



ACL Solutions Guide

Copyright © 2011–2014 All rights reserved.

Legal Notice

Extreme Networks, Inc. reserves the right to make changes in specifications and other information contained in this document and its website without prior notice. The reader should in all cases consult representatives of Extreme Networks to determine whether any such changes have been made.

The hardware, firmware, software or any specifications described or referred to in this document are subject to change without notice.

Trademarks

Extreme Networks and the Extreme Networks logo are trademarks or registered trademarks of Extreme Networks, Inc. in the United States and/or other countries.

All other names (including any product names) mentioned in this document are the property of their respective owners and may be trademarks or registered trademarks of their respective companies/owners.

For additional information on Extreme Networks trademarks, please see:

www.extremenetworks.com/company/legal/trademarks/

Support

For product support, including documentation, visit: www.extremenetworks.com/documentation/

For information, contact:

Extreme Networks, Inc.
145 Rio Robles
San Jose, California 95134
USA

Table of Contents

Preface.....	4
Conventions.....	4
Providing Feedback to Us.....	5
Getting Help.....	5
Related Publications.....	7
Chapter 1: Introduction.....	8
Basic Policy Information.....	8
Chapter 2: Hardware ACLs.....	9
Where Does a Packet Go?.....	9
Match Criteria.....	10
Installing Rules.....	11
Double Wide Mode.....	12
Different ExtremeXOS Installation Types.....	15
Chapter 3: ACL Types.....	17
System ACLs.....	17
Policy Files.....	20
Dynamic ACLs.....	24
Chapter 4: ACL Features.....	26
Chapter 5: ACL Actions.....	29
ACL Counters.....	29
Meters.....	29
Policy-based Routing	30
User Defined Fields.....	32
Chapter 6: Common ACL Examples.....	35
Blocking ARP Traffic.....	35
Permit-Establish Policy.....	35
Traffic Metering.....	36
Layer 3 Name Flow Redirect.....	36
Layer 3 Policy Based Redirect.....	37

Preface

Conventions

This section discusses the conventions used in this guide.

Text Conventions

The following tables list text conventions that are used throughout this guide.

Table 1: Notice Icons





Icon	Notice Type	Alerts you to...
	Note	Important features or instructions.
	Caution	Risk of personal injury, system damage, or loss of data.
	Warning	Risk of severe personal injury.
	New	This command or section is new for this release.

Table 2: Text Conventions

Convention	Description
Screen displays	This typeface indicates command syntax, or represents information as it appears on the screen.
The words enter and type	When you see the word “enter” in this guide, you must type something, and then press the Return or Enter key. Do not press the Return or Enter key when an instruction simply says “type.”
[Key] names	Key names are written with brackets, such as [Return] or [Esc]. If you must press two or more keys simultaneously, the key names are linked with a plus sign (+). Example: Press [Ctrl]+[Alt]+[Del]
Words in <i>italicized type</i>	Italics emphasize a point or denote new terms at the place where they are defined in the text. Italics are also used when referring to publication titles.

Platform-Dependent Conventions

Unless otherwise noted, all information applies to all platforms supported by ExtremeXOS software, which are the following:

- BlackDiamond® X series switch
- BlackDiamond 8800 series switches

- Cell Site Routers (E4G-200 and E4G-400)
- Summit® family switches
- SummitStack™

When a feature or feature implementation applies to specific platforms, the specific platform is noted in the heading for the section describing that implementation in the ExtremeXOS command documentation. In many cases, although the command is available on all platforms, each platform uses specific keywords. These keywords specific to each platform are shown in the Syntax Description and discussed in the Usage Guidelines.

Terminology

When features, functionality, or operation is specific to a switch family, the family name is used. Explanations about features and operations that are the same across all product families simply refer to the product as the "switch."

Providing Feedback to Us

We are always striving to improve our documentation and help you work better, so we want to hear from you! We welcome all feedback but especially want to know about:

- Content errors or confusing or conflicting information.
- Ideas for improvements to our documentation so you can find the information you need faster.
- Broken links or usability issues.

If you would like to provide feedback to the Extreme Networks Information Development team about this document, please contact us using our short [online feedback form](#). You can also email us directly at InternalInfoDev@extremenetworks.com.

Getting Help

If you require assistance, contact Extreme Networks Global Technical Assistance Center using one of the following methods:

Web	www.extremenetworks.com/support
Phone	1-800-872-8440 (toll-free in U.S. and Canada) or 1-603-952-5000 For the Extreme Networks support phone number in your country: www.extremenetworks.com/support/contact
Email	support@extremenetworks.com To expedite your message, enter the product name or model number in the subject line.

Before contacting Extreme Networks for technical support, have the following information ready:

- Your Extreme Networks service contract number
- A description of the failure
- A description of any action(s) already taken to resolve the problem (for example, changing mode switches or rebooting the unit)
- The serial and revision numbers of all involved Extreme Networks products in the network

- A description of your network environment (such as layout, cable type, other relevant environmental information)
- Network load and frame size at the time of trouble (if known)
- The device history (for example, if you have returned the device before, or if this is a recurring problem)
- Any previous Return Material Authorization (RMA) numbers

Related Publications

ExtremeXOS Publications

- *ACL Solutions Guide*
- *EMS Messages Catalog*
- *ExtremeXOS Command Reference Guide*
- *ExtremeXOS Feature License Requirements*
- *ExtremeXOS User Guide*
- *ExtremeXOS Legacy CLI Quick Reference Guide*
- *ExtremeXOS Release Notes*
- *Hardware/Software Compatibility and Recommendation Matrices*
- *Switch Configuration with Chalet*
- *Using AVB with Extreme Switches*

1 Introduction

Basic Policy Information

This guide assumes that you are already familiar with Access Control Lists (ACLs), and that you understand the information presented in the ACL chapter of the *ExtremeXOS User Guide*. Please refer to that document for basic conceptual information about ACLs.

This *ACL Solutions Guide* provides advanced users more detailed information about ACL types, features, implementation details, and policy optimization.

Basic Policy Information

ACLs are used to define packet filtering and forwarding rules for traffic traversing the switch, and in ExtremeXOS, ACLs apply to all traffic.

The most basic format of an ACL is illustrated here:

```
entry <ACL_rule_name> {
    if {
        <match-conditions>;
    } then {
        <action>;
        <action-modifiers>;
    }
}
```

By default, ACL rules are always permitted. Policy file syntax also has a "match any" or "match all" preprocessor just after the word entry and name. Policy files are used for both access lists, and for routing policies. With the **match any** option the intention is that if any of the match criteria match, then the rule becomes true, and the action takes place. This is not the case for access lists as the **match any** option is not available for access lists.

2 Hardware ACLs

Where Does a Packet Go?

Match Criteria

Installing Rules

Double Wide Mode

Different ExtremeXOS Installation Types

ExtremeXOS hardware has multiple stages of ACLs. The VLAN stage is for internal use only, and is not part of this discussion. The ingress stage is a traditional access list, and all actions are supported. The egress stage is similar to ingress in that it has slices and rules, match criteria, and actions. However, the egress access lists are much more limited in scale in allowable fields and available actions. On egress, "drop," "permit," and "meter" are available, and there is some limited marking of packets, changing the dot1p, and changing the diffserv code point. But policy-based redirect, mirroring, and the L2PT feature are not available on egress.

When a packet comes into the hardware—actually a switch on a chip—the chip is performing L2 and L3 processing, is providing MAC capabilities, all packet modifications, and all packet buffering. [Figure 1: Architectural View of Hardware Ingress Access-Lists](#) on page 10 illustrates the Ingress stage, and it occurs after Layer 3 lookup.

Where Does a Packet Go?

The packet comes into the device, and MAC termination and MAC statistics occur. The packet is assigned a VLAN, and goes through Layer 2 (and potentially Layer 3) lookup, and then goes into the ingress ACL processor. The results of the forwarding database lookups can be fed in as metadata into the ingress ACL processor. The hardware uses this as implicit match criteria and explicit match criteria.

Once a packet comes into the ingress ACL stage, the field parser breaks the packet into important fields. There are too many to list here, but a short list could include Layer 2, 3, and 4 fields, MAC source, MAC destination, IP source, IP destination, Layer 4 ports, VLAN ID, outer VLAN, inner VLAN, outer priority bits, inner priority bits, protocol, TCP flags, etc., as well as metadata that is passed in the packet from forwarding database lookups.

From there, once the fields are broken up, they are fed into a number of ACL match engines—what we call slices. A slice is an independently operating TCAM. All Extreme platforms have slices that are different in number, and number of rules per slice. This dictates the achievable scale on a given platform. The basic architecture always contain some collection of slices.

So, for example, a Summit X480 has 16 slices, and each slice has 512 rules. The actual number of rules can vary per platform, and can vary per slice on a given platform. Some of the platforms have different size slices within same packet processor. Some collection of slices have 256 rules, and some collection have 128 rules. An example would be the Summit X670, or the BlackDiamond X-series I/O cards.

When a packet comes in, a parallel lookup is launched—one lookup per TCAM/slice. There can be only one match per slice. Within each search engine there is a match performed from highest precedence to lowest. Once it hits any given rule and matches, it exits out of that particular match engine. But the other match engines can also have a single match, so you can have up to ‘n’ matches for a single packet.

Match Criteria

In determining what match criteria is actually part of the key for each slice, this is an area that varies per platform, so the fields that can be matched within a give slice are actually different. The simple fields are largely the same, but the more complicated combinations of fields will vary between platforms. Refer to the “Policies and Securities” chapter of the *ExtremeXOS User Guide*.

Here is a simple illustration of how a packet traverses a slice.

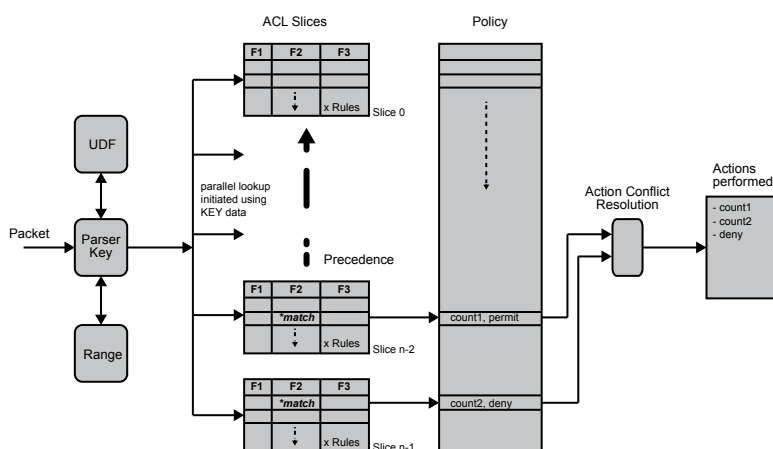


Figure 1: Architectural View of Hardware Ingress Access-Lists

There are some (n) number of field selectors, and each of these field selectors is capable of indicating that a particular field or collection of fields is being matched for this field selector. Typically, the F2 field selector is the widest key. The *ExtremeXOS User Guide* shows the key source fields (the IP source address, IP destination address, and MAC addresses, etc.) that are in the F2 selector. The F1 and F3 selectors have fields like VLAN ID, protocol, and ports, among others.

When the hardware installs an ACL for a set of match criteria that is specified, it performs a group set to select which field selector actually selects each field. The group set dictates the fields that are selected for that slice where the rule was installed.

There are three or four selectors depending on platform, and the *ExtremeXOS User Guide* has a list of field tables per platform.

In some cases the port is a fixed TCAM that is a fixed part of the key. Older platforms can have the port as part of F1, so if an ACL is installed on a physical port, part of the F1 selector has to be the port. But if it is installed on a VLAN, that part of F1 does not have to be selected. Newer platforms (for example the X670 and X460), there is a separate TCAM that handles the fixed part of the key, and the port is part of that.

Each rule is fully maskable within each TCAM. For example, if you select in F2 the IP source address and the IP destination address, that does not mean that every rule in that slice has to (also) match the same bits. You do not have to have the same source address bits or destination address bits. You can have one rule that matches the IP destination address, and not the IP source address, and that address gets masked off of that particular rule. So the next rule could match the destination address, and not the source address, and the source is completely masked off.

Installing Rules

The way Extreme Networks installs a rule into slices is an important point that is worth some discussion.

Users do not see slices; users see policy files that are a list of rules, or dynamic ACLs. Extreme Networks' platforms install the first rule of highest precedence in the highest precedence rule slice.

So when we have a collection of 'n' slices, the precedence of the slices is highest number slice down to lowest number slice. The `show access usage` command displays how many rules are in slices. The highest precedence slices always display last.

Within each slice the indices from lowest index to highest index is also the precedence order of that particular slice. So we always start and install at the highest point in the hardware, which is index 0 of slice n-1.

Installing the first rule dictates the field selector for the entire slice. If you install the next rule (rule 2) in the policy file, and if it chooses fields that are compatible (the same, or slightly different, or the same fields with different bits), that rule can go in the same slice in second highest precedence rule in hardware in that slice.

This can create a problem if the next rule is incompatible, because the fields it selects cannot be put in TCAM based on the field selectors of the first rule. An example is when the switch is in single wide mode, and one rule picks an IPv4 address, and the next rule picks an IPv6 address. Those rules cannot be in the same TCAM. Because the second rule is listed as second priority within ExtremeXOS, the next highest slice (in this case n-2) is allocated. And the hardware starts with the highest precedence rule within that slice, and picks the selectors that correspond with the field being selected (for example, if it was IPv6, F2 would pick IPv6). Now there is one rule in the first rule in highest priority slice, and one rule in second highest priority slice first priority rule of that slice, F3.

The problem is that all of the rules in the highest priority slice can never be used, because you only install rules (in precedence order by EXOS) lower than that second rule.

One of the main points to this Solutions Guide is to describe how to optimize ACL policies. Grouping rules with common field selectors will help maximize ACL hardware resources. One of the most common issues that customers experience is when they have incompatible rules adjacent to one another in a policy file indicating the precedence order. Extreme's algorithm will pick one rule in one slice, and one rule in the next slice. Depending on the platform, there are at most only 16 slices, and at

least only four slices. So you might not use all of the number of rules that are available to you, but you might use all of the number of slices that are available to you.

Installation Performance Caveats

This section offers various performance issues that you might come across, and sometimes offers ways to avoid those installation issues.

TCAM Shifting

ACLs can be time consuming to install, uninstall, and refresh. The optimal way to install rules is through the precedence order (highest precedence first, second highest, third highest). The most inefficient way to install is in the reverse order. When you install the first rule, it is by default highest precedence. If you install next rule, and it is a higher precedence than previous rule, the first rule has to be shifted one rule down. And if you install the next rule, and it is highest precedence, then the first two rules have to be shifted down one by one. So each time you are effectively reinstalling all the rules to accommodate the highest precedence rule. This is called TCAM shifting, and is very resource costly, because installing just one new rule might cause 500 other rules to be reinstalled.

Another way that you might commonly limit installation performance is when you are installing dynamic rules where you have complete control over where an individual rule goes. A common mistake is to create a number of dynamic rules and insert them, and each time you insert them you mark them as "first," meaning you want the highest precedence.

The same thing occurs when you want to uninstall rules. Optimally, you want to uninstall rules in reverse precedence order. It is important to remember that if you uninstall the highest precedence rule first, then everything shifts up, and you have the same performance problem.

There are instances where you attempt to install a small number of rules, and it can take many seconds, many minutes, or even longer to install or uninstall a new rule.

Double Wide Mode

Double wide mode is a global configuration on the switch, and it requires a reboot to take effect. Double wide mode effectively takes two independently working TCAMs and forms one logical slice. In double wide mode you generally reduce the total scale of the number of ACL rules by half. Only certain slices are capable and available for double wide mode. The BlackDiamond X cards and Summit X670 platforms have slices of different sizes, so some slices are available for double wide mode and some are not. So it is possible that when you change to double wide mode you do not necessarily get half the number of rules.

One advantage of double wide mode is that you can effectively combine the field selectors of one slice with field selectors of another slice, and increase the total key width available for a given rule. For example, if you are matching fields that are incompatible, you would not allocate new slices. So if you have an IPv4 rule, then an IPv6 rule, then another IPv4 rule, and you are in single wide mode, that policy

would consume three slices (one rule per slice). Whereas if you are in double wide mode, those rules are no longer incompatible and can be installed in three rules in one wider (larger) slice.



Note

For lower-end platforms like the Summit X440, double wide mode is much more challenging because you have system ACLs that consume ACL resources, and when you double them you only have two double wide slices. This does not allow enough resources for ACLs.

The packet goes through the match engine and all parallel search engines, and the resulting data is a policy per match. Each policy table entry has the capability of all the actions available on the rule in count, meter, redirect, drop, etc. (available in each entry). When you have a single packet that happens to match multiple rules (one rule per slice), then you have a set of actions that are all performed on the packet. There is a conflict resolution stage, where for any actions that conflict with each other (permit/deny, meter/deny, or set a QoS profile 8/QoS profile 6), only the action in highest priority slice will take place.

Examples

If you have three rules matching in n-1, n-2, slice 0 for a single packet (for instance if we set QP6 in n-2 and set QP4 in n-1), during action resolution stage the two actions conflict, but QP4 will be selected because it is in the highest priority slice.



Note

Any actions that do not conflict will all be performed. And this can cause unexpected behavior if you are not careful.

Example 1: If you have a redirection rule in a lower priority slice, and then a permit rule in a higher priority slice, permit and anything else will be performed. If you have rule 1 as **permit**, and rule 2 as **PBR**, the net result is that PBR is performed. You might expect that when you match rule 1 you exit out, but that will not happen. So how you install the rules—whether across slices or in the same slice—will determine how things are performed.

Example 2: If you have two rules that match same packet, and they bump different counters (counter 1, and counter 2), the result is that counters 1 and 2 are both incremented. Here is a typical white hole policy that creates this issue:

```
MAC 1 permit, count 1
MAC 2 permit, count 2
.....
everything else, drop and count (count anything that is unauthorized )
```

And now you insert some rules that make selection of the first set of rules incompatible with the **match all** rule. This creates the scenario where one set of rules is in one slice, and the other set of rules is in another slice. When MAC address 1 comes in, it matches the first rule, but it also matches the **drop all** rule. The actual action for the packet is correct because **permit** conflicts with **deny**, so only the highest priority slice takes effect. The problem is that the packet is permitted, but the counter is bumped and the **deny all** counter is incremented even though there was no unauthorized traffic on the network.

This behavior might change and become configurable in the future. If and when it does change, this behavior will continue to be platform dependent.

External Slices

The Summit X480, and BlackDiamond 8K XL series all have the capability to select a number of profiles as external tables. The default is to select only the Layer 2 FDB and Layer 3 routing table, and the big external TCAM. All of these platforms work as a switch on a chip, and contain all table memories and packet buffering, etc. These platforms have the same switch on a chip, still have packet buffers, and still have integrated table memories, but they have an interface to an external TCAM that we can make really large. This is what we refer to in CLI as *external tables*. Once you select one of the external table options that uses ACLs (there are two: ACL only, or L2/L3 and ACL), then you get an additional slice on these platforms. This is a big slice (on the Summit X480 the size is 60k rules). The properties of this slice are different in terms of the fields that can be matched, the installation performance, and in some cases the actions allowed.

This special external slice is conceptually slice *n* (the 17th slice on a Summit X480). ExtremeXOS dictates that when that slice is available, all user-based ACLs only go into that slice. All other internal resources are used exclusively for system ACLs. The reason is because the match criteria are different, the actions are different, and we want to maintain consistency across a particular configuration, no matter how many rules are installed.

The external slice works like regular slice architecture: when a packet comes in and parallel lookup includes this extra slice, if there is a match, there is an action resolution stage that picks non-conflicting actions and executes them on a packet. This slice is always the highest priority.

Within the external slice there is some subtle behavior—there are three partitions in this slice, IPv4, IPv6, and anything else. When a user ACL is installed into the external slice, and if the match criteria specify IPv4 criteria (IPv4 destination address), that rule is only installed into the partition that matches IPv4 packets. If the match criteria specifies IPv6 criteria (IPv6 source address), that rule is only installed in the partition that matches IPv6 packets. If the rule does not match anything that specifies V4 or V6 (for example it matches MAC address), that rule is installed in each of the partitions. This affects rule scale and rule partitions (60k rules is in IPv4 partition; Pv6 and non-IPv partitions are much smaller [4k or 2k]). So when you are installing rules that just match MAC address, and they are going into each of the three partitions, you are limited by lowest common denominator. At this point, you might be better off using the internal resources.

The main point here is that the external slice is really optimized for IPv4 policies. If you do not have IPv4 policies you probably should not use the external slice. Even within the IPv4 partition there are slightly different fields that you can select. And the number of rules that actually get consumed is different.

In one case, part of the key can be the source physical port. You can have the same policy file applied to multiple ports. Let's look at the previous example:

```
mac 1 permit (cnt 1)
mac 2 permit (cnt 2)
.....
everything else, drop and count (count anything that is unauthorized )
```

Let's assume that the three rules are all compatible, so they can all choose the same fields and be installed on the same slice. If you install the policy on Port 1, the three rules will be put into the highest priority slice, and part of the field selection (either the fixed part of the key, or selectable part of the key) will be port bitmap. Only that port that we specify (Port 1) will be chosen as part of the key.

If you take the same policy and install it on Port 2, which is resident on the same switch on a chip, the software is smart enough to see that the precedence is the same for the policy file, and it is just adding port. So the same three rules are consumed, and part of the key that selects the port bitmap now selects two bits, one corresponding to Port 1 and one to Port 2. In this instance, you are maximizing the hardware resource by only consuming three rules even though you have really installed six ACL rules from the software's perspective. This affects your installation performance, but does not use up ACL resources for hardware.

Two Important Points

One point to mention is the way counters are handled in rule compression. There is a configuration control that affects counter compression. When counter compression is on, the behavior is as described above; the three rules installed in three different hardware rules, each matching two different ports. If a packet comes in on Port 1, it is going to match one of the rules and bump the counter, but if the same data is coming in on Port 2, it will also match the same rule and increment the same counter. So you do not get two different counters for two different ports.

If compression is off, the three rules get expanded into six hardware rules with unique counters, and you can actually count per port.

The external slice does not have the port bitmap part of the key. It always behaves as though counter compression is disabled. The rules are expanded out per port. So if you have three rules on Port 1, and same three rules on Port 2, then there are six rules on the external slice. This is not configurable.

Different ExtremeXOS Installation Types

There are three different installation types in EXOS:

- Port-based
- VLAN-based
- Wildcard-based (any)

In port-based ACL installation, the rules that are installed are only on the chip that controls that physical port. All other chips in the system, the chassis (or stack), or even on a standalone Summit, do not have any ACL rules that correspond to that policy file. When you install on a VLAN, or as a wildcard, all the rules in the policy get installed on all of the chips in the system, regardless of VLAN membership.

In this case, if you have chips of different capacities in the same system (chassis or SummitStack with different slot types), or you have same packet processing capabilities on every slot (but have a different configuration on each chip that's changing the number of ACLs that are installed on those chips), those rules are attempted to be installed in slot order: slot 1-8.



Note

If there is an ACL installation failure, the whole policy is rolled back so there are no partial installations that are left intact. Your effective scale for VLAN or wildcard-based policies becomes the lowest common denominator of the scale of the system.

A policy file has other supporting tables that handle more advanced actions: one is counters, and one is meters. There are other tables in hardware: counter table and meter table. All existing Extreme platforms are capable of supporting packet counters and byte counters on the same rule (excluding

some older platforms). However, through user ACLs in ExtremeXOS, you can only have packet counters or byte counters on a given rule.

**Note**

You have to select either packet counters or byte counters.

Meters are the same, in that there is an out of profile counter, and the count is actually the i- profile counter, so you can effectively count both, and know how many packets are in-profile and how many are out-of-profile.

Example Question: What happens if you install two different meters, in two different slices, with two different rates?

Result: Two meters do not conflict, and are actually applied. What you do not get is a max bandwidth/ min bandwidth type of model. Meters operate completely independent, so when they declare packets out of profile, they are completely independent of each other. The net result is that the rate will be less than both the rates because they are independent.

**Note**

Currently, on ingress policing (meters included), we support green or red packet coloring. Green is anything in profile, red is anything over the committed rate. The rate of the meters is highly accurate, and the granularity is 8 kbps at the lowest rates. As the rate increases, the granularity of rates goes up to 1 megabit per second.

3 ACL Types

System ACLs Policy Files Dynamic ACLs

Access lists in ExtremeXOS are created using two primary interfaces, or types. The following sections describe and discuss System ACLs, Access List Policy files, and Dynamic ACLs.

System ACLs

Generally, system ACLs are utilized for features that are not ACL-based. For example, IP Multicast, Layer 3 Unicast, all QoS capabilities, mirroring on a VLAN, and VLAN statistics all use ACLs. There are also features that install ACLs behind the scenes. Some include IP Security, IDM, XNV, etc. These ACLs generally have a lower precedence than user ACLs. User ACLs can always override them.

For example, if you install an ACL to get IP Multicast missed packets to the CPU, you can install a user ACL that states you do not want a particular group address. But you also want to drop that group address, so that it will never go to the CPU.

This can create unexpected behavior. EAPS uses ACLs with a rule for a master node that states "copy" the CPU, plus drop. The intention is to take an EAPS PDU and send it to the CPU for processing. This works fine until you install a rule.

```
entry one {
    if {
        ethernet-source-address 03-00-00-00-04-00 ;
    } then {
        deny;
    }
}

entry name {
    if {
    } then {
        permit;
    }
}
```

You can add a counter or a meter to see what is going on, but when this particular kind of rule goes into a separate slice, the ACL slice has EAPS system rules that state "copy" EAPS PDUs to the CPU, and "drop" them. Concurrently, there is also a user slice that is a higher priority that says "permit."

This creates the situation where the EAPS packet gets copied to the CPU and gets permitted, which is against what EAPS wants it to do. EAPS PDUs now loop around the network. Currently, as a solution, when you install a policy that does "permit all," the system issues a warning for spanning tree and EAPS. This allows you to be more specific about what you want to permit—anything IP, anything 0806, or you can call out EAPS specifically, and not list it in the permit category.

Another example is a policy file that, even though it does not have permit listed, it has count in the hardware. So when there is a rule that states "count debug," permit actions are actually added. And this conflicts with a "deny" that is stated somewhere else. This has the same potential bad behavior with things like EAPS.

Using "permit all" on EAPS Ports

Consider the following policy entry:

```
entry PermitAll {
  if {
  } then {
    permit;
  }
}
```

If you place this ACL on one, or both of the EAPS ring ports, this will prevent EDP packets from being handled by the CPU because the ACL forces these packets to be switched by hardware. This can create a loop, and you will get the following error message when the ACL is installed on the EAPS ring ports.

```
configure access-list acl1 ports <eaps ports>
```

Note: An unconditional PERMIT action on an EAPS or STP blocked port will result in a loop. Adding an explicit match criteria such as ethernet-type will avoid these rules matching EAPS and STP PDUs.done!

To overcome this loop condition, you could add the following entry to the policy file:

```
entry EAPSPDU {
  if {
    ethernet-destination-address 00:e0:2b:00:00:04;
  } then {
    copy-cpu-and-drop;
  }
}
```

In this instance, the EAPS Flush FDB packets are sent to 00:e0:2b:00:00:07, and the shared port messages are 00:e0:2b:00:00:06.

Monitoring System ACLs

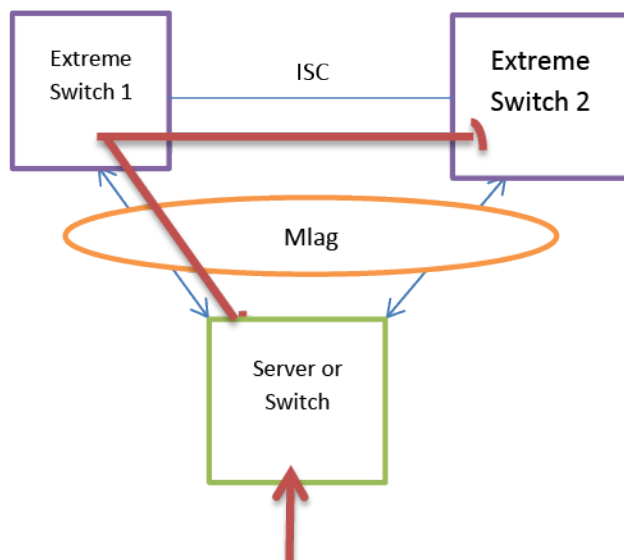
Use the `show access list usage` command to monitor system ACLs. There is an ACL **type** field that attempts to describe what types of rules are in that slice. They are broadly defined as user ACLs, system ACLs, or sometimes, specific system ACLs. Some of these ACLs leverage the fact that all non-conflicting actions take place, and VLAN statistics is one of those.

You always want VLAN statistics to increment, even though you might have user ACLs taking affect (or other system ACLs that are pushing packets to the CPU, or doing QoS). So the system will leverage the fact that it can bump two counters for a single packet. You will see that VLAN stats will grab a slice and will not let any thing else in that slice. There are other features that do this as well.

MLAG and System ACLs

Traditional MLAG

MLAG uses system ACLs in a couple of different ways. One is by blocking certain types of traffic to prevent loops. For situations where you have two Extreme switches that provide MLAG, and another device (a server, or other non-Extreme switch), if traffic comes up from the third device and chooses the path in red, and that traffic happens to be a broadcast packet across the ISC, then you have a set of ACLs that are installed on the ISC port (for convergence, and for blocking traffic to prevent loops). The first rule that is installed matches the source ISC port, and blocks any traffic to any MLAG port that happens to have a peer that is up. This is performed with a block mask capability.



MLAG System ACLs

The other system ACL is dependent on what convergence control mode you use. If you are using fast convergence mode (not the default), there are additional ACLs that are installed that match ports that are down. If you have a port that is down, there is a blocking filter that opens up, and traffic coming down from the network side hits an ACL on the switch with the down port. That ACL matches the destination port of the port that is down. FDB stays in place, performs an FDB lookup that matches the destination port of the down port, and a system ACL performs a redirect action.

Because system ACLs are always lower priority than user ACLs, after a system ACL is installed, and then you start adding user ACLs, the installation performance includes shifting the system ACL rules down every time.

Note

MLAG (and other features such as PVLAN) turns on the fabric side ACL processing to perform the necessary filtering functionality. If you apply an ingress ACL for any VLAN, this will be done on the ingress sides on any of the front port associated Packet Processors. A permit combined with an explicit deny for the ingress VLAN will result in a deny of the return path.



In the case of L3 routing, the packet modifications occur on the ingress packet processor, which means the packets will have the egress VLAN upon being received over the fabric by the egress packet processor. Therefore, with fabric side ACL processing enabled, the egress packet processor will receive an ingress packet for the egress VLAN on the fabric side port coming from the fabric.

The ACL is applied as an ingress VLAN ACL, hence it will hit the deny match criteria by the already modified packet, and this return traffic will be dropped at the fabric ingress side of the packet processor. A solution is to add a rule for the return path, or use port ACLs.

Policy Files

Policy files are text files with a .pol extension. You can create policy files through the CLI, or you can create them offline and then TFTP them into the file system.

Policy files typically have the following syntax:

- A rule entry name, unique within the same ACL policy file, or among Dynamic ACLs.
- An "if / then" clause that contains 0, or more match conditions.
- 0, or one action (permit or deny). If no action is specified, the packet is permitted by default.
- 0, or more action modifiers.

Here is an example of a rule entry:

```
entry udp_acl{
  if {
    source-address 10.10.134.0/24;
    destination-address 10.10.18.16/32;
    protocol udp;
    source-port 190;
    destination-port 1200 - 1250;
  } then {
    permit;
  }
}
```

Note



Policy file syntax also has a "match any" or "match all" preprocessor just after the word entry and name. Policy files are used for both access lists, and for routing policies. **match any** is not available for access lists. For **match any**, the intention is that if any of the match criteria match, then the rule becomes true, and the actions take place. This is not the case for access lists.

Match criteria typically falls into one of 3 categories:

- Layer 2 fields—for example, source MAC address, destination MAC address, VLAN ID, etc.
- Layer 3 fields—for example, source IP address, destination IP address, IP protocol (V4 and V6), etc.
- Layer 4 fields—for example, ICMP type, TCP port numbers, or port range (both source or destination), etc.

Generally, you can mix and match all of these different match criteria in a single rule. For instance, a Layer 2 field matching with Layer 3 and Layer 4 rules. But you cannot have all match criteria in a single rule. There is a certain width of search key that limits the number of match criteria in single rule.

After the "if / then" clause comes a list of actions. This list is quite extensive, and is beyond the scope of this document. The typical ACL actions like "permit" and "deny" are available, but action modifiers are also available. These can include the ability to change packets, change diffserve code point (DSCP), change a .IP value, or change the forwarding decision. Other actions such as mirroring/debug logs are also available.

You cannot list match criteria that conflict with each other in same rule. For example, you cannot have both matching source IPv4 and source IPv6 addresses. Similarly, you can list different actions in the same rule, but you cannot list conflicting actions in the same rule. For instance, you cannot list "permit" and "deny" in same rule.



Note

The order of the rules dictates the precedence within the policy file. So the first is highest, the second is next highest, and so on. The overall precedence in the system is dictated by the method you use to install the ACL.

Validating a Policy File

Once you have created a policy file, use the `check policy` command to validate the syntax that you specified. The `check policy attribute` command displays all possible match criteria on system.



Note

The `check policy` and `check policy attribute` commands only check the policy syntax within EXOS, including the routing policy syntax. This does not guarantee that the policy will install on target platform. Use the `configure access list` command to install the access list.

After you validate the syntax, configure the policy and instantiate it against a port, VLAN, or wildcard (the "any" token applies it to any port or any VLAN in system).

If you want to change a policy file, edit the file on the switch or offline (with TFTP), and issue the `refresh` command. This updates the policy file in hardware in several ways that depend on how extensive the changes are. It might uninstall and reinstall rules, or it might change rule inline with the hardware. For example, if you change the action from "permit" to "deny," the rule is re-installed in hardware rather than deleted and re-added. If you change just one rule, it should just modify that one rule without deleting rules that are not being changed. There is a smart refresh capability that only removes and reinstalls the rules in the policy if there are substantial changes to the policy, and that it cannot determine exactly what changes were made.

Optionally, when you perform a refresh and it has to exchange the entire policy file, you can choose to blackhole all traffic or permit all traffic while it installs the rule. Use the `[enable | disable] access-list refresh blackhole` command to configure this option. This command is enabled by default.

Policy Optimization

This section offers several examples of policies that we recommend that you avoid. We offer these examples as a way for you to think about ways to optimize the way you use ACLs.

Rule order forces different slices

```
entry one {
if {
    source-address 10.1.1.1/32;
} then {
}
}

entry two {
if {
    source-address 1111::2222/128;
} then {
}
}

entry three {
if {
    destination-address 1.1.1.1/32;
} then {
}
}
```

This is a policy you clearly want to avoid. As illustrated above, you have three rules—the first rule is an IPv4 rule, the second rule is an IPv6 rule, and the third rule is an IPv4 rule. If you are in single wide mode, these three rules are in three different slices because you defined them in this particular order.

We know that a packet that matches the first rule can never match the second rule (one is IPv4, the other IPv6). And, for the same reason, there is also no way that a packet that matches the second rule can also match the third rule.

It is possible that a packet that matches the first rule can match the third rule. So you might want to preserve the order of these three rules. A way to optimize them is to list rule one first, rule three next, and rule two last. This way they will only consume two slices.

The previous example was very clear cut, but most optimizations will be more complicated. For example, if you have rules that have more specific data like an ICMP type (which uses UDF), this forces you to use different slice.

```
entry one {
if {
    source-address 10.1.1.1/32;
} then {
}
}

entry two {
if {
    source-address 10.1.1.1/32;
} then {
}
}
```

```

entry two {
  if {
    icmp-type 1;
  } then {
  }
}
entry three {
  if {
    destination-address 1.1.1.1/32;
  } then {
  }
}

```

In this example you have the same problem as the first example. The first and the third rule use an F2 selector which specifies an IPv4 address, and the second rule uses a selector which uses UDF. So now you have to use three slices. You really want to use two slices, but the problem is that a packet matching the first rule can also match the second rule, and even match the third rule. Or there might only be two matches, so the policy file you specified really cannot be reordered unless you state that you want to make sure that ICMP types get handled or counted differently. So in this instance you have to move the rule down or up to optimize the policy.

An important point that this highlights is that the determining factor in choosing a policy is to find out what the specific behavior difference will be.

Another example to pay attention to is non-conflicting actions, and they are a more common, but difficult, problem:

```

entry one {
  if {
    source-address 10.1.1.1/32;
  } then {
    permit;
    count one;
    # Optionally can add other action modifiers
  }
}
entry two {
  if {
    arp-sender-address 1.2.3.4/32;
  } then {
    permit;
    count two;
  }
}
entry three {
  if {
    source-address 0.0.0.0/0; # Match all packets
  } then {
    deny;
    count three;
  }
}

```

In this example, the policy permits authorized things on network and denies everything else i.e permits packets with source address 10.1.1/32 or ARP packets with sender address 1.2.3.4/32 and then deny remaining packets. The rule two installation forces using a different slice and it also forces rule three onto a different slice.

Now you have used three different slices, and all of the non-conflicting actions between rule one and rule three now take place. If you have a packet that matches source address 10.1.1.1, it will be permitted and counted just as you want. Also the same packet will match rule three on different slice. So there is conflict between actions permit and deny in rule one and three respectively. Though both counters one and three get incremented, only permit action will take place since it has higher precedence.

However, the **deny anything else** statement occurs in rule three, which is on a different slice, and it also matches. So now you have permit and deny both happening, but the counter gets incremented twice. And you might think that you have unauthorized access to your network. In effect, what you permitted is what you are actually trying to deny.

To avoid this, you should inspect the policy to see that you can move entry two above entry one. This will allocate one slice. Then entry one, with all of its permit statements is now in the same slice with entry three. Now anything that matches authorized traffic will not cause the **deny all** rule to match, unless there are so many rules that it forces you to use another slice by virtue of the scale.

Dynamic ACLs

Dynamic ACLs were designed for third-party devices and other applications that use ACL infrastructure to control behavior features. Dynamic ACLs provide a way to specify, and effectively apply, the same ACLs with match criteria and actions directly through the CLI. They also provide a way for internal applications to install access lists in the background. For instance, Identity Management, XNV, and OpenFlow all use dynamic ACLs to operate properly.

The syntax for dynamic access lists is as follows:

- 1 Create an access list and give it a name, followed by two double-quoted tokens.
 - The first double quoted token is a list of match criteria.
 - The match criteria is exactly the same syntax as the match criteria in policy files.
 - The second double quoted string is a list of actions.
 - The actions available to dynamic access lists are the same as those for policy files.
- 2 Once you create a rule and give it a name, you can apply the rule to a port, VLAN, or any.

One difference between dynamic access lists and policy files is that for policy files, you can only install one policy file per ACL instance (port, VLAN, any). Dynamic ACLs allow you to install many rules per instance, and more directly influence the precedence of each individual rule. So you can specify in the CLI exactly where this newly inserted rule goes relative to other rules that are already installed.

- 3 Issue the **configure access list add** command and specify the name of ACL, and then assign it precedence (first, last, or before or after an existing rule, followed by the instance type: port, VLAN, or any).

Monitoring Dynamic ACLs

Monitoring dynamic access lists is different from policy files. The **show access list dynamic** command provides several options to view what dynamic ACLs are installed either through the CLI or through certain applications.

Precedence

Dynamic ACLs generally have a higher precedence than policy file ACLs. The same precedence holds for both policy files and dynamic ACLs, and the instantiation type further dictates precedence: port is the highest precedence, followed by VLAN, and then any. Per-rule precedence is specified through a relative indicator in the CLI.

Mixed Card Type Operation

Installation types determine which resources are used for both Dynamic ACLs and Policy File ACLs. When you install an access list on a port, that rule is installed only on the packet processor directly connected to the port. So if you are installing on a chassis, or a stack, or a standalone Summit with port groupings, that rule will only be installed to that packet processor, and will not impact the scale of the other packet processors. However, if you install an ACL policy or dynamic ACL to a VLAN, regardless of the VLAN membership, the access rules will be installed to every packet processor on the system. This also includes the any instantiation type.

This is an important point when discussing chassis and stacks with mixed modes of operation where you have coupled together slots or Summits with different capabilities. For example, if you have a stack with some Summit X440s and some Summit X460s, what is the ACL scale of the system? The answer is that it depends on how they are installed. If you install them on a port basis, you can effectively scale times the number of rules per packet processor. You get full Summit X440 scale on its ports, and separately the full Summit X460 scale on its ports.



Note

If you install them on a VLAN basis, the ACL scale of the system becomes the lowest common denominator of the slot scale which in this case is the Summit X440.



Note

If you install an ACL on a VLAN, and the VLAN does not have any port members, the ACL is still installed on all the slots in a stack, or on a chassis. The `vlan id match criteria` that you insert into a rule in the policy file, or dynamic ACL, allows you to match a specific VLAN but still use the port-based installation type, and conserve resources on a per packet processor basis. If you have an untagged packet that enters system, before it hits the hardware ACL resource it is assigned the internal ID of that VLAN.

4 ACL Features

Many ExtremeXOS features use ACLs when enabled. Some of these features include IP Security, XNV, Identity Management, QoS, OpenFlow, and others. These are provisioned through system ACLs, or dynamic ACLs. System ACLs do not go through the ACL manager application. You cannot issue the `show access list dynamic` command to see them, but if you issue the `show access-list usage` command, you can see hardware usage go up.

Specific actions create features; for example, L2PT is implemented entirely in access list syntax. This provides complete flexibility on what is performed on a packet basis. As an example, you can match any particular MAC address for a well-known protocol (like spanning tree) on ingress, and then will replace the MAC address of the spanning tree BPDU with another MAC address. The MAC address is generally multicast, and through the provider network will tunnel a packet with the other MAC address. On egress, it takes the tunnel MAC address and adds the spanning tree address back on.

Here is an example of ingress rules for spanning tree:

```
Ingress "tunnel"
entry stp-bpdu {
    if {
        ethernet-destination-address 01:80:c2:00:00:00;
    } then {
        replace-ethernet-destination-address 01:0c:0c:cd:cd:d0;
    }
}
```

This example matches the destination address of the spanning tree BPDU .1D. Here we are taking that packet and replacing the destination MAC address with another MAC address used to tunnel BPDUs across infrastructure that might also be running spanning tree .1d.

In this instance, we do not want customer spanning tree BPDUs to be confused for provider BPDUs in the network. There are similar rules for CDP and VTP.

Using this policy, you can do the same thing for CDP, VDP, or any other protocols you want to use.

All protocols have a unique destination MAC address, and they get tunneled with the exact same MAC address. So all CDP, VDP, or spanning tree protocols get that MAC address of 01:0c:0c:cd:cd:d0.

On the tunnel egress side, each rule is looking for that same MAC address, but with a different ethertype or snap type. In case of spanning tree, it is snap encapsulated with a different source SAP and destination SAP. In the case of CDP, it's looking for a snap type of hex 2000. Each egress rule translates the matching packets back to the unique defined protocol destination MAC address.

```
Egress "tunnel"
entry stp-bpdu {
    if {
        ethernet-destination-address 01:00:0c:cd:cd:cd:d0;
        destination-sap 0x42;
        source-sap 0x42;
    } then {
        replace-destination-address 01:80:C2:00:00:00;
    }
}
```

```

    }
}

```

Historically, a point of confusion occurs when the ingress tunnel is applied to an ingress port that provides access into a provider network. This allows tunneling customer spanning tree protocol packets into network provider.

L2PT access-list policies are not supported with egress access lists. Each "ingress" policy is applied to ingress ports attaching to the network (for example, VMAN). Each "egress" policy is applied to the ingress of network ports. This feature is not available on egress access lists.

Here is an example of an ACL policy file you can use to initiate the tunneling of CDP/STP/VTP. You apply this policy to the customer port on tunnel ingress: (`tunnelingress.pol`)

```

entry stp_bpdu {
    if {
        ethernet-destination-address 01:80:c2:00:00:00 ;
    } then {
        replace-ethernet-destination-address 01:00:0c:cd:cd:d0 ;
        count stp_ingress ;
    }
}
entry cdp_pdu {
    if {
        ethernet-destination-address 01:00:0c:cc:cc:cc ;
        snap-type 0x2000 ;
    } then {
        replace-ethernet-destination-add 01:00:0c:cd:cd:d0 ;
        count cdp_ingress ;
    }
}
entry vtp_pdu {
    if {
        ethernet-destination-address 01:00:0c:cc:cc:cc ;
        snap-type 0x2003 ;
    } then {
        replace-ethernet-destination-address 01:00:0c:cd:cd:d0 ;
        count vtp_ingress ;
    }
}
entry pvst_bpdu {
    if {
        ethernet-destination-address 01:00:0c:cc:cc:cd ;
    } then {
        replace-ethernet-destination-address 01:00:0c:cd:cd:d0 ;
        count pvst_ingress ;
    }
}
}

```

Here is an example of a policy file that you can use to terminate the tunneling of CDP/STP/VTP. You apply this policy to the provider port on tunnel egress: (`tunnelingress.pol`)

```

entry stp_bpdu {
    if {
        ethernet-destination-address 01:00:0c:cd:cd:d0 ;
        destination-sap 0x42 ;
        source-sap 0x42 ;
    } then {

```

```
        replace-ethernet-destination-address 01:80:c2:00:00:00 ;
        count stp_egress ;
    }
}
entry cdp_pdu {
    if {
        ethernet-destination-address 01:00:0c:cd:cd:d0 ;
        snap-type 0x2000 ;
    } then {
        replace-ethernet-destination-address 01:00:0c:cc:cc:cc ;
        count cdp_egress ;
    }
}
entry vtp_pdu {
    if {
        ethernet-destination-address 01:00:0c:cd:cd:d0 ;
        snap-type 0x2003 ;
    } then {
        replace-ethernet-destination-address 01:00:0c:cc:cc:cc ;
        count vtp_egress ;
    }
}
entry pvst_pdu {
    if {
        ethernet-destination-address 01:00:0c:cd:cd:d0 ;
        snap-type 0x010b ;
    } then {
        replace-ethernet-destination-address 01:00:0c:cc:cc:cd ;
        count pvst_egress ;
    }
}
}
```

5 ACL Actions

ACL Counters
Meters
Policy-based Routing
User Defined Fields

ACL Counters

Each ExtremeXOS ACL is able to list a counter as an action. You can give the counter a name that can be unique per rule, or you can use the same counter name for multiple rules to aggregate multiple rule counters into a single counter. There are two types of counters: packet counter and byte counter. If you just list count with the name, you get packet counters only (by default).

You can share the same counter between two rules in a single policy file by giving it a common name. Additionally, you can configure counter compression that allows the same rule being installed across multiple ports to use a single hardware resource. So when counter compression is enabled, if you have the same rule on the same counter installed on two ports on the same packet processor or port group, that rule will only use a single hardware rule. Packets that arrive on either of those ports will bump the counter. If counter compression is disabled, the same rule with the same counter installed on two different ports on the same packet processor uses two different hardware rules and two different ACL counters.

Meters

Meters are used to define ingress rate-limiting on BlackDiamond X8 and 8000 series, SummitStack, and Summit family switches. Some platforms also support meters for egress traffic. Meters have the capability to police traffic at a certain, committed rate.

The meter option is another action allowed in policy files or dynamic ACL syntax. You refer to a meter by name and you create a meter through the `create meters name` command. Meters can have committed rate, burst size, and out-of-profile action set. The configured rate configures the policing rate nature of meter. The meter option is an ingress policer, and not a rate shaper. Rate granularity is very fine and accurate, and includes interframe gap and preamble of a packet. The granularity of the rate is generally 8 kbps.

Even though there is one meter per ACL rule, Extreme implements counters so that the number of meters that we allow on a given platform is about half of the number of ACL rules.

There are three out-of-profile actions allowed: drop action (which drops anything above the rate), the ability to mark a packet with drop precedence, or mark the diffserve code point (DSCP) with a configured value. This is considered during egress congestion, or when you actually change a packet DSCP. Additionally, each meter has an associated out-of-profile counter that counts the number of packets that were above the committed-rate (and is subject to the out-of-profile-action).

Extreme only supports red or green. Red is out of profile, green is in profile, or below the rate.

The following sections provide information on meters for specific platforms.

Meters on BlackDiamond X8 and 8000 Series Switches, SummitStack, and Summit Family Switches

- On BlackDiamond 8000 series modules and Summit family switches, the meters are a per-chip, per-slice resource. The BlackDiamond X8 and 8800 series switches, SummitStack, and Summit family switches use a single-rate meter to determine if ingress traffic is in-profile or out-of-profile.
- On BlackDiamond c-, xl-, and xm-series modules and Summit X480, X650, and X670 switches, you can also use single-rate meters to determine if egress traffic is in-profile or out-of-profile.
- When ACL meters are applied to a VLAN or to any, the rate limit is applied to each port group. To determine which ports are contained within a port group, use any of the following commands:
 - `show access-list usage acl-range port port`
 - `show access-list usage acl-rule port port`
 - `show access-list usage acl-slice port port`

Max Burst Size

The max burst size is optional, and if not specified, the maximum size allowed is programmed into the hardware. This value can be quite large. In some cases, if you configure a rate without a burst size, you might wonder why the actual achieved rate is much higher than the configured rate. The reason is that bursting traffic and the burst size is allowing a large amount of data before the rate kicks in. So, in some applications you might need to configure a much smaller value.



Note

The burst size is so big that users sometimes think it has not taken effect. We recommend that if you want the rate to take effect quicker, you should configure a much smaller max burst size value.

Policy-based Routing

Policy-based routing (PBR), or redirection, has a number of variants. PBR was originally implemented with a redirect action where an administrator specifies the next hop IP address (the next hop IP address is when it's resolved, there's an ARP entry that is fully resolved with the port and next hop MAC address), and then the matching traffic gets routed to that next hop instead of to the regular forwarding decision.

A key part of the feature is that the ACL that is applied gets implicit match criteria for L3 routeable. This means that the destination MAC address has to be unicast, and has to match one of the MAC addresses that is used to route (either the switch's MAC address, or one of the virtual MAC addresses if VRRP or ESRP is used).

Policy-based redirection does not occur when packets are slow path forwarded. A common case is that routed packets with IP options are not subject to PBR.

A more advanced capability exists with the `redirect-name` action modifier. You can create a flow of redirect rule or profile, assign it a name, give it next hops and provision the priority of those next hops, and configure a health check mechanism for those next hops. You can also configure a ping check, and the redirection only occurs if the next hop is healthy.

Another feature is Layer 2 PBR, where `redirect-port` and `redirect-port-list` action modifiers allow redirection to a different port than the Layer 2, Layer 3 or multicast forwarding decision, or port list. In this instance, the egress port has to be a member of the egress VLAN that is being used. This is not a way to Layer 2 switch to a port that is not in the same VLAN.

VMANs

A VMAN is a virtual Metropolitan Area Network (MAN) that operates over a physical MAN or Provider Bridged Network (PBN). This feature allows a service provider to create VMAN instances within a MAN or PBN to support individual customers. Each VMAN supports tagged and untagged VLAN traffic for a customer, and this traffic is kept private from other customers that use VMANs on the same PBN.

The ACL `"cvid"` match criteria provides the ability to specify access lists that filter on the inner-VLAN-ID field of a double tagged packet, the customer VLAN ID field of a single tagged packet entering a VMAN UNI/CEP port, or the port-cvid inserted into an untagged packet entering a VMAN UNI port. You can use this feature to perform service-level or customer-level (cvid) rate-limiting and accounting.

You can utilize this match criteria in the following scenarios:

- Tagged VMAN ports: installing an ACL matching `"cvid"` on ingress or egress will match the inner vlan-id of a double tagged packet on a tagged VMAN port.
- Untagged VMAN ports: installing an ACL matching `"cvid"` on ingress or egress will match the single VLAN tag on an untagged VMAN port.
- CEP VMAN ports (with or without VPLS): installing an ACL matching `"cvid"` on ingress or egress will match the single VLAN tag on a CEP VMAN port (without translation).
- CEP VMAN ports with cvid translation (with or without translation): installing an ACL matching `"cvid"` on ingress will match the post-translation cvid. Installing an ACL matching `"cvid"` on egress will match the post-translation cvid.

As an example of CEP VMAN ports, consider the following configuration:

```
create vman vm1 tag 100
config vman vm1 add port 1 cep cvid 7 translate 8
config vman vm1 add port 2 tag
```

Now consider the following ACL policy applied to `"access"` port 1:

```
test.pol:
entry one {
  if {
    cvid 7;
  }
  then {
    count count7;
  }
}
entry two {
  if {
```

```

    cvid 8;
}
then {
    count count8;
}
}
config access-list test port 1
config access-list test port 1 egress

```

This results in “count8” incrementing for ingress, and “count7” incrementing on egress.

Here is another example policy:

```

entry one {
if {
    cvid 7;
    vlan-id 100;    #SVID
}
then {
    count foo;
}
}

```

And here's an example that allow you to perform service-level, or customer-level (cvid) rate-limiting and accounting:

```

doubletag.pol:
entry customer1 {
if {
    cvid 8;
}
then {
    count cust1;
}
}
create vman vm1 tag 100
config vman vm1 add port 21
config vman vm1 add port 22 tag
config access-list doubletag port 21
config access-list doubletag port 21 egress

```

User Defined Fields

User Defined Fields (UDF) provide a powerful mechanism to control network traffic. Most of the operations behind UDFs occur outside of your control, and generally you will not need to configure any UDFs. But there are certain limitations you need to be aware of and reasons why you might not want to use too many of the fields that map to UDFs.

UDF Capabilities

Any fields that are not natively recognized by hardware (for example the IP source address, or the IP destination address), can be matched up to 128 bytes in the packet. Extreme's implementation informs the hardware what is the start offset of the packet.

UDF logic has intelligence beyond the bit offset. It can recognize 1, 2, or 0 VLAN tags, IP options or not, and on some later platforms (Summit X460 and X670) it can actually start the offset in the Layer 4 header regardless of anything in the Layer 3 header (including extension headers for IPv6). This information is important because some of the match criteria takes advantage of it.

Most of Extreme's platforms support four chunks of 32 bits that can specify four different offsets, and those four offsets are parts of the packet. Then there are 32 bits after that point. If it is necessary to match only one byte, we still have to start at that 32 bit offset and mask off the other 24 bits. That uses one chunk. If it is necessary to match multiple fields, or fields longer than 32 bits like a MAC address, this will consume more than one chunk. There are two of those profiles on all of Extreme's platforms. Later platforms have eight chunks of 16 bits (rather than four chunks of 32 bits). The key width is the same, but this provides more granularity in how many fields can be specified.

There are certain user fields that correspond directly to the UDF (like ICMP type [IPv4 and IPv6], IGMP type, and ICMP code), as well as certain field combinations as UDF. As an example, one of the limitations of hardware is when in single wide mode, the F2 selector can select MAC addresses or IP addresses, but not both at the same time. Originally, there were limitations where you could not have a MAC source address and IP destination address and IP source in the same rule. You can overcome this by specifying a UDF profile for that combination of fields in a rule.

So if there is a rule that has those fields in it, allocate one of those UDFs, program the F2 selector, and instead of specifying one of those well-known fields, specify the UDF profile, and match those within the rule. This was originally performed for certain applications that wanted to match identify user based on the source MAC address, as well as to match Layer 3 and Layer 4 fields (XNV and IDM are some of those).

A key point to understand about UDFs is that there are only two profiles. If you specify rules that need a unique UDF beyond those two, you will have a resource condition when you install them. A common issue that occurs is when a user specifies rules that either have a sequence of fields that require a UDF (like a MAC source address and an IP destination/IP source, or an ARP sender address, or ICMP type). So as soon as the user tries to install a third rule, they have a resource condition regardless of how many rules are actually there.

Use case: There is a match criteria called the ARP sender address that is not part of the native hardware field parsing that goes into the UDF. A user wanted to match all ARP packets, so they entered 0.0.0.0/0, which effectively masked off the ARP sender address. This user had other rules that needed UDF like ICMP type, so they ran into a resource condition. What they could have done was use one of the well-known fields that the parser is able to detect (such as ethernet type 0x0806 [this is the ARP ethertype]), and this would have accomplished the exact same thing without using up that UDF resource.

Resource Conditions

The most common, unexpected way that users encounter a resource condition is in the number of slices they use. Generally, they install some small number of rules (for example, 100 rules) on a platform that advertises 2k rules. Then they experience a resource condition.

The second most common way is the total number of rules.

In most cases it some combination of slice and rule constraint. Generally, some of these slices (maybe one, or two) get consumed by system resources (things configured by default like dot1P examination, or IGMP snooping). These system resources consume some rules, while other rules within that slice are not used at all. When some number of slices are used for system ACLs, the remaining slices are used for user ACLs. The slices can often be filled entirely, but they still do not reach the advertised numbers for that platform.

Another resource is a table of Layer 4 port ranges. This resource is platform dependent: on some there are 16, and on others 32.

In this Layer 4 table, there will be a minimum and a maximum. So if you had a minimum port (for example, port 10 to port 20), the range hardware will take any Layer 4 port between those values, and map it to a single value that gets matched in the slice hardware. This allows you to have a single ExtremeXOS policy rule that specifies a range of source ports or destination ports (being Layer 4 ports), and it only consumes one rule in hardware.

If you specified those ranges in the EXOS policy, it is always going to try to consume this resource. If you have the same range and multiple rules, you are only going to consume one of these range elements.

So, for example, if you have

```
entry {
  if {
    source-port 10-20;
    protocol tcp;
  }
}
```

And then you have another rule with the same source-port range, but with protocol UDP, only one of the entries will be used. But if you select a different range, then you have two of these entries. Once you hit this resource (again being either 16 or 32 based on the platform), you also have a resource condition.

You can monitor this with the `show access list usage` command. You cannot monitor the UDF profile with this command.

Additionally, you can have fields within a rule that are not compatible with each other. You cannot have a field select set that matches all the fields that users might want to match. This is sometimes very subtle. Generally, however, the tables discuss fields that are not allowed.

6 Common ACL Examples

Blocking ARP Traffic
Permit-Establish Policy
Traffic Metering
Layer 3 Name Flow Redirect
Layer 3 Policy Based Redirect

This section offers several common examples of ACL policies, some caveats to avoid, and offers solutions to common policy problems.

Blocking ARP Traffic

The following policy blocks ARP between ports, and only copies the ARP packet to the CPU of the switch. This prevents attached devices (clients) to resolve each others' IP address, and serves as a first block to prevent inter-client traffic. You could use this type of policy to perform the same function with DHCP.

```
entry only_arp_to_cpu {
  if match all {
    ethernet-type 0x806 ;
  }
  then {
    copy-cpu-and-drop ;
  }
}
```

Permit-Establish Policy

The following example illustrates a "permit establish" policy. This policy is a basic requirement that allows internal users to accept TCP connections, but prevents them from opening TCP connections. This policy is applied on the ingress.

```
entry sydeny {
  if {
    protocol TCP ;
    TCP-Flags SYN ;
  }
  then {
    deny ;
  }
}
entry tcpallow {
  if {
    protocol TCP ;
  }
  then {
    permit ;
  }
}
```

```

}
}
... then some other entries for other things like UDP, etc.

entry all {
if {
}
}
then {
    deny ;
}
}
}

```

Traffic Metering

This example limits an overnight backup from taking more than half of the bandwidth on a 1 gig link. This is a basic example; you can match on the backup traffic more specifically by changing the match conditions.

- 1 Create one of the two meters. This will drop all metered traffic over the committed rate.

```

#create meter Limit_backup
configure meter "Limit_backup" committed-rate 512 Mbps out-actions drop

```

Or, you can configure this meter. This will allow the metered traffic to use more than the committed rate, but if other traffic needs to use 512 Mbps, then the meter will drop any metered traffic over the committed rate. For this meter to be fully effective it should be placed along the ingress path of the traffic.

```

create meter Limit_backup
configure meter "Limit_backup" committed-rate 512 Mbps out-actions set-drop-
precedence

```

- 2 Now create the policy file:

```

entry Limit_Backup {
if {
    source-address 25.25.25.1/32; # Device sending backup
    destination-address 10.10.10.24/32; #Device receiving the backup
}
}
then {
    meter Limit_backup;
}
}
}

```

- 3 Apply the policy to an ingress port to meter. For efficient metering place the meter closest to the source of the traffic.

Layer 3 Name Flow Redirect

The following example illustrates policy-based Layer 3 with name flow redirect. This example redirects all web traffic (http and https) to a proxy server that is alive. The flow-redirect has two servers configured, and the highest priority flow-redirect will be used. If the flow-redirect destination host does not respond to a ping it will switch to the backup.

- 1 First, configure flow redirect:

```
create flow-redirect ToProxy
configure flow-redirect ToProxy add nexthop 10.20.11.10 priority 200
configure flow-redirect ToProxy add nexthop 10.20.12.10 priority 100
configure flow-redirect ToProxy health-check ping
```

- 2 Now create a policy:

```
entry Allhttp {
if {
    protocol tcp;
    source-address 10.0.0.0/8;
    source-port 80;
}
then {
    redirect-name ToProxy;
    count WebHTTP;
}
}
entry Allhttps {
if {
    protocol tcp;
    source-address 10.0.0.0/8;
    source-port 443;
}
then {
    redirect-name ToProxy;
    count WebHTTps;
}
}
```

- 3 Now apply the ACL on ports, or VLANs.

Layer 3 Policy Based Redirect

This example illustrates Layer 3 Policy-based routing with a single next hop.

On the Sender switch you have VLAN "Incoming" on port 1. On the PBR-Switch (Port 2), you have VLAN "Outgoing" [with a L3 Next Hop].

The Sender switch belongs to subnet 10.0.0.0/24, and sends traffic to IP 10.2.0.2 which belongs to a destination subnet behind device "L3 Next Hop".

The PBR-Switch receives this traffic on VLAN "Incoming" on port 1, and forwards it towards the next hop "L3 Next Hop" out of port 2 on VLAN "Outgoing" (subnet 10.1.0.0/24) according to the installed policy.

This example focuses on the configuration of the PBR-Switch that does the L3 PBR, however some information is necessary:

- "Sender" is sending the following traffic:
 - Source IP: 10.0.0.2
 - Destination MAC: *MAC of "PBR-Switch"*
 - Destination IP: 10.2.0.2

- "L3 Next Hop" has VLAN "Outgoing" configured with IP 10.1.0.2/24 and is connected to PBR-Switch port 2.
- "L3 Next Hop" needs to respond to ARP using its interface 10.1.0.2, so "PBR-Switch" has a valid next-hop address to forward the traffic to.

Configure the following policy on the "PBR-Switch"

- 1 Create policy pbr.pol.

```
X670V-48t.1 # vi pbr.pol
entry pbr {
  if {
    destination-address 10.2.0.0/24;
  }
  then {
    count L3PBR;
    redirect 10.1.0.2;
  }
}
```

- 2 Configure the two VLANs on "PBR-Switch". This example uses untagged ports, so first make sure these ports are not configured untagged at another VLAN. You can use tagged ports as well.

```
create vlan "Incoming"
configure vlan Incoming add ports 1
configure vlan Incoming ipaddress 10.0.0.1 255.255.255.0
enable ipforwarding vlan Incoming

create vlan "Outgoing"
configure vlan Outgoing
configure vlan Outgoing add ports 2
configure vlan Outgoing ipaddress 10.1.0.1 255.255.255.0
enable ipforwarding vlan Outgoing
```

- 3 Apply the ACL.

```
# configure access-list pbr vlan "Incoming" ingress
X670V-48t.3 # show policy pbr
Policies at Policy Server:
Policy: pbr
entry PBR {
  if match all {
    destination-address 10.2.0.0/24 ;
  }
  then {
    count L3PBR ;
    redirect 10.1.0.2 ;
  }
}
Number of clients bound to policy: 1
Client: acl bound once

X670V-48t.4 #
```

- 4 Check if ACL counters are increasing and traffic is forwarding.

```
X670V-48t.4 # show access-list counter
Policy Name      Vlan Name      Port Direction
Counter Name    Packet Count   Byte Count
=====
```

```
pbr Incoming          *          ingress
L3PBR                 22707495

X670V-48t.4 # show access-list counter
Policy Name          Vlan Name      Port Direction
Counter Name         Packet Count  Byte Count
=====
pbr Incoming          *          ingress
L3PBR
X670V-48t.4 #
```