

# In-network, Push-based Network Resource Monitoring

## Scalable, Responsive Network Management

Taylor Groves and Dorian Arnold  
University of New Mexico  
[tgroves,darnold]@cs.unm.edu

Yihua He  
Yahoo Inc.  
hyihua@yahoo-inc.com

### ABSTRACT

We present preliminary work from our experiences with distributed, push-based monitoring of networks at Yahoo!. Network switches have grown beyond mere ASICs into machines which support unmodified Linux kernels and familiar user interfaces. These advances have enabled a paradigm shift in network monitoring. In lieu of traditional approaches where network diagnostics were delivered via SNMP we utilize Sysdb of Arista's EOS to implement a push based approach to network monitoring. This leaves the individual switches in charge of determining what monitoring data to send and when to send it. With this approach – on-switch collection, dissemination, and analysis of interfaces and protocols become possible. This push based approach reduces the feedback loop of network diagnostics and enables network-aware applications, middleware and resource managers to have access to the freshest available data.

Our work utilizes the OpenTSDB monitoring framework to provide a scalable back-end for accessing and storing real-time statistics delivered by on-switch collection agents. OpenTSDB is built on top of Hadoop/HBase, which handles the underlying access and storage for the monitoring system. We wrote two collection agents as prototypes to explore the framework and demonstrate the benefits of push based network monitoring.

### Categories and Subject Descriptors

C.2.3 [Computer Systems Organization]: Network Operations—*Network monitoring*

### General Terms

Resource monitoring, Network management

### Keywords

Push-based monitoring, On-switch monitoring

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [Permissions@acm.org](mailto:Permissions@acm.org).

NDM'13, November 17-21 2013, Denver, CO, USA

Copyright is held by the owner/author(s). Publication rights licensed to ACM.

ACM 978-1-4503-2522-6/13/11 ...\$15.00.

<http://dx.doi.org/10.1145/2534695.2534704>.

### 1. INTRODUCTION

Traditionally, computer and computational science fields have been dominated by computation-intensive problems. In recent years, we have seen a dramatic rise in data-intensive problems, involving the transmission and analysis of massive volumes of data from large networks of sensors or other acquisition devices, simulations or social networks. In addition to new generations of algorithms and data management technologies, the efficient extraction, filtration, mining and knowledge discovery from these data require scalable approaches for network traffic engineering and quality of service (QoS).

Effective traffic engineering and QoS services rely on capabilities that enable localized as well as more holistic profiles of network interactions and performance. This means that the network (both its endpoints and intermediate points) must be instrumented with monitoring capabilities. Current network monitoring typically employs the Simple Network Management Protocol (SNMP). A major shortcoming of SNMP is that it is not scalable for retrieving large collections of data. Wu and Marshall show that SNMP does not provide sufficient mechanisms to achieve payload efficiency when sending a stream of data from even small routing tables [11]. This inefficiency is due largely to extraneous protocol data and SNMP's pull-based approach.

To continue to be feasible and effective, network monitoring techniques must evolve to become more responsive and less intrusive. We propose a new distributed, push-based approach to network resource monitoring that promises to be more scalable, efficient and responsive than the traditional SNMP-based approach. Network switches have grown beyond mere ASICs into full machines that run unmodified Linux kernels with more standard interfaces and capabilities. We leverage these capabilities for on-switch or in-network information collection, dissemination, filtration, aggregation and analysis. Our push-based approach reduces the feedback loop of network diagnostics and enables network-aware applications, middle-ware and resource managers to have access to the freshest available data.

In this paper, we detail our motivations, approach and preliminary experiences with this new approach. Our prototype framework utilizes the OpenTSDB framework [6]; in this environment, we implemented two basic data collection agents to demonstrate the benefits of in-network, push-based network monitoring. Our preliminary results demonstrate performance benefits and show the feasibility of extending on-switch monitoring to production systems.

The organization of the rest of this is as follows: in the

next section, we detail traditional network monitoring approaches and describe their limitations and shortcomings. Then, in Section 2, we describe our motivation for moving away from traditional SNMP approaches and towards on-switch push-based monitoring. In Section 3 we describe our initial experiments and results with this prototype. Finally, in Section 4, we describe the implications of our current results and our future research plans to continue the exploration of this new network monitoring paradigm.

## 2. MOTIVATION

Traditionally, SNMP has been used for collecting information about network devices and protocols. SNMP data collection can be divided into two approaches: a pull-based approach, where managers query SNMP agents for system information and a push-based approach, where SNMP traps are triggered, leading to an asynchronous communication from the agent to manager. Both of these methods rely on an universally defined data structure called the Management Information Base (MIB). While the MIB guarantees a universal view of network devices and protocols, its usage often leads to unnecessary overhead.

SNMP traps are a feature of the protocol meant to communicate asynchronously from the SNMP agent to manager. Traps are classified into 6 generic types: coldStart, warmStart, linkDown, linkUp, authenticationFailure, egpNeighborLoss, and enterpriseSpecific. Usage of SNMP traps have traditionally been focused on providing coarse grained information about device status. Though custom traps can be developed within the enterpriseSpecific domain, SNMP trap still suffer much of the same structural overhead since both managers and agents must parse the MIB on message transmissions and receipts.

One of the major operational issues with SNMP in a large corporation, for example Yahoo, is that polling SNMP on network devices could result in very high CPU utilization for the device. This high CPU utilization may delay or even halt important information processing, such as route re-calculation. By using a push model, we do not have to restrict ourselves to SNMP polling once every  $x$  minutes. Instead, we can have the device report data with the pace at which that data changes and such that it does not impede other more important jobs, that is, smoothing the CPU spikes.

One other issue with SNMP is that certain components of the protocol are not very efficient. For example, a router normally stores its route table in a hashed format in order to conduct fast subnet-to-route lookups. However, SNMP responses for the route are required to be returned in lexicographical order per RFC 1213. Therefore, for each SNMP request the router receives, the hash table must be sorted lexicographically before a SNMP response (protocol data unit) can be built [1]. The larger the route table, the more CPU intensive the sort. On the other hand, by using a push model, we are more flexible and do not have to make the expensive and unnecessary sort before reporting the route data.

Real world usage of SNMP typically suffers from an additional deficiency in terms of reliability and robustness to failure of management nodes. All system information is delivered from the SNMP agents to the SNMP manager. If a manager node fails, the agents cease to report the information needed for monitoring. Though distributed approaches

to management have been explored [16], in practice SNMP connections are formed by agents reporting to a single management node [15].

One alternative to SNMP collection is to perform push-based monitoring from the switch into existing, scalable monitoring solutions. Modern switches have become more powerful, hosting multi-core CPUs, in addition to several gigabytes of memory and optional solid state storage. These switches can now be leveraged to enable intelligent reporting of metrics without the burden of traditional pull based solution and SNMP overhead. We believe that by utilizing on-switch monitoring network-aware applications, middle-ware and data managers will have access to the freshest available data. This is possible due to lower overheads in processing, a shorter feedback loop and intelligent collection.

### 2.1 Next generation network clusters

In the era of cloud computing, there has been an significant demand in inter-node communication bandwidth within a data center. Many applications in large Internet companies consist of thousands of distributed nodes. Efficient communication among these nodes is critical to the performance of these bandwidth-bound applications. For example, Yahoo has been running one of the largest Hadoop clusters [2] consisting of more than 10,000 nodes. Each of these nodes must perform significant data shuffling with other nodes in the same cluster to transport the output of the map phase before the reduce phase can be performed.

Unfortunately, communication bandwidth in large clusters may become oversubscribed by a significant factor in traditional network architectures. To help mitigate such issues, a breed of next generation networks [8, 14, 10, 12, 13] have emerged in recent years. The newly proposed networks leverage a large number of commodity Ethernet switches and provide a very low over-subscription rate for all connected hosts.

However, these new designs still pose significant challenges, particularly for network monitoring, managing and troubleshooting. The number of switches and the number of links among them are significantly larger than in their counterpart traditional networks. If any component of a cluster fails, the configurations on the rest of the devices may need adjustments in order to minimize the failure's impact and re-balance the traffic. This needs to be done in a short time frame, ideally, by an automated system. To meet these challenges, we need a scalable, fast and efficient monitoring system.

## 3. PROTOTYPE FRAMEWORK

### 3.1 The OpenTSDB monitoring framework

We chose to utilize the Open Time Series Database (OpenTSDB) as a scalable monitoring back-end [6]. OpenTSDB is a system built to store, serve and index system metrics on top of HBase [5] and is designed to handle billions of metrics per day. We selected OpenTSDB as our monitoring solution for several reasons. First, it is free and open source. Secondly, by utilizing a distributed data store such as HBase, it is capable of higher throughput than some traditional monitoring systems. Lastly was it's usage of *tags* to easily organize metrics. OpenTSDB can be organized by it's three key components – Time Series Daemons (TSD's), on device collectors, and a distributed data store as seen in Figure 1.

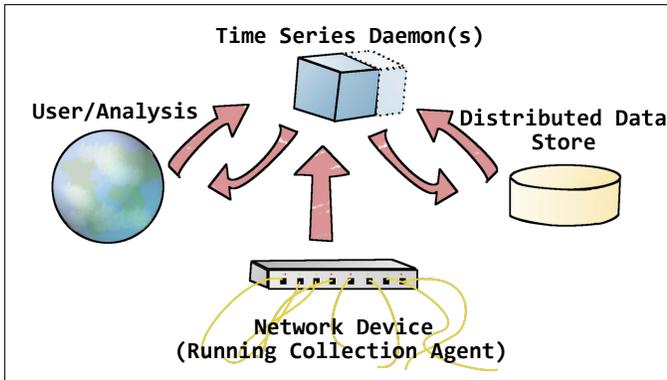


Figure 1: A diagram representing our usage of the OpenTSDB monitoring framework.

TSD's are responsible for receiving metrics from device collectors and pushing those metrics into the distributed data store. The daemons also query the data store on behalf of the user and present the data via a web or command-line interface. In the event that a set of TSD's cannot adequately service a set of collectors the number of daemons can be increased to provide further scalability of data collection.

The collectors are scripts running on the device being monitored (in our case, virtual switches as we describe later). Collectors are responsible for retrieving and reporting time series to the TSD's, where a time series is a combination of four elements: a metric name, Unix time-stamp, a floating point or 64-bit integer value and a set of tags. The tags provide a mechanism for filtering and classifying the data in a meaningful way. For example, a metric for inOctets might additionally have the tags hostname and interface. With these tags the TSD's could present the time series as either sums of input traffic across all hosts and interfaces or a specific interface on a single host. OpenTSDB's related collector package [7] provides added functionality to all the user written collectors, such as on-device deduplication, where devices delay reporting metrics that haven't changed since their last posted update. Preprocessing like this benefits the monitoring system by saving bandwidth and extraneous computation by the management system. In future work we wish to explore other types of preprocessing, as we believe on-switch preprocessing may be a useful avenue for finding savings in network bandwidth and data management.

Lastly, the distributed data store, HBase, is what provides the underlying scalability of the system. By utilizing Hadoop and HDFS [3, 4], HBase supports querying and writing tables with billions of rows and millions of columns.

For the purposes of our prototype cluster all of the collectors, virtual switches, TSD's and distributed data store were hosted on a single physical machine. It should be made clear that this will not be the case when the system is put to real use.

### 3.2 Interface and routing collectors

We implemented two prototype collectors for our initial testing on our virtual cluster, *eos\_interface* and *eos\_routing*, which record interface and BGP metrics, respectively. Both collectors were written to take advantage of Arista's EOS operating system. The *eos\_interface* script collects interface statistics and counters by utilizing the information stored

Name	Tags
intf.outBroadcastPkts	host, interface
intf.outUcastPkts	host, interface
intf.inMulticastPkts	host, interface
intf.outErrors	host, interface
intf.inBroadcastPkts	host, interface
intf.outOctets	host, interface
intf.outDiscards	host, interface
intf.inOctets	host, interface
intf.inUcastPkts	host, interface
intf.inErrors	host, interface
intf.inDiscards	host, interface
intf.outMulticastPkts	host, interface
intf.inBitsRate	host, interface
intf.outBitsRate	host, interface
intf.outPktsRate	host, interface
intf.statsUpdateTime	host, interface
intf.inPktsRate	host, interface
bgp.msgrcv	host, local AS, neighbor, neighbor AS
bgp.msgrsent	host, local AS, neighbor, neighbor AS
bgp.up_down	host, local AS, neighbor, neighbor AS
bgp.state	host, local AS, neighbor, neighbor AS
bgp.pfxrcd	host, local AS, neighbor, neighbor AS
bgp.summary.num_nodes	host, local AS, neighbor, neighbor AS
bgp.summary.num_routes	host, local AS, neighbor, neighbor AS

Table 1: Table showing the different metrics collected by the interface and routing collectors. The tags represent different methods of filtering data with respect to each metric.

in Arista's Sysdb. At the time, the BGP routing statistics were not available via Sysdb, so we pulled them using the show commands from the command line interface (CLI). We deployed each collector as extensions onto the Arista switches by packaging each script in the required SWIX format and copying them into the switch's flash storage. Once a script is running as an EOS extension, there are mechanisms to daemonize or immortalize the extension so it is restarted in the event of failure. To ensure that the extensions persist between switch restarts they are added to the boot-extensions file. Our interface collectors ran every 15 seconds on each switch, while the routing collector ran every 60 seconds. Each metric uses the OpenTSDB tag system to provide custom filtering. This facilitates flexible examination of the system metrics, providing both macro and micro views of the system metrics. A complete list of the statistics and counters collected with their tags can be seen in Table 1.

### 3.3 Virtualized environment

In order to experiment with our monitoring solution, we built a virtualized cluster comprised of virtual switches and connected them with Ethernet bridges. Arista provides a slightly modified switch operating system (vEOS) image to their customers. This vEOS image can be run on a number of common hypervisors such as QEMU-KVM, VMware Player, VirtualBox and etc. To create a cluster, we use libvirt to instantiate 24 such virtual switches with Arista's vEOS image. As depicted in Figure 2, these switches are divided into two virtual chassis (VCs) and a group of 8 Top of Rack switches (TORs). Each VC consists of 4 spine switches and 4 leaf switches. Each of these switches has 16 in-band data ports and 1 management port. In order to connect these virtual switches, Ethernet bridges are created within

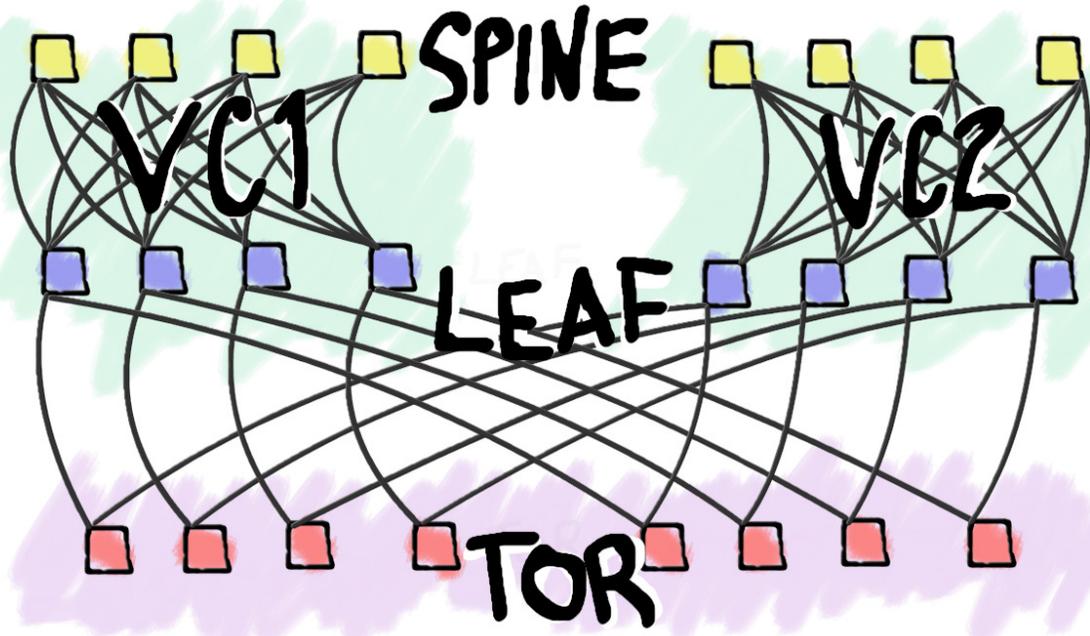


Figure 2: A visual representation of the 24 switches that make up the virtual cluster, divided into leaf, spine and top of the rack (TOR) nodes.

the host machine between any two ports where we would run a cable in real world.

Once the virtual switches and Ethernet bridges are instantiated, we can configure the virtual cluster and run routing protocols the same way as if we configure clusters with real switches. In this particular virtual cluster, we assign IP addresses on each of the interfaces on the virtual switches and run stock BGP protocol among them.

The virtualized environment provides a useful testbed to prototype on-device collection without provisioning the actual hardware. To set this up on dedicated hardware, we would require 24 switches and hundreds of network cables. With this virtualized cluster we can conveniently adjust the link quality or emulate real world problems. Some of this emulation would be more difficult on physical switches where traffic is passed through the ASIC directly.

### 3.4 Prototype results

Even though the CPU performance of virtualized switches does not necessarily reflect the expected performance of physical switches, we can inform the reader that for our cluster the CPU utilization peaked to approximately 15% on the virtual switches and these peaks lasted for under a second as collectors gathered necessary metrics. On dedicated switches we expect the performance to be better, since our 16 core machine was hosting 24 virtual switches, the TSD and the distributed data store. The performance overhead on dedicated machines will be explored in detail in future work.

From the front-end web interface, we saw responses at a varying resolution of a milliseconds upwards to around two seconds. This was highly dependent on the amount of data points queried. This response should be greatly improved

when the data store becomes distributed across multiple physical nodes. On a real system, the latency will change as RPC will be made to separate nodes, rather than hosting all nodes on the same physical machine. Furthermore, the TSD and collectors would have independent computational resources. In our virtual cluster, the HBase nodes were hosted on the same physical node as the TSD's and collectors. In a real system this will be distributed to provide better performance. Despite the over subscription of virtual machines to cores we were able to retrieve metrics at a scale and frequency not practical with current SNMP based systems. Extensive comparisons of scalability and frequency will be explored in future work.

To evaluate our prototype network, we performed a series of tests where traffic flow was generated through the network via an iperf server and client. During this time approximately 2000 metrics were collected every 15 seconds, corresponding to the combination of metrics and interfaces on each switch. We saw no trouble in keeping up with this rate, despite the over subscription of virtual machines to cores. Using the OpenTSDB monitoring solution in combination with on-switch collectors we were able to see a myriad of behaviors. In Figure 3 we were able to see the shift in flow as traffic pivoted from spn1-2-vma to spn3-2-vma. We present this change in flow in Figures 4 and 5. We believe this was likely due to the fact that the virtual switches lack the hardware to consistently map communication flows that an ASIC would normally provide. In additional testing we used traffic controller (tc) combined with netem to emulate packet loss on specified interfaces throughout the network. We were able to observe how this leads to a breakdown of the BGP topology, leading to an eventual loss of application traffic and we were able to observe this in real-time. If

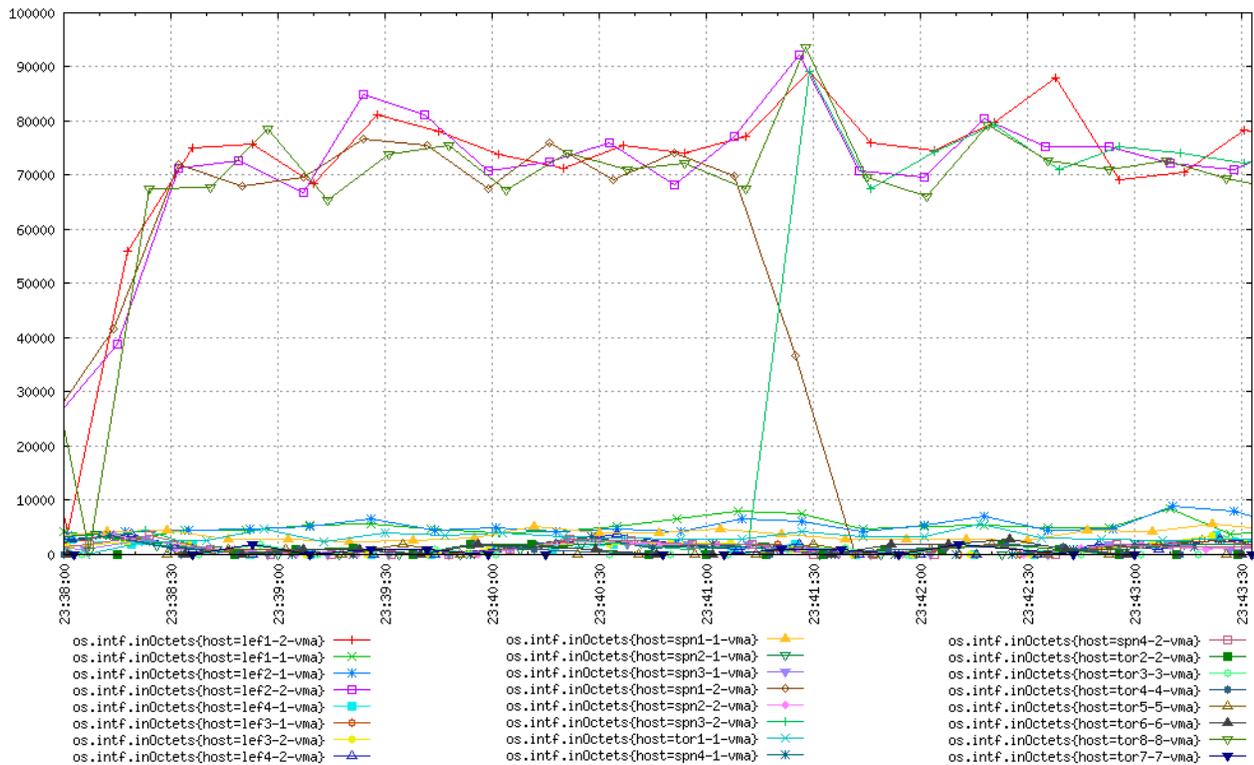


Figure 3: This chart represents an observed data flow from TOR1-1 to TOR8-8. This chart captures the transition from using the intermediate switch SPN1-2 to SPN3-2. The y-axis is the inOctets rate per 15 seconds.

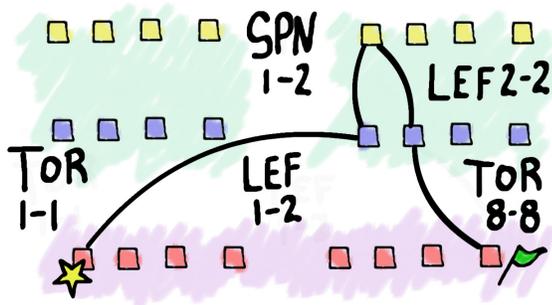


Figure 4: This graphic demonstrates the original flow observed from TOR1-1 to TOR8-8.

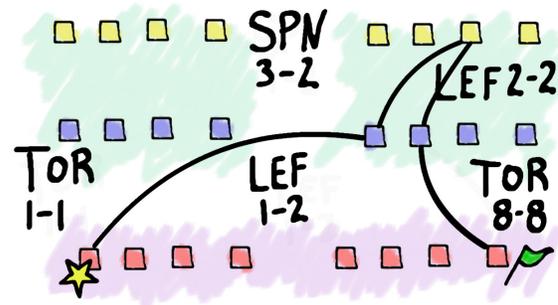


Figure 5: This graphic demonstrates the adjusted flow from TOR1-1 to TOR8-8.

we had been using traditional SNMP approaches to monitor the network, our view of network health would only be updated every 5 minutes. With an SNMP based approach, it is possible that we might have missed this event entirely. We believe that having this rapid access to network data can facilitate better network QOS in addition to providing better support to network-aware applications. In future work we want determine what kind of analysis a on-switch monitoring solution is capable of and explore what kind of failures this system can detect that previous solutions couldn't.

#### 4. CONCLUSIONS AND FUTURE WORK

As large systems continue to grow, network-aware appli-

cations middle-ware and data management will be crucial to efficient system usage. We believe that current strategies for network monitoring fall short in terms of scalability and performance. Furthermore, future network monitoring solutions will need to meet several criteria:

- They will need to provide on-device collection in order to achieve better performance over the network, devices and monitoring managers.
- Monitoring solutions will need to sit on top of a scalable, distributed data store to support an era of large systems and big data.
- Devices should to support in situ analysis of the data

to further distribute the workload and save resources.

With this context, our work demonstrates the potential of on-switch, push-based resource monitoring, which will be essential for future network-aware solutions. It is our belief that when a monitoring solution meets the above criteria it will provide numerous benefits to the user and the application: distributed intelligence, faster feedback loops, smoothing of performance spikes, decreased network usage and ease of use by leveraging existing large scale solutions (HBase, HDFS, Hadoop in our case). As a result, our solution is more scalable and accurate with less overhead on the device, while being easier to manage.

Lastly, this is a area rich with future work. Things we would like to explore in the future include, evaluations of network-aware applications and middle-ware. We want to explore in situ preprocessing, so that the switch reduces extraneous reporting of data. We need to evaluate the performance of monitoring overhead on physical switches and explore how we can smooth the performance spikes associated with collection. With this monitoring system in place, we can explore the real-time analysis of network QOS – examining the causes of congestion, downed links and other failures. All of this future work depends on scalable real-time monitoring.

## 5. REFERENCES

- [1] Ip simple network management protocol (snmp) causes high cpu utilization, "http://www.cisco.com/image/gif/paws/7270/ipsnmphighcpu.pdf".
- [2] Yahoo! launches world's largest hadoop production application, "http://developer.yahoo.com/blogs/hadoop/posts/2008/02/yahoo-worlds-largest-production-hadoop/".
- [3] Hadoop. <http://hadoop.apache.org>, 2013.
- [4] Hadoop file system. [http://hadoop.apache.org/docs/stable/hdfs\\_design.html](http://hadoop.apache.org/docs/stable/hdfs_design.html), 2013.
- [5] Hbase. <http://hbase.apache.org/>, 2013.
- [6] Open time series database (opentsdb). <http://opentsdb.net>, 2013.
- [7] Tcollector. <http://opentsdb.net/tcollector.html>, 2013.
- [8] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.
- [9] L. Andrey, O. Festor, A. Lahmadi, A. Pras, and J. SchÄunwÄd'lder. Survey of snmp performance analysis studies. *International Journal of Network Management*, 19(6):527–548, 2009.
- [10] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VI2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.
- [11] Q. Gu and A. Marshall. Network management performance analysis and scalability tests: Snmp vs. corba. In *Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP*, volume 1, pages 701–714 Vol.1, 2004.
- [12] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *SIGCOMM Comput. Commun. Rev.*, 39(4):63–74, Aug. 2009.
- [13] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 75–86, New York, NY, USA, 2008. ACM.
- [14] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39(4):39–50, Aug. 2009.
- [15] J. Schonwalder, A. Pras, M. Harvan, J. Schippers, and R. van de Meent. Snmp traffic analysis: Approaches, tools, and first results. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 323–332, 2007.
- [16] R. Subramanyan, J. Miguel-Alonso, and J. A. B. Fortes. A scalable snmp-based distributed monitoring system for heterogeneous network computing. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '00, Washington, DC, USA, 2000. IEEE Computer Society.