SDN-based concept for Network Monitoring

Vassil Gourov*
* Sofia, Bulgaria
vassilgourov@yahoo.com

***Abstract*- The deployment of increasing number of real-time services over communication networks raises essential issues for assurance of the quality of services, which requires a clear picture of the network performance. The availability of accurate statistics helps to estimate the traffic flows, to find service degradation due to congestion, as well as to optimize routing. Presently, for network measurement and monitoring are applied various methods which require separate infrastructure, and thus, higher expenses. Most methods are not capable to meet all monitoring requirements, some are not accurate or granular enough, others are adding network load or lack scalability. The paper provides a concept for using Software Defined Networking as a unified monitoring solution, by suggesting how to measure link utilization, packet loss and delay. Initially, some monitoring methods are presented, and the opportunity of using OpenFlow enabled topology in Software Defined Networking for monitoring. The paper proposes a monitoring concept for measuring link usage, packet loss and delay.**

I. INTRODUCTION

In last decades, Internet has evolved into a huge structure interconnecting thousands of networks, and its users have grown exponentially. The network has also experienced deep changes in the services provided and the usage patterns. Present trends towards Internet of Things (IoT) and Factories of the Future (FoF), as well as the development of several new applications and services (such as video streaming, social networking, online gaming, e-banking, e-business, etc.) have raised not only issues of interoperability, but also have added new control requirements and have significantly increased the network complexity.

Present-day networks are growing large and need to support a lot of new applications and protocols. Subsequently, their management complexity is increasing, which reflects on higher expenses due to maintenance and operations [2], as well as, due to human errors in network-downtime [3]. Certain new services (i.e. voice and video delivery) are not capable to operate in a best effort environment, where resources are socially shared, resulting in delays and losses. On the other hand, there is no guarantee that any new architecture would not result in a similar problem a decade from now. Every time when a new infrastructure capable of solving past problems is introduced, new problems emerge. A solution that is able to meet future requirements when they arise is needed, and it is considered that Software Defined Networking (SDN) may play an important role [1].

New Internet-based services need an environment capable to dynamically adjust to changing demands, and able to provide the best point-to-point connection. As the performance of applications depends on the efficient utilization of network resources, it is expected that the exponential growth of users and Internet traffic could create problems in provision of high quality of services (QoS) and meeting users' demands. Thus, accurate traffic measurement (TM) becomes a key aspect for network management, in order to reach QoS, ensure network security and traffic engineering (TE). Due to the large number of flow pairs, high volume of traffic and the lack of measurement infrastructure it has become extremely difficult to obtain direct and precise measurements in Internet Protocol (IP) networks [4]. Current measurement methods use too many additional resources or require changes to the infrastructure configuration, thus, bringing additional overhead. There is obvious a need to find a network management solution able to provide accurate, detailed and real-time picture of the network, being also cheap and easy to implement.

The main problem addressed in this paper is how to monitor network utilization efficiently in real time. The aim is to use SDN as a lever to meet future networking demands by designing a monitoring solution capable to measure network utilization, delay and packet loss and to evaluate it by using OpenFlow (OF) Protocol. Subsequently, the main research questions are [1]: How can monitoring be achieved with Software Defined Networking? What kind of improvements could SDN bring compared to present solutions?

This paper focuses initially on SDN specificity and the available network monitoring solutions. Next a conceptual architecture and a prototype are presented and the implementation of a network monitoring solution in a real test environment using the OF protocol. Finally, some monitoring concept evaluation results are presented.

II. NETWORK MONITORING METHODS

Measuring the traffic parameters provides a real view of the network properties, an in-depth understanding of the network performance and the undergoing processes. Network monitoring is crucial for QoS and assures that the network functions properly. The ability to obtain real traffic data makes possible to analyze network problems, generate traffic matrices, optimize the network using TE techniques or even upgrade it based on future predictions. Finally, a proper network overview allows the routing algorithms to take more appropriate decisions, increasing the resource utilization and decreasing the congested nodes/links [1].

Traditionally, different techniques are used for measuring the amount and type of traffic in a particular network. Generally, two distinct groups of measurement methods are applied: passive and active. The former counts the network traffic without injecting additional traffic in the form of probe packets, while the latter is achieved by generating additional packets. Both are useful

for network monitoring purposes and for collecting statistical data. Other methods focus on measurements on application or network layers of the Open System Interconnection (OSI) model. Network layer measurements use infrastructure components (i.e. routers and switches) to get statistics, whereas Application layer measurements are operating on the upper layer and are easier to deploy as they are application specific. The latter are more granular and could be used also for better service delivery, however, this method requires access to end devices, which Internet Service Providers (ISP) normally do not have [1]. It is important to note that OF provides means to implement any of the methods or combine them if needed, while traditionally every type of measurement requires separate hardware or software installations.

Today, different techniques are applied to measure link usage, end-to-end delay and packet loss. Some monitoring techniques use direct measurement approaches. For example, flow-based measurements such as NetFlow and sFlow [5] rely on packet sampling in order to ensure scalable real-time monitoring. This method, however, has some limitations linked to high overhead and unreliable data [6]. Deep Packet Inspection is heavily used within network monitoring for security reasons and also for high speed packet statistics. Unfortunately, few network devices support it, so very often additional hardware installations are required. Using DPI also creates a network bottleneck point. Another method is based on port counters: Simple Network Management Protocol counters are used to gather information about packet and byte counts across every individual switch interface [7]. Some of the limitations of this method are linked to the switch query frequency (limited to once every 5 minute), the overall resource utilization, the lack of insight into the flow-level statistics and hosts behavior, and thus, the lack of granularity of the monitoring information obtained [1].

Today, delay and packet loss data are mainly obtained by application measurements and a common practice is to use ping. It uses the round trip time (RTT) by sending a number of packets from a source node to a destination node and measures the time it takes for it to return back. For example, Skitter [8] uses beacons that are situated throughout the network to actively send probes to a set of destinations. The link delay is calculated by finding the difference between the RTT measures obtained from the endpoints of the link. However, using such a strategy to monitor the network delay and packet losses requires installing additional infrastructure, because every beacon is limited to monitor a set of links. Using this method accounts additional inaccuracy and uncertainties [1].

Passive measurements are widely used for packet and delay monitoring. An example of passive monitoring is given in [9] and consists of capturing the header of each IP packet and timestamp it before letting it back on the wire. Packet tracers are gathered by multiple measurement points at the same time. The technique is very accurate (microseconds), but requires further processing in a centralized system and recurrent collecting of the traces, which generates additional network overhead. Furthermore, every device needs accurate clock synchronization between every node. Another similar approach is used to measure packet losses [10]. It tags uniquely each packet when it passes trough the source node and accounts if it was received in the end node.

The OF protocol is capable of not only controlling the forwarding plane, but also to monitor the traffic within the network. OpenTM [11] estimates a TM, by keeping track of the statistics for each flow and polling directly from the switches situated within the network. The application decides which switch to query on runtime and converges to 3% error after 10 queries. In the paper presenting it, several polling algorithms are compared for a querying interval of 5 seconds [11].

In [12] an active measurement technique is suggested, whereas the authors use the fact that every new flow request has to pass through the controller. This allows routing the traffic towards one of multiple traffic monitoring systems, to record the traffic or to analyze it with an Intrusion Detection System.

For passive measurements in FlowSense [13] are used some features of OpenFlow in order the measurements to be evaluated from three prospectives: accuracy (compared to polling), granularity (estimate refresh) and staleness (how quickly can the utilization be estimated). FlowSense suggests gathering statistics passively based on the massage the controller receives once the flow has expired.

In [14] is suggested to implement a new SDN protocol for statistic gathering, whereas new software defined traffic measurement architecture is proposed. The authors implement five measurement tasks on top of an OpenSketch enabled network in order to illustrate the capabilities of this approach. The measurement tasks are detection of: heavy hitters (small number of flows account for most of the traffic), super spreader (a source that contacts multiple destinations), traffic changes, flow size distribution, traffic count.

A network monitoring system should be able to observe and display up-to-date network state. It is obvious that several monitoring solutions are already capable to do that in one or another way. However, in order to meet the specific challenges that ISPs face, the following design requirements should be considered in a new monitoring concept [1]:

- Fault detection - Whenever a link or node failure happens, the network monitoring system should be warned as soon as possible.
- Per-link statistics - ISPs require statistics for every link in order to assure QoS within the boundaries of their network, without bandwidth over-provisioning.
- Overhead - The proposed solutions should not add too much network overhead. The overhead should scale no matter how big the network is (as long as the controller can handle them) or the number of active flows at any moment. The component should be able to obtain statistics based on the routing information, thus, sending a query requests only to those devices that are currently active.
- Accuracy - A big difference between the reported network statistics and the real amount of used capacity should be avoided.
- Granularity - The system should be able to account for different type of services. It should be able to make distinction between flows that have specific needs, i.e. require special care (bandwidth, delay, etc.). Furthermore, it should make distinction between applications, as well as, clients.

Finally, the monitoring solution should reduce the amount of additional overhead generated in the network and device as much as possible, without too much degradation of the measurement accuracy.

III. MONITORING IN SOFTWARE DEFINED NETWORKING

A. Emergence of Software Defined Networking concept

Presently, the communication networks architecture rely on devices where the control plane and the data plane are physically one entity, the architecture is coupled to the infrastructure, and every node needs to be separately programmed to follow the operator's policies. In addition, the companies that provide network devices have full control over the firmware and the implementation of the control logic. Thus, the trends in networks development face operators with the challenges of meeting market requirements, and ensuring interoperability and flexibility. Generally, it could be summarized that the main constraints limiting networks evolution include [1]:

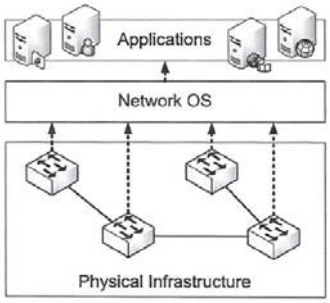

Figure 1. Basic SDN architecture [1]

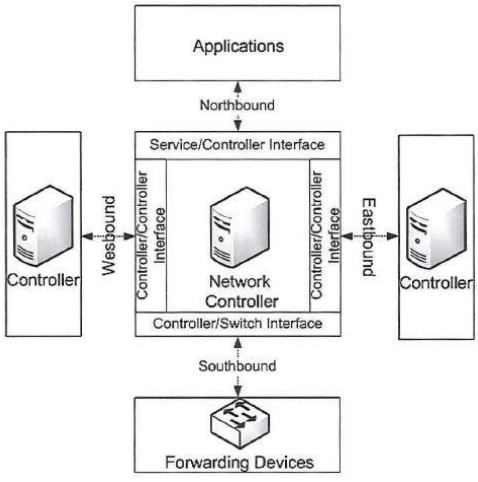

Figure 2. Scheme of OpenFlow controller [1]

TABLE I.
OPEN SOURCE OPENFLOW CONTROLLERS [1]

Name	Language	Developer
Beacon [1]	Java	Stanford University
Floodlight [2]	Java	Big Switch Networks
Maestro [61]	Java	Rice University
NOX [24]	C++, Python	Nicira
POX [33]	Python	Nicira

- complexity: The definition of many new protocols result in difficulties for operators to configure thousands of devices and mechanisms in order to reflect any changes in the network topology or implement a new policy [15].
- scalability: The exponential growth of data demands and the not predictable change of traffic patterns, as well as the emergence of cloud computing and several new applications increase the demand for bandwidth. The scalability problems emerge as networks are no longer capable growing at the same speed, and network providers could not endless invest into new equipment [15].
- dependability: The dependability on equipment vendors and the not sufficient inter-vendor operability face network operators with several difficulties to tailor the network to their individual environment.

Taking into account the need for a network that uses simple, vendor-neutral and future-proof hardware [11], on the one hand, and the ability of software to support all present network requirements (e.g. access control, TE), on the other, the SDN concept emerged as an option for more centralized control system of the whole network [1].

The SDN approach decouples the control plane from the network equipment and places it in a logically "centralized" network operating system (OS), referred to as controller. One way to achieve this is by using a protocol to interconnect the two separated planes, providing an interface for remote access and management. The SDN architecture (Fig. 1) varies with the implementation and depends of the type of network (i.e. data-centre, enterprise and wide area network) and its actual needs. The main idea behind SDN is to abstract the architecture and provide an environment, which would reduce the development time for new network applications and allow network customization based on specific needs. The main goals behind this architecture are to ensure [1]:

- interoperability: using centralized control over the SDN enabled devices from any vendor throughout the whole network;
- simplicity: to eliminate complexity issues and make the network control easier and finer grained, thus increasing reliability and security;
- innovativeness: with the abstraction of the network services from the network infrastructure the entire structure becomes much more evolvable, and network operators would easily tailor the behavior of the network and program new services faster.

The OF protocol is one way to implement the SDN concept and to manage interconnected switches remotely. This protocol allows the controller to install, update and delete rules in one or more switch flow tables, either proactively or reactively, to interconnect the forwarding with the data plane, and to enable part of the control operations to run on an external controller.

Since the controller is the most important element of the SDN architecture, it attracts a lot of efforts and a number of new controllers have been released (Example of some of them is presented in Table I). Its main task is to add and remove entries from the switch flow-tables. The controller (Fig. 2) interacts with a set of switches via OF using the Southbound interface. It is responsible for service and topology management, and could be enhanced with

additional features or could provide information to external applications. Currently, the Northbound communication is not standardized. Some efforts are made to enhance the abstraction level by designing network programming languages on top of the controllers [16], [17]. Via Westbound and Eastbound interfaces the controller is able to communicate with other controllers, several proposals for this interaction are available [18].

Despite SDN abilities to overcome some network problems, certain scalability limitations also exist as a result of the centralized SDN architecture, and the bottleneck that could be formed between the infrastructure and the controller. Some concerns are linked to a bottleneck with the switching equipment, in terms of forwarding capacity (table memory) and the overhead that could be created by constant reactive invocation of the control plane.

B. Concept for OpenFlow monitoring architecture

Using Software Defined Networking could solve some of the current monitoring problems in IP networks. Since, SDN is a new paradigm, some architectural aspects are still under investigation. In order to pay more attention to the research problems that were already outlined, the following two architecture assumptions are made based on study of similar concepts of scholars [1]:

- *First, one "centralized" controller manages all switches and handles all the control operations.*
- *Second, there are no scalability issues for the controller and the switches.*

In order to illustrate and confirm the monitoring abilities, a prototype is implemented as a Python application for POX [19] (a Python-based OF controller that can be used for fast implementation of network control applications). The OF monitoring application (Fig. 3) works as a Core component of the controller, therefore, it has access to all the available information, including routing decisions. It is also capable to directly interact with each switch that supports OF. The discovery component is responsible to build a graph representation of the network topology (topology view). A virtual switch instance is created for every new switch that connects to the controller, and each instance stores switch specific information.

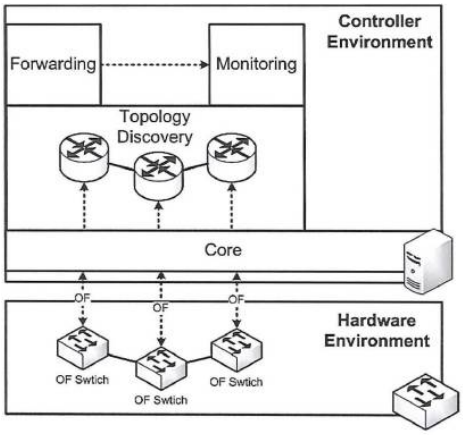

Figure 3. OpenFlow prototype [1]

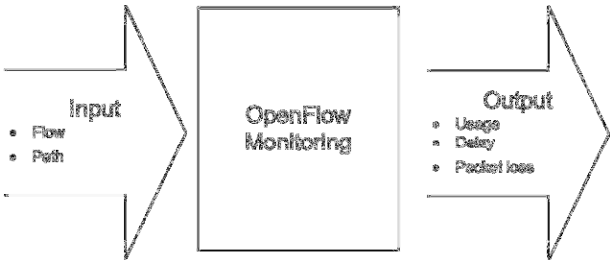

Figure 4. Basic diagram of the monitoring component [1]

How it works? The first thing every OF switch does, once it is started, is to establish a connection with the designated controller. The switch gives its state and link information. This allows the controller to keep a global up-to-date network view. Once a packet enters the network and no matching rule for it exists, it is forwarded towards the controller. The controller inspects the packet and determines how to handle it. Normally, the controller would install new flow entry in every switch table that needs and then return the packet to its source node. This means that the controller has topology view of the network and information about the active flows (IP source/destination, port source/destination, etc.) and the routes they take trough the network. Each switching device within the network contains activity counters, i.e. for OF there are separate table, port, flow and queue counters. The flow and route information should be used as input parameters of the monitoring component (Fig. 4). It is responsible to poll one or multiple network devices per flow, which in terms should return the requested information. Another option is to implement a passive measurement and wait for the switches to send statistics once the flow has expired. Every time the flow is no longer active for some time the switches may send message, indicating the utilization statistics for the flow. The author considers that the monitoring component should make use of the two statistical gathering approaches. The final output should be data for link utilization [1].

C. Monitoring concept

By using SDN to implement a network monitoring system some of the objectives given in II above are already met. Since every device communicates with the controller, there is real-time view on the network status, including links, nodes, interfaces, etc. Furthermore, it provides sufficient granularity and it is capable to monitor the utilization of every link within a given network without sampling any packet or adding more overhead to any of the switches.

OpenFlow allows granular view of the network, but this is done by generating additional network/switch load. Obtaining flow statistics is a task that requires polling for information for every flow separately. The following ways for its improvement are proposed [1]:

- Aggregate flows: Generate only one query per set of flows that share the same parameters, for example the same source destination path instead of polling statistics for every flow separately.
- Data collection schemes: In case that there is no packet loss between the source-destination devices, poll different switches, thus reducing the overhead on a

single switch/link and spreading it evenly. Otherwise, stick to query the last switch only.

- Adaptive polling: Using a recurrent timer does not accommodate traffic changes and spikes. Furthermore, it may miss traffic changes resulting in inaccurate statistics. Hence, an adaptive algorithm that adjusts its query rate could enhance the accuracy and reduce the overhead.

According to the OF switch specifications [20], switches have to keep counters for port, flow table/entry, queue, group, group bucket, meter and meter band. Table II presents the Per Flow Entry counters used. Furthermore, in order to follow the statistics for more than one flow, there is an option to bundle multiple flows in a group and observe their aggregated statistics.

The monitoring concept [1] implements new packet loss and link utilization methods, and known delay measurements. The main processes are depicted in Fig. 5 The monitoring component released in POX registers every "PacketIn" event and creates a unique identification based on the flow information. Additionally, a separate ID is used to distinguish between the network paths. Every flow is assigned to a certain path, and the monitoring component keeps track of every flow that enters and the path it follows through the network. Furthermore, every Switch object also accounts the flows that pass through it. This information is later used to determine the link utilization.

In order to execute a piece of code in the future or assign a recurring event the monitoring component uses the Timer class incorporated in POX. In case, this is the first packet that uses this route, the monitoring component starts a polling timer for every second. Whenever the timer expires it fires an event. During this event a data collection algorithm is used (Round Robin or Last Switch). These two algorithms present a trade-of between accuracy and overhead. Afterwards, a message "StatusRequest" to the chosen switch is sent. This is the query requesting statistics for all the flows that follow the same path. Every path has a separate timer.

When a switch receives a "StatusRequest" message it generates a response. The "StatusReply" message contains the information obtained from the switch counters. On flow level it gives the duration of the flow (in nanoseconds), packet and byte count. Port statistics give more information about the state (both transmitted and received) such as number of dropped packets, bytes, errors and collisions. The controller obtains information for every flow that follows the same path. The polling timer is also adjusted. The controller tracks the time that passed since the last flow routed trough this path was registered, as this time increases, the polling timer also increases. In the implementation, the controller polls every second for the first five seconds, then every five seconds until the 15th second, moving to 15 seconds until the end of the first minute and polling once per minute when there has not been any flow activity for over a minute.

When the switch removes a flow entry from its table, because it was deleted or expired, it also raises a "FlowRemoved" event. Such event means that this flow is no longer active and the monitoring component does not need to account for it anymore. The controller receives a massage that indicates the whole duration of the flow together with the data statistics for this particular flow.

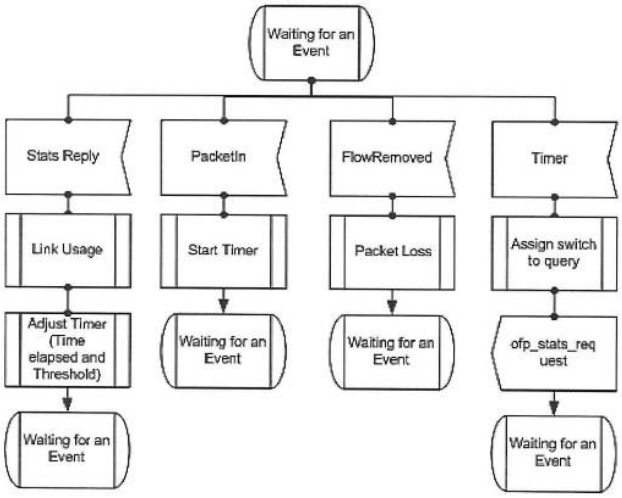

Figure 5. Monitoring algorithm [1]

TABLE II. COUNTERS [20]

Counter	Description
Received Packets	Counts the number of packets
Received Bytes	Counts the number of bytes
Duration (seconds)	Indicates the time the flow has been installed on the switch in seconds
Duration (nanoseconds)	Counts time the flow has been alive beyond the seconds in the above counter

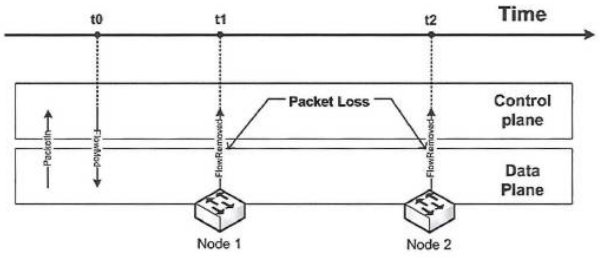

Figure 6. Calculating the packet loss percentage [1]

This is used to obtain packet loss information and undertake actions for the upcoming flows. If it is needed the controller may actively poll for packet loss statistics while the flow is still active. While this method generates additional overhead it is useful for cases when it is required to measure packet loss during the data transfer.

For measuring link packet loss a novel approach is proposed, capable to eliminate the overhead, and based on passive measurements. The main presumption is that packet loss metrics can be generalized on per class basis without loss of accuracy, and that measuring the packet loss for every single flow would not be viable. In order to estimate a stable and accurate link metric, that does not fluctuate too much, a set of measurements are required, more specifically - a metric that represents most of the packets, without accounting for the anomalous changes or the statistical outliers. As in an active network flows terminate every second, thus, the obtained measurements would still be real-time [1].

The whole process for measuring packet loss is depicted in Fig. 6. On a new flow arrival and when the switch does not have any rules installed, the first packet is

sent towards the controller. The controller is then responsible to decide what to do with the packet and eventually install table rules on each switch on the path of the flow. Once the flow has finished each switch indicates this with another message to the controller. The flow is installed at time t_0 with a "FlowMod" message sent from the controller towards every switch on the route of the flow. At time t_1, t_2, up to t_N (where N is the amount of switches), the controller receives "FlowRemoved" messages. Those messages indicate that the flow has expired and give some specific statistics for the flow, such as: the number of bytes, packets and the flow duration. Measuring the packet-loss relies on the fact that each switch sends this information based on its own counters.

Each switch has separate flow counters, but it counts different amount of bytes. This is due to link losses, congested links, etc. Receiving flow information from every switch allows comparing counter statistics and calculating the number of bytes that were lost. Whenever messages that the flow has expired from the same flow are received their recorded packet bytes are compared. This comparison allows determining the packet losses for the particular flow. The technique is sufficient to determine what the current link state for this traffic class is. In case there is a need for flow packet loss, the controller could poll for two or more node flow counters periodically [1].

IV. Evaluation results

The preliminary tests were done in two phases. First, using a virtual environment Mininet 2 (a container-based emulator able to create realistic virtual topology) [21] used hardware consists of Intel Core i5 nodes (the controller included) with four 2.53 GHz cores and 4 GB RAM. The containers mechanism uses groups of processes that run on the same kernel and yet use separate system resources, like network interfaces and IDs. Thus, every emulated switch or host creates its own process. Network links can be assigned specific link properties such as bandwidth and packet-loss. However, like most emulators, Mininet has also some drawbacks, e.g. processes do not run in parallel, instead they use time multiplexing, which may cause delayed packet transmission, not suitable for time accurate experiments.

As the results from the preliminary tests showed promising results, the same experiments were repeated in a real topology. The physical testbed was installed on servers that have Intel(R) Xeon(TM) processor with four 3.00 GHz cores. Every switch uses separate physical machine with Ubuntu 12.04.2 LTS operating system. The testbed uses Open vSwitch [44] as OF enabled switch. Traffic is generated by the Iperf application. This is a network testing tool capable to create TCP and UDP traffic between two hosts, where one is acting as client and the other as server. It measures the end-to-end (either uni- or bi-directional) throughput, packet loss and jitter. NetEm [25] is used, in order to emulate link packet losses and delay. It is an Linux kernel enhancement that uses the queue discipline integrated from version 2.6.8 (2.4.28) and later [1].

Different tests were carried out for measuring link utilization, comparing direct flow and aggregate flow statistics, adaptive and recurring polling, and for testing the proposed packet loss measurement method. The results for link utilization measurements show that [1]:

- using the aggregate flow query method decreases the overhead that is generated;
- the adaptive polling gives better results in terms of accuracy and overhead then recurrent polling.

The new method for measuring the packet loss was tested first in Mininet environment, and then repeated in the testbed. The results showed that the packet-loss varies from flow to flow. The packet-loss distribution results were promising, an average of 0.99% losses per flow and standard deviation of ±0.34 (due to the fact that NetEm uses normal distribution for packet loss emulation).

In order to determine exactly how accurate the method is 18 flows were recorded (Iperf Server report) and then compared with the measured packet loss. The first measurement consisted of sending flows worth of 64 Kbps for the duration of 195 seconds (average call duration). The results obtained matched perfectly with the Iperf Server report [1]. The second set of measurements emulated short term Video connection using MPEG 2 with data rate of 10 Mbps, whereas 10 flows set to continue each for 2 minutes were recorded. The results from both measurements prove that the proposed measurement method gives perfect accuracy.

The test results generally suggest that in order to reduce the network overhead, a technique that aggregates all flows that go through the same network route should be used. In addition, for eliminating the need of trade-off between overhead and accuracy, it is better to base the polling decisions not on the recurrent interval, but on the occurrence of certain event.

Finally, the new measurement method for packet losses has proven to be really accurate, and capable to determine the exact percentage for each link and also for any path. While it is a passive method, it does not impose additional network overhead, and it is not influenced by the network characteristics like the active probing methods that currently exist. The method is capable to provide statistics for every different type of service that passes trough the network.

Possible extensions to the measurements schemes suggested in this paper could be considered. The accuracy could be improved based on a combination of past statistics, link characteristics or weighted measurements results without imposing additional overhead. The adaptive timer requires more tuning, therefore, more research would be necessary on when more samples are needed and when less. More experiments in a real environment would help to fully proof the proposed measurement approaches. For the suggested packet loss method some questions need to be answered like: how much data are enough to take that the reported percentage of packet losses is not a random spike and how long before the data are too old to be considered valid [1].

V. Conclusions

This paper explores the concept of network monitoring implemented in SDN architectures. In terms of network monitoring, SDN allows to build a monitoring solution adjusted to the specific network needs. By using SDN the monitoring system is capable to obtain a complete view of the network that includes nodes, links and even ports. Furthermore, the solutions are capable to obtain fine grained and accurate statistics, for every flow that passes trough the network.

Once there are suitable monitoring systems capable to provide the necessary performance and usage statistics, the next phase is the network optimization phase. Major goal of TE is to enhance the performance of an operational network, at both traffic and resource level. Network monitoring takes an important part in TE by measuring the traffic performance parameters. Additionally, today TE in service provider's networks works on coarse scale of several hours. This gives enough time for offline TM estimation or it's deduction via regressed measurements. Unfortunately, this approach is not always viable, current IP traffic volume changes within seconds (or miliseconds), which could lead to congestion and packet losses in the most crucial moment.

Since SDN is a new architecture still gaining popularity, there are also some questions that need to be answered in terms of routing. Obtaining an accurate and real time view of the network could bring more benefits and open more options. Monitoring the network is the first step towards a SDN forwarding protocol capable to provide sufficient QoS for all types of applications and traffic.

Finally, it should be stressed that all present trends towards IoT, cloud computing, FoF, etc. highly depend on the availability of high-speed networks with certain QoS. While researchers are heavily working on the interoperability of applications and new Internet-based services, if the present problems on the transportation layer are not timely resolved, a real bottleneck for further developments could emerge.

ACKNOWLEDGMENT

The author gratefully acknowledges the MSc Thesis guidance provided by the Network Architectures and Services Group of Delft University of Technology.

REFERENCES

[1] V. Gourov, *Network Monitoring with Software Defined Networking*, Masters Thesis, TU Delft, Netherlands, August 2013.

[2] M. H. Behringer, "Classifying network complexity" in *ReArch '09*, New York, USA, 2009, pp. 13–18.

[3] Z Kerravala, *Configuration management delivers business resiliency*, Technical report, The Yankee Group, 2002.

[4] Q. Zhao, Z. Ge, J. Wang, J. Xu, "Robust traffic matrix estimation with imperfect information: making use of multiple data sources", *SIGMETRICS Perform. Eval. Rev.*, 34(1), 2006, pp. 133–144.

[5] sFlow. Traffic Monitoring using sFlow, URL: http://www.sflow.org/sFlowOverview.pdf. Online, July 2013.

[6] P. L. C. Filsfils, A. Maghbouleh, *Best Practices in Network Planning and Traffic Engineering*, Technical report, CISCO Systems, 2011.

[7] W. Stallings. "SNMP and SNMPv2: the infrastructure for network management", *Comm. Mag.*, 36(3), 1998, pp. 37–43.

[8] B. Huffaker, D. Plummer, D. Moore, K. Claffy, "Topology discovery by active probing", in *SAINT*, Nara, Japan, 2002, pp. 90–96.

[9] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, C. Diot, "Packet-Level Traffic Measurements from the Sprint IP Backbone", *IEEE Network*, 17, 2003, pp. 6–16.

[10] Silver Peak Systems, "How to Accurately Detect and Correct Packet Loss", URL: http://www.silver-peak.com/info-center/how-accurately-detect-and-correct-packet-loss. Online, July 2013.

[11] A. Tootoonchian, M. Ghobadi, Y. Ganjali, "OpenTM: traffic matrix estimator for OpenFlow networks", in *PAM'10*, Berlin, Heidelberg, Springer-Verlag, 2010, pp. 201–210.

[12] J. R. Ballard, I. Rae, A. Akella, "Extensible and scalable network monitoring using OpenSAFE", in *INM/WREN'10*, Berkeley, USA, 2010, pp. 8.

[13] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, H. V. Madhyastha, "FlowSense: monitoring network utilization with zero measurement cost", in *PAM'13*, Berlin, Heidelberg, Springer-Verlag, 2013, pp. 31–41=

[14] M. Yu, L. Jose, R. Miao, "Software defined traffic measurement with OpenSketch", in *NSDI'13*, Berkeley, USA, 2013, pp. 29–42.

[15] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rex-ford, A. Story, D. Walker, "Frenetic: a network programming language", *SIG-PLAN Not.*, 46(9), 2011, pp. 279–291.

[16] ONF, *Software-defined Networking: The New Norm for Networks*, White Paper, April 2012.

[17] A. Voellmy, H. Kim, N. Feamster, "Procera: a language for high-level reactive network control", in *HotSDN'12*, New York, USA, 2012, pp. 43–48.

[18] A. Tootoonchian, Y. Ganjali, "HyperFlow: a distributed control plane for OpenFlow", in *INM/WREN'10*, Berkeley, USA, 2010, pp. 3.

[19] Murphy McCauley, About POX, URL: http://www.noxrepo.org/pox/about-pox/. Online, 2013.

[20] The Open Networking Foundation, OpenFlow Switch Specification v1.3.1, URL: https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.1.pdf. Online, September 2012.

[21] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, N. McKe-own, "Reproducible network experiments using container-based emulation", in *CoNEXT '12*, New York, USA, 2012, pp. 253–264.