Scalable Flow Monitoring for Data Center

A Dissertation Submitted in partial fulfillment of the degree of

Master of Technology
in
Computer Science

By
Nirmoy Das

School of Computer and Information Sciences
University of Hyderabad, Gachibowli
Hyderabad - 500046, India

Month, Year

CERTIFICATE

This is to certify that the dissertation entitled "**Scalable Flow Monitoring for Data center**" submitted by **Nirmoy Das** bearing Reg. No. 11MCMT20, in partial fulfillment of the requirements for the award of Master of Technology in Computer Science, is a bona fide work carried out by him/her under my/our supervision and guidance.

The dissertation has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Anupama Potluri
School of Computer and Information Sciences,
University of Hyderabad

Dean,
School of Computer and Information Sciences,
University of Hyderabad

DECLARATION

I, **Nirmoy Das** hereby declare that this dissertation entitled "**Scalable Flow Monitoring for Data center** " submitted by me under the guidance and supervision of Dr. Anupama Potluri is a bona fide work. I also declare that it has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Date:

Nirmoy Das
Reg. No.: 11MCMT20

Signature of the Student

To,

My Parents and Supervisor.

Contents

1 Related Work **1**

1.1 Edge Monitoring and Collection for Cloud (EMC2) [17] 1

1.1.1 Architecture 1

1.1.1.1 Flow-Table 2

1.1.1.2 NetFlowParser/sFlowParser 3

1.1.1.3 NetFlowCollector/sFlowCollector 3

1.1.2 Advantages and Limitations 4

1.2 Scalable Internet Traffic Measurement and Analysis with Hadoop [16] 4

1.2.1 Architecture 4

1.2.1.1 Traffic collector 5

1.2.1.2 IP packet and NetFlow reader in Hadoop 5

1.2.1.3 Analysis in MapReduce 5

1.2.1.4 Interactive query interface with Hive 6

1.2.2 Advantages and Limitations 6

1.3 *nfdump* [8] 7

1.3.1 Advantages and Limitations 7

1.4 Packet Forwarding with *netmap*[20] 8

1.5 Summary 9

2 Cassandra plugin for *ntop* **10**

2.1 Overview 10

2.2 *ntop* 10

2.2.1 Packet Sniffer 11

2.2.2 Packet Analyzer 11

2.2.3 Traffic Rules 11

2.2.4 Report Engine 11

2.2.5 Plugins 12

2.3 Cassandra Database 12
2.3.1 Cassandra Architecture: 12
2.3.2 Why Cassandra 14
2.4 Cassandra C Client 16
2.4.1 Schema Manipulation API 16
2.4.2 Data Manipulation API 18
2.4.3 Utility API 18
2.5 Cassandra Plugin for *ntop* 19
2.5.1 Architecture 20
2.6 Results 20

3 Scalable Flow Monitoring of OpenVswitch **22**
3.1 Overview 22
3.2 NetFlow Monitoring 22
3.3 Proposed Scalable Solution for Flow Monitoring 25

Bibliography **26**

List of Figures

1.1 Architecture of EMC2 [17]. 2
1.2 Architecture of *nfdump*. 7

2.1 Architecture of *ntop*. 11
2.2 Cassandra Ring [6] 13
2.3 Cassandra Data Model 14
2.4 Cassandra Read/Write operation 15
2.5 Cassandra Performance [4]. 16
2.6 API interaction with Cassandra 17
2.7 Architecture of Cassandra Plugin. 20
2.8 Data Model for Cassandra *ntop* Plugin. 21

List of Tables

1.1 OpenVswitch Performance with *netmap* 8

2.1 Comparison between RDMS schema with Cassandra Data Model . . 13
2.2 Data generated by two node testbed. 14
2.3 APIs for Plugin Support 19

3.1 Netflow Header Format 23
3.2 NetFlow Record Format 24

Chapter 1

Related Work

Flow monitoring protocols like NetFlow [5] and sFlow [12] can provide important information about traffic that passes through a network. However, contemporary networking with its 10Gbps and higher NICs is outpacing the ability to monitor them efficiently. As data centers are getting virtualized with virtual software switches and scaling to thousands of nodes, it is an immediate requirement to have monitoring systems that can scale effectively. There are not many solutions proposed that provide scalable flow monitoring in data center networks. In this chapter, we review some of the literature that deals with scalable flow monitoring in data center networks.

1.1 Edge Monitoring and Collection for Cloud (EMC2) [17]

Edge Monitoring and Collection for Cloud (EMC2) is a scalable network-wide monitoring service for data centers. EMC2 stays inside the host computer to monitor virtual switches. Monitoring at virtual switch is scalable due to the distributed nature of the storage of the collected information.

1.1.1 Architecture

Figure 1.1 shows the architecture of EMC2.

EMC2 is a multi-threaded application that contains the following modules:

1. Flow-Table : Flow-Table is an in-memory 2-level hash table.

2. NetFlowParser : It parses the NetFlow datagrams and updates the Flow-Table.

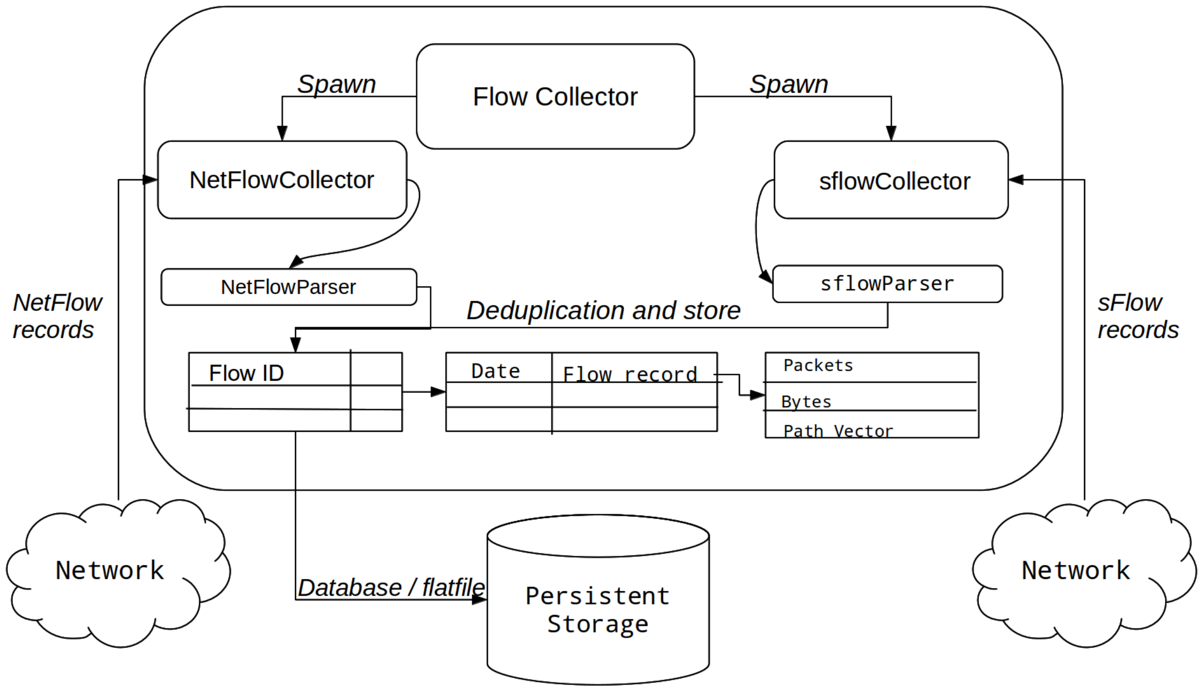

Figure 1.1: Architecture of EMC2 [17].

3. sFlowParser : It parses the sFlow datagrams and updates the Flow-Table.

4. NetFlowCollector : It accepts the NetFlow datagrams and creates the parser thread upon receiving NetFlow datagrams.

5. sFlowCollector : It does the same task like the NetFlowCollector but for sFlow datagrams.

6. FlowCollector : It invokes two thread – NetFlowCollector and sFlowCollector – for accepting flow datagrams.

1.1.1.1 Flow-Table

Flow-Table is a 2-level in-memory hash table. The primary key for the hash table is the Flow ID which is formed based on the layer 3 source and destination addresses. Flow ID maps to another hash table where timestamp is the key and flow record is the value. Each flow record contains number of packets, number of bytes and optional path vector.

1.1.1.2 NetFlowParser/sFlowParser

NetFlowCollector and sFlowCollector create these two parser threads upon receiving a NetFlow/sFlow datagram respectively. These parser threads parse the datagram and update the Flow-Table. They also perform two important tasks:

1. Deduplication.
2. Data rate prediction in presence of sampling.

Deduplication: Deduplication prevents duplicate flow records from being added to the Flow-Table. It uses the following algorithm.

Algorithm 1: Detect Duplicate Flow

```
if $flow - ID$ not exist then
    add flow to the flow table.
    return
else
    if Same exporter then
        update the flow table.
        return
    else
        report duplicate flow.
        update path vector.
        return
    end if
end if
```

Data Rate Prediction in Presence of Sampling: Sampling rate is specified in flow datagrams. Parser thread predicts the data rate by multiplying sampling rate with the length of the packet.

1.1.1.3 NetFlowCollector/sFlowCollector

NetFlowCollector/sFlowCollector are collector thread that wait for new NetFlow or sFlow datagrams and spawn a new NetFlowParser/sFlowParser thread upon receiving a datagram.

1.1.2 Advantages and Limitations

The authors state the following advantages of EMC2:

- Scalable monitoring as EMC2 monitors host *vswitches* in a distributed fashion by storing the information as flat files in those hosts itself instead of sending them to a centralized collector.

However, flat files are not really built for scalability unlike many other distributed databases available today such as Cassandra, Big Table etc.. Therefore, it is not clear how much of performance can be obtained by storing the information in flat files in the host itself. Considering that most data centers work with SANs rather than local disks, this may not be as scalable as claimed.

1.2 Scalable Internet Traffic Measurement and Analysis with Hadoop [16]

Hadoop [1] is a distributed computing platform that uses a distributed file system(HDFS) and MapReduce [7] programming model. A Hadoop cluster consists of commodity hardware that can scale to thousands of nodes to store huge amounts of data. It can perform massive data analytics operations on the available data using MapReduce. Storing NetFlow data on Hadoop and analysis using MapReduce offers scalable traffic measurement and analysis.

1.2.1 Architecture

Traffic measurement and analysis system with Hadoop consists of the following modules:

1. Traffic collector.
2. IP packet and NetFlow reader in Hadoop.
3. Analysis in MapReduce.
4. Interactive query interface with Hive [3].

1.2.1.1 Traffic collector

Traffic collection is done by a high-speed packet capture driver and load balancer. Load balancer forwards packets into multiple Hadoop data nodes evenly. Traffic collector also reads trace files stored on the disk and writes them into HDFS. Trace files contains Netflow packets or IP packets in *libpcap* format.

1.2.1.2 IP packet and NetFlow reader in Hadoop

Storing binary trace files into Hadoop specific sequence files needs more computation power as every packet has to be sequentially read from the trace file and stored into HDFS. The authors have developed new Hadoop APIs to store trace files directly into HDFS. As there is no distinct mark to find out the end of a packet record in *libpcap* format, authors proposed a heuristic algorithm using timestamp-based bit pattern.

Timestamp-based heuristic algorithm using MapReduce to identify packet records: MapReduce job invokes multiple map tasks to process each HDFS block in parallel. Each of the map tasks follows these steps to identify the packet records in a HDFS block:

1. Read two records using the *libpcap* 16-byte header.
2. Check timestamp value, that should be within duration of captured time.
3. Difference between wired packet length and captured length should be less than maximum packet length.
4. Check whether timestamp difference between the two packet records is within the defined threshold.

1.2.1.3 Analysis in MapReduce

The authors have implemented analysis tools for processing IP packet as well as NetFlow packets using MapReduce algorithms. The tools implemented are:

1. IP Layer analysis tools provide the following analysis jobs:
    (a) Host and port count statistics.
    (b) Periodic flow statistics.

    (c) Periodic sample traffic statistics.

2. TCP layer analysis computes the following statistics:
    (a) RTT.
    (b) Retransmission.
    (c) Throughput.
3. NetFlow analysis provides
    (a) Human readable flow statistics.
    (b) Aggregated flow statistics.
    (c) Top flows sorted by a key such as packet count or byte count.

1.2.1.4 Interactive query interface with Hive

Hive is a data warehousing system build on top of Hadoop that allows us to generate MapReduce jobs using SQL like query. IP analysis MapReduce jobs process NetFlow packets on HDFS and store flow record and IP statistics into Hive tables. A user can then query the Hive tables using the interactive web-based user interface.

1.2.2 Advantages and Limitations

Advantages of Hadoop based traffic measurement and analysis are

1. Scalable storage.
2. MapReduce operations on flow data.

Disadvantages of Hadoop based traffic measurement and analysis are

1. Low response time- “the fastest MapReduce job takes 15+ seconds” [2].
2. Hadoop NameNode was a single point of failure which was solved in later version of Hadoop 2.0.0 with passive NameNodes.
3. Multiple NameNodes required to get high availability [2].

1.3 *nfdump* [8]

nfdump provides a set of tools to capture and analyse NetFlow packets. The set of tools are :

1. *nfcapd*
2. *nfdump*
3. NfSen

nfcapd reads data from the network and stores them into the disk. It also rotates the file to limit size of file. *nfdump* allows *tcpdump* style filter expression for processing NetFlow data stored by *nfcapd* and displaying them on terminal or writing them into a file. NfSen gives a graphical overview of NetFlow data using RRDTool [11];

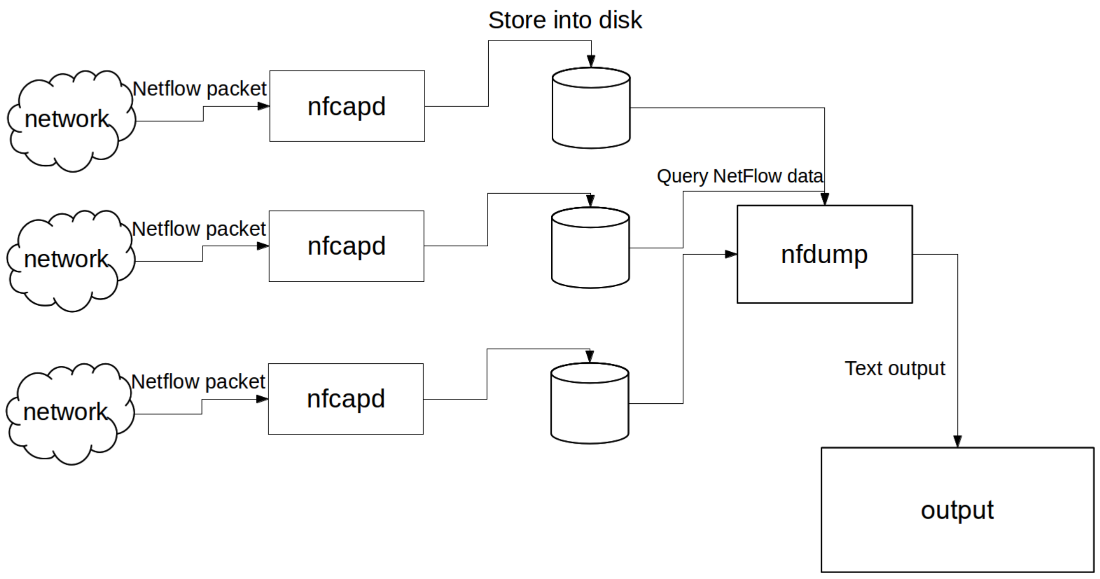

Figure 1.2: Architecture of *nfdump*.

1.3.1 Advantages and Limitations

nfdump is suitable for small scale NetFlow analysis. In case of large scale NetFlow monitoring disk file based storage is not sufficient.

1.4 Packet Forwarding with *netmap*[20]

This section describes about performance shortfalls of OpenVswitch [9], a study done by Luigi Rizzo and his team. *netmap* [19] is a framework to reduce the cost of moving traffic between the hardware and the host stack. As our network infrastructures moving from Fast Ethernet to Gigabit Ethernet and host stack APIs are not fast enough to handle such speed, framework like *netmap* is essential for such environment.

netmap achieves efficient I/O by removing all unnecessary run time costs and system calls. *netmap* exposes its API to userspace processes and provides a shadow copies of NIC rings. Memory regions containing *netmap* rings and packet buffers is shared with a process using *mmap*(). A user process can access NIC ring buffers directly, so transferring data can be done with zero copy operations. *netmap* has a small *libpcap* compatible API for migrating existing application.

OpenVswitch is a multi-layer, OpenFlow [18] compliant virtual switch designed to automate network through programmatic extensions. OpenVswitch is currently heavily used for network virtualization. So performance analysis of OpenVswitch is critical as we are modifying it to support scalable flow monitoring.

Authors have used *netmap* for *libpcap* alternative to improve the performance of OpenVswitch from 65 Kpps to 1.3 Mpps. After resolving few architectural limitations of OpenVswitch, authors have able to get three times performance improvement. Table 1.1 shows the performance of OpenVswitch under various environment.

So from this table we can easily find impressive performance gain by OpenVswitch

Configuration	**Kpps**
Linux userspace	50
FreeBSD userspace	65
Linux Kernel	300
Optimize OpenVswitch + FreeBSD	790
Optimize OpenVswitch + FreeBSD + *netmap*	3050

Table 1.1: OpenVswitch Performance with *netmap*

using *netmap*. This paper [20] also gives us internal architecture of OpenVswitch that helped to modify OpenVswitch for our second approach to have scalable flow monitoring with $Vswitch$.

1.5 Summary

In this section I have discussed available solutions for scalable flow monitoring. The issues with current solutions are

1. Lack of scalable storage [17] [8].
2. Lack of high availability data storage [16].
3. Real-time flow analysis is difficult with current systems [16].

In the next chapter I describe the design and implementation of modified *ntop*(a NetFlow collector) that tries to solve a few of these issues.

Chapter 2

Cassandra plugin for *ntop*

2.1 Overview

In this chapter I describe about *ntop*, Cassandra and the implementation of Cassandra plugin for *ntop*. I also specify the design and implementation of Cassandra client for C that was developed using C-python API.

2.2 *ntop*

ntop is an open-source traffic measurement application written in C. *ntop* design follows the UNIX philosophy: applications can be divided into small independent pieces that co-operate to achieve a common goal. *ntop* has the following modules:

1. Packet Sniffer - Capture packets using *libpcap* library and also from UNIX sockets.

2. Packet Analyzer - Analyze packets captured by Packet Sniffer.

3. Traffic Rules - *ntop* allows traffic rules for capturing packets to filter out unnecessary packets.

4. Report Engine - Report Engine displays analyzed output in an interactive web-based user interface.

5. Plugins - Using plugins anyone can extend *ntop* to support extra features.

2.2.1 Packet Sniffer

Packet Sniffer captures packets using *libpcap* library and stores them into internal buffer. This helps reduce packet drops in a busty traffic environment. *libpcap* is supported by all major Operating Systems, that allows *ntop* to be portable to Windows and UNIX variants.

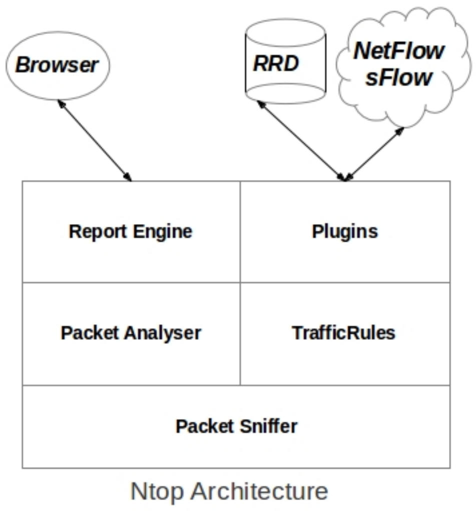

Figure 2.1: Architecture of *ntop*.

2.2.2 Packet Analyzer

Packet Analyzer gets packets from Packet Sniffer and processes those packets. Sniffed packets contains information about status of the network and that information is calculated by Packet Analyzer and then stored in RRD for future references.

2.2.3 Traffic Rules

ntop allows user to specify what kind traffic a user is interested in. Using *libpcap* filter expression *ntop* achieves this goal. Traffic Rules help *ntop* to reduce some burden on memory as well CPU, make *ntop* faster as it processes less number of packets.

2.2.4 Report Engine

ntop contains a web server by which users from any geological location can monitor their network. Report Engine provides a beautiful user interface with time series graph drawn using RRDTool. Using Report Engine a user can change the behavior of *ntop* by changing its configuration parameters.

2.2.5 Plugins

ntop has flexible design that allows user to add their own plugins. At startup *ntop* searches shared libraries (like .so, .dll files) to load plugins. A plugin can access *ntop*'s global data structure and can use API exported by *ntop*.

2.3 Cassandra Database

Cassandra is a highly scalable and highly available database initially developed by Facebook using two famous approaches: GFS [15] from Google and Dynamo [14] from Amazon. Cassandra is highly used in ebay and Netflix.

Cassandra's big data features are []

1. Elastic scalability.
2. High availability.
3. Distributed database design with no single point of failure.
4. Blistering linear performance.
5. Multiple datacenter based data distribution.

2.3.1 Cassandra Architecture:

Cassandra has a peer-to-peer distribution model. Therefore all nodes in a Cassandra cluster are symmetric. Any node can take read/write requests from clients and forwards them to correct node on the cluster. This design makes Cassandra scalable. Cassandra uses Gossip [22] protocol for intra-ring communication to support decentralization and partition tolerance. Gossip protocol helps to detect failures and recovery Cassandra node as well. Figure 2.2 describe about Cassandra ring.

Cassandra Data Model: Cassandra uses dynamic schema, column oriented data model. A database in a Cassandra named by keyspace consists of column families. A column family is a group of key-value pairs sorted by column name. A node in Cassandra cluster gets a token range at start up and also gets ownership of all

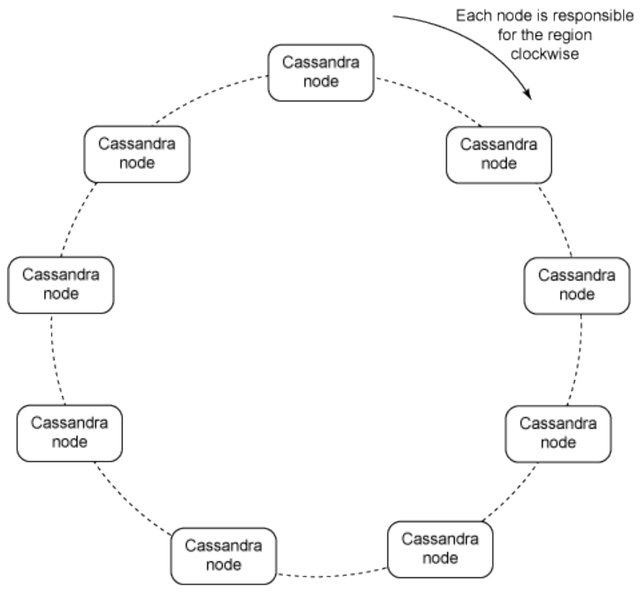

Figure 2.2: Cassandra Ring [6]

keys that fall into the token range. Keys in a column families sorted according to the partition of the Cassandra cluster. There are two partitioning scheme for a Cassandra cluster.

1. Random Partitioner: It uses 128 bit MD5 hash value of key to get a token. It is default partitioning strategy that helps automatic load balancing as md5 hash value ensure key distribution.

2. Ordered Partitioner: It uses byte order of key as token. It allows range scans over rows.

Figure 2.3 describes Cassandra data model.
Below table compares RDBMS schema with Cassandra data model.

RDMS	**Cassandra**
Database	Keyspace
Table	Column Falily
Entity	Column Name
Value	Column Value

Table 2.1: Comparison between RDMS schema with Cassandra Data Model

Memtables, SSTables and commit log: Cassandra uses in-memory data structures called memtable for caching recent writes to a Cassandra node. Memtables are

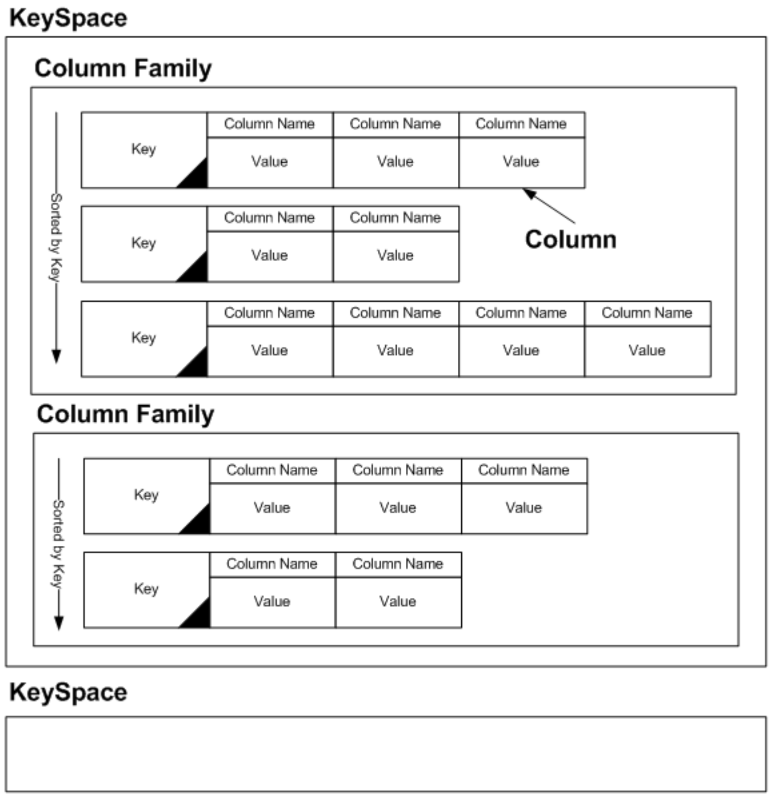

Figure 2.3: Cassandra Data Model

later stored into disk as SSTables depending on timeout or memtable threshold size. In a write operation a Cassandra node first writes into commit log then to memtable. Data recovery is done by commit log. In case of reading Cassandra maintains row cache key cache to make read operation faster. Figure 2.4 describes read/write operations in a Cassandra node.

2.3.2 Why Cassandra

In one of our testbeds we are able to generate approximately 1000 NetFlow packets per second with only two nodes having 1 Gbps NIC card. The table 2.2 provides the amount of data generated by our testbed.

Packets generated	Time	Size of generated data
1000	1 second	1.4MB
60 k	1 minute	85 MB
3.6 m	1 hour	5 GB

Table 2.2: Data generated by two node testbed.

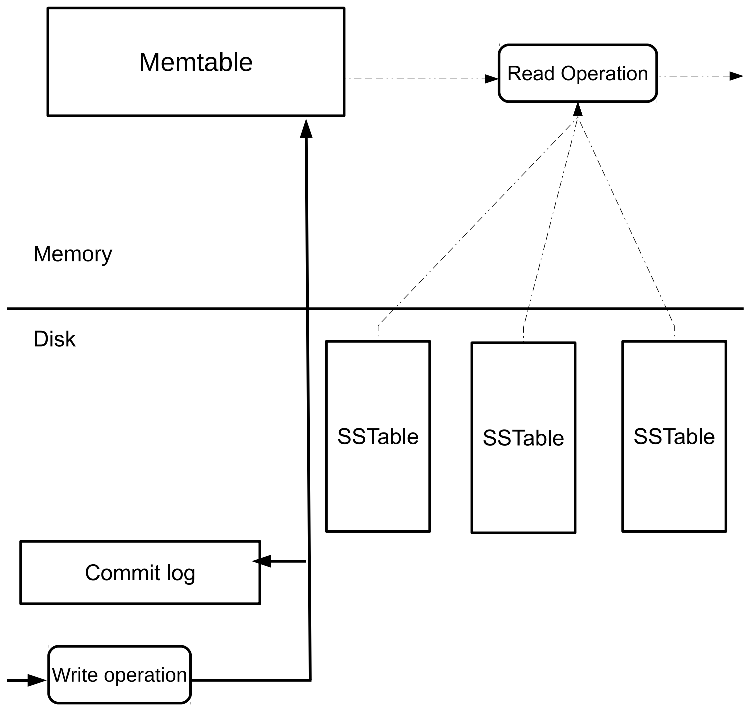

Figure 2.4: Cassandra Read/Write operation

It is clear that we cannot use RDMS based databases for storing flows in data center networks as they create explosive amount of data in a short time.

The features required of a database for flow monitoring are:

1. Scalability : so that huge amount of data can be stored according to the demand.
2. High write throughput : As flow records are generated at a rapid pace in data center networks, a flow monitoring system needs to write them fast.
3. Reasonable read performance : For real-time flow processing.
4. MapReduce support: For offline flow processing it needs to support MapReduce to enable massive data crunching operation.

Redis: Redis is written in C. It has fast read-write performance but is not scalable. Redis cluster, which is expected to support scalability, is going to be launched by the end of 2013 [10].

HBase: HBase is written in Java. It is scalable, has good read/write performance but is suitable only for batch processing. It has a single point of failure with Hadoop NameNode due to which HBase may lose data.

Cassandra: Cassandra meets all the requirements that are needed.

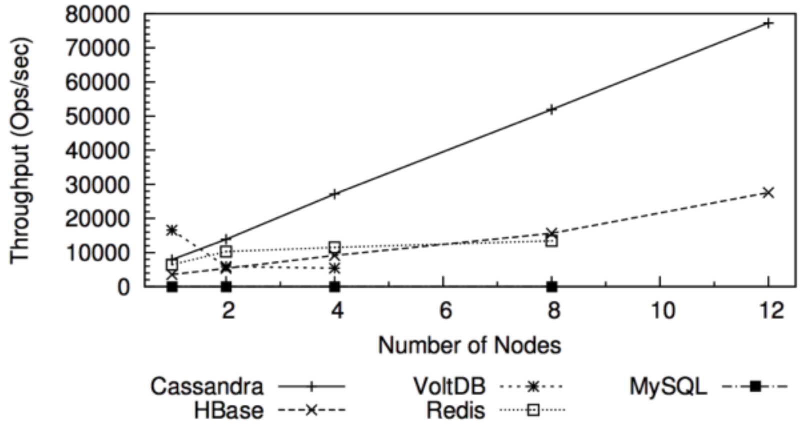

Figure 2.5: Cassandra Performance [4].

2.4 Cassandra C Client

Cassandra C client is a wrapper developed by me around Cassandra Python client, *Pycassa*. This allows any user to call Cassandra API from a C program. Pycassa uses Thrift RPC call to communicate with Cassandra. Fig. 2.6 shows the architecture of the Cassandra C API.

Cassandra C client API has three major classes of APIs. These are

1. Schema Manipulation API: Manages schema definitions of Cassandra.
2. Data Manipulation API: Inserts or retrieves data from Cassandra.
3. Utility API: API that do some common useful work.

2.4.1 Schema Manipulation API

These APIs allow one to create or delete keyspace and column families.

1. **cassandraCreateKeyspace**(name, server, rf, strategy): Create a keyspace on the server with the given replication factor(rf) and strategy. Supported strategies are :

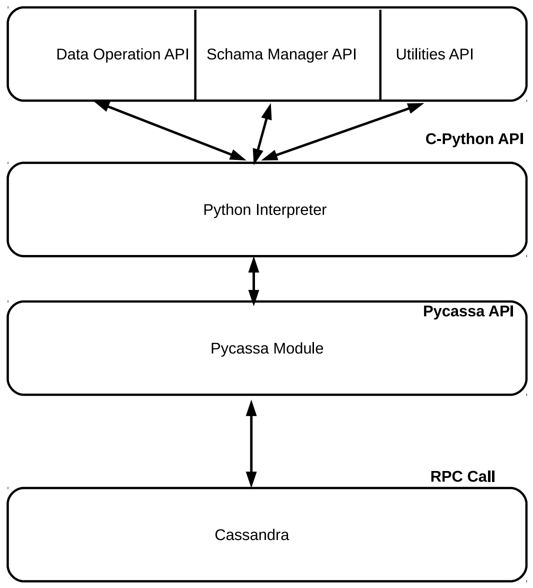

Figure 2.6: API interaction with Cassandra

   (a) SimpleStrategy : Replication strategy that simply chooses consecutive nodes in the ring for replicas.

   (b) NetworkTopologyStrategy : Replication strategy that puts a number of replicas in each data center.

   (c) OldNetworkTopologyStrategy : OldNetworkTopologyStrategy is provided for backwards compatibility with old Cassandra installations.

2. **cassandraCreateColumnFamily**(name, keyspace, comparator, server): Create a column family on the keyspace with the given comparator. Supported comparators are :

   (a) AsciiType

   (b) DoubleType

   (c) IntegerType

   (d) BytesType

   (e) LongType

   (f) FloatType

   (g) TimeUUIDType

3. **cassandraDropColumnFamily**(name, keyspace, server): Delete the given column family, *name*, from the keyspace.

4. **cassandraDropKeyspace**(name, server): Delete keyspace, *name*, from the server.

2.4.2 Data Manipulation API

These APIs deal with insertion and retrieval of data from the Cassandra server.

1. **cassandraConnect**(keyspace, columnfamily, server): Create a connection and store the connection object in the internal Python dictionary for later use. As connection creation takes time, connection objects are cached in the Python dictionary.

2. **cassandraInsert**(id, key, value, column name): Insert a key-value pair in the column family referred by *id*.

3. **cassandraTimeSeriesInsert**(keyspace, cf, server, key, column_name, value, bkt_size): cassandraTimeSeriesInsert is a simplified way of inserting time series data into Cassandra with predefine data model. It takes all required arguments and performs the write operation. The argument *bkt_size* specifies size of the bucket. A Python dictionary maintains most recent connection objects to make write fast. Figure 2.8 describes about the data model.

4. **cassandraGet**(id, key): Return a dictionary of key-value pairs from the column family referred by *id*.

5. **cassandraGetItem**(dictionary, position, name, value): Return *name* and *value* from given *position* of the *dictionary*.

2.4.3 Utility API

These APIs provide general functionality that we need.

1. **isExistPycassa**(): Check existence of pycassa on the system.

2. **isExistKeyspace**(name, server): Check existence of keyspace on the server.

3. **isExistColumnfamily**(name, keyspace, server): Check existence of column-family, *name*, on the *keyspace*.

2.5 Cassandra Plugin for *ntop*

Cassandra Plugin is developed in C as a shared object. *ntop* uses shared objects principles to provide plugins, that allows dynamic linking of compiled shared objects to *ntop* to add extra features.

The APIs supported by *ntop* for plugin support are given below:

API name	**Details**
int(*IntFunct)(void);	Called at initialization of the plugin.
void(*VoidFunct)(u_char);	Called at termination of the plugin.
void(*PluginHTTPFunct)(char* url);	HTTP request handler function.

Table 2.3: APIs for Plugin Support

API of Cassandra Plugin: Given below are the APIs developed by me for a Cassandra Plugin for *ntop*.

1. **initCassandraFunct** (void): Initialize Cassandra Plugin. It checks for pycassa module, availability of the Cassandra server and then invokes cassandraMainLoop.
2. **termCassandraFunct** (u_char termNtop): Terminates Cassandra Plugin.
3. **handlecassandraHTTPrequest** (char *_url): It gets a web request from the web browser and serves them.
4. **cassandraMainLoop**(void) : This API does all major work described below:
   (a) Reads configuration files and initializes the data structure needed to store data.
   (b) Identifies sniffed interface, gets statistics from global variables and stores into Cassandra.
   (c) Identifies NetFlow socket, gets data from global variables and stores them to Cassandra.

2.5.1 Architecture

Figure 2.4 shows the architecture for Cassandra Plugin.

Cassandra Plugin for *ntop* stores all statistics in a Cassandra keyspace using cassan-

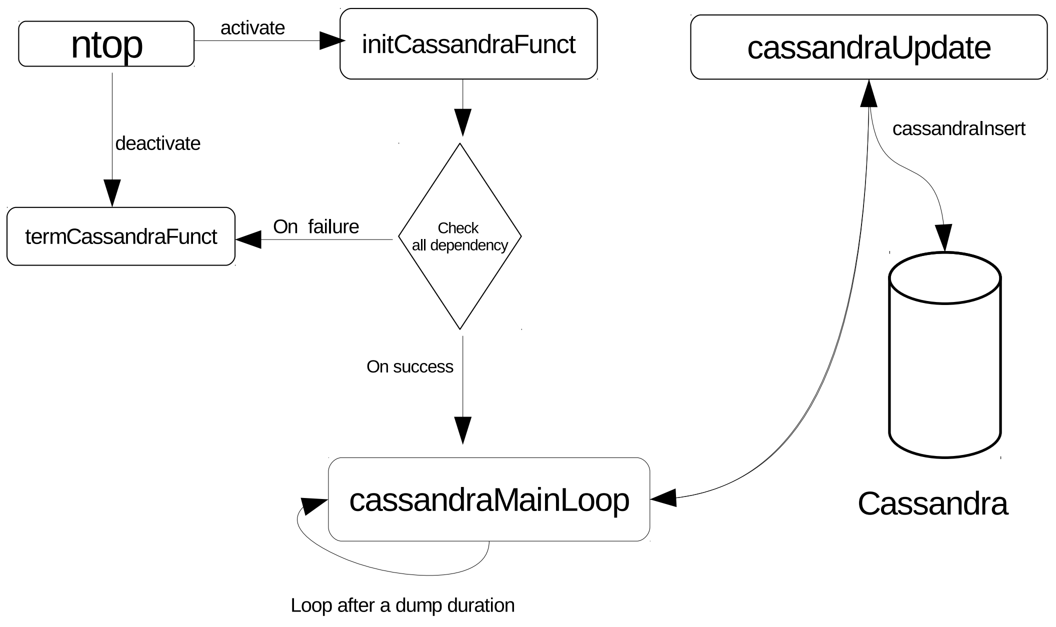

Figure 2.7: Architecture of Cassandra Plugin.

draTimeSeriesInsert() API. Figure 2.8 shows the data model used by cassandraTimeSeriesInsert() API. TimeUUIDType is data type for a key in Index column family. Unix timestamp is not useful when time series data events occurs at same time. TimeUUIDType is a unique value even for the events that generated at the same timestamp. This plugin going to take dump interval, bucket size for cassandraTimeSeriesInsert(), keyspace prefix from a configuration file. There will be one keyspace for every instance of *ntop*. Keyspace prefix helps to resolve any collusion. It is preferable to use uuid as keyspace prefix. *ntop* has multiple counters for every interface that it monitors. Counters belong to a interface stored into column family created with the name of the interface.

We have tested our Cassandra Plugin *ntop* as well *ntop* running without our plugins. We are sharing the results in the next section.

2.6 Results

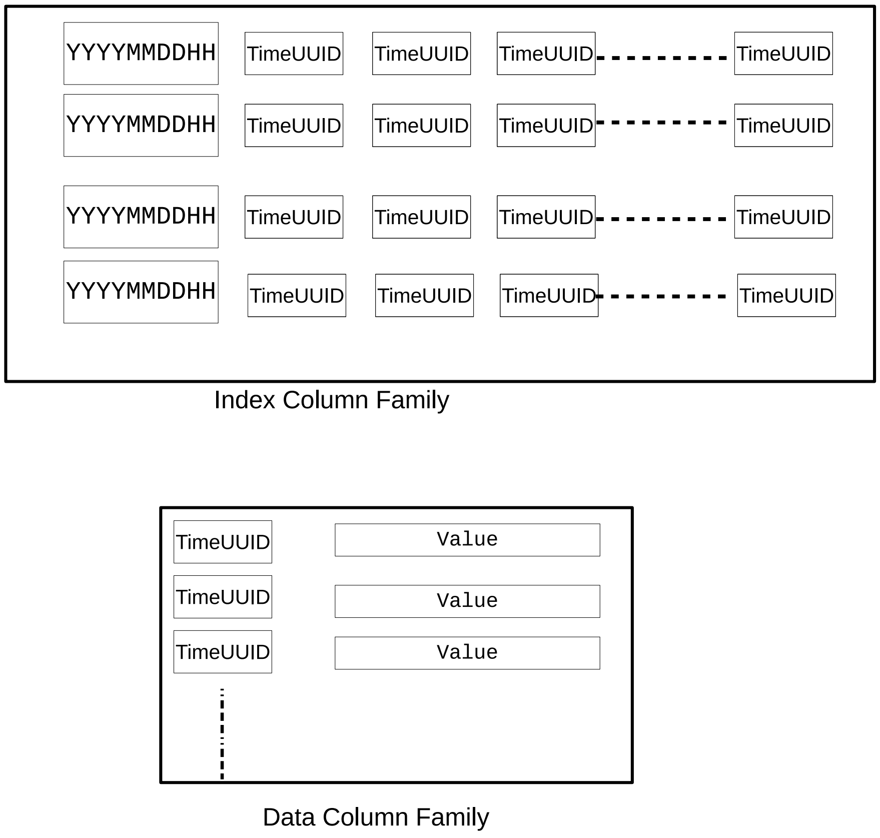

Figure 2.8: Data Model for Cassandra *ntop* Plugin.

Chapter 3

Scalable Flow Monitoring of OpenVswitch

3.1 Overview

OpenVswitch, a multilayer software switch that supports NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag as management protocols is been heavily used for network virtualization platforms. In this chapter I describe about design and implementation of OpenVswitch modification to support scalable flow monitoring. We have able to enable writing NetFlow data directly to the Cassandra database from OpenVswitch using shared memory and Python Daemon. The last section detail outs our findings of experiments that done on modified OpenVswitch.

3.2 NetFlow Monitoring

NetFlow is the de-facto protocol for flow monitoring. Routers maintains "flow cache" contains flow records that forwarded by the router. These flow records later exported to a NetFlow collector using UDP. Flows are identify by IP address, source and destination port, protocol and type of service. When a packet reaches one of its interface, it update "flow cache" if packet belongs to existing flows otherwise it creates a new flow. A flow is ended depending on these rules:

1. When a TCP RST or FIN received.
2. Active time out is over.
3. Inactive time out is over.

Name	**Description**	**Length in Bytes**
Version	Type of record format.	2
Count	Number of flow records contained.	2
SysUpTime	How long the system has been up and running.	4
Epoch	UNIX timestamp value when the packet was sent.	4
Nanoseconds	Residual nanoseconds after epoch second.	4
Flow Seen	Total number of flow seen since the exporter began emitting flow detail records.	4
Engine Type	User-configurable value (0-255) assigned to the exporter.	1
Engine ID	User-configurable value (0-255) assigned to the exporter.	1
Sampling Interval	First two bits hold the sampling mode; remaining 14 bits hold value of sampling interval.	2

Table 3.1: Netflow Header Format

4. When "flow cache" is over.

Other than flow key each flow records contains timestamps of first and last packet received, number of packet and bytes received, SNMP index. These flow records fields can be use for multiple analysis, like port numbers can be use for identifying application patterns and so on. Table 3.2 describes about NetFlow packet header anf table 3.2 describes about NetFlow record format.

OpenVswitch as NetFlow Exporter: OpenVswitch supports NetFlow protocol. OpenVswitch, a Openflow software switch that maintains basic flow statistics for each OpenFlow rules. Flow key in define by OpenFlow switch is different than flow key in general NetFlow enabled router or switchs. OpenFlow switches maintain "flow table", an entry in "flow table" contains three fields . These are :

1. A packet header that define flow tuple which can include upto 10 tuple in below.
   (a) In Port.
   (b) VLAN ID.
   (c) Source MAC.
   (d) Destination MAC.
   (e) Source IP.
   (f) Destination IP.
   (g) Ethernet Type.

Name	**Description**	**Length in Bytes**
Source Address	Source IP address.	4
Destination Address	Destination IP address.	4
Next Hop	IP address of next hop router.	4
Input	SNMP index of input interface.	2
Output	SNMP index of output interface	2
Packet Count	Total packets in the flow.	4
Byte Count	Total bytes in teh flow.	4
First	System up time when flow started.	4
Last	System up time when flow ended.	4
Source Port	TCP/UDP source port.	2
Destination Port	TCP/UDp destination port.	2
Pad1	Unused byte.	1
TCP Flags	Cumulative OR of TCP flags.	1
Protocol	IP protocol type (for example, TCP = 6; UDP = 17).	1
TOS	Type of service	1
Source AS	Autonomous system number of the source, either origin or peer.	4
Destination AS	Autonomous system number of the destination, either origin or peer.	4
Source Mask	Source address prefix mask bits.	4
Destination Mask	Destination address prefix mask bits.	4
Pad2	Unused	2

Table 3.2: NetFlow Record Format

    (h) IP Protocol.
    (i) Source IP Port.
    (j) Destination IP Port.

2. A action, which defines how the packet should be processed.

3. Statistics, which keep track of number of packets and bytes, time since last packet seen.

Using fields 1 and 3 a OpenFlow enable switches creates NetFlow packets. Though OpenVswitch calculates NetFlow in different way, it improves network visibility to 100% [13].

Limitations of NetFlow Analysis using on Collectors: NetFlow monitoring can be done either by storing flow records in some database and later analyzed by another application or NetFlow collector itself can collect and analyze flow records and store the analyzed data into database.*ntop* follows these analysis-store approach that is one of the reason why *ntop* losses packets on high speed network. Next section describes our proposed solutions to the problem.

3.3 Proposed Scalable Solution for Flow Monitoring

Currently OpenVswitch can exports NetFlow packets as UDP packets to a collector like *ntop* that receives NetFlow records then analysis them. UDP is connection less protocol, it does not guarantee reliable communication. Packets may get lost by *ntop* while processing because of socket buffer overflow. In previous chapter, our results shows that if *ntop* has less socket buffer it tent to losses packets. Our solution tries to avoid NetFlow collector all together allowing OpenVswitch host to store NetFlow records directly to scalable database, Cassandra. Now any NetFlow analysis tool can read flow records from the database and then analysis them. As OpenVswitch is distributed in nature writing flow records directly from OpenVswitch host to Cassandra provides scalable way to collect flow records. Cassandra is scalable and distributed database offer scalable storage

Bibliography

[1] **Apache Hadoop**. http://hadoop.apache.org/.

[2] **Apache Hadoop and Hbase**. http://www.slideshare.net/cloudera/sf-nosql2011/58.

[3] **Apache Hive**. http://hive.apache.org/.

[4] **Cassandra Performance Review**. http://www.datastax.com/dev/blog/2012-in-review-performance.

[5] **Cisco IOS NetFlow**. http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html.

[6] **Consider the Apache Cassandra database**. http://www.ibm.com/developerworks/library/os-apache-cassandra.

[7] **Mapreduce**. http://en.wikipedia.org/wiki/MapReduce.

[8] **nfdump Home Page**. http://nfdump.sourceforge.net/.

[9] **Open Vswitch: A Open Virtual Switch**. http://openvswitch.org/.

[10] **Redis Cluster**. http://redis.io/topics/cluster-spec.

[11] **RRDTool Home Page**. http://oss.oetiker.ch/rrdtool/.

[12] **sFlow Official Site**. http://www.sflow.org.

[13] **Traffic visibility play a role in Software-Defined Networking**. http://www.bradreese.com/blog/12-27-2012.htm.

[14] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. **Dynamo: amazon's highly available key-value store**. In *Proceedings of twenty-first ACM*

SIGOPS symposium on Operating systems principles, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[15] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. **The Google file system**. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[16] Yeonhee Lee and Youngseok Lee. **Toward scalable internet traffic measurement and analysis with Hadoop**. *SIGCOMM Comput. Commun. Rev.*, **43**(1):5–13, January 2012.

[17] Vijay Mann, Anilkumar Vishnoi, and Sarvesh Bidkar. **Living on the edge: Monitoring network flows at the edge in cloud data centers**. In *Fifth International Conference on Communication Systems and Networks, COMSNETS 2013*, 2013.

[18] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. **OpenFlow: enabling innovation in campus networks**. *SIGCOMM Comput. Commun. Rev.*, **38**(2):69–74, March 2008.

[19] Luigi Rizzo. **Netmap: a novel framework for fast packet I/O**. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, USENIX ATC'12, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association.

[20] Luigi Rizzo, Marta Carbone, and Gaetano Catalli. **Transparent acceleration of software packet forwarding using netmap**. In *Proceedings of the IEEE INFOCOM 2012*, pages 2471–2479, 2012.

[21] Rusty Russell. **virtio: towards a de-facto standard for virtual I/O devices**. *SIGOPS Oper. Syst. Rev.*, **42**(5):95–103, July 2008.

[22] Rajagopal Subramaniyan, Pirabhu Raman, Alan D. George, and Matthew Radlinski. **GEMS: Gossip-Enabled Monitoring Service for Scalable Heterogeneous Distributed Systems**. *Cluster Comput*, page 2006.