OpenBox Framework Specification

Version 1.1.0 - Draft - January 2016

1. OpenBox Service Instance

An OpenBox service instance (OBI) is an application that performs one or more processing blocks in the processing pipeline. Depending on its role, its input and output differ.

1.1. Processing Blocks

A processing block is a logical unit that performs some action on a packet. Each processing block has its configuration parameters that customize its behavior, as well as sets of read and write handlers, which allow querying and setting properties from and to the block. Each block may have zero or more outputs. The configuration defines which output will be used for each packet.

1.2. Processing Graph

Connecting the output of one block to the input of another block creates a *processing graph*. This graph describes the flow of packets in the OBI.

1.3. OBI Local Storage

1.3.1. Metadata Storage

Metadata is a short-lived ***virtual*** key-value store for passing information between different processing blocks and between different OBIs.

It is important to note the virtuality of the metadata storage: it may sometimes be omitted in whole or in part by specific OBI implementations, although it should still be possible to use it logically in applications and directives from controller. For example, in software, it may not be efficient to store header field values in the metadata store as these are already available in the packet header struct. Thus, such implementation can ignore a directive to store a header field in metadata storage and instead, when encountering a directive to load a value that was stored using the former directive, load the value directly from the available packet header struct.

1.3.2. Session Map Store

For each session, the Session Map Store is a key-value store that keeps information across multiple packets of the same session. This object is disposed by OBI either after a FIN message is found or if the session times out. Specific processing blocks are defined to store state information or retrieve it.

2. OpenBox Controller

The OpenBox Controller (OBC) is a server that controls all OBIs in the network, or a set of OBIs in the network. The OBC has an API for programmers to develop OpenBox applications on top of it (Northbound API). The OBC is responsible to aggregate the applications programmed using this API to messages sent to OBIs, such that ultimately, the OBIs will provide the behavior as required by the OpenBox applications.

Communication between controller and OBIs is done using an HTTP REST server. As communication is asynchronous and can be initiated by both parties, both parties run their own REST server to handle incoming messages.

3. The OpenBox Protocol

The OBC communicates with OBIs using the OpenBox protocol via a bidirectional REST interface. This section lists the required messages of the protocol.

Name	**Purpose**	**Direction**	**Synchronous**
Hello	Sent from OBI to OBC to initialize communication. This message includes parameters that tell the capabilities of the sender OBI.	Up	No
KeepAlive	Keep alive message (OBI -> OBC). If not received by OBC for more than a specified duration, OBI is considered to be down.	Up	No
ListCapabilitiesRequest	OBC requests OBI to tell it the capabilities of OBI in terms of optional and vendor defined functions	Down	No
ListCapabilitiesResponse	OBI response for ListCapabilitiesRequest	Up	No
SetParametersRequest	OBC sets parameters of OBI, such as keepalive interval, log and storage servers address, and more	Down	No
SetParametersResponse	OBI acknowledge the successful	Up	No

	execution of a SetParametersRequest		
GetParametersRequest	OBC requests OBI a listing of its parameters	Down	No
GetParametersResponse	OBI response for GetParametersRequest	Up	No
GlobalStatsRequest	OBC requests OBI for some global statistics	Down	No
GlobalStatsResponse	OBI reply for a GlobalStatRequest	Up	No
ReadRequest	Send a request to read one or more read handles from one or more blocks	Down	No
ReadResponse	Result of a read request	Up	No
WriteRequest	Send a request to write one or more write handles in one or more blocks	Down	No
WriteResponse	Acknowledgement of a successful write request	Up	No
SetProcessingGraphRequest	OBC sends the complete processing graph to OBI. This include the configuration for all blocks used, and the connections between them	Down	No
SetProcessingGraphResponse	OBI acknowledge the successful execution of a SetProcessingGraphRequest	Up	No
Alert	Send alert from OBI to OBC	Up	No
BarrierRequest	OBC requests OBI to complete all previous requests up until the receiving of this message before processing next message from OBC. (If OBI processes messages completely sequentially, it should simply ignore this message)	Down	No
Error	Error message (OBC to OBI or OBI to OBC)	Up/Down	No

The following messages are optional:

Name	Purpose	Direction	Synchronous
AddCustomModuleRequest	Allows installing a new module in the data plane by sending it from the controller. The controller should send the module in the format expected by data plane (e.g. binary file encoded with base64), and it can optionally also send textual translation code, that will allow the data plane to translate the configuration of new processing blocks defined by the new module.	Down	No
AddCustomModuleResponse	OBI acknowledge the successful execution of a AddCustomModuleRequest	Up	No
RemoveCustomModuleRequest	Remove a custom module that was previously added to data plane	Down	No
RemoveCustomModuleResponse	OBI acknowledge the successful execution of a RemoveCustomModuleRequest	Up	No

3.1. REST API

A REST server should be running at OBC and OBI. The default TCP port for OpenBox REST servers (on controller and on OBIs) is 3636 (This may change in future). Messages are pushed to the REST server of the other side using POST method, where the URL is the /message/ directory followed by the name of the pushed message.

For example, the Hello message will be pushed from OBI to OBC by sending a REST request as follows:
POST /message/Hello
With content type "application/json", where the message object is sent as the payload of the request. The object should include the "type" field with its value equals to the name of the message.

The connection may use standard HTTP compression and/or encryption, if supported and configured by both sides.

3.2. Transactions

Each message sent from OBC to OBI is a *transaction*, in the sense that it is expected to succeed in a whole, or, in case of a failure, rollback to the state that was in the OBI before the message began processing.

Each such message includes a unique (numeric) *xid* argument. This number identifies the transaction. When an asynchronous message is sent, it is expected that the receiving side acknowledge the receiving of a message with a 200 OK response, if the message is valid. It may delay the execution of the message, for example due to some transient load. Once the message is executed/processed, the result (if such exists) is sent back to the other side as a new message with the original *xid* value on the other REST channel. The receiver of the result should return a 200 OK response if the result message is valid. In case of an error during the execution of a message, an Error message with the original *xid* value should be sent to the other side.

3.3. KeepAlive

Each OBI is expected to send KeepAlive messages every defined interval. If not received by controller, it can infer that the OBI has gone down. Controller may set the interval for each OBI using the SetParametersRequest message. However, upon startup of OBI, it should immediately start sending KeepAlive messages. The default KeepAlive interval time is 10 seconds.
The behavior for the case when OBC does not respond to KeepAlive messages, or when it returns some error code, is left for OBI implementor to decide and thus is not defined in this document. It is suggested that in any case of an unresponsive controller, after a few retries, OBI will consider the controller to be down and restart initial connection attempts by sending Hello messages from time to time, until OBC comes up.

3.4. Error Messages

Each error message has an error type and subtype. Several types and subtypes are defined in the protocol. Additional types and subtypes can be defined by OBI or OBC vendors, but these should be supported in both sides of the channel in any case.

Each error message may also have a textual error message and some extended message (such as stack trace or other debug information).

Error types:

```
BAD_REQUEST         // The message cannot be understood
FORBIDDEN           // The request is forbidden
UNSUPPORTED         // The message is not supported by the receiver
INTERNAL_ERROR      // There was an internal error
```

3.4.1. Error subtypes:

```
BAD_REQUEST:
        BAD_VERSION             // Wrong or non-matching OpenBox version
        BAD_TYPE                // Bad "type" value in message
        BAD_GRAPH               // Bad processing graph was given
        BAD_BLOCK               // Bad processing block was given
        BAD_BLOCK_IMPL          // Bad processing block implementation was given
        BAD_CONNECTOR           // Bad processing graph connector was given
        BAD_HEADER_MATCH        // Bad header match map was given
        BAD_PAYLOAD_MATCH       // Bad payload match list was given
        BAD_FILE                // Bad file was given for AddCustomModule
        ILLEGAL_ARGUMENT        // Illegal argument was given (general)
        ILLEGAL_STATE           // The system was in a state where it is illegal to execute
                                // the given message

FORBIDDEN:
        NOT_PERMITTED           // Operation is not permitted
        NO_ACCESS               // Operation cannot be executed as there is no access to a
                                // required resource

UNSUPPORTED:
        UNSUPPORTED_VERSION     // OpenBox version is not supported
        UNSUPPORTED_BLOCK       // Stage is not supported
        UNSUPPORTED_MESSAGE     // Message is not supported
        UNSUPPORTED_OTHER       // Something else is not supported (details in textual
                                // message)

INTERNAL_ERROR:
        ADD_MODULE_FAILED       // The process of adding a custom module failed
        INTERNAL_ERROR          // There was an internal error. Details may appear
                                // in textual message.
```

3.5. Connection Setup Process

#	Direction	Message	Description
1	Up	Hello	OBI tells OBC its existence and details such as ID, location, capabilities, etc. OBC returns 200 OK, or some error code followed by an out-of-bound error message.
2	Down	SetParametersRequest	Optional: Set OBI parameters
3	Down	AddCustomModuleRequest	Optional: Install custom modules
4	Down	BarrierRequest	Wait for module installation to complete
5	Down	SetProcessingGraphRequest	Set processing graph for this OBI
6	Down	BarrierRequest	OBC tells OBI to hold the execution of any subsequent until the execution of SetProcessingGraph is completed.

3.6. Processing Blocks

There are five class of processing blocks: Terminals, Classifiers, Modifiers, Shapers and Statics. These classes are used by OBC to perform processing graph merge and possibly other operations.

3.6.1. Required blocks

The following blocks are required to exist in a fully functional OBI implementation. An OBI that only provides partial functionality should only implement the blocks in italics.

Block Name	Class	Description
FromDevice	*T*	*Read packets from an interface*
FromDump	T	Read packets from a dump
ToDevice	*T*	*Write packets to an interface*
ToDump	T	Write packets to a dump
Discard	*T*	*Drop packets*
HeaderClassifier	C	Classify on header fields
RegexClassifier	C	Classify using a regex match

RegexMatcher	C	Matches a packet against a set of regexes
HeaderPayloadClassifier	C	Classify on both header fields and payload regex matches
VlanEncapsulate	M	Push a VLAN tag
VlanDecapsulate	M	Pop a VLAN tag
DecIpTtl	M	Decrement IP TTL value by 1
Ipv4AddressTranslator	M	Network address translator
NetworkHeaderFieldsRewriter	M	Rewrite header fields
NetworkDirectionSwap	M	Swap network directions
Alert	St	Send alert to OBC
Log	St	Log packet
Queue	Sh	Queue packets (FIFO)
ProtocolAnalyzer	C	Classify on protocol type
SessionClassifier	C	Classify on session information
SessionStore	S	Store session information
FlowTracker	M	Mark flow
GzipDecompressor	M	Decompress HTTP Gzip packet
EncapsulateMetadata	*M*	*Encapsulate packet with metadata*
WriteMetadata	*M*	*Write value to packet's metadata*
ReadMetadata	*C*	*Classify on metadata*
DecapsulateMetadata	*M*	*Decapsulate metadata from packet to metadata store*

3.6.2. Optional blocks

The following blocks are defined by the protocol and can be implemented by OBIs.

Block Name	Type	Description
ContentClassifier	C	
FrontDropQueue	M	
RandomEarlyDetectionQueue	M	
BpsShaper	M	
PpsShaper	M	
BpsRateClassifier	C	
PpsRateClassifier	C	
Store	S	
Restore	M	
BeginTransaction	M	
CommitTransaction	M	
RollbackTransaction	M	
HtmlNormalizer	M	
JsNormalizer	M	
GzipCompressor	M	

4. Appendix 1 - OpenBox Protocol Message Structure

4.1. Header Match Fields

(for HeaderClassifier and HeaderPayloadClassifier implementations)

4.1.1. Required match fields:

ETH_DST	*IP_PROTO*	TCP_SRC
ETH_SRC	IPV4_SRC	TCP_DST
ETH_TYPE	IPV4_DST	UDP_SRC
VLAN_VID		UDP_DST
VLAN_PCP		

4.1.2. Optional match fields:

ARP_OP	IP_DSCP	SCTP_SRC
ARP_SPA	IP_ECN	SCTP_DST
ARP_TPA	IPV6_SRC	ICMPV4_TYPE
ARP_SHA	IPV6_DST	ICMPV4_CODE
ARP_THA	IPV6_FLABEL	ICMPV6_TYPE
MPLS_LABEL	IPV6_ND_TARGET	ICMPV6_CODE
MPLS_TC	IPV6_ND_SLL	
MPLS_BOS	IPV6_ND_TLL	
PBB_ISID	IPV6_EXTHDR	
TUNNEL_ID		
PBB_UCA		

See pages 77-78 in OpenFlow 1.5 spec for description of these fields and their prerequisites.

4.1.3. Custom Match Fields

Vendors may add additional match fields. OBI implementation should list optional and custom match fields in the Hello / ListCapabilitiesResponse messages. OBC must be aware of these fields in order to check they are supported and use them.

4.2. Protocol Messages

4.2.1. Hello

type: "Hello"
xid: Number
dpid: Number (dpid = Data Path ID - a unique identifier of the sender)
version: String (supported OpenBox version)
capabilities: Map<String: String>
Protocol optional capabilities that can be declared here:

- Optional protocol messages:
  "proto_messages": [list of message names]
- Processing blocks
  "processing_blocks": Map<String, List<String>>
  Each map entry maps an *abstract block name* to a list of supported implementations, ordered from the most recommended implementation to the least recommended one, in terms of processing performance
- Match fields (from Section 4.1.2. and 4.1.3.):
  "match_fields": [list of optional/custom match fields]
- Complex match (See HeaderClassifier block for details)
  "complex_match": list with only one of the following Strings:
  - "full" - supports all types of complex match values
  - "lists" - supports lists of values
  - "ranges" - supports ranges of any types
  - “basic” - supports only a single
- Protocol analyzer additional protocols (See ProtocolAnalyzer block)
  "protocol_analyser_protocols": [list of protocol names]

4.2.2. KeepAlive

type: "KeepAlive"
xid: Number
dpid: Number

4.2.3. Global Statistics

Request:
type: "GlobalStatsRequest"
xid: Number

Response:
type: "GlobalStatsResponse"
xid: Number
stats: Map<String, Value>
Keys can be:
current_load: Number in [0,1]

avg_load: Number in [0,1]
avg_minutes: Number (of minutes to calculate avg_load)
uptime: Number (of seconds since last boot)
cpus: Number (of logical cores used by OBI)
memory_rss: Number (of bytes)
memory_vms: Number (of bytes)
memory_usage: Number (percentage 0-100)

Reset:

type: "GlobalStatsReset"
xid: Number

4.2.4. Read/Write Requests

Read request:

type: "ReadRequest"
xid: Number
block_id: String
read_handle: String

Read response:

type: "ReadResponse"
xid: Number
block_id: String
read_handle: String
result: Value

Write request:

type: "WriteRequest"
xid: Number
block_id: String
write_handle: String
value: Value

Write response:

type: "WriteResponse"
xid: Number
block_id: String
read_handle: String

4.2.5. Processing Graph

Set processing graph(s):

type: "SetProcessingGraphRequest"
xid: Number

required_modules: List of required modules (should be installed first).
blocks: List of Block objects
connectors: List of Connector objects
report_back: Boolean (Default: false)

Block object:
type: String
name: String
config: Map<String, Value>

Connector object:
src: String (Block ID)
src_port: Number (Output port number)
dst: String (Block ID)
dst_port: Number

Acknowledgement:
type: "SetProcessingGraphResponse"
xid: Number
blocks: List of Block objects (optional, if report_back in the request is true)
connectors: List of Connector objects (optional, if report_back in the request is true)

4.2.6. Storage and Log Servers

Get storage server:
type: "GetStorageServer"

Get log server:
type: "GetLogReaquest"

4.2.7. Barrier Request

type: "BarrierRequest"
xid: Number

4.2.8. Error

type: "Error"
xid: Number
error_type: String
error_subtype: String
message: String
extended_message: String

Note: If the error was generated due to a specific request then the XID should be the request's XID. Otherwise, it should be set to another unique value.

4.2.9. Alert

type: "Alert"
xid: Number
origin_dpid: Number (DPID of sender OBI)
messages: List of alerts with the following structure:
    id: Number (a unique number for this alert, can be sequential)
    timestamp: Number
    message: String
    severity: Number
    packet: String (blob)
    origin_block: String (identifier of the block that originated the alert)
    app_id: String (identifier of the app that requested this alert)

4.2.10. Log

type: "Log"
xid: Number
origin_dpid: Number (DPID of sender OBI)
messages: List of log messages with the following structure:
    id: Number (a unique number for this log message, can be sequential)
    timestamp: Number
    message: String
    severity: Number
    packet: String (blob)
    origin_block: String (identifier of the block that originated the log message)

4.2.11. AddCustomModuleRequest

type: "AddCustomModule"
xid: Number
module_name: String
module_content: String
content_type: String ("application/octet-stream")
content_transfer_encoding: String ("base64")
translation:
    execution engine specific object that represents the translation from protocol block notation to execution engine configuration

4.2.12. RemoveCustomModuleRequest

type: "RemoveCustomModule"
xid: Number
module_name: String

4.2.13. ListCapabilitiesRequest

type: "ListCapabilitiesRequest"

xid: Number

4.2.14. ListCapabilitiesResponse

type: "ListCapabilitiesResponse"
xid: Number
capabilities: Map<String, String> (as defined in the Hello message)

4.2.15. SetParametersRequest

type: "SetParametersRequest"
xid: Number
parameters: Map<String, Value>
Possible keys:
keepalive_interval: Number (of seconds)
log_server_address: String (IP Address)
log_server_port: Number
storage_server_address: String (IP Address)
storage_server_port: Number
log_messages_buffer_size: Number (of messages)
log_messages_buffer_timeout: Number (in milliseconds)
alert_messages_buffer_size: Number (of messages)
alert_messages_buffer_timeout: Number (in milliseconds)

4.2.16. SetParametersResponse

type: "SetParametersResponse"
xid: Number

4.2.17. GetParametersRequest

type: "GetParametersRequest"
xid: Number
parameters: List of strings (empty means all parameters)

4.2.18. GetParametersResponse

type: "GetParametersResponse"
xid: Number
parameters: Map<String, Value>

5. Appendix 2 - OpenBox Processing Blocks

Each processing block will be defined by a JSON with following fields:
name(string, required), description(string, optional), config(list(Arg), required), read_handles(list(Handle), required), write_handles(list(Handle), required).
The block uses the Arg and Handle JSON object to represent a configuration argument and a read/write handle, respectively.
The Arg JSON has the following fields:
name(string, required), type(string, required), description(string, optional), required(boolean, optional).
The Handle JSON has the following fields:
name(string, required), type(string, required), description(string, optional).

5.1. FromDevice

Reads packets from network device

5.1.1. Configuration

Name	Required	Type	Description
devname	true	string	The name of the network interface
sniffer	false	bool	Specifies whether FromDevice should run in sniffer mode. In non-sniffer mode. Default is true (sniffer mode).
promisc	false	bool	FromDevice puts the device in promiscuous mode if PROMISC is true. The default is false.

5.1.2. Read Handles

Name	Type	Description
count	integer	Returns the number of packets that have passed through since the last reset.
byte_count	integer	Returns the number of bytes that have passed through since the last reset.
rate	number	Returns the recent arrival rate, measured by exponential weighted moving average, in packets per second.
byte_rate	number	Returns the recent arrival rate, measured by exponential weighted moving average, in bytes per second.

drops	string	Returns the number of packets dropped by the kernel, probably due to memory constraints, before FromDevice could get them. This may be an integer; the notation "<*d*", meaning at most *d* drops; or "??", meaning the number of drops is not known.

5.1.3. Write Handles

Name	Type	Description
reset_counts	null	Reset all counts and rates to zero.

5.2. FromDump

Reads packets from a PCAP file.

5.2.1. Configuration

Name	Required	Type	Description
filename	true	string	The name of the PCAP file.
timing	false	bool	If true, then FromDump tries to maintain the timing of the original packet stream. The first packet is emitted immediately; thereafter, FromDump maintains the delays between packets. Default is false.
active	false	bool	If false, then FromDump will not emit packets (until the `active' handler is written). Default is true.

5.2.2. Read Handles

Name	Type	Description
count	integer	Returns the number of packets that have passed through since the last reset.
byte_count	integer	Returns the number of bytes that have passed through since the last reset.
rate	number	Returns the recent arrival rate, measured by exponential weighted moving average, in packets per second.
byte_rate	number	Returns the recent arrival rate, measured by exponential weighted moving average, in bytes per second.

5.2.3. Write Handles

Name	Type	Description
reset_counts	null	Reset all counts and rates to zero.
active	bool	Set the state of ‘active’ to true or false

5.3. ToDevice

Sends packets to network device.

5.3.1. Configuration

Name	Required	Type	Description
devname	true	string	The name of the network interface

5.3.2. Read Handles

Name	Type	Description

5.3.3. Write Handles

Name	Type	Description

5.4. ToDump

Write packets to a PCAP file.

5.4.1. Configuration

Name	Required	Type	Description
filename	true	string	The name of the PCAP file.

5.4.2. Read Handles

Name	Type	Description

5.4.3. Write Handles

Name	Type	Description

5.5. Discard

Drops a packet. This block has no output ports.

5.5.1. Configuration

Name	Required	Type	Description

5.5.2. Read Handles

Name	Type	Description
count	number	Number of packets dropped by this block

5.5.3. Write Handles

Name	Type	Description
reset_counts	null	Reset counter

5.6. ContentClassifier

Classifies packets by contents. The Classifier has N outputs, each associated with the corresponding pattern.

5.6.1. Configuration

Name	Required	Type	Description
pattern	true	array(string)	A pattern is a set of clauses, where each clause is either "offset/value" or "offset/value%mask". A pattern matches if the packet has the indicated value at each offset. The clauses in each pattern are separated by spaces. A clause consists of the offset, "/", the value, and (optionally) "%" and a mask. The offset is in decimal. The value and mask are in hex. The length of the value is implied by the number of hex digits, which must be even. "?" is also allowed as a "hex digit"; it means "don't care about the value of this nibble". If present, the mask must have the same number of hex digits as the value. The matcher will only check bits that are 1 in the mask. A clause may be preceded by "!", in which case the clause must NOT match the packet. As a special case, a pattern consisting of "-" matches every packet. The patterns are scanned in order, and the packet is sent to the output corresponding to the first matching pattern. Thus more specific patterns should come before less specific ones. You will get a warning if no packet could ever match a pattern. Usually, this is because an earlier pattern is more general, or because your pattern is contradictory (`12/0806 12/0800').

5.6.2. Read Handles

Name	Type	Description
count	array<Integer>	Returns the number of packets matched each pattern.
byte_count	array<Integer>	Returns the number of bytes matched each pattern.
rate	array<Number>	Returns the recent matching rate for each pattern, measured by exponential weighted moving average, in packets per second.
byte_rate	array<Number>	Returns the recent matching rate for each patterni, measured by exponential weighted moving average, in bytes per second.

5.6.3. Write Handles

Name	Type	Description
reset_counts	null	Reset all counts and rates to zero.

5.7. HeaderClassifier

Classifies packets by header fields. The Classifier has N outputs, each associated with the corresponding pattern. If the packet does not match any rule it will be discarded.

5.7.1. Configuration

Name	Required	Type	Description
match	true	Array(map<string, value>)	A match is a mapping between field name and the expected value in this field. Value must match the type of the field (e.g., TCP port only accepts integers in [0, 64K) and IPv4 address expects a string with the structure of an IPv4 address). IP address can be (optionally) be succeeded by a '%' and then a mask value of the exact same type (e.g., "ipv4_src" mapped to "10.0.0.1%255.0.0.0"). An integer value can be (optionally) masked by adding a '%' followed by a mask (e.g., tcp_src mapped to 80%0x00ff will be interpreted as tcp_src & 0x00ff == 80). Complex Values If the OBI and the field support complex values (this

			should be declared by OBI in HELLO message), value can be a string wrapped in brackets ([,]), this means either a range or several distinct possible values: - "[a:b]" - all integers from a to b - "[a:]" - all integers greater than or equal to a - "[:b]" - all integers less than or equal to b - "[value1,value2,...]" - either one of the values Note that the last line implies a recursive definition, i.e., the following value is also possible: "[80,8080,[16384:]]". Prerequisites When using match fields one must enforce correct packet structure (e.g., if matching on "ipv4_src", there should be a preceding mapping between "eth_type" and "0x800"). A match that does not validate prerequisites may be rejected.

5.7.2. Read Handles

Name	Type	Description
count	array<Integer>	Returns the number of packets matched each pattern.
byte_count	array<Integer>	Returns the number of bytes matched each pattern.
rate	array<Number>	Returns the recent matching rate for each pattern, measured by exponential weighted moving average, in packets per second.
byte_rate	array<Number>	Returns the recent matching rate for each patterni, measured by exponential weighted moving average, in bytes per second.

5.7.3. Write Handles

Name	Type	Description
reset_counts	null	Reset all counts and rates to zero.

5.8. RegexMatcher

Matches packet's header and/or payload against a set of regex patterns. The block has 2 outputs:

Output 0 - matched packets.

Output 1 - Unmatched packets.

5.8.1. Configuration

Name	Required	Type	Description
pattern	true	array(string)	A list of pattern to match against. Each pattern must be legal regex (with no backtracking).
payload_only	false	bool	If true, the match will be only on the payload part of the packet. The payload is determent by the set of network layers. Default is false.
match_all	false	bool	If true, a packet is considered a match only if it matches all of the patterns. If false, a packet is considered a match if it matches any of the patterns. Default is false.

5.8.2. Read Handles

Name	Type	Description
count	Array<Integer>	Returns a list with 2 elements: the first is the number of packets matched and the second is the number packets unmatched.
byte_count	Array<Integer>	Returns a list with 2 elements: the first is the number of packets bytes matched and the second is the number packets bytes unmatched.
rate	Array<Number>	Returns a list with 2 elements: the first is the rate of packets matched and the second is the rate of packets unmatched., measured by exponential weighted moving average, in packets per second
byte_rate	number	Returns a list with 2 elements: the first is the rate of packets matched and the second is the rate of packets unmatched., measured by exponential weighted moving average, in bytes per second
paylaod_only	bool	Read the ‘payload_only’ value.
match_all	bool	Read the ‘match_all’ value.

5.8.3. Write Handles

Name	Type	Description
payload_only	bool	Set the ‘payload_only’ value.

match_all	bool	Set the ‘match_all’ value.
reset_counts	null	Reset all counts and rates to zero.

5.9. RegexClassifier

Classify a packet using a regex match on its content. The Classifier has N outputs, each associated with the corresponding pattern. If a packet doesn’t match any rule it will be discarded.

5.9.1. Configuration

Name	Required	Type	Description
pattern	true	array(string)	A list of pattern to match against. Each pattern must be legal regex (with no backtracking).
payload_only	false	bool	If true, the match will be only on the payload part of the packet. The payload is determent by the set of network layers. Default is false.

5.9.2. Read Handles

Name	Type	Description
count_i	number	Returns the number of packets matched pattern_i
byte_count_i	number	Returns the number of bytes matched pattern_i
rate_i	number	Returns the matching rate for pattern_i, measured by exponential weighted moving average, in packets per second.
byte_rate_i	number	Returns the matching rate for pattern_i, measured by exponential weighted moving average, in bytes per second.
paylaod_only	bool	Read the ‘payload_only’ value.

5.9.3. Write Handles

Name	Type	Description
payload_only	bool	Set the ‘payload_only’ value.
match_all	bool	Set the ‘match_all’ value.
reset_counts	null	Reset all counts and rates to zero.

5.10. VlanEncapsulate

Expects Ethernet-encapsulated packets as input. Adds an 802.1Q shim header. The resulting packet looks like an Ethernet packet with type 0x8100.

5.10.1. Configuration

Name	Required	Type	Description
vlan_vid	true	Integer	A 12 bit VLAN VID.
vlan_dei	false	Integer	VLAN DEI bit (default is 0)
vlan_pcp	false	Integer	VLAN Priority Code Point, a number between 0 and 7. Defaults to 0.
ethertype	false	Integer	Specifies the ethertype designating VLAN encapsulated packets. The default is 0x8100 (standard 802.1Q customer VLANs); other useful values are 0x88a8 (for 802.1ad service VLANs, aka QinQ) and 0x9100 (old non-standard VLANs).

5.10.2. Read Handles

Name	Type	Description
vlan_vid	Integer	Return the VLAN VID value
vlan_pcp	Integer	Return the VLAN PCP value
vlan_tci	Integer	Return VLAN TCI value
ethertype	Integer	Return the ethertype value

5.10.3. Write Handles

Name	Type	Description
vlan_vid	Integer	Set the VLAN VID value
vlan_pcp	Integer	Set the VLAN PCP value
vlan_tci	Integer	Set VLAN TCI value
ethertype	Integer	Set the ethertype value

5.11. VlanDecapsulate

Expects a potentially 802.1Q VLAN encapsulated packet as input. If it is encapsulated, then the encapsulation is stripped, leaving a conventional Ethernet packet.

5.11.1. Configuration

Name	Required	Type	Description

5.11.2. Read Handles

Name	Type	Description

5.11.3. Write Handles

Name	Type	Description

5.12. DecIpTtl

Expects IP packet as input. If the ttl is <= 1 (i.e. has expired), DecIpTtl sends the packet to output 1. Otherwise it decrements the ttl, re-calculates the checksum, and sends the packet to output 0.

5.12.1. Configuration

Name	Required	Type	Description
active	False	bool	If false do not decrement any packet's TTL. Default to true.

5.12.2. Read Handles

Name	Type	Description
count	Array<Integer>	Returns a list with 2 elements: the first is the number of packets not dropped and the second is the number of dropped packets.
byte_count	Array<Integer>	Returns a list with 2 elements: the first is the number of packets bytes not dropped and the second is the number of dropped packets bytes.
rate	Array<Number>	Returns a list with 2 elements: the first is the rate of packets not dropped and the second is the rate of dropped packets, measured by exponential weighted moving average, in packets per second
byte_rate	number	Returns a list with 2 elements: the first is the rate of packets not dropped and the second is the rate of dropped packets, measured by exponential weighted moving average, in bytes per second
active	number	Returns the value of 'active'

5.12.3. Write Handles

Name	Type	Description
reset_counts	null	Reset all counts and rates
active	bool	Switch the state of active.

5.13. Ipv4AddressTranslator

Rewrites the source address, source port, destination address, and/or destination port on TCP and UDP packets, along with their checksums. Ipv4AddressTranslator implements the functionality of a network address/port translator (NAPT). Ipv4AddressTranslator maintains a mapping table that records how packets are rewritten. The mapping table is indexed by flow identifier, the quintuple of source address, source port, destination address, destination port, and IP protocol (TCP or UDP). Each mapping contains a new flow identifier and an output port. Input packets with the indexed flow identifier are rewritten to use the new flow identifier, then emitted on the output port. A mapping is written as follows:
(SA, SP, DA, DP, PROTO) => (SA', SP', DA', DP') [*OUTPUT]
When Ipv4AddressTranslator receives a packet, it first looks up that packet in the mapping table by flow identifier. If the table contains a mapping for the input packet, then the packet is rewritten according to the mapping and emitted on the specified output port. If there was no mapping, the packet is handled by the INPUTSPEC corresponding to the input port on which the packet arrived. (There are as many input ports as INPUTSPECs.) Most INPUTSPECs install new mappings, so that future packets from the same TCP or UDP flow are handled by the mapping table rather than some INPUTSPEC.

5.13.1. Configuration

Name	Required	Type	Description
input_spec	true	array(string)	The 4 forms of INPUTSPEC handle input packets as follows: - 'discard': Discard input packets. - 'pass OUTPUT': Sends input packets to output port OUTPUT. No mappings are installed. - 'keep FOUTPUT ROUTPUT': Installs mappings that preserve the input packet's flow ID. Specifically, given an input packet with flow ID (SA, SP, DA, DP, PROTO), two mappings are installed: (SA, SP, DA, DP, PROTO) => (SA, SP, DA, DP) [*FOUTPUT] and (DA, DP, SA, SP, PROTO) => (DA, DP, SA, SP) [*ROUTPUT]. Thus, the input packet is emitted on output port FOUTPUT unchanged, and packets from the reply flow

			are emitted on output port ROUTPUT unchanged. - ‘pattern SADDR SPORT DADDR DPORT FOUTPUT ROUTPUT’: Creates a mapping according to the given pattern, 'SADDR SPORT DADDR DPORT'. Any pattern field may be a dash '-', in which case the packet's corresponding field is left unchanged. For instance, the pattern '1.0.0.1 20 - -' will rewrite input packets' source address and port, but leave its destination address and port unchanged. SPORT may be a port range 'L-H'; IPRewriter will choose a source port in that range so that the resulting mappings don't conflict with any existing mappings. The input packet's source port is preferred, if it is available; otherwise a random port is chosen. If no source port is available, the packet is dropped. To allocate source ports sequentially (which can make testing easier), append a pound sign to the range, as in '1024-65535#'. To choose a random port rather than preferring the source, append a '?'. Say a packet with flow ID (SA, SP, DA, DP, PROTO) is received, and the corresponding new flow ID is (SA', SP', DA', DP'). Then two mappings are installed: (SA, SP, DA, DP, PROTO) => (SA', SP', DA', DP') [*FOUTPUT] (DA', DP', SA', SP', PROTO) => (DA, DP, SA, SP) [*ROUTPUT] Thus, the input packet is rewritten and sent to FOUTPUT, and packets from the reply flow are rewritten to look like part of the original flow and sent to ROUTPUT.
tcp_timeout	false	Integer	Time out TCP connections every *time* seconds. Defaults to 24 hours. This timeout applies to TCP connections for which payload data has been seen flowing in both directions.
tcp_done_timeout	false	Integer	Time out completed TCP connections every *time* seconds. Defaults to 4 minutes. FIN and RST flags mark TCP connections as complete.
tcp_nodata_timeout	false	Integer	Time out non-bidirectional TCP connections every *time* seconds. Defaults to 5 minutes. A

			non-bidirectional TCP connection is one in which payload data hasn't been seen in at least one direction. This should generally be larger than TCP_DONE_TIMEOUT.
tcp_guarantee	false	Integer	Preserve each TCP connection mapping for at least *time* seconds after each successfully processed packet. Defaults to 5 seconds. Incoming flows are dropped if the mapping table is full of guaranteed flows.
udp_timeout	false	Integer	Timeout UDP connections every *time* seconds. Default is 5 minutes.
udp_streaming_timeout	false	Integer	Timeout UDP streaming connections every *time* seconds. A "streaming" connection, in contrast to an "RPC-like" connection, comprises at least 3 packets and at least one packet in each direction. Default is the UDP_TIMEOUT setting.
udp_guarantee	false	Integer	UDP connection mappings are guaranteed to exist for *time* seconds after each successfully processed packet. Defaults to 5 seconds.
reap_interval	false	Integer	Reap timed-out connections every *time* seconds. Default is 15 minutes.
mapping_capacity	false	Integer	The maximum number of mappings this rewriter can hold.

5.13.2. Read Handles

Name	**Type**	**Description**
mappings_count	number	Returns the number of mappings in the mapping table.
mapping_failures	number	Returns the number of mapping failures, which can occur, for example, when the Rewriter runs out of source ports, or when a new flow is dropped because the Rewriter is full.
length	number	Returns the number of flows in the flow set. This is generally the same as mappings_count, but can be more when several rewriters share a flow set.
capacity	string	Return the capacity of the flow set. The returned value is two space-separated numbers, where the first is the capacity and the second is the short-term flow

		reservation.
tcp_mappings	string	Returns a human-readable description of the Rewriter's current set of TCP mappings. An unparsed mapping includes both directions' output ports; the relevant output port is starred.
udp_mappings	string	Returns a human-readable description of the Rewriter's current set of UDP mappings.

5.13.3. Write Handles

Name	Type	Description
capacity	number	Set the capacity of the flow set.

5.14. NetworkHeaderFieldsRewriter

Rewrite fields in the network headers. It does not recalculates the checksums.

5.14.1. Configuration

Name	Required	Type	Description
eth_dst	false	MacAddress	Ethernet MAC address
eth_src	false	MacAddress	Ethernet MAC address
eth_type	false	Integer	Ethernet Type address (or VLAN next protocol)
ipv4_proto	false	Integer	IPV4 next protocol
ipv4_src	false	IPv4Address	IPv4 source address
ipv4_dst	false	IPv4Address	IPv4 destination address
ipv4_dscp	false	integer	IPv4 DSCP value.
ipv4_ecn	false	integer	IPv4 ECN value
ipv4_ttl	false	integer	IPv4 TTL
tcp_src	false	integer	TCP source port
tcp_dst	false	integer	TCP destination port
udp_src	false	integer	UDP source port
udp_dst	false	integer	UDP destination port

5.14.2. Read Handles

Name	Type	Description
eth_dst	false	Ethernet MAC address
eth_src	false	Ethernet MAC address
eth_type	false	Ethernet Type address (or VLAN next protocol)
ipv4_proto	false	IPV4 next protocol
ipv4_src	false	IPv4 source address
ipv4_dst	false	IPv4 destination address
ipv4_dscp	false	IPv4 DSCP value.
ipv4_ecn	false	IPv4 ECN value
ipv4_ttl	false	IPv4 TTL
tcp_src	false	TCP source port
tcp_dst	false	TCP destination port
udp_src	false	UDP source port
udp_dst	false	UDP destination port

5.14.3. Write Handles

Name	Type	Description
eth_dst	false	Ethernet MAC address
eth_src	false	Ethernet MAC address
eth_type	false	Ethernet Type address (or VLAN next protocol)
ipv4_proto	false	IPV4 next protocol
ipv4_src	false	IPv4 source address
ipv4_dst	false	IPv4 destination address
ipv4_dscp	false	IPv4 DSCP value.
ipv4_ecn	false	IPv4 ECN value
ipv4_ttl	false	IPv4 TTL

tcp_src	false	TCP source port
tcp_dst	false	TCP destination port
udp_src	false	UDP source port
udp_dst	false	UDP destination port

5.15. NetworkDirectionSwap

Swaps the direction of the requested network layers.

5.15.1. Configuration

Name	Required	Type	Description
ethernet	False	Boolean	If set to true the direction of the ethernet layer will be swapped.
ipv4	False	boolean	If set to true the IPv4 source and dest address will be swapped.
ipv6	False	boolean	If set to true the IPv6 source and dest address will be swapped.
tcp	False	boolean	If set to true the TCP source port and dest port will be swapped.
udp	False	boolean	If set to true the UDP source port and dest port will be swapped.

5.15.2. Read Handles

Name	Type	Description
ethernet	boolean	Return the ‘ethernet’ value
ipv4	boolean	Return the ‘IPv4’ value
ipv6	boolean	Return the ‘IPv6’ value
tcp	boolean	Return the ‘tcp’ value
udp	boolean	Return the ‘udp’ value

5.15.3. Write Handles

Name	Type	Description
ethernet	boolean	Set the ‘ethernet’ value

ipv4	boolean	Set the ‘IPv4’ value
ipv6	boolean	Set the ‘IPv6’ value
tcp	boolean	Set the ‘tcp’ value
udp	boolean	Set the ‘udp’ value

5.16. Alert

Sends an alert to OBC. The packet is passed through.
Alert object:
{
    message: String (a message string to send with the alert)
    severity: Number (alert message severity (1 - lowest, 5 - maximal). Default: 1)
    app_id: String (a string that represents the application that requested this alert)
}

5.16.1. Configuration

Name	Required	Type	Description
alerts	true	Array(Alert)	An array of Alert objects as defined above.
attach_packet	false	bool	Should the packet that caused the log action be attached to the log message. (default: false)
packet_size	false	number	If attach_packet is true, how many bytes from the packet to attach. negative value means all the packet, (Default: 64)

5.16.2. Read Handles

Name	Type	Description

5.16.3. Write Handles

Name	Type	Description

See alert message structure in Appendix 1.

5.17. Log

Sends a log message to the Log Server. The packet is passed through.

5.17.1. Configuration

Name	Required	Type	Description
message	true	string	A log message string to send.
severity	false	number	Log message severity (1 - lowest, 5 - maximal). Default: 1
attach_packet	false	bool	Should the packet that caused the log action be attached to the log message. (default: false)
packet_size	false	number	If attach_packet is true, how many bytes from the packet to attach. negative value means all the packet, (Default:64)

5.17.2. Read Handles

Name	Type	Description

5.17.3. Write Handles

Name	Type	Description

See log message structure in Appendix 1.

5.18. Queue

Store packets in FIFO queue. Drops incoming packets if the queue already holds CAPACITY packets.

5.18.1. Configuration

Name	Required	Type	Description
capacity	true	number	The number of packets the queue can hold.

5.18.2. Read Handles

Name	Type	Description
length	Integer	Returns the current number of packets in the queue.
highwater_length	Integer	Returns the maximum number of packets that have ever been in the queue at once.
drops	Integer	Returns the number of packets dropped by the queue so far

capacity	number	Returns the queue's capacity

5.18.3. Write Handles

Name	Type	Description
reset_counts	null	Reset all counts to zero.
reset	null	Drops all packets in queue.

5.19. FrontDropQueue

Store packets in drop-from-front FIFO queue. Drops incoming packets if the queue already holds CAPACITY packets.

5.19.1. Configuration

Name	Required	Type	Description
capacity	true	number	The number of packets the queue can hold.

5.19.2. Read Handles

Name	Type	Description
length	number	Returns the current number of packets in the queue.
highwater_length	number	Returns the maximum number of packets that have ever been in the queue at once.
capacity	number	Returns the queue's capacity

5.19.3. Write Handles

Name	Type	Description
reset_counts	null	Reset all counts to zero.
reset	null	Drops all packets in queue.

5.20. RandomEarlyDetectionQueue

Implements the Random Early Detection packet dropping algorithm. Packets which need to be dropped can be emitted on output 1 if something is connected to it.

5.20.1. Configuration

Name	Required	Type	Description

capacity	true	number	The number of packets the queue can hold.
min_threash	true	number	When the average queue length less than or equal to MIN_THRESH, input packets are never dropped.
max_threash	true	number	must be greater than or equal to MIN_THRESH. When the average queue length equals MAX_THRESH, input packets are dropped with probability MAX_P. When the average queue length is between MIN_THRESH and MAX_THRESH, input packets are dropped with probability linearly varying from 0 to MAX_P. For behavior above MAX_THRESH, see the SIMPLE argument.
max_p	true	number	Real number between 0 and 1. The probability of dropping a packet if the average queue length equals MAX_THRESH.
stability	false	number	This number determines how stable the average queue size is -- that is, how quickly it changes due to fluctuations in the instantaneous queue size. Higher numbers mean more stability. The corresponding conventional RED parameter is w_q; STABILITY equals -log_2(w_q). STABILITY should equal 1. log_2 (1 - e^(-1/K)), where K is the link bandwidth in packets per second. Default STABILITY is 4. This is very low (unstable) for most purposes; it corresponds to a link bandwidth of roughly 15 packets per second, or a w_q of 0.25. The NS default setting for w_q is 0.002, corresponding to a STABILITY of roughly 9. A STABILITY of 0 means always use the instantaneous queue length.
Gentle	false	bool	If true (the default), implement the Gentle RED variant first proposed by Sally Floyd in October 1997. In this variant, when the average queue length is between MAX_THRESH and 2*MAX_THRESH, input packets are dropped with probability linearly varying from MAX_P to 100%; at lengths above 2*MAX_THRESH all packets are dropped. If GENTLE is false, then at lengths above MAX_THRESH all packets are dropped.

5.20.2. Read Handles

Name	Type	Description
length	number	Returns the current number of packets in the queue.
highwater_length	number	Returns the maximum number of packets that have ever been in the queue at once.
capacity	number	Returns the queue's capacity
min_thresh	number	Returns the MIN_THRESH configuration parameter.
max_thresh	number	Returns the MAX_THRESH configuration parameter.
drops	number	Returns the number of packets dropped so far.
avg_queue_size	number	Returns the current average queue size.

5.20.3. Write Handles

Name	Type	Description
reset_counts	null	Reset all counts(highwater and drops) to zero.
reset	null	Drops all packets in queue.

5.21. BpsShaper

Shapes traffic to maximum rate (Bytes per second)

5.21.1. Configuration

Name	Required	Type	Description
rate	true	number	The maximum rate in Bytes/Second

5.21.2. Read Handles

Name	Type	Description
rate	number	Returns the current value of 'rate'

5.21.3. Write Handles

Name	Type	Description
rate	number	Set the value of 'rate'.

5.22. PpsShaper

Shapes traffic to maximum rate (Packets per second)

5.22.1. Configuration

Name	Required	Type	Description
rate	true	number	The maximum rate in Packets/Second

5.22.2. Read Handles

Name	Type	Description
rate	number	Returns the current value of ‘rate’

5.22.3. Write Handles

Name	Type	Description
rate	number	Set the value of ‘rate’.

5.23. BpsRateClassifier

Classifies packet stream based on the rate of packet arrival. The rate is measured in bytes per second using an exponential weighted moving average. *n* rate arguments will have *n+1* outputs. It sends packets out the output corresponding to the current rate. If the rate is less than RATE1 packets are sent to output 0; if it is >= RATE1 but < RATE2, packets are sent to output 1; and so on. If it is >= RATE*n*, packets are sent to output *n*.

5.23.1. Configuration

Name	Required	Type	Description
rate	true	array(string)	Each RATE is a bandwidth, such as "384 kbps". Earlier rates in the list must be less than later rates.

5.23.2. Read Handles

Name	Type	Description
count_i	number	Returns the number of packets emitted to output i
byte_count_i	number	Returns the number of bytes emitted to output i
rate_i	number	Returns the rate for output_i, measured by exponential weighted moving average, in packets per second.
byte_rate_i	number	Returns the rate for output_i, measured by exponential weighted moving average, in bytes per second.

5.23.3. Write Handles

Name	Type	Description
reset_i	null	Reset counts and rates of protocol i to zero.
reset_all_counts	null	Reset all counts and rates to zero.

5.24. PpsRateClassifier

Classifies packet stream based on the rate of packet arrival. The rate is measured in packets per second using an exponential weighted moving average. *n* rate arguments will have *n+1* outputs. It sends packets out the output corresponding to the current rate. If the rate is less than RATE1 packets are sent to output 0; if it is >= RATE1 but < RATE2, packets are sent to output 1; and so on. If it is >= RATE*n*, packets are sent to output *n*.

5.24.1. Configuration

Name	Required	Type	Description
rate	true	array(number)	Each RATE is the number of packets per second. Earlier rates in the list must be less than later rates.

5.24.2. Read Handles

Name	Type	Description
count_i	number	Returns the number of packets emitted to output i
byte_count_i	number	Returns the number of bytes emitted to output i
rate_i	number	Returns the rate for output_i, measured by exponential weighted moving average, in packets per second.
byte_rate_i	number	Returns the rate for output_i, measured by exponential weighted moving average, in bytes per second.

5.24.3. Write Handles

Name	Type	Description
reset_i	null	Reset counts and rates of protocol i to zero.
reset_all_counts	null	Reset all counts and rates to zero.

5.25. Store

Sends packet to storage server with some key

5.25.1. Configuration

Name	Required	Type	Description
key	true	String	Key for key-value store where the packet will be stored
soft_timeout	true	Number	Number of seconds to expire the packet record from the time it was last accessed (0 means no soft timeout)
hard_timeout	true	Number	Number of seconds to expire the packet record from the time it was stored (must be positive)
packet_size	false	number	If attach_packet is true, how many bytes from the packet to attach. negative value means all the packet, (Default:64)
buffer_size	false	number	How many alerts to buffer before sending them to OBC. if value <=1 it means not to buffer.Default is 1.
buffer_timeout	false	number	The maximum number of milliseconds to hold alerts in buffer before forcibly sending them. Default is 0.

5.25.2. Read Handles

Name	Type	Description
count	number	Returns the number of packets stored
bytes	number	Returns the number of bytes stored

5.25.3. Write Handles

Name	Type	Description
reset	null	Reset all counts zero.

5.26. Restore

Reads packet from storage given a key and if restore succeeded, replaces the content of ingress packet with the content of restored packet, starting at the given offset, or according to the given protocol name.
This block has two output ports: [0] for successfully restored packets, [1] for unrestored packets.

5.26.1. Configuration

Name	Required	Type	Description
key	true	String	Key for key-value store

offset	false	Number	Offset in packet where content restore begins
protocol	false	String	A protocol name that in fact sets the offset in packet where content of that protocol begins

One of the parameters *offset* or *protocol* must be specified.

5.26.2. Read Handles

Name	Type	Description
count	number	Returns the number of packets successfully restored
bytes	number	Returns the number of bytes successfully restored

5.26.3. Write Handles

Name	Type	Description
reset	null	Reset all counts zero.

5.27. ProtocolAnalyzer

Identifies protocols on the packet and tags its flow accordingly. Possibly also forwards the packet on a specific path.

5.27.1. Configuration

Name	Required	Type	Description
protocols	true	Array(string)	Each protocol (array element) is a protocol name (see list below).

5.27.2. Read Handles

Name	Type	Description
count_i	number	Returns the number of packets matched protocol_i
byte_count_i	number	Returns the number of bytes matched protocol_i
rate_i	number	Returns the matching rate for protocol_i, measured by exponential weighted moving average, in packets per second.
byte_rate_i	number	Returns the matching rate for protocol_i, measured by exponential weighted moving average, in bytes per second.

5.27.3. Write Handles

Name	Type	Description
reset_i	null	Reset counts and rates of protocol i to zero.
reset_all_counts	null	Reset all counts and rates to zero.

Required protocols:
arp, ipv4, tcp, udp, ftp, http, https, smtp, icmp

5.28. SessionClassifier

Classifies packet according to session information and directs it to the corresponding processing path.
Session information is stored in the session storage.
This block has N+1 outputs where output 0 is for packets that do not match any of the rules, and output i+1 is for packets that match the i'th rule.

5.28.1. Configuration

Name	Required	Type	Description
match	false	Array(map<string K, bool isField, value V>)	Each match is a mapping between the value stored in the session storage with key K, and either the value of the field (specified as a string in V) in the packet, or the value V (may be masked with % as described before). Value of isField determines whether the comparison should be done with some field value or with an exact value.

5.28.2. Read Handles

Name	Type	Description
count_i	number	Returns the number of packets matched pattern_i
byte_count_i	number	Returns the number of bytes matched pattern_i
rate_i	number	Returns the matching rate for pattern_i, measured by exponential weighted moving average, in packets per second.
byte_rate_i	number	Returns the matching rate for pattern_i, measured by exponential weighted moving average, in bytes per second.

5.28.3. Write Handles

Name	Type	Description
reset_i	null	Reset counts and rates of pattern i to zero.
reset_all_counts	null	Reset all counts and rates to zero.

5.29. SessionStore

Stores a value in session storage for later usage by SessionClassifier.

5.29.1. Configuration

Name	Required	Type	Description
fields	true	Map<string F, string K, value M>	A mapping from a field F from packet to be stored in session storage, to their corresponding key K and their corresponding mask M. M must be of the type of the field (e.g. Int16 for TCP_SRC or IPv4Address for IPV4_DST).

5.29.2. Read Handles

Name	Type	Description

5.29.3. Write Handles

Name	Type	Description

5.30. BeginTransaction

Begins a transaction. All following blocks until a corresponding CommitTransaction or RollbackTransaction block should be treated as a transaction - they should either succeed as a whole, or fail without leaving any trace.

5.30.1. Configuration

Name	Required	Type	Description
name	true	string	Transaction name

5.30.2. Read Handles

Name	Type	Description

count	number	Number of packets that started a transaction

5.30.3. Write Handles

Name	Type	Description
reset	null	Resets the counter

5.31. CommitTransaction

Attempts to commit the transaction. This block has two outputs: [0] for successful commit and [1] for unsuccessful commit. An unsuccessful commit results in an implicit rollback.

5.31.1. Configuration

Name	Required	Type	Description
name	true	string	Transaction name

5.31.2. Read Handles

Name	Type	Description
count_success	number	Number of packets that successfully committed a transaction
count_failure	number	Number of packets that failed committing a transaction

5.31.3. Write Handles

Name	Type	Description
reset	null	Resets the counters

5.32. RollbackTransaction

Rollbacks a transaction.

5.32.1. Configuration

Name	Required	Type	Description
name	true	string	Transaction name

5.32.2. Read Handles

Name	Type	Description
count	number	Number of packets that were rolled back

5.32.3. Write Handles

Name	Type	Description
reset	null	Resets the counter

5.33. FlowTracker

5.34. Decompressors

5.35. Normalizer

5.36. Compress

5.37. Encrypt

5.38. Encapsulate

5.39. EncapsulateMetadata

Encapsulates a packet using NSH together with some metadata. This is used when the OBI provides only part of the processing pipeline, and another OBI is used to continue the pipeline.

5.40. WriteMetadata

Write a value to packet's metadata

5.41. ReadMetadata

Classify the packet according to a value from its metadata

5.42. DecapsulateMetadata

Decapsulate an NSH-encapsulated packet and extract the metadata from it

5.43. HeaderPayloadClassifier

Classifies packets by header fields and payload regex match. The Classifier has N outputs, each associated with the corresponding pattern. If the packet does not match any rule it will be discarded.

PayloadPattern object:

type: "PayloadPattern"
pattern: String
is_regex: Boolean // It is ok to assume this is always true for now
from: Integer [Optional] // First start index - it is ok to not support this for now
to: Integer [Optional] // Last start index - it is ok to not support this for now

Match object:
type: "HeaderPayloadMatch"
header_match: Map<String, Value> // As defined in HeaderClassifier
payload_match: Array(PayloadPattern)

5.43.1. Configuration

Name	Required	Type	Description
match	true	Array(Match)	Each Match object is a join of a header match, as defined in the description of the HeaderClassifier block, and payload matches, as defined above.

5.43.2. Read Handles

Name	Type	Description
count	array<Integer>	Returns the number of packets matched each pattern.
byte_count	array<Integer>	Returns the number of bytes matched each pattern.
rate	array<Number>	Returns the recent matching rate for each pattern, measured by exponential weighted moving average, in packets per second.
byte_rate	array<Number>	Returns the recent matching rate for each patterni, measured by exponential weighted moving average, in bytes per second.

5.43.3. Write Handles

Name	Type	Description
reset_counts	null	Reset all counts and rates to zero.

5.44. SetTimestamp

Sets the specified timestamp, or the current system time, to packets.

5.44.1. Configuration

Name	Required	Type	Description
timestamp	false	string	Timestamp

5.44.2. Read Handles

Name	Type	Description

5.44.3. Write Handles

Name	Type	Description

5.45. SetTimestampDelta

SetTimestampDelta replaces nonzero packet timestamps with deltas.

5.45.1. Configuration

Name	Required	Type	Description
range	false	string	Sets the type of delta. The default is RANGE, which means the delta relative to the first nonzero packet timestamp encountered. Other possibilities are NOW, which means the delta between its current timestamp and now, and FIRST, which means the delta between its first timestamp annotation and its current timestamp.

5.45.2. Read Handles

Name	Type	Description
first	string	Returns the earliest nonzero timestamp observed, if any.

5.45.3. Write Handles

Name	Type	Description
reset	null	Clears the timestamp record. Future packet timestamps will be measured relative to the next nonzero timestamp encountered.