WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

SDN Architecture for Scalable Resource Management for Big Data Governance in Cyber Security

Ankit Kumar[1],Onkar Priyadarshi[1], Pallavi M S[1],Rounak Bajaj[1], Prof. Sagari S M[2]

[1]Student, Department of Computer Science & Engineering, Dayananda Sagar College of Engineering, Bengaluru 560078, India.
[2]Professor, Department of Computer Science & Engineering, Dayananda Sagar College of Engineering, Bengaluru 560078, India.

ABSTRACT:

Software-Defined Network (SDN) is a networking paradigm featuring the separation of the control plane from the data plane of the network devices. SDN offers distinct advantages over traditional networks due to its enhanced centralized management and network programmability. The control plane is physically distributed but logically centralized to meet the issue of scalability. Since network traffic is both spatially and temporally dynamic, a multi-controller organization in SDN needs a load balancing procedure to deal with local overloads effectively. The primary motivation behind this work is to provide a framework that learns from the network traffic and seeks to balance it quickly, decrease the unnecessary migration cost and improve the in-packet request-response rate. To this end, we use the load ratio deviation between the controllers and generate optimal migration triplets, consisting of the migrate-out and migration-in domains and a set of switches for migration. To obtain the global optimal controller load balancing with the lowest cost, we utilize an online Q-learning with the constraints of maximum efficiency and no migration conflicts. Our work has shown good results in a variety of scenarios with different load distribution on switches. Using simulation results we demonstrate the convergence behaviour of the proposed scheme to the optimal policy.

SDN has abilities to provide superior management and security of a network and allows us to program the network for better ease of use and performance. However, SDN is vulnerable to attacks, DDOS attacks are the most dangerous and threatening attacks in a network, as it can flood the network and block access to the server network with large counts of packets and make use of network resources to deny response for further requests incoming. In a cloud environment DDOS attacks are known only to increase. The method presented is to combine statistical and machine learning methods to efficiently detect and mitigate DDOS attacks in SDN. Implementation of this method is done using RYU controller and Mininet network simulator with OpenFlow SDN protocol, the machine learning algorithm implemented has achieved accuracy of 99.26% and a detection rate of 100% in detecting and mitigating DDOS attacks in a software defined network

Keywords:Load Balancing, SDN Networks, DDOS Attack

Introduction

Cloud computing trend has seen a rapid growth in the last couple of years in fields of industries and academic, due to the essential characteristics it can offer over traditional network. Software define network (SDN) has seen significant growth in the field of networking and trending cloud networks. SDN is a networking technology which improves the network performance and management, it enables us to have a centralized management of the network and allows us to program the network devices (Xia et al. (2015)). Software defined networking decouples the data and control plane of the network device and enables the control plane to be programmed by a SDN controller. SDN architecture is comprised of three-layer namely infrastructure layer where are the networking devices like switches and hosts are present, control layer where the controller is implemented and the application layer for networking applications. SDN is used to monitor and control the network from in single place, which helps in changing and configuring network devices easily in the network. It interns improves the scalability, performance, controllability and provides flexibility and cloud management. Software defined network-based cloud environments are deployed in cloud computing networks to have better security and control over the network and provide networking as a service.
Researchers have developed a distributed SDN architecture to alleviate the issues associated with a single controller. The goal is evenly distributing the network's load among all of the controllers. Moreover, if a controller fails, its load should be ideally distributed among the remaining controllers, thus making the architecture scalable and robust. In this context, researchers have introduced the controller placement problem (CPP) since the network performance of a distributed SDN architecture depends on the layout of the controllers.

In, authors have studied of work on controller placement problem in SDN by researchers. In these papers, CPP in SDN is mostly divided into based on either performance matrices or network traffic i.e., static or adaptive (dynamic) controller placement problem. In scenario of static controller placement, main concern of researchers is optimal placement of the controllers to control the set of switches based on one-time consideration of the performance metrics. In adaptive controller placement, the initial placement of controllers are based on a particular performance metric. Thereafter, the assignment of the switches to controllers are changed dynamically in time or space in accordance with variation in network traffic. Thus, balancing the controller load basically boils down to the problem of switch migration.

We discovered that switching a switch is based on historical or present traffic conditions during our examination of the state of the art on switch migration. However, researchers have discovered that for large time scales (hours or weeks), Internet traffic has a seasonal trend, whereas for small time scales, it is highly chaotic (seconds or minutes). As a result, methods for learning traffic behavior are being developed, and then a controller is being chosen, and the switches are being reassigned to that controller. This method will show to be far more advantageous. Learning the traffic of genuine SDN networks would be the biggest problem here. The issue of Internet traffic modelling is still a work in progress. For a variety of MDP-

modeled situations, the reinforcement learning (RL) algorithm is a well-studied machine learning approach. A learning agent senses the environment and takes an action at each state to complete a well-defined goal and maximize the collected action rewards in reinforcement learning. The RL algorithm has the advantage of being able to make time-series decisions in a discrete space while evolving action rules, which is important for load balancing problems in SDN, according to its learning capabilities. Thus, our main objective of this work is to use reinforcement learning to learn the traffic behavior to balance the load of the network in a dynamic environment. We make use of an online ε-greedy based Q-learning which makes the reward cost optimization along with the exploration and exploitation strategy. Based on the result, we can claim that our method improves load balancing in the dynamic environment of SDN.

Related works

Dixit et al.[3] [4] proposed to address the issue by using the equal controller mode (specified in OpenFlow v1.2) while transitioning a controller from master to slave. They also propose expanding or contracting the controller pool depending on whether the controller load jumps the upper or lower threshold.Wang et al. [13] proposed a framework Switch Migration-based Decision Making (SMDM) where they compute load diversity, a ratio of controller loads. If the load diversity between two controllers exceeds a certain level, switch migration occurs. The switches selected for migration are those with less load and higher efficiency.Hu et al. [6] proposed Efficiency-Aware Switch Migration (EASM), where they measure the degree of load balancing using the normalized load variance of the controller load. In the event of the load difference matrix exceeding a threshold, the controller load is presumed to be imbalanced. The threshold is a function of the difference between the maximum and minimum controller load.Filali et al. [5] use the ARIMA time series model to predict switch's load. The forecasting allows finding the time step at which a controller will become overloaded and accordingly schedule a switch migration in advance. The authors use a predetermined threshold to identify overloaded controllers.The work by Zhou et al. [14] are among those few who consider the problem of load oscillation due to inappropriate switch migration. The issue of load oscillation is where underloaded controllers used to offload the traffic load of overloaded controllers, themselves become quickly overloaded. Ul Haque et al. [12] address to balance the variation in the controller load by employing a controller module, which is a set of controllers. Their method estimate the number of flows that the switches will produce on a regular interval and accordingly activates the appropriate number of controllers.Chen et al. [2] use a game-theoretic method to solve the problem of controller load balancing. The underloaded controllers are modelled as players who compete for switches from overloaded controllers. The payoffs are determined when an underloaded controller is selected as the master controller of a victim switch.Sangho et al. [9] propose the switch-aware reinforcement learning load balancing (SAR-LB) for achieving balanced Load Distribution with Reinforcement Learning-Based Switch Migration in Distributed SDN Controllers.

Methodology

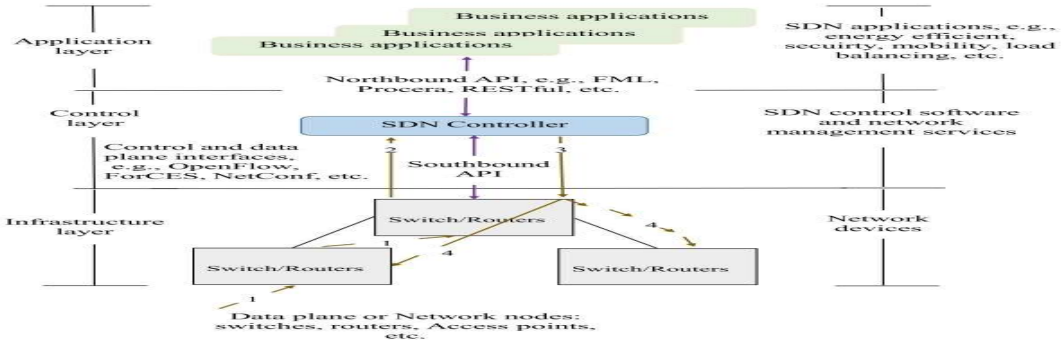
We use a ε-greedy strategy to train the Q-network by investigating a random action with probability ε and exploiting an action with probability 1 – ε that maximizes the predicted long-term payoff. We acquire a new state and the resulting reward for each action we take. This is treated as a real-life experience that will be saved for later use. This procedure's algorithm is listed below.
Let's have a look at each of the steps we take in this algorithm.
•Initialization: We first initialize the Q-matrix to all zeros. The hyper- parameters are set next.
We set γ to 0.99, α to1 0.6 , and b(n) to 1(n+1)n
We also initialize ε to 0.7. We then decrement this by 0.0025 per iterationtill it reaches a minimum of 0.05. Finally, Smax, Sn, and l0 are initialized.
•State updation: For each iteration we first update the state based on the incoming or outgoing flows. The state consists of number of flows on each controller as mentioned earlier.
•Decide action: Next we make a choice between exploration and exploitation based on ε value in that current iteration. Based on this, we decide upon one of the 3 actions. Unless the action is do nothing, the next state isdifferent from the current state. We make necessary updations and store the next state.
•Reward calculation: Now that we have the next state, we make calculations for reward, cost and discounted reward. Update the Q-value for the current state-action pair. Finally we set the current state as next state and update ε value. Additionally in this step we make a note of the cumulative discount reward, current discrete coefficient and number of switch exchanges, which will be later necessarily for making plots.

Algorithm 1: Selection Algorithm of Migration Domain

input : $R = \{R_{C1}, R_{C2}, \ldots, R_{Cn}\}$: Controller's load ratio
$\overline{R}$: Mean load ratio of all controllers
D: Discrete Coefficient between the controllers
output: Out-migration domain O, in-migration domain I, migration queue Q
initialize: Set O, I and Q to empty sets
begin
foreach R_{Ci}, $R_{Cj} \in R=\{R_{C1}, R_{C2}, \ldots, R_{Cn}\}$ **do**
$D_{Ci,Cj}$: Calculate the discrete coefficients
if $D_{Ci,Cj} > D$ *and* $R_{Ci} > \overline{R}$ *and* $R_{Ci} > R_{Cj}$ **then**
Add Ci to the out-migration domain O
Add Cj to the in-migration domain I
if $D_{Ci,Cj} < D$ *and* $R_{Ci} < \overline{R}$ *and* $R_{Ci} < R_{Cj}$ **then**
Add Cj to the out-migration domain O
Add Ci to the in-migration domain I
Add (C_i, C_j) to the migration queue Q

OpenFlow is a communications protocol for SDN that gives access to the forwarding plane of a network switch or router over the network in software defined network.
Mininet is a network emulator which creates a network of virtual hosts, switches, con- trollers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking.In a virtual environment to simulate a large network, Mininet is the open-source network simulator for Software Defined Network. The primary reason to use the Mininet is that it supports OpenFlow Protocol, which is essential for the network configuration and com- putation for Software Defined Network. It also provides an inexpensive platform for developing, testing, and creating custom topologies in the network.
Ryu Controller is an open, software-defined networking (SDN) Controller designed to increase the agility of the network by making it easy to manage and adapt how traffic is handled. Which is a Python-based programmable controller tool.

Iperf is a network performance tool that is used to measure the bandwidth and data- gram loss in a network. This project measures the Transport Control Protocol (TCP) and User Datagram Protocol (UDP) network throughput and data streams. The iperf tool helps to measure the network performance by creating a client and server functionality for both source and destination node.
The presented SDN framework, data plane has multiple node/hosts virtually created using mininet and all these are connected to the openflow switch which the defines the SDN protocols and the openflow protocol communicates with the control plane of the framework. Control plane controls the data plane and the switches and define rules and also monitors the network traffic flow, here Ryu controller is used as the controller which provides the programming capabilities and allows us to control the routing operations in
the network. The control plane is programmed using pyhton as ryu is a python based controller and uses a python based API to communicate with the application layer, which in our case is network traffic applications.
The network topology is designed using mininet network simulator, the network has 25 hosts/nodes and one single openflow switch and one ryu controller. All the hosts are connected to the switch and the switch is connected to the controller. All of these hosts and switch are controlled by the ryu controller, any port under attack will be blocked immediately.



4. Dataset

We consider a system with k = 5 controllers and generate data for a variety of scenarios. To simulate real world environments we consider the following cases:
•Heavy Load: This is the scenario where the load on switches is more. This means that the arrival rate on the switches will be much higher than the service rate. In this case we consider the following rate of values to generate the data.
- λ = random numbers in range (50, 100)
- μ = a random number in range (1, 5)

•Light Load: This is the scenario where the load on switches is less. This means that the arrival rate on the switches will be much smaller than the service rate. In this case we consider the following rate of values to generate the data.
- λ = random numbers in range (1, 5)
- μ = a random number in range (50, 100)

•Skewed Load: This is the scenario where the load on switches is skewed. This means that the arrival rate on the switches will be much higher than the arrival rate on other switches. For our purpose let's put a very high arrival rate on randomly chosen 2 switches. In this case we consider the following rate of values to generate the data.
- λ = high: random numbers in range (50, 100); low: random numbers in range (5, 10)
- μ = a random number in range (25, 50)

4. Experimentation & Results

The figures for all 3 cases are shown below. We make the following observations:
•The cumulative discounted reward converges to a specific value in all cases. This happens after roughly 400 iterations and is a good sign because this validates that the Q-Learning is functioning effectively.
•The discrete coefficient is much more random and higher in case of light load than in the case of heavy and skewed load as shown in Fig 7.5. In the latter cases, we notice that the discrete coefficient gradually decreases till it stays roughly in a small range, which means that the system load has stabilized and is equally distributed.
•The number of switches exchanged is plotted cumulatively, and we notice that it is smaller in case of light load. For heavy and skewed systems this number reaches around 4000 which is roughly close to the chosen Smax.
•For heavy load scenario, we in Fig 7.2 see that the discrete coefficient is in the range of 0.25-0.5 roughly. This is a good value, since the system is

constantly being disturbed by new arrivals and departures or flows. We also see that the cumulative discounted reward converges to -55 roughly around 400 iterations. After that the change is very small to be visible on the curve.•For light load scenario, we see that the discrete coefficient is random. This is justified by the fact that the discrete coefficient here is a measure of the deviation of the number of flows in each switch. Since flows are serviced very quickly, the system usually stays in a state where load is on a few controllers only and the others have 0 load. This gives a large D value.
•For skewed load scenario in Fig 7.8, we observe that the discrete coefficient values are more random that the heavy load scenario. This is also backed by the fact that in case of skewed load, inflow is constantly more on a few switches and very less on others.

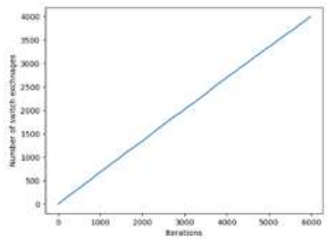


Fig. 7.4 Number of switches exchanged with time - Load Heavy

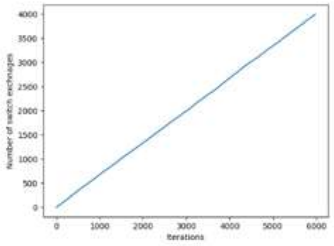
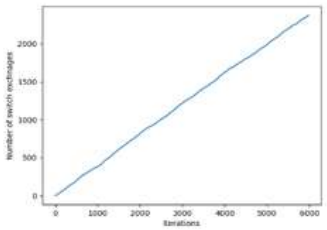


Fig. 7.7 Number of switches exchanged with time - Load Lightig.



7.10 Number of switches exchanged with time - Load Skewed

The presented method uses both statistical and machine learning methods to detect and mitigate DDOS attacks in a software defined network. The implemented method requires to train the SVM ML algorithm for detecting the attack in a network. Detection and Mitigation takes place after the data has been collected and the control- ler is set to detection state, then when the normal traffic is generated the SVM algorithmpredicts it as normal traffic and when the attack traffic is generated it instantly detects the traffic as DDOS attack traffic and blocks the port from which the traffic is incoming. Once the port has been blocked the controller is set to a 120sec hardtime, after which it unblocks the port. But, if the attack is still active it again detects and blocks the port for another 120 seconds. After blocking the normal traffic flow is allowed in the network from other ports. This process keeps on going as long as the attack lasts.

Conclusion and Future works

This report focused on SDN Controller load balancing with real traffic networks using reinforcement learning. The results obtained have motivated us to extend this work. Our next goal would be work on real data obtained from different sources and also explore other alternatives such as deep Q-learning to capture the network traffic behavior to balance the controller and efficient switch migration considering dynamic networks.

Software defined network provides us the capabilities to design and perform operations in the network by programming which is not case with traditional networks. Using SDN to detect and mitigate DDOS attacks in cloud environment was the main goal of this work. The implemented method is a combination of statistical features like source of IP, speed of flow entries , flow count and ratio of flow-pair and SVM machine learning algorithm to detect and predict DDOS attacks in the network, experimented results shows the presen ted method can provide efficient accuracy and malicious traffic detection rate of accurate percentage with zero false predictions of the traffic. However, security is never full proof and can always be shatter same way implemented method has a drawback, an attack from trusted IP sources can be used to send malicious traffic in the network which the SVM won't be able to predict.

In future the implemented method can be designed to have multiple switches and controller in the network with a comprehensive network packet analyzer. Currently features are being used in the statistical analysis, furthermore features can be extracted and used with ML algorithm to have better and accurate prediction of malicious traffic.

References

1. Q. Yan, F. R. Yu, Q. Gong and J. Li, "Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges," in IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 602- 622, Firstquarter 2016.
2. Y. Xu, H. Sun, F. Xiang and Z. Sun, "Efficient DDoS Detection Based on K-FKNN in Software Defined Networks," in IEEE Access, vol. 7, pp. 160536-160545, 2019.
3. S. Dong and M. Sarem, "DDoS Attack Detection Method Based onImproved KNN with the Degree of DDoS Attack in Software-Defined Networks," in IEEE Access, vol. 8, pp. 5039- 5048, 2020.
4. S. Dong, K. Abbas and R. Jain, "A Survey on Distributed Denial of Service (DDoS)Attacks in SDN and Cloud Computing Environments,"in IEEE Access, vol. 7, pp. 80813-80828, 2019.
5. A. Santos da Silva, J. A. Wickboldt, L. Z. Granville and A. SchaefferFilho, "ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN," NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, Istanbul, 2016, pp.27-35.
6. Z. Chen, F. Jiang, Y. Cheng, X. Gu, W. Liu and J. Peng, "XGBoost Classifier for DDoS Attack Detection and Analysis in SDN-Based Cloud," 2018 IEEE International Conference on Big Data and Smart Computing (BigComp), Shanghai, 2018, pp. 251-256.
7. N. Meti, D. G. Narayan and V. P. Baligar, "Detection of distributed denial of service attacks using machine learning algorithms in software defined networks," 2017 International Conference on Advances in Computing, Communications and Informatics, Udupi, 2017, pp. 1366-1371.
8. T. A. Tang, L. Mhamdi, D. McLernon, S.A. R. Zaidi and M. Ghogho, "Deep learning approach for Network Intrusion Detection in Software Defined Networking," 2016 International Conference on Wireless Networks and Mobile Communications, Fez, 2016, pp. 258- 263