Rapid Automated Indication of Link-Loss

Fast recovery of failures is becoming a hot topic of discussion for many of today's Big Data applications such as Hadoop, HBase, Cassandra, MongoDB, MySQL, MemcacheD and other data intensive distributed applications. Many of these applications do not use sophisticated and integrated failover mechanisms and as a result utilize heartbeat timeouts or general communication stack timers to detect and recover from node failures.

Arista's Rapid Automated Indication of Link-Loss (RAIL) feature leverages the switch intelligence to proxy send TCP Session Resets (RST) or ICMP Destination Unreachable messages (in the case of any non-TCP packet) for nodes that have failed or have lost link connectivity to the switch. This unique Arista EOS feature can help facilitate dramatically faster failover times for distributed datacenter applications, which are either batch or stream oriented and are sensitive to node failures.

TCP Session Reset

The majority of data center applications today are based on connection oriented TCP protocol communication due to its integrated data delivery guarantees, loss recovery and congestion management mechanisms. TCP relieves the application of these functions and consequently obscures them. The TCP protocol specification includes a method to immediately terminate a TCP connection when it is no longer needed or valid. When a host receives a TCP session reset (RST), it stops sending and receiving data for that connection and closes the port. This has two main benefits: it frees up kernel resources on the host and allows the application to quickly recover from failures by establishing new TCP connections to alternate systems if they exist. If a TCP RST is not received, the session remains open and in a stalled state waiting for either TCP or application timeouts to expire, increasing failover times of the application. Typical TCP session timeouts are in the 13 to 30 minute range (in the absence of an application-specified TCP timeout), as per RFC1122. With RAIL enabled, **failure recovery times go from minutes down to just a few milliseconds or less.**

TCP Session Reset

Some datacenter applications rely on connectionless or UDP based communication between systems. UDP is often employed when latency, efficient reception by multiple systems and bandwidth are of primary concern. However, UDP does not offer any data protection or bandwidth management mechanisms that are associated with TCP communications. Consequently, the application is responsible for these functions. In the case of a failure, an ICMP Destination Unreachable message can be generated to the sending host to allow the application to choose another target system to facilitate fast recovery.

Arista EOS RAIL (Rapid Automated Indication of Link-Loss)

To support the proxy sending of either TCP or ICMP Destination Unreachable messages for directly attached servers, the Arista switch is configured to monitor a subnet or group of subnets. For this the switch is set as the default gateway for each server that is directly connected. If an interface (configured to be monitored) transitions to the down state:

- An entry is made on the switch supervisor (CPU) to reject packets to this destination (discovered when the link was in the up state)
- An entry is programed into the switching FIB (Forwarding Information Base) table that redirects frames with the associated MAC address to the supervisor (CPU)
- When the supervisor receives a matching packet it will send back to the originating host(s) either a TCP RST or an ICMP unreachable message for both UDP and ICMP packets
- The switch continues to operate in the "proxying" state for a user configured time period
- Once the sending host receives the RST or ICMP unreachable the application can take immediate action to failover instead of waiting for lengthy TCP or application timers to expire.

Once a server is discovered, the four states RAIL supports are Up, Down, Proxying and Inactive:

Table 1: RAIL Server States

RAIL Server State	Description
Up	Server is up
Proxying	Link down was detected and server is being proxied
Down	Link down was detected and proxy is not enabled
Inactive	Server goes to Inactive state if: • The MAC entry or ARP entry of the server gets deleted • The RAIL configured interface goes down and the proxy/down lifetime is complete.

RAIL Configuration

Prior to configuration of the RAIL feature ensure that the switch is either an Arista 7050 or 7150 Series Datacenter switch and it is configured as the default gateway for the subnet being monitored. This ensures that IP addresses can be mapped to MAC addresses and physical ports.

Note that RAIL in conjunction with VMs in not currently supported.

From the switch CLI:

```
switch(config)# monitor server-failure
switch(config-server-failure)# no shutdown
switch(config-server-failure)# proxy
switch(config-server-failure)# proxy lifetime <1-10080> *How long to proxy
switch(config-server-failure)# network <subnet/mask>
switch(config-server-failure)# interface <name> *Can be a single port or port-channel
```

Feature verification and show commands:

```
switch#show monitor server-failure
Server-failure monitor is enabled.
Proxy service: enabled
Proxy lifetime: 10 minutes
Networks being monitored: 3
  10.1.0.0/16       : 3 servers
  132.23.23.0/24    : 1 servers
  44.11.11.0/24     : 0 servers

switch#show monitor server-failure ?
 history  History of failed servers
 servers  Servers being monitored that are active
 >        Redirect output to URL
 >>       Append redirected output to URL
 |        Output modifiers
 <cr>
```

Check the recent history of server failures:

```
switch# show monitor server-failure history
Total server failures: 4
Server IP     Server MAC              Interface             Last Failed
---------     -----------------       -----------           -------------------
10.1.67.92    01:22:ab:cd:ee:ff       Ethernet17            2013-02-02 11:26:22
44.11.11.7    ad:3e:5f:dd:64:cf       Ethernet23            2013-02-10 00:07:56
10.1.1.1      01:22:df:42:78:cd       Port-Channel6         2013-02-09 19:36:09
10.1.8.13     01:33:df:ee:39:91       Port-Channel5         2013-02-10 00:03:39
```

Check the status of servers being monitored that are active:

```
switch#show monitor server-failure servers
Active servers: 4
Server IP     Server MAC          Interface          State          Last Failed
----------    -----------------   --------------     ---------      -----------
44.11.11.7    ad:3e:5f:dd:64:cf   Ethernet23         down           0:03:21 ago
10.1.1.1      01:22:df:42:78:cd   Port-Channel6      up             4:35:08 ago
10.1.8.13     01:33:df:ee:39:91   Port-Channel5      proxying       0:07:38 ago
132.23.23.1   00:11:aa:bb:32:ad   Ethernet1          up             never
```

Collect detailed information on server-failure servers:

```
switch#show monitor server-failure servers ?
 A.B.C.D   Show server with IP address
 all       Show all servers
inactive  Inactive servers
 proxying  Servers being proxied
 >         Redirect output to URL
 >>        Append redirected output to URL
 |         Output modifiers
 <cr>
```

Show the detailed state of a single server:

```
switch#show monitor server-failure servers 44.11.11.7
Server information:
 Server IP Address              : 44.11.11.7
 MAC Address                    : ad:3e:5f:dd:64:cf
 Current state                  : down
 Interface                      : Ethernet23
 Last Discovered                : 2013-01-06 06:47:39
 Last Failed                    : 2013-02-10 00:07:56
 Last Proxied                   : 2013-02-10 00:08:33
 Last Inactive                  : 2013-02-09 23:52:21
 Number of times failed         : 3
 Number of times proxied        : 1
 Number of times inactive       : 18

switch#show monitor server-failure servers all
Total servers monitored: 5

Server IP     Server MAC          Interface        State        Last Failed
----------    -----------------   ------------     ---------    -----------
10.1.67.92    01:22:ab:cd:ee:ff   Ethernet17       inactive     7 days, 12:47:48 ago
44.11.11.7    ad:3e:5f:dd:64:cf   Ethernet23       down         0:06:14 ago
10.1.1.1      01:22:df:42:78:cd   Port-Channel6    up           4:38:01 ago
10.1.8.13     01:33:df:ee:39:91   Port-Channel5    proxying     0:10:31 ago
132.23.23.1   00:11:aa:bb:32:ad   Ethernet1        up           never
```

Show servers that are currently inactive:

```
switch#show monitor server-failure servers inactive
Inactive servers: 1

Server IP     Server MAC          Interface        State        Last Failed
----------    -----------------   ------------     --------     -------------
10.1.67.92    01:22:ab:cd:ee:ff   Ethernet17       inactive     7 days, 12:48:06 ago
```

Show servers currently being proxied:

```
switch#show monitor server-failure servers proxying
Active failed servers being proxied: 1

Server IP     Server MAC          Interface          State         Last Failed
----------    -----------------   --------------     --------      ------------
10.1.8.13     01:33:df:ee:39:91   Port-Channel5      proxying      0:11:08 ago
```

RAIL Deployment Scenario with Apache HBase

HBase is an open source, massively scalable, NoSQL distributed database based on Google's BigTable that is commonly used in conjunction with Hadoop's HDFS (Hadoop Distributed File System). HDFS itself is a "logical" filesystem that runs on top of an operating systems native filesystem (such as ext4). The largest HBase clusters can scale to over 1000 servers with >1PB of storage. HBase is very latency sensitive as it offers extremely fast writes (1-3ms, 1-10k writes/sec/node) and reads (<1-3ms in memory cache, 10-30ms on disk, 10-40k reads/sec/node). The HBase architecture consists of Clients, ZooKeeper, HBase Master, RegionServers and HDFS data nodes. It is common to run RegionServers on all HDFS data nodes. The architecture is logically connected as shown below:

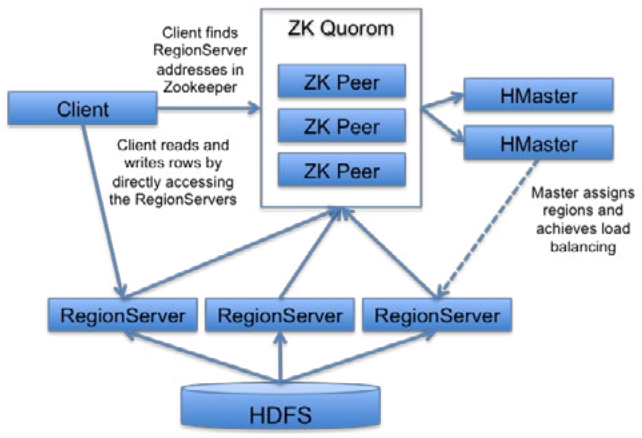
ZK Quorom
Client finds RegionServer addresses in Zookeeper
Client
ZK Peer
ZK Peer
ZK Peer
HMaster
HMaster
Client reads and writes rows by directly accessing the RegionServers
Master assigns regions and achieves load balancing
RegionServer
RegionServer
RegionServer
HDFS

Figure 1: Apache HBase – Image courtesy of Cloudera

HBase utilizes a Write Ahead Log (WAL) and considers a write to the database complete when the operation is successfully logged to the WAL in the form of an HDFS SequenceFile. This allows data to be recovered by an alternate RegionServer in the event of a failure. The data is then copied to a MemStore (in system RAM) and eventually (once a threshold is met) flushed to an HFile for long-term storage and access. This is important to understand from a recovery perspective.

The HBase read/write operations follow the same basic pattern of 4 independent TCP connections:

1. The Client connects to ZooKeeper quorum to determine the "-ROOT-" RegionServer
2. The Client connects to the RegionServer and queries for the correct .META. server
3. The Client connects to the .META RegionServer to find the correct RegionServer for the table of interest
4. The Client connects to and executes the PUT/GET on the targeted Region

* Each RegionServer utilizes the HDFS Client to write to the filesystem through TCP pipelining and offers read accessibility through its block replication/redundancy architecture *

It's important to note that all of these connections are TCP based utilizing custom HBase RPC calls and HDFS read/write operations and are therefore subject to stalled TCP connections during node failures. These stalled sessions can cause serious access latency issues with HBase and associated client applications. This is where Arista's RAIL functionality can facilitate fast failover and recovery times. In the case of a node failing while it is in the HDFS pipelining process, the system will stall and wait for timers to expire; this typically takes 30-60 seconds. If the RAIL feature is enabled, a TCP RST will be sent for all known (by session packets received by the switch) connections that were on the now dead node. **This causes a timely reaction by the connecting node and forces an immediate fail-fast.**

Arista RAIL in Action

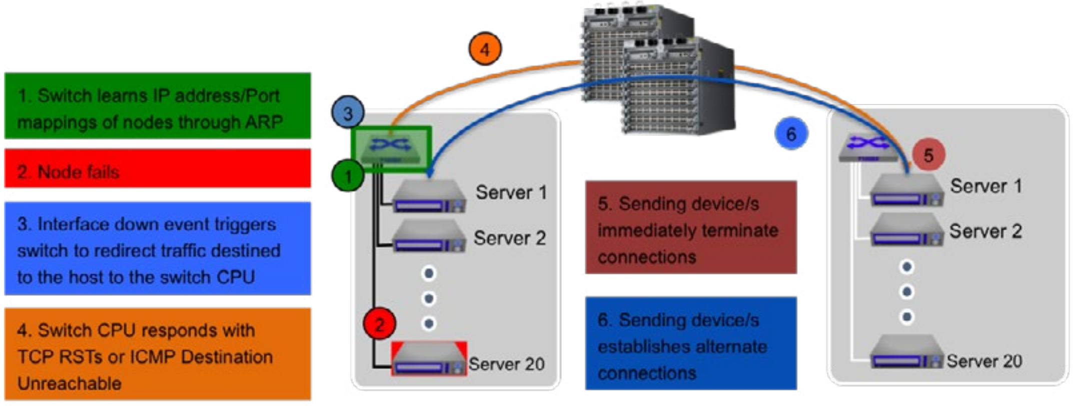
1. Switch learns IP address/Port mappings of nodes through ARP
2. Node fails
3. Interface down event triggers switch to redirect traffic destined to the host to the switch CPU
4. Switch CPU responds with TCP RSTs or ICMP Destination Unreachable
5. Sending device/s immediately terminate connections
6. Sending device/s establishes alternate connections
Server 1
Server 2
Server 20
Server 1
Server 2
Server 20

Figure 2: Arista RAIL in action

The Arista RAIL feature can be applied generally in the datacenter on systems that are clustered, provide redundancy options and/or are hosting time sensitive applications to meet the high demands found in today's datacenters.

Conclusion

The Arista RAIL feature accelerates application recovery by proxying the server connection and closing connections. This feature saves time and improves job performance for Hadoop and distributed computer intensive applications.

Santa Clara—Corporate Headquarters
5453 Great America Parkway,
Santa Clara, CA 95054

Phone: +1-408-547-5500
Fax: +1-408-538-8920
Email: info@arista.com

Ireland—International Headquarters
3130 Atlantic Avenue
Westpark Business Campus
Shannon, Co. Clare
Ireland

Vancouver—R&D Office
9200 Glenlyon Pkwy, Unit 300
Burnaby, British Columbia
Canada V5J 5J8

San Francisco—R&D and Sales Office
1390 Market Street, Suite 800
San Francisco, CA 94102

India—R&D Office
Global Tech Park, Tower A & B, 11th Floor
Marathahalli Outer Ring Road
Devarabeesanahalli Village, Varthur Hobli
Bangalore, India 560103

Singapore—APAC Administrative Office
9 Temasek Boulevard
#29-01, Suntec Tower Two
Singapore 038989

Nashua—R&D Office
10 Tara Boulevard
Nashua, NH 03062