Dhana: An Economic-Oriented Approach for Traffic Management using Software-Defined Networking

Matheus Saueressig, Muriel F. Franco, Eder J. Scheid, João Davi M. Nunes,
Jéferson C. Nobre, Lisandro Z. Granville

Institute of Informatics (INF), Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
{msaueressig, mffranco, ejscheid, jdmnunes, jcnobre, granville}@inf.ufrgs.br

2025 IEEE Symposium on Computers and Communications (ISCC) | 979-8-3315-2420-3/25/$31.00 ©2025 IEEE | DOI: 10.1109/ISCC65549.2025.11325808

***Abstract*—Software-Defined Networking (SDN) offers a flexible, programmable approach to network management by decoupling the control and data planes. While SDN technical advantages, such as improved network performance and security, are well-documented, its economic implications remain underexplored, particularly in prioritizing services based on business value. This paper introduces Dhana, a novel SDN-based traffic management approach that integrates economic considerations. Dhana dynamically prioritizes high-value services by analyzing network components using metrics like downtime costs and service-level agreement (SLA) compliance. The goal is to minimize economic loss during network congestion or failures, even if technical global optimization is partially sacrificed. Tests show that Dhana can take many paths according to its needs and has not significant overhead.**

***Index Terms*—Routing, SDN, Traffic Engineering**

I. Introduction

Software-Defined Networking (SDN) separates the control and data planes, allowing dynamic, centralized network management with improved security, performance, and cost efficiency [1]. SDN allows real-time monitoring and adjustment of network behavior to meet the demands of environments like data centers, cloud services [2], Content Delivery Networks (CDN) [3], and server clusters [4].

Beyond technical advantages, SDN offers economic benefits by reducing Capital Expenditures (CAPEX) and Operational Expenditures (OPEX) [5]. However, limited research explores real-time economic prioritization of services, particularly in heterogeneous networks where services vary in economic value [6]. For example, in networks supporting critical services (*e.g.*, e-commerce or internal communications), prioritizing high-value services during congestion or cyberattacks can support the mitigation of potential financial losses [7]. SDN can also be used for security purposes, such as identifying malicious packets [8] and detecting flood attacks [9], preventing economical loss due to security vulnerability.

Although SDN has been used to optimize traffic management [10], [11], most of the research focuses on technical efficiency, neglecting the economic impacts of network operations [12]. This creates opportunities to incorporate economic considerations into SDN management, addressing scenarios where the financial impact outweighs technical optimization.

We introduce Dhana, a novel approach that integrates economic metrics (*e.g.*, downtime costs, Mean Time to Repair (MTTR), and SLA compliance) into the SDN traffic management. Dhana prioritizes high-value services, dynamically reallocating traffic to minimize economic losses during congestion or failures, even if it sacrifices purely technical optimization. The word Dhana comes from Pali, which means wealth.

This paper is structured as follows. Section II reviews related work, Section III details Dhana's implementation, Section IV presents experiments and results, and Section V summarizes findings and future directions.

II. Related Work

In this section, we review state-of-the-art techniques in network traffic management, particularly focusing on rerouting traffic and load balancing.

Nyx [13] is designed to mitigate DDoS attacks by rerouting traffic between a deployer Autonomous System (AS) and a critical AS. It uses BGP to minimize congestion, allowing for Routing Around Congestion (RAC) by appending the congested AS's number to the advertised path. However, [14] highlights a trade-off between establishing detour paths and maintaining path isolation, where the latter prevents non-critical flows from sharing these paths, thus reducing bandwidth for critical services. Therefore, the lack of a global network view complicates effective rerouting.

In SDN, approaches like those in [15] and [16] address congestion management. The [15] proposes an algorithm using Dijkstra's shortest path to dynamically reroute traffic in a fat-tree topology, tested in a Mininet environment, demonstrating significant performance improvements. Meanwhile, [16] focuses on cloud data centers, implementing a flow-based local rerouting scheme to detect congestion and redistribute traffic efficiently, enhancing load balancing and fault tolerance.

These works progressed towards enabling services, even in congestion situations. However, none of them evaluates if all the services are worth helping in more extreme situations. Furthermore, the question of whether it is worth helping services with lower profit by distributing loads equally and enabling traffic to flow through important points to services which have higher profit remains unanswered. Although there are many applications for improving overall network performance, there is still a need to address situations where you must choose services to be preserved and their activities maximized in order to improve the gains of a business that is dependent

on a network infrastructure. Dhana is the very first solution to bring the economic aspect to traffic management.

III. Dhana Approach

Dhana was conceived as a traffic optimizer guided by economic interests. Its purpose is to promote the availability of highly valuable services by avoiding traffic congestion and also to maintain a hierarchical relationship between the services in the network. If a service does not present enough economic contribution, it does not receive special treatment. The implementation of Dhana and examples of usage are publicly available at Github[1]. In the next subsections, we describe in detail the design, architecture, and implementation of Dhana.

A. Dhana Approach

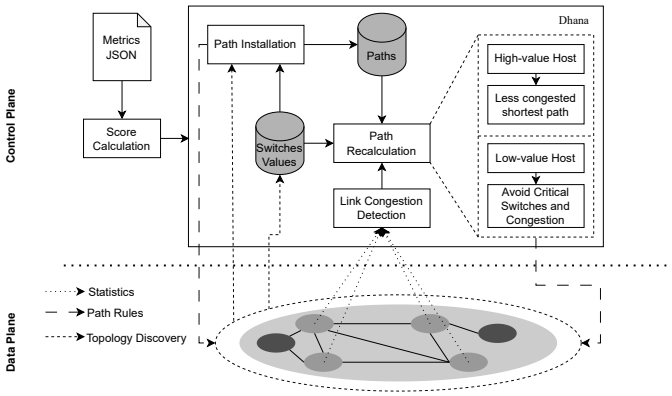


Fig. 1: The Dhana's Approach

Dhana ensures the availability of critical services in OpenFlow-based networks by dynamically managing paths through three core modules: Path Installation, Link Congestion Detection, and Path Recalculation (Figure 1).

In the Path Installation module, the controller calculates host scores from operator-defined metrics (e.g., in a `.JSON` file). High-value hosts are assigned the most efficient, least congested paths, while lower-value hosts are routed to avoid critical switches identified using betweenness centrality.

The Link Congestion Detection module monitors switch ports to identify congestion. When congestion is detected, the Path Recalculation module dynamically adjusts the paths. High-value hosts are rerouted to less congested paths, while low-value hosts avoid congested routes and critical switches. This approach optimizes network flow, prioritizing critical services and reducing disruptions.

B. Path Calculation and Selection

To achieve its goal, Dhana requires a global view of the network to select the best path between hosts based on the value of the service. This involves evaluating economic metrics such as uptime SLA percentage, downtime cost per hour, and MTTR. Metrics such as high uptime SLA enhance profitability, while others, such as high downtime costs, reduce it. These values are derived from company reports and analyses of operational costs and revenues. As shown in [17], key factors that influence service profitability can improve network planning, while [18] provides a framework for assessing Total Cost of Ownership (TCO), useful for economic analysis. Both can be used as a way to chose metrics for economical analysis.

Dhana is not limited to these methodologies; any measurable factor that affects profitability can be used as a metric. Such data can be stored in formats like `.JSON`, enabling the controller to assign scores to hosts based on these metrics.

Using these metrics, we calculate a score for the host, and this score is used to give special treatment to host services during path calculation. How to calculate the score lies in the network operator and its team's decision, given that creating a highly sophisticated methodology for score calculation is out of the scope of the work, which prioritizes the idea of how to choose paths given established economical values for the services in the network. We are currently exploring the idea of establishing this methodology for future work.

In our implementation, we define and use a weighted composite score to evaluate the services in the network since we can attribute relevance to each metric, allowing us to address each economic aspect of the service while respecting the importance of these aspects. We also implemented this type of score for its simplicity, as we chose this weighted sum for a simple and intuitive way to obtain scores for the hosts.

The general formula for calculating the weighted composite score is given by Eq. 1:

$$\text{Composite Score} = \sum_{i=1}^{n} w_i \cdot \frac{x_i - \min(X_i)}{\max(X_i) - \min(X_i)} \qquad (1)$$

Where,

n is the total number of metrics;

x_i is the value of the $_i - th$ metric for the service being evaluated;

$\min(X_i)$ is the minimum value of the $_i - th$ metric across all services;

$\max(X_i)$ is the maximum value of the $_i - th$ metric across all services;

w_i is the weight assigned to the $_i - th$ metric (based on its importance).

Each metric is normalized to a scale of 0-1 using min-max normalization to ensure comparability, since metrics such as MTTR (time) and SLA uptime (percentage) impact network economics differently. After normalization, each metric is multiplied by a weight reflecting its importance. Weights are determined by analyzing the economic impact of each metric, such as using linear regression to model profitability or the Analytical Hierarchy Process (AHP) for pairwise comparisons.

The final composite score is calculated by summing all weighted normalized metrics, representing the economic relevance of the host's service.

In our example scenario, we defined five metrics: Asset Value, Downtime Cost, SLA percentage, Max Requests per Second (RPS), and MTTR, with their respective weights,

[1] https://github.com/MSaueressig/Dhana

which were randomly selected for the sake of testing the approach's functionality:

- Asset Value: $w_1 = 0.3$
- Downtime Cost: $w_2 = 0.25$
- SLA: $w_3 = 0.2$
- Max Requests per Second: $w_4 = 0.15$
- Mean Time to Recovery: $w_5 = 0.1$

The network operator must configure the equation according to the parameters provided. One must be aware of when to sum or subtract in the formula. In our scenario, MTTR is subtracted from the sum because the higher its value, the higher its cost. A high MTTR does not mean higher profitability, but an asset value does, which is why we sum the asset value. The network operator must pay attention to the kind of impact the metric has on the profitability, whether it is negative or positive. A threshold is then chosen based on a rank expressed by Eq. 2:

$$\text{Rank} = P \times (N + 1) \tag{2}$$

where P is the desired percentile (e.g., 0.90 for the 90th percentile) and N is the number of scores, *i.e.*, 10% of the highest scores will be considered highly valuable hosts. The percentile is adjusted by trial and error, so the operator must do a tuning process to determine how much of the services must be classified as high value to optimize the network.

High-value hosts receive priority during path calculation, gaining access to the most efficient and least congested paths. Lower-value hosts are directed away from these paths by avoiding critical switches, ensuring availability of them. These switches have high centrality betweenness centrality, which measures a switch's importance by its role as a bridge in shortest paths. Hosts that achieve composite scores above the defined threshold will be granted the paths that use these switches, allowing them to utilize the most efficient and least congested paths.

The betweenness centrality formula is presented in Eq. 3:

$$C_B(v) = \sum_{\substack{s \neq v \\ t \neq v}} \frac{\sigma(s, v, t)}{\sigma(s, t)} \tag{3}$$

Where,

$\sigma(s, t)$ is the total number of shortest paths from node s to node t.

$\sigma(s, v, t)$ is the number of those shortest paths that pass through node v.

The summation is over all pairs of nodes s and t in the graph, excluding v.

When the network initializes, the controller computes host values using composite scores. OpenFlow switches establish connections with the controller, while hosts send ARP packets to populate the ARP table and map MAC to IP addresses. The controller identifies switches and hosts, calculates switch centrality using a betweenness centrality algorithm, and builds a connectivity graph. It then computes optimal paths using Depth-First Search (DFS) to find all possible paths, caching them for future use. High-value hosts are assigned the shortest paths, while lower-value hosts receive paths that avoid critical switches. Path selection for lower-value hosts follows specific prioritization steps:

1. Each path has a base cost, C_{path}, which is calculated by summing the link costs between switches along the path as it can be seen in Eq. 4:

$$C_{\text{path}} = \sum_{i=1}^{n-1} C_{\text{link}}(s_i, s_{i+1}) \tag{4}$$

where $C_{\text{link}}(s_i, s_{i+1})$ is the cost of the link between switch s_i and switch s_{i+1}.

2. For each path, the number of critical switches, k_{crit}, is counted. A critical switch is defined as a switch with a centrality value higher than a given threshold. This relationship is expressed by Eq. 5:

$$k_{\text{crit}} = \sum_{i=1}^{n} \mathbb{1}(s_i \in \text{Critical Switches}) \tag{5}$$

where $\mathbb{1}$ is the indicator function, which is 1 if s_i is a critical switch, and 0 otherwise. In our scenario, we use the average value of the switches.

Let: - $S = \{(s_1, v_1), (s_2, v_2), \ldots, (s_n, v_n)\}$ be the set of all switches s_i and their respective values v_i. - The average switch value V_{avg} is expressed by Eq. 6:

$$V_{\text{avg}} = \frac{\sum_{i=1}^{n} v_i}{n} \tag{6}$$

where n is the total number of switches and v_i represents the value associated with switch s_i.

The set of critical switches C is then defined as the set of switches whose value v_i is greater than the average value:

$$C = \{s_i \in S \mid v_i > V_{\text{avg}}\}$$

In other words, a switch s_i is considered critical if its value exceeds the average value of all switches.

3. Longer paths may be rewarded with an incentive, I_{length}, which is proportional to the length of the path, as can be seen in Eq. 7:

$$I_{\text{length}} = \lambda \cdot n \tag{7}$$

where n is the number of switches in the path and λ is a positive constant of the incentive for choosing longer paths.

4. A penalty, P_{crit}, is applied for each critical switch in the path. This penalty is proportional to the number of critical switches, and it is expressed by Eq. 8:

$$P_{\text{crit}} = \mu \cdot k_{\text{crit}} \tag{8}$$

where μ is a constant representing the penalty for each critical switch.

5. The total cost of the path, considering both incentives and penalties, is given by Eq. 9:

$$C_{\text{total}} = C_{\text{path}} - I_{\text{length}} + P_{\text{crit}} \tag{9}$$

6. The paths are then sorted by their total cost, and the path with the lowest C_{total} is selected as can be seen in Eq. 10:

$$\text{Selected Path} = \arg\min_{\text{path}} C_{\text{total}} \tag{10}$$

Following that, the controller installs OpenFlow rules on the switches, using either direct actions or group tables for

multipath routing, adjusting flow entries dynamically based on the paths selected to achieve efficient traffic distribution. The controller also handles link and switch events, updating the network graph whenever the topology changes.

C. Congestion Detection and Monitoring

To detect congested links, we implemented a monitoring loop on the controller that periodically requests port statistics from switches. When switches respond, the controller processes the replies by recording the current time, calculating the difference in transmitted and received bytes since the last event, and determining the total bytes transferred. The utilization is calculated by the Eq. 11:

$$\text{utilization} = \frac{\text{total\_bytes}}{\text{bandwidth\_limit}} \tag{11}$$

Links exceeding an 80% utilization threshold are flagged as congested. Upon detecting congestion, a handler function requests flow statistics to identify the source and destination of the flows causing congestion, as port stats only indicate the affected switch port. The flow stats handler processes each statistic to retrieve match conditions, packet/byte counts, and switch IDs. It filters for IPv4 flows to determine flow origins and destinations. Utilization is calculated for each flow to assess its impact on the switch using Eq. 12.

$$\text{utilization} = \left(\frac{\text{byte\_count}}{\text{bandwidth}}\right) \times 100 \tag{12}$$

If the utilization exceeds a threshold (*e.g.*, 80%), the function calls a path recalculation function to adjust the paths and avoid the congested link.

D. Path Recalculation and Rerouting

To recalculate the path and reroute the flow, we access the data structure that holds all paths calculated by the DFS, so we do not need to recalculate these paths again. The process of selecting the paths for high-valuable hosts and lower-valuable hosts is slightly different, involving two primary modifications: **1.** Congestion Awareness: During recalculation, all possible paths between source and destination are evaluated based on their congestion levels. For each path, the number of congested links c_{cong} is calculated, and paths are sorted by this count:

$$\text{Paths with congestion counts} = \{(p_1, c_{\text{cong},1}), (p_2, c_{\text{cong},2}), \ldots, (p_n, c_{\text{cong},n})\} \tag{13}$$

The path with the fewest congested links is selected by the Eq. 14:

$$\text{Best path} = \arg\min_{\text{path}} c_{\text{cong}} \tag{14}$$

2. Host Value-Based Selection: For high-value hosts (hosts with a value greater than a certain threshold), the path selection favors paths that have the least congestion and are shorter, as can be seen in Eq. 15:

$$\text{Best path for high-value hosts} = \arg\min_{\text{path}} (C_{\text{path}}, c_{\text{cong}}) \tag{15}$$

If two paths have the same number of congested links, the list is sorted by length. For low-value hosts, path selection takes into account critical switches and congestion. Total cost is modified to include a reward for longer paths and a penalty for critical switches, which is expressed by the Eq. 16:

$$C_{\text{total}}^{\text{recalc}} = C_{\text{path}} - I_{\text{length}} + P_{\text{crit}} + c_{\text{cong}} \tag{16}$$

where c_{cong} is the number of congested links on the path. The path minimizing this recalculated cost is chosen using Eq. 17:

$$\text{Best path for low-value hosts} = \arg\min_{\text{path}} \left(C_{\text{total}}^{\text{recalc}}\right) \tag{17}$$

Here, we prioritize the paths with least critical switches; then, if the paths have the same number of critical switches, the controller takes the shortest one. This concludes the functionality of Dhana. The next section will show an evaluation done testing the mechanisms described in this section and how Dhana performs generally.

IV. Evaluation

This section provides a quantitative evaluation to showcase the feasibility and scalability of Dhana. For that, different experiments were conducted in different topologies, and the path was computed in order to ensure critical services (from an economic perspective) have high availability even in situations of cyberattacks or high usage of the network.

To implement Dhana, we developed a Ryu controller using Python as the programming language. The controller is called by the script and initialized, waiting to connect to a network. We have built different topologies on Mininet to test Dhana's behavior. An example of topology can be seen in Figure 2. These topologies were built on MiniEdit, an extension of Mininet. The switch used by the SDN network is Open vSwitch, which functions as a virtual switch to facilitate communication between different virtual machines (VMs) or network namespaces, as well as physical network devices, and operates on various platforms, including Linux and is widely used in SDN environments.

At first glance, it seems that we chosen topologies have nothing to do with real-life topologies, but we provided them with a purpose. The scenarios and topologies were carefully chosen to be as illustrative and complex as possible. We decided to partially avoid complex scenarios to show clearly and visually how Dhana changes its paths and how it can work in different scenarios with many options available. We wanted to show that if Dhana has two congested paths, it will choose a third one, and so on. To illustrate the behavior intuitively, we decided to use a simple topology, but Dhana can also perform properly on complex topologies. We built the topologies with many paths and redundant connections because we wanted to highlight their capability of dealing with these complex situations, even if they could be more realistic. To summarize, we decided that these scenarios would fully demonstrate Dhana's functionality in scenarios where it can choose different paths according to length and congestion, showing these choices in this evaluation section. This does not mean that Dhana can't deal with realistic scenarios. In reality, being capable of working in such drastic and highly complex topologies guarantees its operability in much simpler and more realistic topologies.

A. Path Changes

The Fig. 3 from (a) to (c) shows a recalculation of paths along congestion events from a less valuable host to a highly

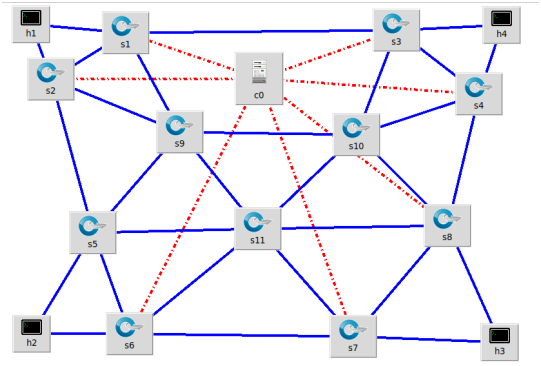
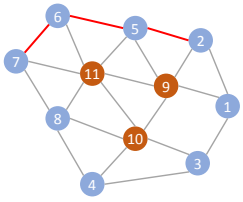
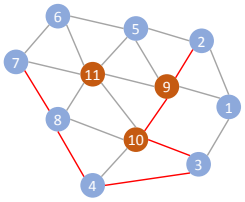


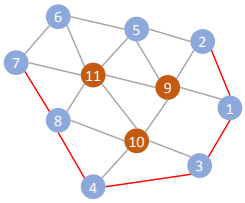
Fig. 2: Network Topology



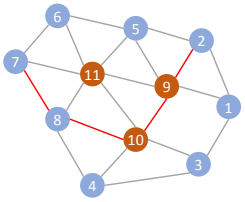
(a) Initial Path (Low-value)



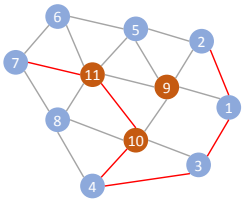
(b) Path After First Change (Low-value)



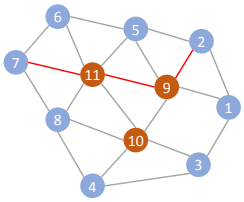
(c) Path After Second Change (Low-value)



(d) Initial Path (High-value)



(e) Path After First Change (High-value)



(f) Path After Second Change (High-value)

Fig. 3: Illustration of Path Changes for low-value hosts (left column) and high-value hosts (right column) in the Algorithm.

valuable host. The network is represented as a graph, the nodes being switches and the red line the path. Each time a path gets congested, a new path is chosen, the shortest path with the least number of congested links. The orange nodes represent the critical switches that the receiver host benefits from utilizing since most of the shortest paths use them.

These figures show the dynamic change of paths according to the needs of the network. In a scenario where a less valuable host is a destination, the paths that avoid critical switches are mainly chosen, as depicted in Fig. 3 from (d) to (f). In situations where paths with critical switches are chosen, either because the other paths are too congested or because it is a bottleneck node. But in most cases, the controller will avoid critical switches, taking turns between the available paths.

B. Scalability

In our experiments, Dhana faced scalability challenges as switches increased, but it remains adaptable for complex topologies. Real-time path recalculation using DFS, with a time complexity of O(V + E), proved impractical due to exponential growth beyond eight switches (Fig. 4), causing significant overhead during congestion events. To address this, DFS is now used only during path installation, with discovered paths hashed for quick lookups during congestion. To optimize Dhana, it is recommended to run a ping after network setup or adding switches to save all possible paths for future use.

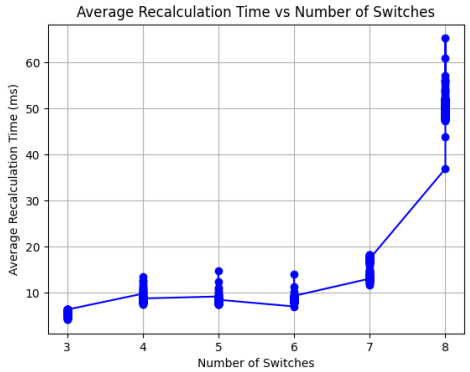


Fig. 4: Average Recalculation Time according to the number of switches

However, as more switches are installed, the path installation takes longer, and the list of possible paths grows, which might become impractical. Fortunately, SDN allows multiple controllers for a single network. A good way of scaling the solution is to use multiple controllers for network clusters of approximately 8 switches, each one of them running Dhana. The peripheral switches in the clusters connecting one cluster to another would serve as signs to a path leading to a high-valued host or low-valued host. Dhana would install and recalculate paths within the cluster to these switches, so it can navigate through the clusters and reach the hosts. Special flow rules would be installed to treat switches between clusters. This way, Dhana can function in networks of all sizes.

C. Discussion and Limitations

We conducted a UDP packet flow test using iperf [19] to simulate congestion between a high-value host (server) and a low-value host (client). Although Dhana demonstrates limitations when relying on a single controller, there is a trade-off in packet loss, bitrate, and jitter. Path recalculation introduces packet loss spikes due to the overhead of path hash access and congestion calculations. Despite slight effects on bitrate and jitter, communication performance remains largely

unaffected in practical scenarios with intermittent congestion and provides continuous availability to the services.

This test represents an extreme case of continuous congestion. In typical situations, Dhana's congestion detection effectively alleviates traffic bottlenecks but does not replace a Network Intrusion Detection System (NIDS). As a mitigation tool, Dhana aids in the management of flood attack symptoms. For full protection, it should work alongside an Intrusion Prevention System (IPS), providing real-time congestion and path change data to signal mitigation strategies.

Dhana is designed to optimize profitability, such as in CDN scenarios. Here, the downlink between edge and source servers is prioritized, as it handles large-scale content delivery critical to user experience and revenue. During congestion, Dhana prioritizes client traffic on optimal paths while redirecting less-critical uplink traffic to secondary routes.

In DDoS attacks, where malicious traffic overwhelms critical links, Dhana alleviates the impact by rerouting traffic and alerting operators to link congestion and path changes. This enables IPS systems to respond effectively while Dhana minimizes disruption to critical services.

V. Conclusion and Future Work

In this paper we have introduced Dhana, an economic-oriented approach for traffic management using SDN. This is an effort to bring economical aspects to SDN traffic management and to prove that it should be possible for network operators to minimize their economic losses and secure the functionality of services using traffic management.

Dhana demonstrates effective dynamic path recalculation in response to congestion events, especially for high-value hosts. However, scalability issues arise as the number of switches increases, leading to exponential growth in path recalculation times and potential performance overhead. By employing hash-based path storage and advocating for the use of multiple controllers within network clusters, scalability can be enhanced.

In future work, we will focus on large-scale solutions and minimizing the overhead for Dhana so we can extract its full potential. We want to explore the possibility of Dhana being part of a robust IPS, one that considers economical aspects during its execution.

Acknowledgments

This work was partially supported by the São Paulo Research Foundation (FAPESP) under grant number 2020/05152-7, the PROFISSA project, and is part of CNPq process 316662/2021-6. It is also part of the INCT of Intelligent Communications Networks and the Internet of Things (ICoNIoT), funded by CNPq (proc. 405940/2022-0) and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) Finance Code 88887.954253/2024-00.

References

[1] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 7th ed. Pearson, 2016.

[2] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: an sdn platform for cloud network services," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 120–127, 2013.

[3] H. Yang, H. Pan, and L. Ma, "A review on software defined content delivery network: A novel combination of cdn and sdn," *IEEE Access*, vol. 11, pp. 43 822–43 843, 2023.

[4] H. Zhang and X. Guo, "Sdn-based load balancing strategy for server cluster," in *2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems*, 2014, pp. 662–667.

[5] M. Karakus and A. Durresi, "Economic impact analysis of control plane architectures in software defined networking (sdn)," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[6] M. Franco, C. Omlin, O. Kamer, E. Scheid, L. Granville, and B. Stiller, "SECAdvisor: A Tool for Cybersecurity Planning using Economic Models," in *24º Brazilian Symposium on Information and Computational Systems Security (SBSeg 2024)*. Porto Alegre, RS, Brasil: SBC, 2024, pp. 554–569.

[7] M. F. Franco, L. Z. Granville, and B. Stiller, "CyberTEA: a Technical and Economic Approach for Cybersecurity Planning and Investment," in *36th IEEE/IFIP Network Operations and Management Symposium (NOMS 2023)*, Miami, USA, 2023, pp. 1–6.

[8] A. G. Avran, E. Ak, K. Duran, G. Yurdakul, and G. Seçinti, "Securing southbound interface in sdns: Utilizing support vector machines for openflow packet classification," in *2023 IEEE 28th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2023, pp. 258–263.

[9] S. A. Madoune, S. Senouci, M. A. Setitra, and J. Dingde, "Toward robust ddos detection in sdn: Leveraging feature engineering and ensemble learning," in *2024 21st International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 2024, pp. 01–07.

[10] A. A. Neghabi, N. Jafari Navimipour, M. Hosseinzadeh, and A. Rezaee, "Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature," *IEEE Access*, vol. 6, pp. 14 159–14 178, 2018.

[11] U. Jena, P. Das, and M. Kabat, "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, Part A, pp. 2332–2342, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157819309267

[12] L. Wang, "The impact of network load balancing on organizational efficiency and managerial decision-making in digital enterprises," *Academic Journal of Sociology and Management*, vol. 2, no. 4, p. 41–48, Jul. 2024. [Online]. Available: https://www.suaspress.org/ojs/index.php/AJSM/article/view/v2n4a07

[13] J. M. Smith and M. Schuchard, "Routing around congestion: Defeating ddos attacks and adverse network conditions via reactive bgp routing," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 599–617.

[14] M. Tran, M. S. Kang, H.-C. Hsiao, W.-H. Chiang, S.-P. Tung, and Y.-S. Wang, "On the feasibility of rerouting-based ddos defenses," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1169–1184.

[15] A. Srikanth, P. Varalakshmi, V. Somasundaram, and P. Ravichandiran, "Congestion control mechanism in software defined networking by traffic rerouting," in *2018 Second International Conference on Computing Methodologies and Communication (ICCMC)*, 2018, pp. 55–58.

[16] R. Kanagevlu and K. M. Mi Aung, "Sdn controlled local re-routing to reduce congestion in cloud data center," in *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, 2015, pp. 80–88.

[17] J. Xiao and R. Boutaba, "Assessing network service profitability: Modeling from market science perspective," *IEEE/ACM Transactions on Networking*, vol. 15, no. 6, pp. 1307–1320, 2007.

[18] C. Mas Machuca, "Expenditures study for network operators," in *2006 International Conference on Transparent Optical Networks*, vol. 1, 2006, pp. 18–22.

[19] C.-H. Hsu and U. Kremer, "IPERF: A framework for automatic construction of performance prediction models," in *Workshop on Profile and Feedback-Directed Compilation (PFDC), Paris, France*. Citeseer, 1998.