

Controller scheduling for continued SDN operation under DDoS attacks

Sunghoon Lim, Seungnam Yang, Younghwa Kim, Sunhee Yang and Hyogon Kim

There exists a way that attackers can identify software defined networks (SDNs). Knowing the vulnerabilities of a SDN, the attackers can mount a saturation attack on the SDN controller with the aim of incapacitating the entire SDN. Therefore, the controller should have an architecture to weather out such an attack while continuing operation. A scheduling-based architecture is proposed for the SDN controller that leads to effective attack confinement and network protection during denial of service (DoS) attacks.

Introduction: There exists a way that attackers can identify networks that employ software defined networks (SDNs) for control [1]. To incapacitate an identified SDN, the attackers could mount a saturation attack towards the SDN controller simply by sending a large volume of new flows, e.g. from a botnet to a known server in the given SDN [2]. Since every new flow should be handled by the controller for flow entry creation on the traversed switches, the SDN controller can be overwhelmed by flow creation type denial of service (DoS) attacks. A flooded controller will show poor responsiveness to the flow requests from other unaffected flow switches, so that they too will be indirectly rendered less capable of handling new flows.

Although provisioning large resource for the controller is one way to cope with the aforementioned DoS attacks, a more systematic approach is desired. Unfortunately, there is a dearth of even the most rudimentary work on how to design the SDN controller for continued operation under the aforementioned attack scenario. Although there is a body of literature on SDN security, most of it is focused on mitigating attacks at flow switches where the attack traffic actually flows [3]. Therefore, in this Letter, we discuss an architecture that helps the controller weather out the DoS attacks targeted at it independently of the attack mitigation measures working in the switches on the actual attack flow paths. Specifically, we propose a scheduling-based scheme that contains most of the attack traffic at attack ingress switches so that the SDN network as a whole can continue normal operation.

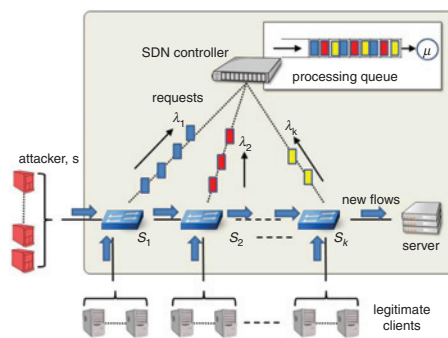


Fig. 1 One-dimensional model of DDoS attack on SDN-controlled network

Scheduling-based attack containment: A real SDN-controlled network configuration could be more complex, but for ease of analysis let us consider a simplified model depicted as in Fig. 1. We assume that the attacker(s) do not know the exact SDN controller location (e.g. IP address), so they send a large number of new flows to a known service ('server') in the network as a means of overwhelming the controller. Suppose that the attack flow path $P = (S_1, S_2, \dots, S_k)$ is composed of k flow switches from the attack ingress switch S_1 to a server which is connected to S_k . All the switches traversed by the attack are shared by legitimate clients. When the attack commences, attack packets arriving at switches begin to generate spurious flow entry creation requests from the switches in P . Meanwhile, legitimate clients connect to the server using TCP. However, a connection setup attempt may fail because the controller overwhelmed by the spurious flow creation requests cannot respond to the legitimate request at one of the switches (e.g. S_2) on the intended TCP connection path (e.g. client $\rightarrow S_2 \rightarrow S_3$). In particular, if the TCP connection is not established after a predefined number of TCP SYN segment retransmissions because a corresponding flow

entry is not made at S_i ($1 \leq i \leq k$), the connection setup is deemed a failure.

Our proposal is simple: modify the controller model in Fig. 1 so that the single request processing queue at the controller is logically subdivided into k queues, each of which corresponds to a flow switch. Namely, the requests from the same switch are enqueued in the corresponding logical queue, and the controller serves these logical queues with a scheduling discipline, e.g. round-robin. Consequently, it can create an isolated allocation of controller processing capacity to each switch. For convenience, we call this scheme 'MultiQ'. To evaluate its impacts, we compare it with two other schemes. In 'Static', the flow switches are directed to begin rate limiting on flow requests whose rate is $\lambda_i \geq \lambda_{\max}$ ($1 \leq i \leq k$), where λ_{\max} is a predefined parameter. In OpenFlow, OFPF_METER_MOD message with OFPM_CONTROLLER as the meter ID can be used to set up a meter for this purpose [4]. The second scheme is 'SingleQ', which is the model depicted in Fig. 1. For comparison of these three schemes, we perform an emulation on Mininet [5] that implements OpenFlow 1.3 switch logic. We assume that a botnet mounts a DDoS attack, where each bot sends UDP flood (or SYN flood), with each carrying a different flow identity. The controller is assumed to have the processing capacity μ [reqs./s] and has queue capacity L_{\max} [reqs.]. In the Static scheme, we assume $\lambda_{\max} = \mu/k$. In MultiQ and SingleQ, each queue has space for L_{\max}/k flow requests, with $k = 1$ for SingleQ.

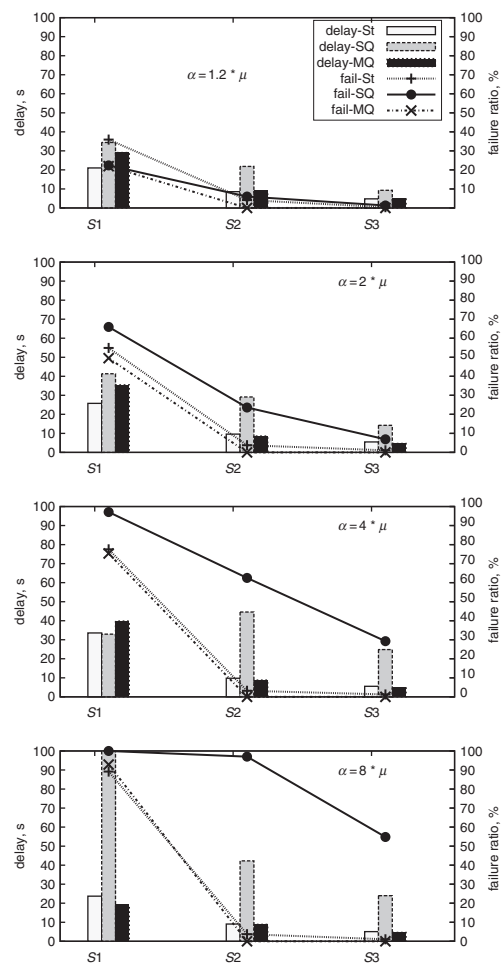


Fig. 2 TCP connection setup delay (bars) and connection failure ratio (lines)

Evaluation: To overload the controller in our emulation, we set $\alpha > \mu = 50$, where α is the aggregate flow arrival rate purely from the DDoS attack from the botnet (i.e. besides legitimate requests). In particular, we set α/μ to be between 1.2 (mild attack) and 8 (severe attack). Legitimate clients are modelled to be 'persistent'. Namely, as soon as they successfully establish a TCP connection by creating the flow entries on all the switches along the connection path or they fail to do so after the maximum number of retransmissions, they immediately initiate another connection. This process repeats until the emulation ends at $t = 127$ s. During this process, we measure the TCP connection

setup failure rate R_f , the connection setup delay D_s (for successful connections only), and the number of successfully established connections across the system (N_s). We set the legitimate traffic injected at each switch to be under 0.1μ . Fig. 2 shows the performance of the three compared schemes for $k=3$ (the smallest non-trivial topology) and $L_{\max}=150$. With $\mu=50$, L_{\max} provides 3 s buffering and multiple retransmitted TCP SYN for the same connection are not queued together. Finally, higher numbers of switches k result in qualitatively similar results.

Let us first discuss the result at the front switch S_1 that bears the brunt of the attack. We see in Fig. 2 that with the exception of $\alpha=1.2\mu$, SingleQ shows the worst R_f . It also exhibits significantly higher D_s . In contrast, MultiQ and Static show comparable delays and failure rates. Most importantly, their interior (S_1 and S_2) protection performance is very good. The delays are under 10 s and the failure rates are close to zero even for very high μ 's. This starkly contrasts with the SingleQ failure ratio that can increase to more than 90% at S_2 under 8μ . We can observe that an isolation mechanism, either the rate limiter at the flow switches or the locally separated queue at the controller, is necessary in the SDN architecture for unaffected operation under severe saturation attacks on the controller.

Although MultiQ and Static are comparable in terms of R_f and D_s , Static has issues. First, it has smaller N_s than MultiQ (see Table 1), which stems from its inflexibility. Even if the controller has idle capacity, each switch cannot request higher than the rationed rate, e.g. $f(k)=\mu/k$. Overbooking could be considered by having $\Sigma f(k) > \mu$, but it would cause the controller to become more like SingleQ as $\Sigma f(k)$ gets larger. Secondly, rationing is not scalable in the number of switches k . If k grows large (it is expected to be so in real SDNs), μ/k will be a severe performance-limiting factor in Static. MultiQ does not suffer from the scalability problem.

Table 1: Performance difference in N_s between MultiQ and Static

α	Scheme	Successful connections	Utilisation (%)
1.2μ	Static	1497	88.5
	MultiQ	1862	100
2μ	Static	1350	91.9
	MultiQ	1788	100
4μ	Static	1296	92.4
	MultiQ	1761	100
8μ	Static	1345	93.6
	MultiQ	1748	100

Finally, we can attempt to validate the R_f values from the emulation above, through analysis. Although a thorough analysis would require a model complete with α, μ , legitimate request rate and TCP retransmission parameters among others, we will use a simpler model in this Letter. The legitimate client's connection setup behaviour can be described with a Markov model where each state $\langle n, j \rangle$ denotes the n th SYN transmission trying to set up the flow entry at the j th switch on the path. Each transition in the Markov chain corresponds to a connection setup attempt (i.e. SYN re/transmission) and a_j refers to the probability of successful flow creation at the j th switch. Note that a connection setup succeeds if the flow creation request succeeds at each flow switch on the connection path, first to the server and then back to the client. For instance, for the legitimate users entering the network through S_1 , the flow entries should be created across $S_1, S_2, \dots, S_k, S_k, \dots, S_2, S_1$, possibly using multiple SYN/[ACK] retransmissions in the process. For the forward sub-path ($S_1 \rightarrow S_k$), any failure will cause the TCP connection originator to retransmit the SYN. However, for the reverse sub-path ($S_k \rightarrow S_1$), any failure will cause the TCP destination will retransmit SYN/ACK. Fig. 3 shows the Markov model for legitimate clients that enter the network through S_1 . Let P_{s_i} be the probability of one-way connection setup success with i SYN transmissions. Then, from the Markov model we obtain $P_{s_1} = a_1 a_2 \dots a_{2k} = S$, $P_{s_2} = \sum_{i=1}^{2k} (1 - a_i) S$, $P_{s_3} = \sum_{i=1}^{2k} \sum_{j=2}^{2k} (1 - a_i)(1 - a_j) S$ and so on. In general, the one-way connection setup succeeds with l retransmissions whose probability is given as

$$P_{s_{l+1}} = \sum_{i_1=1}^{2k} \dots \sum_{i_l=l-1}^{2k} (1 - a_{i_1})(1 - a_{i_2}) \dots (1 - a_{i_l}) S$$

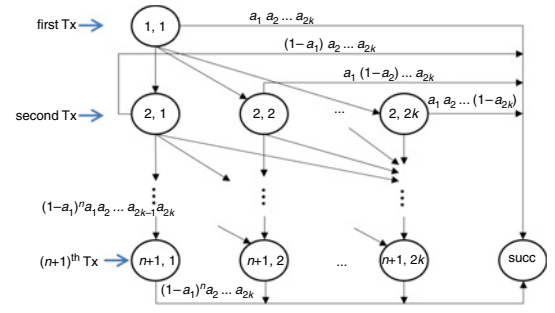


Fig. 3 Markov model for legitimate clients that enter the network at S_1

Then we can compute $R_f = 1 - \sum_{l=0}^N P_{s_{l+1}}$, where N is the maximum number of retransmission attempts. For instance, for $\alpha=4\mu$ where SingleQ and MultiQ show non-negligible difference in R_f for the legitimate clients entering through S_1 , we can apply the simple model. We use the a_j values produced by emulation here: $(a_1, a_2, a_3) = (17, 20, 20.9\%)$ in SingleQ and $(9.3, 80.8, 73.3\%)$ in MultiQ, respectively. The success probabilities are symmetric on the return sub-path. Expectedly, a_j 's in SingleQ are similar across switches, as they share a single queue at the controller. Moreover, a_1 with MultiQ is much lower than with SingleQ. This is because of the logical queue isolation at the controller: at the cost of protecting S_2 and S_3 , the drop rate for S_1 is much higher. Finally, a_3 under MultiQ is lower than for S_2 . This is because the legitimate connection requests successfully passing S_2 are added to S_3 legitimate requests. With $N=6$ and $K=3$, the Markov model gives us R_f for SingleQ and MultiQ as 98.4 and 87.2%, respectively. The model is slightly more pessimistic than emulation in MultiQ, due to the fact that the model ignores the N SYN/ACK retransmissions possible by the server.

Conclusion: Upon a DDoS attack, attack traffic classification and filtering should be quickly started at the flow switches on the attack perimeter. Meanwhile, however, the SDN controller can be temporarily overwhelmed by the spurious flow requests generated by the attack traffic. This Letter shows that a simple scheduling-based isolation of flow requests processing at the controller can prevent the attack from affecting other flow switches inside the attack perimeter by way of the controller. In future, we will investigate how the scheduling mechanism should be designed for better protection of the SDN operation.

Acknowledgment: This research was funded by the Ministry of Science, ICT & Future Planning (MSIP), Korea, in the ICT R&D Program 2014.

© The Institution of Engineering and Technology 2015

Submitted: 28 January 2015 E-first: 20 July 2015

doi: 10.1049/el.2015.0334

One or more of the Figures in this Letter are available in colour online.

Sungheon Lim, Seungnam Yang and Hyogon Kim (Korea University, Seoul, Republic of Korea)

✉ E-mail: hyogon@gmail.com

Younghwa Kim and Sunhee Yang (ETRI, Daejeon, Republic of Korea)

References

- Shin, S., Gu, G., and Anderson, P.: 'Attacking software-defined networks: a first feasibility study'. Proc. of ACM HotSDN, 2013
- Kreutz, D., Ramos, F.M.V., and Verissimo, P.: 'Towards secure and dependable software-defined networks'. Proc. of ACM HotSDN, 2013
- Braga, R., Mota, E., and Passito, A.: 'Lightweight DDoS flooding attack detection using NOX/OpenFlow'. Proc. of IEEE LCN, 2010
- Open Networking Foundation: 'OpenFlow switch specification 1.3.4', March 2014
- Mininet: 'An instant virtual network on your laptop (or other PC)'. Available at <http://www.mininet.org>