# Enhanced Packet Processing v2 Guide

EDM04-31 - Version 6

## Protection Against Harmful Interference

When present on equipment this document pertains to, the statement "This device complies with part 15 of the FCC rules" specifies the equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the Federal Communications Commission [FCC] Rules.

These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment.

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction document, may cause harmful interference to radio communications.

Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at their own expense.

## Extra Components and Materials

The product that this manual pertains to may include extra components and materials that are not essential to its basic operation, but are necessary to ensure compliance to the product standards required by the United States Federal Communications Commission, and the European EMC Directive. Modification or removal of these components and/or materials, is liable to cause non compliance to these standards, and in doing so invalidate the user's right to operate this equipment in a Class A industrial environment.

## Disclaimer

Whilst every effort has been made to ensure accuracy, neither Endace Technology Limited nor any employee of the company, shall be liable on any ground whatsoever to any party in respect of decisions or actions they may make as a result of using this information.

Endace Technology Limited has taken great effort to verify the accuracy of this document, but nothing herein should be construed as a warranty and Endace shall not be liable for technical or editorial errors or omissions contained herein.

In accordance with the Endace Technology Limited policy of continuing development, the information contained herein is subject to change without notice.

## Website

http://www.endace.com

## Copyright 2009 - 2012 Endace Technology Ltd. All Rights reserved.

# Contents



# Introduction

This document explains how to configure the Enhanced Packet Processing for DAG cards. It covers the capabilities of each of the enhanced processing steps (parse, extract, classify and steer) and describes the software interface used to configure them. This document then provides programming examples to demonstrate how to combine these functions to achieve high level functionality such as filtering, Hash Load Balancing and duplication.

This document is applicable to the following DAG cards:

- 7.4S
- 7.5G4
- 8.1SX
- 9.2X2
- 9.2SX2

## Applicable DAG software version

The features in this document require <da> software release 4.2.2 or greater.

## Related documents

The following is a list of documents referred to in this document. These are available from the Support section of the Endace website at https://support.endace.com/:

- *EDM04-35 dagcat-setup Software Guide v2*
- *EDM04-30 dagfilter-loader Software Guide*
- *EDM11-01 ERF Types*

## Glossary of Terms

| Term | Definition |
|---|---|
| CAT | Color Association Table |
| DMA | Direct Memory Access |
| ERF | Extended Record Format |
| GMR | Global Mask Register |
| HAT | Hash Association Table |
| HLB | Hash Load Balancing |
| TCAM | Ternary Content Addressable memory |
| WAN | Wide Area Network |

## Default Packet Processing Functions

When performing packet capture, all Endace DAG cards perform the same basic functions:

1. Receive traffic on one or more physical ports.
2. De-encapsulate and extract packets from the received bit stream. Ethernet or Packet Over Sonet (POS) encapsulations are typical, depending on port type.
3. Wrap each received packet in an Extended Record Format (ERF) record. Insert a timestamp into the ERF header to record the exact time the packet was received. Insert the port number the packet was received on and set the ERF record type to ERF 1. TYPE_HDLC_POS or ERF 2. TYPE_ETH. For multi-port cards, the traffic from all ports are merged into a single incoming packet stream (ensuring timestamp order per port is maintained).
4. Write the ERF record via hardware controlled Direct Memory Access (DMA) to a stream buffer that is shared with the application and resides on the host computer.

Steps 1-4 implement 100% capture of all packets and are documented in each DAG Card User Guide.

# Enhanced Packet Processing overview

Some DAG cards and their associated software provide Enhanced Packet Processing functions that can be used to simplify and accelerate packet processing software on the server. You may need to download the latest software and firmware release from the Endace website to access these features on your DAG card. Refer to the most recent DAG Card User Guide to determine if Enhanced Packet Processing is available on your DAG card.

With the Enhanced Packet Processing capability the following types of enhanced capabilities can be achieved:

1. **Filtering:** Filtering compares the fields in the packet's protocol headers to a list of user defined filter rules. As a result, the packets matching the filter are tagged with a user defined color value. This color value is subsequently used to steer the packet to specific stream buffer(s) or to drop the packet.
2. **Hash Load Balancing (HLB):** The HLB module takes the packets and applies one of a number of user selectable load balancing algorithms. This results in each packet being tagged with a bin number. The bin number is used to help determine the stream the packets are sent to. The user can choose the number of bins used in the packet tagging.
   HLB is typically used to optimize the parallel processing done by multi-core servers, since each core will only see a subset of the full traffic load. The user normally wants to distribute traffic equally (or roughly equally) across the stream buffers while preserving "flow coherence", meaning, all packets associated with a single TCP/IP session must always go to the same stream buffer. Hashing is the tool used to classify the flow coherent streams and mathematically assign a "bin number" to each packet. Bin numbers are then associated with specific stream buffers to achieve the desired load balancing. The hashing algorithms are "flow coherent" so all packets that are a part of a particular session will be tagged with the same bin number.
3. **Steering:** This is where the results of the previous steps are used to determine which stream the packet should be steered to. Optionally, the interface / port number the packet originated from can also be used in the steering process. The rules that determine how the packet steering is accomplished are defined by the user. This gives a large amount of flexibility in the packet steering process. The user can choose to use some, all or none of the color, HLB bin value and interface / port numbers in the steering process.
4. **Duplication:** One of the unique benefits of the Steering feature is the ability to steer the same packet to multiple streams. Packet duplication can simplify software architectures where different applications need to process the same packet. For example, all traffic can be sent to stream buffer 0 for use by a capture to disk application, while only a subset of traffic is sent to stream buffer 2 for handling by a traffic analysis application. For this document, packet duplication is defined as writing a single packet into more than one stream buffer. It does not mean writing the packet twice into the same stream buffer.

When properly configured by the application software, the following processing steps work together to implement the Enhanced Packet Processing functions (page 3).

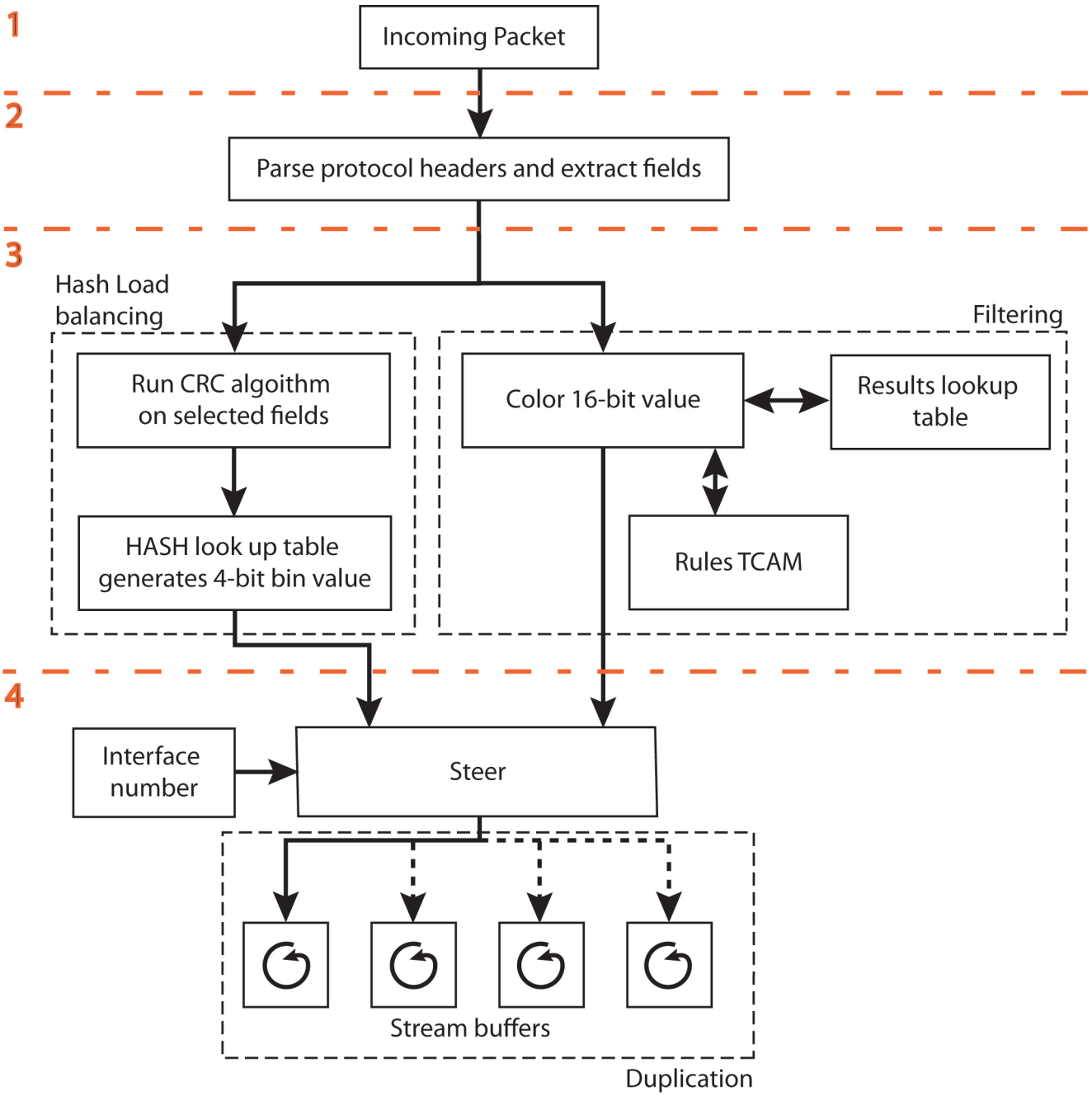
 

# Enhanced Packet Processing functions

The Enhanced Packet Processing functions are:

1. Receive traffic on one or more physical ports. For further details, see Packet Receive (page 5). This involves de-encapsulating and extracting packets from the received bit stream. Ethernet or Packet Over SONET (POS) encapsulations are assumed, depending on port type. Each packet is then wrapped in an Extended Record Format (ERF) record. Insert the timestamp, interface / port number and set the ERF record type to ERF 1. TYPE_HDLC_POS or ERF 2. TYPE_ETH. For multi-port cards, merge the traffic from all ports into a single incoming packet stream.
2. Extract the EtherType or PPP protocol field and a subset of the TCP/IP packet header fields. These are called the *extracted fields* and are used by the packet classification function below. For further details, see Packet Parsing and Header Field Extraction (page 5).
3. Tag the packet using algorithms defined by the user based on the values in the extracted fields. Two types of classification can be performed simultaneously: Hash Load Balancing and Filtering.
   - Hash Load Balancing – runs a configurable subset of the extracted fields through a standard CRC32 algorithm to produce a value out of a user definable range of bin numbers. For further details, see Hash Load Balancing (page 8).
   - Filtering – each of the packet's extracted header fields are compared against user defined rules loaded into the Ternary Content Addressable Memory (TCAM). A filter rule match results in the packet being tagged with a user-defined color. For further details, see Filtering (page 10).
4. Compares the tags associated with each packet and optionally the interface / port number to the user-defined values in the CAT (Color Association Table) color. This lookup table allows the user to define combinations of color, HLB bin and interface / port number and use these to steer the packet into one or more stream buffers in host memory. The number of stream buffers supported by each DAG card depends on type of card. Sending a packet to more than one stream buffer is valid and results packet duplication to multiple stream buffers. For further details, see Steering (page 11).

***Note:***
*These steps are illustrated in the following diagram.*



# Packet Receive

## Functional Description

Traffic is received on one or more physical ports, packets are de-encapsulated and extracted from the received bit stream, each packet is wrapped in an ERF record and the time-stamp and receive port are written into the ERF header. Traffic from all ports is merged into a single datapath.

Configuration of the Packet Receive function is fully described in your DAG Card User Guide, which describes how to load and initialize the DAG card and prepare the packet receive function for basic packet capture. For the remainder of this document, it is assumed that:

- the DAG card drivers have been installed,
- the appropriate firmware has been loaded on the DAG card, and,
- for DAG cards that support multiple protocols, the DAG card has been configured to receive the desired protocol.

# Packet Parsing and Header Field Extraction

After the packet receive process is complete, the DAG card extracts a subset of the packet's headers for later use by the Hash Load Balancing and classification functions. This extraction process differs slightly depending on whether the interface is Ethernet or SONET/SDH.

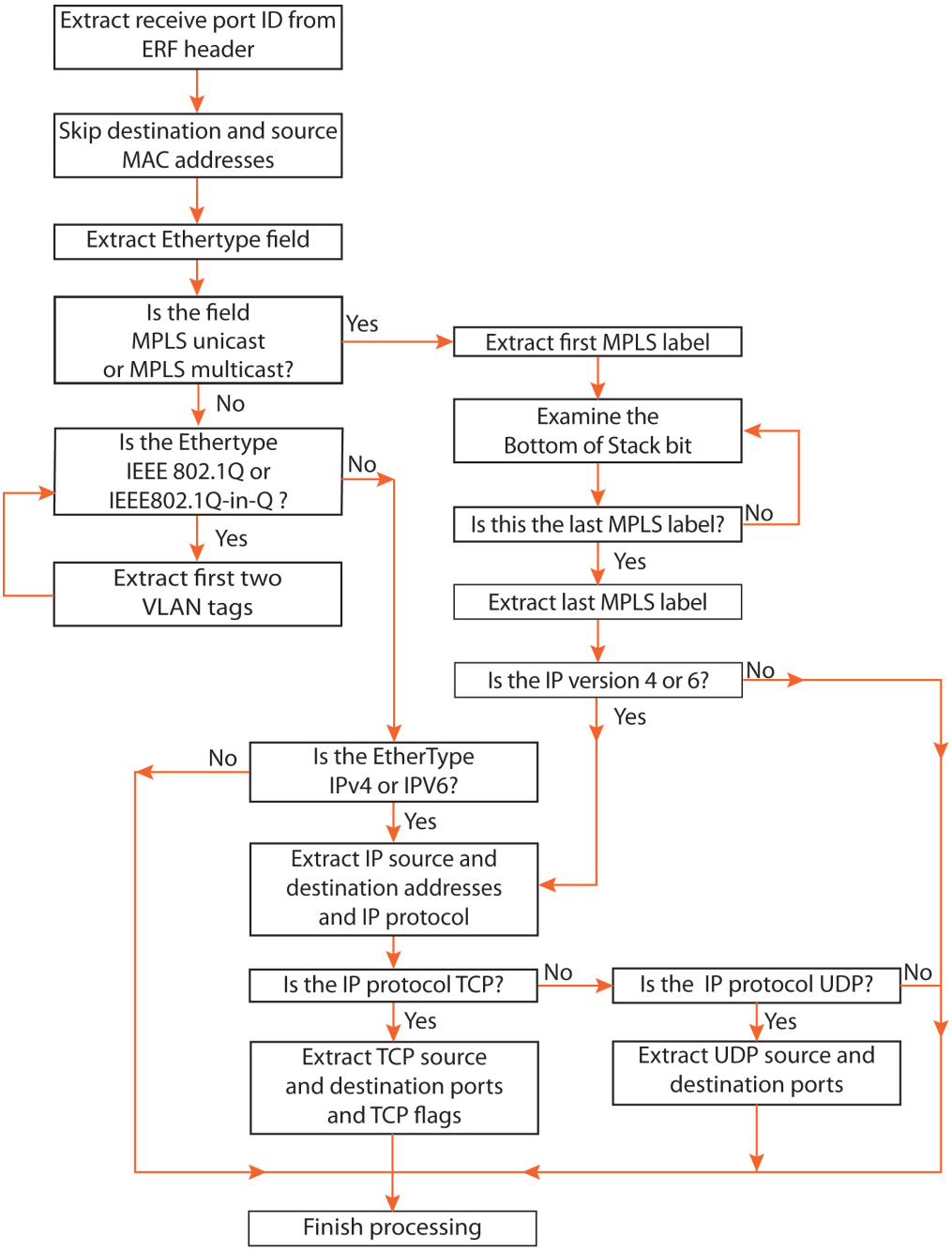
| **Filterable and Classifiable Packet Protocol Header Fields** | |
|---|---|
| Physical Interface | Interface – 2 bits |
| Layer 2: Ethernet | Ethernet Type – 2 bytes |
| Layer 2: Packet-over-SONET | PoS Protocol Field – 2 bytes |
| Layer 2/3: VLAN (Ethernet only) | VLAN Tag – 2 bytes |
| Layer 2/3: Q-in-Q (Ethernet only) | Both VLAN Tags – 2 bytes 1st VLAN Tag; 2 bytes 2nd VLAN Tag |
| Layer 2/3: MPLS | First and last MPLS headers – 4 bytes top MPLS header; 4 bytes bottom MPLS header |
| Layer 3: IPv4 | Source and Destination IP addresses – 4 bytes source address; 4 bytes destination address<br>Next Protocol – 1 byte |
| Layer 3: IPv6 | Source and Destination IP addresses – 16 bytes source IP address; 16 bytes destination IP address<br>Next Header – 1 byte |
| Layer 4: TCP | Source and Destination Ports – 2 bytes source port; 2 bytes destination port<br>TCP Flags – 6 bits |
| Layer 4: UDP | Source and Destination Ports – 2 bytes source port; 2 bytes destination port |

## Functional description - Ethernet

The TCP/IP header's location is determined dynamically (i.e. on a packet by packet basis) because the TCP/IP header's offset from the beginning of the packet depends on the layer 2 encapsulation used, as defined by the EtherType field. Supported IP encapsulation options allow 0, 1, or 2 VLAN or MPLS tags. Although the these tags cannot be used with HLB, VLAN or MPLS tags can be filtered and extracted upon and do not interfere with IP and TCP/UDP header field extraction.

Header field extraction proceeds until the protocol parser has extracted all the desired fields or until the parser can proceed no further because the protocol is not supported. The parser assigns zeros to any fields it could not extract from the packet.

For Ethernet, the parsing of these fields is explained in the following diagram:

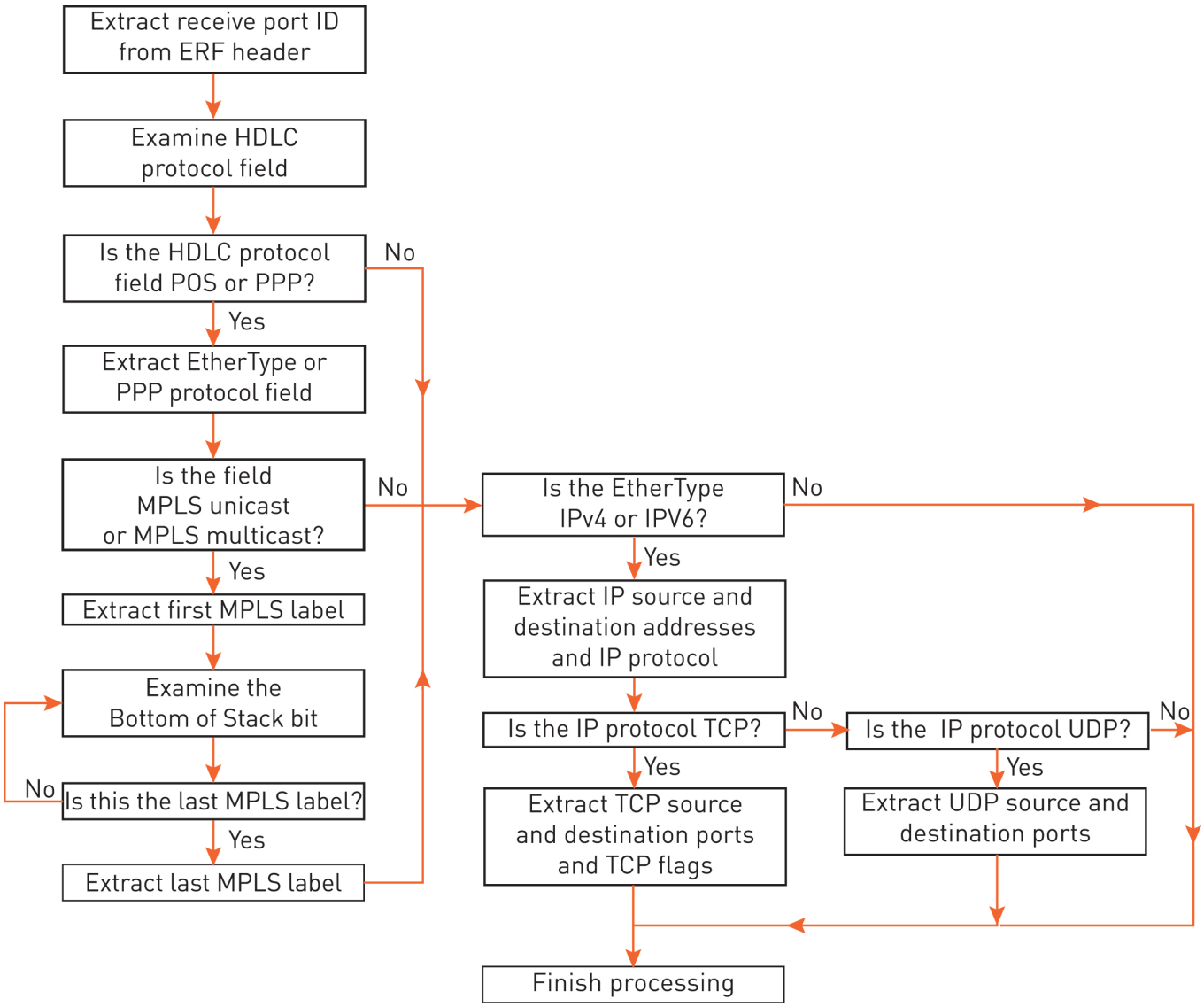



The extracted fields are concatenated together and form a contiguous word that is passed to the TCAM for filter lookup.

## Functional description – SONET/SDH

For SONET/SDH ports, the encapsulations supported are PPP in HDLC like framing (RFC1661) and Cisco HDLC (POS in this document).

The parsing is explained in the following diagram:




## How to Configure

There is no configuration required for the packet parsing function. The default behaviour is described above.

# Hash Load Balancing

## Functional description

Hash Load Balancing occurs on a configurable subset of the extracted header fields described in Packet Receive (page 5). The same subset applies to all packets received on all ports. Each extracted field is padded out to 32 bits, XOR'd together to form a single 32 bit word, which is then passed through a standard CRC32 algorithm. The 10 bit output of the CRC algorithm is sent to a user configurable lookup table called the Hash Association Table (HAT). The output of the HAT is an **N** bit value (based on the hash_width value), which is known as the packet's hash-bin.

Hash Load Balancing in this way maintains flow coherence, meaning, all packets from a single TCP/IP session produce the same 10-bit hash result so that the bi-directional stream of packets forming the TCP/IP session are always kept together for processing by a single application. Hash Load Balancing is, in some ways, the inverse of filter-based classifying. Filters are typically used to classify packets by identifying specific TCP/IP flows. With Hash Load Balancing, the TCP/IP headers are used to generate what is essentially a random value (the 10 bit hash result). It is up to the application to determine how to sort these randomized flows into the hash bins. The number of hash-bins is determined by the hash_width and can be between 2 and 256, the maximum number of hash-bins is limited by the maximum number of streams, see your DAG Card User Guide for details. Typically the goal is load balancing – the process of assigning a well defined subset of flows to each core of a multi-core server.

For example, with the `hash_width` set to 4 there are 16 hash-bins. If there are 4 processor cores available to handle packet flows, the application could configure the HAT to sort all traffic into four bins, leaving the remaining 12 bins empty. Those four bins would then be steered to four stream buffers, see Steering (page 11), and each processor core would be given a single buffer to handle. Alternatively, the HAT could be configured to use all 16 bins, with steering used to assign four bins per stream buffer.

The obvious question is why use a HAT at all? Why not just produce 4 bits directly from the CRC32 algorithm? The purpose of the HAT is to allow applications to "tweak" the assignment of flows to bins. For example, if the application finds that one flow is generating 25% of the traffic, it could use the HAT to map the 10-bit hash result for that flow to its own hash bin.

***Note:***
*The HAT can be modified "on-the-fly" while packet capture is occurring. However, changing the HAT during packet capture may result in a temporary splitting of a single TCP/IP flow across two bins.*

As mentioned above, a subset of the extracted header fields can be used for hashing. The subset to use must be configured before packet capture is enabled, and it cannot be changed without halting packet capture. The subsets supported are as follows:

- 2-tuple (IP src + IP dst)
- 3-tuple (2-tuple + IP Protocol)
- 4-tuple (3-tuple + Interface)
- 5-tuple (3-tuple + TCP/UDP src + TCP/UDP dst)
- 6-tuple (5-tuple + Interface)

The DAG card's Hash Load Balancing function is optional, and is enabled using the `dagconfig` tool provided by Endace. Hash Load Balancing can be enabled or disabled without reprogramming the `hat_range` but data capture must be halted in order to enable or disable Hash Load Balancing.

## dagconfig attributes - HLB

The following `dagconfig` attributes are applicable to the HLB module part of Enhanced Packet Processing. Using the `dagconfig -S` and `-G` options you can set and get values of the listed attributes. For more details about `dagconfig`, see your DAG Card User Guide.

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>hash_encoding_from_ipf</td><td>Enables (on) or disables (off) hash encoding in the DAG card.</td></tr>
<tr><td>n_tuple_select</td><td>Sets the hashing tuple value.
<table>
<tr><th>Tuple</th><th>Definition</th><th>n_tuple_select Values</th></tr>
<tr><td>2-tuple</td><td>IP src + IP dst</td><td>2</td></tr>
<tr><td>3-tuple</td><td>2-tuple + IP Protocol</td><td>3</td></tr>
<tr><td>4-tuple</td><td>3-tuple + Interface</td><td>4</td></tr>
<tr><td>5-tuple</td><td>3-tuple + TCP/UDP src + TCP/UDP dst</td><td>5</td></tr>
<tr><td>6-tuple</td><td>5-tuple + Interface</td><td>6</td></tr>
</table></td></tr>
<tr><td>hat_range</td><td>Sets the HAT ranges for the DAG card. Used in conjunction with the hash_width attribute. The number of hash-bins (and the number of streams) is set by the <code>hash_width</code> value (up to 32). Once the number of hash-bins is set, you need to distribute the hash output over the number of hash-bins created. The range is 0 to 1000.<br>Note:<br>You must define a hat_range for all hash-bins. The hat_range must include the whole range. You may need to tune these ranges to ensure even distribution.<br>For example :<br><code>dagconfig -d0 -S hat_range=0-90:90-160:160-200:200-250:250-310:310-370:370-430:430-500:500-560:560-620:620-680:680-750:750-810:810-870:870-940:940-1000</code><br>Caution:<br>If, by mistake, you leave gaps in the hat_range the packets will either be dropped (and the deliberate_drop count increases) or will go to stream 0.</td></tr>
<tr><td>hash_width</td><td>Sets the number of hash-bins available. 0 to 8 bits as required. The number of hash-bins created is 2 to the power of the <code>hash_width</code>.
<table>
<tr><th>Hash-bin</th><th>Number of streams</th></tr>
<tr><td>0</td><td>No Hash Load Balancing occurs.</td></tr>
<tr><td>1</td><td>2</td></tr>
<tr><td>2</td><td>4</td></tr>
<tr><td>3</td><td>8</td></tr>
<tr><td>4</td><td>16</td></tr>
<tr><td>5</td><td>32</td></tr>
</table>
Used in conjunction with the <code>hat_range</code> attribute. Once set you then need to set the <code>hat_range</code> to match the number of streams. Default = 4.<br>Note:<br>Ensure the entered <code>hash_width</code> value is appropriate for the number of streams configured on the DAG card, i.e. if you select "Z8" mode, set the <code>hash_width</code> to 3 because "Z8" mode has 8 streams.</td></tr>
</table>

# Filtering

## Functional Description

After the packet header fields are extracted the next step is to color the packet via protocol header field matching (also called filtering).

A user configurable TCAM (Ternary Content Addressable Memory) is used to simultaneously compare each bit of the packet's extracted fields against a series of bit masks (comprised of 0/1/don't care) that have been loaded into the TCAM by the application. These bit masks are typically called the "filter rules" against which the packet header is compared. The size of the TCAM determines the number of filter rules that can be stored by the TCAM. The size is product specific and is documented in your DAG Card User Guide.

When filtering each packet, the extracted header fields are presented to the TCAM, which simultaneously compares the extracted fields against each filter rule in the TCAM. Depending on the filter rules loaded into the TCAM, more than one match is possible per packet. No indication of multiple matches is provided – only the lowest numbered address that matches is returned (which equates to the first matching rule in the user defined rule file used to program the TCAM). If no match is found the TCAM returns the highest address supported by that TCAM (i.e. the last filter rule loaded). Typically an explicit "match everything" rule is included at the end of the filter record file loaded into the TCAM in order to ensure a rule match always occurs at a configured location. Alternatively you can set the `sram_miss_value` attribute, if a rule other than "match everything" is desired at the end. This would be used to specify a specific color value for packets that do not match any rules in the TCAM. See *EDM04-30 dagfilter-loader Software Guide* for further details.

In order to program the filter module, the user writes a number of filter rules in the filter rule set and tags each rule with a color. The rules are then loaded into the TCAM and the rule index and associated color (or tag) is stored in the color table. For historical reasons the filter module provides a 16 bit table for color values, however not all of these can be used. This is due to a limitation in the size of the Color Association Table (CAT) used to steer the packets to the desired receive stream (see Steering (page 11)). The size of the CAT is card dependent, see your DAG Card User Guide for more details.

Each packet is matched against all rows in the TCAM (one filter rule per row in the filter rule set). The index of the first matched row is used as a look up in the color table. The resulting value from the color table is then associated with the packet and passed to the steering module.

## dagconfig attributes - filtering

The following `dagconfig` attributes are applicable to the DAG cards. Using the `dagconfig -S` and `-G` options you can set and get values of the listed attributes.

| Option | Description |
|---|---|
| `ipf_enable` | Enables (on) or disables (off) IP Filtering on the DAG card. Default is On. |
| `sram_miss_value` | Used to set the color of the packets (as a decimal integer) that does not match any current filter rule. |
| `activate_bank` | `-S` - Sets which Bank is active, either Bank 0 (off) or Bank 1 (on).<br>`-G` - Get the number of the currently active Bank. |

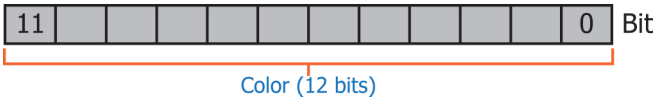
# Steering

## Functional description

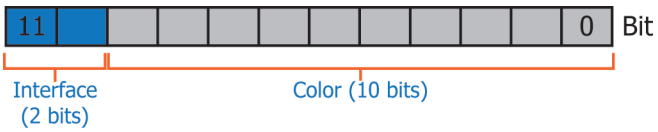
The DAG card hardware performs packet steering per-packet, per stream-buffer. Each stream buffer's steering logic uses the packet's CAT value as a key in a lookup table, the return value determines whether to DMA the packet to one or more stream buffers, or to drop (i.e. filter out) the packet. This value is a bitmap with one bit per stream. If at least one bit is set, the packet is steered to that stream (or streams where multiple bits are set). If no bits are set then the packet is dropped. Caution should be used when allowing packets to be duplicated across multiple stream buffers since this can drastically increase the bus bandwidth and cause packet loss. The number of stream buffers supported by each DAG card depends on the type of card.

The CAT value is a combination of the filter module color, the packets hash-bin value and the interface id. Since the use of HLB and interface ID is optional there are four combinations of how the CAT value can be constructed. The width of the CAT (also known as the number of input bits) is card dependent, see the relevant DAG card User Guide. Let N = `number_of_input_bits` - 1 giving the following combinations:

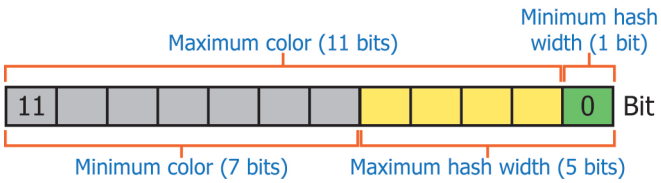
interface ID disabled, HLB disabled: Bits [N:0] = Bits [N:0] from the packet's filter color.



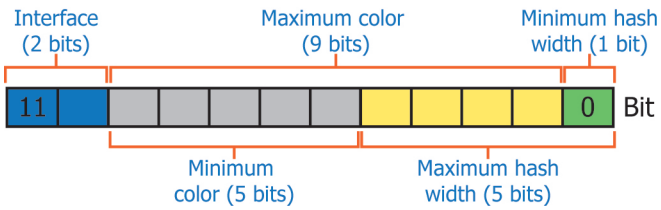

interface ID enabled, HLB disabled: Bits [N:N-1] = The interface number the packet was captured on
Bits [N-2:0] = Bits [N-2:0] from the packet's filter color.




interface ID disabled, HLB enabled: Bits [N:hash_width] = Bits [N-hash_width:0] from the packet's filter color.
Bits [hash_width-1:0] = HLB hash-bin value.




Interface ID enabled, HLB enabled: Bits [N:N-1] = The interface number the packet was captured on.
Bits [N-2:hash_width] = Bits [N-hash_width-2:0] from the packet's filter color.
Bits [hash_width-1:0] = HLB hash-bin value.




So, the total number of usable filter color bits depends on the number of HLB bits and the interface ID (port number) bits used:

```
Number_of_color_bits = number_of_input_bits - hash_width - interface_id_bits
```

The CAT is arranged in two banks, active and standby. Rules are always written to the inactive bank by `dagcat-setup` and after a successful write, the active and standby banks are switched. The user can manually switch banks without rewriting the rules by setting the `bank_select` attribute with `dagconfig`. The CAT can also be completely bypassed by setting the `by_pass` attribute in which case all packets go to stream0. See Steering and CAT attributes (page 12).

## dagconfig attributes - steering and CAT

The following `dagconfig` attributes are applicable to the CAT (Color Association Table) part of Enhanced Packet Processing steering module and CAT. Using the `dagconfig -S` and `-G` options you can set and get values of the listed attributes. For more details about `dagconfig`, see your DAG Card User Guide.

| Option | Description |
|---|---|
| `bank_select` | Activates a CAT bank without loading a new configuration. Bank 0 = off, Bank 1 = on. |
| `by_pass` | Enables (off) or disables (on) the CAT. |
| `interface_overwrite_enable` | Enables (on) or disables (off) the use of Interface / Port ID. If enabled the top two bits contain the interface/port information. |
| `deliberate_drop_count` | Read only. Use with `dagconfig -G` only.<br>Displays the number of packets dropped at the CAT due to not matching a stream. |
| `number_of_input_bits` | Read only. Use with `dagconfig -G` only.<br>Displays the number of input bits in the CAT (Color Association Table). The default is 12. |
| `stream_drop_countn` | Displays the number of packets dropped by stream "n". |

The steering function maintains a counter of all packets that are intentionally dropped because no stream buffer is configured to receive it. Under specific conditions the controller for each stream buffer may need to drop packets that are intended to be captured. This only occurs in two scenarios:

1. The application is not keeping up, the stream buffer is full, and the on-card FIFO used for that stream buffer has overflowed.
2. There is, in aggregate, more traffic being received than can be sent over the computer bus due to the inherent bandwidth limit of the bus. Performing a lot of packet duplication can create this scenario. If the bus becomes too busy then typically all stream buffers begin to experience some packet loss until the bus bandwidth is free again.

***Note:***
*If neither hashing nor filter based classification are enabled all traffic is steered to stream buffer 0. To maintain backwards compatibility, for these ERF record types (ERF type is 1 or 2) the colour field within the ERF header is used as a drop counter field. The steering hardware updates the drop counter field on every packet to reflect the number of dropped packets since the last packet sent.*

# Configuration Examples

***Note:***
*Please see* EDM04-35 dagcat-setup Software Guide v2 *and* EDM04-30 dagfilter-loader Software Guide *for information on the commands and syntax used to program the filter and steering modules.*

## Example 1:

You want to load balance all HTTP (TCP port 80) traffic across four stream buffers and drop all non-HTTP traffic. The DAG card to be used has two ports. The filter rule set to be used is `accept_port80.rule` and contains the following lines:

```
100 tcp dst-port {0000000001010000}
200 all
```

1. Load the filter rule set into the TCAM:
   ```
   dagfilter-loader -d0 --initialize --init-ports 2  -f  accept_port80.rule
   ```
   The above command:
   - Initializes the TCAM on the DAG 0 card with 2 sections (one for each port).
   - Writes the filter rule set (`accept_port80.rule`) to Bank 1, the inactive bank.
   - Makes that Bank 1 active.
2. Enable hash load balancing:
   ```
   dagconfig -d0 -S hash_encoding_from_ipf=on
   ```
3. Specify the number of fields to be used for CRC:
   ```
   dagconfig -d0 -S n_tuple_select=2
   ```
4. Configure the traffic in four hash-bins (25% each):

   ***Note:***
   *Ensure there are no spaces in the hat range.*

   ```
   dagconfig -d0 -S hat_range=0-250:250-500:500-750:750-1000
   ```
5. The CAT needs to be configured so packets marked as hash-bin 0 are set to Stream 0, packets with hash-bin 1 are set to Stream 2 etc.
   The rule file for the `dagcat-setup` is called `cat_config.rule` and contains the following lines:
   ```
   hash 0 color 100 stream 0
   hash 1 color 100 stream 2
   hash 2 color 100 stream 4
   hash 3 color 100 stream 6
   ```
   ***Note:***
   *Packets with color 200 (rest of the traffic) is intentionally not mapped to any streams. Those packets will be dropped and will be reported by the deliberate drop count.*

   ```
   dagconfig -G deliberate_drop_count
   ```
6. Load the rule to the CAT:
   ```
   dagcat-setup -d0 -f cat_config.rule
   ```

## Example 2:

You want to run three different applications and they all must receive the same data.

Classification should be enabled to pass all the packets. Hash load balancing is not required as all the packets are routed to the same stream(s). All CAT entries have bits 0, 1 and 2 set, allowing the packets to be duplicated to streams 0, 2 and 4.

1. Enable pass-all filter:
   ```
   80 all
   ```
2. The rule file for the `dagcat-setup` is called `cat_config.rule` and contains the following line to route all the packets to 3 streams.
   ```
   color 80 stream 0,2,4
   ```
3. Load the rule to the CAT:
   ```
   dagcat-setup -d0 -f cat_config.rule
   ```

## Example 3:

You want to run two applications:

- application 1 is to receive all traffic on stream 0,
- application 2 is to receive the header portion of all traffic to/from IP address `192:100:100.0/24` on stream 2.
  The snap length for application 2 will be set to 128 to shorten the packets it receives.

1. Set the snap length on Stream 2 to 128 Bytes, the rest of the streams will have the default snap-length of 10240 Bytes. Stream 2 is receive stream 1, and so use `stream_snaplength1`.
   ```
   dagconfig -d0 -S stream_snaplength1=128
   ```
2. The filter rule set for the dagfilter-loader is called `two_app_filter.rule` and contains the following lines, which classifies the packets with the given IP address with color 100 or 101.
   ```
   100  src-ip {11000000110010011001000--------}
   101  dst-ip {11000000110010011001000--------}
   200  all
   ```
3. Load the filter rule set into the TCAM:
   ```
   dagfilter-loader -d0 --initialize --iface 0 -l ethernet -f two_app_filter.rule
   ```
4. The rule file for the `dagcat-setup` is called `cat_config.rule` and contains the following lines. Any packet with color 100 or 101 will go to stream 0 and stream 2. All other packets (color 200) will go to stream 0 only.
   ```
   color [100-101] stream 0,2
   color 200 stream 0
   ```
5. Load the rule to the CAT:
   ```
   dagcat-setup -d0 -f cat_config.rule
   ```

# Example 4:

You want to run load balance the incoming traffic to five streams. A filter rule set is not required unless you also wish to filter the incoming traffic.

1. Enable hash load balancing:
   ```
   dagconfig -d0 -S hash_encoding_from_ipf=on
   ```
2. Specify the number of fields to be used for CRC:
   ```
   dagconfig -d0 -S n_tuple_select=2
   ```
3. Configure the traffic in five hash-bins (20% each):

   ***Note:***
   *Ensure there are no spaces in the hat range.*

   ```
   dagconfig -d0 -S hat_range=0-200:200-400:400-600:600-800:800-1000
   ```
4. The CAT needs to be configured so packets marked as hash-bin 0 are set to Stream 0, packets with hash-bin 1 are set to Stream 2 etc. The rule file for the `dagcat-setup` is called `cat_config.rule` and contains the following lines:
   ```
   hash 0 stream 0
   hash 1 stream 2
   hash 2 stream 4
   hash 3 stream 6
   hash 5 stream 8
   ```
5. Load the rule to the CAT:
   ```
   dagcat-setup -d0 -f cat_config.rule
   ```

# Example 5:

You want to load balance all traffic with specified VLAN fields across four stream buffers and drop all other traffic. The DAG card to be used has two ports. The filter rule set for the dagfilter-loader is called `accept_vlan.rule` and contains the following lines:

```
100 vlan-1 {0000 0000 0000 0010}
200 all
```

1. Load the filter rule set into the TCAM:

   ```
   dagfilter-loader -d0 --initialize --init-ports 2 -f accept_port80.rule
   ```

   The above command does the following things.
   - Initializes the TCAM on the DAG 0 card with 2 sections (one for each port).
   - Writes the filter rule set (`accept_vlan.rule`) to Bank 1, the inactive bank.
   - Makes that Bank 1 active.
2. Enable hash load balancing:

   ```
   dagconfig -d0 -S hash_encoding_from_ipf=on
   ```
3. Specify the number of fields to be used for CRC:

   ```
   dagconfig -d0 -S n_tuple_select=2
   ```
4. Configure the traffic in four hash-bins (25% each):

   ***Note:***
   *Ensure there are no spaces in the hat range.*

   ```
   dagconfig -d0 -S hat_range=0-250:250-500:500-750:750-1000
   ```
5. The CAT needs to be configured so packets marked as hash-bin 0 are set to Stream 0, packets with hash-bin 1 are set to Stream 2 etc. The rule file for the `dagcat-setup` is called `cat_config.rule` and contains the following lines:

   ```
   color 100 hash 0 stream 0
   color 100 hash 1 stream 2
   color 100 hash 2 stream 4
   color 100 hash 3 stream 6
   ```

   ***Note:***
   *Packets with color 200 (rest of the traffic) is intentionally not mapped to any streams. Those packets will be dropped and will be reported by the deliberate drop count. This counter will indicate the number of packets that do not match the specified VLAN tags.*

   ```
   dagconfig -G deliberate_drop_count
   ```
6. Load the rule to the CAT

   ```
   dagcat-setup -d0 -f cat_config.rule
   ```

# Example: DAG 8.1SX with Hash Load Balancing

## Overview

This describes how to configure the DAG card for Hash Load Balancing (HLB) across multiple stream buffers. The DAG 8.1SX is a single port, 10 Gigabit Ethernet and SONET (SDH) OC-192c (STM-64c) network monitoring card.

## Perspective

Endace essentially provides a multi-threaded solution to natively single-threaded network applications by applying a load-balancing algorithm to an incoming data stream. Traffic captured from a single interface or multiple interfaces is evenly distributed to multiple server cores while maintaining flow and session continuity. With an instance of a common single-threaded application deployed on each core (with processor affinity) the effective power of the network service is increased by a factor corresponding to the number of cores within the system.

## Related documentation

Before attempting to configure the DAG 8.1SX with Hash Load Balancing functionality, it is recommended that users familiarize themselves with existing Endace documentation.

- *EDM01-21 DAG 8.1SX Card User Guide* – the DAG 8.1SX Card User Guide. This document addresses installation, configuration, running a data capture session, synchronizing clock time and data formats
- *EDM04-35 dagcat-setup Software Guide v2* – the technical Guide for configuring the Color Association Table (CAT) and control packet steering.

## DAG 8.1SX Hash Load Balancing requirements

The following components are required for the DAG 8.1SX with Hash Load Balancing capabilities:

- DAG 8.1SX network monitoring card
- DAG 4.2.2 software release or greater

## Example: DAG 8.1SX Hash Load Balancing

This example explains how to use the DAG 8.1SX to Hash Load Balance over eight streams.

- Hash Load Balance to eight receive streams via a 2-tuple (Source and Destination IP addresses). All while maintaining flow coherence, meaning, all packets from a single bi-directional TCP/IP session are always kept together for processing by a single application.

### Setting up the FPGA

1. Before configuring the FPGA, load the dag drivers.
   ```
   modprobe dagmem dsize=768M
   dagload
   ```
2. Program the FPGA.
   ```
   dagrom -d0 -rvp -f dag81sxpci_bfs-eth.bit
   ```

### Configuring the card

1. Set default parameters in the DAG card.
   ```
   dagconfig -d0 default
   ```
2. Allocate memory across eight receive streams. Endace recommends allocating at least 32MB per receive stream.
   ```
   dagconfig -d0 mem=96:0:96:0:96:0:96:0:96:0:96:0:96:0:96:0
   ```

### Enabling Hash Load Balancing

The following steps enable Hash Load Balancing across eight streams in predefined mode ("z8") with 2-tuple information.

1. Enable IP hashing module, by default it is in BYPASS mode.
   Enabling the IP hashing module effectively brings the Hash Load Balancing module into the main data path within DAG 8.1SX firmware.
   ```
   dagconfig -d0 -S ipf_enable=on
   dagconfig -d0 -S hash_encoding_from_ipf=on
   ```
2. Select 2-tuple hashing.
   ```
   dagconfig -d0 -S n_tuple_select=2
   ```
   See Hash Load Balancing (page 8) for details on the mapping of tuples and their associated `n_tuple_select` values.

   ***Note:***
   If 5-tuple or 6-tuple hashing is selected and the underlying Layer 4 protocol is not TCP or UDP, the Hash Load Balancing algorithm will revert back to a 3-tuple.
3. Define a hash width.

   This specifies the maximum number of streams that packets can be hash load balanced across. In this case to eight receive-streams so the `hash_width` is 3. See Hash Load Balancing (page 8) for details on the `hash_width` options.
   ```
   dagconfig -d0 -S hash_width=3
   ```
4. Define hash bin range.
   ```
   dagconfig -d0 -S hat_range=0-125:125-250:250-375:375-500:500-625:625-750:750-875:875-1000
   ```

   ***Note:***
   In this example, the hat_range is distributed to each of the eight receive streams. After initial setup, it is worth analyzing how the network traffic is actually distributed to each receive stream and tune the ranges accordingly for a relatively even distribution.
5. Configure Hash Load Balancing feature in "z8" predefined mode.
   ```
   dagcat-setup -d0 -m z8
   ```

   ***Note:***
   Refer to EDM04-35 dagcat-setup Software Guide v2 for further details on the predefined modes.

### Accessing Configuration and Status API variables

1. Read Configuration and Status API variables.
   ```
   dagconfig -d0 -T
   ```

### Viewing DAG card statistics

1. Verify the link is set up correctly.
   ```
   dagconfig -d0 -s
   ```
2. Verify data is seen on a particular stream "n" (where "n" is particular receive stream).
   ```
   dagbits -d0:n -cvv
   ```
3. Read drop count for particular stream "n".
   ```
   dagconfig -d0 -G stream_drop_countn
   ```

Once a level of comfort has been established with the traffic distribution being Hash Load Balanced approximately equally, the end user can attach their own packet processing application to each of the eight receive streams.

#### *Changing hash bin ranges dynamically*

You can dynamically tune the hash bin range values while the captures session is running. Simply run the following command:

```
dagconfig -d0 -S hat_range=0-200:200-300:300-375:375-500:500-625:625-750:750-900:900-1000
```

Which produces a output weighting of:

```
Stream 0: 20%
Stream 2: 10%
Stream 4: 7.5%
Stream 6: 12.5%
Stream 8: 12.5%
Stream 10: 12.5%
Stream 12: 15%
Stream 14: 10%
```

## Known Issues

Refer to the relevant DAG software release notes for details on known issues.

Please report any other observed issues to support@endace.com.

# Example: DAG 9.2X2 with Hash Load Balancing

## Overviews

This example describes how to configure the DAG 9.2X2 card to Hash Load Balancing (HlB) an input data stream to eight receive stream buffers via 2-tuple hashing (source and destination IP addresses).

The DAG 9.2X2 is a dual-port, 10 Gigabit Ethernet network monitoring card.

## Perspective

Endace essentially provides a multi-threaded solution to natively single-threaded network applications by applying a load-balancing algorithm to an incoming data stream. Traffic captured from a single interface or multiple interfaces is evenly distributed to multiple server cores while maintaining flow and session continuity. With an instance of a common single-threaded application deployed on each core (with processor affinity) the effective power of the network service is increased by a factor corresponding to the number of cores within the system.

## Related documentation

Before attempting to configure the DAG 9.2X2 with Hash Load Balancing functionality, it is recommended that users familiarize themselves with existing Endace documentation.

- *EDM01-36 DAG 9.2X2 Card User Guide* – the DAG 9.2X2 Card User Guide. This document addresses installation, configuration, running a data capture session, synchronizing clock time and data formats
- *EDM04-35 dagcat-setup Software Guide v2* – the technical Guide for configuring the Color Association Table (CAT) and control packet steering.

## DAG 9.2X2 Hash Load Balancing requirements

The following components are required for the DAG 9.2X2 with Hash Load Balancing capabilities:

- DAG 9.2X2 network monitoring card
- DAG 4.0.1 software release or greater

### Example: DAG 9.2X2 Hash Load Balancing

This example explains how to use the DAG 9.2X2 to Hash Load Balance over eight streams.

- Hash Load Balance to eight receive streams via a 2-tuple (Source and Destination IP addresses). All while maintaining flow coherence, meaning, all packets from a single bi-directional TCP/IP session are always kept together for processing by a single application.

### Setting up the FPGA

1. Before configuring the FPGA, load the dag drivers.
   ```
   modprobe dagmem dsize=768M
   dagload
   ```
2. Program the FPGA.
   ```
   dagrom –d0 –rvp –f dag92xpci_bfs-xge.bit
   ```

### Configuring the card

1. Set default parameters in the DAG card.
   ```
   dagconfig –d0 default
   ```
2. Allocate memory across eight receive streams. Endace recommends allocating at least 32MB per receive stream.
   ```
   dagconfig –d0 mem=96:0:96:0:96:0:96:0:96:0:96:0:96:0:96:0
   ```

## Enabling Hash Load Balancing

The following steps enable Hash Load Balancing across eight streams with 2-tuple information.

1. Enable IP hashing module.
   Enabling the IP hashing module effectively brings the Hash Load Balancing module into the main data path within DAG 9.2X2 firmware.

   dagconfig –d0 –S ipf_enable=on
   ```
   dagconfig -d0 -S hash_encoding_from_ipf=on
   ```
2. Select 2-tuple hashing (source and destination IP addresses).
   ```
   dagconfig -d0 -S n_tuple_select=2
   ```
   See Hash Load Balancing (page 8) for details on the mapping of tuples and their associated `n_tuple_select` values.

   ***Note:***
   If 5-tuple or 6-tuple hashing is selected and the underlying Layer 4 protocol is not TCP or UDP, the Hash Load Balancing algorithm will revert back to a 3-tuple.
3. Define a hash width.
   This specifies the maximum number of streams that packets can be load balanced across. In this case to eight receive-streams so the `hash_width` is 3. See Hash Load Balancing (page 8) for details on the `hash_width` options.
   ```
   dagconfig -d0 -S hash_width=3
   ```
4. Define hash bin range.
   ```
   dagconfig -d0 -S
   ```
   ***Note:***
   In this example, the hat_range is distributed to each of the eight receive streams. After initial setup, it is worth analyzing how the network traffic is actually distributed to each receive stream and tune the ranges accordingly for a relatively even distribution.
5. Create a Color Association Table (CAT) configuration file.

   The example configuration file (cat.config) is as follows:
   ```
   hash 0 stream 0
   hash 1 stream 2
   hash 2 stream 4
   hash 3 stream 6
   hash 4 stream 8
   hash 5 stream 10
   hash 6 stream 12
   hash 7 stream 14
   ```
6. Load the file into the CAT.
   ```
   dagcat-setup -d0 -f cat.config
   ```
7. Start packet processing.
   Attach an instance of your packet processing application to each of the receive streams "n" (where "n" is particular receive stream).
   For example:
   ```
   dagbits -d0:n decode
   ```

## Accessing Configuration and Status API variables

1. Read Configuration and Status API variables.
   ```
   dagconfig -d0 -T
   ```

## Viewing DAG card statistics

1. Verify the link is set up correctly.
   ```
   dagconfig -d0 -s
   ```
2. Verify data is seen on a particular stream "n" (where "n" is particular receive stream).
   ```
   dagbits -d0:n -cvv
   ```
3. Read drop count for particular stream "n".
   ```
   dagconfig -d0 -G stream_drop_countn
   ```

Once a level of comfort has been established with the traffic distribution being Hash Load Balanced approximately equally, the end user can attach their own packet processing application to each of the eight receive streams.

### *Changing hash bin ranges dynamically*

You can dynamically tune the hash bin range values while the captures session is running. Simply run the following command:

```
dagconfig -d0 -S hat_range=0-200:200-300:300-375:375-500:500-625:625-750:750-900:900-1000
```

Which produces a output weighting of:

```
Stream 0: 20%
Stream 2: 10%
Stream 4: 7.5%
Stream 6: 12.5%
Stream 8: 12.5%
Stream 10: 12.5%
Stream 12: 15%
Stream 14: 10%
```

## Known Issues

Refer to the relevant DAG software release notes for details on known issues.

Please report any other observed issues to support@endace.com.

# Version History

| Version | Date | Reason |
|---|---|---|
| 1 | August 2009 | First release. Based on EDM 04-26. |
| 2 | September 2009 | Updated dagconfig HAT attributes (n_tuple_select). Updated dagconfig CAT attributes (hash_width). |
| 3 | December 2009 | Major rewrite. Added worked examples for the DAG 8.1SX. removed classification. |
| 4 | February 2010 | 3.4.3 release. Added 7.5G4 and multi-port. Made examples generic and removed DAG 8.1SX references. |
| 5 | September 2010 | 4.0.1 release. Added details about applicable DAG software. Added 9.2X2 to list of supported cards. Added details for hash_width. Added using 9.2X2 example. Added stream_drop_countn dagconfig attribute. Removed 8.1SX Single port EPP example - not tested against latest release. |
| 6 | May 2012 | DAG 4.2.2 release. Update DAG card list. Updated details of examples. |

endace.com