Panorama

Prof. M.S. Gaur
Ankit Gangwal

Department of Computer Science and Engineering,
Malaviya National Institute of Technology,
Jawaharlal Nehru Marg, Malviya Nagar, Jaipur, Rajasthan, 302017

Acknowledgements

This material is based upon work supported in part by **ISEA-II (Information Security Education and Awareness, Phase II)** project. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of ISEA-II.

Contents

Chapter 1

Introduction

Software-defined networking (SDN) [1, 2] is a recently emerging paradigm. The key concept is to partition a network into data plane and control plane. The data plane consists of many forwarding devices that provide actual forwarding of data packets. The data plane is controlled by the control plane. The control plane consists of at least one decision-making entity called the controller. The controller has a global view of the network of its domain, and it communicates with the forwarding devices to obtain real-time traffic information. Utilizing the information about topology and traffic statistics, the controller can determine the route of data packets in the network.
OpenFlow [3] is a protocol that allows the controller to communicate with the switches. It establishes a secure communication channel between the controller and switches. The controller instructs the switches by simply updating their 'flow table' via the OpenFlow protocol. Apart from adding a new flow table entry, the controller can delete/modify existing entries. It can also query real-time traffic statistics using the OpenFlow protocol.
A graphical user interface has advantages over command line interface. Panorama is a lightweight application designed to accelerate the work of OpenFlow users, explorers and researchers. Visualizing state (traffic statistics, topology information, etc.) of an OpenFlow network is now easier with Panorama.

Chapter 2

Features

Panorama provides a real-time bird's eye view of an OpenFlow Network. Panorama collects and depicts vital information about the network.

2.1 N/w Configuration

Network configuration focuses on the information related to network topology.

1. **Switch Discovery:** When a switch connects to a controller handshake messages are exchanged and the controller learns that a switch with a unique identifier called "DataPath ID" (DPID) is connected to it. Information stored for each discovered link includes:

   (a) Serial number.

   (b) Software version.

   (c) Vendor.

   (d) Hardware type.

   (e) DPID.

   When a connection to a switch has been terminated (either because it has been closed explicitly or the switch was restarted) the controller learns that the switch is disconnected.

2. **Link Discovery:** LLDP (Link Layer Discovery Protocol) is used to discover links between switches. OpenFlow controller sends PACKET_OUT messages to every discovered switch. Those packets are controller-specific LLDP packets. The controller ask the switch to broadcast this packet with its DPID. When the broad-casted packet is received by an adjacent switch, it sends this packet to the controller asking for a forwarding rule and the controller learns that there is an unidirectional link the two switches. The entire process is illustrated in Figure 2.1. Information stored for each discovered link includes:

   (a) DPID of switches, between which the link exists.

   (b) Port number of the switches.

   When a switch detects that a link to an adjacent switch has been removed or failed, it informs the controller by raising an event.

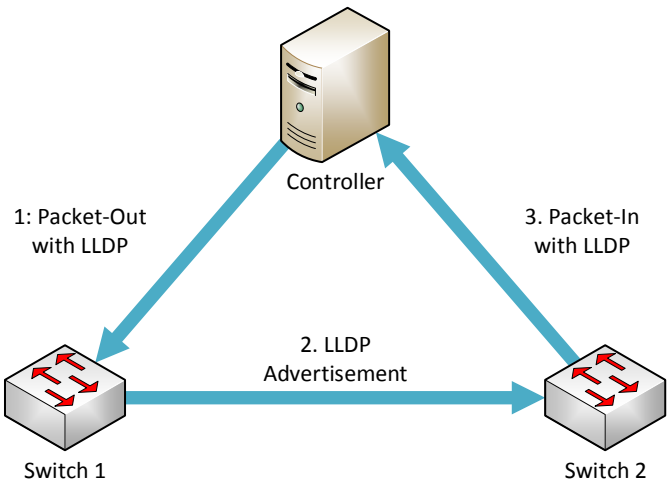


Figure 2.1: Link Discovery in OpenFlow Networks

3. **Host Discovery:** When a packet from a host reaches to the switch it connected to, the switch matches the packet with existing rules. For the first packet from a host, there would be no matching rule. Hence, the switch sends this packet to controller asking for a rule. By inspecting the packet the controllers learns the location of the host. Since, a host might move from one switch to another or DHCP may reassign IP to hosts, Panorama maintains a timer for each discovered hosts. A clean up is performed periodically. The duration of this period is greater than expected hard timeouts. Information stored for each discovered host includes:

   (a) IP Address.

   (b) MAC Address.

   (c) Time, which defines the time when the host was discovered.

2.2 Port Stats

OpenFlow PORT_STATS messages are used to obtain per port statistics for every port on a switch. When a switch receives such a request, it replies with number of transmitted packets, transmitted bytes, transmit error, packet dropped by TX, collisions, received packets, received bytes, receive errors, packet dropped by RX, packets with RX overrun, frame alignment errors, CRC errors for each of its port.

port_no	tx_packets	tx_bytes	tx_errors	tx_dropped	collisions	rx_packets	rx_bytes	rx_errors	rx_dropped	rx_over_err	rx_frame_err	rx_crc_err

Table 2.1: Main components of Port Statistics

2.3 Aggregate Stats

Aggregate Statistics for a switch include total number of flows installed, total number of packets and total number of bytes processed by the switch.

flow_count	packet_count	byte_count

Table 2.2: Main components of Aggregate Statistics

2.4 Flow Stats

OpenFlow FLOW_STATS messages are used to obtain per flow statistics for every flow rule installed on a switch. A flow table consists of several flow entries, and each flow table entry contains:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Table 2.3: Main components of a flow entry

1. **Match Fields:** These are the field against which packets are matched. A match field may be exact or may be wild-carded (match any value).

Ingress Port	Ether Src.	Ether Dst.	Ether Type	VLAN Id	VLAN Priority	MPLS Label	MPLS Traffic Class	IP Src.	IP Dst.	IP Proto.	IP ToS	Transp. Src. Port	Transp. Dst. Port

Table 2.4: Main components of the Match field

2. **Priority:** It is an integer number that defines the matching precedence of the flow entry. The higher the number, the higher the priority.

3. **Counters:** Counters are updated when packets are matched. Counters are maintained for each port, queue, flow entry, flow table, meter, meter band and group.

4. **Instructions:** An instruction is executed, when a packet matches a flow entry. Instruction may be "Required Instruction" or "Optional Instruction". Instructions either modify pipeline processing or contain a set of actions to be performed for the match.

5. **Timeouts:** A flow entry may have two type of timeouts hard and idle. Depending upon the type of timeouts, they define either the maximum amount of time or idle time before a flow is evicted by a switch.

6. **Cookies:** Controller usages cookies to filter flow modification, statistics and deletion but not at the time of processing packets.

An OpenFlow switch should essentially have at least one flow table, and it can optionally have more than one flow tables as well. The advantage is that many network deployment require orthogonal processing of packets (e.g., QoS, ACL and routing). Using a single to force all those processing creates a huge rule-set. Using multiple tables may properly decouple those processing. Packet processing through multiple flow tables is called the OpenFlow pipeline processing. Panorama reports pipeline processing in the instructions field.

2.5 Link Throughput

One of the important aspect of network monitoring is to estimate throughput over network links. Panorama calculates uni-directional and bi-directional throughput between a pair links using the PORT_STATS received from the switches. Calculation of uni-directional and bi-directional throughput in bits per second (bps) is shown in Eq. 2.1 and Eq. 2.2 respectively.

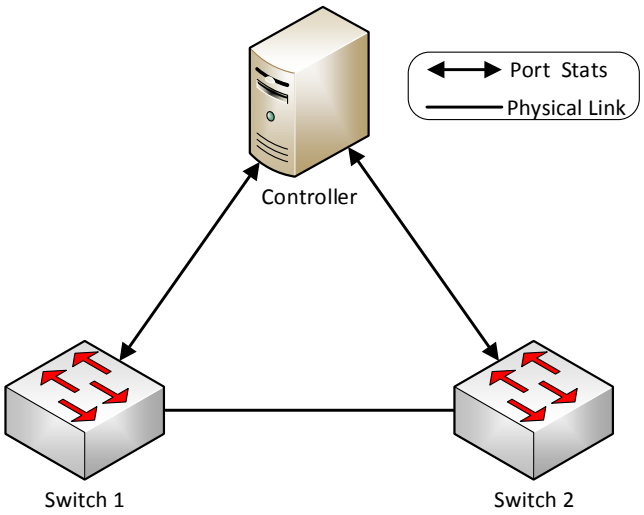


Figure 2.2: Measuring link throughput

$$Throughput^{Uni}_{S1 \rightarrow S2} = (Cur^{S1}_{tx\_bytes} - Pre^{S1}_{tx\_bytes}) * 8.0/(Cur^{S1}_{time} - Pre^{S1}_{time}) \quad (2.1)$$

$$Throughput^{Bi}_{S1 \leftrightarrow S2} = (Cur^{S1}_{(tx+rx)\_bytes} - Pre^{S1}_{(tx+rx)\_bytes}) * 8.0/(Cur^{S1}_{time} - Pre^{S1}_{time}) \quad (2.2)$$

Chapter 3

Installation

3.1 Prerequisites

Panorama requires following packages to be pre-installed on the system:

1. POX [4]
2. OpenFlow 1.0
3. Python 2.7
4. Web browser with javascript support.

3.2 Setting-up the Environment

If not already done, set path to `pox.py` under pox directory (`~/pox/`) in your *$PATH* variable. To set the path through terminal write:

```
$ PATH=$PATH:~/pox/
$ export PATH
```

3.3 Getting the Project Files

Get the project code from git repository and copy the project directory under pox (`~/pox/ext/`).

Chapter 4

How to Start

Panorama can be launched either manually or automatically.

1. **Manual Execution:**

   (a) Open a new terminal and run a controller module. This module provides forwarding rules. You may run your own controller module as well.

   ```
   $ cd pox
   $ pox.py forwarding.l2_learning
   ```

   (b) Open another terminal and run the monitoring module.

   ```
   $ cd (project working directory i.e. ~/pox/ext/panorama)
   $ pox.py openflow.of_01 --port=5566 panorama.panorama
   ```

   (c) Open another terminal and start a network in Mininet [5, 6]. You may also create your own custom topology (see Appendix A) in which OpenFlow enabled switches connect to both controller module and monitoring module.

   ```
   $ cd (project working directory i.e. ~/pox/ext/panorama)
   $ sudo python topology/panorama_mininet.py
   ```

   (d) If a browser window/tab doesn't come up automatically, open a browser window/tab and navigate to localhost.

   ```
   http://localhost:8080/
   ```

2. **Automatic Execution:** Simply run the bash script.

   ```
   $ cd (project working directory i.e. ~/pox/ext/panorama)
   $ bash launch.sh
   ```

Chapter 5

Experiment and Verification

Panorama is tested in emulation environment as well as on physical testbed. The emulation was done using Mininet as the network emulator on a laptop with following specification:

- AMD A4-5000 1.5 GHz Quad Core APU with Radeon HD Graphics
- 4 GB of RAM
- 500 GB HDD

The physical testbed comprises of an HP Aruba 5406R zl2 Switch with Panorama running on a PC. To verify dpctl [7] utility is used. Appendix B shows the captured screen-shots. The specification of the PC are listed below.

- 4th Gen Intel Core i3 processor with Intel HD Graphics
- 4 GB of RAM
- 500 GB HDD

Chapter 6

Future Work

Delay introduced by a link and link loss play a crucial role in Quality-of-Service provisioning. The work in [8] demonstrates mechanisms to measure link delay and loss. In future, we shall incorporate those measurements and we also hope to develop Panorama as controller independent module.

6.1 Link Delay

The controller injects probe packets to calculate delay on the links. It generates UDP packet and sends it to Switch 1. The controller also installs a new rule in flow table of Switch 1 instructing Switch 1 to send this probe packet to Switch 2. At $t1$ the packet flows from Switch 1 to Switch 2. Since no matching rule is installed in flow table of Switch 2, the switch returns the packet to the controller at $t2$. The controller receives this packet at $t3$. The controller keeps constant connection with every switch, hence, it knows the delay between it and the switches, as shown in Eq. 6.1. This enables controller to estimate the delay between every pair of switches, as shown in Eq. 6.2.

$$T1 = T3 = 0.5 * (T_b - T_a) \tag{6.1}$$

$$T_a = OF_{request\_send\_time} \; and \; T_b = OF_{reply\_receive\_time}$$

$$Time_{total} = T1 + T2 + T3 \Rightarrow T2 = Time_{total} - T1 - T3 \tag{6.2}$$

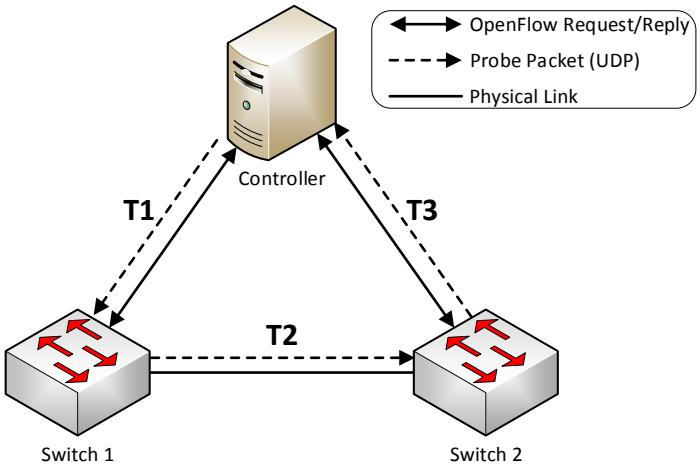


Figure 6.1: Measuring link delay

6.2 Link Loss

6.2 When a flow arrives and the switch does not have any matching rule installed, the first packet of the flow is sent to the controller. The controller instructs the switch what to do with the packet. It instructs the switch by installing table rules on the switch for the flow. As soon as the flow expires, the switch indicates this event to the controller. The entire process is illustrated in Figure 6.2. The flow is installed at time $t1$ using FLOW_MOD messages. At time $t2$ and $t3$, the controller receives FLOW_REMOVED messages from Switch 1 and Switch 2 respectively. Those messages include specific statistics for the flow, e.g., the number of packets, bytes. The packet-loss can be measured on the basis of those statistics, as shown in Eq. 6.3.

$$Loss_{\%} = (1 - packets_{Switch2}/packets_{Switch1}) * 100 \tag{6.3}$$

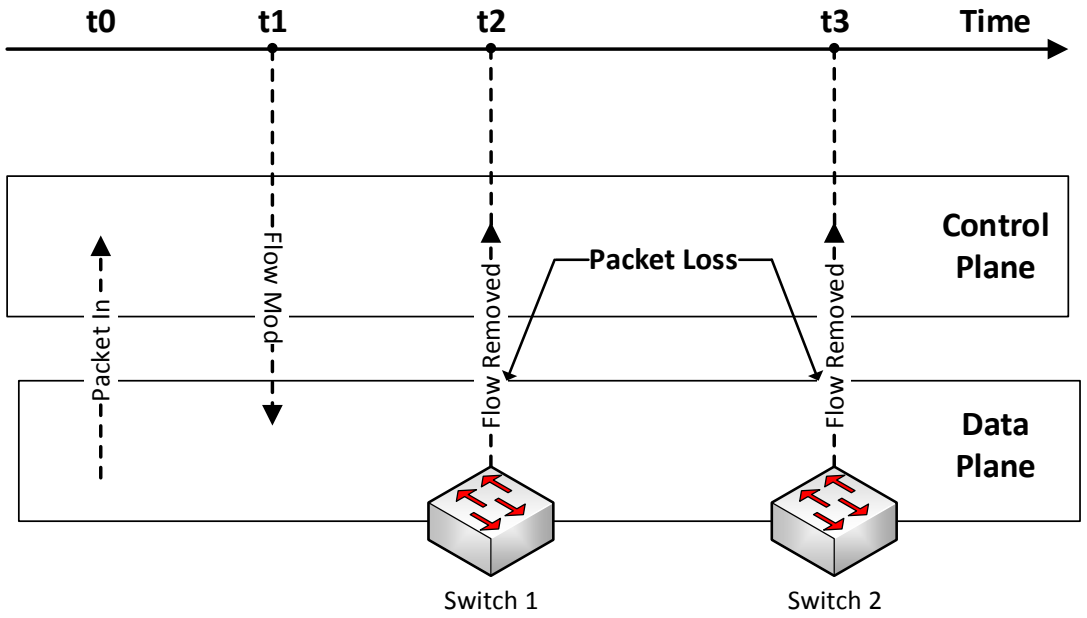


Figure 6.2: Measuring link loss

Bibliography

[1] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 4, pp. 1955–1980, 2014.

[2] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti *et al.*, "A survey of software-defined networking: Past, present, and future of programmable networks," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 3, pp. 1617–1634, 2014.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[4] POX, http://www.noxrepo.org/pox/about-pox/, last accessed on Mar. 01, 2016.

[5] Mininet, http://mininet.org/, last accessed on Mar. 01, 2016.

[6] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[7] dpctl, https://github.com/CPqD/ofsoftswitch13/wiki/, last accessed on Mar. 01, 2016.

[8] V. N. Gourov, "Network Monitoring with Software Defined Networking: Towards OpenFlow network monitoring," Ph.D. dissertation, TU Delft, Delft University of Technology, 2013.

Appendix A

Custom Topology Script

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSKernelSwitch, UserSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.link import Link, TCLink
from mininet.util import custom, pmonitor

def topology():
    print "*** Creating network."
    net = Mininet( controller=RemoteController, link=TCLink, switch=
        OVSKernelSwitch )

    print "*** Creating nodes"
    h1 = net.addHost( 'h1', mac='00:00:00:00:00:01', ip='10.0.0.1/8' )
    h2 = net.addHost( 'h2', mac='00:00:00:00:00:02', ip='10.0.0.2/8' )
    s3 = net.addSwitch( 's3', listenPort=6673, mac='00:00:00:00:00:03' )
    s4 = net.addSwitch( 's4', listenPort=6674, mac='00:00:00:00:00:04' )

    c10 = net.addController( 'c10', controller=RemoteController, ip='127.0.0.1',
        port=6633 )
    c11 = net.addController( 'c11', controller=RemoteController, ip='127.0.0.1',
        port=5566 )

    print "*** Creating links"
    net.addLink(s3, h1, cls=TCLink)
    net.addLink(s3, s4, cls=TCLink, bw=1, delay='0ms', loss=0)
    net.addLink(s4, h2, cls=TCLink)

    print "*** Starting network"
    net.build()
    c10.start()
    c11.start()
    s3.start( [c10,c11] )
    s4.start( [c10,c11] )
    CLI( net )

    print "*** Stopping network"
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    topology()
```

Listing A.1: Sample topology script

Appendix B

Screenshots

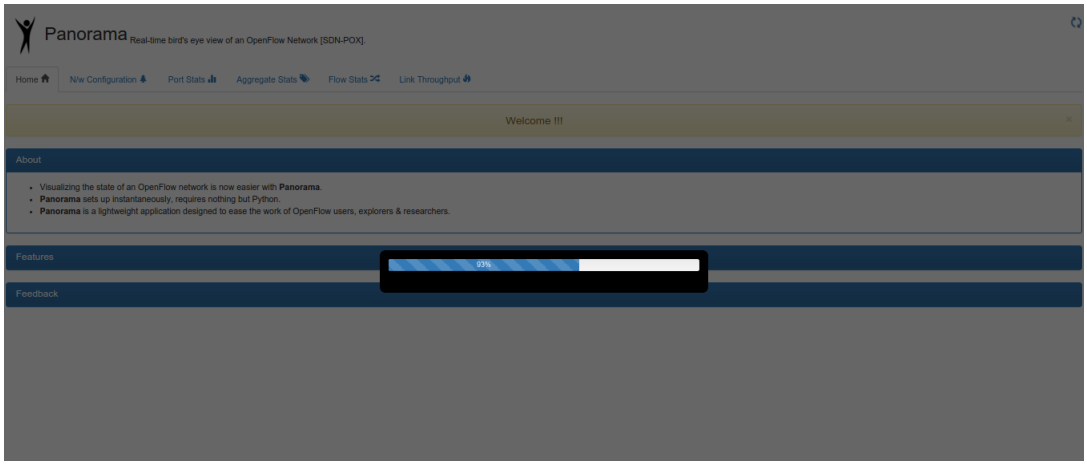


Figure B.1: Loading

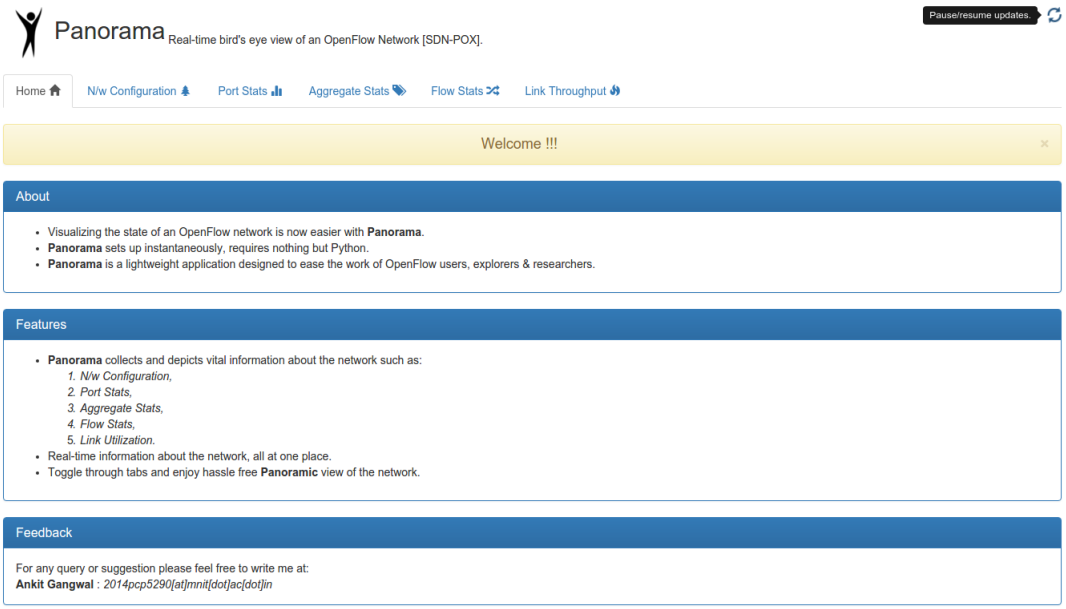


Figure B.2: Home tab

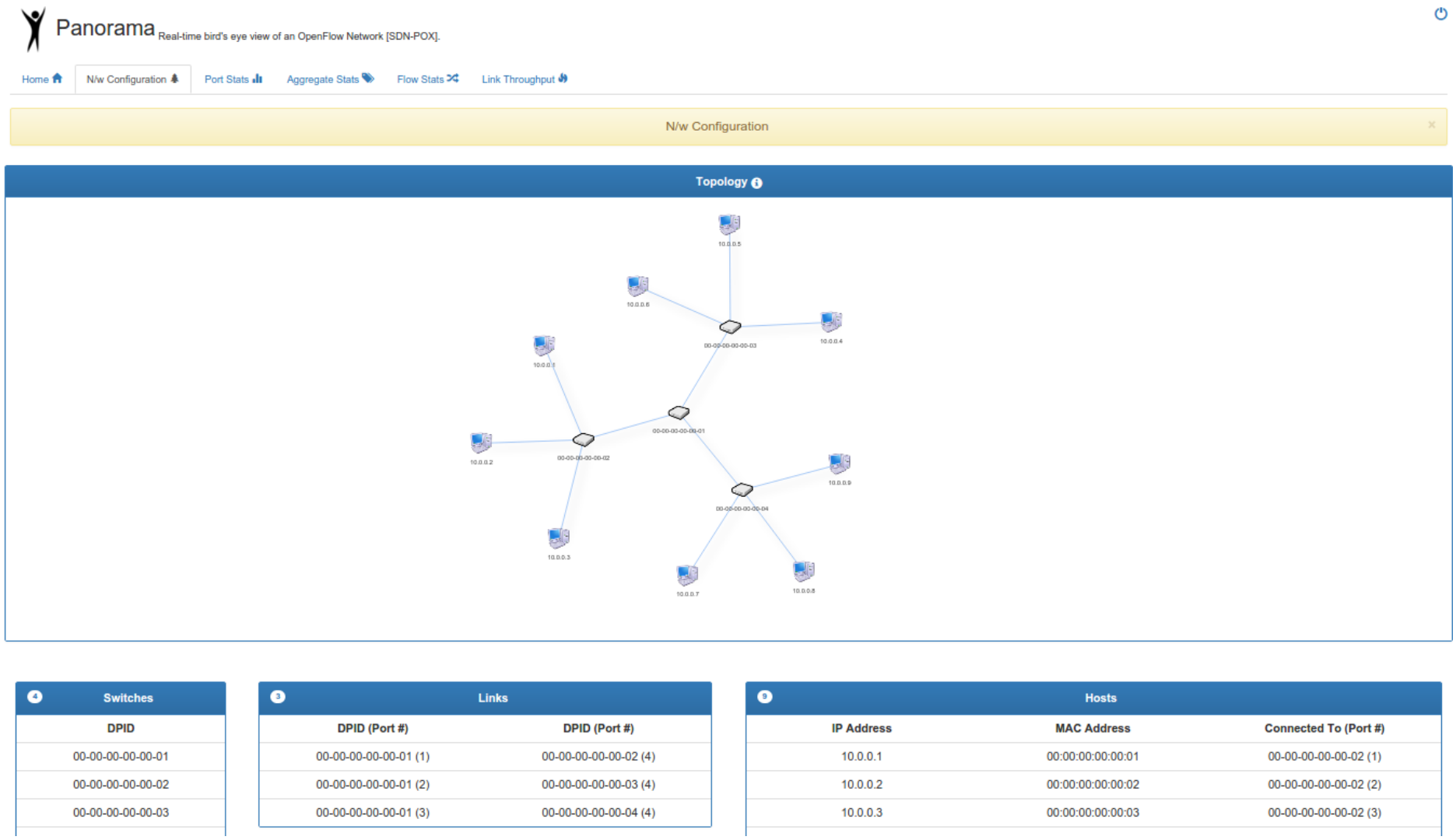


Figure B.3: N/w Configuration tab

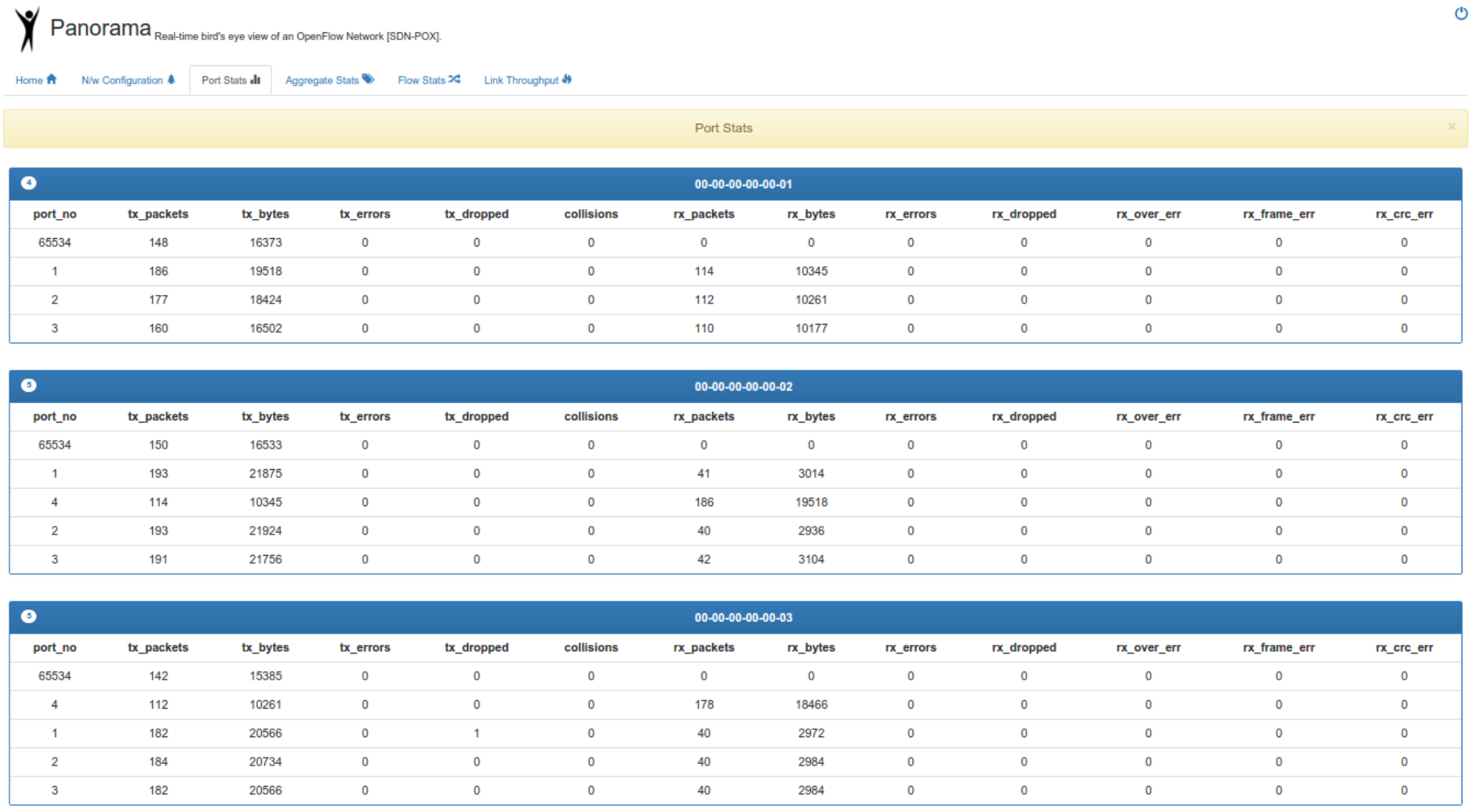


Figure B.4: Port Stats tab

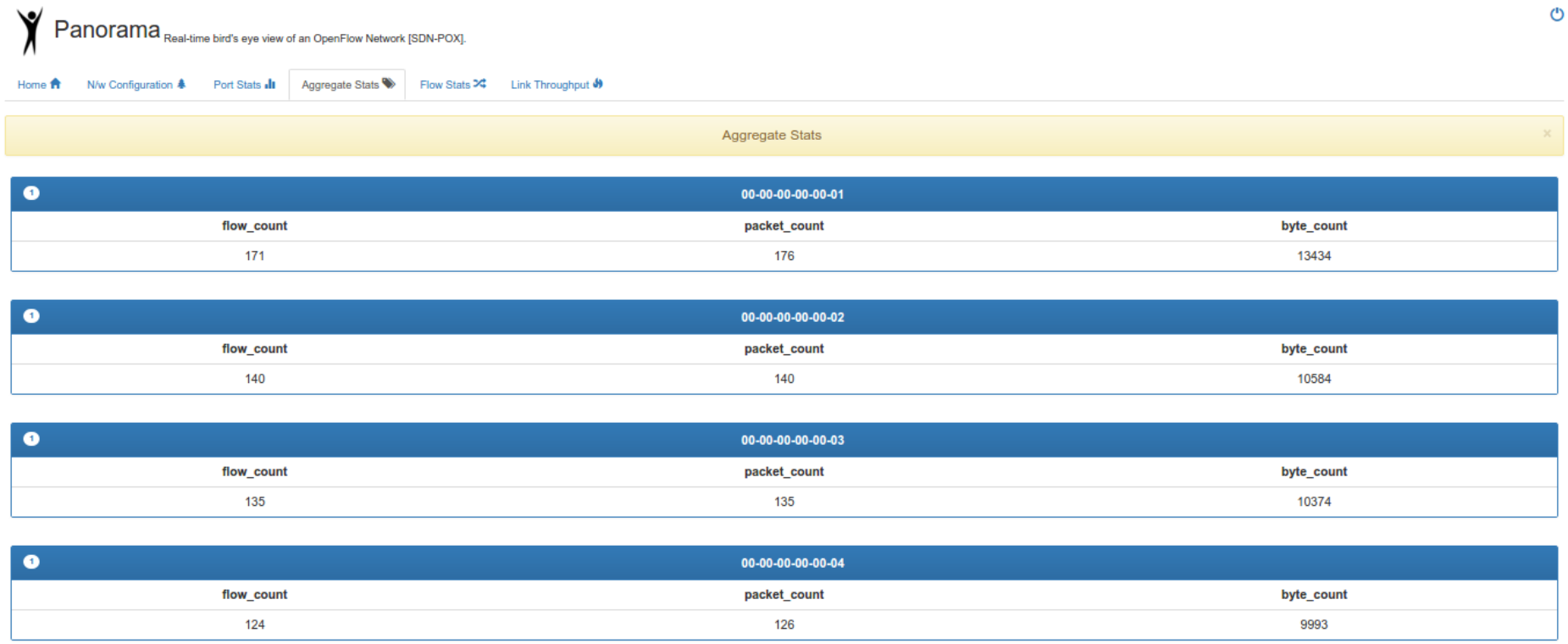


Figure B.5: Aggregate Stats tab

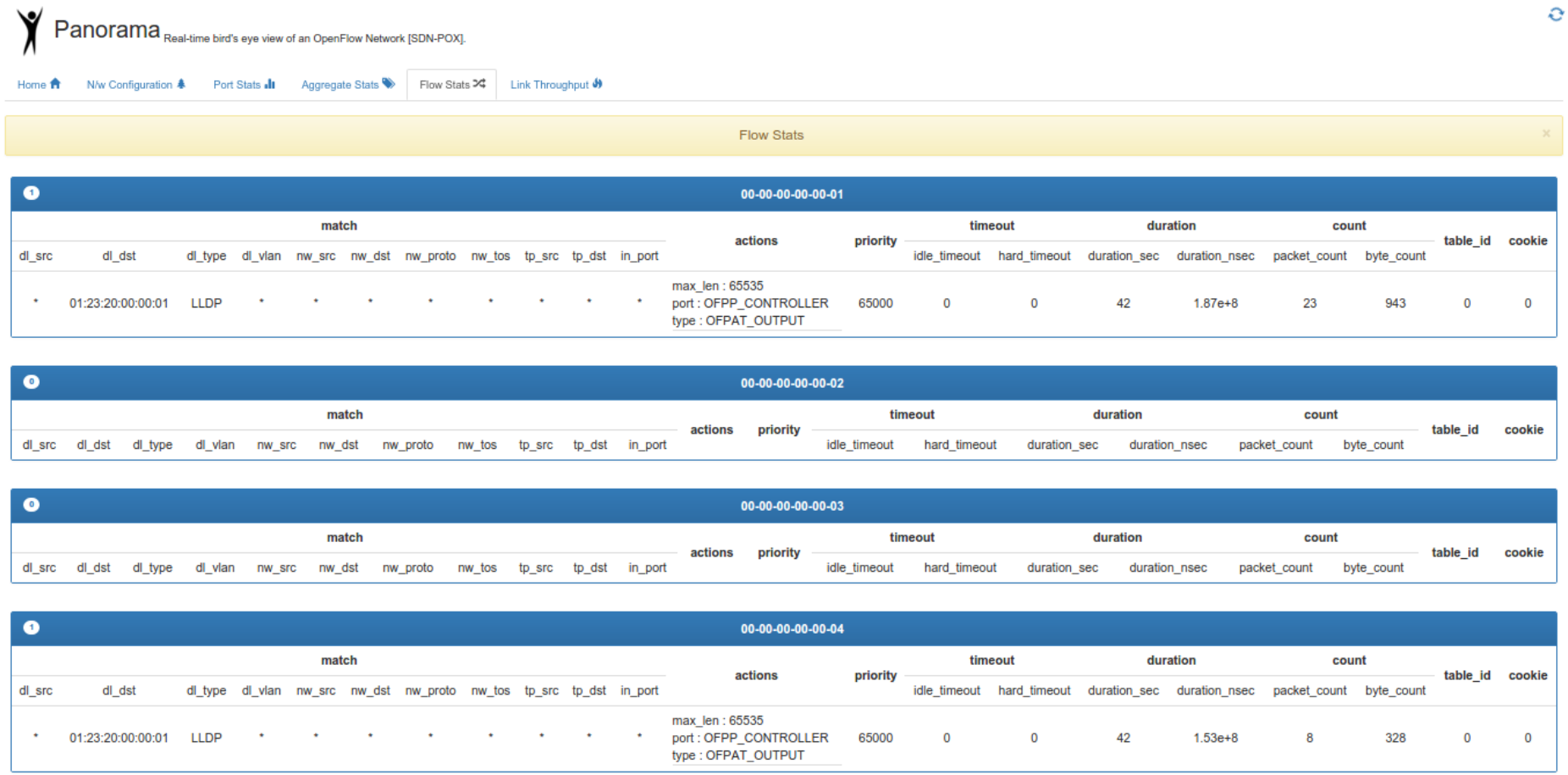
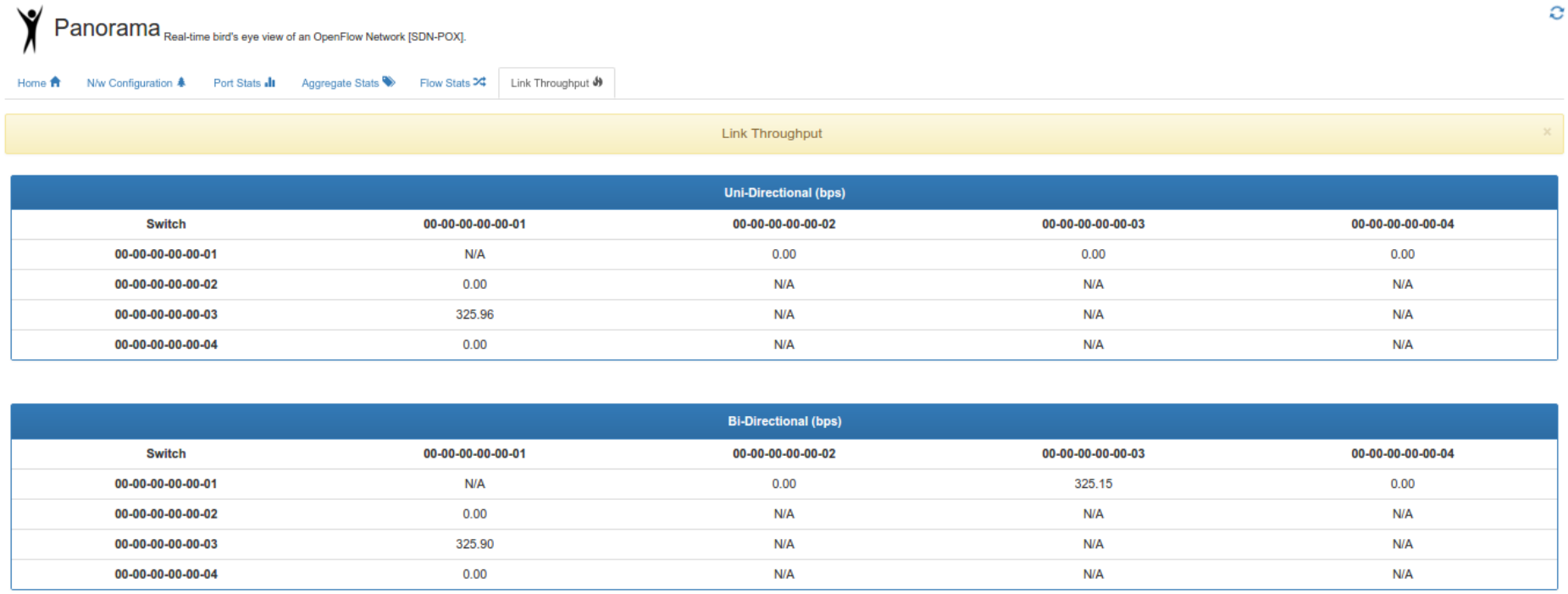


Figure B.6: Flow Stats tab



Uni-Directional (bps)				
Switch	**00-00-00-00-00-01**	**00-00-00-00-00-02**	**00-00-00-00-00-03**	**00-00-00-00-00-04**
00-00-00-00-00-01	N/A	0.00	0.00	0.00
00-00-00-00-00-02	0.00	N/A	N/A	N/A
00-00-00-00-00-03	325.96	N/A	N/A	N/A
00-00-00-00-00-04	0.00	N/A	N/A	N/A

Bi-Directional (bps)				
Switch	**00-00-00-00-00-01**	**00-00-00-00-00-02**	**00-00-00-00-00-03**	**00-00-00-00-00-04**
00-00-00-00-00-01	N/A	0.00	325.15	0.00
00-00-00-00-00-02	0.00	N/A	N/A	N/A
00-00-00-00-00-03	325.90	N/A	N/A	N/A
00-00-00-00-00-04	0.00	N/A	N/A	N/A

Figure B.7: Link Throughput tab