Journal of Network and Systems Management (2024) 32:35
https://doi.org/10.1007/s10922-024-09815-x

DDT: A Reinforcement Learning Approach to Dynamic Flow Timeout Assignment in Software Defined Networks

Nathan Harris Jr.[1] · **Sajad Khorsandroo**[1]

Received: 25 July 2023 / Revised: 17 February 2024 / Accepted: 23 February 2024 /
Published online: 21 March 2024

Abstract

OpenFlow-compliant commodity switches face challenges in efficiently managing flow rules due to the limited capacity of expensive high-speed memories used to store them. The accumulation of inactive flows can disrupt ongoing communication, necessitating an optimized approach to flow rule timeouts. This paper proposes Delayed Dynamic Timeout (DDT), a Reinforcement Learning-based approach to dynamically adjust flow rule timeouts and enhance the utilization of a switch's flow table(s) for improved efficiency. Despite the dynamic nature of network traffic, our DDT algorithm leverages advancements in Reinforcement Learning algorithms to adapt and achieve flow-specific optimization objectives. The evaluation results demonstrate that DDT outperforms static timeout values in terms of both flow rule match rate and flow rule activity. By continuously adapting to changing network conditions, DDT showcases the potential of Reinforcement Learning algorithms to effectively optimize flow rule management. This research contributes to the advancement of flow rule optimization techniques and highlights the feasibility of applying Reinforcement Learning in the context of SDN.

Keywords Software defined networking · SDN · OpenFlow · Timeout · Machine learning · Reinforcement learning

Nathan Harris Jr. and Sajad Khorsandroo have contributed equally to this work.

✉ Nathan Harris Jr.
ncharris1@aggies.ncat.edu

Sajad Khorsandroo
skhorsandroo@ncat.edu

1 Department of Computer Science, College of Engineering, North Carolina A & T State University, 1601 E Market St, Greensboro, NC 27411, USA

1 Introduction

OpenFlow [1] is a prevalent protocol that emerged in the realm of Software-Defined Networking (SDN). It was developed through a collaboration between Stanford University and the University of California at Berkeley and subsequently standardized for SDN by the Open Networking Foundation (ONF) [2]. The SDN reference architecture comprises three key components: a controller, which ensures the enforcement of policies pertaining to network flows, flow tables located on the data plane, also referred to as switch(es), and the OpenFlow protocol responsible for facilitating communication between the controller and the switch.

Commodity switches that comply with the OpenFlow protocol typically utilize Ternary Content Addressable Memories (TCAMs) to store flow rules. These are high-speed memories capable of performing a complete content lookup in a single CPU clock cycle. As a result, they offer significantly faster performance, around 400 times faster, compared to ordinary RAM-based storage [3]. However, this enhanced speed comes at the cost of consuming approximately 100 times more energy and being considerably more expensive than regular RAM-based memories. Due to the cost and capacity limitations of TCAMs, flow tables in SDN switches often support a restricted number of flow entries. This limitation can lead to a problematic scenario known as flow table overflow [4, 5]. When inactive flows accumulate in the switch's flow tables, the tables become bloated and eventually reach their capacity. At this point, an active flow may be evicted to make space for a new entry, resulting in the disruption of ongoing communication [6]. Consequently, this can cause a delay in packet processing due to the premature eviction of the flow's rule which can adversely affect the switch's performance. Flow table overflow not only impacts the switch's ability to efficiently handle network traffic but also burdens the controller with an increased workload, leading to a degradation in overall system performance.

To effectively handle flow table content and mitigate potential issues of table bloat or overflow in OpenFlow-compatible switches, two approaches can be considered: proactive and reactive. The proactive approach aims to prevent flow table overflow by carefully managing and allocating resources. It involves setting predefined thresholds and monitoring the flow table utilization in real time. When the utilization approaches a certain threshold, proactive measures such as flow table expansion or flow entry consolidation can be taken to ensure sufficient capacity for incoming flows. By addressing the issue before it occurs, the proactive approach helps maintain smooth network operation and minimize disruptions. Proactive strategies help prevent flow table overflow and maintain consistent performance but may require additional resources and continuous monitoring [7]. On the other hand, the reactive approach responds to flow table overflow after it has already happened. When the flow table reaches its capacity and an active flow needs to be evicted, the reactive approach focuses on making quick decisions to prioritize important flows and mitigate the impact on network performance. This can involve intelligent eviction policies, such as removing least-recently-used or low-priority flows, to free up space for new entries. The reactive approach aims to handle flow table overflow situations in real time, minimizing the disruption caused by evictions. Achieving such a solution becomes increasingly difficult in highly dynamic flow-based environments [8]. Moreover, this approach lacks scalability, especially when multiple controllers are involved, as it exponentially increases complexity. As the network's scale grows, obtaining an accurate global network view becomes more challenging due to the varying polling periods of each controller and the synchronization periods among them [9]. Both solutions involve the controller utilizing per-flow timeouts to manage the lifespan of each flow rule [10]. With this approach, the controller assigns a timeout value to each flow entry in the flow table. When the specified time elapses, the flow entry is automatically removed from the flow table. This method has been investigated by researchers and has shown promise as a viable solution [11, 12].

The assignment of flow timeouts can be done either statically or dynamically. The static approach involves the use of a fixed timeout value for all flows in the network or system. This means that regardless of the actual duration or activity of a flow, it will be terminated after a predefined period of time. By using a single, fixed timeout value, configuration and management become simpler. Additionally, all flows are treated equally, ensuring consistent handling across the network. However, this approach can be inefficient and inflexible. Certain flows may require longer timeouts than others, leading to premature termination and potential disruptions. Furthermore, static timeouts may not effectively adapt to dynamic network conditions or varying flow requirements. On the other hand, the alternative approach to static timeout assignment is dynamic flow timeout assignment. This method involves assigning different timeout values to flows based on their specific characteristics, requirements, or the state of the network. By adopting this approach, more precise control over flow duration can be achieved, enabling adaptability to the evolving conditions of the network. The dynamic flow timeout assignment approach offers flexibility, efficiency, and effectiveness. Each flow is assigned an appropriate timeout value, allowing longer flows to complete while terminating shorter flows earlier. The dynamic timeouts can respond to changing network conditions by adjusting timeout values using close to real-time information. By tailoring timeouts to individual flows, network resources can be utilized more effectively.

1.1 A Promising Solution via Reinforcement Learning

Reinforcement learning (RL) is a powerful approach that can be effectively utilized in dynamic flow timeout assignment. By leveraging its ability to learn from data and experience, reinforcement learning algorithms can adapt to changing network conditions and flow requirements. This adaptive nature allows the system to continually update its timeout assignment policies based on real-time feedback, ensuring that flows receive appropriate timeout values. Moreover, reinforcement learning enables the optimization of specific objectives, such as minimizing disruptions, reducing latency, or maximizing throughput, by formulating these objectives as rewards or penalties. With fine-grained control over flow timeout assignment, the algorithm can tailor timeout values to individual flows, optimizing resource utilization and meeting flow-specific requirements. By handling the complexity of decision-making and considering various factors, reinforcement learning provides an intelligent and dynamic approach to flow timeout management, leading to improved network performance. Accordingly, this paper aims to design and implement an RL model to actively modify the idle timeout value of flow entries in an SDN topology. The model uses network statistics from previous pollings to identify traffic patterns. Based on these patterns and the results of the previous choice, the model adjusts the idle timeout value of the incoming flow entries autonomously.

1.2 Contributions

This paper presents the following contributions. First, the paper introduces Delayed Dynamic Timeout (DDT), an RL-based framework for dynamic flow timeout assignment in Openflow-compatible data planes. DDT utilizes a cache module and a *Q*-learning algorithm to adjust the timeout value of flow rules in response to current network characteristics. Second, a custom environment is designed for the application of the RL solution to SDN. This environment incorporates relevant network characteristics as input parameters and evaluation metrics. It avoids imposing limitations such as flow rule entry caps or thresholds on a switch's flow tables. Notably, the solution is not simulated using general-purpose environments such as OpenAI GYM [13], as they are unsuitable for modeling highly dynamic and volatile programmable network infrastructures. Simulating in such environments can potentially lead to inaccurate or unreliable results. Third, the evaluation results demonstrate that the proposed solution based on DDT achieves both data plane effectiveness (flow rule entries match rate) and efficiency (flow table entries activity rate). This outperforms the common practice of using static timeout values in SDN topologies. These findings provide network providers with a competitive advantage for optimal network management in environments where the network traffic pattern changes frequently, even on a sub-hourly basis.

The remainder of this paper is organized as follows. Section 2 provides a literature review, examining the utilization of machine learning in SDN with a focus on resource management and optimal timeout. Section 3 outlines our framework's methodology, including its limitations, data collection process, system design, and evaluation methodology. The implementation details of the framework are discussed in Sect. 4. The evaluation results are presented in Sect. 5 and comprehensively analyzed. Lastly, Sect. 6 offers concluding remarks and proposes potential future research directions.

2 Background and Related Work

OpenFlow defines two types of timeouts: *idle-timeout* and *hard-timeout*. The *idle-timeout* causes a flow entry to be evicted after a specific period of inactivity. On the other hand, the *hard-timeout* causes a flow entry to be removed from a flow table after a designated time, irrespective of its activity or inactivity during that period. If a timeout value is too short, then flows may be evicted prematurely, as all the packets matching the flow may not have arrived before eviction. This can result in excessive flow table misses, leading to redundancy and an increased workload for the controller, thereby impacting performance [10]. On the other hand, if a timeout value is too long, premature eviction will be reduced. However, flow rules may occupy the flow table longer than necessary, causing the aforementioned flow table bloat and flow table overflow.

Flow table optimization plays a crucial role in enhancing the performance and security of SDN by efficiently utilizing the limited storage capacity of switches. Researchers have proposed several approaches to tackle this challenge. One such approach is the Intelligent Timeout Master, which utilizes a cache module within the controller to collect flow expiration times. By leveraging this data, it can predict the expiration times of new flows based on their specific characteristics [14]. Another approach is TimeoutX, which considers various factors such as traffic characteristics, flow types, and the current utilization ratio of the Flow Table. By taking these factors into account, TimeoutX dynamically determines the timeout for each entry instead of relying on a static value [15]. In the work presented by Guo et al. [16], a solution called SofTware-defined Adaptive Routing (STAR) is proposed. STAR comprises three modules, including a Flow Management module that evicts flows based on their usage. The routing module periodically polls the Flow Management module to determine if new flows can be accepted. Kim et al. [17] propose a dynamic adjustment of the timeout value for each flow rule based on current traffic conditions. Similarly, Lu et al. [18] investigate dynamically adjusting the flow entry lifecycle by estimating flow statistics and setting a timeout value according to the occurring traffic. In the research presented by Liu et al. [19], the authors estimate the number of flows that may appear at the next sampling and set an idle timeout value based on the estimation. Wang et al. [20] implement proactive eviction using an adaptive algorithm that predicts idle timeout. Panda et al. [21] propose adjusting the hard timeout for a flow as packets arrive, ensuring it does not exceed the static maximum timeout value. They also utilize the Least Recently Used (LRU) algorithm for idle timeout when the TCAM reaches 90% capacity.

While the aforementioned solutions have proven to be effective, they are inherently human-centric, making them susceptible to errors. Additionally, the involvement of humans in the decision-making process introduces a time delay, hindering the convergence of these solutions. Therefore, there is a need for improvement by introducing a certain degree of autonomy. One promising avenue for enhancement is the application of Machine Learning (ML) techniques. By leveraging ML, decision-making within networks can be optimized and automated, leading to more efficient and reliable flow table optimization. ML algorithms can learn from historical data, analyze complex patterns, and make intelligent decisions in real time, reducing the reliance on manual intervention. Integrating ML into flow table optimization can enable networks to adapt dynamically to changing traffic conditions, optimize resource allocation, and improve overall performance. Autonomous decision-making algorithms can continuously learn from network behavior, adapt to new scenarios, and make proactive adjustments to flow rules and timeouts. This autonomy can lead to faster convergence, reduce human errors, and ultimately enhance the performance and security of SDN.

Initial attempts at applying ML to networking have predominantly relied on Supervised Learning techniques [22]. While these methods are efficient and powerful, they have certain limitations. Supervised Learning requires structured and accurately labeled data, as well as manual feature engineering. These constraints make Supervised Learning algorithms rigid and less effective in the context of dynamic flow rule management in networking. To overcome the limitations of Supervised Learning, Unsupervised Learning techniques have been explored. These methods can identify correlations and patterns within data without the need for labeled examples. However, the outputs of Unsupervised Learning techniques are not guaranteed to be useful and can be less accurate compared to Supervised Learning. Reinforcement Learning (RL), on the other hand, offers a promising approach to network optimization. RL allows an agent to interact with an environment, learning from rewards or punishments associated with different actions to achieve a desired behavior. DRL-FTO [23] and HQTimer [24] are examples of RL-based solutions that utilize Q-learning to optimize the timeout values of flows. These approaches passively collect network statistics, which correlate to an optimal timeout value. Given the advancements in RL techniques, such as the Twin Delayed Deep Deterministic Policy Gradient algorithm (TD3) [25], and the continuous improvements in computational technologies and hardware [26], RL is considered one of the most appropriate approaches for dynamically adjusting the timeout values of flows in the context of SDN.

3 Methodology

3.1 Twin Delayed Deep Deterministic Policy Gradient

Twin Delayed Deep Deterministic Policy Gradient (TD3) is a model-free reinforcement learning (RL) algorithm. Unlike RL algorithms that rely on a known "model" of the environment, TD3 makes predictions through trial-and-error without explicitly representing the environment. As an actor-critic method, TD3 leverages the principles of a Markov Decision Process (MDP) to guide its decision-making process. The environment encompasses

- a set of states S
- a set of actions A
- transition dynamics $T = P(s_{t+1} \mid s_t, a_t)$
- an immediate reward function $R(s_t, a_t)$
- a discount factor $\gamma \in [0,1]$,

in which an actor (i.e., Deterministic Policy) is a function that maps states to deterministic actions. Specifically, $\pi(s;\theta_\pi) : S \rightarrow A, \quad a = \pi(s;\theta_\pi)$,where π is the policy function, and θ_π represents the parameters of the actor network. The actor agent selects an action that has a quantifiable impact on the environment. Meanwhile, the critic agent evaluates the efficacy of the chosen action, and the actor agent then strives to enhance its action selection based on the critiques provided by the critic agent, as illustrated in Fig. 1. In other words, the critic estimates the action-value function, denoted as $Q(s, a)$, where Q is the action-value function, and θ_Q represents the parameters of the critic network.

As an extension of the Deep Deterministic Policy Gradient algorithm (DDPG) [27], TD3 (refer to Algorithm 1) is an off-policy method that combines policy optimization and Q-Learning. This means that the agent explores actions based on their potential benefits, rather than relying on preestablished behavior. However, DDPG often suffers from overestimation bias, which refers to the tendency to overestimate uncertain estimates. To address this issue, TD3 introduces several key improvements. Firstly, it implements Clipped Double Q-Learning, which employs two Q-functions, namely $Q_1(s, a;\theta_{Q_1})$ and $Q_2(s, a;\theta_{Q_2})$. The smaller of the two values is selected to form the targets in the loss functions, reducing the overestimation bias. Secondly, TD3 incorporates Delayed Policy Updates. It updates the policy less frequently compared to the Q-functions, which helps to stabilize the learning process and prevent the policy from overfitting to noisy Q-function estimates. Moreover, TD3 incorporates target networks with update rules as $Q'_1 \leftarrow \tau Q_1 + (1-\tau)Q'_1$, $Q'_2 \leftarrow \tau Q_2 + (1-\tau)Q'_2$, and $\pi' \leftarrow \tau\pi + (1-\tau)\pi'$, where τ represents the target network update rate. These target networks are delayed copies of the actor and critic networks. The critics are then trained to minimize the following temporal difference (TD) error:

$$y = R(s_t, a_t) + \gamma \min(Q'_1(s_{t+1}, \pi'(s_{t+1})), Q'_2(s_{t+1}, \pi'(s_{t+1}))),$$

where the Bellman error for the critics becomes:

$$L(\theta_Q) = \mathbb{E}_{(s,a,r,s')}\left[(Q_1(s, a;\theta_{Q1}) - y)^2 + (Q_2(s, a;\theta_{Q2}) - y)^2\right].$$

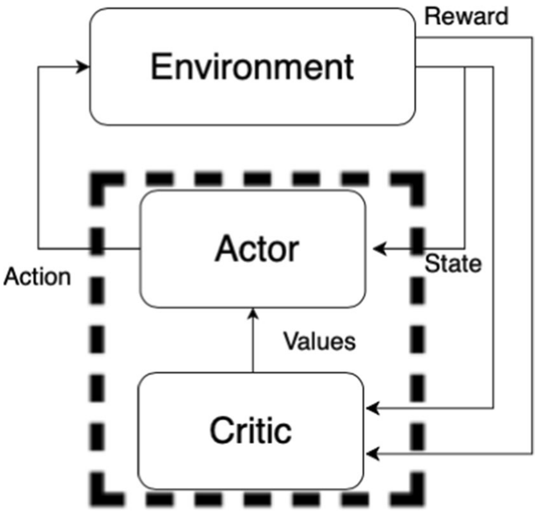

Fig. 1 Basic actor-critic model

By periodically updating the target networks using the following deterministic policy gradient,

$$\nabla_{\theta_\pi} J(\theta_\pi) = \mathbb{E}_{s\sim D}\left[\nabla_a Q_1(s, a;\theta_{Q1})\big|_{a=\pi(s)} \nabla_{\theta_\pi} \pi(s;\theta_\pi)\right]$$

where $J(\theta_\pi) = \mathbb{E}_{s\sim D}\left[Q_1(s, \pi(s);\theta_{Q1})\right]$, TD3 achieves more stable and reliable value estimations. Lastly, TD3 employs Target Policy Smoothing. This technique adds noise, denoted by ϵ to the target actions during the evaluation step as:

$$a' = \pi'(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c).$$

Here, ϵ is a random value sampled from a normal distribution with mean 0 and standard deviation σ, and the *clip* function ensures that the resulting value is bounded within the range $[-c, c]$. This is a common technique to control the magnitude of exploration noise added to the actions in reinforcement learning algorithms. This also helps to address the variance in Q-function errors and leads to better exploration during training. These enhancements make TD3 a powerful and effective algorithm for continuous control tasks in reinforcement learning.

Algorithm 1 TD3

```
1:  Initialize critic networks $Q_{\theta_1}$, $Q_{\theta_2}$, and actor network $\pi_\phi$ with random parameters
    $\theta_1$, $\theta_2$, $\phi$
2:  Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
3:  Initialize replay buffer $\mathcal{B}$
4:
5:  for $t = 1$ to $T$ do
6:      Select action with exploration noise $a \sim \pi(s) + \epsilon$,
7:      $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
8:      Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$
9:      Sample mini-batch of $\mathbf{N}$ transitions $(s, a, r, s')$ from $\mathcal{B}$
10:     $\tilde{a} \leftarrow \pi'_\phi(s) + \epsilon$, $\epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
11:     $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
12:     Update critics $\theta_i \leftarrow \min_{\theta_i} \frac{1}{\mathbf{N}} \sum (y - Q_{\theta_i}(s, a))^2$
13:     if $t$ mod $d$ then
14:         Update $\phi$ by the deterministic policy gradient:
15:         $\nabla_\phi \mathbf{J}(\phi) = \frac{1}{\mathbf{N}} \sum \nabla_a Q_{\theta_1}(s, a)|a = \pi_\phi(s) \nabla_\phi \pi_\phi(s)$
16:         Update target networks:
17:         $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
18:         $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
19:     end if
20:     end for
```

3.2 Network Data Collection via OpenFlow Messages

OpenFlow employs a message system that enables both passive and active methods for monitoring network traffic. Passive methods rely on switches within the network to send messages to the controller when specific events occur. These methods have the advantage of not generating additional overhead, minimizing the impact on network performance. However, the information that can be collected through passive methods is limited. Active methods, on the other hand, allow the controller to send statistics requests to query individual or multiple switches within the network, receiving statistics replies that can include details such as switch descriptions, flow information (individual or aggregated), flow table statistics, port and queue statistics, as well as vendor-specific information. While active monitoring provides a wider range of information, frequent querying of switches can negatively impact network performance by consuming resources and increasing overhead.

In our approach, we adopt a hybrid strategy by leveraging both passive and active methods to collect network statistics effectively. By combining these techniques, we can gather a comprehensive set of information while minimizing the impact on network performance. To implement this approach, we utilize the capabilities of the Ryu SDN controller [28], which offers a flexible and extensible framework for building SDN applications. The Ryu controller can be customized to handle various OpenFlow messages based on the specific requirements of any network application. In our proposed solution, we have utilized the following OpenFlow messages for our implementation. It should be noted that the SDN data plane, nor its innate functions, are altered in any way. Therefore, there is no added memory or resource consumption imposed by the solution. As per the OpenFlow specification [29], the minimum sizes for request and reply messages are 122 bytes and 174 bytes, respectively. Therefore, a single measurement, which includes both request and reply messages for flow statistics, would involve the exchange of 296 bytes. This, however, can be impacted by the polling frequency and the switch's location in the network [30].

3.2.1 Packet Processing

When a packet arrives at a switch, the switch first checks its flow table to find any existing matching flow rule entries. If a matching rule is found, the packet is processed accordingly based on that rule. However, if there is no matching flow rule, the switch generates a Packet_In message for the packet and forwards it to the SDN controller [31]. This Packet_In message provides the SDN controller with the necessary information to make a decision on how to handle the packet. Upon receiving the Packet_In message, the controller analyzes its contents using its packet_in_handler. This handler allows for the extraction and collection of relevant information from the incoming packet, such as the source IP address, destination IP address, communication protocol, source port, destination port, and more [32]. Based on this information, the SDN controller can make an informed decision on how to process the packet. Subsequently, the controller installs a new flow rule in the switch's flow table to ensure future packets with similar characteristics are handled efficiently.

3.2.2 Flow Rule Instillation

To install or modify flow rules in the switch's flow table, the controller utilizes the Add_Flow message. This message specifies the match criteria, actions to be performed, and the priority for the flow rule [31]. In the implementation, the controller's flow_mod_handler method is employed, which takes advantage of the specification of match fields. This allows for more precise or coarse-grained manipulation of flow rules based on specific attributes. For example, attributes such as idle_timeout and the OFPFF_SEND_FLOW_REM flag can be set using this message [32]. By leveraging these capabilities, the controller can effectively manage and customize the flow rules in the switch's flow table according to the requirements of the network.

3.2.3 Traffic Monitoring

The SDN controller has the capability to request statistics from an SDN switch using the Stats Request message. In response to this request, the switch provides the requested statistics by sending a Stats Reply message. These statistics can include various information such as individual or aggregate packet counts, byte counts, the total number of flow rule entries in the switch's flow table, duration for specific flows, and more [31]. Ryu, being a flexible SDN controller framework, offers several methods to handle the Stats Reply response from the switch. These methods include *(i) flow_stats_reply_handler*, which provides statistics for individual specified flow(s); *(ii) flow_stats_reply_handler*, which provides aggregate flow statistics; and *(iii) table_stats_reply_handler*, which offers specified table(s) statistics [32]. By utilizing these methods, the controller can effectively process and extract the desired statistics received in the Stats Reply message from the switch. This allows for comprehensive monitoring and analysis of network performance and traffic patterns within the SDN environment.

3.2.4 Flow Rule Removal

Whenever a flow rule is removed from the switch's flow table, whether it is initiated by the controller or due to expiration or an error, the switch sends a Flow Removed message to the controller [31]. This message serves to inform the controller about the details of the removed flow, including associated statistics and the reason for its removal. To handle these Flow Removed messages effectively, Ryu provides the flow_removed_handler method. This method enables the controller to track and monitor the flow rule entries based on the specified match criteria [32]. By utilizing this handler, the controller can receive and process the Flow Removed messages, allowing for efficient management and analysis of flow rule lifecycles within the network.

3.3 Feature Extraction

The absence of an SDN environment in the OpenAI Gym toolkit [33] leads to a lack of standardization and reproducibility in code. To address this limitation, we leverage the statistics collected from the switch. These statistics serve as the basis for

Table 1 Features used to define state and reward in our RL model

Parameter	Message
Table-Miss packet_In Inter-arrival Time	packet_in_handler
Flow duration	flow_removed_handler
Previous value	–
Miss rate	packet_in_handler
Inactive rate	flow_stats_reply_handler
Hit rate	packet_in_handler
Use rate	flow_stats_reply_handler

defining the features used to construct the state space and reward function for the RL model, as depicted in Table 1.

3.3.1 State Space

In our model, the State is a 5-dimensional vector constructed of features derived from statistics collection. As seen in Table 1, our model uses the flow_stats_reply_handler, packet_in_handler, and flow_removed_handler messages to formulate the following input parameters:

Table-Miss Packet_In Inter-arrival Time An averaged measurement, per unit of time-seconds, between the sending of the packet_in message by the SDN switch for previously installed flow rule entries.

Flow duration An averaged measurement of duration, the period between the arrival of the first and last packet, associated with a unique flow rule entry in the switch's flow table that has been removed due to expiring via idle timeout.

Flow table miss rate The percentage of Packet_In messages that correspond to previously installed flow rule entries matching the entries present in the cache module but not installed in the switch's flow table at the time of polling provides a metric for evaluating the effectiveness of the timeout value assigned to the flow rule entries currently occupying the flow table. This metric helps assess how well the timeout value is managing the removal of expired or unused flow rule entries from the switch's flow table.

Flow table inactive rate The inverse of the measurement of the activeness of flow rule entries present in the switch's flow table, calculates the percentage of flow rule entries actively receiving packets, at the time of polling. This metric indicates the usage of flow rule entries currently occupying the flow table.

Previous value the previous timeout value selected by the RL agent in our model.

3.3.2 Action Space

The Action Space in our RL Agent refers to the set of actions available for decision-making. In our implementation, the action set ranges from 1 to 10, where each action corresponds to a specific timeout value in seconds. This is based on the recognition of the ON-OFF pattern in traffic. According to the model cited in [34], internet traffic can be classified into two states: ON, when data is actively transmitted, and OFF, when data transmission is idle. The static timeout value approach is acknowledged in [35] as potentially inefficient for capturing the dynamic patterns of network traffic.

3.3.3 Reward Function

The Reward Function is computed using information obtained from the cache module, packet_in_handler, and flow_stats_reply_handler messages. The Reward Function consists of two key metrics. *(i) Match Rate:* This metric represents the inverse of the percentage of incoming Packet_In messages that belong to previously installed flow rule entries matching the existing entries in the cache module but not present in the switch's flow table at the time of polling. *(ii) Active Rate:* This metric measures the activeness of flow rule entries in the switch's flow table and calculates the percentage of flow rule entries that are actively receiving packets at the time of polling. These metrics serve as essential components of the Reward Function and play a crucial role in guiding the RL Agent's learning process to optimize its decision-making and overall performance.

3.4 Traffic

It is widely recognized that most internet traffic exhibits an ON-OFF pattern [34]. According to this model, traffic can be classified into two states: ON, when data is actively transmitted, and OFF, when data transmission is idle. The authors of [36] further explain that data is transmitted in fixed intervals, and this is illustrated in Fig. 2. In the figure, individual squares of different colors represent Flow 1, Flow 2, and Flow 3, denoted as F1, F2, and F3, respectively. These squares signify the packets being transmitted during periods of activity (ON), while the absence of squares represents the periods of inactivity (OFF) when no data is being sent. Moreover, the varying increments between squares associated with each flow indicate the

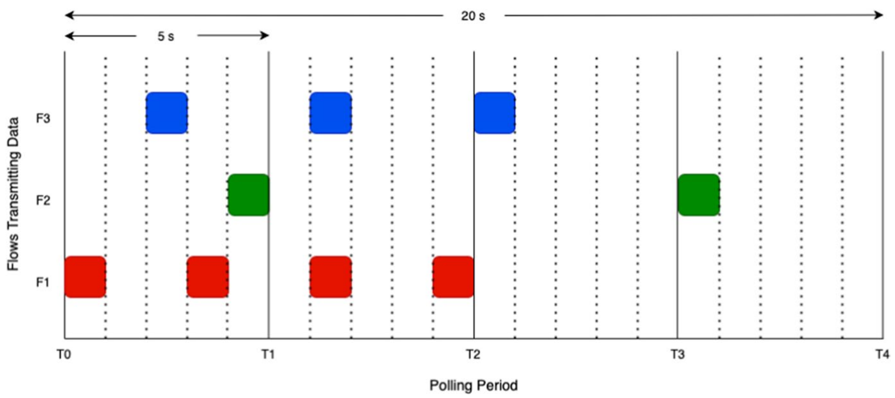

Fig. 2 ON-OFF traffic model

differences in intervals between the transmission of successive packets, with each flow following its unique pattern characterized by various durations and periods of activity and inactivity. However, it is essential to consider that using a static timeout value might not be suitable for capturing the dynamic patterns and characteristics of network traffic, as discussed in [35]. The flow of data on the global internet (aka internet traffic) encompasses a variety of information exchanged between devices and servers. Datacenter traffic represents a subset of the broader category of internet traffic. In the realm of data centers, this specific type of traffic involves the exchange of data among servers, storage devices, and networking equipment to facilitate the seamless functioning of applications and services. The on-off traffic model is characterized by bursts of high activity followed by periods of inactivity. This bursty nature is reflective of the way certain applications or services operate in data centers. For example, a database query might result in a burst of traffic while retrieving or updating records, followed by idle periods when no data transmission is needed. Therefore, this is a characteristic that can be observed in various contexts, including data centers [37]. Relying on a static timeout can lead to inefficient resource utilization and delays in flow completion, such as flow table bloat and flow table overflow. Therefore, it is crucial to explore adaptive approaches that can better accommodate the dynamic nature of internet traffic.

To generate traffic for model training, testing, and evaluation purposes, we made a significant observation based on [38]: nearly 80% of flows in a dynamic network environment last less than 10 s. In light of this finding, our training environment is designed to generate 10 flows per second. This rate is carefully scaled to mimic close to real-world conditions, where the median flow arrival rate is reported to be 10^5 flows per second (equivalent to 100 flows every millisecond) based on the results presented in [37, 38].

3.5 Framework

DDT is implemented as an application in the controller and works in real-time. It is comprised of two main components: a cache, and the RL algorithm. DDT uses the information collected from the cache and OpenFlow messages as input parameters, as well as to calculate the reward. As the RL agent receives input, it calculates Q-Values for all the possible actions for a given state. The RL agent chooses the action that has the largest Q-Value as the optimal action.

3.5.1 Cache

Our solution incorporates the cachetools Python library [39] to create a cache module with a specific purpose. This module is designed to monitor the time intervals between successive Packet_In messages from flows that previously had flow rule entries installed in the switch's flow table. To achieve this, we utilize the TTL class of the cache implementation provided by the library, which extends the standard least recently used (LRU) approach by introducing a per-element time-to-live (TTL) feature. Each element stored in the cache is associated with a predefined

TTL, indicating the duration it will remain in the cache. Once an element reaches its TTL, it automatically expires and is removed from the cache. Here, the process begins when the packet_in_handler of the controller receives a packet_in message. It extracts essential information, such as the Ethernet destination address, Ethernet source address, source IP address, destination IP address, and IP protocol, to create a unique identifier for the corresponding flow. The cache is then searched for this unique identifier. If it is not present in the cache, it is added along with a timestamp denoting the time of insertion. Conversely, if the unique identifier is already in the cache, it indicates that a flow rule entry was previously installed within the specified TTL time-frame but was evicted prematurely due to reaching the TTL's limit. In such cases, the relevant statistics are recalculated, and the timestamp is updated. The TTL's duration is a crucial factor in the cache module's effectiveness. If the TTL is set too short, the agent may not accurately penalize instances where larger intervals between data transmissions lead to premature eviction of flow rule entries. On the other hand, if the TTL is excessively long, the agent might incorrectly punish multiple transmissions between end hosts whose initial or previous transmission had ended. This could affect the accuracy of statistic collection, as the timestamps associated with unique identifiers are used to calculate the time elapsed since their initial or last installation up to the current update. Striking the right balance in setting the TTL is essential for ensuring accurate and reliable statistic collection.

3.5.2 Deep RL

Before the initial polling period begins, incoming flows will be assigned an idle timeout of 10 s as the default value. This choice ensures that the majority of incoming flows will timeout upon completion rather than being prematurely evicted. This approach allows the network to establish an average flow duration, aiding in better flow management. As packets start arriving, the cache module generates unique keys, as described earlier, to create corresponding flow rule entries, which are then added to the switch's flow table. Once the first polling period starts, the application initiates queries to the network's switch(es) using the controller's OpenFlow message handlers to collect various network statistics associated with the unique keys stored in the cache module. These statistics include Table-Miss Packet_In Inter-arrival Time, Average Flow Duration, Flow Table Miss Rate, and Flow Table Inactive Rate. Our solution explores the ON-OFF traffic model and emphasizes the importance of adaptive approaches. In our evaluation, a flow can last a minimum of 1 s, a maximum of 10 s, or anywhere in between. Therefore, the approximation for the optimal idle timeout value for a flow is between 1 and 10, inclusive, as a flow's idle interval cannot be less than 1 s and is not likely to exceed 10 s. The dynamic nature of the traffic and momentary accuracy of collected statistics are considered and require a sufficient polling window. If the polling window is too short, it will lead to incomplete data capture, potentially missing crucial changes in traffic patterns; if the polling window is too large, it may result in oversampling and excessive computational overhead, diminishing the real-time adaptability of the agent. The agent polls the switch every 5 s, gathering information and calculating average values over a 20-s period. By regularly polling the switch and analyzing data over a 20-s timeframe, the agent gains a more comprehensive understanding of the network's behavior. It should be noted that the polling interval does not directly impact reaction time as the timeout value is based on the average of the values collected over a 20 s period. This approach allows the agent to adapt to the dynamic nature of traffic, contributing to better resource utilization and flow completion efficiency. Additionally, the Previous Idle Timeout Value, which is chosen by the RL agent, is also considered as an input state. Furthermore, the application gathers data on the percentage of flows that are actively receiving packets and the percentage of packets matched to flow rule entries during the 20-s polling window. These metrics are utilized to calculate the reward for the RL agent. By analyzing these statistics and metrics, the RL agent gains valuable insights into the network's characteristics and behavior, enabling it to decide whether a different timeout value should be assigned to incoming flows. In summary, the combination of default idle timeouts, flow rule entries based on unique keys, and extensive network statistics empowers the RL agent to make informed decisions about optimizing timeout values for incoming flows.

3.6 Training

The environment simulates an SDN topology comprising a single SDN controller connected to an OpenFlow-compatible switch, which, in turn, is connected to *200* end-hosts. Each host can send and receive data with any other host, generating network traffic with a flow ratio of *9:1*, where approximately *90%* of flows last less than 10 s and the remaining *10%* last *10* s or more. Network traffic is simulated using the Multi-Generator (MGEN) traffic generation tool [40] at a rate of *10* flows per second. The toolset generates real-time traffic patterns so that the network can be loaded in a variety of ways.

During the training phase, the agent polls the switch every *5* s to gather information and calculates average values over a 20-s period. It then stores the previous state, current state, reward, and action in the replay buffer, treating them as transitions to be used for training. The agent employs an *epsilon*-greedy exploration strategy during training, starting with an exploration rate of *1.0* and decaying it until it reaches a minimum value of *0.01*. The agent can learn from transitions stored in the replay buffer, which helps optimize its policy, irrespective of the current policy. The entire training process consists of *1000* episodes, each lasting *300* s (*5* min), and the agent utilizes the replay buffer to optimize its policy throughout the training phase. The total training time spans a minimum of *300,000* s. Overall, the agent undergoes *13,208* training iterations over *15,000* steps, employing a batch of 30,000 transitions, corresponding to *3,000,000* flows in the network.

4 Implementation

The testbed environment utilized for both training and testing was based on a Lambda Cloud instance, featuring 30 virtual Intel Xeon Platinum 8358 CPUs @ 2.60GHz (vCPUs) with 24 GB of virtual RAM (vRAM) per vCPU.

In the software environment, the following components were used: Mininet [41], a lightweight network emulator; Ryu SDN Controller; Open vSwitch [42], Mininet's default virtual switch; OpenFlow v1.3; and the MGEN traffic generation tool [40]. The operating system employed for the software environment was Ubuntu 20.04.5 LTS (64-bit).

While OpenFlow may not be considered state-of-the-art, it remains widely used, supported by various open-source and proprietary SDN controllers such as RYU [28], OpenDaylight [43], and Nox [44]. It is integral in Open vSwitch and OpenStack, as well as SEBA for broadband access and COMAC for converging mobile and broadband networks. Google utilizes OpenFlow extensively in projects like B4, Maglev, Jupiter, Andromeda, and Espresso. Furthermore, flow tables are a fundamental component of the data plane in SDN, and they are not specific to any particular protocol. Therefore, this problem exists as a broader challenge within SDN, and consequently, any SDN protocol or switch that employs matching and tables [1, 45] can benefit from this solution. Therefore, OpenFlow was chosen because of its available documentation and use in publicly available literature that propose solutions to this problem. Version 1.3 was selected due to its capabilities necessary to implement our proposed solution and its adaptability to newer versions. While there may not exist a one-to-one equivalency, any logic used to develop a solution in OpenFlow should be portable to a P4 environment with sufficient modification [46].

5 Evaluation

The model's performance was evaluated by comparing it against static timeout values ranging from 1 to 10 on a dataset comprising *1000* flows. These flows were generated with the same size and transmission rate parameters as those used during the training phase.

Table 2 highlights the trends in flow rule entries concerning different timeout values. It is observed that longer timeouts lead to a decrease in active entries, while

Table 2 Timeout values and metrics

Timeout value	Avg. active	Avg. match	Packet in messages
DDT	0.79	0.80	7826
1	1.00	0.72	8616
2	0.94	0.73	8546
3	0.93	0.74	8380
4	0.92	0.77	7905
5	0.90	0.79	7889
6	0.76	0.80	7798
7	0.71	0.83	7707
8	0.65	0.86	7610
9	0.61	0.90	7470
10	0.53	1.00	7441

match rates remain relatively stable. Moreover, longer timeouts result in higher match values and reduced Packet_In Messages. Figure 3 demonstrates the superiority of Delayed Dynamic Timeout (DDT) over static timeouts ranging from 1 to 10. DDT consistently outperforms the static alternatives in terms of various metrics. Overall, the choice of timeout significantly impacts network activity, match rates, and processing efficiency. The model's contributions in this context include generalizability, flexibility, scalability, reduced training time, and improved efficiency.

5.1 Analysis of the Obtained Results

After analyzing Table 2, it becomes evident that there exists a correlation between the timeout value and its impact on the match rate and activity rate. To begin with, Fig. 4 illustrates a notable decreasing trend in the average number of flow rule entries actively receiving packets as the timeout value increases. This suggests that larger timeout values lead to a reduction in the average number of active transmissions or processes. Consequently, it implies that longer timeout values result in a lower overall activity level of flow rule entries within the switch's flow table. Furthermore, when examining the average match rate of flow rule entries in Fig. 3, there is no clear trend across the different timeout values. The average match rate remains relatively stable, indicating a consistent level of success or efficiency in the operations or queries being performed, regardless of the timeout value. Here, the analysis reveals that increasing the timeout value has a discernible impact on the activity rate of flow rule entries, while the average match rate remains consistent across different timeout values.

There also exists a relationship between the average amount of flow rule entries actively receiving packets and the average match rate, as seen in Fig. 5. As

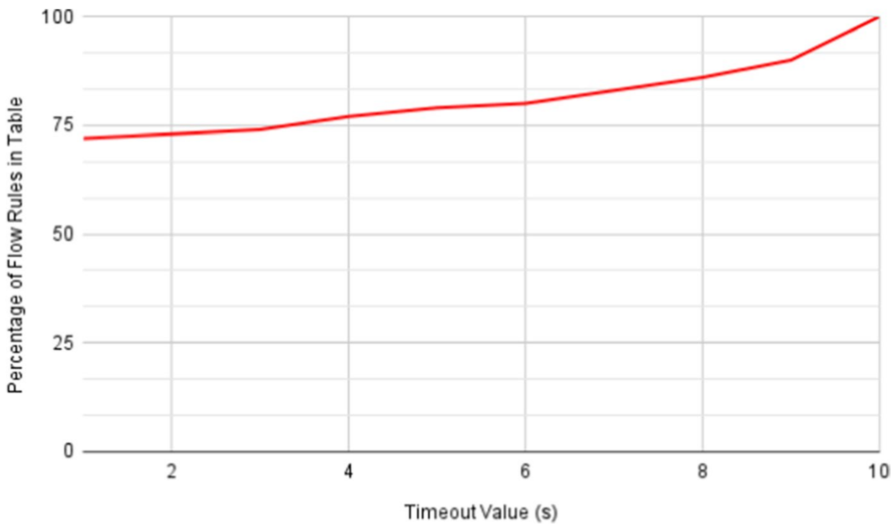

Fig. 3 Percentage of rule entries matching incoming packets

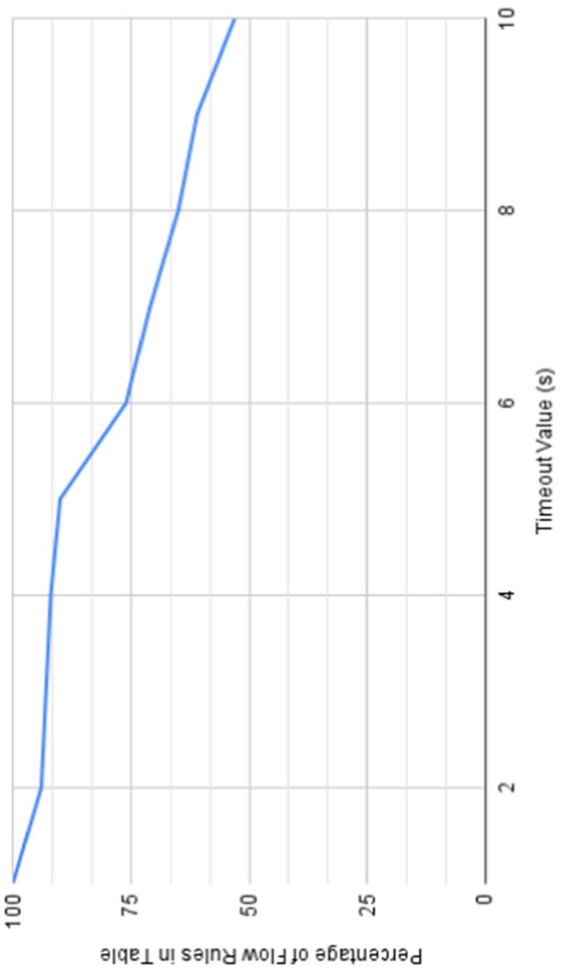

Fig. 4 Percentage of rule entries actively receiving packets

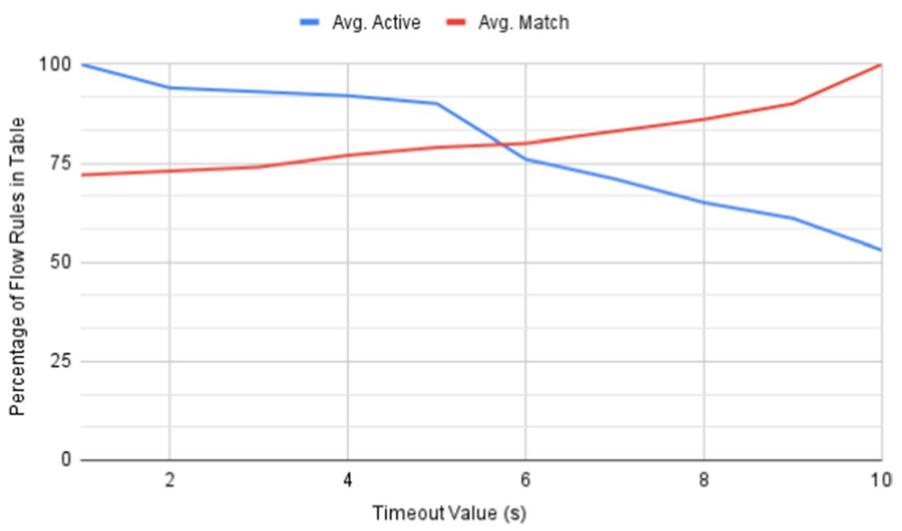

Fig. 5 Comparison of activity & match among static values

the average active value decreases with increasing timeout values, there is a slight increase in the average hit values. Figure 6 illustrates the comparison between the solution, Delayed Dynamic Timeout (DDT), and static timeout values ranging from 1 to 10. The graph shows that the average amount of flow rule entries actively receiving packets or the average match rate value of DDT are relatively

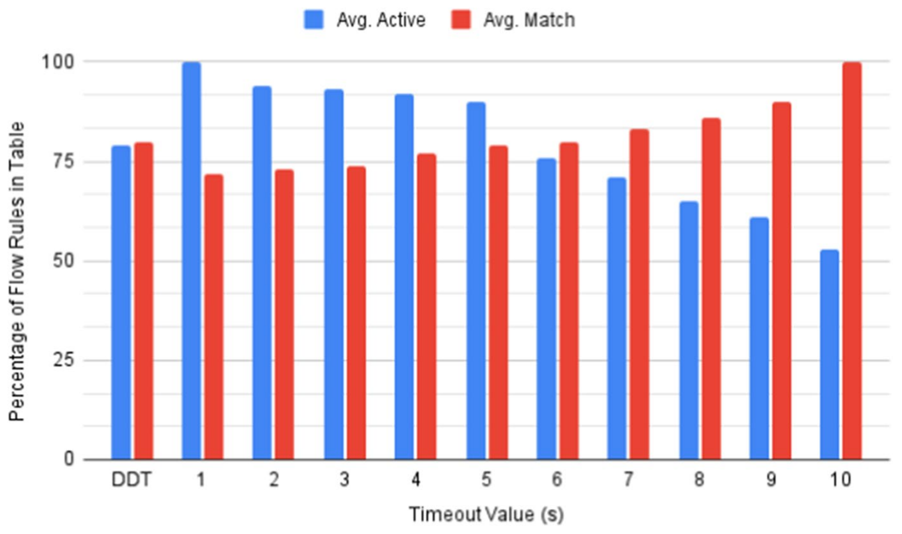

Fig. 6 Delayed Dynamic Timeout (DDT) compared to static values

higher than those of the static timeout values, indicating that DDT achieves better performance in terms of active transmissions and match ratio.

5.2 Comparative Analysis with State-of-the-Art Solutions

DDT focuses on autonomous decision-making within the network brain, representing a step toward the realization of self-operating networks. Aligned with research findings discussed in the relevant state-of-the-art literature, as detailed in Table 3, DDT also introduces distinctive features, making its unique contribution to the existing body of related literature.

5.3 Limitations

Our solution relies on multiple computationally complex and resource-intensive neural networks. However, actively polling the network's switch(es) introduces communication cost and overhead, consuming additional computational resources. Furthermore, using Mininet as a testbed environment imposes limitations on the scalability of the network due to the sharing of computational resources among devices and applications within the emulation.

The lack of available code and standardized benchmarks for RL-based SDN timeout solutions presents challenges in evaluating and comparing different approaches. The absence of a common framework or reference implementation makes it difficult to assess the performance, reliability, and effectiveness of various timeout solutions consistently and objectively. Additionally, there is no standardized method for simulating network traffic with real-world characteristics, such as bursty traffic and temporal variations. The absence of realistic traffic scenarios in evaluations hinders accurate assessments of the proposed solutions' performance and robustness in practical deployments.

Another concern is that the reference model and OpenFlow protocol specification do not define a clear method for forming a global network view. As a result, obtaining a comprehensive and accurate representation of the network state can be challenging. Moreover, data collected by polling the network is only accurate at the time of collection, which can lead to skewed calculations and inaccurate decisions during traffic bursts or data reset periods. The assumptions about data collection time may not always hold true, affecting data accuracy in critical moments.

6 Conclusion and Future Work

In this paper, we presented Delayed Dynamic Timeout (DDT), an innovative approach that utilizes Reinforcement Learning (RL) to optimize resource management in Software-Defined Networking (SDN) through dynamic flow timeout assignment. By leveraging the inherent capabilities of the OpenFlow protocol, advancements in RL techniques, and the incorporation of a cache module, DDT dynamically adapts the idle timeout values of flow rule entries based on real-time changes in

Table 3 Comparative analysis of DDT and existing solutions

Solution	Network planes	Algorithm	Implementation	Metrics	Results commentary
HQTimer [24]	Uses reference SDN Architecture No modification to the switch specification	Q-learning	Simulation using synthesized datasets	Table hit rate Number of overflow occurrences	No rule dependency problem Higher table-hit rate than static idle timeout Less overflow occurrences than static idle timeout
Adaptive data forwarding [47]	Both control and data planes are involved Requires OpenFlow or switch design modification	A statistical control mechanism	Simulation using MATLAB	Table hit rate	Improved flow table hit rate and capacity utilization
DRL-FTO [23]	Uses just the control plane	Deep reinforcement learning (Deep Q-Network (DQN))	Proof of concept emulation Synthesized traffic using Iperf	Number of exchanged controller-switch messages	Minimized controller-switch communication overhead
DRL-Idle [48]	Statistical analysis on both control and data planes	A heuristic based on deep reinforcement learning	Simulation using synthesized datasets	Flow-based Service Time (FST) Total number of flow installations	Improved QoS through flow management
DeepPlace [49]	Uses both the control and the data planes	Deep reinforcement learning	Statistical model evaluated by emulation Synthesized traffic using Hping3	Number of used match-fields in a flow rule QoS violation ratio	Assured QoS of traffic Decreased flow table overflow in the data plane Best for Software-Defined Internet of Things networks

Table 3 (continued)

Solution	Network planes	Algorithm	Implementation	Metrics	Results commentary
DDT (This work)	Uses just the control plane No modification to the southbound Protocol or the switch specification	Twin delayed deep deterministic policy gradient (TD3)	Proof of concept emulation Synthesized traffic using MGEN	Flow match rate Number of active flows in a table at a time	Higher average active transmissions Improved flow entry match ratio Minimized controller-switch communication overhead

network traffic. Our evaluation of DDT against static timeouts showed quantifiable improvement, particularly in terms of match or activity rate, while maintaining a commendable level of efficiency in other categories. This highlights the potential of RL-based solutions for achieving better resource allocation and performance in SDN environments.

Looking ahead, we plan to expand the applicability of DDT by implementing it on a more complex SDN topology, closely resembling real-world networks. Currently limited to a single switch, our future work involves scaling DDT to accommodate multiple switches, thereby increasing its practicality and relevance in larger network architectures. Additionally, we aim to explore the use of programmable data planes and distributed RL training techniques. By reducing the training and runtime overhead on the SDN controller, we can enhance DDT's efficiency and decision-making capabilities, making it even more suitable for larger and more dynamic network scenarios.

Acknowledgements This work was supported in part by National Science Foundation (NSF) Grants #2113945 and #2200538. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agency.

Author Contributions Nathan Harris Jr. and Sajad Khorsandroo contributed equally to the conception of this research and the subsequent documentation. Nathan Harris Jr. is responsible for the design, coding, evaluation, and analysis of the proposed method.

Funding Open access funding provided by the Carolinas Consortium. This work was supported in part by National Science Foundation (NSF) Grants #2113945 and #2200538.

Data Availability The datasets and trained model(s) generated and/or analysed during the current study are not publicly available due to this work's use in Doctoral research but are available from the corresponding author on reasonable request upon completion of degree.

Code Availability https://github.com/thedeu2e/DDT-App. Due to use in Doctoral research, the code in this repository is being updated with unpublished findings and is not publicly available. However, it can be made available from the corresponding author upon reasonable request upon completion of the degree.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical Approval Not applicable.

Consent to Participate Not applicable.

Consent for Publication Not applicable.

References

1. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. rev. **38**(2), 69–74 (2008)
2. Open Networking Foundation. https://opennetworking.org/
3. Katta, N., Alipourfard, O., Rexford, J., Walker, D.: Infinite cacheflow in software-defined networks. In: Proceedings of the Third Workshop on Hot Topics in Software-defined Networking, pp. 175–180. ACM (2014)
4. Khorsandroo, S., Tosun, A.S.: Time inference attacks on software defined networks: Challenges and countermeasures. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), pp. 342–349. IEEE (2018)
5. Khorsandroo, S., Tosun, A.S.: White box analysis at the service of low rate saturation attacks on virtual SDN data plane. In: 2019 IEEE 44th LCN Symposium on Emerging Topics in Networking (LCN Symposium), pp. 100–107. IEEE (2019)
6. Qiao, S., Hu, C., Guan, X., Zou, J.: Taming the flow table overflow in openflow switch. In: Proceedings of the 2016 ACM SIGCOMM Conference, pp. 591–592 (2016)
7. Wang, L., Song, C., Xu, Z., et al.: Proactive mitigation to table-overflow in software-defined networking. In: 2018 IEEE Symposium on Computers and Communications (ISCC), pp. 00719–00725. IEEE (2018)
8. Curtis, A.R., Mogul, J.C., Tourrilhes, J., Yalagandula, P., Sharma, P., Banerjee, S.: Devoflow: scaling flow management for high-performance networks. ACM SIGCOMM Comput. Commun. Rev. **41**(4), 254–265 (2011)
9. Aslan, M., Matrawy, A.: On the impact of network state collection on the performance of SDN applications. IEEE Commun. Lett. **20**(1), 5–8 (2015)
10. Zarek, A., Ganjali, Y., Lie, D.: Openflow Timeouts Demystified. Univ. of Toronto, Toronto (2012)
11. Ryu, B., Cheney, D., Braun, H.-W.: Internet flow characterization: adaptive timeout strategy and statistical modeling. In: Workshop on Passive and Active Measurement (PAM), vol. 105 (2001)
12. Li, T., Zhou, H., Luo, H., You, I., Xu, Q.: Sat-flow: multi-strategy flow table management for software defined satellite networks. IEEE Access **5**, 14952–14965 (2017)
13. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
14. Zhu, H., Fan, H., Luo, X., Jin, Y.: Intelligent timeout master: dynamic timeout for SDN-based data centers. In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 734–737. IEEE (2015)
15. Zhang, L., Wang, S., Xu, S., Lin, R., Yu, H.: Timeoutx: an adaptive flow table management method in software defined networks. In: 2015 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. IEEE (2015)
16. Guo, Z., Liu, R., Xu, Y., Gushchin, A., Walid, A., Chao, H.J.: Star: preventing flow-table overflow in software-defined networks. Comput. Netw. **125**, 15–25 (2017)
17. Kim, T., Lee, K., Lee, J., Park, S., Kim, Y.-H., Lee, B.: A dynamic timeout control algorithm in software defined networks. Int. J. Future Comput. Commun. **3**(5), 331 (2014)
18. Lu, M., Deng, W., Shi, Y.: Tf-idletimeout: improving efficiency of TCAM in SDN by dynamically adjusting flow entry lifecycle. In: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 002681–002686. IEEE (2016)
19. Liu, Y., Tang, B., Yuan, D., Ran, J., Hu, H.: A dynamic adaptive timeout approach for SDN switch. In: 2016 2nd IEEE International Conference on Computer and Communications (ICCC), pp. 2577–2582. IEEE (2016)
20. Wang, D., Li, Q., Wang, L., Sinnott, R.O., Jiang, Y.: A hybrid-timeout mechanism to handle rule dependencies in software defined networks. In: 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 241–246. IEEE (2017)
21. Panda, A., Samal, S.S., Turuk, A.K., Panda, A., Venkatesh, V.C.: Dynamic hard timeout based flow table management in openflow enabled SDN. In: 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), pp. 1–6. IEEE (2019)
22. Boutaba, R., Salahuddin, M.A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., Caicedo, O.M.: A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. J. Internet Serv. Appl. **9**(1), 1–99 (2018)

23. Haq, F., Naaz, A., Bantupalli, T.P.K., Kataoka, K.: Drl-fto: dynamic flow rule timeout optimization in SDN using deep reinforcement learning. In: Asian Internet Engineering Conference, pp. 41–48 (2021)
24. Li, Q., Huang, N., Wang, D., Li, X., Jiang, Y., Song, Z.: Hqtimer: a hybrid q-learning-based timeout mechanism in software-defined networks. IEEE Trans. Netw. Serv. Manag. **16**(1), 153–166 (2019)
25. Fujimoto, S., Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: International Conference on Machine Learning, pp. 1587–1596. PMLR (2018)
26. Usama, M., Qadir, J., Raza, A., Arif, H., Yau, K.-L.A., Elkhatib, Y., Hussain, A., Al-Fuqaha, A.: Unsupervised machine learning for networking: techniques, applications and research challenges. IEEE Access **7**, 65579–65615 (2019)
27. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
28. RYU. https://ryu-sdn.org
29. Specification, O.S.: Open networking foundation. Version ONF TS-015 **1**(3), pp. 1–164 (2013)
30. Yahyaoui, H., Zhani, M.F., Bouachir, O., Aloqaily, M.: On minimizing flow monitoring costs in large-scale SDN networks
31. Foundation, O.N.: OpenFlow Switch Specification. https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf
32. OpenFlow v1.3 Messages and Structures: Ryu 4.34 documentation. https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html
33. Vidyadhar, V., Nagaraj, R., Ashoka, D.: Netai-gym: customized environment for network to evaluate agent algorithm using reinforcement learning in open-AI gym platform. Int. J. Adv. Comput. Sci. Appl. **12**(4) (2021)
34. Zhao, Y., Zhang, B., Li, C., Chen, C.: On/off traffic shaping in the internet: motivation, challenges, and solutions. IEEE Netw. **31**(2), 48–57 (2017)
35. Chandrasekaran, B.: Survey of Network Traffic Models. Waschington University in St. Louis CSE **567** (2009)
36. Adas, A.: Traffic models in broadband networks. IEEE Commun. Mag. **35**(7), 82–89 (1997)
37. Benson, T., Anand, A., Akella, A., Zhang, M.: Understanding data center traffic characteristics. ACM SIGCOMM Comput. Commun. Rev. **40**(1), 92–99 (2010)
38. Kandula, S., Sengupta, S., Greenberg, A., Patel, P., Chaiken, R.: The nature of data center traffic: measurements & analysis. In: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, pp. 202–208 (2009)
39. cachetools 5.3.1. https://pypi.org/project/cachetools/
40. Laboratory, U.N.R.: Multi-Generator (MGEN) traffic generation tool (2021). https://github.com/USNavalResearchLaboratory/mgen
41. Kaur, K., Singh, J., Ghumman, N.S.: Mininet as software defined networking testing platform. In: International Conference on Communication, Computing & Systems (ICCCS), pp. 139–42 (2014)
42. Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., et al.: The design and implementation of open vswitch. In: 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15), pp. 117–130 (2015)
43. OpenDaylight. https://opendaylight.org/
44. Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S.: Nox: towards an operating system for networks. ACM SIGCOMM Comput. Commun. Rev. **38**(3), 105–110 (2008)
45. Harkous, H., Jarschel, M., He, M., Pries, R., Kellerer, W.: P8: P4 with predictable packet processing performance. IEEE Trans. Netw. Serv. Manag. **18**(3), 2846–2859 (2020)
46. P4.org: Clarifying the differences between P4 and OpenFlow. Open Networking Foundation (2021). https://opennetworking.org/news-and-events/blog/clarifying-the-differences-between-p4-and-openflow/
47. Xie, L., Zhao, Z., Zhou, Y., Wang, G., Ying, Q., Zhang, H.: An adaptive scheme for data forwarding in software defined network. In: 2014 Sixth International Conference on Wireless Communications and Signal Processing (WCSP), pp. 1–5. IEEE (2014)
48. Jiménez-Lázaro, M., Berrocal, J., Galán-Jiménez, J.: Flow-based service time optimization in software-defined networks using deep reinforcement learning. Comput. Commun. (2024)
49. Nguyen, T.G., Phan, T.V., Hoang, D.T., Nguyen, H.H., Le, D.T.: Deepplace: deep reinforcement learning for adaptive flow rule placement in software-defined IoT networks. Comput. Commun. **181**, 156–163 (2022)

Nathan Harris Jr. is a Computer Science Doctoral Candidate at North Carolina Agricultural & Technical State University College of Engineering. He received a bachelor's degree in computer science from the University of Arkansas at Pine Bluff and a master's degree in computer science from Jackson State University. Nathan is also a Department of Defense (DoD) Science, Mathematics, and Research for Transformation (SMART) Scholarship-for-Service recipient. His research interests lie in the application of reinforcement learning to software defined networking (SDN) with the aim of creating self-driving networks.

Sajad Khorsandroo obtained his Ph.D. from the University of Texas at San Antonio in 2019. He subsequently joined the Department of Computer Science at North Carolina A&T State University. Currently serving as an Assistant Professor within the department, his research focuses on self-driving networks, the application of AI/ML in computer networks, and cybersecurity. Dr. Khorsandroo has secured funding for his research endeavors from notable federal and state agencies such as NSF, DoD, and Carolina Cyber Network, as well as from industry partners including Palo Alto Networks, Inc.