2 0 4 0 0 0 0 0 0 0 0
0 0 ff ff 0 0 0 1 55 18 f3 f4 0 b 94 ea
0 0 0 78 0 0 0 78 11 22 33 44 55 66 77 88
99 aa bb cc 81 0 0 1 8 0 45 2 0 66 7f af
d3 a0 40 6 40 59 1 2 3 4 5 6 7 8 10 1
d2 24 10 2 2a 4 50 0 16 d0 5f eb d3 d0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 c 8a 6c f0
```

The Agent deciphers this packet as follows:

- The first four bytes are the magic number `(a1 b2 c3 d4).`
- The major version is 2 `(0 2),` which is two bytes.
- The minor version number `(0 4)` is two bytes.
- The total length of the PCAP file header is 24 bytes, and the PCAP packet length is 16 bytes.
- The four bytes from offset 32 indicate the captured packet length (`00 00 00 78`), and the subsequent four bytes `(00 00 00 78)` indicate the length of the packet on the wire.

If the captured packet length is smaller than the packet length on the wire, then the input is discarded, and an appropriate error code is returned by the Agent.

In this document, wherever a packet is used in a JSON request/response, a partial encoded packet with trailing dots, as shown below, is used for brevity.

```
"packet": "0010203232.."
```

[5] For more information about the PCAP magic number, refer to PCAP documentation available on the Internet.

Live Traffic Triggered Trace-Profile

Instead of the client providing a packet and requesting a trace-profile, the client can provide 5-tuple information and request a trace-profile for packets matching the specified criteria. The Agent copies the packets matching the conditions specified in the 5-tuple to the CPU, and then injects them to the switch ASIC to retrieve the packet trace profile. If this feature is supported, then the string `Live-PT` is included in the response to the API `get-switch-properties`.

The 5-tuple parameters are listed in Table 17.

Table 17: 5-Tuple Parameters

Parameter	*Type*	*Description*
src-ip	String	IP address is used to match the source IP address of the packet.
dst-ip	String	IP address is used to match destination IP address of the packet.
Protocol	String	Protocol number is used to match layer-4 protocol number in the packet.
l4-src-port	String	Layer-4 source port number is used to match the L4 source port of the packet.
l4-dst-port	String	Layer-4 destination port number is used to match the L4 destination port of the packet.

The client may specify a wild card for any of these parameters. A wild card indicates a “match any” condition for that parameter. A wild card is specified using the string *any* in the JSON request.

Up to three wild-cards in a given 5-tuple are accepted by the Agent.

Trace profile information is sent asynchronously by the Agent for traffic-triggered trace-profile requests.

Timestamps

The trace-profile data contains the parameters packet-received-time-stamp, packet-ingress-time-stamp and packet-egress-time-stamp.

- The packet-received-time-stamp indicates the CPU time when the packet is received.
- The packet-ingress-time-stamp indicates the ASIC time when the packet is received by the ASIC.
- The packet-egress-time-stamp indicates the ASIC time when the packet leaves the ASIC.

The ability for the Agent to report either/both of the timestamps, packet-ingress-time-stamp and packet-egress-time-stamp, is determined by the capability of the ASIC. If the ASIC does not provide the necessary data, the Agent does not include those fields in the response JSON.

Both of the timestamps, packet-ingress-time-stamp and packet-egress-time-stamp, are valid only for live traffic triggered trace-profile reports.

Deciphering Trace-Profile

The ASIC reports the results of the packet trace (trace-profile) for different switching and routing tables. The results are presented under various categories, called *realms*.

Table 18 provides the list of realms and the associated results for each realm. It also lists the available fields/ information for each realm.

Table 18: Deciphering Trace-Profile Realms

Realm	*Index*	*Trace-Profile Parameters*
lag-link-resolution	–	lag-id, lag-members, dst-lag-member, fabric-lag-id[a], fabric-lag-members, fabric-lag-dst-member
ecmp-link-resolution	ecmp-level	ecmp--group-id, ecmp-members, ecmp-dst-member, ecmp--dst-port, ecmp--next-hop-ip
link-resolution	–	dst-port

[a] The three parameters fabric-lag-id, fabric-lag-members and fabric-lag-dst-member are deprecated, and the trace-profile reports sent by the Agent do not hold any value in these parameters. These are kept in the JSON structure for maintaining backward compatibility and may be removed in a future version.

The LAG and ECMP resolution parameters are standard parameters and need no further explanation. The `dst-port` parameter in the link-resolution realm indicates the egress-port on which the packet exited the switch.

A sample trace-profile report is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-packet-trace-profile",
    "asic-id": "1",
    "version": "1",
    "time-stamp": "2014-11-18 - 00:15:04 ",
    "report": [
        {
            "port": "1",
            "trace-profile": [
                {
                    "realm": "lag-link-resolution",
                    "data": {
                        "lag-id": "2",
                        "lag-members": ["1", "2", "3", "4"],
                        "dst-lag-member": "4"
                    }
                },
                {
                    "realm": "ecmp-link-resolution",
                    "data": [
                        {
                            "ecmp-group-id": "200256",
                            "ecmp-members": [
                                ["100005", "3.3.3.2", "15"],
                                ["100006", "4.4.4.2", "16" ]
                                ],
                            "ecmp-dst-member": "100005",
                            "ecmp-dst-port": "15",
                            "ecmp-next-hop-ip": "3.3.3.2"
                        }
                    ]
         }
            ]
        },
        {
            "port": "2",
            "trace-profile": [
                {
                    "realm": "lag-link-resolution",
                    "data": {
                        "lag-id": "2",
                        "lag-members": ["1","2","3","4"],
                        "dst-lag-member": "4"
                    }
                },
                {
                    "realm": "ecmp-link-resolution",
                    "data": [
                        {
                            "ecmp-group-id": "200256",
                            "ecmp-members": [
               ["100005", "3.3.3.2", "15"],
              ["100006", "4.4.4.2", "16"]
                         ],
                            "ecmp-dst-member": "100005",
                            "ecmp-dst-port": "15",
                            "ecmp-next-hop-ip": "3.3.3.2"
                        }
                    ]
                }
            ]
        }
    ],
    "id": 1
}
```

API

Overview

This section lists various APIs supported by the Packet Trace feature. A detailed description of each API is available in subsequent sections.

Table 19: Packet Trace APIs

API	*Type*	*Description*
"configure-packet-trace-feature"	Configuration	Configure global options for packet trace and reporting.
"get-packet-trace-feature"	Status Reporting	Obtain the current configuration for packet trace.
"get-packet-trace-lag-resolution"	Status Reporting	Request the LAG resolution report for a given packet/ packet match criterion.
"get-packet-trace-ecmp-resolution"	Status Reporting	Request the ECMP resolution report for a given packet/ packet match criterion.
"get-packet-trace-profile"	Status Reporting	Request the complete packet resolution report for a given packet/packet match criterion.
"cancel-packet-trace-lag-resolution"	Configuration	Stop a previously initiated LAG resolution command.
"cancel-packet-trace-ecmp-resolution"	Configuration	Stop a previously initiated ECMP resolution command.
"cancel-packet-trace-profile"	Configuration	Stop a previously initiated Packet Trace profile command.
"cancel-request"	Configuration	Stop a previously initiated periodic reporting command.

Configuration/Clear API

configure-packet-trace-feature

The `configure-packet-trace-feature` command sets up the Packet Trace functionality on the Agent. The parameters associated with this command are described inTable 20.

Table 20: configure-packet-trace-feature Parameters

Parameter	*Type*	*Description*
packet-trace-enable	Boolean	Determines whether the Packet Trace feature should be active or not on the Agent. By default, the Packet Trace feature is inactive on the Agent.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "configure-packet-trace-feature",
    "asic-id": "1",
    "params": {
        "packet-trace-enable": 1
    },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

cancel-packet-trace-profile

The `cancel-packet-trace-profile` command is used to cancel the trace-profile request that has been made by the collector through the `get-packet-trace-profile` API. The parameters associated with this command are described in Table 21.

Table 21: cancel-packet-trace-profile Parameter

Parameter	***Type***	***Description***
id	Integer	Identifier of the trace-profile request made by the collector. The ID used in this command is the same as the ID of the corresponding `get-packet-trace-profile` request.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "cancel-packet-trace-profile",
    "asic-id": "1",
    "params": {
        "id": 2
    },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

cancel-packet-trace-lag-resolution

The `cancel-packet-trace-lag-resolution` command is used to cancel the lag-resolution request that has been made by the collector through the `get-packet-trace-lag-resolution` API. The parameter associated with this command is described in Table 22.

Table 22: cancel-packet-trace-lag-resolution Parameter

Parameter	***Type***	***Description***
id	Integer	The ID of the lag-resolution request already made by the Collector. The ID used in this command is same as the `id` parameter of the corresponding `get-packet-trace-lag-resolution` request.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "cancel-packet-trace-lag-resolution",
    "asic-id": "1",
    "params": {
        "id" : 2

    },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

cancel-packet-trace-ecmp-resolution

The `cancel-packet-trace-ecmp-resolution` command is used to cancel the ecmp-resolution request that has been made by the collector through the `get-packet-trace-ecmp-resolution` API.

Table 23: cancel-packet-trace-ecmp-resolution Parameter

Parameter	*Type*	*Description*
id	Integer	The ID of the ecmp-resolution request made by the collector. The ID used in this command is the same as the `id` parameter of the `get-packet-trace-ecmp-resolution` request.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "cancel-packet-trace-ecmp-resolution",
    "asic-id": "1",
    "params": {
        "id" : 2

    },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

Status/Reporting API

get-packet-trace-feature

The `get-packet-trace-feature` command is used to retrieve the current configuration of the Packet Trace functionality on the Agent.   This API does not require any parameters.

For the response, the Agent returns the same parameters as that of the `configure-packet-trace-feature` command.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-packet-trace-feature",
    "asic-id": "1",
    "params": {},
    "id": 1
}
```

An associated sample response is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-packet-trace-feature",
    "asic-id": "1",
    "version": "1",
    "result": {
        "packet-trace-enable": 1
    },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

get-packet-trace-lag-resolution

The `get-packet-trace-lag-resolution` command is used to retrieve the LAG resolution. There are two sets of parameters to this API depending on the following two possibilities:

- Option 1: The client prefers a trace-profile for an injected debug/PCAP packet.
- Option 2: The client prefers to specify a match criterion and needs a trace-profile for live traffic matching that criterion.[6]

The set of parameters for Option 1 are described in Table 24.

Table 24: get-packet-trace-lag-resolution Option 1 Parameters

Parameter	***Type***	***Description***
packet	String	The packet[a] to which the trace-profile is requested. The Requester has to send a full packet to the Agent. The Agent injects this packet into the ingress pipeline as it receives the packet.
port-list	Array of String	Determines the ports on which the trace-profile is requested.
collection-interval	Integer	Determines the periodicity with which the trace-profiles are collected from the ASIC. The units for this parameter are seconds. The default value for this is 60 seconds. The minimum value of interval is 0 second. If the value is 0, the trace-profile report is sent as response to request and periodic reporting is not enabled.
drop-packet	Boolean	Determines whether the ASIC has to drop the packet or forward it once the trace-profile is collected. The default value is true.

[a] The packet format is described in “Packet Format”.

The set of parameters for Option 2 are described in Table 25.

[6] This method is described in “Live Traffic Triggered Trace-Profile”.

Table 25: get-packet-trace-lag-resolution Option 2 Parameters

Parameter	***Type***	***Description***
src-ip	String	IP address is used to match the source IP address of the packet.
dst-ip	String	IP address is used to match destination IP address of the packet.
protocol	String	Protocol number is used to match the L4 protocol number in the packet.
l4-src-port	String	L4 source port number is used to match the L4 source port of the packet.
l4-dst-port	String	L4 destination port number is used to match the L4 destination port of the packet.
port-list	Array of String	Determines the ports on which trace-profile is requested.
packet-limit	Integer	Determines the maximum number of packet samples for configured interval (as specified by the collection-interval parameter below) to be sent to the collector. Default value is 5.
collection-interval	Integer	Determines the time period during which maximum of n (as configured using packet-limit) trace-profiles are collected from the ASIC. The units for this parameter are seconds. The default value for this is 10 seconds. The minimum value of interval is 1 second.
drop-packet	Boolean	Determines whether ASIC has to drop the packet or forward once the trace-profile is collected. Default value is true.

A sample JSON-RPC request containing the packet is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-packet-trace-lag-resolution",
    "asic-id": "1",
    "params": {
        "packet": "000001000002….",
        "port-list": ["1","2","3","4-10"],
        "collection-interval": 10,
        "drop-packet": 1
    },
    "id": 1
}
```

An associated sample response for the above requests is shown below. The response is asynchronous if the collection interval is non-zero.

```
{
    "jsonrpc": "2.0",
    "method": " get-packet-trace-lag-resolution",
    "asic-id": "1",
    "version": "3",
    "time-stamp": "2014-11-18 - 00:15:04 ",
    "packet-info": [{
       "packet-received-time-stamp": "2014-11-18 - 00:15:00 ",
       "packet-ingress-time-stamp": "2014-11-18 - 00:15:00 ",
       "packet-egress-time-stamp": "2014-11-18 - 00:15:00 ",
       "packet": "0010203232.."
   }],
    "report": [
        {
            "port": "1",
            "lag-link-resolution": {
                "lag-id": "2",
                "lag-members": [
                    "1",
                    "2",
                    "3",
                    "4"
                ],
                "dst-lag-member": "4"
            }
        },
        {
            "port": "2",
            "lag-link-resolution": {
                "lag-id": "2",
                "lag-members": [
                    "1",
                    "2",
                    "3",
                    "4"
                ],
                "dst-lag-member": "4"
            }
        }
    ],
    "id": 1
}
```

A sample JSON-RPC request containing 5-tuple information is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-packet-trace-lag-resolution",
    "asic-id": "1",
    "params": {
        "src-ip": "192.168.1.1",
        "dst-ip": "10.10.1.1",
        "protocol": "4",
        "l4-src-port": "27",
        "l4-dst-port": "35",
        "port-list": ["1", "2", "3", "4-10"],
        "packet-limit": 5,
        "collection-interval": 1,
        "drop-packet": 1
    },
    "id": 1
}
```

The profile information is sent asynchronously for the request if the 5-tuple match criterion is specified.

An associated sample response for the above request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": " get-packet-trace-lag-resolution",
    "asic-id": "1",
    "version": "3",
    "time-stamp": "2014-11-18 - 00:15:04 ",
   "packet-info": [{
       "packet-received-time-stamp": "2014-11-18 - 00:15:00 ",
       "packet-ingress-time-stamp": "2014-11-18 - 00:15:00 ",
       "packet-egress-time-stamp": "2014-11-18 - 00:15:00 ",
       "packet": "0010203232.."
   }],
    "report": [
        {
            "port": "1",
            "lag-link-resolution": {
                "lag-id": "2",
                "lag-members": [
                    "1",
                    "2",
                    "3",
                    "4"
                ],
                "dst-lag-member": "4"
            }
        },
        {
            "port": "2",
            "lag-link-resolution": {
                "lag-id": "2",
                "lag-members": [
                    "1",
                    "2",
                    "3",
                    "4"
                ],
                "dst-lag-member": "4"
            }
        }
    ],
   "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

get-packet-trace-ecmp-resolution

The `get-packet-trace-ecmp-resolution` API retrieves the ECMP resolution. There are two sets of parameters to this API depending on the two possibilities below.

- Option 1: The client prefers a trace-profile for a pre-available packet.
- Option 2: The client prefers to specify a match criterion and needs a trace-profile for live traffic matching that criterion.[7]

The set of parameters for Option 1 are described in Table 26.

[7] This method is described in "Live Traffic Triggered Trace-Profile".

Table 26: get-packet-trace-ecmp-resolution Option 1 Parameters

Parameter	Type	Description
packet	String	Packet to which trace-profile is requested. The requester has to send a full packet to the Agent, and the Agent will inject this packet into the ingress pipeline as if it received the packet.
port-list	Array of String	Determines the ports on which a trace-profile is requested.
collection-interval	Integer	Determines the periodicity with which the trace-profiles are collected from the ASIC. The units for this parameter are seconds. The default value for this is 60 seconds. The minimum value of interval is 0 second. If the value is 0, the trace-profile report is sent as response to request and periodic reporting is not enabled.
drop-packet	Boolean	Determines whether the ASIC has to drop the packet or forward once the trace-profile is collected. Default value is true.

The set of parameters for Option 2 are described in Table 27.

Table 27: get-packet-trace-ecmp-resolution Option 2 Parameters

Parameter	Type	Description
src-ip	String	IP address is used to match the source IP address of the packet.
dst-ip	String	IP address is used to match destination IP address of the packet.
protocol	String	Protocol number is used to match L4 protocol number in the packet.
l4-src-port	String	Layer-4 source port number is used to match the L4 source port of the packet.
l4-dst-port	String	Layer-4 destination port number is used to match the L4 destination port of the packet.
port-list	Array of String	Determines the ports on which trace-profile is requested.
packet-limit	Integer	Number of samples made by the Agent in the collection interval. Default value is 5.
collection-interval	Integer	Determines the time period during which maximum of n (as configured using packet-limit) trace-profiles are collected from the ASIC. The units for this parameter are seconds. The default value for this is 10 seconds. The minimum value of interval is 1 second.
drop-packet	Boolean	Determines whether the ASIC has to drop the packet or forward once the trace-profile is collected. Default value is true.

A sample JSON-RPC request containing packet is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-packet-trace-ecmp-resolution",
    "asic-id": "1",
    "params": {
        "packet": "000001000002….",
        "port-list": ["1","2","3","4-10"],
        "collection-interval": 10,
        "drop-packet": 1
    },
    "id": 1
}
```

An associated sample response is shown below. The response is asynchronous if the collection interval is non-zero.

```
{
    "jsonrpc": "2.0",
    "method": " get-packet-trace-ecmp-resolution",
    "asic-id": "1",
    "version": "3",
    "time-stamp": "2014-11-18 - 00:15:04 ",
    "packet-info": [{
       "packet-received-time-stamp": "2014-11-18 - 00:15:00 ",
       "packet-ingress-time-stamp": "2014-11-18 - 00:15:00 ",
       "packet-egress-time-stamp": "2014-11-18 - 00:15:00 ",
       "packet": "0010203232.."
   }],

    "report": [
        {
            "port": "1",
            "ecmp-link-resolution": [
                {
                    "ecmp-group-id": "200256",
                    "ecmp-members": [
                  ["100005", "3.3.3.2", "15"],
                  ["100006", "4.4.4.2", "16"]
                    ],
                    "ecmp-dst-member": "100005",
                    "ecmp-dst-port": "15",
                    "ecmp-next-hop-ip": "3.3.3.2"
                }
            ]
        },
        {
            "port": "2",
            "ecmp-link-resolution": [
                {
                    "ecmp-group-id": "200256",
                    "ecmp-members": [
                  ["100005", "3.3.3.2", "15"],
                  ["100006", "4.4.4.2", "16"]
                    ],
                    "ecmp-dst-member": "100005",
                    "ecmp-dst-port": "15",
                    "ecmp-next-hop-ip": "3.3.3.2"
                }
            ]
        }
    ],
    "id": 1
}
```

A sample JSON-RPC request containing 5-tuple information is shown below.

```
{
    "jsonrpc": "2.0",
    "method": " get-packet-trace-ecmp-resolution",
    "asic-id": "1",
    "params": {
        "src-ip": "192.168.1.1",
        "dst-ip": "10.10.1.1",
        "protocol": "4",
        "l4-src-port": "27",
        "l4-dst-port": "35",
        "port-list": ["1", "2", "3", "4-10"],
        "packet-limit": 5,
        "collection-interval": 1,
        "drop-packet": 1
    },
    "id": 1
}
```

An associated sample response is shown below. The profile information is sent asynchronously for the request if the 5-tuple match criterion is specified.

```
{
    "jsonrpc": "2.0",
    "method": " get-packet-trace-ecmp-resolution",
    "asic-id": "1",
    "version": "31",
    "time-stamp": "2014-11-18 - 00:15:04 ",
    "packet-info": [{
        "packet-received-time-stamp": "2014-11-18 - 00:15:00 ",
        "packet-ingress-time-stamp": "2014-11-18 - 00:15:00 ",
        "packet-egress-time-stamp": "2014-11-18 - 00:15:00 ",
        "packet": "0010203232.."
    }],

    "report": [{
        "port": "1",
        "ecmp-link-resolution": [{
            "ecmp-group-id": "200256",
            "ecmp-members": [
                    ["100005", "3.3.3.2", "15"],
                    ["100006", "4.4.4.2", "16"]

            ],
            "ecmp-dst-member": "100005",
            "ecmp-dst-port": "15",
            "ecmp-next-hop-ip": "3.3.3.2"
        }]
    }],
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

get-packet-trace-profile

The get-packet-trace-profile API retrieves the Packet Trace egress resolution. This API obtains the LAG resolution (get-packet-trace-lag-resolution), ECMP resolution (get-packet-trace-ecmp-resolution) and a link resolution from the ASIC.

There are two sets of parameters to this API depending on the following two possibilities:

- Option 1: The client prefers a trace-profile for a pre-available packet.
- Option 2: The client prefers to specify a match criterion and needs a trace-profile for live traffic matching that criterion.[8]

The Agent attempts to resolve LAG and ECMP for the packet(s) obtained. The response includes one of the following:

- The LAG resolution (same as response of resolution get-packet-trace-lag-resolution).
- The ECMP resolution (same as response of resolution get-packet-trace-ecmp-resolution).
- The link resolution (egress port information) if the packet could not be traced to either a LAG or ECMP.

The set of parameters for Option 1 are described in Table 28.

Table 28: get-packet-trace-profile Option 1 Parameters

Parameter	*Type*	*Description*
packet	String	Packet to which trace-profile is requested. Requester have to send full packet to Agent, Agent will inject this packet into the ingress pipeline as if it receives the packet.
port-list	Array of String	Determines the ports on which trace-profile is requested.
collection-interval	Integer	Determines the periodicity with which the trace-profiles are collected from the ASIC. The units for this parameter are seconds. The default value for this is 60 seconds. The minimum value of interval is 0 second. If the value is 0, the trace-profile report is sent as response to request and periodic reporting is not enabled.
drop-packet	Boolean	Determines whether ASIC has to drop the packet or forward once the trace-profile is collected. Default value is true.

The set of parameters for Option 2 are described in Table 29.

Table 29: get-packet-trace-profile Option 2 Parameters

Parameter	*Type*	*Description*
src-ip	String	IP address is used to match the source IP address of the packet.
dst-ip	String	IP address is used to match destination IP address of the packet.
protocol	String	Protocol number is used to match layer-4 protocol number in the packet.
l4-src-port	String	Layer-4 source port number is used to match the L4 source port of the packet.
l4-dst-port	String	Layer-4 destination port number is used to match the L4 destination port of the packet.
port-list	Array of String	Determines the ports on which trace-profile is requested.
packet-limit	Integer	Number of samples made by the Agent in the collection interval. Default value is 5.

[8] This method is described in "Live Traffic Triggered Trace-Profile".

Table 29: get-packet-trace-profile Option 2 Parameters (Cont.) (Cont.)

Parameter	Type	Description
collection-interval	Integer	Determines the time period during which maximum of n (as configured using packet-limit) trace-profiles are collected from the ASIC. The units for this parameter are seconds. The default value for this is 10 seconds. The minimum value of interval is 1 second.
drop-packet	Boolean	Determines whether ASIC has to drop the packet or forward once the trace-profile is collected. Default value is true.

A sample JSON-RPC request containing packet is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-packet-trace-profile",
    "asic-id": "1",
    "params": {
        "packet": "000001000002….",
        "port-list": ["1","2","3","4-10"],
        "collection-interval": 10,
        "drop-packet": 1
    },
    "id": 1
}
```

An associated sample response is shown below. The response is asynchronous if the collection interval is non-zero.

```
{
    "jsonrpc": "2.0",
    "method": "get-packet-trace-profile",
    "asic-id": "1",
    "version": "3",
    "time-stamp": "2014-11-18 - 00:15:04 ",
    "packet-info": [{
       "packet-received-time-stamp": "2014-11-18 - 00:15:00 ",
       "packet-ingress-time-stamp": "2014-11-18 - 00:15:00 ",
       "packet-egress-time-stamp": "2014-11-18 - 00:15:00 ",
       "packet": "0010203232.."
   }],

    "report": [

{
            "port": "1",
            "trace-profile": [
                {
                    "realm": "lag-link-resolution",
                    "data": {
                        "lag-id": "2",
                        "lag-members": ["1", "2", "3", "4"],
                        "dst-lag-member": "4"
                    }
                },
                {
                    "realm": "ecmp-link-resolution",
                    "data":
                       [
                        {
                            "ecmp-group-id": "200256",
                            "ecmp-members": [
                  ["100005", "3.3.3.2", "15"],
                  ["100006", "4.4.4.2", "16"]],
                            "ecmp-dst-member": "100005",
                            "ecmp-dst-port": "15",
                            "ecmp-next-hop-ip": "3.3.3.2"
                        }
                    ]
                },
        {
           "realm": "link-resolution",
           "data": [{
                "dst-port": "15"
           }]
              }
            ]
        },
        {
            "port": "2",
            "trace-profile": [
                {
                    "realm": "lag-link-resolution",
                    "data": {
                        "lag-id": "2",
                        "lag-members": ["1", "2", "3", "4"],
                        "dst-lag-member": "4"
                    }
                },
                {
                    "realm": "ecmp-link-resolution",
                    "data":
                            [
                        {
                            "ecmp-group-id": "200256",
                            "ecmp-members": [
                  ["100005", "3.3.3.2", "15"],
                  ["100006", "4.4.4.2", "16"]],
                            "ecmp-dst-member": "100005",
                            "ecmp-dst-port": "15",
                            "ecmp-next-hop-ip": "3.3.3.2"
                        }
                    ]
                },
        {
           "realm": "link-resolution",
           "data": [{
                "dst-port": "15"
           }]
              }
            ]
        }
    ],
    "id": 1
}
```

A sample JSON-RPC request containing 5-tuple information is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-packet-trace-lag-resolution",
    "asic-id": "1",
    "params": {
        "src-ip": "192.168.1.1",
        "dst-ip": "10.10.1.1",
        "protocol": "4",
        "l4-src-port": "27",
        "l4-dst-port": "35",
        "port-list": ["1", "2", "3", "4-10"],
        "packet-limit": 5,
        "collection-interval": 1,
        "drop-packet": 1
    },
    "id": 1
}
```

An associated sample response is shown below. The profile information is sent asynchronously for the request if the 5-tuple match criterion is specified.

```
{
    "jsonrpc": "2.0",
    "method": "get-packet-trace-profile",
    "asic-id": "1",
    "version": "13",
    "time-stamp": "2014-11-18 - 00:15:04 ",
    "packet-info": [{
        "packet-received-time-stamp": "2014-11-18 - 00:15:00 ",
        "packet-ingress-time-stamp": "2014-11-18 - 00:15:00 ",
        "packet-egress-time-stamp": "2014-11-18 - 00:15:00 ",
        "packet": "0010203232.."
    }],

    "report": [{
        "port": "1",
        "trace-profile": [{
            "realm": "lag-link-resolution",
            "data": {
                "lag-id": "2",
                "lag-members": ["1", "2", "3", "4"],
                "dst-lag-member": "4"
            }
        }, {
            "realm": "ecmp-link-resolution",
            "data": [{
                "ecmp-group-id": "200256",
                "ecmp-members": [
                    ["100005", "3.3.3.2", "15"],
                    ["100006", "4.4.4.2", "16"]],
                "ecmp-dst-member": "100005",
                "ecmp-dst-port": "15",
                "ecmp-next-hop-ip": "3.3.3.2"
            }]
        },
        {
           "realm": "link-resolution",
           "data": [{
               "dst-port": "15"
           }]
              }
    }],
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

Section 7: Black Hole Detection

Overview[9]

Black holes refer to logical places in the network where incoming or outgoing traffic is silently discarded. A black hole condition arises when the traffic is directed towards an incorrect path, which causes the packets to traverse the same set of nodes in a loop. This continues until the packet's Time-to-Live (TTL) expires, at which point the packet is discarded. This condition is called *black holing* as the packet fails to reach the destination node.

A black hole in the data plane is a symptom of an inconsistent configuration of one or more routing tables in the network data plane. Such inconsistency may be the result of a benign condition that will heal by itself, such as a link or switch failure, and the awareness of the failure has not fully propagated to the entire network. The inconsistency might also be a persistent condition such as a hardware bug, a software defect, or a configuration mistake.

The BroadView Agent allows clients to monitor for potential black holes in the network. Upon appropriate configuration, the Agent notifies the client of a likely black hole, while also providing the specific packets that are being black holed. It allows network operators to separate the benign from the problematic causes, take immediate corrective action to avoid persistent packet loss, and help isolate the root cause.

Optionally, the BroadView Agent can also send samples of black holed packets to an sFlow receiver.

Black Hole Ports

In a typical spine-leaf topology, the uplink ports on the leaf switches connecting back to spine switches are called *spine ports*. Packets that enter leaf switches from spine switches tend not to go back to spine switches. If a packet comes into a leaf switch from one of the spine ports and egresses from another of the spine ports, it is usually a symptom of black holing. Such packets are likely to be forward within the same spine-leaf-spine loop and will eventually be discarded.

In the context of this feature, the set of ports where the packet cannot both ingress and egress are called *black hole ports*. Typically, these are the spine ports.

[9] A white paper describing the Black Hole Detection and Avoidance feature is available on the Broadcom.com website. Refer to *Black Hole Detection by BroadView™ Instrumentation Software Abstract* (BHD-WP101-R.pdf)

Reporting Black Hole Events

The BroadView Instrumentation Agent monitors the black hole ports and reports each black holing event to the configured client. The event report includes the following information:

- Ingress port
- Egress port
- Packet that caused the event

This information enables the client to take appropriate corrective action. It is also possible to configure the Agent to send the black holing packets to an sFlow receiver connected to one of the switch ports.

Configurability

The following configurability is provided by the BHD feature.

- The BHD feature can be enabled or disabled on the Agent.
- The client provides the list of black hole ports.
- The BHD feature can be configured to monitor for black holing packets and send black holing event reports to the client (as notification reports).
- The client can configure a watermark level for black holing packets (in packets/sec) to ignore transient conditions.
- The BHD feature can be configured to send the black holing packets to an sFlow receiver connected to one of the switch ports.

Some of the BHD capabilities, such as reporting to an sFlow receiver, require specific ASIC capabilities. If the client configuration cannot be supported on the underlying hardware, the Agent returns an appropriate error.

API

Overview

This section lists APIs supported by the Black Hole Detection (BHD) feature. A detailed description of each of the APIs is available in subsequent sections.

Table 30: Black Hole Detection APIs

API	Type	Description
“black-hole-detection-enable”	Configuration	Configure global options for BHD.
“configure-black-hole”	Configuration	Defines a port set constituting a black hole and reporting parameters.
“cancel-black-hole”	Configuration	Clears a previously set black hole configuration and reporting parameters.
“get-black-hole-detection-enable”	Status Reporting	Obtain current BHD configuration.
“get-black-hole”	Status Reporting	Obtain the current black hole configuration and reporting parameters.
“get-black-hole-event-report”	Notification	Report from Agent indicating a black-holed packet event.
“get-sflow-sampling-status”	Status Reporting	Obtain the sFlow reporting status.
“cancel-request”	Configuration	Stop a previously initiated periodic reporting command.

Configuration API

black-hole-detection-enable

The `black-hole-detection-enable` command enables the Black Hole Detection functionality on the switch. The parameter associated with this command are described in Table 31.

Table 31: black-hole-detection-enable Parameter

Parameter	Type	Description
enable	Boolean	When enabled, the BroadView Agent enables the Black Hole Detection feature. When disabled, Black Hole Detection feature is disabled on the Agent and any existing configuration is discarded. By default, the feature is turned OFF.

A sample JSON-RPC request to enable BHD is shown below.

```
{
    "jsonrpc": "2.0",
    "method": " black-hole-detection-enable",
    "asic-id": "1",
    "params": {
        "enable": 1
    },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

configure-black-hole

The `configure-black-hole` command configures the Black Hole Detection functionality on the switch. The parameters associated with this command are described in Table 32.

Table 32: configure-black-hole Parameters

Parameter	***Type***	***Description***
port-list	String	List of ports where the traffic is monitored for black holes.
sampling-method	String Enumeration	Sampling method used to sample packets. Supported values are *agent* and *sflow*.

Sampling parameters for Agent sampling are provided in Table 33.

Table 33: configure-black-hole Agent Sampling Parameters

Parameter	***Type***	***Description***
water-mark	Integer	This represents the traffic rate in packet/second, above which traffic is considered as black holed. Sampling starts only after the watermark level is crossed. The default is 1024.
sample-periodicity	Integer	Time interval in seconds. The default is 10 seconds.
sample-count	Integer	Number of samples to be sent for every *sample-periodicity* interval. The default is 2. If sample-count is set to 0, then sampling is stopped.

Sampling parameters for sFlow sampling are provided in Table 34.

Table 34: configure-black-hole sFlow Sampling Parameters

Parameter	***Type***	***Description***
mirror-port	String	Port number on which sFlow encapsulated sample packet is sent out.
vlan-id	Integer	The VLAN identifier of the sFlow encapsulation header.
destination-ip	String	Destination IP address in the sFlow encapsulation header.
source-udp-port	Integer	Source UDP port number in the sFlow encapsulation header.
destination-udp-port	Integer	Destination UDP port number in the sFlow encapsulation header.
sample-pool-size	Integer	Represents the packet pool size for sampling. One packet is sampled out of *sample-pool-size* packets. Minimum is 1024. The default is 1024.

A sample JSON-RPC request to configure BHD using Agent sampling is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "configure-black-hole ",
    "asic-id": "1",
    "params": {
        "port-list": ["1", "2", "3", "4"],
        "sampling-method": "agent",
        "sampling-params": {
            "water-mark": 200,
            "sample-periodicity": 15,
            "sample-count": 10
        }
    },
    "id": 1
}
```

A sample JSON-RPC request to configure BHD using sFlow agent sampling is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "configure-black-hole ",
    "asic-id": "1",
    "params": {
        "port-list": ["1", "2", "3", "4"],
        "sampling-method": "sflow",
        "sampling-params": {
            "encapsulation-params": {
                "vlan-id": 1,
                "destination-ip": "1.1.1.1",
                "source-udp-port": 1234,
                "destination-udp-port": 4321
            },
            "mirror-port": "10",
            "sample-pool-size": 1024
        }
    },
    "id": 1
}
```

The Agent acknowledges the command, and returns an appropriate error code if needed.

cancel-black-hole

The `cancel-black-hole` API removes all black hole configurations on the switch. There are no parameters associated with this command.

A sample JSON-RPC request to cancel black hole is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "cancel-black-hole",
    "asic-id": "1",
    "params": {
    },
    "id": 1
}
```

The Agent acknowledges the command and returns an appropriate error code if needed.

Status/Reporting API

get-black-hole-detection-enable

The `get-black-hole-detection-enable` API retrieves the current status of the BHD feature on the Agent. There are no parameters to this API. The parameters for response are same as that of "black-hole-detection-enable" API.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-black-hole-detection-enable",
    "asic-id": "1",
    "params": {},
    "id": 1
}
```

An associated sample response is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-black-hole-detection-enable",
    "asic-id": "1",
    "version": "2",
    "result": {
        "enable": 1
    },
    "id": 1
}
```

get-black-hole

The `get-black-hole` API retrieves the current configuration of the Black Hole Detection feature on the Agent. There are no parameters to this API. The parameters for response are same as that of "configure-black-hole" API.

A sample JSON-RPC request is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-black-hole",
    "asic-id": "1",
    "params": {},
    "id": 1
}
```

An associated sample response is shown below for Agent sampling.

```
{
    "jsonrpc": "2.0",
    "method": "get-black-hole",
    "asic-id": "1",
    "version": "2",
    "result": {
        "port-list": ["1", "2", "3", "4"],
        "sampling-method": "agent",
        "sampling-params": {
            "water-mark": 200,
            "sample-periodicity": 15,
            "sample-count": 10
        }
    },
    "id": 1
}
```

An associated sample response is shown below for SFlow sampling.

```
{
    "jsonrpc": "2.0",
    "method": "get-black-hole",
    "asic-id": "1",
    "version": "2",
    "result": {
        "port-list": ["1", "2", "3", "4"],
        "sampling-method": "sflow",
        "sampling-params": {
            "encapsulation-params": {
                "vlan-id": 1,
                "destination-ip": "1.1.1.1",
                "source-udp-port": 1234,
                "destination-udp-port": 4321
            },
            "mirror-port": "10",
            "sample-pool-size": 1
        }
    },
    "id": 1
}
```

get-sflow-sampling-status

The get-sflow-sampling-status API retrieves the current sFlow status for black-holed traffic. The parameter associated with this command is described in Table 35.

Table 35: get-sflow-sampling-status Parameter

Parameter	***Type***	***Description***
port-list	String	List of ports on which the sFlow sampling status is requested.

The REST JSON request message for the same is shown below.

```
 {
    "jsonrpc": "2.0",
    "method": "get-sflow-sampling-status",
    "asic-id": "1",
    "params": {
         "port-list":["1","2"]
    },
    "id": 1
}
```

The parameters associated with the response are described in Table 36.

Table 36: get-sflow-sampling-status Response Parameters

Parameter	*Type*	*Description*
port	String	Port number.
sflow-sampling-enabled	Boolean	sFlow sampling status.
sampled-packet-count	Integer	Total number of packets sampled since sFlow sampling is enabled.
black-holed-packet-count	Integer	Total number of packets black-holed since sFlow sampling is enabled.

An associated sample response is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-sflow-sampling-status",
    "asic-id": "1",
    "time-stamp": "2014-11-18 - 00:15:04 ",
        "version": "2",
    "report": {
        "data": [{
            "port": "2",
            "sflow-sampling-enabled": 1,
            "sampled-packet-count": 100,
            "black-holed-packet-count": 1000
        }, {
            "port": "3",
            "sflow-sampling-enabled": 1,
            "sampled-packet-count": 200,
            "black-holed-packet-count": 2000
        }]
    },
    "id": 1
}
```

Notifications

get-black-hole-event-report

The REST JSON message `get-black-hole-event-report` is used to send asynchronous black hole event reports, including the sampled packet, to the client. This is applicable only when the Agent is configured in Agent sampling mode.

The parameters associated with this notification are described in Table 37.

Table 37: get-black-hole-event-report Parameters

Parameter	*Type*	*Description*
ingress-port	String	Port number on which the packet has entered the switch.
egress-port-list	String	List of egress ports where the packet would be sent out.

Table 37: get-black-hole-event-report Parameters

Parameter	*Type*	*Description*
black-holed-packet-count	Integer	Total number of packets black holed.
sample-packet	String	Sampled packet in PCAP format with BASE 64 encoding.

A sample notification message is shown below.

```
{
    "jsonrpc": "2.0",
    "method": "get-black-hole-event-report",
    "asic-id": "1",
    "version": "2",
    "time-stamp": "2014-11-18 - 00:15:04 ",
    "report": {
        "ingress-port": "1",
        "egress-port-list": ["2",  "3"],
        "black-holed-packet-count": 100,
        "sample-packet": "0010203232.."
    }
}
```

Sampled packets larger than 1588 bytes are truncated and might appear as malformed at the client side.

Section 8: Reference Applications

Also available, along with the Agent, is the BroadView Analytics application. This application is Java-based, runs on a PC, and can interact with multiple switches running the BroadView Agent. This application showcases the visibility and telemetry information provided by the Agent.

Section 9: License

Section 10: Source Code

BroadView software is available in two packages:

- An OEM and ODM Development Package (ODP), which is a full source code package distributed under Broadcom SLA.
- A Community Development Package (CDP), which is an Open API library with Application Development Kit distributed on GitHub.

Section 11: Documentation

Technical documents are located in the GitHub repository, including:

- Compile/Build procedure

Appendix A: JSON Payload Revision History

Table 38: JSON Payload Revision History

Previous Version	*New Version*	*Change Description*	*Example*

0	1	New fields introduced in response message for `get-bst-feature` command.
		• stats-in-percentage
		• async-full-reports
		• trigger-rate-limit
		• trigger-rate-limit-interval
		• send-snapshot-on-trigger
			`{`
			`…`
			`"method": "get-bst-feature",`
			`"result": {`
			`"stat-in-percentage": 0,`
			`"async-full-reports": 1,`
			`"trigger-rate-limit": 0,`
			`"trigger-rate-limit-interval": 0,`
			`"send-snapshot-on-trigger": 0`
			`},`
			`…`
			`}`

0	1	New fields introduced in the notification message of `trigger-report`:
		• realm
		• counter
			`{`
			`…`
			`"method": "trigger-report",`
			`"realm": "ingress-port-priority-group",`
			`"counter": "um-share-buffer-count",`
			`…`
			`}`

1	2	New fields introduced in the response message for get-switch-properties command:
		• uid
		• agent-ip
		• agent-port
		• time-stamp
			`{`
			`…`
			`"method": "get-switch-properties",`
			`"time-stamp": "2015-10-18 - 00:15:04",`
			`"uid": "0000d80bb99bbbbb",`
			`"agent-ip": "192.168.1.2",`
			`"agent-port": "8080"`
			`…`
			`}`

Table 38: JSON Payload Revision History (Cont.)

Previous Version	New Version	Change Description	Example

2	3	New fields introduced in the response/ notification messages for the following commands:
		• get-packet-trace-lag-resolution,
		• get-packet-trace-ecmp-resolution
		• get-packet-trace-profile
		• packet-info
		• packet-received-time-stamp
		• packet-ingress-time-stamp
		• packet-egress-time-stamp
		• packet
			`{`
			`….`
			`"time-stamp": "2014-11-18 - 00:15:04 ",`
			`"packet-info": [{`
			`      "packet-received-time-stamp": "2014-11-18 - 00:15:00",`
			`       "packet-ingress-time-stamp": "2014-11-18 - 00:15:00",`
			`        "packet-egress-time-stamp": "2014-11-18 - 00:15:00",`
			`        "packet": "0010203232.."`
			` }],`
			`….`
			`}`

3	4	New API get-port-queue-map in BST.
		New fields introduced in the API/ response/notification messages to support hardware-queue to user-queue mapping.
			`{`
			`    "jsonrpc": "2.0",`
			`    "method": "configure-bst-thresholds",`
			`    "asic-id": "1",`
			`    "params": {`
			`        "realm": "egress-uc-queue",`
			`        "port": "2",`
			`        "user-queue": 0,`
			`        "uc-threshold": 1000`
			`    },`
			`    "id": 1`
			`}`

Section 12: Revision History

Revision	*Date*	*Change Description*
BroadView-SP100-R	04/26/17	Initial release

Web: www.broadcom.com
Corporate Headquarters: San Jose, CA