A3N: Agile Application-Awareness in Software-Defined Networks

He Cai[1], Jun Deng[1], Yuhua Zhang[1], Xiaofei Wang[1], Sangheon Pack[2]
[1]Tianjin Key Laboratory of Advanced Networking, Tianjin University, Tianjin, China.
[2]Mobile Networks and Communications Laboratory, School of Electrical Engineering, Korea University, Korea.

***Abstract*—With the rapid development of various real-time services, there is urgent need for application-aware capabilities in realtime network to meet the higher demand for network's quality of service (QoS), security policy and so on. Accurate and cost-effective collection of flow information is needed. However, the contradiction between the real-time, accuracy and performance cost in the passive traffic information collection mode of old board makes it difficult to build application-aware realtime network of high-accuracy. In this paper, we propose an agile application-awareness network (A3N) for software-defined networks (SDN). A3N implements a *flow-based self-adaptive sampling strategy* (FSS) for the incoming traffic and combines a parallel deep packet inspection (DPI) with high-performance to perceive network traffic changes. Experimental results demonstrate the proposed A3N can gain good performance with low costs, and also provide real-time application-aware services with deep operational visibility.**

I. Introduction

Network traffic application-awareness is a fundamental function that can be used to operate and manage network efficiently. Many network management systems (e.g. traffic engineering, QoS provisioning, etc.) require accurate network status information in real time [1]. Hence, the application-aware capability in real time is needed. However, real-time applications of perception cannot be achieved in the traditional network due to its static and closure properties.

Software-defined network (SDN) [2] is a new type of network architecture that separates the control plane from the data plane. The emergence of SDN can effectively break the limitations of traditional network, and provide the possibility for the construction of application-aware network.

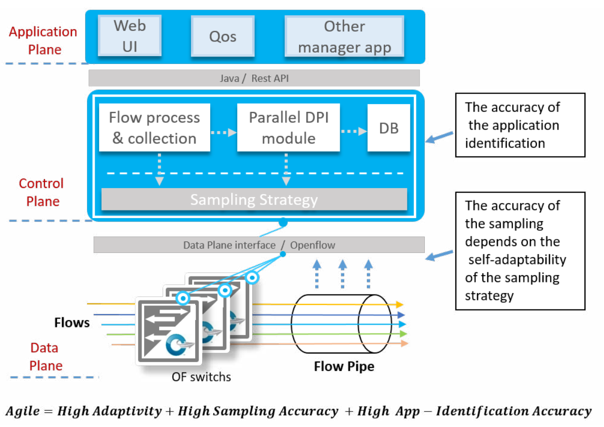

Application Plane
Web UI
Qos
Other manager app
Java / Rest API
Flow process & collection
Parallel DPI module
DB
Sampling Strategy
Control Plane
The accuracy of the application identification
The accuracy of the sampling depends on the self-adaptability of the sampling strategy
Data Plane interface / Openflow
Flows
Data Plane
OF switchs
Flow Pipe
Agile = High Adaptivity + High Sampling Accuracy + High App − Identification Accuracy

Fig. 1: The definition of agile application-aware network

The basis of application-awareness is the collection of flow information and application identification. Deep packet inspection (DPI) is an accurate and efficient method of application layer identification [3], which can analyze the contents of the packet to achieve identification. Most current solutions use the dedicated DPI equipments in the gateway to analyze traffic, but they cannot conduct a fine-grained analysis of the entire network. Also, the current collection mechanism is to collect all flows from all devices. In this way, the frequent collection of traffic will cause a sharp increase in computing and a decrease in performance. And large collection intervals will cause a lack of real time and accuracy. So the existing mechanism cannot perceive the change of network traffic accurately and capture flows effectively, which results in flows missing or meaningless performance consumption.

Aiming at solving the problem of collecting flows information when achieving global fine-grained awareness of the entire network traffic, we are motivated to: 1) realize a highly efficient and self-adaptive flow-based sampling algorithm with low costs; 2) combine the algorithm with high-accurate parallel DPI to identify applications which provide deep operation visibility of SDN based on real time.

As Fig.1 shows, we propose an agile application-awareness network (A3N) for SDNs. To achieve agility, A3N should provide: 1) high adaptivity on network changes and 2) high accuracy on flow sampling and 3) high application identification. To this end, A3N implements a *flow-based self-adaptive sampling algorithm* (FSS). It adopts a closed-loop control theory [4] and heuristic algorithms to collect flows. Our results demonstrate that the sampling algorithm can achieve high-accuracy with low costs. By combining DPI, A3N can provide real-time and app-aware services with deep operation visibility. Our contributions include:

- We implemented SDN-based sampling of flows and dynamic control methods of sampling.
- We implemented a self-adaptive sampling mechanism that controls the *interval (w)* and *sampling time (t)* to achieve lower performance consumption and higher accuracy.

The remainder of this paper is organized as following: related studies are introduced in Sec.II, and we describe system architecture and model verification in Sec.III. The FSS algorithm is introduced in Sec.IV. The experiments of our algorithm are introduced in Sec.V. Sec.VI is the conclusion.

II. Related Work

With the emergence of OpenFlow and SDN techniques, the researches of application-awareness networks have attracted more and more attention. Study in [5] is based on OpenFlow, and an application-aware network is constructed by combining the traffic classification DPI and machine learning in SDN control plane. In the article [1], they proposed and designed a controller that combines DPI to classify traffic at the control layer, which provides a feasible solution for the combination of DPI and SDN control plane to build SDN application-awareness network. However, they don't contain any traffic collection strategy. Research in [6] proposed different schemes for the combination of DPI and SDN, which provide a reference for constructing an application-aware network based on SDN and DPI. There also exist several studies for sampling strategy in traditional network and SDN. In the article [7] [8], the adaptive sampling methods are proposed in the traditional network. Some of the aforementioned existing studies have proposed the new network architectures which are applied to specific areas, while the rest mostly has focused on the sampling and inspection of traffic packets in either traditional or SDN-based network.

III. System Description

A. System Architecture

A3N includes several components: **Flow sampling strategy:** A low-cost, high-accuracy, real-time sampling algorithm is proposed, which will be detailed in Sec.IV. It bases on closed-loop control theory and heuristic algorithm. Through flow sampling, it can predict network flow situation to set up a reasonable flow collection strategy for each switch. **Flow processing & collection:** When packets are sampled to the controller, it is responsible for the packets sorting, classifying and quantifying into multi-dimensional flows, and then start sampling strategy as well as deliver them to the parallel DPI module. The flows are send parallel to the DPI for application identification. In A3N, we choose nDPI,an excellent open source DPI, which can identify up to 220 mainstream applications [3].

The controller collects flows, then makes multi-dimensional analysis and adjusts the reasonable sampling strategy for the current flow states of each node. Finally, A3N has well-defined Rest APIs and Java APIs that can provide real-time and application-aware services of deep operational visibility.

B. Analysis of Traffic Sampling Mechanism

Traffic sampling mechanism has a very important effect on the accuracy and performance of the sampling system. It is divided into two types: *packet-driven* and *time-driven* sampling mechanism. In *packet-driven* mode, if sampling rate is x, the switch node will sample a packet every $\frac{1}{x}$ packets. In *time-driven* mode, if the sampling interval is w and the sampling time is t, the switch node will continuously sample all the packets passing through the switch node after every w-t. For building the application-aware network, the *time-driven* approach is more advantageous because it can capture all packets for a period of time. It can greatly improve the accuracy of application identification by DPI if there is a flow of continuous packets. Thus, our algorithm uses time-driven sampling mechanism.

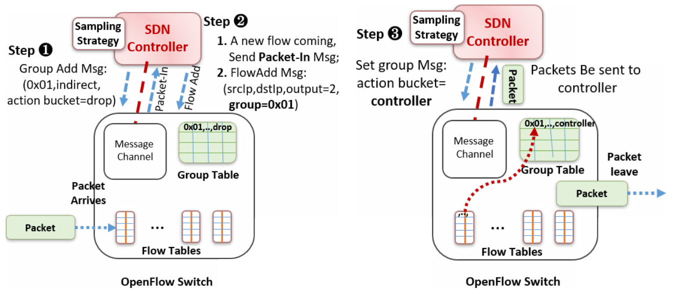
C. Using OpenFlow1.3 to Collect Traffic

The traffic collection method based on OpenFlow has been mentioned in many articles. In the article [1] [5], the most basic packet-in method is used to collect the traffic. Research in [6] is based on the multi-level flow tables, and adds extra flow entries for each flow entry.

We introduce a new traffic collection based on OpenFlow1.3. It is based on the *Group Table* and *Flow Table*, which can greatly reduce the difficulty and overhead of traffic collection. As Fig.2 shows, the workflow is as follows:

- **Step 1**: pre-insert a group entry for each switch in the SDN. *(groupId=0x01,type=indirect,action buckets=drop)*. The *Indirect* type is specifically designed to support the same forwarding action for multiple flow entries.
- **Step 2**: when the first packet of a flow arrives at switch, the controller sends a *FlowAdd* message to insert a forwarding action for the flow into the switch. At this point, we append *group:0x01* action to the FlowAdd message. This will cause all subsequent packets of the flow to perform the action defined by the *group:0x01*.
- **Step 3**: the controller implements sampling by a simple group table instruction: when controller sets the *group:0x01* action as *CONTROLLER*, all packets of each flow under the switch are sent to controller; It will stop sampling when the action is set as *DROP*.

There is no need to add any additional flow entries, each switch node requires only one group entry and an OpenFlow instruction, in which flows sampling can easily be achieved.



(a) Pre-insert *group entry: 0x01*, and associate the *flow entry* with the *group:0x01* when a new flow arrives

(b) By setting the action instructions of the *group: 0x01* to implement traffic sampling or stop sampling

Fig. 2: Sampling methodologies

D. Model Validation and Establishment

In the time-driven mechanism, to investigate the effects of *Interval (w)* and *Sampling Time (t)* on sampling results of flows, we conducted 132 flow sampling experiments with different combinations of w ($w \in \{300ms, 400ms, ..., 1400ms\}$) and t ($t \in \{100ms, 110ms, ..., 200ms\}$) on a single switch. The traffic data we used is collected by *the WIDE Project* [9] on the ISP traffic at 14:00-14:15 on 05/03/2017. After filtering and sorting, we reserved 128 flows , a total of 64900 packets and lasts for 79 seconds. A part of the experimental results and the ground truth show in Fig.3.

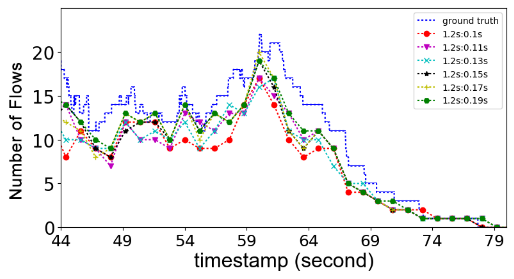


Fig. 3: A part of experiments results and the ground truth

By the exploration and validation of w and t, we can build related optimization models. And by adjusting w and t precisely, it will be possible to implement the self-adaptive sampling strategy with high accuracy, real time and low costs.

IV. Algorithm design and Description

Because of the dynamic changes of network traffic, the best solution is to find the most appropriate w and t for the current moment of network traffic situation. The value of w should reflect the shifting trends of numbers and type of flows in the current network traffic. When the numbers and types of flows change too large, the w should be smaller, which means that the sampling frequency should be faster, so as to effectively prevent the leakage of some short flows, and to improve accuracy. If the changes of numbers and types are very small, it means that the current network flow is stable, and the w can be larger, which can greatly reduce the sampling frequency to ensure the accuracy. Thus, we can select the reasonable w for the current switch according to the shifting trends in a short time. We use the shifting trends between S_i (denotes i_{th} sample) and S_{i-1} to predict the trend in a short period, and then determine w_{i+1}.

First, we define the $\widehat{FD}$. F_i denotes the set of flows obtained by S_i. $\widehat{FD_i}$ represents the flow gap in networks between S_{i-1} and S_i, which is determined by the numbers and types of flows. The formula is as follows:$\widehat{FD_i} = |\,(F_{i-1} \cup F_i) - (F_{i-1} \cap F_i)\,|$ The trend in a short time can be expressed as $k_i = \frac{\widehat{FD_i}}{w_i}$. k_i represents the current flow trend, so we can predict the w_{i+1}. Set the array $W = [1200, 1100, .., 200](ms)$,and let $w \in W$. We need to establish the mapping between k and W. We divide the first quadrant into *Length(W)* parts, and then the angle of each part (α) is $\alpha = \frac{\Pi}{2 \cdot Length(W)}$. Here we give the mapping function $f(k_i) = |\,\frac{(tan^{-1} k_i)}{\alpha}\,|$, which represents the array number of W.

Now the algorithm description about adjusting w is given: 1) at the end of S_i, first to get the set F_i; 2) then calculate the values of $\widehat{FD_i}$ and k_i; 3) so the $w_{i+1} = W[f(k_i)]$; 4) go to the next sample and then loop it.

Given the situation that there are two flows f_1, f_2 in the network, the frequency of their packets are $fq_1 = 110ms/packet, fq_2 = 120ms/packet$, if the current sampling time $t = 100ms$. Obviously f_1, f_2, with the probability of 9.1% and 16.7%, respectively, cannot be captured; If $t = 120ms$, f_1, f_2 can be almost 100% captured. The probability of capturing f_k by current sampling strategy P_k is:

$$P_k = \begin{cases} 1, fq_k \le t_i \\ 1 - \dfrac{fq_k - t_i}{fq_k}, fq_k > t_i \end{cases} \tag{1}$$

And hence we will consider three cases: ***Case 1:*** Assume that $f_{k(i-1)}$ is a flow captured by S_{i-1}, in the case of $P_k < 1$, if packet of f_k is not captured in the S_i, the sampling strategy should reasonably determine the existence of f_k and simultaneously increase t_{i+1}, that is, make $t_{i+1} > t_i$, to prevent missing f_k again. The most reasonable value should be $t_{i+1} = fq_k$. If f_k is still alive, then it can be captured by S_{i+1}. In practice, there may be multiple f_k that are not captured for each sample, and the t_i should be compromised. To let t_{i+1} equal to the average of all fq of flows that are not captured by S_i is a simple and effective method. Thus, we define U as the set of flows that $fq \ge t_i$ and are captured in S_{i-1} but not in S_i.

$$U = \{f_k \mid f_k \in F_{i-1} \wedge f_k \notin F_i \wedge P_k < 1\} \tag{2}$$

$$U \neq \emptyset \Rightarrow t_{i+1} = \frac{1}{|\,U\,|} \cdot \sum_{f_k \in U} fq_k \tag{3}$$

Case 2: When $U = \emptyset$, if there is a set $D = \{f_k \mid f_k \in F_{i-1} \wedge f_k \notin F_i \wedge P_k = 1\} \neq \emptyset$ in S_i, in this case, it is highly likely that the flows have died, but it is also possible that the flows are not captured caused by the instability of the network traffic. So the value of t should be kept constant as much as possible, that is $t_{i+1} = t_i$. ***Case 3:*** When $F_{i-1} \subseteq F_i$, t_{i+1} should be equal to $maxFq = max\{fq_k \mid f_k \in S\_F_i\}$. This makes t_{i+1} decrease or remain constant. Thus, the value of t can close to the maximum value of the flows frequency in network to ensure that all flows are captured as much as possible, while at the same time making t to be as small as possible.

This algorithm, which can adjust the most reasonable t value according to the frequency of the network flows, is called ***Viscosity Decreases, Mean Increases(VDMI)***. It is a closed-loop control theory, which is also very similar to the congestion control strategy of TCP.

Algorithm 1 VDMI algorithm for t

```
1. start the $i_{th}$ sampling;
2. sampling ends; get $U$ and $D$, calculate frequency for flows;
3. if $U \neq \emptyset$ then
4.     $t_{i+1} = \frac{1}{|U|} \cdot \sum_{f_k \in U} fq_k$; // $t_{i+1}$ should be $t_{i+1} \le w_{i+1}$; if a flow appears twice in U, the flow is considered dead, should not participate in the calculation of $t_{i+1}$, and remove from $U$, otherwise considered still alive.
5. else if $D \neq \emptyset$ then    $t_{i+1} = t_i$;
6. else calculate the $maxFq$ , $t_{i+1} = maxFq$;
7. $i = i + 1$; return step 1;
```

The combination of adaptive algorithm of w and Algorithm 1 is the *Flow-based self-adaptive sampling algorithm*. It can reasonably calculate w and t for each switch according to the trend of the network flows and the frequency of each flow.

V. Experiments and Results

We build a testbed based on the Floodlight controller and Mininet, then we implemented our algorithm in the controller. The testbed contains two Dell XPS 8920 hosts, with Intel Core i7-7700 3.6GHz 8 core CPU, 16G RAM. One of them uses the Ubuntu 16.04.2 LTS to run the topology and the other runs the Ubuntu 14.04 LTS as the SDN controller. In order to compare with the previous 132 fixed w and t sampling experiments, the traffic source will be the same as them. In the experiments, we set $t_{max} \leq 200ms$, $t_{min} \geq 10ms$ .The topology is shown in Fig.4, we let VM1 send traffic to VM2.

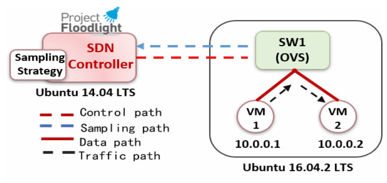


Fig. 4: Topology

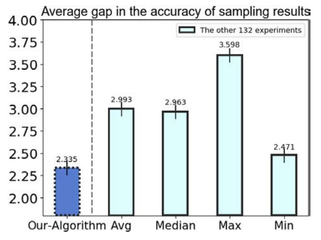
The experiment is mainly to verify the effectiveness and performance of the *Flow-based self-adaptive sampling algorithm*. Fig.6(a) shows how the values of w and t are adaptively changed as time and network traffic change during the experiment. We compare the results of this experiment with the results of the 132 experiments mentioned above, then we conducted four aspects of comparison, including: the average gap in the accuracy of the sampling results, the average gap in the degree of fitting of the trend, the average gap of flows and the total sampling time. The total sampling time can be used to measure the cost simply.

As shown in Fig.5, compared to the other 132 groups sampling results, the sampling results of our algorithms have the lowest average gap in the accuracy, 21.98% lower than their average, which means that our algorithm has the best sampling accuracy. Its total sampling time is 7.02% lower than the average of the contrast experiments. The other two aspects of contrast also have an obvious advantage. The above results show that FSS algorithm has high sampling accuracy while ensuring lower costs.

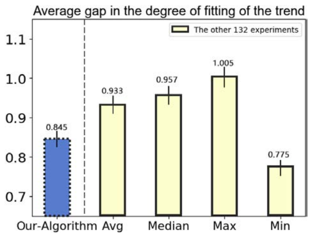
As described in Sec.III-A, we modify the nDPI source code so that it can be parallel, integrated and finally built the A3N successfully. A3N can provide real-time application-aware services of deep operational visibility, so we implemented web GUI demo. As Fig.6(b) shows, the number of flows, the type of application for each flow, the path each flow passes, the start time of the flow and so on in current network can be provided in real time directly.

VI. Conclusion

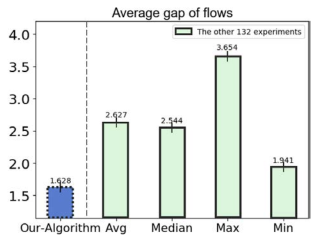
In this paper, aiming at the difficulties and requirements of building application-aware networks, we present the adaptive sampling technique and system, A3N, supported by SDN and DPI techniques. Our proof-of-concept tests show that the algorithm can achieve good performance for providing real-time agile application-awareness successfully.



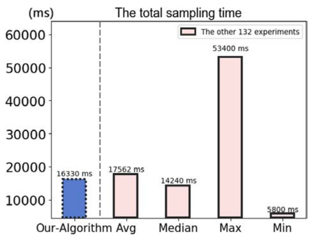
(a) The average gap in the accuracy of the sampling results



(b) The average gap in the degree of fitting of the trend

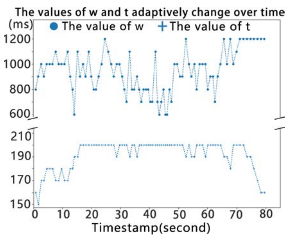


(c) The average gap of flows

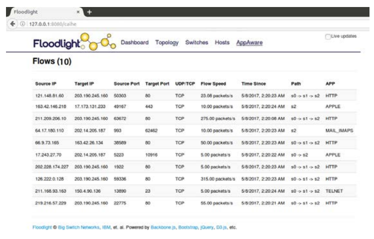


(d) The total sampling time

Fig. 5: Experimental comparison results



(a) w and t adaptively change



Flows (10)

Source IP	Target IP	Source Port	Target Port	UDP/TCP	Flow Speed	Time Since	Path	APP
121.148.81.60	203.190.245.160	50303	80	TCP	23.08 packets/s	5/8/2017, 2:20:23 AM	s0 -> s1 -> s2	HTTP
163.42.146.218	17.173.131.233	49167	443	TCP	10.00 packets/s	5/8/2017, 2:20:24 AM	s2	APPLE
211.209.206.10	203.190.245.160	63672	80	TCP	275.00 packets/s	5/8/2017, 2:20:08 AM	s0 -> s1 -> s2	HTTP
64.17.180.110	202.14.205.187	993	62462	TCP	10.00 packets/s	5/8/2017, 2:20:23 AM	s2	MAIL_IMAPS
66.9.73.165	163.42.26.134	38589	80	TCP	50.00 packets/s	5/8/2017, 2:20:23 AM	s0 -> s1 -> s2	HTTP
17.243.27.70	202.14.205.187	5223	10916	TCP	5.00 packets/s	5/8/2017, 2:20:22 AM	s0 -> s2	APPLE
202.228.174.227	203.190.245.160	1922	80	TCP	5.00 packets/s	5/8/2017, 2:20:23 AM	s0 -> s1 -> s2	HTTP
126.222.0.128	203.190.245.160	59336	80	TCP	315.00 packets/s	5/8/2017, 2:20:23 AM	s0 -> s1 -> s2	HTTP
211.168.93.163	150.4.90.136	13890	23	TCP	5.00 packets/s	5/8/2017, 2:20:24 AM	s0 -> s1 -> s2	TELNET
219.216.57.229	203.190.245.160	22775	80	TCP	55.00 packets/s	5/8/2017, 2:20:21 AM	s0 -> s1 -> s2	HTTP

Floodlight © Big Switch Networks, IBM, et. al. Powered by Backbone.js, Bootstrap, jQuery, D3.js, etc.

(b) A3N web GUI Demo

Fig. 6: Experimental results of A3N web GUI

References

[1] S. Jeong, D. Lee, J. Choi, J. Li and J. Hong, "Application-aware Traffic Management for OpenFlow Networks", in *APNOMS*, pp.1-5, Oct 2016.

[2] N. Mckeown, T. Anderson, H. Balakrishnan, G. Parulker, L. Peterson, J. Reaford, S. Shenker and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks", in *ACM SIGCOMM*, Apr 2008.

[3] T. Bujlow, V. Carela-Espanol and P. Barletros, "Independent Comparison of Popular DPI Tools for Traffic Classification", in *Computer Networks*, pp.75-89, Jan 2015.

[4] J.J. DiSteffano, A.R. Stubberud and I.J. Williams, "Feedback and Control Systems", *New York, McGraw-Hill*, 1990.

[5] Y. Li and J. Li, "MultiClassifier: A Combination of DPI and ML for Application-layer Classification in SDN", in *Systems and Informatics (ICSAI), 2014 2nd International Conference on*, pp.682-686, Nov 2014.

[6] Y. Zhao, P. Zhang and Y. Jin, "Netography: Troubleshoot Your Network with Packet Behavior in SDN", in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pp.878-882, Apr 2016.

[7] Y. Gan, Y. Zhang and D. Qian, "Adaptive Sampling Measurement for High Speed Network Traffic Flow", in *Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on International Conference on*, pp.1-4, Sep 2009.

[8] B.Y. Choi, J. Park and Z.L. Zhang, "Adaptive Packet Sampling for Flow Volume Measurement", in *ACM SIGCOMM Computer Communication Review*, 9-9, Jul 2002.

[9] "the WIDE Project", [online] Available: http://mawi.wide.ad.jp/mawi/.