SDN Project Progress and Specifications:

SDN High-level Explanation …

OVS Configuration Information …

POX controller information …

Priority Switch Documentation …

Crysis Eye Application Suite Introduction ...

Crysis Eye Mobile Application Documentation …

Go Video Server Overlay Documentation …

Crysis Eye Incident Commander Display Documentation ...

Hardware Needs ...

Next Steps ...

Software-Defined Networking - High Level Explanation

Software-defined networking is a new networking paradigm, focused around the idea of making networks easier to manage and manipulate. The idea behind it is to separate the data in packets from their control information, allowing the latter to be handled by a secondary network which communicates with a controller, the decision-making focus of the network.



This means that an SDN requires two network to run simultaneously which are separated save for their overlapping connections at switches. The first, which handles transmission of data across the network, behaves much like a typical network does.

However, the switches throughout this system have a key difference from normal switches; rather than use routing tables to dictate routing logic, they store dynamically adjustable and more verbose versions of routing tables known as flow tables.

These flow tables are not robust, typically having several holes for unknown routing mappings across the network. Whenever an unknown mapping of source to destination values if found within a packet, the switch notifies the controller that it needs a way to handle the packet by sending a packet routing request that contains all of the packet’s header information. The controller upon receiving it has many options; it can either install a route permanently or temporarily (it will expire after a set amount of time elapses), it can instruct the switch to flood or drop the packet, or it can simply offer routing instructions for the received packet without installing a route in the switch’s flow table.

Through these methods, the controller can dynamically adjust the behavior of the network, making routing decisions on the fly using any components of incoming/outgoing packets’ header information, or set static rules throughout the network to ensure low latency throughout the network. It is the ability for the controller to utilize both of these routing methods simultaneously with no need to pull or disable any switches along the network that makes an SDN such a strongly desired networking foundation.

OVS Configuration Information

In order to build a physical SDN testbed, we opted to purchase and configure Raspberry Pi’s as openflow virtual switches (OVS). The OVS software has already been installed on the pi’s we currently have, but information on the configuration process can if needed be found at:

<https://sdntestbed.wordpress.com/2015/07/03/install-open-vswitch-on-raspberry-pi/>

Once configured, the ovs software may need to be adjusted in order for it to properly function within desired topologies. Because of how the OVS software works, it will completely consume all information transferred to it through ports assigned to it, meaning that the pi itself can not communicate across any ethernet ports assigned to an OVS bridge. As such, it is important to configure any running bridges accordingly based on your topology.

The commands to do this are as follows:

* ‘Sudo ovs-vsctl list-bridges’ - lists current bridges
* ‘Sudo ovs-vsctl add-bridge [name]’ - add bridge named [name]
* ‘Sudo ovs-vsctl del-bridge [name]’ - delete bridge named [name]
* ‘Sudo ovs-vsctl list-ports [bridge]’ - lists current ports assigned to [bridge]
* ‘Sudo ovs-vsctl add-port [bridge] [name]’ - add port [name] to [bridge]
* ‘Sudo ovs-vsctl del-bridge [bridge] [name]’ - delete port [name] from [bridge]



In order to maintain both the control and data networks, all ethernet ports meant to be connected to the same data network (across which devices will communicate) must be assigned to the same bridge for each pi. Additionally, each pi will need at least one port freed up in order to communicate with the controller pi. To the left is a diagram showing a basic mapping of a possible topology for the SDN, wherein each pi is connected to each other pi (in case any of the connections should fail) and contains one free connection for other devices (in this case, the two priority servers and the ICD can be connected).

It should be noted that any number of ethernet ports may be assigned to the same bridge, and as such the above system can easily be adjusted to handle more devices by simply attaching one to a new port on any of the pi’s, making sure to then assign said port to its main bridge.

POX Controller Information

The controller’s purpose is to act a centralized network logic processing center. Switches on the network by default start with empty flow tables, and request routing logic when unsure of how to handle received packets. The controller has many options in terms of how it can handle a routing request, sending either new rules, one-time routing specifications, or commands to drop or flood the packet. It can furthermore make these decisions using any part of the packet header, allowing it to screen or re-adjust packet destinations based on type, source, or even size.

For this project, we have chosen to use the POX controller. Written in Python, it is much lighter and thus easier to work with than other controllers currently available. The following resources can be helpful when learning or working with POX:

* Programming in POX guide: <https://openflow.stanford.edu/display/ONL/Programming+in+POX>
* POX Documentation: <https://openflow.stanford.edu/display/ONL/POX+Wiki>

Priority Switch Documentation

The POX controller utilizes a py script which outlines an object representing an arbitrary switch in the network for handling routing logic. These objects are spawned by the controller whenever a new switch is connected to the network, and are mapped to their respective switches so that internal state variables can be stored independently for each switch. From there, events within the object definition are used to process different types of requests made by the switch to the controller.

For this project, we have created a modified l2 learning switch which we have named the PrioritySwitch. This object currently has the ability to specify a prioritized source, and based on constants defined within their code reroute data packets from this source to a prioritized destination. In order to create a largely unobtrusive network (one that allows communication unrelated to the prioritization system to continue on as it normally would), data is only redirected when sent from a specific ip range (192.168.100.2-192.168.100.6) to the default server, and will only be redirected to the priority server if necessary.

The current system also allows for the ICD’s (more on this later) ip address to be set. The ICD can send out packages in order to adjust prioritization levels within the controller. This is done by intercepting packets travelling from the ICD to any ip address within the above range (192.168.100.2 - 192.168.100.6), setting the destination packet as the prioritized source IP, and dropping the packet. This means that, while the ICD can communicate with other devices normally, it currently can not make direct responses to any of the listed ip addresses.

Crysis Eye Application Suite - High Level Explanation

The Crisis Eye App Suite was designed to work on top of the SDN network to demonstrate how a network sorting incoming data by source could be utilized. Specifically, Crysis Eye is an application suite which handles processing multiple incoming streams of real-time video and display them to an Incident Commander, who not only has a view of all incoming streams, but who can select streams to prioritize in order to improve their quality without inhibiting any of the other incoming streams.

The suite is made up of three components: A mobile application from which live video can be streamed to the network, a video server overlay which compiles incoming stream information and handles spawning/despawning instances of vlc media player to process and output streams, and a browser-based Incident Commander Display to automatically receive stream information and begin streaming video processed on our video servers accordingly:

Currently these are all linked statically across the testbed, with corresponding ip addresses found as reserved on the router as well as within the code of each component. While the server and ICD connections will need to remain static (this should not be a problem), the mobile app connections can easily be generalized to work with arbitrary numbers of connections.

Communication begins with the mobile app, which sends control data and stream data to the default video server. If a source has been prioritized, its stream will be redirected automatically by the network to the priority video server, but outside of this their processing remains the same; the video server overlay will read the data and instantiate a copy of vlc to process and stream it. Each server then notifies the ICD of active connections so that the ICD can begin pulling new streams to display them together.

The task of reprioritizing sources is handled then handled by the ICD. When a stream is clicked within the site, it will send an empty data packet to one of the mobile sources. As noted above, the network will ignore this packet, using it instead to pull a new prioritized IP address from the packet’s intended destination ip.

Crysis Eye Mobile Application Documentation

The Mobile App is a native android application using **NO** cross-platform API. It uses a pulled in framework for capturing the video-feed off of the Android Device and utilizes RTP + RTCP to send data directly to VLC.

Native Android Application uses:

PL: Java

IDE: Android Studios

Android Device: HTC M9

Android Version: 6.0.0+

The Android Application also utilizes GPS and control flag packets to direct the video server on how to interact with VLC.

The Native Application **ONLY** works on HTC Devices from testing. This is due to framework being used to transmit the video-feed over the SDN network.

NOTE: Samsung devices 'kind-of' work but their firmware causes memory-leaks in the app.

These are the key components to the software developed for showing the viability of the SDN.

Go Video Server Overlay Documentation

The data from the mobile applications is received first by a Go overlay for the video server designed to handle processing of the GPS and control flag packets received by the device and communicating to VLC (which must be installed on a machine acting as a video server) what is necessary to instantiate a new outgoing stream of the video feed received from the mobile app.

Once the stream has been created within VLC, the Go overlay begins forwarding both stream and GPS information to the ICD so that the ICD can connect to VLC and begin pulling the stream. The data is sent as a JSON array of objects containing the information for each individual stream in the following format:

[

// one specification of GPS/Stream data to be received by the ICD

{

"sourceIP" : "XX.XX.XX.XX", // mobile device sending the stream

"serverIP" : "XX.XX.XX.XX", // server currently processing the stream

"path" : "pathTo/vlcStream.ogg", // path to stream (used by VLC)

"latitude" : "30.22134299999999", // latitude of mobile device

"longitude" : "-92.04442999999998", // longitude of mobile device

"viewDir" : "5.49778714378" // orientation of mobile device

},

...

]

The Go Server Overlay will also monitor the incoming data to check for dropped streams. Should it fail to receive data from a specific IP address within a specific timeframe, it will instruct VLC to drop and clean up the stream instance used to process it, then remove the entry from its internal mapping, causing it to no longer send data for said IP address to the ICD.

The functionality of the Go Overlay was designed to create a dynamic memory stream processor that could act with anonymity; since the SDN redirects data between servers on its own, it is important that each server has the ability to act independently of each other and that data processing can begin and end spontaneously (when the network switches data flow). With this design the servers do not need to be notified of ny information regarding reprioritization nor whether or not they are a priority server, allowing new servers to be quickly and easily installed into the system.

Crysis Eye Incident Commander Display Documentation

The Incident Commander Display is the final component of the Crysis Eye app suite, and is responsible for collecting and displaying both streams and their respective gps data. It is browser-based, relying almost entirely on javascript for its functionality, and operates as follows:



Upon receiving a JSON array of packet information in the format mentioned above **(1)**, it will parse and store these objects within an object map using each stream’s source IP as a key **(2)**. It also appends timestamps to each object for monitoring their age, discarding them after 30 seconds.

The ICD simultaneously instantiates a new video element on the page **(3)**, using events tied to it to force the element to attempt reloading the stream should it stall on the browser. If the element fails to load the stream after a set amount of time the video element is removed from the page and its corresponding entry in the object map is removed.

The ICD will also pull a new map from google static maps every four seconds to display the position and orientation of each streaming device next to the collection of streams **(4)**. Currently the ICD does not show a mapping between the GPS points and streams, but as they originate from the same object in the mapping this task should not be difficult.

Finally, the ICD is designed to send an empty packet to a mobile device when its associated video element is clicked **(5)**. As mentioned in the PrioritySwitch section, this packet is dropped by the network controller, with its destination (the mobile device in question) being set as the new priority source. The video servers will automatically adjust to this switch accordingly, and as the ICD starts receiving data for a given mobile device from the new server it will simply replace the old data in its stream map with the newly received data and reload the stream as needed.

Current Hardware Needs

Currently the project only has equipment for the sDN testbed, with the app suite running on devices pulled from around IRI. In order to ensure that its development can continue efficiently, dedicated equipment will be needed over the course of the project:

* SDN Project:
  + 1 control network router (to make setting up control layer easier)
* Crysis Eye Mobile App:
  + 1-2 HTC M9 phones (verified working on this device)
  + 1-4 android devices (future development to expand app compatibility)
  + 1-2 iphones (only needed in the case app needs to be ported over)
* Crysis Eye Video Servers:
  + 2-4 video servers
    - \*\*\* these devices primarily need strong processors and high amounts of memory for handling multiple incoming streams. Talk to Mike about what machines would work for a good candidate.
  + + 2-4 video servers during project expansion
* Crysis Eye Incident Commander Display:
  + 1 decent laptop
    - We managed to make an old dell from LITE work properly as the ICD host, but a better machine may be needed as more streams are sent to it.

The first item in each list will be needed ASAP in order to continue proper development and testing of the app suite. Beyond that, items can be purchased as needed by the project depending on the direction it takes.

Next Steps

The current version of the app suite and priority switch offer a rough conceptual demo of our intended final suite. While the current system has the core functionality in place, There are several ways in which this functionality can be expanded and/or improved. Below are a collection of ways in which the network and app suite can be improved upon:

* SDN Network:
  + In-parallel packet sending behavior:

Devise a way to send multiple packets throughout the network between steps while properly handling duplicate packets reaching destination successfully

* + Database for complex prioritization
  + Dynamic load balancing
  + Selective reprioritization packet-dropping
* Crysis Eye Mobile App:
  + Chat/messaging system
  + Expanding app compatibility
* Crysis Eye Video Server:
  + Custom fine-tuning video server
  + Generalized stream control data generation
* Crysis Eye ICD:
  + Chat/messaging system
  + Delay to stream source-switching