USING MX SERIES AS A SERVER LOAD BALANCER

Leveraging ECMP and the Trio 3D Chipset to Integrate Functionality
# Table of Contents

- Introduction .................................................................................................................. 3
- Scope ................................................................................................................................ 3
- Design Considerations ..................................................................................................... 3
- Server Load Balancing Overview .................................................................................... 3
- Configuring the MX as a Server Load Balancer (for Transparent Proxy) .................... 6
  - Transparent Proxy Configuration .................................................................................. 7
  - ECMP Configuration ..................................................................................................... 9
  - RPM Configuration ....................................................................................................... 10
  - SLAX Script ................................................................................................................... 12
- Summary ......................................................................................................................... 14
- About Juniper Networks ................................................................................................. 14

# Table of Figures

- Figure 1. Load balancer service paths ............................................................................. 4
- Figure 2. Load balancer functions .................................................................................. 4
- Figure 3. Selecting servers based on link utilization ....................................................... 5
- Figure 4. Using MX as SLB for OTT transparent caching ............................................. 6
- Figure 5. MX L3 SLB functions ...................................................................................... 7
Introduction

Internet services from the earliest web sites to today’s growing cloud-based services have scaled by load balancing across sets of servers that logically appear as a single network element. Within a data center, many applications are satisfied by the Layer 3 (L3) server load balancer (SLB) features, which are implemented in Junos software. This application note describes how to configure the MX Series router to support L3 SLB.

Scope

This document provides an overview on how the MX Series router provides L3 SLB functionality. In addition, this document explores basic and advanced SLB functionality, which can aid network administrators in evaluating the features that satisfy their SLB requirements.

Design Considerations

The MX Series router supports reliable L3 SLB functionality directly onto the chassis without additional hardware. This capability begins with the Trio 3D chipset’s ability to identify and act on L3 information directly on hardware so that only the traffic that is required for the specific SLB is forwarded to the SLB function. Another crucial feature for L3 SLB functionality is the unique ability for the Trio 3D to route the SLB traffic based on a symmetric hash generated without special provisioning or additional hardware components. While symmetric routing can be configured on non-Trio 3D chipset-based Junos routers, support of it greatly eases configuration of L3 SLB functionality described herein. The capability described here has been tested on an MX Series router running Junos 10.2 software.

The SLB functionality was validated in conjunction with transparent proxy (tproxy) functionality on the MX router. The MX Series is equipped with a 16-port 10GE Modular Port Concentrator (MPC) for connecting to clients and the network, as well as a Dense Port Concentrator (DPC) supporting the 1GE connection to Juniper Networks Media Flow Controllers (MFC), which are used for caching.

Server Load Balancing Overview

L3 SLB functionality can be used within a data center to distribute the workload among multiple servers and can be used to improve service availability by directing traffic only to those servers which are operational and available.

The primary role of the SLB is to improve scaling by mapping inbound service requests to a pool of physical resources (typically servers). Early service load balancing uses Domain Name System (DNS) to map a single URL to multiple physical servers, which may be located at the same or different locations. This allows resource pools to be configured in a straightforward manner and is particularly useful for geographic server load balancing (GSLB), which distributes loads to other POPs if the local servers cannot support the request. To do so, we provision the GSLB with alternate GSLB addresses in case no local resources can service adequately any incoming requests. As a result, requests are forwarded to the alternate GSLB.

Functional SLBs, as either separate devices or integrated into network elements such as routers, have evolved to provide more sophisticated and application-oriented capabilities. The load-balanced service request is sent to a virtual IP address and port number that represents the service interface, but transparently directs the request to the load balancer. The IP address and port number is inspected and an available resource is selected, using a round-robin or other algorithm. This mapping must be stateful so that all associated connection traffic is NAT’ed in the same way to ensure that the same service resource is delivering the service to the client. If packets are required to route back though the SLB, the client source IP is NAT’ed to an interface on the SLB, otherwise the client IP address is retained as the source IP address. The latter mode is called direct server return (DSR), and is commonly used when streaming or delivering content back to the client to save SLB bandwidth and reduce latency.
The most basic functional segmentation is based on how deep the SLB can look into the packet to determine where to forward the packet. These include:

- **Layer 3 SLB**: The SLB looks at the source and or destination IP addresses to determine where to forward the request. An application such as over-the-top (OTT) content caching, where all requests use the same TCP port (HTTP uses port 80), only requires L3 balancing within the data center.

- **Layer 4 SLB**: These devices can make load-balancing decisions based on TCP/UDP ports as well as IP addresses. Because the physical servers behind load balancers often run applications that use multiple session interfaces (such as HTTP or FTP), providing service availability becomes more complex than single session cases. For instance, when an online provider offers a service that combines HTTP (port 80) and a proprietary service (port 49200); for the service to be properly balanced, the SLB must only consider servers that provide both types of services. Figure 1 illustrates this simple example, where SLB occurs across servers A and C because only these servers support the required ports.

![Figure 1. Load balancer service paths](image)

- **Layer 7 SLB**: The SLB can look beyond the IP header to determine where to forward the packet. A common SLB feature enhancement is to add DPI (deep packet or application layer inspection) to filter traffic based on information in the application layer.

![Figure 2. Load balancer functions](image)
Figure 2 shows a few of the many SLB functions available to improve service availability and reliability. Some of the potential functions include:

- **Connection management:** The SLB can offload processor- or memory-intensive functions from the individual servers, allowing them to spend more resources servicing client requests. For example, the SLB can be the termination point for SSL encryption. Traditional SLB vendors have added caching to their feature sets, but often become throughput bottlenecks because their cache architectures are general purpose and limited in functionality and therefore are unable to handle multiple object sizes. However, VoIP softswitches are prime examples of application-oriented SLB requirements, as control plane-aware traffic policing, as well as client-facing security requirements often demand the highly customizable and feature-integrated SLB.

- **Resource selection:** The SLB can determine which server to forward the request to, based on server location or available bandwidth. The simplest (and perhaps most common) mechanism is round-robin, where requests are forwarded sequentially to the next server in the list. In addition, many SLBs can detect server outages and non-responsive applications, and can remove these from the forwarding list.

- **Health monitoring:** The SLB can bypass servers that appear to be non-responsive or sluggish. Physical server and path load monitoring are other ways to improve service performance as the SLB makes the server selection. These are independent factors, though path load is more relevant to configurations where the SLB is in the return path, as opposed to DSR mode. In Figure 3, the SLB would select Server A over Server C since it is lightly loaded (shown by line thickness).

Figure 3. Selecting servers based on link utilization

Adding resource health monitoring functions and using information derived from it to influence the resource selection algorithm significantly improves service availability and reliability. The most basic type of health monitoring is L2/L3 connection testing, for instance issuing a PING. This technique verifies whether the service resource network interface is reachable and active. However, it does not verify that the actual service application is running and available to process requests. PING also does not provide path or service application load information, which would serve to enhance further the resource selection and availability capabilities of the load balancer.
Configuring the MX as a Server Load Balancer (for Transparent Proxy)

Figure 4 shows an example of OTT caching using transparent proxy in the MX routers, which are also configured to support L3 SLB functionality. This application was used to validate SLB functionality in the MX Series router. The MX supports L3 round robin server load balancing.

As summarized in Figure 5, configuring the MX to support SLB functionality (for transparent proxy) requires the following steps:

1. Configure the transparent proxy functionality. This is done by identifying the endpoints, which will have their requests redirected to the MFC.
2. Configure the router to use Equal Cost Multi-Path (ECMP) to load-balance across multiple servers. ECMP is used as a simple equal weight next-hop server selection algorithm. The SLB routing table contains only the addresses of the pool of next-hop servers.
3. Configure the router to use Real-time Performance Monitoring (RPM) to detect when a server failure occurs. RPM probes also track round trip times, along with jitter and latency to be calculated by being carrying within Internet Control Message Protocol (ICMP) and User datagram Protocol (UDP) echo and time stamp, HTTP Get, and TCP connection messages. For configurations where RPM probe replies are not available (next hop does not support RPM), only RTT and next-hop availability are supported.
4. Create a SLAX (Stylesheet Language Alternative Syntax) script to redirect traffic when a failure occurs. SLAX scripts allow a RPM response state to trigger alternate next hops. SLAX is a language for writing Junos commit and automation scripts and is an alternative to Extensible Stylesheet Language Transformations (XSLT). Although SLAX has a distinct syntax, it has the same semantics as XSLT.
The configuration for each step is shown below.

**Transparent Proxy Configuration**

The following snippet shows how to configure the MX router to support transparent proxy (tproxy). Client- and network-facing interfaces include the tproxy filter statement, which define tproxy operation. Include and exclude lists are also defined to simplify the filter configuration.

```plaintext
Interfaces{
  xe-2/0/0 {
    description "Directly connected clients";
    vlan-tagging;
    unit 1014 {
      vlan-id 1014;
      family inet {
        filter {
          input tproxy;
        } address 10.254.14.1/24;
      } family iso;
    }
  }
  xe-2/0/1 {
    description "To metro network";
    unit 0 {
      description "to MX-2 (core network)";
      family inet {
        filter {
          input tproxy;
        } address 10.253.1.1/30;
      } family iso;
    }
  }
  ge-0/2/9 {
    description "to MFCs";
  }
}
```
unit 0 {
  family inet {
    address 10.251.11.1/29;
  }
  family iso;
}
}
policy-options { # Addresses to be included and excluded in the Firewall filter
  prefix-list tproxy-client-include {
    0.0.0.0/0;
  }
  prefix-list tproxy-client-exclude;
  prefix-list tproxy-origin-exclude {
    10.253.1.0/30;
    10.253.1.4/30;
    172.28.113.1/32;
  }
  prefix-list tproxy-origin-include {
    0.0.0.0/0;
  }
}
firewall {
  family inet {
    filter tproxy {
      term origin-to-exclude { # Pass excluded traffic
        from {
          destination-prefix-list {
            tproxy-origin-exclude;
          }
          protocol tcp;
          destination-port 80;
        }
        then {
          count tproxy-origin-exclude;
          accept;
        }
      }
      term origin-from-exclude { # Pass excluded traffic
        from {
          source-prefix-list {
            tproxy-origin-exclude;
          }
          protocol tcp;
          source-port 80;
        }
        then {
          count tproxy-origin-exclude;
          accept;
        }
      }
      term client-to-origin { # Apply ‘tproxy’ topology to the included
        traffic
        from {
          source-prefix-list {
            tproxy-client-include;
          }
        }
        then { /* Perform TProxy policy */
          # TProxy policies here; the traffic is now
          # redirected to the destination
        }
      }
    }
  }
}
ECMP Configuration

The following snippet shows how to configure an MX router to support ECMP.

**NOTE:** Port numbers should not be used because this might result in asymmetric hashing, thereby preventing content from being forwarded to the correct MFC.

```plaintext
## ECMP configuration
forwarding-options { # Perform Layer 3 Hash during forwarding for ECMP
    hash-key {
        family inet {
            layer-3;
        }
    }
    enhanced-hash-key { # Do not include destination and source ports
        family inet {
            no-destination-port;
            no-source-port;
        }
    }
}
```
policy-options {
    policy-statement load-balancing-policy { # For ECMP
        then {
            load-balance per-packet;
        }
    }
}

routing-options {
    rib :tproxy.inet.0 {
        static {
            route 0.0.0.0/0 next-table inet.0;
            route 0.0.0.0/1 {
                qualified-next-hop 10.251.11.2 { # Two Equal Cost Paths (MFC1)
                    metric 5;
                }
                qualified-next-hop 10.251.11.3 { # Two Equal Cost Paths (MFC2)
                    metric 5;
                }
            }
            route 128.0.0.0/1 {
                qualified-next-hop 10.251.11.2 { # Two Equal Cost Paths (MFC1)
                    metric 5;
                }
                qualified-next-hop 10.251.11.3 { # Two Equal Cost Paths (MFC2)
                    metric 5;
                }
            }
            maximum-paths 16;
        }
        forwarding-table { # For ECMP
            export load-balancing-policy;
        }
    }
}

RPM Configuration

RPM monitors the availability of the servers using PING tests. The following snippet shows how to configure the MX router to support RPM. If the server does not respond, then the SLAX script is invoked to remove the route to that server.

```plaintext
## RPM configuration
system {
    root-authentication {
        encrypted-password "$1$qZvmmCrt$G6OLbCyQj1wPinJ1bGNk1"; # SECRET-DATA
    }
    scripts { # Needed to refer the slax script
        op {
            file Healthcheck.slax;
        }
    }
    event-options {
        policy mfc2_down {
            events PING_TEST_FAILED;
            within 10 {
```
trigger on 1;
}
attributes-match {
    PING_TEST_FAILED.test-name matches "^mfc2_status$";
    PING_TEST_FAILED.test-owner matches "^mfc$";
}
then {
    event-script Healthcheck.slax {
        arguments {
            rib :tproxy.inet.0;
            next-hop 10.251.11.9;
            action inactive;
        }
    }
}
}
policy mfc2_up {
    events PING_TEST_COMPLETED;
    within 10 {
        trigger on 1;
    }
    attributes-match {
        PING_TEST_COMPLETED.test-owner matches "^mfc$";
        PING_TEST_COMPLETED.test-name matches "^mfc2_status$";
    }
    then {
        event-script Healthcheck.slax {
            arguments {
                rib :tproxy.inet.0;
                next-hop 10.251.11.9;
                action active;
            }
        }
    }
}
}
services {
    rpm {
        probe mfc {
            test mfc2_status {
                probe-type icmp-ping;
                target address 10.251.11.9;
                probe-count 5;
                probe-interval 1;
                test-interval 1;
                thresholds {
                    total-loss 3;
                }
            }
        }
    }
}
SLAX Script

The following code snippet shows the SLAX script which removes or re-installs the route to a server.

```xml
/*
 * NAME: Healthcheck.slax
 * PURPOSE: Created to detect the state of an off-path device
 * When DOWN is detected the next-hop route of the specified RIB
 * will be deactivated, and reactivated when it comes back up
 * set the “default” gateway (in case all next-hops are down)
 * to 20 or so
 */

version 1.0;
ns junos = "http://xml.juniper.net/junos/*/junos";
ns xnm = "http://xml.juniper.net/xnm/1.1/xnm";
ns jcs = "http://xml.juniper.net/junos/commit-scripts/1.0";
import "../import/junos.xsl";

/*
 * Parms that are passed in from the configuration or cli
 */

var $arguments = {
  <argument> {
    <name> "rib";
    <description> "Routing table to adjust";
  }
  <argument> {
    <name> "next-hop";
    <description> "next-hop IP to adjust";
  }
  <argument> {
    <name> "action";
    <description> "What to do (active/inactive)";
  }
}
param $rib;
param $next-hop;
param $action;
match / {
  /*
   * Open a connection with mgd. use config private
   */

  var $con = jcs:open();
  var $open = <open-configuration> {
    <private>;
  }
  var $result = jcs:execute( $con, $open );
  if (not($con)) {
    call write-output($level = "user.error", $id = "ERROR:
```
"$, $type = "Healthcheck.slax[Error]: ", $where = "b", $message = "Not able to connect to local mgd";

*/
* XML structure for adjusting the routes
*
*/

var $xml = {
    <configuration> {
        <routing-options> {
            <rib> {
                <name> "tproxy.inet.0";
                <static> {
                    <route> {
                        <name> "0.0.0.0/1";
                        if ($action == "inactive") {
                            <qualified-next-hop
                                inactive="inactive">
                                <name> $next-hop;
                            </name>
                        } else {
                            <qualified-next-hop
                                active="active">
                                <name> $next-hop;
                            </name>
                        }
                    }
                }<route> {
                    <name> "128.0.0.0/1";
                    if ($action == "inactive") {
                        <qualified-next-hop
                            inactive="inactive">
                            <name> $next-hop;
                        </name>
                    } else {
                        <qualified-next-hop
                            active="active">
                            <name> $next-hop;
                        </name>
                    }
                }
            }<route> {
            }
        }<static> {
    }
</configuration> {"}
Summary

Server load balancers improve scaling and availability by directing requests to the appropriate server, which can support the request. SLBs come with a wide range of features and functions, with round-robin load balancing being commonly deployed.

For load balancing within the data center, the MX Series router with Trio-based MPCs provides an integrated capability, which can eliminate the need for additional third-party hardware. Implementing Equal Cost Multi-Path (ECMP) allows the MX to forward requests to multiple servers. Implementing Real-time Performance Monitoring (RPM) provides a means to monitor the availability of each server, and; creating SLAX scripts enables the router to bypass failed servers.

About Juniper Networks

Juniper Networks is in the business of network innovation. From devices to data centers, from consumers to cloud providers, Juniper Networks delivers the software, silicon and systems that transform the experience and economics of networking. The company serves customers and partners worldwide. Additional information can be found at www.juniper.net.