SciPass Design Document

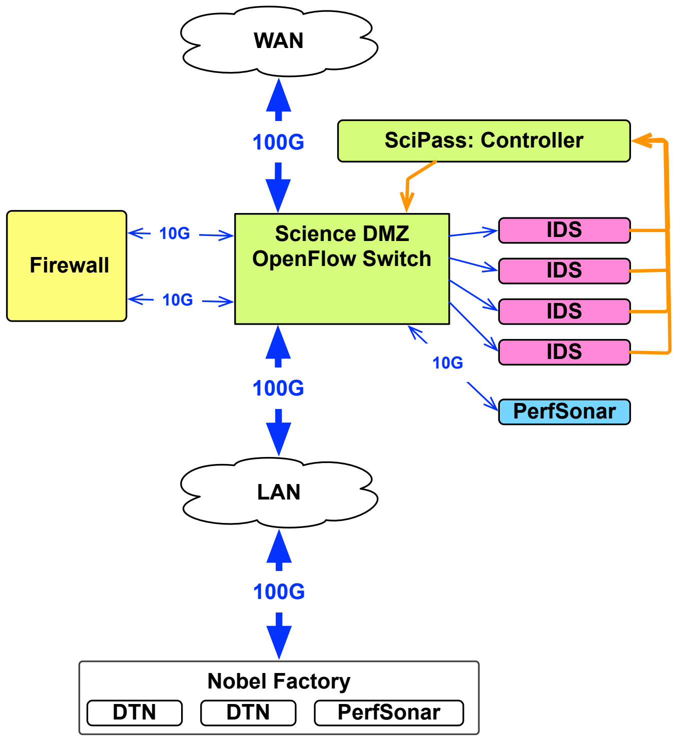
Edward Balas
May 2014

Overview:

SciPass is an OpenFlow based Science DMZ and IDS load balancer application that is deployed with an OpenFlow Switch as network appliance. It can operate as a simple IDS load balancer or as an inline firewall bypass with integrated IDS load balancing. SciPass provides APIs for integration with IDS systems such that the load balancer can react to sensor load and the sensor can instruct the balancer to send or not sent certain flows through the sensor. The system will contain 2 load balancer modules, one relies only on OpenFlow, the other uses a combination of sFlow and OpenFlow.

System Design

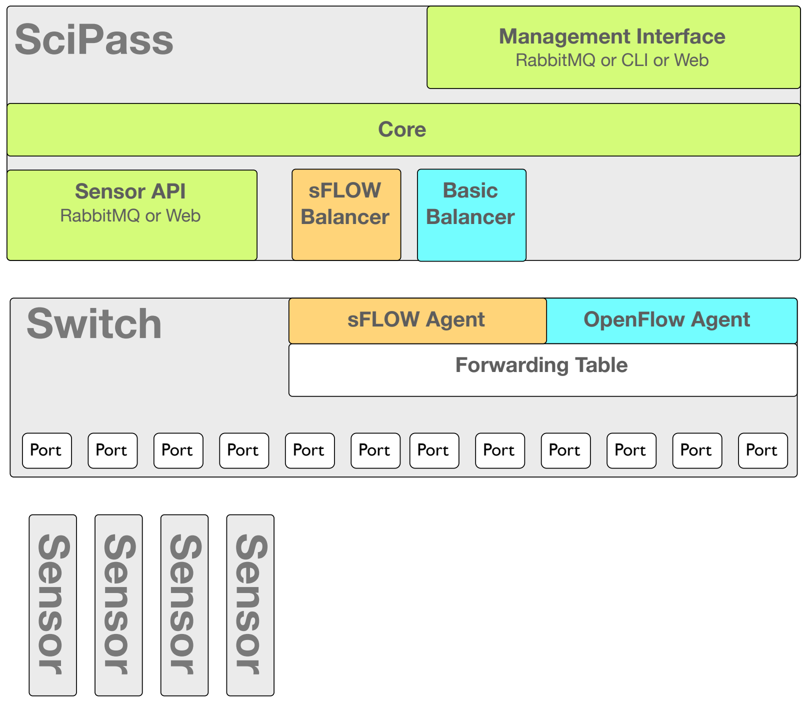
The system consists of an OpenFlow enabled switch, a cluster of IDS sensors, a SciPass controller server and a PerfSonar performance assurance server.



Component Design

The SciPass system will be designed around a set of separate processes that communicate internally over a Rabbit MQ message bus.

The SciPass core process is responsible for managing the flows on the switch and providing methods to get and set relevant data. At least one Balancer must be run , but the system is designed such that balancers are module and the switch from one balancer to another is a run time decision that does not require software releases.



Basic OpenFlow Load Balancer Module

Description:

Based purely on OpenFlow, this module discovers which portions of the configured local prefix range are causing unbalanced sensor load and by de-aggregating hot prefixes and shifting them to unloaded sensors

Approach:

0. The sensors will periodically report estimated load using a provided web

service. Values are in the range of 0 to 100 with 100 indicating max possible load.

1. Take an input prefix such as 10.0.13.0/24 and create a number of subnets. The number will be based on the number of sensors * a configurable multiplier. To subnet we will likely need to find the smallest power of 2 value greater than the sensor number * multiplier. For example assume our prefix 10.0.1.0/24, and we have 8 sensors and a multiplier of 1. We then need to 8 /27 subnets.

2. Once we have our subnets, we add 16 rules into the switch forwarding table, 2 for each subnet with 1 rule for each direction of forwarding.

3. After running for a period of time, the system checks the bytes match on the corresponding OpenFlow rules. The module then combines prefix load with the sensor load metric to get a table that looks like this

Sensor	Prefix	Gbytes	Load
1	10.0.13.0/27	9	90
2	10.0.13.32/27	1	10
3	10.0.13.64/27	1	10
4	10.0.13.96/27	1	10
5	10.0.13.128/27	1	10
6	10.0.13.160/27	1	10
7	10.0.13.192/27	0	0
8	10.0.13.224/27	1	10

This table is the central data structure used by the module. The module then evaluates each sensor's load and determines that sensor 1 is too high.

4. The module checks the number of prefixes associated with the sensor 1, there is 1 and it is responsible for all traffic, only option is to split the subnet by taking 10.0.13.0/27 and breaking it into 10.0.13.0/28 and 10.0.13.16/28

5. The module then replaces the 2 forwarding rules for the /27 with 4 forwarding rules for both /28s.

6. The module waits for a configured period of time, after which it checks the byte per sec on each of the new /28s. In addition it creates a per prefix load estimate for each. This is based on load/total traffic on interface / proportion of total traffic associated with a given prefix. This approach is rather simplistic and may not be adequate in the long term.

7. Finally, the module checks for existing sensor that has available load and bandwidth, in this this case sensor 7 is idle, 10.0.13.0/28 is assigned to sensor 7

10.0.13.16/28 remains on sensor 1.

After doing this we estimate that sensor 1 will drop to 4.5 gig and load of 45, and that sensor 7 will come up to 4.5 and load 45.

Sensor	Prefix	Gbytes	Load
1	10.0.13.0/28	4.5	45
2	10.0.13.32/27	1	10
3	10.0.13.64/27	1	10
4	10.0.13.96/27	1	10
5	10.0.13.128/27	1	10
6	10.0.13.160/27	1	10
7	10.0.13.192/27	0	45
8	10.0.13.224/27	1	10
7	10.0.13.16/28	4.5	45

Now clearly traffic is not something that is so evenly distributed. It is uneven and dynamic. The load balancer will make one change per tuning cycle then observer traffic and then react, continually repeating

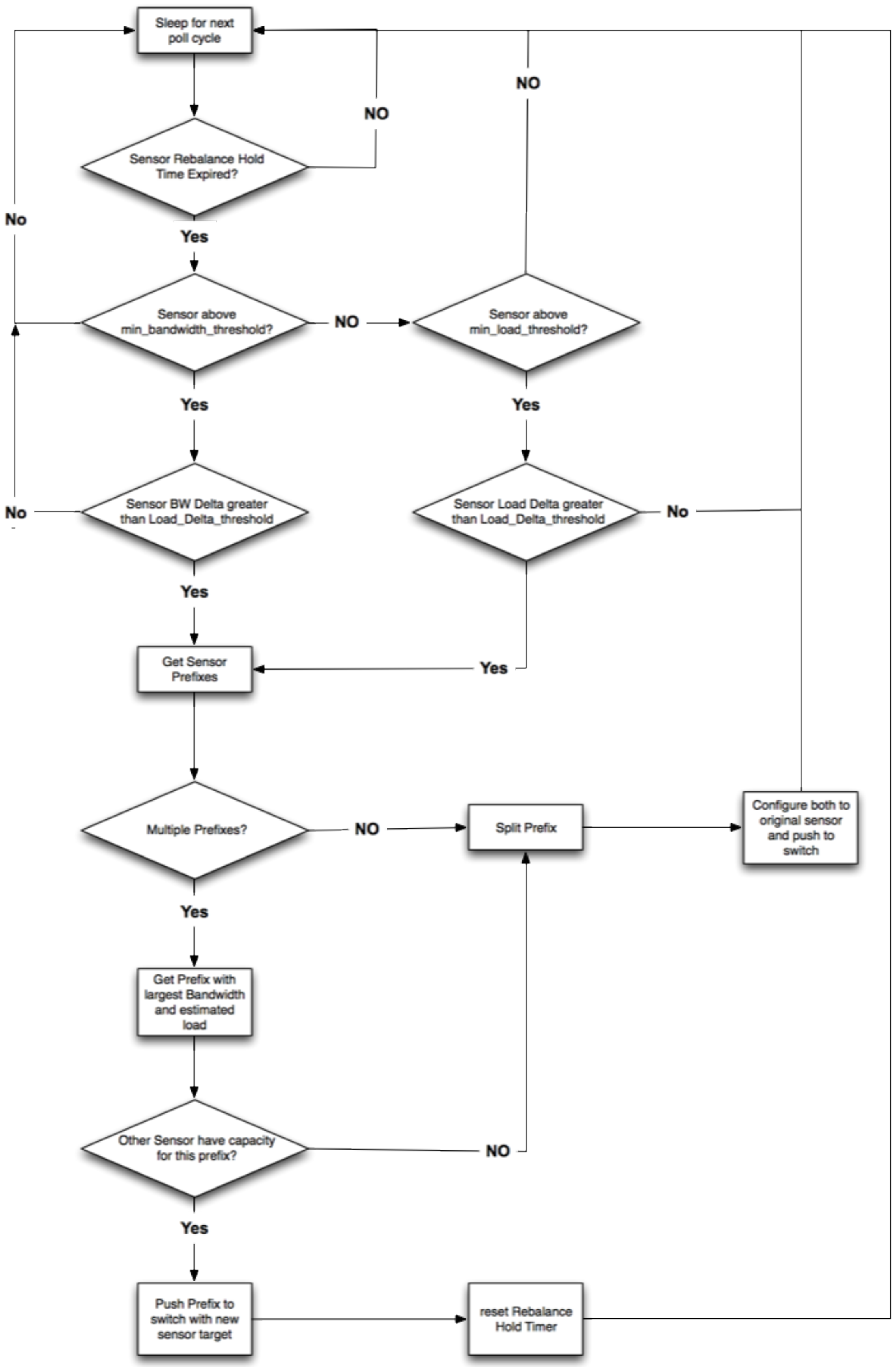

Sleep for next poll cycle
Sensor Rebalance Hold Time Expired?
NO
Yes
NO
Sensor above min_bandwidth_threshold?
NO
Sensor above min_load_threshold?
No
Yes
Yes
Sensor BW Delta greater than Load_Delta_threshold
No
Sensor Load Delta greater than Load_Delta_threshold
No
Yes
Get Sensor Prefixes
Yes
Multiple Prefixes?
NO
Split Prefix
Configure both to original sensor and push to switch
Yes
Get Prefix with largest Bandwidth and estimated load
Other Sensor have capacity for this prefix?
NO
Yes
Push Prefix to switch with new sensor target
reset Rebalance Hold Timer

Figure 1 Flow Chart of balancing Algorithm

Balancer Config Options:

1. sensor_bw_alarm

volume of traffic at which we need to alarm because we are worried we will impact forwarding or IDS functioning

2. sensor_load_alarm

loat metric at which we need to alarm because the sensor may be resource bound and unable to check the input traffic

3. sensor_load_min_threshold

This represents the lowest load on a sensor where we will consider tuning any of the prefixes on a sensor. For example if we set the sensor_load_min_threshold to 60, and one sensor is at 50 and all others are at 1 we will not rebalance based on load values.

4. sensor_bandwidth_min_threshold

This represents the lowest load for a sensor where we will consider rebalancing associated prefixes. For example if we set the sensor_bandwidth_min_threshold to 50 and one sensor is at 30 and the others are at 1 gbps, then rebalancing will not happen based on bandwidth values.

If either the sensor_load > sensor_laod_min_threshold OR sensor_bw > sensor_bw_min_threshold, then we consider balancing.

5. rebalance_hold time

Minimum amount of time to wait between balacing actions. This controls the rate at which things can adjust.

6. rebalance_max_prefix_length

Maximum prefix length beyond which the balancer will not spit, defaults to /32 for v4

6. sensor_load_delta_threshold

7. sensor bandwidth_load_delta_threshold

API

For troubleshooting we need methods to examine the central table data structure to keep an eye on how well things are balanced.

Unit tests

1. /24 with 8 sensors and all traffic going to 1 host.

expected behavior:

system will split into increasingly more specifc prefixes, when it goes to split prefix at some point it will hit the max prefix length and give up.

2. /24 with 8 sensors and all traffic going to 2 hosts each host generating 1 gig of traffic

expected behavior:

system pre splits by 8 so the 2 hosts are already handled by different sensors, if 1 gig does not exceed balancing hreshold nothing further happens.

3. /24 with 8 sensors and 8 hosts all assigned squential addresses and each representing 10gbps of traffic

system presplits by 8 /27 but all hosts are in the same /27. A sort time later Sensor 1 exceeds bandwidth and load threshold so we alarm, then we split the /27 into a /28 and update the rules but not changing which sensor. A poll cycle later we alarm again, then grab one of the /28s, which has the largest load. we see if we can move this to another sensor but find it wont fit, so we again split. We repeat this process until we have broken down the loaded /28 down to a set of /32s and move those to each of the sensors. In the end every sensor is above reballance threshold but we end up bellow the sensor_load and bw_delta_threshold so we do no further balancing

Need More Tests

Core Module

At the heart of this module is an OpenFlow controller and an API for other components to use get sensor health, flow statistics and configuration information. In addition the API provides methods to interact with the switch forwarding table.

API

get_sensors()
    returns the list of configured sensors and their current value

    returns list of sensors, with the following fields:
     -sensor_id
     -sensor_name
     -associated switch_port
     -heartbeat_timestamp
    - load_metric

add_sensor()
    -input
        -name
        -switch_port

    - returns sensor_id

update_sensor()
    - input
        -sensor_id
        - name
        - switch_port

    - returns 1 for success, - for error

replace_sensor()
    this takes two sensor Ids and migrates all rules from the old_sensor to the new
    if successfull the other remains but has no associated rules.

    - input
        - old_sensor_id
        - new_sensor_id

- returns 1 for success, - for error

delete_sensor()

deletes a sensor, will error if there are associated rules

- input
    - sensor_id

- return 1 on succss, - on error

report_sensor_health()

This method is used to update the health of a sensor. It is expected that load balancer modules will monitor the heartbeat. If a sensor fails to report in the configured interval, the balancer should try to migrate the rules off to other sensors.

- input

----

get_rules();

- gets all rules in the system returning the full set of options

- return list of rules, for each
    - rule_id
    - action
    - match
    - sensor
    - allow_tuning

    - bps
    - age

add_rule();

-input
    -action
    -match
    -sensor
    -allow_tuning

replace_rule();

takes in existing rule id and an array of new rules to replace it. This is used to split rules as atomically as possible. The caller is responsible for managing which parts of the flow space are covered by the system.

- input
    - old_rule_id
    - new_rules
        - action
        - match
        - sensor
        - allow_tuning

- returns 1 on success, - on error

update_rule();

- allows one to edit a rule's action or sensor directive, match are constant once created

- input
    - rule_id
    - action
    - sensor
    - allow_tuning

delete_rule();

- removes a rule

- input
    - rule_id

- returns 1 on success, - on error

Switch Requirements

1. must be able to match port IP_SRC, IP_DST, IP_Proto and Port.
2. must be able to support variable length IP_SRC and IP_DST Masks using the wildcard field.
3. must support multiple output actions in particular we need the ability to receive a packet in on one port, and sent it out 2 ports.
4. Should be able to strip vlan and MPLS tags while sending out multiple output ports.

Development Plan

Phase 1: Flow Scale functional replacement

Development will first focus on providing a viable replacement of FlowScale. This will require the development of the following:

1. core module
2. basic openflow balancer
3. APIs need for sensors to express load and whitelist requests
4. CLI utilities for managing system
5. Test functionality in lab

Phase 2: Inline bypass / Science DMZ

The second phase will build upon the functional load balancer by adding the following:

1. Add whitelist with bypass mode
2. Develop IDS rules and processing modules to parse GridFTP and determine data channels and pass this information back to SciPass
3. Test functionality in lab