SciPass version 1.0

GlobalNOC Product Requirements Document
2/10/2014

Abstract:

SciPass is an an OpenFlow Science DMZ solution which provides a dynamic and tunable Science Bypass to corporate firewalls and deep packet inspection systems to allow for sustained high bandwidth science data transfers. The product provides 3 main features:

1. Integrated Intrusion Detection System load balancing similar to that found in FlowScale
2. API for for IDS feedback, tuning
3. A means to control forwarding based on feedback

The objective of this release is to provide a functional replacement for FlowScale, having improved robustness and enhanced control of network forwarding. The system will support running as either as a simple load balancer or as load balancer and DMZ switch.

Anticipated Uses:

IU UISO Engineers
R&E engineers operating DMZ networks

Test User Contacts

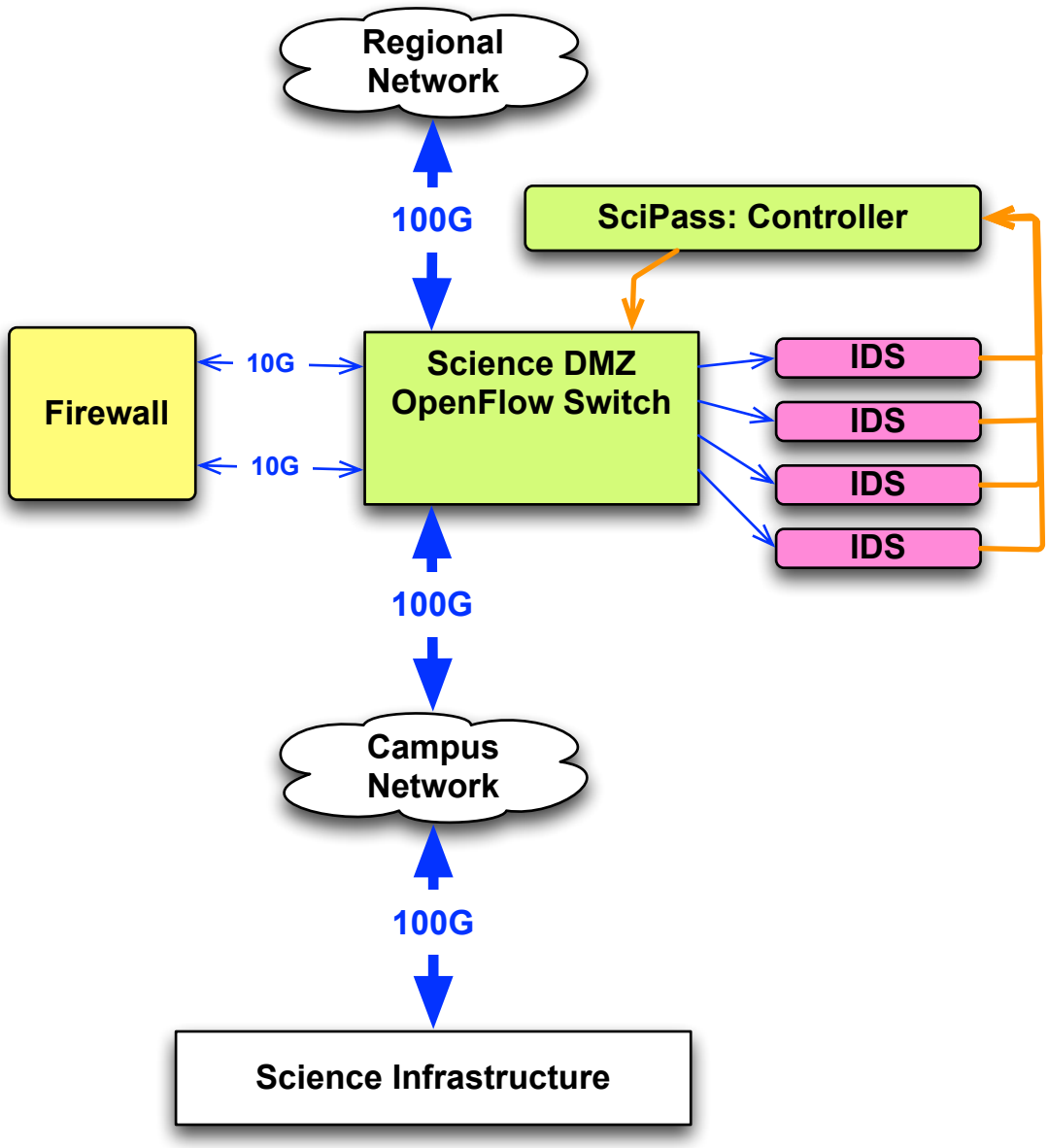
Keith Lehigh

Decision Makers

Keith Lehigh
Edward Balas

Conceptual Model

Conceptually, the idea is to use the same OpenFlow switch as an integrated Science DMZ providing IDS load balancing and Corporate Firewall Bypass with the SciPass Controller coordinate all operations on the DMZ switch.



Vision/Scope/Problem Statement:

Today, FlowScale acts as a stateless, policy based load balancer. The current code operates in production prototype mode and has known deficiencies. The objective for this release is to improve robustness and to ensure it operates effectively when load balancing between multiple Bro sensors.

The current version has a rather simplistic balancing mechanism that proactively assigns subnets to sensors, one defines a number of prefixes and the system de-aggregates these into subnets and sends traffic for each more specific subnet to a different sensor in a round robin manner. It has no automatic means to dynamically

rebalance traffic, or for the sensors or lacks the ability for a sensor or administrator to say that a given type of traffic is not interesting.

As second problem we would like to address relates to Campus DMZ networks and the performance limits of corporate firewalls and their inability to support 100g flows. To address this limit the SciPass system will include the ability to programmatically define which flows will bypass the Firewall, in effect providing a means to white list certain flows and avoid consuming resources on the firewall.

Intended Use Case:

For this release, we are considering two use cases: IDS clusters and Firewall Bypass

1. supporting a cluster of IDS sensors that are processing data from a single input Ethernet. We want to build a system capable of taking an input of 1 or more 100G Ethernet interfaces fanning this out between N sensors. To better support the IDS array use case we need to provide a feedback mechanism between sensors and the FlowScale so that the system as a whole can adapt to shifting traffic dynamics.
2. The second use case in mind relates to GridFTP or other transfers, the thought is that the initial portion of the transaction could traverse the firewall and be examined by the IDS. If the IDS believes the flow to be trustworthy, then it could ask SciPass to insert a whitelist / dynamic bypass thus allowing the transfer to use up to line rate on the switch.

Both use cases assume a campus edge deployment of an OpenFlow Switch, for use case 1, the switch can be installed outside the forwarding path, simply receiving frames from a SPAN port. For the second use case the OpenFlow Switch would be in the critical forwarding path and can be used to affect packet forwarding.

General Requirements:

1. Must support a high availability configuration with synchronized state between so that if one control server dies the other can pick up automatically without loosing policy or forwarding rules on the associated switch
2. Must ensure policy data and current state is persistently stored for restoration on software restart.
3. Must have the ability for a sensor to define certain traffic as uninteresting
4. Must be designed for long-lived production operation, including automatically re-establishing connections to databases. Having proper init scripts and reloading config on HUP.
5. Must provide concise human readable logs, avoiding stack traces where possible.

6. Must support multiple input and output ports, such that we can have say 3 10gbps ports where we receive monitoring traffic and 6 output ports where we send traffic to sensors (assuming in such a scenario and even traffic distribution and a 5Gbps capability for each sensor)
7. Must have a CLI/Craft interface for management as its primary interface.

Balancing Requirements

Much of the work in this version relates to the need for more dynamic balancing approaches. These are as follows

Whitelists: avoid uninteresting traffic

Often a sensor knows that certain types of traffic are uninteresting, this could be traffic to or from a particular IP, a specific tuple of src/dst IP, proto and ports or just traffic to or from a particular port. In ether case, a web services interface must exist so that a sensor can express this to FlowVisor and it should act such that:

1. Whitelist rules should be evaluated before rules that balance traffic, thus they must have a higher priority in the switch
2. Sensors must be able to set and retrieve and remove whitelist entries
3. When adding an entry, the sensor may set an inactive and hard timeout (subject to the capabilities of the switch) these timeouts will default to never.

Traffic Adaptive Balancing

Static balancing by IP tuple does not work well because associated traffic loads are not consistent across IP and computational load required for deep packet inspection can vary by traffic type or pattern even when the volumes are equal. Alternate Balancing mechanisms should be created such that

1. The system can model the current load on each sensor and uses this to allocate traffic to sensors in relative inverse proportion to their load.
2. The system can programmatically determine sensor load and trigger a rebalance if the change in load is sufficiently high.
3. The system can model traffic volume associated with rules it has inserted on the switch.
4. The system can model traffic volumes associated with interesting traffic at granularities greater than expressed in OpenFlow rules using other mechanisms such as sFlow.
5. The system can create balancing logic based on unequal sized rules. For example imagine we are looking at 10.13/16 and in current model we break this up into 256 /24s and round robin these to 8 sensors. If a single /24 is responsible for 99% of the traffic we are going to have a problem. Alternate approaches are desired where the dominant / 24 is split up and balanced between the 8 sensors along with the rest of the traffic.

6. The system can create more specific balancing rules. If the distribution of traffic is such that a handful of hosts is responsible for the majority of the load it is desirable to have a model where the majority of all traffic is split across sensors and then rules are added at a higher priority which then split out more specific subsets of traffic for rebalancing.

Sensor Health Feedback

An API must be provided so that sensors can provide feedback on its health and load follows:

1. Each sensor must have a unique id that corresponds to a port
2. sensor can take itself offline via API, when it does this system should re balance traffic to other sensors
3. Sensor must provide a means for the controller to query or for the sensor to push its load on a periodic (configurable frequency on order of 10 seconds). This load value is expressed as an integer from 0 to 100. The load balancer will then weight it's balancing based on this load estimate.

Programmatic API

Flowscale is one component of many that must work in a coordinate manner. All policy and load/performance data must be available via programmatic interface for integration into a larger system. Possible use cases for this are a system that spins up additional sensors on demand and then rebalances to adjust to cyclical load.

Forwarding Control Requirements

This system will be able to control forwarding within the OpenFlow switch to provide the following capabilities

1. For specific flows across the switch defined as a set of src and dst header values be able to forward traffic directly between the uplink and downlink port bypassing the firewall uplink and downlink ports entirely. These rules are to be configured as fixed duration, infinite or based on idle timeout.
2. Provide a means to blackhole traffic matching a defined set of src and dest header values for a fixed duration, infinite or based on idle timeout duration.
3. Must support multiple uplink ports, for instance it could be the case that all traffic but science traffic traverses a 10g link but then there is a dedicated 100g for just specific science flows.
4. Must provide a simple interface to manually control forwarding.

Programmatic API

As with the Flowscale features, the forwarding control features need to be programmatically accessible via API. In particular we need to ensure that an

efficient feedback mechanism with the IDS allows that to control which flows can bypass the firewall or blackhole traffic.

Technical Requirements:

1. Must be based on a supported controller with a roadmap for OpenFlow 1.3 support
2. Must include RPM and adhere to GRNOC coding and package standards
3. APIs must have remove bindings using web services
4. APIs should have rabbitMQ or other message passing bindings for high performance interface.