Contents lists available at Science-Gate

International Journal of Advanced and Applied Sciences

Journal homepage: http://www.science-gate.com/IJAAS.html

Convergence time aware switch migration algorithm for SDN (CTSMA) cloud datacenter

S. R. Deepu *, B. S. Shylaja, R. Bhaskar

Department of Information Science and Engineering, Dr. Ambedkar Institute of Technology, Bangalore, India

ARTICLE INFO

Article history:
Received 18 February 2022
Received in revised form
15 May 2022
Accepted 19 May 2022

Keywords:
Datacenter network
Topology aware
Switch migration
Convergence time
Response time

ABSTRACT

Multi-controller deployment in a software-defined network improves the system's stability and scalability. However, since network traffic fluctuates, it presents a new problem for balancing loads on remote controllers. Controller Adaption and Migration Decision (CAMD) and Dynamic and Adaptive Load Balancing (DALB) frameworks are developed for efficient balancing of load on the controller to solve the problem of controller overload due to dynamic network traffic. CAMD was considered to be more efficient than DALB, but when the network is more dynamic, and the incoming traffic flow is elephant flow this leads to the overall reduction in system performance. This study proposed a Convergence Time aware Switch Migration Algorithm (CTSMA) that solved the network challenge when the network is more dynamic and incoming traffic flow is more. This research developed an enhanced switch migration algorithm to address the network difficulty of dynamically changing incoming load. Because of the imbalanced distribution of load on the controllers, processing flows will have longer response times and the controllers' throughput will be reduced. Switch migration is the best method of resolving the issue. Present techniques, on the other hand, focus solely on load balancing performance while ignoring migration efficiency, thereby leading to large migration costs and excessive control overheads. To increase the load and migration efficiency of controllers, this research work developed a convergence time aware switch migration method. To find the group of underloaded controllers in the network, the improved framework looked at controller volatility and average load status. Performance comparison indicators included controller throughput, reaction time, and convergence time. According to simulation studies, CTSMA outperforms CAMD by cutting controller reaction time by roughly 6.1%, increasing controller throughput by 8.0% on average, keeping a decent load balancing rate, lowering migration costs, and maintaining the best load balancing rate.

1. Introduction

The software-defined network (SDN) (Yan et al., 2015; Hu et al., 2018) which separates the data, control plane, and moved control plane to a centralized controller, is the next phase of network architecture. The "Network Operating System" is the controller, which has a complete picture of the network topology as well as complete control over network resources. In the data plane, devices such as routers and switches serve only as forwarding devices.

In a traditional IP network, the hardware and software are purchased as a package from a vendor and configured as needed. Any extra features that need to be introduced or any software bugs that need to be fixed must be handled by the vendor. Because vendor software is proprietary, it has slowed the development and deployment of new services. SDN removes vendor dependency. The hardware and software are offered separately, and users can load their preferred network operating system. The controller determines how data flows in the data plane (Canini et al., 2022).

SDN is a networking technology, i.e., still in its infancy. The basic goal of SDN architecture is to make it simple to program networks and control network hardware via software. The components of the SDN framework are:

* Corresponding Author.
Email Address: deepusrd@gmail.com (S. R. Deepu)
https://doi.org/10.21833/ijaas.2022.08.013
Corresponding author's ORCID profile:
https://orcid.org/0000-0002-6104-3149

1. OpenFlow switches (Data plane)
2. Controller (Control plane)
3. SDN applications

The data plane is made-up of forwarding devices like switches and routers that are linked together using wired and wireless radio channels. These devices include flow rules that allow them to do actions on incoming packets such as dropping or forwarding packets or sending them to a particular port. Fig. 1 shows the SDN controller.

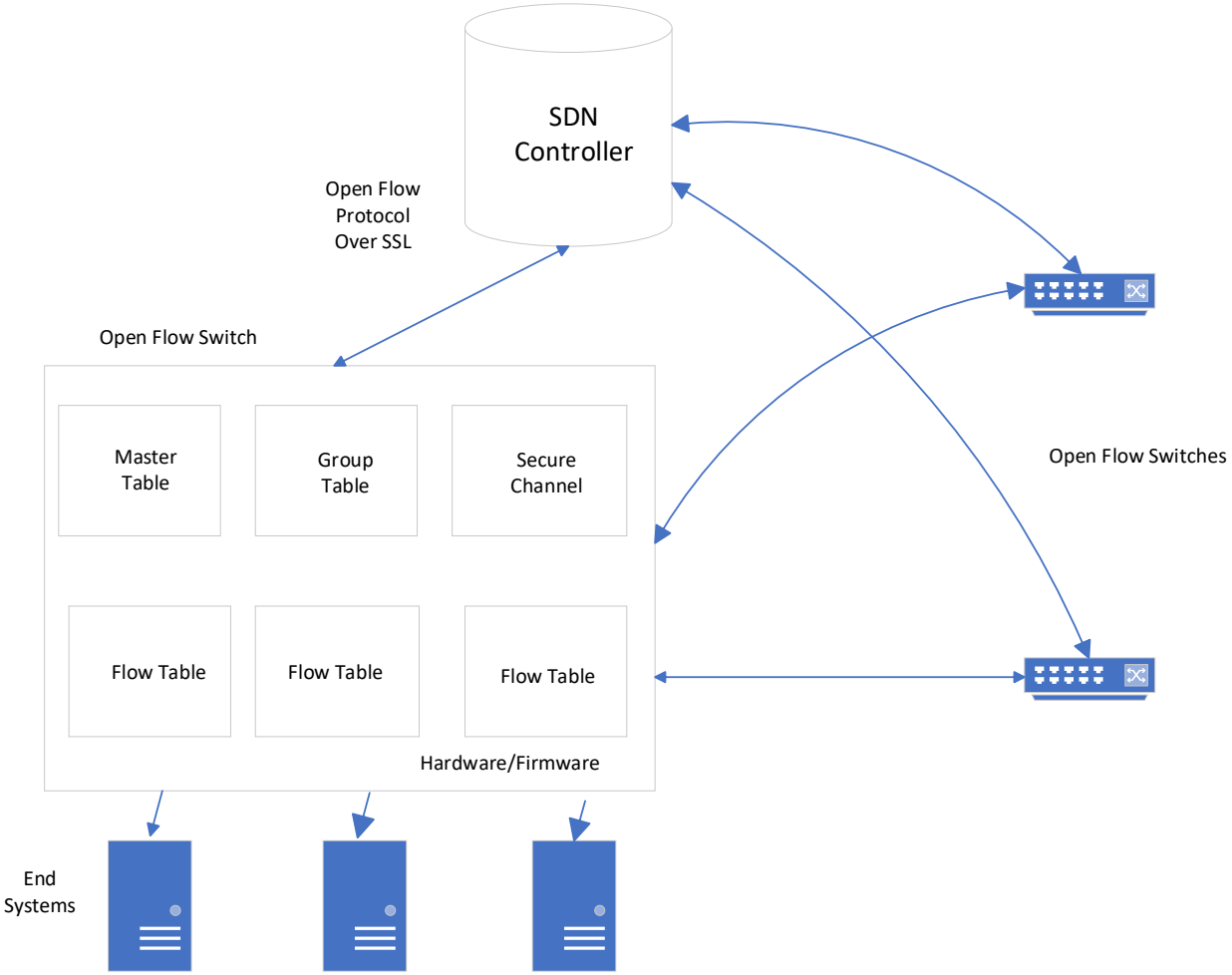

Fig. 1: SDN controller

An incoming packet is matched to a certain flow, and the function to be executed is specified in the flow table. A flow table sends a flow to a group table, which causes various actions to be triggered. Flow performance is triggered by the meter table.

The controller establishes connections with the OpenFlow switches and makes flow forwarding decisions. Network intelligence is centered on the controller of a software-defined network, which maintains a global perspective of the network. SDN increases resource utilization and efficiency of the network by resolving varied application requirements, the network's programmability, and multi-tenant capabilities. Despite all these outstanding advances, the controllers' main challenge is managing a high quantity of control messages. Huge data transfer occurs in a mature software-defined system, causing a bottleneck at the switch. The controller becomes unstable because of the large volume of control requests (Lakhani and Kothari, 2020).

Load balancing is a network operation in computer networks that distributes traffic evenly across numerous servers and connections. It aids in the optimization of network performance and efficiency by boosting throughput and reducing reaction time and latency. Load balancers are used by all web organizations to manage incoming network traffic, ensuring that all servers are evenly loaded, and congestion is avoided. The present load balancers are entirely hardware-based, resulting in a single point of failure, and making scaling difficult. Load balancers in SDN are software programs that run on top of SDN controllers to give flexibility (Ali et al., 2018).

In today's internet environment, networks must handle a tremendous volume of traffic, which is generated only by client requests. It's quite tough for a single server to handle a huge amount of traffic. As a result, we have many copies of the same server. As a front end, a network load balancer will be used. When a request comes in, it will be routed to the least loaded server based on the load strategy. The load balancer is a major flaw in traditional networks since it is hardware-based, inflexible, and vendor-specific. Because it is not programmable, network

managers are unable to build their own code. As a result, we've switched to SDN load balancers, which are configurable and don't require any special hardware.

Static or dynamic load balancing strategies are available. The static mapping b/w controllers and switches generate reliability concerns and, as a result, reduces network performance due to the not equal load distribution within controllers. Dynamic approaches are always more successful than static solutions since the load assignment procedure is based on the network traffic pattern. In SDN, packets in control messages sent by switches overburden a controller. SDN load balancing considers several aspects, such as throughput, utilization of resources, energy usage, time of execution, and peak load ratio (Sufiev et al., 2019).

Authors have utilized both approaches to solving load balancing concerns in the literature. However, research on switch migration-based load balancing has dominated in recent years since it allows for dynamic load control and flexible deployment (Xue et al., 2019). A novel switch migration approach is described and developed in this work, along with an elaborated process for the load balancing of the control plane.

The rest of the paper is organized as follows: Section 2 describes the Background, section 3 describes the system architecture, section 4 describes the results and discussion, and section 5 describes the conclusion.

2. Background

This part examines recent load-balancing and distributed controller research accomplishments which supports the depiction of our research background and theoretical foundation.

To balance loads of all controllers, a super controller is used in a centralized approach to network load balancing, such as the system designs proposed in ElastiCon (Dixit et al., 2013). ElastiCon introduced the architecture of a flexible distributed controller in which the controller pool is dynamically enlarged or reduced in response to changing traffic conditions. ElastiCon automatically balances the load among controllers to resolve load imbalances caused by geographical and temporal oscillations in traffic conditions, ensuring consistent performance regardless of traffic dynamics.

The improved Switch Migration Decision Algorithm (ISMDA) framework was also suggested in Adekoya et al. (2020). When the incoming load is elephant flow, this framework solved the network challenge. During the controller load imbalance phase, the switch migration framework's balancing module, which operates on each controller, is started. To determine the set of underloaded controllers in the network, the enhanced framework evaluated controller variance and controller average load status. The created efficient migration model was utilized to determine both the migration cost and load-balancing variation for the best controller selection among the underloaded controllers.

The Controller Adaption and Migration Decision (CAMD) framework is a switch migration-based load balancing technique that efficiently picks both switch and target controller while minimizing reaction time (Sahoo and Sahoo, 2019). This effective switch migration method offers better load balancing, faster response times, and higher end-user service quality. CAMD chooses appropriate switches to migrate from the present controller to a lightly loaded controller, lowering migration costs, and increasing controller resource usage.

Cui et al. (2019) explained a load balancing technique using a Software-Defined Wireless Sensor Network (SDWSN). This study created a mathematical model that employs multi-path routing methods to reduce the usage of a maximum link of wireless sensor networks. The recommended technique uses the benefits of the global perspective of the WSN provided by SDN to implement the best and most flexible traffic scheduling. The concentrate was on how to manage wireless sensor networks in a smart city intelligently.

Because traffic is unpredictable, networks such as data centers are dynamic, and traffic congestion can occur at any time. Chiang et al. (2021) offered an SDN-based server cluster with a dynamic load balancing scheme that can be applied to a variety of data center network topologies. During flow transmissions, the recommended approach enables path changes. This ensures that traffic load balancing works across many communication pathways in SDN data center networks, preventing congestion.

3. System architecture

In the previous research work developed, the AC* algorithm (Shylaja et al., 2021) is an SDN routing system based on TPDL. Its main goal is to reduce topology discovery overheads by replacing the traditional shortest path approach with a more robust route selection scheme. One of the basic concepts in AC* is the usage of TPDL as prior knowledge. By leveraging the knowledge in TPDL, the controller will offer a simple environment for ensuring that the entire network works. With the addition of additional components, AC* can accommodate topological changes and failures. The controller has been critical in balancing the network's overall load. It does this by keeping track of the states of switches and connections (Chakravarthy and Amutha, 2019).

As the network's central authority, the controller has made global routing decisions. It's responsible for the network's overall management. The controller has made its routing decision after receiving the first inbound packet. All other packets in the flow are subject to this rule. Every flow table of switch has the relevant rule changed. The demand on the network to make routing decisions is increasing as the number of packets grows.

This method is based on a centralized design that ensures centralized networking system control and monitoring. To acquire connection state information, this technique requires NIB to be updated on a regular basis.

The packet transmission path for the SDN is made up of many switches and links. A single switch and numerous controllers can be connected, and vice versa. By transferring multiple messages, a centralized controller is responsible for balancing the network's total load (Hamdan et al., 2021). There are two elements to it: (a) With the help of NIB, load measurement is utilized to gather the load of the network's entities. It oversees deciding whether a highly overloaded entity should be shut down by comparing its overall load to a threshold value (Mokhtar et al., 2021). Fig. 2 shows the system architecture.

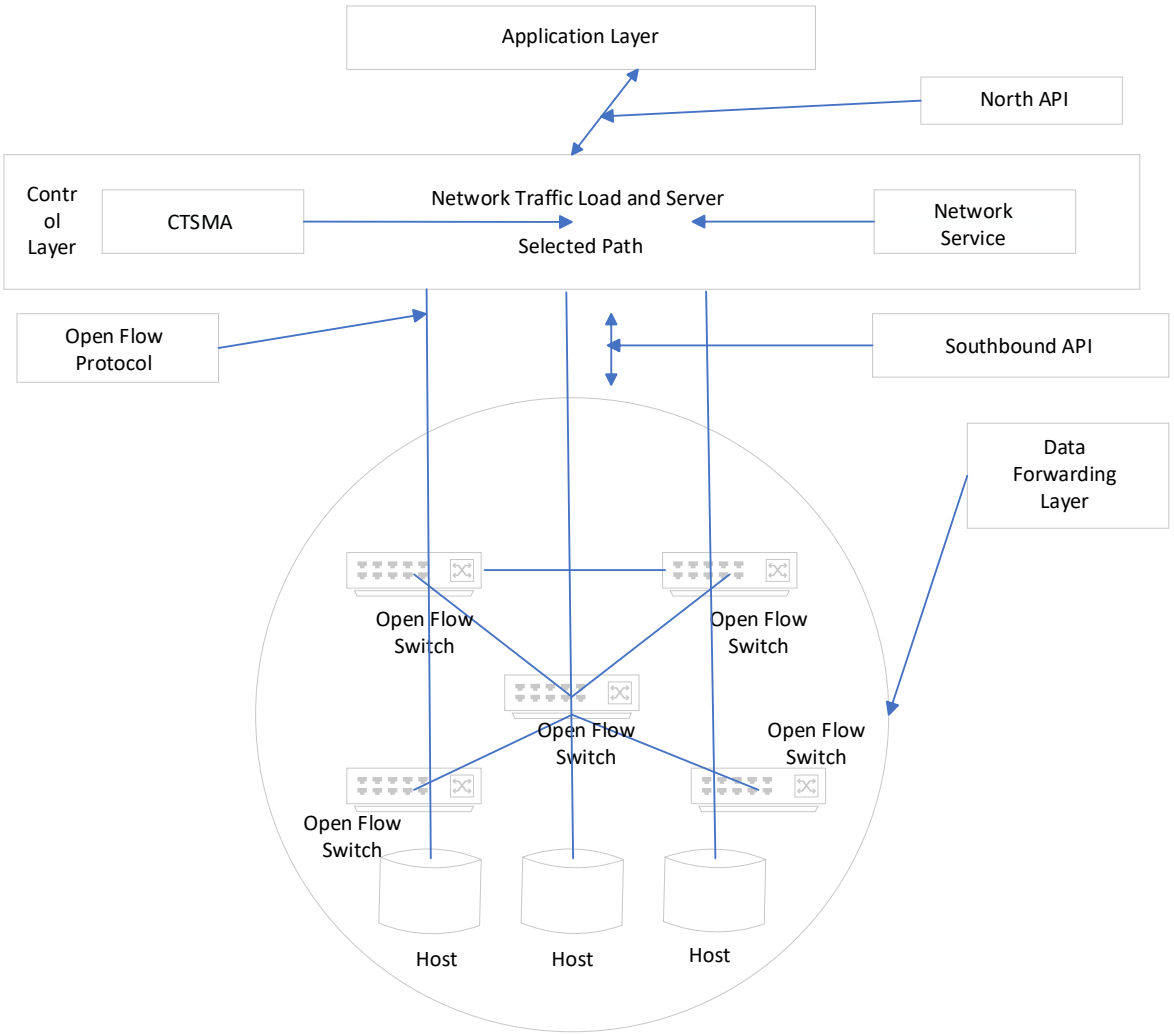

Fig. 2: System architecture

By moving the weight of this overloaded controller to other switches in the network via alternate path routing, the decision-maker has balanced the load of this overloaded controller. The state information for all network entities is updated on a regular basis by the NIB.

SDN design uses a finite number of controllers, C=c1, c2, ..., cm, |C|=m. Each controller is capable of processing1, 2, ..., n packets per second. The forwarding plane is made up of a finite number of switches, S=s1, s2, ..., sm, |S|=m. Each controller in such an architecture oversees numerous switches in its domain. Table 1 shows important notations.

We define a boolean variable x_i^k 0, 1 in a master/slave SDN model, with x_i^k=1 if ck refers to the master controller of the switch si. Otherwise, x_i^k equals 0.

Table 1: Important notations

No.	Notation	Description
1	W_{cm}	The switches set handled by the mth controller
2	t_m	mth controller
3	w_i	ith switch
4	I_i^m	Switch $w_i$time t load
5	I_t^m	time t load for Controller t_m
6	I^*	The controller average load o
7	ρ_t^m	mth controller Response time of at time t

The centralized controller gathered the load of each controller in the network during the Load measurement step to identify the overloaded one. The num of messages sent from the switches per unit time is used to calculate the controller's load. Hello messages, packet in messages, and Echo messages are examples of message types (Sahoo et al., 2020). The message's packet takes up most of the space.

The load statistics from this step include the message arrival rate from the switch (Ar), the total number of flow table entries (Nf), and the round-trip time (RTT) from the switch to the controller (Rtt). The load factor for each controller is determined by the switch load.

The switch must be carefully managed in order for the load factor of an overloaded controller to be perfectly balanced. The entries in the flow table, message arrival rate, and round trip duration are the three characteristics used to select the switch. Because of the enormous flow table, the controller is under a lot of strain. If the message arrival rate is high, the controller is most likely overwhelmed by the switch load. C_l is used to calculate controller load, as shown in the following equation.

$$C_l = w_1 \times N_f + w_2 \times A_r + w_3 \times R_{tt}$$

The weight coefficients w1, w2, and w3 added together equal 1.0. The NIB database, which is administered by the centralized controller, collects and updates all of this data about the switch load factor. The first phase's output supplied information on the overloaded controller. To balance the system, the load must be balanced efficiently.

$$I_t^m = \sum_{i=1}^{k} I_t^i \text{ X } x_i^m \quad (1)$$

Minimizing the load disparity between controllers ensures load balancing. As a result, the following optimization issue has been formulated:

$$I_t^m = min\left\{\frac{1}{n}\sum_{m=1}^{n}|I_t^m - I^*|\right\} \quad (2)$$

$$I^* = \frac{1}{n}\sum_{m=1}^{n} I_t^m \quad (2a)$$

$$\sum_{m=1}^{b} x_a^m = 1, \forall 1 \le a \le b \quad (2b)$$

$$x_a^m \in \{0,1\}, \forall 1 \le a \le b, and, \forall 1 \le m \le n \quad (2c)$$

Each switch must be connected to a single master controller, according to constraint (2b). The binary nature of the variable x_a^m is guaranteed by constraint (2c). To maintain optimum network performance, each controller must have almost the same load. However, if traffic flow variance is significant, a controller load can be uneven in comparison to the other. Some switches must be relocated if the load is unbalanced. However, waiting until the controller load reaches its maximum capacity before migrating switches can result in a loss in system performance and traffic flow processing capability. A primitive switch migration is required to avoid these problems before an overload develops. Unnecessary switch migration can, of course, degrade system performance, necessitating the use of a good load prediction model.

Algorithm 1: Switch migration scheduling algorithm

Inputs:
$\forall ck \in C, \forall si \in S, \forall \alpha k \in \alpha$
Outputs:
Migration Schedule
Initialize:
CM: *Migration actions for the set of controllers*
CA: *Without migration actions for the set of controllers*
Tol: *overload threshold for the controller*
$Listmigration = null$
$for\ all\ ck \in CA$
{
$Pol = overload\ prediction\ (Ck, T)$;
$[Ck --> controller, T --> Threshold]$
$if\ Pol = -1$
{
$flag11 = 0$
$for\ all\ si \in Sck$
{
$if\ Lkt + Pol - lit + Pol \le Tol$
$and\ flag1 = 0$
{
add si to Listmigration.
$flag1 = 1$
}
}
$if\ flag1 == 0$
{
choose the switch with the smallest load and
add it to migrateList.
}
}
}
$while\ Listmigration = null$
{
select the switch si with the smallest Pol.
sort the CA controllers in an ascending
according to their load Lkt.
$for\ all\ ck \in CA$
{
$if\ lit + Lkt < \alpha k$
{
$add\ (si \longrightarrow ck)$ *into MigrationSchedule.*
move ck to CM.
delete si from Listmigration.
}
}
}

Algorithm 2: Prediction overload

Inputs:
$ck \in CA$.
T: *overload threshold for the controller.*
Outputs:
Pol: *the controller ck predicted overload step.*
Initialize: $W_{ol}^{ck} = -1$
The switches set handled by controller ck is denoted by Sck
The generated load by switch si at time t is denoted by lit.
Predicted generated load at the with step is denoted by lit $+ w$
$for\ all\ s_i \in S_{ck}$
{
$for\ w = 1, w \le W, w++$
{
$$I_t^m = \sum_{i=1}^{k} I_t^i * x_i^m$$
}
}

$for\ w = 1, w \le W, w++$
{
$$I_{t+w}^m = \sum_{i \in W_{cm}} I_{t+w}^i$$
$if\ Lkt + w \ge T$

{
$W_{ol}^{ck} = w$
break
}
}
return W_{ol}^{ck}

For migration purposes, the switch selection module takes the highest load switch from the overloaded controller. The load judgment module provided useful statistics such as roundtrip duration, switch flow table entries, and rate of packet arrival. For controller selection and threshold estimations, the average arrival rate of messages is used, although roundtrip time and flow table entries are used for the selection of controller,

$$Sl_{migrate} \leq \frac{Cl_{overloaded} - Cl_{target}}{2} \quad (3)$$

where, Cl_{target} indicates a load of the appropriate controller, $Cl_{overloaded}$indicates overloaded controller load and which is the target controller. $Sl_{migrate}$ indicates the overloaded controller load for the migrated switch. A switch should be able to connect to more controllers concurrently in this work. A master controller, on other hand, is completely operational and responds to the switch's requests at any one moment, while others can be in slave or equal mode. A messaging library protocol, such as ZeroMQ (ZMQ) or Zookeeper, is used to alter the role, and HATCP is utilized to ensure the delivery of messages between switch and controller.

4. Results and discussion

In this section, the study's experimentation is detailed. For the performance evaluation, this research work considers a simulation approach and assessed throughput, packet loss, number of migrations, and response time.

The experiment is conducted on a computer with an Intel Core i7-6800HQ processor running at 3.0GHz, 32GB of 1800MHz memory, Windows 11 operating system. The Simulation is carried out using the Jupyter notebook compiler which is an open-source (FOSS) software that permits the users to write code. In the algorithm, the controller threshold was supposed to be set at 3000p/s. Present load on controllers A1-D1 is set to 600p/s, processing rate to 72%, and a total sum of load incoming generated starts at (10000-27000) p/s with a size of 30 before any migration phase.

For each iteration, the generated CTSMA used a predefined Controller Threshold (CT) of 3000p/s on all simulated controllers. The load on controllers A1, B1, C1, and D1 was set at 600p/s before any migration phase. The processing rate is reduced to 70% to simulate a real-world control environment. In each iteration, the experiment was repeated 1800 times with a total incoming load of (10000-27000) p/s.

The results of this experiment are compared to two other similar studies Sahoo and Sahoo (2019) and Mokhtar et al. (2021) in the literature. The throughput of a controller indicates the number of packets it can successfully process. The average controller throughput is computed using the traffic generated during the simulation procedure. Overall load incoming in this experiment is between 10000 p/s and 27000 p/s. Controller throughput for each of investigated studies, as well as CTSMA, is shown in Fig. 3. The graph demonstrates that the CTSMA's number of successfully processed flow requests is higher than the number of successfully processed flow requests by the reviewed works. The CTSMA throughput is 7.4 % higher than the CAMD and around 1.1% higher than the DALB.

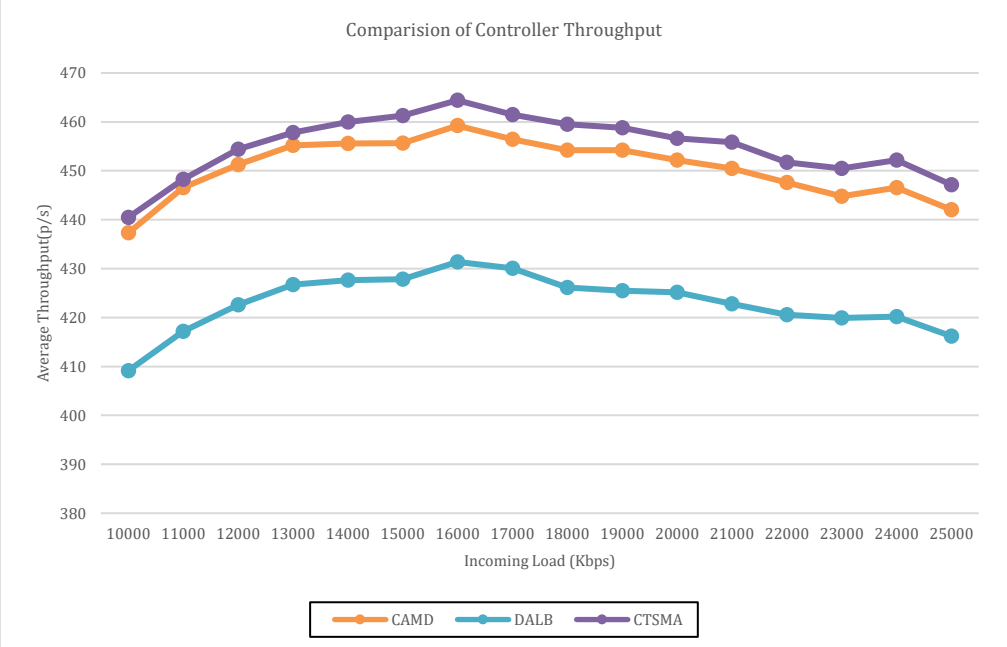

Fig. 3: Comparison of controller throughput

When there is an imbalance in the controller load, reaction time in a network is likely to increase. In this study, the response time was calculated using Eq. 4 developed by the Net forecast.

In this equation, all variables are fixed throughout the estimate because this study is primarily concentrated on studying the influence of loss of packet on response time. The packets which are rejected are used to evaluate packet-loss values.

The formula that has been employed is,

$$TR = Rd + Rt \quad (4)$$

where,

$$R_{delay} = 2\left[D + C_p + C_{tcp}\right] + \left[D + \frac{[C_p + C_{server}]}{2}\right]\frac{AT-2}{mf} + Dl_n r\left[\frac{AT-2}{mf} + 1\right] + GT\left(\frac{L1}{1-L1}\right) \quad (5)$$

and,

$$R_t = \frac{MAX\left[8p\frac{1+OHR}{BW} * D\frac{PL}{WS}\right]}{1-\sqrt{L}} \quad (6)$$

In Eq. 4, TR represents response time, Rdelay represents delay time of propagation, and Rt represents the delay time of transmission.

In Eq. 5, D represents the delay in the round trip; Cp represents present processing time; Ctcp represents processing of server TCP; Cserver represents processing time of server; AT represents application turns; mf represents multiplex factor; Dlnr represents the ratio of packet loss and G represents TCP timeout. Similarly, in Eq. 6, PL represents the length of payload; OHR represents the ratio of overhead; BW represents minimum path bandwidth and WS represents effective window size. CTSMA performs better than Controller Adaption and Migration Decision(CAMD) and DALB with about 5.8%.

To compare the 3 methods in the simulation analysis, this research employed average response time. The average response time of the 3 methods increases as the incoming load grows. The suggested CTSMA outperforms the Controller Adaption and Migration Decision(CAMD) and DALB in terms of response time, with roughly 5.7% and 1% less, respectively, as shown in Fig. 4.

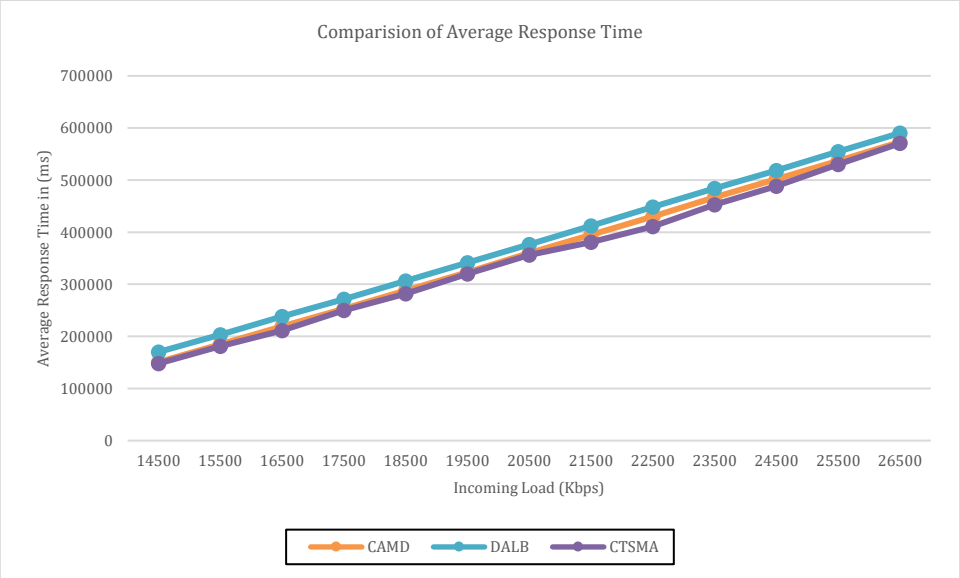

Fig. 4: Comparison of average response time

Convergence Time: The system consists of one or more routers located in a cloud environment and when the link failure takes place within a system, the break is noticed, and traffic is re-routed.

Time taken to reroute the packet is the measure of convergence time. Convergence time is calculated using the following Eq. 7:

$$Conv_{time} = N_{rp} * h_c * \alpha \quad (7)$$

where, N_{rp} represents the number of routing packets. h_c represents the hop counts from router to target and α represents the ratio of the number of sent packets and the number of received packets.

In the simulation analysis CAMD, CTSMA, and DALB methods are considered for research employed convergence time.

The convergence time for CTSMA decreases compared to existing Controller Adaption and Migration Decision (CAMD) and DALB as shown in Fig. 5.

5. Conclusion

This research presented CTSMA as a solution to the problem of SDN load imbalance caused by changes in network scale dynamically. This work developed a convergence time aware switch migration approach for balancing multi-controller loads and seeks to optimize the convergence time and switch migration process by introducing migration efficiency. The simulation results show that CTSMA achieves low controller response time

and high controller throughput when the network scale changes.

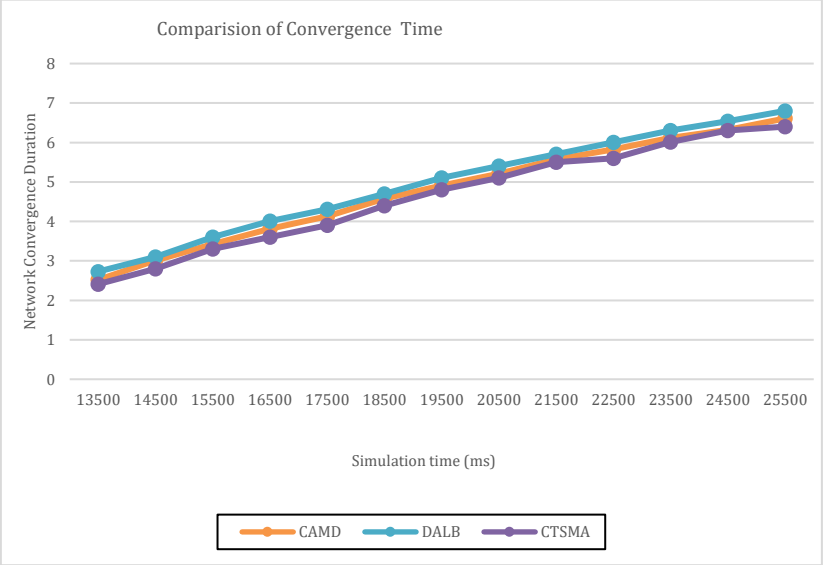

Fig. 5: Comparison of convergence time

Acknowledgment

The authors would like to thank Dr. Ambedkar Institute of Technology, Bangalore, and Visvesvaraya Technological University (VTU), Belagavi, Karnataka.

Compliance with ethical standards

Conflict of interest

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

Adekoya O, Aneiba A, and Patwary M (2020). An improved switch migration decision algorithm for SDN load balancing. IEEE Open Journal of the Communications Society, 1: 1602-1613. https://doi.org/10.1109/OJCOMS.2020.3028971

Ali TE, Morad AH, and Abdala MA (2018). Load balance in data center SDN networks. International Journal of Electrical and Computer Engineering, 8(5): 3086-3092. https://doi.org/10.11591/ijece.v8i5.pp3084-3091

Canini M, Salem I, Schiff L, Schiller EM, and Schmid S (2022). Renaissance: A self-stabilizing distributed SDN control plane using in-band communications. Journal of Computer and System Sciences, 127: 91-121. https://doi.org/10.1016/j.jcss.2022.02.001

Chakravarthy VD and Amutha B (2019). Path based load balancing for data center networks using SDN. International Journal of Electrical and Computer Engineering (IJECE), 9(4): 3279-3285. https://doi.org/10.11591/ijece.v9i4.pp3279-3285

Chiang ML, Cheng HS, Liu HY, and Chiang CY (2021). SDN-based server clusters with dynamic load balancing and performance improvement. Cluster Computing, 24(1): 537-558. https://doi.org/10.1007/s10586-020-03135-w

Cui X, Huang X, Ma Y, and Meng Q (2019). A load balancing routing mechanism based on SDWSN in smart city. Electronics, 8(3): 273. https://doi.org/10.3390/electronics8030273

Dixit A, Hao F, Mukherjee S, Lakshman TV, and Kompella R (2013). Towards an elastic distributed SDN controller. ACM SIGCOMM Computer Communication Review, 43(4): 7-12. https://doi.org/10.1145/2534169.2491193

Hamdan M, Hassan E, Abdelaziz A, Elhigazi A, Mohammed B, Khan S, and Marsono MN (2021). A comprehensive survey of load balancing techniques in software-defined network. Journal of Network and Computer Applications, 174: 102856. https://doi.org/10.1016/j.jnca.2020.102856

Hu T, Guo Z, Yi P, Baker T, and Lan J (2018). Multi-controller based software-defined networking: A survey. IEEE Access, 6: 15980-15996. https://doi.org/10.1109/ACCESS.2018.2814738

Lakhani G and Kothari A (2020). Fault administration by load balancing in distributed SDN controller: A review. Wireless Personal Communications, 114(4): 3507-3539. https://doi.org/10.1007/s11277-020-07545-2

Mokhtar H, Di X, Zhou Y, Hassan A, Ma Z, and Musa S (2021). Multiple-level threshold load balancing in distributed SDN controllers. Computer Networks, 198: 108369. https://doi.org/10.1016/j.comnet.2021.108369

Sahoo KS and Sahoo B (2019). CAMD: A switch migration based load balancing framework for software defined networks. IET Networks, 8(4): 264-271. https://doi.org/10.1049/iet-net.2018.5166

Sahoo KS, Tiwary M, Sahoo B, Mishra BK, RamaSubbaReddy S, and Luhach AK (2020). RTSM: Response time optimisation during switch migration in software-defined wide area network. IET Wireless Sensor Systems, 10(3): 105-111. https://doi.org/10.1049/iet-wss.2019.0125

Shylaja BS, Deepu SR, and Bhaskar R (2021). Topology dependent ant colony based routing scheme for software defined networking in cloud. In: Nayak J, Behera H, Naik B, Vimal S, and Pelusi D (Eds.), Computational intelligence in data mining. Smart Innovation, Systems and Technologies, 281. Springer, Singapore. https://doi.org/10.1007/978-981-16-9447-9_26

Sufiev H, Haddad Y, Barenboim L, and Soler J (2019). Dynamic SDN controller load balancing. Future Internet, 11(3): 75. https://doi.org/10.3390/fi11030075

Xue H, Kim KT, and Youn HY (2019). Dynamic load balancing of software-defined networking based on genetic-ant colony optimization. Sensors, 19(2): 311-322. https://doi.org/10.3390/s19020311 **PMid:30646575 PMCid:PMC6358931**

Yan Q, Yu FR, Gong Q, and Li J (2015). Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some

research issues, and challenges. IEEE Communications Surveys & Tutorials, 18(1): 602-622. https://doi.org/10.1109/COMST.2015.2487361