SOCIETY FOR SCIENCE AND EDUCATION
UNITED KINGDOM

TNC TRANSACTIONS ON NETWORKS AND COMMUNICATIONS

VOLUME 6, NO. 5
ISSN: 2054 -7420

Efficient On-Line Traffic Policing in Software Defined Network

Lie Qian

Dept. of Chemistry, Computer & Physical Science, Southeastern Oklahoma State University, Durant, OK, USA
lqian@se.edu

ABSTRACT

On-line traffic such as conversational call and live video on the Internet are not pre-recorded and has no exact information about each session before the traffic happens. S-BIND (Confidence-level-based Statistical Bounding Interval-length Dependent) traffic model characterizes such traffic for QoS admission (GammaH-BIND) and policing purpose. A state-dependent token bucket based statistical regulator was proposed to police the traffic using S-BIND parameters. Recently, Software Defined Network (SDN) is proposed to decouple the control plane from the data plane, which enables low cost commodity design in switches and flexible network feature deployments through software implementation in centralized controllers. To deploy the state-dependent token bucket statistical regulator in SDN, extensions to the current SDN are needed. In this paper, design options for S-BIND traffic regulator in SDN switches are presented and analyzed. Among these options, a single meter design is chosen based on the cost and efficiency comparison between them. The needed switch implementation and OpenFlow protocol's extensions to realize the regulator in SDN is given at the end of the paper.

Keywords: Software Defined Network, OpenFlow, QoS, Policing

1 Introduction

After 30 years' fast growth, Internet is still growing in number of users, amount of data transmitted and data transmission speed. Internet is playing a more important role in economy, politics, education, entertainment and healthcare every day. Users are expecting more than simply retrieving desired information from the internet now days. Quality of Service (QoS) refers to the capability of a network to provide more than best effort service to chosen network traffic [1]. In best effort service, the network doesn't distinguish between packets from different users, or different applications. All packets are treated in the same way while the network devices try to forward every packet as fast as possible. There is no guarantee to the service received by any packets in the term of delay, packet loss rate, jitter and etc. A packet delivering Facebook page update is treated in the same way as a packet delivering video streaming on Netflix.

To fulfill QoS requirements, traffic in the network need to be controlled. A new traffic flow needs to be approved by a process called Admission Control [1, 13-20]. Algorithms are used in Admission Control to decide if the existence of the new traffic flow will affect all traffic's QoS. Traffic descriptor-based admission control [17-20] uses existing and new traffic's characterizations (parameters) to calculate the QoS that

DOI: 10.14738/tnc.65.4975
Publication Date: 14th September 2018
URL: http://dx.doi.org/10.14738/tnc.65.4975

could be received. To effectively characterize the network traffic with a set of parameters, especially for online traffic, which is not pre-recorded such as conversation call, live video, Confidence-level-based Statistical Bounding Interval-length Dependent (S-BIND) [1] model was proposed along with GammaH-BIND admission control algorithm. After being admitted, the traffic also need to be constantly monitored and regulated to ensure the real traffic going through the network be bounded by the declared S-BIND parameters. The regulator proposed in [2] is composed of a series of state-dependent token buckets. In [3], these token buckets are optimized to regulate (drop the excessive packets) the incoming traffic in a more efficient and accurate way. However, to deploy S-BIND together with the GammaH-BIND admission control algorithm and the state-dependent token buckets based regulator, there is a very common challenge in now days' network device industry: vendors of the network devices are reluctant to implement new services into their products before the services become widely accepted globally. On existing devices, making changes to deploy new services is not easy while traditionally forward functions (data plane) and control functions (control plane) are coupled together in network devices. [4-6]

Network functions are categorized into data plane and control plane. Data plane includes functions such as packet switching, dropping and modification. Control plane is responsible for routing, admission control, network monitoring, per-flow control, topology discovery, QoS support, etc. In traditional network devices, data plane and control plane are bounded together. Functions from both planes are implemented in the device by its vendor, which makes the control plane very difficult to change when new network service is needed on existing network devices. To deploy a new internet architecture, a new protocol or a new network policy, the control plane in every device needs to be reconfigured, tested, debugged and even rebuilt by its vendor.

Software Defined Network (SDN) is proposed to decouple the control plane from the data plane [7]. In SDN, data plane is still implemented by the device vendor in hardware while the control plane is realized by software in one or multiple centralized controllers. Software such as network management applications, network operating systems and OpenFlow protocols work together to control how packets are handled in each devices' data plane. Deploying new internet architecture, network management, services or protocols is simplified to software update in the controllers and no change is needed in the large number of traffic forwarding devices. Network services such as QoS, virtualization, security, traffic engineering, Information centered networking, forensic analysis, transferrable network applications, deep packet inspection, cloud data center, dynamic middle box deployment, virtual collaborative working environment, VLAN, etc. are becoming more realistic in SDN [8-9]. While many complicated control functions are moved to controllers, the majority network devices only need to deal with data plane which is relatively simple and standardized. This encourages low-cost commodity design in the data plane network devices, which lowers the cost, improves performance and enables cross vendor compatibility.

With SDN in the picture, deploying the on-line traffic S-BIND admission control [1] and traffic regulator solution [3] becomes more practical than before. The admission control could be implemented in the centralized controllers of SDN as software to make admission decisions. If admitted, new table entries for new flows are installed in the switches on the flow's route using OpenFlow protocol [10]. Controllers also need to install regulators in the ingress switches to regulate the flows. Current OpenFlow's meter design is not capable of providing needed token bucket's behavior as described in [3]. In this paper a solution to enable S-BIND characterized traffic's policing in Software Defined Network is proposed. First, couple meter design options are presented. The comparison and analysis of these design options lead to the choice of the single meter design option. To realize the chosen single meter option, needed OpenFlow protocol extensions and switches' meter implementation are presented the next.

The remainder of this paper is organized as follows. Section 2 presents the background review including S-BIND model together with its regulator and software defined network. In section 3 and 4, couple SDN switch token bucket design options are presented and compared. The new regulator design, implementation and protocol extensions are discussed in Section 5. Section 6 gives the future work and conclusion.

2 Related Works

In this section, a brief review of the S-BIND traffic model, its regulator, software defined network and OpenFlow is given. In this paper's network model, a centralized controller is used in a network domain. This controller serves as QoS controller and SDN controller. It is responsible for admission control decision and policies installation in the forward elements (switches) in the domain.

2.1 S-BIND Traffic Model

Binding traffic models use parameters to describe and bound the traffic volume. Traffic constraint function, denoted as b(t), is the essential part for binding traffic models. A network traffic flow is said to be bounded by b(t) if during any interval of length of u, the amount of traffic transmitted by this flow is less or equal to b(u),

In [1], authors developed a Confidence-level-based Statistical BIND (S-BIND) traffic model. The S-BIND model defines *p* time interval rate pairs as:

$$\{(R_k, I_k) \mid k = 1, \ldots, p\} \tag{1}$$

where, $I_1 < I_2 < \ldots < I_{k-1} < I_k$. I_k denotes the time interval length, and R_k is the rate, at which the flow is allowed to send in any period of I_k statistically. In S-BIND, random variable S_k is defined as:

$$S_k(a) = prob\left(\frac{A_j[t, t+I_k]}{I_k} \le a\right), \forall t \ge 0 \tag{2}$$

Here, $A_j[t_1,t_2]$ denotes the amount of arrivals in traffic flow within time interval $[t_1,t_2]$. S_k reflects the distribution of flow's transmission rate over time interval I_k and has density function $s_k(a)$. For each time interval I_k, R_k is defined in S-BIND by using random variable S_k's density function $s_k(a)$ as following:

$$\int_0^{R_k} s_k(t)dt = \varepsilon \tag{3}$$

ε is set between 0 and 1. The smaller ε is set, the smaller R_k will be, and the higher network utilization can be expected in admission control because more traffic will be admitted. For different kinds of on-line traffic such as conversation, videoconference, high motion video, low motion video, game data and etc., S-BIND parameters can be pre-defined with different confidence levels through experiments or statistical data analysis.

Along with GammaH-BIND admission algorithm developed in [1], S-BIND can achieve maximum valid network utilization for both low bursty and high bursty traffic.

2.2 Token Bucket Based Statistical Regulator

Admitted traffic flows need to be monitored and regulated during their lifetime so that they don't violate their declared S-BIND parameters. A token bucket based statistical regulator was originally proposed in [2] and later improved in [3]. In the solution, multiple state-dependent token buckets with dynamic token generation rates are cascaded together to regulate S-BIND traffic. One bucket is deployed for each linear segment in the constraint function b(t) which is derived from the S-BIND parameters. A bucket for a linear segment between (b_s,t_s) and (b_e,t_e) has the behavior given in Figure 1, where $\Delta b=b_e-b_s$, $\Delta t=t_e-t_s$, $r=\Delta b/\Delta t$, and b_{jump} is defined as:

$$b_{jump} = \begin{cases} b_s - \Delta b & if\ \Delta t < t_s \\ b_s - t_s r & if\ \Delta t \geq t_s \end{cases} \qquad (4)$$

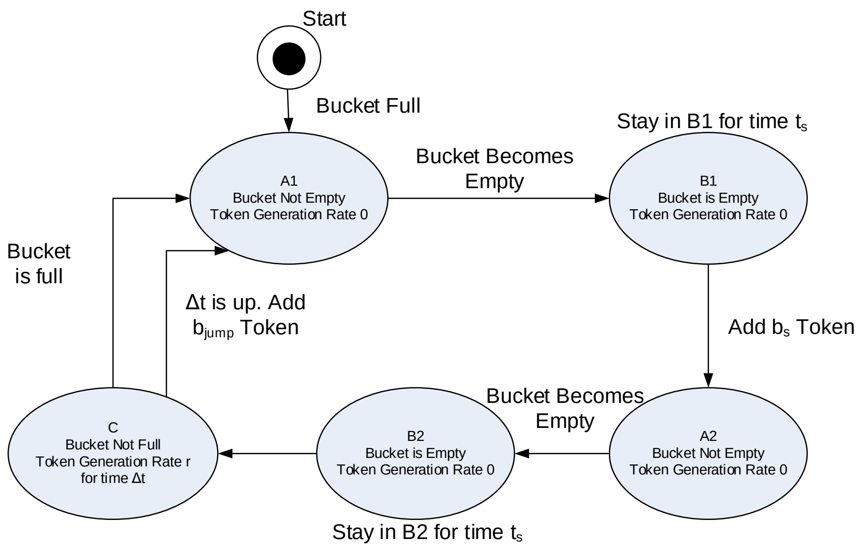

Figure 1: Token Bucket's Behavior

In the regulator designed in [3], it is possible that a bucket, say *B*, has its constraint function below another bucket *C*'s constraint function at all time interval t. In another word, bucket *B*'s output is always less than the amount of traffic allowed by bucket *C* at any time interval. In this case, there is no need for bucket *C* and an algorithm is developed in [3] to reduce the traffic policing burden in the regulator by removing *C*'s bucket.

Different from traditional deterministic token buckets which drop incoming packets when there is no enough token left in the bucket, the regulator proposed in [2] and [3] is allowed to forward packets even when the token is not enough statistically based on the confidence level ε specified in S-BIND traffic model. When a packet *p* with size *s* needs to be transmitted while the token left in bucket is $t<s$, the transmission probability of the packet is calculated as (5).

$$P_{transmit} = \begin{cases} \dfrac{(1-\varepsilon) - \dfrac{t_{neg}}{t_{neg}+t_{pos}}}{(1-\varepsilon)} & if \dfrac{t_{neg}}{t_{neg}+t_{pos}} < (1-\varepsilon) \\ 0 & \text{otherwise} \end{cases} \qquad (5)$$

In (5), the transmitting probability P mainly depends on the variables t_{neg} and t_{pos}, while ε is a confidence level parameter from S-BIND traffic model, which should be a constant for the life time of a bucket. t_{neg} is counted as the time when the token bucket has negative amount of token and t_{pos} is calculated as the total time minus t_{neg}.

2.3 Software Defined Network

Traditional network devices like switches bound data plane (packet forwarding, dropping and modification) and control plane (QoS, routing, network monitoring, per-flow control) in the same equipment. When any new network service, protocol or architecture need to be deployed, the control plane usually requires significant modifications. Even when the change could be accommodated by reconfiguration, still all devices need to be reconfigured by the network administrator. More often, the change needed is too significant and go beyond what is allowed in reconfiguration. In such cases new devices that support the new control plane need to be purchased and deployed if you are lucky to find them on the market while most vendors are reluctant to implement new service into their products before the service become widely accepted on the market, which is not always the case for newly proposed services, protocols or architectures.

Software Defined Network (SDN) was proposed to decouple the control plane and data plane [7]. The rationality is based on the fact that the data plane is relatively simple, stable and high performance demanding while the control plane is complicated, changes along with new services and requires flexibility. In SDN, the data plane remains in network devices like switches and the control plane is moved to one or multiple centralized controllers. The controllers install policies in the network devices to control how network traffic are processed in each switch. Network device vendors can focus on improving the data plane performance and lowering the cost of the devices and don't need worry about the constantly changing control plane [11]. Whenever the control plane needs to be changed, controllers could be reprogrammed (install a new software) to be capable of installing different policies in switches to realize the new services or protocols. With SDN, the network service that can be provided in a network is not limited by what the vendor implemented inside the devices anymore, which encourages and enables new network technology, services, protocols and architecture's design; as well as testing and deployment, services such as QoS, virtualization, security, traffic engineering, Information centered networking, forensic analysis, transferrable network applications, deep packet inspection, cloud data center, dynamic middle box deployment, virtual collaborative working environment, VLAN, etc. [12]

In a SDN, one or multiple controllers are deployed to make network management decisions. Network management applications, network operating systems and protocol interfacing the controller and switches are installed in the controllers. Network management applications is responsible for making network control decision. Network operating systems provide API to allow quick and easy network management applications programming and shield the network management application from the heterogeneous network technology. Openflow [10] protocol is installed in both switches and controllers. Openflow defines how the rules from the controller could be installed, updated, and removed in the switches and how these rules can be used in the switches for packet processing.

When the first packet of a new flow comes to a switch in the network domain. The packet could match no rule in the switch and be encapsulated and forwarded to a controller. The network operating system in the controller runs the network management application to process the packet and decides how this flow should be processed in the future. The decision is translated to a series of rules to be installed in on route switches in the network. Using OpenFlow protocol, these rules will be sent to and installed in the involved switches.

2.4 OpenFlow Switches

In an OpenFlow switch [10], there are one or multiple flow tables and one group table. Using OpenFlow protocol, the controller can add, modify and remove entries in the tables inside each switch. Each entry in any table consists of match fields, counters, and a set of instructions to apply to matching packets. Packets may need to be processed by multiple flow tables one by one when they come to a switch. When a packet is processed in a table, if a match is found, the associated instruction of that matched entry will be executed. If no match is found, a special miss entry in that table will determine the fate of the packet. The packet could be forwarded to the controller using Packet-in message, dropped or forwarded to the next flow table in the same switch depending on the policy installed in the miss entry. Controller can use Modify-State message to add flow entries in any table.

For a matched packet, instructions associated with the matched flow entry are executed. Some of the instructions edit the action set associated with the packet. All actions in the action set will be executed after the packet finishes its processing in the last flow table in the same switch. In addition to the actions in the action set, “Apply-Action” instruction could execute action of choice during the table’s processing. Instruction “Goto” specify which table is the next to process the packet. Instructions also can send metadata to the next table to assist the packet processing there. Actions are defined in OpenFlow to describe the forwarding, modification of the packet, applying meters to the packet, sending the packet to specific queue for QoS purpose, etc.

Meters are used in the switch to compare the incoming packets rate from one flow or a group of flows against certain predefined rate to decide if a packet should be forwarded or not. Action “meter” in the action set or action list (used in “Apply-Action” instruction) triggers a specific meter to be applied to a packet. One meter could be used against multiple flows. Also one packet from a flow could have multiple meters applied to it and the packet need pass all meter’s check to be forwarded. Inside a meter, there could be one or more meter bands. Each meter band has its own rate setting. When one packet needs to be processed by a meter, at most one meter band will be applied. The meter bands with the rates lower than current measured rate will be considered and among which the highest one will be chosen (if the measured rate is lower than all band rates, no meter band will be chosen). Meters together with associated meter bands are installed by SDN controllers through OpenFlow protocol messages.

3 Token Bucket Design in SDN

Enabling token bucket’s behavior as described in [2][3] is critical in developing online traffic regulator in a Software Defined Network. In this section first the SDN network management model is presented, which describes the overall architecture in SDN for new traffic admission control and traffic policing. Then three token bucket design options are discussed.

3.1 SDN Network Management Model

When the first packet of a new flow comes to the network, the ingress switch cannot find a flow entry (except the default no-match entry) matching this packet in its table and will forward this packet to the controller, which is defined in OpenFlow. Network management application in the controller extracts the

S-BIND traffic parameters from the packet and performs admission control based on admission algorithms such as [1]. If the new flow is denied, a packet with cancelation flag will be sent toward the source and no new policy needs to be installed in any switch. If admitted, a packet with acceptance flag will be sent to the source of the flow. The controller also calculates how many token buckets are needed and the parameters of each bucket. Based on the calculation, table entries and meters will be installed in the ingress switch to regulate the new flow's traffic.

3.2 Regulator Design Options

As described in [2], regulators for S-BIND traffic are composed of multiple token buckets, which has state-dependent dynamic token generation rate and statistical packet forwarding. Each bucket is deployed for one linear segment on S-BIND's constraint function. Optimization algorithm proposed in [3] could reduce the number of buckets needed through removing redundant token buckets. Still multiple buckets are usually needed for each flow. Fortunately, meters are defined in OpenFlow.

Meters are used in OpenFlow to regulate traffic in SDN. Switch table entries could use instructions to apply meters to incoming packets. Multiple meters could be applied to the same packet. Vice versa, the same meter could be used to regulate traffic from a group of flows. S-BIND and its regulator are per-flow control. Therefore, one meter only applied to one flow for S-BIND traffic. Multiple meters could be needed for each flow to realize multiple token bucket's behavior.

The meter and meter bands defined in OpenFlow [10] at this moment can only control traffic using 2 simple parameters: rate and burst size. When the incoming traffic burst exceed the burst size, the rate is applied to decide if forward or flag the packets. The measuring window length (1ms, 100ms, 1s or others) are determined by the manufacturer and not modifiable in the controller on per flow base. Such simplicity in the meter band design prohibits it from realizing needed policing behavior in S-BIND regulator.

Each token bucket's behavior is depicted in Figure 1. There are 5 states in the bucket's behavior. Among the 5 states, states A1 and A2 have similar behavior, in which there is no token generated and bucket leaves these 2 states only when the remained token value hit zero. States B1 and B2 have similar behavior, in which there is no token generated and the bucket leaves these 2 states after t_s timeout elapsed. State C is different from all 4 states mentioned above. There is a token generation rate r in state C and bucket could leave state C in 2 scenarios; either timeout Δt runs out or the bucket become full of token (b_s amount of token in the bucket). Due to the different behaviors in these 5 states, there are 3 options to implement one bucket's behavior: in one, two or three OpenFlow meters.

3.2.1 Three-Meter Design Option

Based on the discussion above, all 5 states' behavior could be categorized into 3 groups as shown in Table 1. In the first candidate design option, 3 meters are used to implement one token bucket. One meter is deployed for states in each row in Table 1. One meter is used for state A1 and A2 (say meter A). One meter is used for state B1 and B2 (say meter B). One meter is used for state C (say meter C).

Table 1: Bucket's Behavior in 5 States

State	Token Generation Rate	Exit Condition
A1, A2	0	Token =0
B1, B2	0	Reach Timeout
C	r	Token=b_s OR Reach Timeout

Three meters, A, B and C, are setup in the ingress switch to realize the behavior of this bucket. Flow entries are added into four tables (say table ID 0-3) to apply meters for one bucket. The first table (table 0) doesn't apply any meter. It decides where the packet will go to (table 1-3). Table 0 is deemed as a master table for this bucket. Table 1 has the flow entry running instruction to apply action “meter A”. Table 2 does that for meter B and table 3 for meter C. When a packet is matched in table 0, it will be forwarded to one of the 3 tables (ID 1-3) to be policed by one meter. OpenFlow protocol needs to be extended to include one new state field of type unsigned integer in table's flow entry to indicate where the incoming packet should be forwarded to (only table 0 will use it). One new instruction should be added into the protocol for table 0 to forward packet to the table with ID indicated by the new state field. The new instruction could be a variation of “GOTO” instruction currently defined in OpenFlow used to forward a packet to another table. The existing “GOTO“ instruction requires an argument to specify the table ID [10]. The new version “GOTO” instruction could be executed without argument and use the new state field's value by default as the ID of the destination table.

The new state field should be initialized to 1 (if table 1 is used to apply meter A) by the controller during flow entry installation. One important modification to the switch is to allow the flow entry's new state field be accessed by meters. One link or address to the new state field needs to be added into OpenFlow's meter design. If the meter is used to police S-BIND traffic, the link will point to the new state field in the policed flow's master table's entry. All A, B, C meter's behavior should include the action of updating the new state field when the bucket leaves its current state, so that the next packet will be forwarded to a different table to be policed by another meter.

3.2.2 Two-Meter Design Option

In the second design option, the fact that state C is the only state with non-zero token generation rate is considered. Two meters are used to implement one token bucket. One meter is used for state A1, A2, B1 and B2 (say meter AB), where token generation is zero. One meter is used for state C, called meter C. Three tables (table ID 0-2) are used to realize one bucket's behavior. Table 0 still serves as the master table, which is responsible for forwarding the packet to either table 1 or table 2. In order to do that, the new state field is still needed in table 0 and is accessible to both AB and C meters. Table 0 can use the new argument-less “GOTO” instruction to forward packets to table 1 or 2 based on the new state field's value. Table 1 applies meter AB and table 2 applies meter C.

3.2.3 Single-Meter Design Option

The last design option in consideration is to use one meter (say meter ABC) for one token bucket. All 5 states' behavior are implemented in one meter. One table (any table is fine, not necessarily table 0) is used to apply the ABC meter. In this design, packets of the same flow will always be policed by the same ABC meter applied in the same table, therefore the new state field mentioned in previous 3.2.1 and 3.2.2's design options is not needed in the table entry. Also the associated extra access to the new state field from any meter is not needed anymore.

4 Design Option Analysis

The three design options (1-3 meter designs) described in section 3 require some extensions in current OpenFlow protocol and SDN switches. Extensions could be needed in table's entry formats and/or meter's implementation. In this section, the three design options are compared and analyzed in the terms of implementation complexity and cost. The results show that single meter implementation is the best choice.

4.1 Single Bucket Implementation Cost Analysis

In the single meter design option (as described in 3.2.3), there is no extra cost added in the switch table. The ABC meter keeps track of variables such as tokens left, timeout left, positive token time, negative token time, and state (token generation rate is a constant set by the controller, no need to be maintained). Positive token time and negative token time are used to track time duration in which the token amount is above or below zero. These two variables are used for statistical packet forwarding when there is no enough token to forward a packet [1]. Variable “state” is used to trace the state of the meter. The ABC meter could be in 5 different states and the variable uses values 0-4 to indicate the current state so that the meter could behave accordingly.

When there are 2 meters used for each bucket (Two-Meter design option in 3.2.2), 3 table entries need to be installed in 3 different tables for one bucket. One of them serves as the master table, and it needs a new field state in the entry to indicate where the packet should be forwarded to. The second and third table are meter tables, which apply meter AB or meter C respectively. In meter AB, variables for token left, timeout left, positive token time, negative token time, and state need to be maintained. State is used to distinguish states A1, A2, B1 and B2. In meter C, variables for token left, timeout left, positive token time, and negative token time need to be maintained (token generate rate is a constant set by controller, no need to be maintained).. Positive token time and negative token time should be shared between 2 meters, to achieve that, 2 more fields need to be added into master table’s entry to record these 2 values. Similarly, an extra token field needs to be added in the master table’s entry. It is used to transfer the remained token value from AB meter to C meter or vice versa. When AB meter leaves state B2 or C meter leaves state C, the meters will write the 3 values (positive token time, negative token time and token fields) to the master table’s entry. The next time when the meters start to process packets, they will copy the two values from the master table.

When there are 3 meters used for each bucket (Three-Meter design option in 3.2.1), 4 table entries needed to be installed in 4 different tables, one of which is the master table. Master table entry needs a new state field and 3 fields for positive/negative token time and token. Meter A needs to maintain variable token left, positive and negative token time. Meter B needs to maintain variables time out, positive token time, negative token time and a toggle bit. The toggle bit is used to distinguish between state B1 and B2, who has different exit behaviors (B1 switches to meter A and B2 switches to meter C). In meter C, variables for token left, timeout left, positive token time, and negative token time need to be maintained (token generate rate is a constant set by controller, no need to be maintained).

4.2 Single Bucket Cost Comparison

Table 2 summarizes cost for one bucket in all three solutions. Let’s first compare the Two-Meter design option and Three-Meters design option. Concerning the flow table cost, Three-Meter design needs to provide everything Two-Meter design needs and one extra table entry. Both design options need provide a C Meter. Three-Meter design’s B meter’s complexity is almost the same as AB meter in Two-Meter design (the only difference is that state in AB meter is an integer to distinguish between 4 states A1, A2, B1, and B2. Toggle bit in B meter is a boolean to distinguish between states B1 and B2). To gain the couple bits saving in B meter compare to AB meter, Three-Meter design has to throw in another A meter (with one more set of token left, positive time and negative time variables) which totally offsets the gain. Based on these comparisons, the Two-Meter design is obviously a better design than Three-Meter design.

Table 2: Cost Comparison Between 3 Design Options

	ABC Meter (Single Meter Design)	**AB Meter +C Meter (Two-Meter Design)**	**A Meter + B Meter + C Meter (Three-Meter Design)**
Table Cost	1 table entry	3 table entries	4 table entries
	No extra field	4 extra fields in master table	4 extra fields in master table
Meter Cost	token left positive time negative time timeout state	token left X 2 positive time X 2 negative time X 2 timeout X 2 state (only in AB Meter)	token left X 3 positive X3 negative time X 3 timeout X 2 (only B, C Meters) toggle bit (only B meter)

The comparison between Single-Meter design and Two-Meter design is not obvious in the meter part. 1-meter design has one very complicated meter which traces everything. Regarding to the variable tracing, ABC meter is the same as AB and one more than C meter (variable “state”). ABC meter needs to trace 4 states which complicates the control logic in the meter’s implementation. However, Two-Meter design needs to implement all these control logics separated in 2 different meters. Therefore, regarding to control logic’s complexity, Two-Meter design doesn’t have clear advantage over Single-Meter design and it almost doubles the number of variables need to be traced (2 copies of token left, timeout, and positive/negative times in two meters)

Finally let’s consider the flow table part between Single-Meter design and Two-Meter design. Single-Meter design doesn’t need any modification on current OpenFlow’s flow table design. No new field needed and only one table needed for flow traffic policing purpose. In Two-Meter design, more serious extra burden is that the new fields need to be accessed in the meter (both read and write). First this requires the meter to remember which table applies it. That adds one more address or link field in the AB and C meter and OpenFlow’s meter setup message. Generally, a meter could be applied in multiple tables and link it with one specific table violate the switch design principle. Secondly, letting the meter access flow table’s entry, even modify table’s entry, could be denied by switch’s architecture for security reason. Single-Meter design totally avoided such extra access to tables from meters and it is the obvious preferred solution for the design.

4.3 Regulator with Multiple Buckets

Now let’s consider the case when multiple buckets are used to regulate one flow. When Single-Meter design is used, only 1 table is needed for all buckets. The same table entry can apply meters from all buckets. Therefore, the number of table entries and tables will not scale up with the number of buckets needed for a flow. When Two-Meter design is used, we have to use a different set of three tables for each bucket. If we try to use the same set of three tables, a packet will be either sent to one table to be policed by all AB meters from all buckets or sent to another table to be policed by all C meters from all buckets. This is wrong. Because of different token bucket sizes and different time out length between different buckets, entering and exiting states are not synchronized between buckets. It is possible that at a time, a packet needs to be policed by AB meters from some buckets and C meters from the other buckets. In this case, the master table will have to forward the packet to both tables and that will force the packet to be policed by both AB and C meters from all buckets, which first is unfairly too strict to the packet. Secondly it is incorrect because one packet's processing moves both AB and C meters' state forward (reducing token twice, countdown timeout twice, once in each meter), which could cause too many packets dropped and the flow is over policed. To solve this problem, a different set of 3 tables needed for each bucket. If there are *n* buckets needed, *3n* tables and *3n* table entries are needed. If Three-Meter design is chosen, *4n* tables and *4n* table entries will be needed for the same reason.

From the discussion above, the conclusion is that when the number of buckets needed in a regulator increase, Single-Meter design's table cost (number of table entries) is constant while Two and Three-Meter designs' cost are linear to the number of buckets. Again Single-Meter design is the obvious best choice.

5 Protocol Design and Meter Implementation

To implement the Single-Meter design discussed in previous sections in a Software Defined Network, the OpenFlow protocol needs to be extended to enable the installation of the new ABC meter in switches. ABC meter needs to be implemented in switches. This section presents the protocol extension and ABC meter implementation in SDN

5.1 OpenFlow Protocol Extension

To install meter bands together with a meter in a switch, the controller in the SDN sends a Meter Modification Message with the structure of opf_meter_mod to switches. The opf_meter_mod structure contains header, command, flags, meter id and a list of meter band head structures [10]. Each meter band header structure (defined as ofp_meter_band_header structure) in the list defines one meter band in the meter. To install an ABC meter, the opf_meter_mod structure will use the header, command, flags and meter id as they are defined in current OpenFlow protocol. Exactly one extended meter band header structure will be attached to the end of the meter modification message to install the newly defined meter band (say ABC meter band). No list needed, there is only one ABC meter band in an ABC meter.

```
/* Common header for all meter bands */
struct ofp_meter_band_header {
    uint16_t        type;     /* One of OFPMBT_*. */
    uint16_t        len;      /* Length in bytes of this band. */
    uint32_t        rate;     /* Rate for this band. */
    uint32_t        burst_size; /* Size of bursts. */
};
OFP_ASSERT(sizeof(struct ofp_meter_band_header) == 12);
```

Figure 2: Current Meter Band Head Data Structure

Currently ofp_meter_band_header structure is defined as Figure 2 [10]. First extension needed is to include the new ABC meter band type in the meter band type enumerates. There are 3 types defined already OFPMBT_DROP = 1, OFPMBT_DSCP_REMARK = 2, and 0xFFFF for experimenter. In the extended version, one new type OFPMBT_TOKEN = 3 is added in the ofp_meter_band_type enumerates for the new ABC meter band type. In an ABC meter band, several constants need to be setup by the controller as listed in Table 3. These constants' values are calculated by the controller based on the flow's S-BIND parameters and sent to switches in ofp_meter_band_header. Therefore, The fields listed in Table 3 need to be added into the ofp_meter_band_header structure. The new ofp_meter_band_header is shown in Figure 3.

Table 3: Constants Setup by Controller in a ABC Meter Band

Constants Field Name	Type	Description
b_start	uint32_t	b_s for the S-BIND segment
b_end	uint32_t	b_e for the S-BIND segment
t_start	uint32_t	t_s for the S-BIND segment
t_end	uint32_t	t_e for the S-BIND segment $r = \Delta b/\Delta t$
conf_lev	uint32_t	Confidence Level ε from S-BIND parameters

```
struct ofp_meter_band_header{
        uint16_t        type;
        uint16_t        len;
        uint32_t        rate;
        uint32_t        burst_size;

        /*new field for CS-BIND regulator*/
        uint32_t        b_start;
        uint32_t        b_end;
        uint32_t        t_start;
        uint32_t        t_end;
        uint32_t        conf_lev;
}
OFP_ASSERT(sizeof(struct ofp_meter_band_header) == 32
```

Figure 3: Extended Meter Band Head Data Structure

5.2 Meter Implementation

An ABC meter band is configured by the controller during meter setup with following constants: b_start, b_end, t_start, t_end, conf_lev, b_jump. Values of these constants are sent to the switch from the controller attached in oft_meter_band_header structure as described in Figure 3 except b_jump. Value of b_jump could be calculated from b_start, b_end, t_start and t_end as in formula (4) by switch.

Table 4: Variables Maintained in an ABC Meter Band

Variable Name	Type	Initial Value
token	int32_t	b_start
state	uint16_t	0
timeout	uint32_t	FFFF
p_time	struct ofp_time	0
n_time	struct ofp_time	0
Time	struct ofp_time	Current time
Rate	uint32_t	0

The ABC meter band also maintains variables during its life time to keep track of the status in the meter. These variables and their initial values are listed in Table 4. Struct ofp_time is based on the format defined in IEEE 1588-2008 with 2 subfields. A "seconds" field (64bit) and a "nanoseconds" field (32bit). "rate" is in kilobits per second. "timeout" is in millisecond.

The implementation of the meter band is given in Figure 4. The procedure described in pseudocode in Figure 4 is executed when this meter band is applied to a packet p. The meter band makes decision either to forward or drop the packet. All meter variables listed in Table 4 will be updated as described in the pseudocode.

```
Procedure PacketProcessing (packet p)
Step 1:
    if token>0, update p_time
    else update n_time
    if rate>0, update token
    if timeout<FFFF, update timeout
Step 2:
    if token>=sizeof(p)
        update token
        forward p
    else
        use formula (5) to make forward decision
        drop or forward p
        if forward, update token
Step 3:
    Update time with current system time
    Update state, token, timeout based on following
    if state=0 AND token<=0,
        state++, timeout=t_start
    if state=1 AND timeout=0,
        state++, token+=b_start, timeout=FFFF
    if state=2 AND token<=0,
        state++, timeout=t_start
    if state=3 and timeout=0,
        state++, rate = (b_end-b_start)/(t_end-t_start),
        timeout= t_end-t_start
    if state=4 AND timeout=0,
        state=0, token+=$b_{jump}$, timeout=FFFF, rate=0
    if state=4 AND token=b_start,
        state=0, timeout=FFFF, rate=0
```

Figure 4: ABC Meter Band Pseudocode

Each incoming packet is regulated by the ABC band in three steps. First step, time related variables are updated such as positive time, negative time, token amount and remaining timeout. In the second step, the forward or drop decision is made based on current token amount, packet size and positive/negative time if needed. In step 3, the band checks to see if switching state is needed. If needed, the state variable, which is used to indicate which one of the 5 states the band is in (A1, B1, A2, B2 and C), will be updated along with the variables that need to be reset during state switching.

6 Future Work and Conclusion

When there are multiple buckets' ABC meters for the same flow setup in a switch, the controller has total control over how to setup the apply instructions in the table to apply these meters. In paper [3], optimization is presented to optimize the number of buckets but no discussion about the sequence in which these buckets should be applied to regulate the traffic. In SDN, adapting the meter application sequence for different flows becomes possible. For a packet that is supposed to be dropped by the regulator, applying the meters in a proper sequence could help to drop most of such packets in first or second bucket's meter band and save the rest meter's processing cost. How the controller can determine such optimal meter application sequence for different flows will be the next step. With the optimization the real cost of the regulator such as CPU cost, memory cost, and communication cost could be evaluated through simulation and experiments.

In this paper, the design options for CS-BIND traffic regulator in SDN network are presented and analyzed. The single table, single meter design is the preferred choice due to its performance advantage and implementation efficiency. To implement the regulator in SDN, OpenFlow protocol needs to be extended to accommodate the new meter band type. The meter band's implementation is presented in pseudocode.

REFERENCES

[1] Lie Qian, Anard Krishnamurthy, Yuke Wang, Yiyan Tang, P. Dauchy, and Albert Conte, *A New Traffic Model and Statistical Admission Control Algorithm for Providing QoS Guarantees to On-Line Traffic*. Proceedings of IEEE Global Telecommunications Conference, 2004, GLOBECOM, vol. 3, pp. 1401-1405.

[2] Lie Qian, Yuke Wang, and Hong Shen, *Token Bucket Based Statistical Regulator for S-BIND Modeled On-Line Traffic*. Proceedings of IEEE International Conference on Communications, 2005, ICC, vol. 1, pp. 125-129.

[3] *Lie Qian, Efficient On-Line Traffic Policing for Confidence Level based Traffic Model*. Transactions on Networks and Communications (TNC), Vol. 4, No. 5, 2015, pp. 28-41

[4] L. Sun, K. Suzuki, C. Yasunobu, Y. Hatano, and H. Shimonishi, *A net- work management solution based on OpenFlow towards new challenges of multitenant data center*, in *Proc. 9th APSITT*, 2012, pp. 1–6.

[5] Sharafat, S. Das, G. Parulkar, and N. McKeown, *MPLS-TE and MPLS VPNS with OpenFlow*, *ACM SIGCOMM* Comput. Commun. *Rev.*, vol. 41, no. 4, pp. 452–453, Aug. 2011.

[6] N. Blefari-Melazzi, A. Detti *et al.*, *An OpenFlow-based testbed for information centric networking*, in Proc. Future Netw. Mobile Summit, 2012, pp. 4–6.

[7] H. Yin *et al.*, *SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains*, Jun. 2012, Internet draft. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf

[8] W. Xia, Y. Wen, C. Foh, D. Niyato and H. Xie, *A Survey on Software-Defined Networking*, IEEE Communication Surveys & Tutorials, Vol. 17, no. 1, pp. 27-51, 1st Quarter 2015

[9] F. Hu, Q. Hao and K. Bao, *A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation*, IEEE Communication Surveys & Tutorials, Vol. 16, no. 4, pp. 2181-2206, 4th Quarter 2014

[10] *OpenFlow Switch Specification, version 1.5.1*, Open Networking Foundation, March 26, 2015, [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[11] S.H. Yeganeh, A. Tootoonchian, and Y. Ganjali. *On scalability of software-defined networking*. IEEE Commun. Mag., 51(2):136–141, February, 2013.

[12] E. Kissel, G. Fernandes, M. Jaffee, M. Swany, and M. Zhang, *Driving software defined networks with xsp*. In SDN12: Workshop on Software Defined Networks, 2012.

[13] J. Qiu and E. W. Knightly, *Measurement-based admission control with aggregate traffic envelopes*, IEEE/ACM Transactions on Networking, vol. 9, no. 2, pp. 199-210, April 2001.

[14] B. Statovci-Halimi, *Adaptive admission control for supporting class-based QoS*, 2010 6th EURO-NF Conference on Next Generation Internet (NGI), pp. 1-8, 2010.

[15] L. Breslau, E. W. Knightly, S. Shenker, I. Stoica, and H. Zhang, *Endpoint admission control: architectural issues and performance*, Proc. Of ACM SIGCOMM'00, pp. 57-69, September 2000.

[16] V. Elek, G. Karlsson, and R. Ronngre, *Admission control based on end-to-end measurements*, Proc. Of IEEE INFOCOM, March 2000.

[17] E. W. Knightly, *H-BIND: a new approach to providing statistical performance guarantees to VBR traffic*, Proc. Of IEEE INFOCOM '96, pp. 1091--1099, March 1996.

[18] E. W. Knightly and N. B. Shroff, *Admission control for statistical QoS: theory and practice*, IEEE Network, vol. 13, Issue 2, pp. 20--29, 1999.

[19] H. A. Harhira, and S. Pierre, *A Mathematical Model for the Admission Control Problem in MPLS Networks with End-to-End delay guarantees*, Proc. Of 16th International Conference on Computer Communications and Networks, pp. 1193-1197, 2007.

[20] S. Alwakeel, and A. Prasetijo, *Probability admission control in class-based Video-on-Demand system*, 2011 International Conference on Multimedia Computing and Systems (ICMCS), pp. 1-6, 2011.