DART: Data Plane Load Reduction for Traffic Flow Migration in SDN

Ilora Maity, Student Member, IEEE,
Sudip Misra, Senior Member, IEEE, and Chittaranjan Mandal, Senior Member, IEEE

***Abstract*—In this paper, we present a traffic-aware flow migration approach, which reduces data plane load in Software-Defined Networking (SDN) during a network update. SDN update involves rerouting of multiple traffic flows to accommodate new flows. An unplanned flow migration schedule overloads the data plane by burdening the data links and flooding the rule-space of capacity-constrained SDN switches. The overload of data links and switches blocks the update process, and the network fails to address the Quality of Service (QoS) demands of the traffic flows, especially latency-sensitive flows. Prior approaches migrate flows without considering load reduction of the data plane along with QoS demands of the flows. In this work, we propose a load reduction strategy that prioritizes traffic flows based on QoS demands and aims to avoid link congestion and rule-space overflow during flow migration. The proposed scheme significantly reduces the maximum data link bandwidth usage. In particular, the maximum data link bandwidth usage is 13.22% less than the two-phase update approach.**

***Index Terms*—DN, Traffic Flow Migration, Data Plane Load, Coalition Graph Game, Rule-Space Management.DN, Traffic Flow Migration, Data Plane Load, Coalition Graph Game, Rule-Space Management.S**

I. Introduction

SDN involves heterogeneous devices connected to a programmable network that dissociates control logic from forwarding elements or switches [1] [2]. For large-scale networks, the devices associated with the switches are enormous in number and generate a massive amount of traffic flows [3]. Therefore, new flows are generated frequently, and an initial route is assigned to each new flow based on the existing flow-rules. However, based on the available bandwidth of the data links and latency requirement of the flows, the network operators may migrate the paths of some existing and/or new flows [4]. So, flow migration is an essential feature of network updates. Additionally, the flows are heterogeneous in terms of QoS demand [5], and the majority of the flows are latency-sensitive [6]. Therefore, the flow migration process should adhere to the traffic characteristics of the flows. Otherwise, the update delay makes the final route invalid for the majority of the flows. Moreover, the migration process should be feasible in terms of link capacity and rule-space capacity to avoid excessive processing delay and poor user experience.

SDN data plane has two elements — (1) SDN switches and (2) data links which are the links between the switches. Switches store forwarding information in the form of flow-rules in flow tables [7] and the rule-space capacity of each switch is limited [8]. Similarly, the data links also have a capacity constraint in terms of bandwidth usage. So, the data plane load comprises of rule-space and link bandwidth usages.

The authors are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India (Email: imaity@iitkgp.ac.in; smisra@cse.iitkgp.ac.in; chitta@iitkgp.ac.in).

SDN update routinely routes existing flows to a new path to accommodate new flows. During flow migration, controllers transmit update packets to the required SDN switches to modify flow-path from an initial path to a final path. Accordingly, the switches install new rules. However, if any switch has insufficient rule-space for the installation of a new flow-rule, the traffic flows involving the corresponding switch are routed incorrectly and suffer packet loss. Similarly, congestion in one of the links in the routing path of a flow degrades network performance by causing packet loss.

Therefore, the migration of traffic flows should consider important parameters, such as feasibility and consistency. A *feasible* flow migration reduces data plane load by ensuring that no links get congested after the migration. The feasibility constraint confirms that all the switches involved in the update process have enough rule-space capacity to install the new flow-rules required for the migration. On the other hand, a *consistent* flow migration guarantees that each migrating flow follows either old or new configuration (not both) after the initiation of migration [9]. A consistent flow migration should be blackhole free so that no packet is dropped and loop free so that no flow forms loop due to incorrect forward during migration [10]. Moreover, the migration process should consider the heterogeneous latency demands of the flows to avoid QoS violations. Therefore, feasible, consistent, and traffic-aware flow migration is necessary to maintain network performance.

Figure 1 shows the necessity of forming a feasible flow migration schedule. The nodes in each graph represent SDN switches, and the edges represent data links connecting the switches. The capacity of each data link is 1 Gbps. For this scenario, we consider two to-be-migrated traffic flows — f_1 and f_2, each having a bandwidth of 1 Gbps. The old paths of f_1 and f_2 are $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_5$ and $s_3 \rightarrow s_4 \rightarrow s_5$, respectively. The new paths of f_1 and f_2 are $s_1 \rightarrow s_2 \rightarrow s_4 \rightarrow s_5$ and $s_3 \rightarrow s_5$, respectively. Therefore, the sets of to-be-updated switches for f_1 and f_2 are $\{s_2, s_4\}$ and $\{s_3\}$, respectively. A flow is consistent if each packet of a flow is entirely processed by either old flow-rules or new flow-rules. For example, if a packet of f_2 is processed by an old flow-rule at s_3, then the packet is forwarded to s_4. In this scenario, consistent processing ensures that the packet is handled by an old flow-rule at s_4 to reach s_5. On the other hand, if a packet of f_2 is processed by a new flow-rule at s_3, the packet reaches the destination s_5 directly. Rule-space capacity constraint is another aspect that we consider for flow migration. From the scenario depicted in Figure 1, we observe that an additional flow-rule is installed in s_4 for the migration of f_1 because s_4 is not present in the old path. Therefore, if s_4 does not have free rule-space for the installation of the new flow-rule, all the packets of f_1 reaching s_4 are dropped. However, for s_2 and s_3, no additional rule-space is required if the new rules replace the old ones. Additionally, we consider link capacity constraint and QoS demands of the flows. Let f_1 is more latency-sensitive than f_2. Therefore, if f_1 is migrated before f_2, the data link from s_4 to s_5 is congested. However, if f_2 is migrated before f_1, the link from s_3 to s_5 is congested. Therefore, a QoS-aware migration schedule should also consider the link

capacity constraint. Motivated by this scenario, in this work, we form a feasible flow migration schedule that satisfies the bandwidth demand, rule-space requirement, and QoS demands of the flows by migrating the flows in groups with a consistent update process.

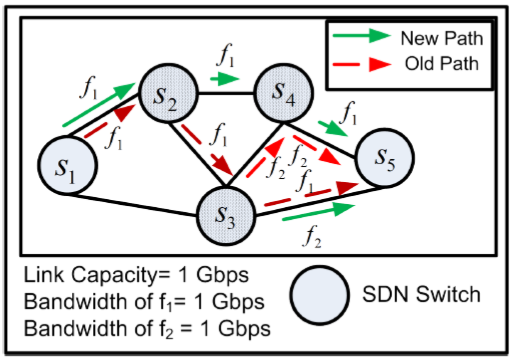


Fig. 1: Motivating Scenario

In this work, we propose a feasible and consistent flow migration approach for SDN, that balances data plane load during the migration process. The proposed approach for data plane load reduction for traffic flow migration in SDN, named DART, consists of three modules — (a) *generation of QoS-aware migration schedule*, (b) *generation of feasible migration schedule*, and (c) *rule-space management*. Initially, DART formulates a coalition graph game [11] to generate a QoS-aware flow migration schedule. Based on the initial schedule, DART verifies the possibility of link congestion and prepares a feasible migration schedule. The *rule-space capacity management* module deletes selected rules from the switches having low free rule-space. Finally, the flow-paths are updated consistently according to the schedule. The primary contributions of our work are as follows:

- We formulate an Integer Linear Program (ILP) to maximize the number of latency-sensitive flows in each update stage, considering data plane load.
- We formulate a coalition graph game to determine the set of flows that must be migrated together. In this game, we compute the utility value for each to-be-migrated flow. Based on the utility, related flows are grouped, and an initial flow migration schedule is formed.
- Based on the initial migration schedule, we design an algorithm to transform the initial migration schedule to a feasible schedule, which reduces link capacity violations.
- Additionally, we analyze the rule-space usage in the switches and propose an algorithm that ensures the required rule-space for the migration process.

II. Related Work

A. Consistent Flow Migration

Consistent flow migration approaches aim to preserve packet consistency while rerouting flows. Reitblatt *et al.* [12] proposed a two-phase update approach to guarantee packet consistency. In this scheme, the first phase installs new flow-rules. However, at this stage, the packets are processed by old flow-rules only. The second phase stamps each incoming packet with a new version number. Subsequently, the packets with new version numbers are processed by new flow-rules. After the processing of all the old packets, a garbage collection phase removes the older flow-rules. However, this approach wastes rule-space due to the storage of two versions of each flow-rule for the entire update duration. Moreover, in the second phase, the new path of flow may become functional before other to-be-migrated flows and cause link congestion. McGeer *et al.* [13] proposed a buffered update approach that buffers the incoming packets at the control plane during an ongoing update. Additional rules are installed in each switch to redirect packets to the control plane. Mizrahi *et al.* [14] proposed a time-triggered scheme for flow update which starts each update phase at a specific time. Lu *et al.* [15] proposed an update approach that preserves packet consistency by scheduling rule updates at specific time windows. However, precise time synchronization depends entirely on the characteristics of particular SDN switches. Basta *et al.* [16] proposed a flow migration approach to reduce the number of times a switch is updated to migrate all flows. In this approach, the shortest common supersequence is generated based on the switches in the new paths of the to-be-migrated flows. The switches are updated sequentially according to the supersequence. This approach considers homogeneous flows.

B. Capacity-Aware Flow Migration

Capacity-aware flow migration schemes focus on link capacity and rule-space capacity constraints during flow migration. Ludwig *et al.* [17] proposed an ordered-update technique that schedules flow-path update in multiple rounds. In this case, a round starts only after the completion of the previous round. This work aims to minimize the number of controller interactions and guarantee consistent flow migration. This approach does not store additional flow-rules. However, in this approach, existing packets processed by old flow-rules may be processed by new flow-rules after the completion of an update round. Therefore, in this scenario, the ordered update approach suffers from inconsistent forwarding. Vissicchio and Cittadini [18] considered rule-space capacity constraint and proposed a flow migration scheme by combining the ordered update approach with version numbering technique of two-phase update approach. This scheme reduces the use of additional rules. However, this approach does not consider the link capacity constraint. Zheng *et al.* [19] proposed a time-synchronized scheme, named Chronicle, that schedules the migration of multiple flows considering the link capacity constraint. In this approach, the migrating flows are divided into update blocks, and the update of each block is scheduled based on resource dependency. However, the computational complexity of the proposed approach is high. In another work, Amiri *et al.* [20] designed a polynomial-time algorithm for congestion-free migration of two flow. However, this approach does not work for more than two migrating flows.

Table I shows the parameters considered in DART compared to the related works.

Synthesis: From the exhaustive study of existing literature, it is evident that there exists a need for a consistent traffic flow migration scheme for SDN, which considers both rule-space capacity and link capacity constraints. Moreover, existing solution approaches ignore the diverse QoS demands of flows. An uncoordinated schedule causes link congestion and rule-space overflow. Therefore, in this work, we consider flow-specific demands to generate a congestion-free flow migration schedule for SDN. Additionally, we ensure the rule-space availability required for the installation of new flow-rules.

III. System Model

The system includes a set of network elements and a set of traffic flows.

1) Network Elements: As shown in Figure 2, SDN involves heterogeneous devices that transmit flows to switches via gateways. The rule-space of each switch is managed by an SDN controller. Let C and S denote the set of SDN controllers and the set of SDN switches, respectively. The switches are connected through data links and the controllers are connected through inter-controller links. Moreover, a switch connects with

TABLE I: Differences between DART and Related Works

Work	*Packet Consistency*	*Rule-Space Capacity*	*Link Capacity*	*QoS of Traffic Flows*	*Synchronized SDN*
Reitblatt *et al.* [12], McGeer *et al.* [13], Basta *et al.* [16]	✓	×	×	×	×
Mizrahi *et al.* [14], Lu *et al.* [15]	✓	×	×	×	✓
Ludwig *et al.* [17], Vissicchio and Cittadini [18]	✓	✓	×	×	×
Zheng *et al.* [19]	✓	×	✓	×	✓
Amiri *et al.* [20]	✓	×	✓	×	×
DART	✓	✓	✓	✓	×

a controller through a control link. At time t, the rule-space usage for switch $s_a \in S$ is represented as $R_a(t)$. In this work, we consider exact-match flow-rules, where a flow-rule is placed for each traffic flow [7]. In this work, we assume that the timeout value of a flow-rule is set to either a default value or a value based on the application type. Let R^{max} be the rule-space capacity of a switch. At time t, the bandwidth usage and capacity of the data link between s_a and s_b is denoted by $l_{ab}(t)$ and w_{ab}, respectively.

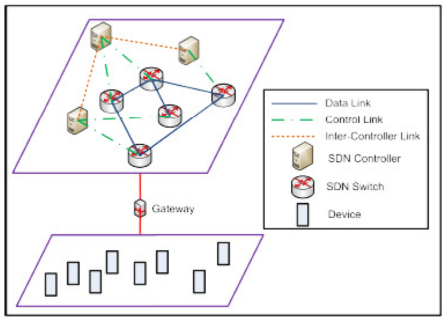


Fig. 2: SDN Architecture

2) Traffic Flows: Let F denote the set of traffic flows. A flow $f_j \in F$ is expressed by a tuple $<src(f_j), dest(f_j), bw(f_j), P(f_j), T_j^{max}>$, where $src(f_j)$ denotes the source, $dest(f_j)$ is the destination, $bw(f_j)$ is the bandwidth of f_j, $P(f_j)$ is the ordered set of switches along the path of f_j, and T_j^{max} is the maximum allowable delay for f_j. We define latency-sensitivity index (LSI) for f_j as:

$$\alpha(f_j) = \begin{cases} 0.5 + \frac{\overline{T} - T_j^{max}}{\overline{T}} & \text{if } T_j^{max} < \overline{T}, \\ 0.5 - \frac{T_j^{max} - \overline{T}}{T_j^{max}} & \text{if } T_j^{max} > \overline{T}, \\ 0.5 & \text{otherwise}, \end{cases} \tag{1}$$

where $\overline{T}$ is the average flow processing delay in the network. Equation (1) ensures that a traffic flow with a less maximum allowable delay has high LSI. Subsequently, we normalize the LSI to the $0-1$ range. Mathematically, $\alpha(f_j) = \frac{\alpha(f_j) - \alpha_{min}}{\alpha_{max} - \alpha_{min}}$, where α_{min} and α_{max} denote the minimum and the maximum LSI given by Equation (1).

$F^{'} \subset F$ denotes the set of to-be-migrated flows. A flow f_j is a to-be-migrated flow if $P(f_j) \neq P^{'}(f_j)$, where $P^{'}(f_j)$ is the new path of f_j after migration. In this work, we assume that the initial and final paths of the traffic flows are determined by the control plane. Additionally, we assume that the source and destination of a flow $f_j \in F^{'}$ are same in both initial and final paths.

***Definition* 1** (To-Be-Updated Switch). *A switch s_a is termed a to-be-updated switch for flow $f_j \in F^{'}$ if the new path of f_j includes at least one data link involving s_a that is not present in the old path.*

Let $S^{'}(f_j)$ denote the set of to-be-updated switches for $f_j \in F^{'}$. The migration of f_j involves the update of each switch $s_a \in S^{'}(f_j)$. For the migration of f_j, the controller sends update packets to all switches in the set $S^{'}(f_j)$. Therefore, the maximum rule update time required f_j is $T_{f_j} = \left(|S^{'}(f_j)| - 1\right)\Delta + \delta_{sc} + \sum_{s_a \in S^{'}(f_j)} \gamma(s_a)$, where Δ is the maximum time interval between dispatch of two successive update messages from the controller, δ_{sc} is the maximum controller-to-switch delay [14], and $\gamma(s_a)$ is the update time of s_a [21].

Let us consider that the network update procedure for traffic flow migration begins at time t_0. After t_0, a packet is termed ***old*** if it is processed by an old flow-rule. Otherwise, the packet is termed ***new***. Therefore, the migration of a traffic flow $f_j \in F$ is termed consistent when each old packet follows the old path only, and each new packet follows the new path only. We express consistent flow migration as:

$$\Psi(f_j) = \begin{cases} 1 & \text{if } f_j \text{ is consistent during migration}, \\ 0 & \text{otherwise}. \end{cases} \tag{2}$$

Let the flow migration process be divided into multiple update stages, and in each stage, single or multiple flows are migrated, based on the flow migration schedule. Let M be the total number of update stages. To express the flow migration schedule, we define a binary variable as:

$$\chi(f_j, m) = \begin{cases} 1 & \text{if } f_j \in F^{'} \text{ is migrated in stage } m, \\ 0 & \text{otherwise}. \end{cases} \tag{3}$$

***Definition* 2** (Correlated Flow). *Two flows f_i and f_j are correlated if at least one common link exists between the old (new) path of f_i and the new (old) path of f_j.*

The flows of an update stage are migrated in parallel if multiple controllers initiate them. Therefore, the completion time of a stage depends on the maximum rule update time of the flow that consumes the maximum time among all flows of the stage.

***Definition* 3** (Stage Completion Time). *The completion time of a stage m is $D_m = max\left(\chi(f_j, m)T_{f_j}\right), \forall f_j \in F^{'}$.*

***Definition* 4** (Flow Migration Duration). *The migration duration of each flow which is migrated in stage m is $D_m^R = (m^0 - t_0) + D_m$, where m^0 is the time when stage m starts.*

The objective of this work is to include the maximum number of latency-sensitive flows in each update stage, considering data plane load. Therefore, we formulate the load-aware flow migration problem (LFMP) as:

$$\underset{\chi}{\text{Maximize}} \sum_{f_j \in F^{'}} \chi(f_j, m)\alpha(f_j), \forall m \in [1, M] \tag{4}$$

subject to

$$\sum_{f_j \in F^{'}} \chi(f_j, m)\Psi(f_j) = \sum_{f_j \in F^{'}} \chi(f_j, m), \forall m \in [1, M], \tag{5}$$

$$R_a(t) \leq R^{max}, \quad \forall s_a \in S^{'}(f_j), \forall f_j \in F^{'}, \\ t \in [m^0, m^0 + D_m], m \leq M, \tag{6}$$

$$D_m^R \leq T_j^{max}, \chi(f_j, m) = 1, m \leq M, \forall f_j \in F^{'}, \tag{7}$$

$$\sum_{m=1}^{M} \chi(f_j, m) = 1, \forall f_j \in F^{'}, \tag{8}$$

$$l_{ab}(t) \leq w_{ab}, \forall s_a, s_b \in S, a \neq b, \\ t \in [m^0, m^0 + D_m], m \leq M \tag{9}$$

Equation (5) expresses the consistency constraint for traffic flows in each update stage. The consistency constraint ensures that each flow is consistent during migration. Equation (6) represents the rule-space capacity constraint of switches during the processing of each update stage. Equation (7) ensures that the migration duration of each migrating traffic flow f_j is within the maximum allowable delay T_j^{max} of the flow. Equation (8) ensures that each flow is migrated only once. Equation (9) denotes the link capacity constraint during the processing of each update stage.

Theorem 1. *LFMP is NP-hard.*

Proof: To prove the NP-hardness of LFMP, we reduce LFMP to the well-known $0-1$ multidimensional knapsack problem [22]. The $0-1$ $h-$dimensional knapsack problem, which is an NP-hard problem, involves a set of items so that each item has a $h-$dimensional weight vector and a value. The goal is to maximize the total value of items included in the knapsack, given a knapsack with a fixed $h-$dimensional capacity vector. The decision to include an item in a knapsack is binary, i.e., an item is added to the knapsack as a whole or not added. We construct an instance I of the LFMP for an update stage m. We reduce I to an instance $I^{'}$ of the $0-1$ two-dimensional knapsack problem. Each flow f_j in I refers each item in $I^{'}$. The vector $(bw(f_j), R_a(t)), \forall s_a \in S^{'}(f_j)$ corresponds to weight of each item. LSI $\alpha(f_j)$ correspond to the value of each item. The vector (w_{ab}, R^{max}) is mapped to the capacity of the knapsack. In this case, the value of the decision variable $\chi(f_j, m)$ is restricted to 0 or 1, depending on whether f_j is migrated in stage m or not. The goal of I is to find a feasible solution that includes the maximum number of flows with high LSI in each update stage without violating the link capacity and rule-space capacity constraints for any flow. Therefore, the optimal solution of I is also the optimal solution to the instance of the $0-1$ knapsack problem $I^{'}$. Hence, the LFMP is also NP-hard. ∎

As the optimization problem in Equation (4) is NP-hard, we propose a heuristic approach for solving LFMP.

IV. DART: The Proposed Scheme

In this section, we present the proposed scheme, DART, which has three modules — 1) generation of QoS-aware migration schedule, 2) generation of feasible migration schedule, and 3) rule-space management. The QoS-aware migration scheduling module analyzes the QoS demand of each migrating flow and generates an initial flow migration schedule. The feasible migration scheduling module evaluates whether the initial flow migration schedule is feasible or not and updates the schedule to avoid link congestion. The rule-space management module checks the available rule-space in each to-be-updated switch and frees up rule-space as per the requirement. Lightly loaded controllers can execute the first two modules related to the generation of migration schedules in the control plane. For each switch, the corresponding master controller executes the rule-space management module.

A. Generation of QoS-Aware Migration Schedule

We formulate a coalition graph game to form groups of flows so that each group is migrated in each update stage. In this game, $F^{'}$ is the set of players. Each coalition $A_k \in F^{'}$ denotes the set of flows $\{f_1, f_2, \ldots, f_{|A_k|}\}$ which are migrated in an update stage. Within a coalition, the flow with the highest LSI is termed as the coalition-head. Therefore, a coalition-head has $|A_k|-1$ children nodes, which are termed as coalition members. To form the coalitions, the proposed game constructs a coalition graph $G=(F^{'}, E)$, where E is the set of edges representing the correlation between flows as defined in Definition 2. So, there exists an edge between $f_i \in F^{'}$ and $f_j \in F^{'}$ if f_i and f_j are correlated flows.

1) Suitability of the Coalition Graph Game for QoS-Aware Migration Scheduling: The coalition game applies a fair and cooperative strategy that is beneficial to all players. Moreover, coalition game theory reduces computation overhead associated with large-scale networks because the coalitions are formed in a distributed manner. On the other hand, compared to greedy algorithms, in a coalition game-based approach, the probability of getting stuck at local optima is low because the stable coalition is formed based on multiple applications of the merge-and-split rules. The computation time in the coalition game-based approach is significantly less than the ILP solution generated by a solver. Hence, coalition game is widely used for cooperative communication systems to serve several objectives, including power-aware routing, collaborative task sensing, and relay transmission [23].

Motivated by the aforementioned advantages of coalition game, in this work, we use coalition game theory to migrate the traffic flows, which cooperatively decide the optimum strategy to satisfy their QoS demands and achieve Pareto optimal distribution of link capacity. Moreover, the correlation between flows serves as a critical aspect for forming the groups as the update of one flow may cause link congestion in the flow-path of a correlated flow. Accordingly, we formulate a coalition graph game to form a QoS-aware flow migration schedule, where migrating flows form cooperative groups, which are migrated simultaneously in an update stage for optimal utilization of the available link capacity.

***Definition* 5** (Coalition Structure). *A coalition structure is a set of coalitions $V_A = \{A_1, A_2, \ldots, A_M\}$, where $\bigcup_{k=1}^{M} A_k = F^{'}, A_k \cap A_l = \phi, \forall k \neq l$.*

2) Utility Function of a Coalition: The controllers aim to maximize the cumulative payoff obtained from the utility functions of the coalitions. Let $U(A_k, V_A)$ denote the utility of a coalition $A_k \in V_A$ and $u_j(.)$ denote the utility of a player $f_j \in A_k$. The marginal utility of each flow f_j increases with decrease in the maximum rule update time of the flow. Mathematically, $\frac{\partial u_j(.)}{\partial T_{f_j}} < 0$. The utility function $u_j(.)$ varies linearly with the LSI, and the number of correlated flows in the coalition (N_j) so that a high number of flows are migrated in an update stage depending on their traffic characteristics. Therefore, we get $\frac{\partial u_j(.)}{\partial \alpha(f_j)} > 0$, and $\frac{\partial u_j(.)}{\partial N_j} > 0$. Therefore, we define the utility of a flow f_j as:

$$u_j(.) = N_j \left(\alpha(f_j) - \frac{T_{f_j}}{T_j^{max}} \right) \tag{10}$$

Hence, the utility function $U(A_k, V_A)$ is formulated as:

$$U(A_k, V_A) = \begin{cases} \sum\limits_{f_j \in A_k} u_j(.) & \text{if } |A_k| > 1, \\ 0 & \text{otherwise}. \end{cases} \tag{11}$$

The total utility of the coalitions in V_A is:

$$U(V_A) = \sum_{k=1}^{M} U(A_k, V_A) \tag{12}$$

Equation (10) and Equation (11) ensure that the utility of a coalition increases as more latency-sensitive flows with less maximum rule update time and more number of correlated flows are included in the coalition. Therefore, selecting a coalition with high utility at an earlier update stage increases the possibility that latency-sensitive flows are migrated without violating the QoS requirements.

3) Coalition Graph Formation: The to-be-migrated traffic flows, which are the players of the coalition graph game, form the coalition graph based on the utility function defined in Equation (12). We consider that the proposed coalition graph game is hedonic, which implies that a player has a preference for the choice of the coalition.

***Definition* 6** (Preference Relation). *The relation $V_A \succ_{F^{''}} V_B$*

denotes that the way V_A partitions $F^{''}$ is preferred to the way V_B partitions $F^{''}$, where $F^{''} \subseteq F^{'}$ is a set of players.

In this work, we consider Pareto order [24] as the basis for the preference relation $\succ$. According to Pareto order, a coalition structure V_A is preferred over another V_B if the change of coalition structure from V_B to V_A improves utility for at least one player without decreasing the utility of any other player. Let $u_j(A)$ denote the utility of player f_j which is a member of coalition $A_k \in V_A$. Mathematically,

$$V_A \succ_{F^{''}} V_B \Leftrightarrow \{u_j(A) \geq u_j(B)\}, \forall f_j \in F^{''},$$
$$F^{''} = \bigcup_{k=1}^{|V_A \cup V_B|} A_k, \forall A_k \in V_A \cup V_B, \quad (13)$$

with at least one player f_x having the strict inequality $u_x(A) > u_x(B)$.

The coalitions are updated incrementally based on merge and split rules as follows:

***Definition* 7** (Merge Rule). *Merge any set of coalitions $\{A_1, A_2, \ldots, A_k\}$ where $\{\bigcup_{l=1}^{k} A_l\} \succ_{F^{''}} \{A_1, A_2, \ldots, A_k\}$, $F^{''} = \bigcup_{l=1}^{k} A_i$. Therefore, $\{A_1, A_2, \ldots, A_k\} \rightarrow \bigcup_{l=1}^{k} A_l$.*

***Definition* 8** (Split Rule). *Split any set of coalitions $\bigcup_{i=1}^{k} A_l$ where $\{A_1, A_2, \ldots, A_k\} \succ_{F^{''}} \{\bigcup_{l=1}^{k} A_l\}$, $F^{''} = \bigcup_{l=1}^{k} A_l$. Therefore, $\bigcup_{l=1}^{k} A_l \rightarrow \{A_1, A_2, \ldots, A_k\}$.*

Therefore, multiple coalitions merge into a large coalition if merging is preferable to the set of players according to Equation (13). Similarly, one large coalition splits into multiple coalitions if splitting is beneficial to the set of players. To restrict the search space for the merge operation, we consider a greedy approach to decide the potential candidates for the attempt of merging. In this approach, a coalition A_l attempts to merge with coalition A_k only if there exists at least one edge $e_{ij} \in E$ between $f_i \in A_l$ and $f_j \in A_k$. This constraint ensures that the merged utility is always positive.

***Definition* 9** (Stable Coalition). *A coalition $A_k \in V_A$ is stable if*

1) *no player f_j can improve its utility by leaving its coalition A_k and acting individually.*
2) *no other coalition $A_l \in V_A$ can improve its utility by joining A_k.*

***Definition* 10** (Stable Coalition Structure). *A coalition structure V_A is stable if $A_k \in V_A, \forall k \in [1, M]$ is stable.*

Algorithm 1 describes the generation of the initial migration schedule. Initially, the flows form singleton coalitions. The Initial Migration Scheduling Algorithm (IMSA) sorts the coalitions in descending order based on the LSI values of the coalition-heads. In each iteration, each coalition A_k forms a potential candidate list L_k. The list L_k is sorted based on the LSI values of the coalition-heads. A_k attempts to merge with the first coalition in L_k. If the merge attempt is successful, both coalitions are merged. Otherwise, A_k attempts to merge with the next coalition in the list. This merge process can be performed distributively, where each coalition makes a greedy attempt to merge with the coalitions in its potential candidate list. After completing greedy merge attempts for all coalitions, the split operation is performed, if any split is possible. The merge and split process is repeated until V_A is stable. The initial migration schedule $\chi^{'}$ is formed by scheduling the flows of each coalition from v_A in each update stage.

The time complexity of IMSA depends on the number of merge and split attempts. For $|F^{'}|$ flows, the maximum number of possible coalitions is $|F^{'}|$. In the worst case, each coalition attempts to merge with all the others. In this case, the first coalition makes $|F^{'}| - 1$ merge attempts, the second coalition requires $|F^{'}| - 2$ merge attempts, and so on. Therefore, the maximum number of merge attempts is $\frac{|F^{'}|(|F^{'}|-1)}{2}$. However, in a practical scenario, the number of merge attempts is significantly less as each coalition attempts to merge only with coalitions in the potential candidate list. In the worst case, the split operation of a coalition involves finding all partitions of the coalition. The total number of partitions is given by the Bell number [25], which grows exponentially with the number of players in the coalition. However, in a practical scenario, once a coalition splits based on the Pareto order as stated in Equation (13), no further split is attempted. Therefore, the total number of split attempts is significantly less in practice.

Algorithm 1 Initial Migration Scheduling Algorithm (IMSA)

```
INPUT: $F^{'}$                                  ▷ Set of migrating flows
OUTPUT: $\chi^{'}$                              ▷ Initial migration schedule
PROCEDURE:
 1: $E \leftarrow E \cup \{e_{ij}\}$ if $f_i \in F^{'}$ and $f_j \in F^{'}$ are correlated flows
 2: $A_k \leftarrow A_k \cup \{f_k\}, V_A \leftarrow V_A \cup \{A_k\}, \forall f_k \in F^{'}$
 3: do
 4:     Sort $V_A$ in descending order of the LSI values of the coalition-heads
 5:     for all $A_k \in V_A$ do
 6:         Form potential candidate list $L_k$ using $E$
 7:         Sort the coalitions in $L_k$ in descending order of the LSI values of the coalition-heads
 8:         if merge attempt successful for $A_l \in L_k$ then
 9:             Merge $A_k$ and $A_l$ using Definition 7 and Update $V_A$
10:         else
11:             Attempt merge with $A_{l+1} \in L_k$
12:         end if
13:     end for
14:     Split coalitions in $V_A$ using Definition 8 and Update $V_A$
15: while $V_A$ is not stable
16: Set $\chi^{'}(f_j, k) = 1, \forall f_j \in A_k, \forall A_k \in V_A$
17: return $\chi^{'}$
```

Theorem 2. *IMSA converges to a stable coalition structure.*

Proof: Initially, each player forms an individual coalition having zero utility. Therefore, a player has the lowest utility value when it acts individually. In subsequent iterations, each player tries to increase its utility via the merge and split operations. This process continues if at least one player is capable of improving its utility by joining another coalition and a new coalition structure is formed. However, the number of partitions of a set with a finite number of elements is finite [25]. Therefore, the number of coalition structures generated by IMSA is finite as $F^{'}$ is a finite set, and IMSA reaches a final coalition structure. Moreover, the termination of the merge and split process implies that no coalition can improve its utility by joining another coalition. Therefore, IMSA generates a stable coalition structure, V_A. ■

B. Generation of Feasible Migration Schedule

The coalitions from the stable coalition structure V_A are selected one-by-one for consistent flow migration, and only one coalition is migrated in each update stage. However, the migration of a flow may trigger congestion in one or multiple links. This is because those links have to-be-migrated flows which are scheduled to be migrated in a later stage. Therefore, prospective link congestion makes a flow migration schedule infeasible. Therefore, we propose a greedy heuristic algorithm

Algorithm 2 Feasible Migration Scheduling Algorithm (FMSA)

```
INPUTS: $\chi^{'}$, $V_A$
OUTPUT: $\chi$                ▷ Feasible migration schedule
PROCEDURE:
 1: while $m \neq |V_A|$ do
 2:     for all $f_j \in F^{'}$ do
 3:         if $\chi^{'}(f_j, m) = 1$, $D^R_{m+1} \leq T^{max}_j$, and migration of $f_j$ violates link capacity constraint then
 4:             Set $\chi(f_j, m+1) = 1$ and update $V_A$
 5:         end if
 6:     end for
 7: end while
 8: return $\chi$
```

to analyze the feasibility of the initial migration schedule and prepare the final migration schedule that reduces the data link load. Algorithm 2 shows the steps for the generation of a feasible flow migration schedule.

Each iteration of the Feasible Migration Scheduling Algorithm (FMSA) checks the initial migration schedule and determines whether the migration of the flows in a stage is feasible in terms of the link capacity constraint. If any flow violates the link capacity constraint, FMSA moves the infeasible flow to the next update stage. A flow f_j belonging to stage m is moved to the next stage $m+1$ only if the flow migration duration for stage $m+1$ does not exceed the maximum allowable delay T^{max}_j of the flow f_j. As we migrate the flows in each update stage together, the possibility of link congestion reduces for some links, and some infeasible flows become feasible. Therefore, FMSA takes a greedy approach to allocate the infeasible flows to the nearest update stage. FMSA runs in $O(|F^{'}|)$ time as each flow in an update stage checks for link capacity violation based on the bandwidth usage data of the links, which is available to the controller.

C. Rule-Space Management

FMSA generates the final migration schedule, which reduces link congestion during flow migration. However, another part of the data plane load is rule-space usage. SDN switches have limited rule-space, and the overflow of rule-space makes the migration process inconsistent and incomplete. However, in each stage, we update the switches based on the approach proposed in our earlier work, CURE [26]. This approach deletes old rules immediately after installing new flow-rules. Therefore, the switches, which are part of both old and new paths of a flow, require no additional rule-space. However, the switches, which only belong to the new path, require the installation of additional flow-rules to define the new path. So, we propose a heuristic algorithm to ensure that these switches have enough capacity to address the additional rule-space requirement.

The proposed rule-space management process requires the deletion of unimportant flow-rules from the switches, which have low free rule-space. To select the rules that are no longer required, we estimate the popularity of the rules stored in the rule-space of a switch. We sort the rules of the corresponding switch in descending order of the received packet count. For s_a, the rule popularity is denoted by $\Theta = \{\theta_1, \theta_2, \theta_3, \ldots, \theta_{R_a(t)}\}$, where θ_k is the probability that a flow matches with the k^{th} rule. We estimate the rule popularity based on Zipf distribution [27], which is $\theta_k = \frac{\frac{1}{k^\epsilon}}{\sum\limits_{b=1}^{R_a(t)} \frac{1}{b^\epsilon}}$, where ϵ is the skewness of the rule popularity. The value $\epsilon = 0$ denotes uniform popularity distribution and a larger ϵ signifies more uneven rule popularity.

Algorithm 3 shows the steps of the rule-space management process based on the feasible flow migration schedule. The Rule-space Management Algorithm (RSMA) identifies the set of switches $S^{''}$, requiring the installation of additional flow-rules. Additionally, RSMA estimates the rule-space requirement for each switch in $S^{''}$. To identify the overloaded switches, RSMA checks if the available rule-space for any switch in $S^{''}$ is less than the additional rule-space required for migration. Finally, RSMA frees the required rule-space in the overloaded switches by deleting the required number of rules starting with the least popular rule with remaining timeout greater than a pre-defined threshold ζ. The value of ζ is fixed at a high value if the traffic load of the network is high and new flow-rules are frequently installed. The time complexity of RSMA is composed of two parts — the time complexity for the formation of $S^{''}$ and the time complexity for the reduction of rule-space usage in the overloaded switches. Each flow is visited to identify the set of switches for inclusion in $S^{''}$. This operation is completed in $O(|F^{'}|)$ time. The rule-space reduction process takes $O(|S|)$ time because, in the worst case, the reduction must be performed for all switches. Therefore, RSMA run in $O(|F^{'}| + |S|)$ time.

Algorithm 3 Rule-Space Management Algorithm (RSMA)

```
INPUTS: $\chi$, $\lambda$
OUTPUT: $S^{''}$
PROCEDURE:
 1: while $m \neq |V_A|$ do
 2:     for all $f_j \in F^{'}$ do
 3:         if $\chi(f_j, m) = 1$ then
 4:             $S^{''} \leftarrow S^{''} \cup \Big(P^{'}(f_j) \setminus P(f_j)\Big)$
 5:             $addRules(s_a) \leftarrow addRules(s_a) + 1, \forall s_a \in \Big(P^{'}(f_j) \setminus P(f_j)\Big)$    ▷ Additional rule-space required for migration
 6:         end if
 7:     end for
 8: end while
 9: for all $s_a \in S^{''}$ do
10:     if $R^{max} - R_a(t) < addRules(s_a)$ then
11:         Delete $addRules(s_a) - (R^{max} - R_a(t))$ less popular rules with remaining timeout greater than $\zeta$
12:     end if
13: end for
14: return $S^{''}$
```

D. Consistent Flow Migration

The set of to-be-updated switches for update stage m, is $S_m = \bigcup_{j=1}^{|F^{'}|} S^{'}(f_j)$, where $\chi(f_j, m) = 1$. For consistent flow migration, in each update stage m, DART processes the old packets and starts buffering the new packets at the switches in S_m. This step ensures packet consistency. After processing all the old packets, new flow-rules are installed, and old flow-rules are deleted. This step addresses the rule-space capacity constraint of the SDN switches as only a single version of a rule is installed at a time. After the installation of all the required flow-rules, DART processes the buffered packets [26]. A to-be-updated switch $s_a \in S_m$ uses buffer of a suitable neighbor switch to store the new packets if the buffer of s_a is full. In this case, s_a receives queued packets from the neighbor switch after stage m is complete [26]. For $s_a \in S_m$, the increase in packet queueing delay due to flow migration is $q_a = \frac{1}{\mu}\left(-\frac{1+Q\rho^{Q+1}-(Q+1)\rho^{Q}}{(1-\rho)(1-\rho^{Q+1})} + \frac{1+Q(\rho^{'})^{Q+1}-(Q+1)(\rho^{'})^{Q}}{(1-\rho^{'})(1-(\rho^{'})^{Q+1})}\right)$, where μ denotes the service rate at the switch, Q is the size of the switch buffer, ρ is the traffic intensity at the switch before stage m starts, and $\rho^{'}$ is the traffic intensity at the switch after stage m completes [26].

Theorem 3. *Flow migration in DART is blackhole free.*

Proof: Let $f_j \in F^{'}$ be a flow that is scheduled to be migrated in stage m. In stage m, new flow-rules are installed in all switches in $S^{'}(f_j)$. However, the old packets are processed by the old flow-rules before updating the first switch in stage m. As the update of the first switch in stage m starts, the new packets are queued until all switches in stage m complete update. Once stage m completes update, the queued packets are handled by the new flow-rules. Therefore, all packets that enter a switch belonging to the old path $P(f_j)$ or to the new path $P^{'}(f_j)$ is equal to the packets that leave the switch. Since, no packet of a flow f_j is dropped, the flow migration process in DART is blackhole free. ■

Theorem 4. *Flow migration in DART is loop free.*

Proof: All the old packets of a flow $f_j \in F^{'}$ are processed by old rules entirely. New rules are installed to all switches in $S^{'}(f_j)$ before processing the new packets. Therefore, each packet in f_j either follows the old path $P(f_j)$ or the new path $P^{'}(f_j)$. Since, no packet is processed by incorrect flow-rules, the flow migration in DART is loop free. ■

V. Performance Evaluation

A. Simulation Settings

We evaluate the performance of DART by implementing a mesh network with 12 switches using OMNeT++. The source and destination of each traffic flow are generated randomly and the packets in each flow follow Poisson's distribution. Additionally, we set the packet arrival rate per switch as $0.005-0.025$ million packets per second (mpps) [26]. We set the bandwidth of each traffic flow to 0.0001 – 0.39 Gbps [28]. The maximum link capacity is set to 9.92 Gbps [28]. For the simulation, we consider that 80% flows are latency-sensitive with the maximum allowable delay $1-200$ ms [29]. Table II shows the simulation parameters.

B. Benchmark Schemes

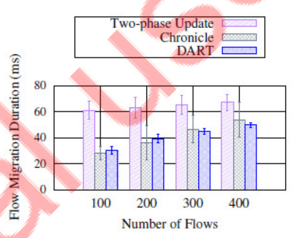
We compare the performance of DART with two benchmark schemes — two-phase update [12] and Chronicle [19]. The two-phase update is not incremental and schedules all traffic flows simultaneously for migration. The two-phase update scheme updates the ingress switches after updating the internal switches. Chronicle partitions each migrating flow into update blocks and the first switch in each update block is updated. We select the two-phase update as one of the benchmark schemes to show the effectiveness of incremental flow migration. We select Chronicle to show the importance of considering flow-specific QoS requirements along with the link capacity constraint.

C. Performance Metrics

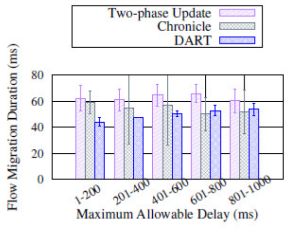
- Flow migration duration: The migration duration of a traffic flow is defined in Definition (4). This metric quantifies the time required for a flow to change its path from old to new.
- Maximum data link bandwidth usage: This metric shows the maximum bandwidth usage of the data links during flow migration. A high bandwidth usage signifies that the possibility of link congestion is high.
- Rule-space usage for flow migration: This metric measures the rule-space required for the flow migration process. This metric is important because of the rule-space capacity limitation of SDN switches.
- QoS violated flows: QoS violated flows are flows that have a migration duration greater than the maximum allowable delay. This metric shows the QoS-awareness of DART.
- Packet queueing delay: DART queues the new packets during an ongoing flow migration. We measure the average packet queueing delay to quantify the increased latency for flow processing.

TABLE II: Simulation parameters

Parameter	Value
Number of traffic flows	$20-400$
Bandwidth of a traffic flow	0.0001 – 0.39 Gbps [28]
Maximum link capacity	9.92 Gbps [28]
Number of switches	12
Maximum controller-to-switch delay (δ_{sc})	4.87 ms [26]
Maximum time interval between dispatch of successive update messages from the controller (Δ)	5.24 ms [26]
Average packet arrival rate	0.005 – 0.025 mpps [26]
Maximum allowable delay	$1-1000$ ms [29]
Rule popularity skewness (ϵ)	0.56
Rule-space capacity (R^{max})	500 flow-rules



(a) Effect of Traffic Load



(b) Effect of Latency-Sensitivity

Fig. 3: Flow Migration Duration

D. Result and Discussion

1) Flow Migration Duration: We estimate the average migration duration by varying the number of flows. From Figure 3a, we observe that the average flow migration duration for DART with 400 flows is 26.02% less than that of two-phase update. According to Definition 3, the average flow migration duration depends on the flow that consumes the maximum time among all flows of an update stage. DART achieves better performance because and DART migrates the flows incrementally in multiple update stages and places time-consuming flows in later update stages so that the latency-sensitive flows are not affected. However, two-phase update migrates all flows at a time and the migration process completes after all flows are migrated. Moreover, we observe that the average flow migration duration of Chronicle is similar to that of DART. This is because the Chronicle is a time-triggered approach where controllers send update messages in advance and the to-be-updated switches are updated at the time scheduled for the corresponding update block.

Figure 3b portrays the effects of LSI on the flow migration duration for 400 traffic flows. For this experiment, we form 5 groups, each having 80 flows. The maximum allowable delay of the flows in the group 1, group 2, group 3, group 4, and group 5 are $[1, 200]$ ms, $[201, 400]$ ms, $[401, 600]$ ms, $[601, 800]$ ms, and $[800, 1000]$ ms, respectively. We observe that the average flow migration duration for both DART decreases as the latency-sensitivity of migrating flows increases. In particular, for DART,

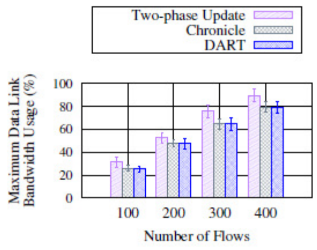


Fig. 4: Maximum Data Link Bandwidth Usage

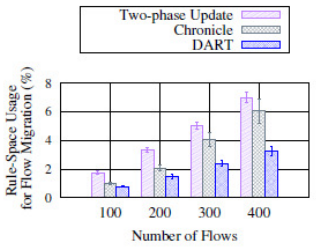


Fig. 5: Rule-Space Usage for Flow Migration

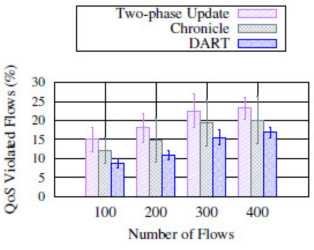


Fig. 6: QoS Violated Flows

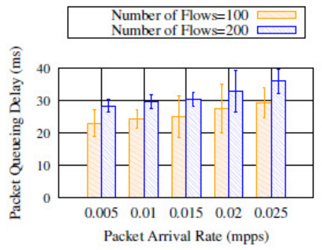


Fig. 7: Packet Queueing Delay for DART

the average migration duration of the flows in the group 1 is 18.43% less than the flows in the group 5. However, the variation of latency-sensitivity does not affect the migration duration of benchmark schemes. Therefore, it is evident that DART prioritizes latency-sensitive flows and schedules their migration earlier to satisfy the QoS demands.

2) Maximum Data Link Bandwidth Usage: We analyze the maximum data link bandwidth usage as it is the primary contributor to the data plane load. Figure 4 sketches the maximum data link bandwidth usage with varying number of flows. We observe that the maximum data link bandwidth usage is less for both DART and Chronicle because both schemes reduce the possibility of link congestion. As mentioned in FMSA, DART prepares the feasible flow migration schedule considering the link capacity constraint. Therefore, for high traffic load, DART proves to be a reliable scheme that reduces data loss caused by link congestion.

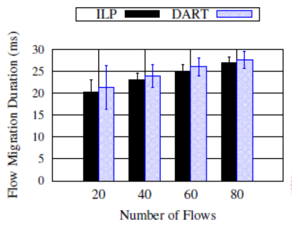
3) Rule-Space Usage for Flow Migration: For DART and the benchmark schemes, we estimate the additional rule-space requirement because of the flow migration process. Figure 5 shows the average rule-space usage with varying traffic load. We observe that DART uses 53.85% and 40.18% less rule-space as compared to the two-phase update and Chronicle, respectively. This is due to the fact that RSMA deletes the required number of less popular rules to accommodate new flow-rules, and DART performs consistent flow migration where old flow-rules are not stored redundantly.

4) QoS Violated Flows: We analyze the amount of QoS violation considering heterogeneous traffic where each flow f_j has a different QoS requirement in terms of the maximum allowable delay T_j^{max}. From Figure 6, we observe that QoS violation in DART is 33.89% less than the same using the two-phase update and 21.69% less than Chronicle. This is because DART migrates the traffic flows in order of the LSI values so that each flow satisfies the QoS demand. In Addition, DART considers the link capacity constraint and schedules feasible flows together.

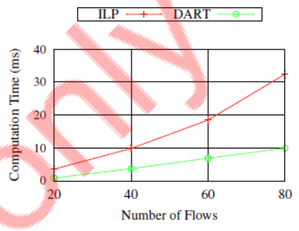
5) Packet Queueing Delay: DART preserves packet consistency by buffering new packets arriving at the to-be-updated switches until the corresponding update stage completes. Figure 7 shows the average packet queueing delay for DART with varying packet arrival rate. The simulation result implies that although DART queues packets for ensuring packet consistency, the packets are processed with a consistent rate.

From the above analysis, it is evident that the proposed scheme, DART, significantly reduces the peak load of the data links and additional rule-space usage for flow migration with acceptable flow migration duration. Additionally, it is noteworthy to observe that DART achieves remarkable performance in terms of addressing QoS demands of heterogeneous flows considering heterogeneous traffic as an essential parameter of realistic networks.

We solve the ILP formulated in Equation (4) using Gurobi Optimizer [30]. Figure 8 shows the comparison between the ILP solution and the proposed heuristic approach, DART. We observe that DART achieves performance similar to the ILP solution while having low computation time. In particular, for 80 flows, the average flow migration duration for DART is 2.37% more compared to the ILP solution. This is because DART solves the LFMP by forming coalitions based on the utility function expressed in Equation (12) which aims to include the maximum number of latency-sensitive flows in each coalition or each update stage.



(a) Flow Migration Duration



(b) Computation Time

Fig. 8: Comparison between ILP Solution and DART

VI. Conclusion

In this paper, a traffic-aware flow-migration approach for data plane load reduction in SDN was presented. The proposed scheme, named DART, migrates traffic flows in different update stages. Each update stage is formed based on the QoS demand of the flows, and bandwidth usage of the links. DART also addresses the rule-space capacity constraint so that no switch reaches its rule-space capacity limit as a result of a flow migration. We performed a detailed analysis of the performance of DART, considering traffic load and QoS demand. Simulation results show that DART reduces the additional rule-space usage by 53.85%, and QoS violation by 33.89% compared to the two-phase update. To summarize, DART is computationally efficient and is capable of addressing the challenges of the next generation networks, which require QoS-aware processing with acceptable performance.

In the future, we plan to extend this work considering compressed rules. We also plan to study the impact of network disruptions such as link failure and traffic spike. Additionally, the future extension of this work includes the reduction of packet queueing delay and controller overhead.

Acknowledgement

A preliminary version of this work is published in IEEE ICC 2020, Dublin, Ireland (June 7-11, 2020), DOI: 10.1109/ICC40277.2020.9148983.

References

[1] A. Mondal, S. Misra, and I. Maity, “Buffer Size Evaluation of OpenFlow Systems in Software-Defined Networks,” *IEEE Syst. J.*, vol. 13, no. 2, pp. 1359–1366, Jun. 2019.

[2] S. Bera, S. Misra, and N. Saha, "Traffic-Aware Dynamic Controller Assignment in SDN," *IEEE Trans. Commun.*, vol. 68, no. 7, pp. 4375–4382, 2020.

[3] M. Gharbieh, H. ElSawy, A. Bader, and M. Alouini, "Spatiotemporal Stochastic Modeling of IoT Enabled Cellular Networks: Scalability and Stability Analysis," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3585–3600, Aug. 2017.

[4] K. Foerster, L. Vanbever, and R. Wattenhofer, "Latency and Consistent Flow Migration: Relax for Lossless Updates," in *IFIP Networking Conference*, 2019, pp. 1–9.

[5] N. Saha, S. Misra, and S. Bera, "QoS-Aware Adaptive Flow-Rule Aggregation in Software-Defined IoT," in *Proc. IEEE GLOBECOM*, 2018, pp. 206–212.

[6] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "PIAS: Practical Information-Agnostic Flow Scheduling for Commodity Data Centers," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 1954–1967, 2017.

[7] OpenFlow Switch Specification Version 1.5.1, *Open Networking Foundation*, Accessed: May, 2020. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[8] I. Maity, A. Mondal, S. Misra, and C. Mandal, "Tensor-Based Rule-Space Management System in SDN," *IEEE Syst. J.*, vol. 13, no. 4, pp. 3921–3928, Dec. 2019.

[9] F. Yaghoubi, M. Furdek, A. Rostami, P. Öhlén, and L. Wosinska, "Consistency-Aware Weather Disruption-Tolerant Routing in SDN-Based Wireless Mesh Networks," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 2, pp. 582–595, Jun. 2018.

[10] M. T. I. Ul Huque, G. Jourjon, C. Russell, and V. Gramoli, "Software Defined Network's Garbage Collection With Clean-Up Packets," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 4, pp. 1595–1608, Dec. 2019.

[11] N. Bogdanović, D. Ampeliotis, and K. Berberidis, "A Coalitional Game Theoretic Outlook on Distributed Adaptive Parameter Estimation," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 3, no. 2, pp. 416–429, Jun. 2017.

[12] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for Network Update," in *Proc. ACM SIGCOMM*, New York, NY, USA, 2012, pp. 323–334.

[13] R. McGeer, "A Safe, Efficient Update Protocol for Openflow Networks," in *Proc. HOT SDN*, New York, NY, USA, 2012, pp. 61–66.

[14] T. Mizrahi, E. Saat, and Y. Moses, "Timed Consistent Network Updates in Software-Defined Networks," *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3412–3425, Dec. 2016.

[15] J. Lu, H. Xiong, F. He, Z. Zheng, and H. Li, "A Mixed-Critical Consistent Update Algorithm in Software Defined Time-Triggered Ethernet Using Time Window," *IEEE Access*, vol. 8, pp. 65 554–65 565, 2020.

[16] A. Basta, A. Blenk, S. Dudycz, A. Ludwig, and S. Schmid, "Efficient Loop-Free Rerouting of Multiple SDN Flows," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 948–961, Apr. 2018.

[17] A. Ludwig, J. Marcinkowski, and S. Schmid, "Scheduling Loop-Free Network Updates: It's Good to Relax!" in *Proc. ACM PODC*, 2015, p. 13–22.

[18] S. Vissicchio and L. Cittadini, "FLIP the (Flow) table: Fast lightweight policy-preserving SDN updates," in *IEEE INFOCOM*, 2016, pp. 1–9.

[19] J. Zheng, B. Li, C. Tian, K. Foerster, S. Schmid, G. Chen, J. Wu, and R. Li, "Congestion-Free Rerouting of Multiple Flows in Timed SDNs," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 968–981, 2019.

[20] S. A. Amiri, S. Dudycz, M. Parham, S. Schmid, and S. Wiederrecht, "On Polynomial-Time Congestion-Free Software-Defined Network Updates," in *IFIP Networking Conference*, 2019, pp. 1–9.

[21] X. Wen, B. Yang, Y. Chen, L. E. Li, K. Bu, P. Zheng, Y. Yang, and C. Hu, "RuleTris: Minimizing Rule Update Latency for TCAM-Based SDN Switches," in *Proc. IEEE ICDCS*, 2016, pp. 179–188.

[22] A. Dalgkitsis, P. Mekikis, A. Antonopoulos, and C. Verikoukis, "Data Driven Service Orchestration for Vehicular Networks," *IEEE Trans. Intell. Transp. Syst.*, pp. 1–10, 2020, doi: 10.1109/TITS.2020.3011264.

[23] M. W. Baidas and A. B. MacKenzie, "Altruistic Coalition Formation in Cooperative Wireless Networks," *IEEE Trans. Commun.*, vol. 61, no. 11, pp. 4678–4689, 2013.

[24] W. Saad, Z. Han, M. Debbah, and A. Hjorungnes, "A Distributed Coalition Formation Framework for Fair User Cooperation in Wireless Networks," *IEEE Trans. Wireless Commun.*, vol. 8, no. 9, pp. 4580–4593, Sep. 2009.

[25] M. Ahmed, M. Peng, M. Abana, S. Yan, and C. Wang, "Interference Coordination in Heterogeneous Small-Cell Networks: A Coalition Formation Game Approach," *IEEE Syst. J.*, vol. 12, no. 1, pp. 604–615, Mar. 2018.

[26] I. Maity, A. Mondal, S. Misra, and C. Mandal, "CURE: Consistent Update With Redundancy Reduction in SDN," *IEEE Trans. Commun.*, vol. 66, no. 9, pp. 3974–3981, Sep. 2018.

[27] A. F. Tayel, S. I. Rabia, and Y. Abouelseoud, "An Optimized Hybrid Approach for Spectrum Handoff in Cognitive Radio Networks With Non-Identical Channels," *IEEE Trans. Commun.*, vol. 64, no. 11, pp. 4487–4496, Nov. 2016.

[28] Abilene Dataset, Accessed: May, 2020. [Online]. Available: http://www.cs.utexas.edu/ yzhang/research/AbileneTM

[29] S. F. Abedin, M. G. R. Alam, S. M. A. Kazmi, N. H. Tran, D. Niyato, and C. S. Hong, "Resource Allocation for Ultra-Reliable and Enhanced Mobile Broadband IoT Applications in Fog Network," *IEEE Trans. Commun.*, vol. 67, no. 1, pp. 489–502, Jan. 2019.

[30] Gurobi Optimizer, *Gurobi Optimizer Reference Manual*, Accessed: May, 2020. [Online]. Available: http://www.gurobi.com