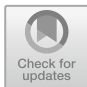

Enhanced Relaxed Loop Free Updates in Software Defined Network

Bahri Jamal(✉), Marouane Sebgui, and Zouhair Guennoun

Equipe de Recherche en Smart Communications – ERSC, Ecole Mohammadia d'Ingénieurs – EMI, Mohammed V University, Rabat, Morocco
jamal.bahri@yahoo.fr, {sebgui,zouhair}@emi.ac.ma

Abstract. Considering the general problem of updating switches in a software defined network architecture, temporary loops can occur in the case of inappropriate scheduling of node updating order. This problem and others like a black hole also called consistency problems must be avoided. This work focuses mainly on round-based approaches, especially in relaxed loop freedom. The round-based solution has gained more attraction in literature due to the absence of impact on switch's resource than the stamping/tagging solution. However previous works focus on rapidly computing the update schedule or minimizing the schedule, and did not provide tradeoff between rapidly computing the schedule and reducing the make span of the schedule. In this work we have developed a new enhancement of the Peacock algorithm combining a fast update schedule computing and reducing the schedule make span to at most $2log_2(n) - 1$ rounds.

Keywords: Consistent Updates · SDN · Relaxed Loop Freedom

1 Introduction

In a software defined network, there is a separation between the control plane and the data plane (or forwarding plane). The control plane is held centrally by the controller, and the data plane is held by the network switches. The controller determines how the network (switches) should behave, by providing instructions or rules to the switch, while the latter is responsible for implementing these rules. However, it is up to the controller to verify the correctness of these rules, their coherence and their consistency in terms of respecting the intended plan.

Operation management in network, such as building and updating configuration and updates, often leads to temporary service unavailability due to how these tasks are sequenced. Consider a path from a source s to a destination d formed by node_1; node_2;…node_n, updating node_j before node_i; this might lead to a loop, a black hole…, or other consistency problems [1–3]. The issue raised here is finding a way to schedule the order of updating nodes in order to prevent service downtime. In the SDN context, the controller should not only define the path (rules), but it should also provide correct sequencing of the updates.

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2025
Published by Springer Nature Switzerland AG 2025. All Rights Reserved
D. Perakovic and L. Knapcikova (Eds.): FABULOUS 2024, LNICST 596, pp. 18–33, 2025.
https://doi.org/10.1007/978-3-031-72393-3_3

This article is mainly focused on the problem of loop freedom, this type of consistency is dependent on the relations between switches involved in the updates. This kind of scheduling updates known as rounds-based technique, involves partitioning the network into a subset of a network elements, where each subset can be updated in an arbitrary manner. This should preserve the consistency properties of loop freedom.

Loop freedom consistency is divided into two categories: strong loop freedom (SLF) and relaxed loop freedom (RLF) [4]. The first requires that no packet in the network enters a loop during updates (a loop should not occur in the network at any time). The second tolerates loops, but not on the path from a source to a destination subject to an update (only traffic from the source to the destination do not enter a loop at any time).

In the literature, two strategies are proposed to address loop freedom without affecting switch resources or adding packet tagging (no overhead to the network node resources): round based [4, 5, 14] and Node-based [13, 15] strategies. Round-based objectives split the network into several sets. Each set can be updated in one round, without inducing loops, by computing the minimum rounds schedule. The node-based approach aims at maximizing the number of switches that can be updated simultaneously without inducing loops. The rounds-based objective has attracted more attention. Recent work tried to answer the question of the existence of a k-round update schedule in each type of SLF and RLF. Authors in [4] and completed in [5] have proved that scheduling updates exist in RLF in $O(log(n))$ rounds, namely solving the problem in at most $6log_2(n)$, but it also showed that some instances can be solved in O(1). Authors in [14] proposed an update schedule in $O(log(n)log(log(n))$ rounds in SLF.

For a large size network, such as the service provider network, the problem of scheduling updates is very challenging due to the high number of network elements, and due to the size of the network routing table, the controller should:

1) Produce optimal scheduling, allowing fast transition to new rules or configuration, and preserving the network consistency.
2) Compute the schedule in the fastest way, as the number of updates is very high, and changes are very frequent in SDN.

In our work we are interested in the round-based technique for scheduling updates. By combining 1) and 2) we provide a new enhancement to the Peacock algorithm [5]. Our results lead to lower the upper bound to only $2log_2(n) - 1$.

The paper is organized as follows. In Sect. 2, we present an overview of related works, and in Sect. 3 we discuss the problem statement. In Sect. 4, we provide a mathematical model for the problem. In Sect. 5, we provide our algorithm. In Sect. 6 we detail our environment testing setup, and evaluate the performance of our algorithm against Peacock. Finally, we conclude our work in Sect. 7.

2 Related Works

The initial work of Mark et al. [1] and its extension in [3] explored policy consistency in SDN updates and introduced per-packet and per-flow consistency. They provided an abstract implementation model update, using their proposed mechanism called "two-phase commit". In per-packet consistency, each packet should be processed either by the

set of old rules in the entire network switch, or the new rules. In per-flow consistency, a packet belonging to the same flow should be handled by the same set of rules, either old or new. In either case, a mixture of old and new rules should never be used.

In the context of a data center, the work of Liu et al. [6] adds congestion freedom to the space consistency. The proposed algorithm named "zUpdate", addresses the problem of traffic migration. This has also been treated in [7–12] for providing a multi-stage update plan.

The limitation of the previous approaches results from the co-existence of the old and the new configuration at the same time, adding also the amount of change in SDN network updates, that causes the network switch resource to be oversubscribed, especially when the old configuration still processes traffic that did not leave the network, causing a delay in committing the new configuration.

The work of Mahajan et al. in [13] and [14] highlights the consistent updates in the SDN network and provides a basic analysis of update consistency properties: loop freedom, black hole freedom, packet coherence and bandwidth limitation. The first properties of loop freedom imply that no packet enters a loop in the network at any time during updates. Black hole freedom requires that packets arriving at a network switch should be processed by installed predefined rules; in it absence, the packet will be handled according to the switch policy. In packet coherence, the packet should be processed by a set of rules, either the old ones or the new ones, and not a combination of the two. Finally, the bandwidth limitation increases when traffic arriving in a network switch link is greater than its capacity.

3 Problem Statement

In SDN, arbitrarily updating network rules can generate loops, as shown previously. These loops can be harmful for network performance. Round-based approach is proposed to prevent loops by scheduling updates for a subset of rules that do not explicitly create loops. Let's consider a network, when the controller must update a path consisting of a combination of rules to a new path. This can be modeled as a graph $G(V, E_{op} \cup E_{np})$, consisting of nodes (V), and the set of old edges E_{op}, which represents the old set of rules, and the set of new edges E_{np}, which represents the set of new rules.

The main idea of the round-based approach is to produce an update schedule $U = (U_1, \ldots, U_m)$, that allows transition from E_{op} to E_{np} to guarantee that at each iteration t, knowing that all the previous updates are done (i.e. denoted $U_{<t} = U_1 \uplus U_2 \ldots . U_{t-1}$), the update U_t can be done safely without inducing a loop.

The proposed solution in [5], refered as Peacock, is a round based sequencing update algorithm which is based on a repetition of two operations: shortcut and prune. In each odd rounds, the algorithm updates the graph in order to reduce the distance between the source and destination. This is the operation of shortcut (based on edges sorting by distance). In even rounds, it updates nodes which are not on the path from the source to the destination (i.e., isolated nodes). This operation is known as the "prune operation". Authors in [5] demonstrate that, in the worst case, the number of nodes on the path is reduced by $n/3$.

In the example in Fig. 1, in the first round, the algorithm starts with a shortcut operation; it selects the nodes which have a forwarding edge with maximum distance.

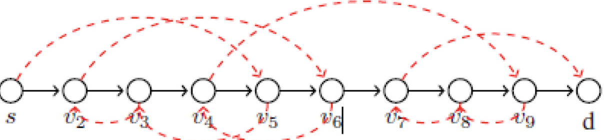

Fig. 1. An Update example Instance.

Here there is one node, v_4, which merges with $Out_2(v_4) = v_9$ then $U_1 = \{v_4\}$. In the second round U_2, the algorithm proceeds to the pruning; it updates nodes not on the current active path from s to d which are v_5, v_6, v_7, v_8 and merges them with their corresponding Out_2, we note $U_2 = \{v_5, v_6, v_7, v_8\}$. Using the same logic, we get $U_3 = \{s, v_9\}$; $U_4 = \{v_2\}$; $U_5 = \{v_3\}$.

In this paper, the objective is to reduce the number of rounds by an optimal selection of nodes to be updated. Our proposed algorithm is based on the same steps, except for the selection criteria which we modify in order to maximize the nodes to be updated in odd rounds and nodes to be updated in the next even rounds. In the previous example, Peacock produces 5 rounds while our solution can solve it in just 3 rounds.

The following tables describes our variables used in this model (Table 1).

Table 1. Key notation

Notation	Description
s	The source node
d	The destination node
v_i	The ith node
V	The set of nodes
E	The set of edges connecting nodes
E_f	The set of forward edges
E_b	The set of backward edges
Out_1	The outgoing node in the old path
Out_2	The outgoing node in the new path
e_{i1}	The outgoing edge from node v_i in the old path
e_{i2}	The outgoing edge from node v_i in the new path
U_i	The set of nodes updated in i^{th} round
S	The set of nodes with forwarding outgoing edge
g	The gain of subset S
n	The number of nodes
$G(n)$	The graph of n nodes

4 Network Model

We consider a network as a set of n switches (nodes) managed by one controller, modeled as a graph $G(V, E_{op} \cup E_{np})$. In our work, we will consider only a set of network rules connecting one source to one destination. We also consider only common nodes between the old path and the new path as loops are caused by them. The old path is formed by n nodes and edges $e_{i1} = (v_i; Out_1(v_i))$ for $i < n-1$, $E_{op} = \{(v_i; Out_1(v_i)), v_i \in V\}$ is the subset of edge e_i connecting node v_i to node $Out_1(v_i)$. The new path is also formed by n nodes and edges e_{i2}= $(v_i; Out_2(v_i))$ connecting node v_i to node $Out_2(v_i)$. These edges form the subset $E_{np} = \{(v_i; Out_2(v_i)), v_i \in V\}$. We assume that $Out_1(v_i)\#Out_2(v_i)$; meaning that the outgoing edge from the same node in the new path and in the old path does not point to the same node as shown in Fig. 2.

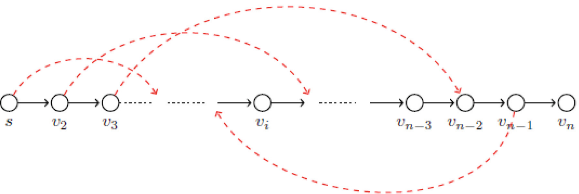

Fig. 2. Network model: the solid edges refer to old path and the dashed one refer to the new path.

It is important to notice that the two paths lead from s to d with no loop. As the graph is directed, the nodes can be ordered based on the old path in an ascendant manner.

$$s < v_2 < \ldots, v_{n-1} < v_n = d$$

In this model, all possible combinations of new paths are considered as a permutation of V where each element does not appear in its old position. More precisely for a network size of n, and since s and d do not change their position; this configuration corresponds to a derangement in the size of sub-factorial of $(n-2)$ (noted $!(n-2)$).

These hypothesis will be used for the proofs of our algorithm, analysis and measurement, we will assume that $s = v_1$ and $v_n = d$.

In the following we will use the following definitions and lemmas to provide proof of correctness of our algorithm.

Definition 1: As in [5], each edge in the directed graph $e_i = (v_i, v_j)$ represents a rule from the node v_i to the node v_j. Since they belong to the path from s to d and are ordered ascendly (i.e., $v_i < v_j$), if v_j is closer to the destination we define this edge as forward and we denote the subset of these edges as E_f, if v_i is closer to the destination then this edge as defined backward, and we denote this subset as E_b.

In the following, to avoid ambiguity, we will use this classification (i.e., forward and backward) for edges in the new path.

From Definition 1 and for each $v_i \in$ path from s to d:

- e_{i2} is forward if and only if $Out_2(v_i) > v_i$
- e_{i2} is backward if and only if $Out_2(v_i) < v_i$

In subset E_f, s and v_1 having forward edge e_2,and at least there is a node with edge forward (e_{i2}) which leads to the destination d.

Definition 2: In the new path, two edges from v_i and v_j overlap if and only if:

- v_i, v_j belong to the path from s to d.
- e_{i2}, e_{j2} are forward.
- $v_i < v_j$ Then $v_i < v_j < Out_2(v_i)$.
- $v_j < v_i$ Then $v_i < Out_2\left(v_j\right) < Out_2(v_i)$.

Definition 3: The distance of a forward edge e_{i2} is the number of edges skipped in the old path, expressed as $dist(e_i)$.

Definition 4: The simplest form of a loop exists in the directed graph $G(V, E_{op} \cup E_{np})$ when two nodes v_i and v_j exist such that there is a path from v_i to v_j and from v_j to v_i simultaneously.

Considering Definition 4, the loop between v_i and v_j is the formation of a combination of forward edges and backward edges leading from v_i to v_j and back from v_j to v_i.

Lemma1: For any update problem with respect to the hypothesis in this model, at least s and v_1 have forward edge e_2, and there is always a node which lead to the destination node d.

Proof: the derangement affects only nodes between s and$d$, with respect to the node sorting from s to d:

- s will always point in the new rule to a node which is higher and less than d.
- v_1 cannot point to s in the new rule, because s is not concerned by derangement, and thus will point to a higher node and may point to d.
- At least there is always a node in the deranged nodes that will point to d, because in this model the reachability should be maintained from s to d.

Lemma 2: Updating node v_i where e_{i2} is a forward does not create a loop.

Proof: Considering the reduction of the graph of $G(V, E_{op} \cup E_{np})$ to the graph $G'(V, E_{op} \cup E_b)$, and since e_{i2} is forward, the outgoing node in the new path is nearer to the destination than the node v_i, all packets from the source to the destination are sent toward the destination.

Lemma 3: If during an update, a node v_i is detached from the active path s to d then the update of this detached node can be done in the next update without causing a loop.

Proof: As in our case, we concern only about loops in the current path from s to d, these nodes detached from the path will not receive any packets with source s and destination d and thus do not create loops.

5 Algorithm

To resolve this update problem, we will use the previous Lemmas 1, 2 and 3 to build the algorithm of the update. We will be considering the following approach:

In each round as in Peacock algorithm:

- In an odd round, add the selected node with forward edge to the update sequence. and by Lemma 2 will not create loops.
- In an even round, add nodes, which are not in the current path from s to d to the update sequence. According to Lemma 3 this subset won't create loops.
- Resulted graph after odd and next even round respect Lemma 1.
- Repeat until all nodes are updated.
- Each node added to a round will be merged (fused) with its outgoing nodes according to the new rule (Out_2). The resulting node after node v_i is merged with v_j, will inherit $Out_1(v_j)$ and $Out_2(v_j)$. The size of the graph will be reduced, leaving only non-updated nodes.

Algorithm 1: Computing Update schedule

```
Result: Set U = (U_1, ..., U_m)
t = 1;
while G nonsingular do
    compute all edge forward and backward;
    U_t = ∅;
    if t is odd then
        add the subset S of node according to (3) to U_t;
        merge node in U_t with Out_2;
        t++;
    else
        add node not on the path s~d to U_t;
        merge node in U_t with Out_2;
        t++;
    end
end
Return U_1, ..., U_m;
```

The sequential procedures involved in the update process is outlined in Algorithm 1. It is worth emphasizing that we maintain a record of the source node throughout the algorithm's execution. This is because the nodes are sorted according to the current path from s to d Even in cases where the source node is included in a round; the subsequent outgoing node assumes the role of the new source node.

To address the problem at hand, we break it down into two distinct sub-problems. In the odd rounds and the subsequent even rounds, our objective is to maximize the selection of nodes explained in 5.1, constituting the first sub-problem. Once this sub-problem is resolved, the resulting graph becomes the basis for the second sub-problem. We then initiate the process again from step 1) in algorithm 1 with this modified graph.

By partitioning the problem into two subproblems, we can determine the number of update rounds required for the graph $G(n)$ based on $G(n-p)$, where $p > 1$. In each odd and subsequent even round, p nodes are selected, resulting in a remaining count of $n-p$ nodes that need to be updated. This division enables us to accurately calculate the number of rounds needed for the update process.

In the following we introduce Lemma 4 to provide proof of Algorithm 1 termination.

Lemma 4: For any update instance $G(n)$ with respect to condition in this model, selecting at least one forward node in the odd round, and selecting the detached nodes in the next even round, the resulted $G(n-p)$ is an update instance, and the Algorithm 1 terminate.

Proof: At least this forward node v_j have a distance of 2 and will cause only one node v_{j+1} which is between v_j and $Out_2(v_j)$, because of the deranged position in the new path, v_{j+1} is Out_2 of an other node v_k, and there exist another node v_i which is $Out_2(v_{j+1})$. After updating and merging nodes, the configuration of the resulted graph is as follow:

$$E_{op} = \{Out_1(\mathrm{s}), Out_1(v_1), ..Out_1\left(v_{j+2}\right), Out_2(v_k), \ldots Out_1(v_{n-1})\}$$

$$E_{np} = \{Out_2(\mathrm{s}), Out_2(v_1), ..Out_2\left(v_{j+2}\right), ., Out_2(v_k), ..Out_2(v_{n-1})\}$$

Such that:

- $Out_2(v_k) = v_{j+1} = v_i$, because v_{j+1} was merged with v_i.
- For every node v_i; $Out_1(v_i) \neq Out_2(v_i)$ are conserved.
- E_{op} and E_{np} lead from s to d
- The new path corresponds to a derangement of the nodes between s and d.

Recursively after each odd and next even round the size of the graph is reduced until all nodes are updated; this ensures that the Algorithm 1 terminates.

5.1 Maximize the p Nodes in the First Sub-problem

As explained previously in Sect. 3, the Peacock algorithm disregards the number of nodes to be updated in each round. In our work, we aim at maximizing the number of nodes. To achieve this goal, we present in the following the approach used to select the maximum number of p nodes.

We observe that in each odd round several nodes will be detached from the path s to d. Let S be a collection of node having non-overlapping edges, such that $\mathrm{S} \subseteq V$ satisfies the following conditions:

- S contains at least one node with a forward edge.
- If $u, v \in$ S then their edges do not overlap.
- This subset can be only updated in odd rounds.
- Updating nodes in S reduce the length of the path from s to d

Each selected S according to our selection criteria will generate nodes that are detached from the path s to d after updating nodes in S. Let R be the subset of nodes detached due to the update of the subset S. Consider now the general case, each updated

node v_j in S will generate a number of nodes detached from the active path from s to d corresponding to $dist(e_{j2}) - 1$ nodes (as defined in Definition 3), nodes in S will generate the subset R with the size expressed in Eq. (1).

$$|R| = \sum_j dist(e_{j2}/v_j \in S) - |S| \tag{1}$$

Nodes in R can be updated safely in the next even round, so that the resulting number of nodes to be updated in an odd round and the next even round is derived in Eq. (2). This target number is noted p. Let g be the maximum of p expressed in Eq. (3)

$$|R| + |S| = \sum_j dist(e_{j2}/v_j \in S) = p \tag{2}$$

$$\max_{S \subset V}(\sum_j dist(e_{j2}/v_j \in S)) \triangleq g \tag{3}$$

The next step is to maximize p, by finding S satisfying the objective function in (3). We first consider the reduced graph $G(V, E_{op} \cup E_{np} \backslash E_b)$ and set the edge distance as follows:

- For each edge in E_{op}, the distance is equal to 0.
- For each edge in $(E_{np} \backslash E_b)$, the distance is as defined in Definition 3.

Finding the target S satisfying objective function in (3) is equivalent to finding the shortest path from the source to the destination.

Proof: to prove the equivalence, let P the shortest path from s to d, let S be the subset corresponding to g.

We need to proof that the nodes in S have thier edges e_{i2} in P and the length of P is $n - 1 - |R| = n - 1 - g + |S|$.

Let p' the path associated with S. Such that, each $v_j \in S$; we have $e_{j2} \in p'$; if p' is a shortest path than $lenght(p') = n - 1 - |R| = n - 1 - g + |S|$ is the smallest. Then g is maximal.

Our proposed algorithm, Algorithm 2, aims to identify the Set S that corresponds to g. It operates by taking the graph $G(V, E_{op} \cup E_f)$ as input and efficiently determining the shortest path from s to d. Consequently, the algorithm selects the appropriate S associated with the shortest path.

Algorithm 2: Computing the subset S

Input: $G(V, E_{op} \cup E_f)$
Result: Subset S which satisfy (3)
Find the shortest path from s to d
Get all new added edges $(e_{j2} \in E_f)$
Return nodes v_j corresponding to the added precedent e_{j2}

If we apply our solution to the example in Fig. 1, we have two shortest path corresponding to $S = \{(s, v_7)\}$ or $S = \{(v_2, v_7)\}$ In the first round we can select $S = (s, v_7)$; which leads to the detached nodes (v_2, v_3, v_4) generated by the update of s; and (v_8, v_9) generated by the update of v_7, i.e. $R = (v_2, v_3, v_4, v_8, v_9)$. We can use Eq. (1) to verify that the number of nodes in R is equal to 5 according to Eq. (4).

$$|R| = dist(e_{s2}) + dist(e_{72}) - |S| = 5 \tag{4}$$

Following this, the next round involves updating these 5 nodes, resulting in a cumulative update of 7 nodes by the end of the second round.

It is also possible to consider $S = \{(v_2, v_7)\}$ for the first update, which will result in $U_1 = (v_2, v_7)$ and $U_2 = (v_3, v_4, v_5, v_8, v_9)$; this solutions also maximizes the number of updated nodes.

Our final Lemma 5 serves as finding lower bound of g.

Lemma 5: For any update instance during the odd round and next even rounds we have:

$$g \geq n/2 \tag{5}$$

Proof: Suppose that g is less than $\frac{n}{2}$. In this case, g is at least equal to $\frac{n}{2} - 1$. The corresponding Set S contains at least one node, denoted as v_p and all other forward edges overlap with this edge e_{p2}. According to Lemma 1, there are at least two nodes with forward edge e_{i2}, specifically nodes s (the source node) and v_l (the node leading to the destination node d).

Since v_p Corresponds to the maximum value of g, we can establish that $dist(e_{12}) + dist(e_{l2}) \leq diste_{p2}$. Furthermore, it follows that Out_2(s= v_1) is at least the node immediately following v_p, and v_l is the node preceding $Out_2(v_p)$. Consequently, this configuration does not allow v_l to reach d directly. To reach the destination, $Out_2(v_l)$ would have to skip more than three edges.

5.2 Upper Bounds

Using Lemma 5 in each odd rounds and next even round the size of the graph is reduced at least by half, and thus the updating problem needs $2log_2(n) - 1$ rounds to update all nodes in the worst case. We named it "Enhanced".

6 Performance Evaluation

6.1 Methodology

In the first part of this section we evaluate the performance of our solution and provide a benchmark against Peacock algorithm in term of number of rounds produced for each network size, and the time it take to compute it.

To do so, we use a lab environment server (Intel Xeon® E5-1620), to generate scenarios of rules that we apply both to our solution, and Peacock.

Therefore, for different network sizes, we generate new rules using derangements, as stated at the beginning, in a way to cover all possibilities. Let's remind that the source node and destination node do not change their initial position, and thus are not subject to derangement.

To illustrate our contribution, we consider the size of network varying from 8 to 100 nodes. Hence due to the increasing size of possibilities (i.e., $!(n-2) = \left[\frac{(n-2)!}{e}\right]$) we will consider two scenarios. In the first scenario, when the size of the network is up to 12, we apply Peacock and our algorithm to all possible derangements. In the second scenario, when the size of the network exceeds 12, we use random generator to construct rules and then apply Algorithm 1.

In the first scenario, we use all possible derangements for a small size (8, 10, 12). For each derangement the new rules are constructed by appending the generated list of nodes to the source node and finally appending the destination node.

In the second scenario, we use random generator by randomly picking a node from the list of nodes (except the source node and destination node). The construction of the new rule is done by appending this selected node to the source, until all nodes being added, and finally append the destination. These nodes, which were randomly picked should not be in the same order as in the old rule. We limit our simulation to 10000 samples for each network size.

In each scenario we measure the time consumed to compute the schedule and the number of rounds produced by each algorithm.

In the second part of this evaluation, we evaluate the performance using Ryu controller and Mininet to generate the required topology, we limit to two network size of 20 and 40 nodes due to resource constraints. And evaluate the time to update the network. For this purpose, we use OF1.4 as a protocol for communication between Ryu and Mininet's switches.

To accurately measure the time required for updating the entire network, we utilized the bundle feature for configuring rules. This feature offers the advantage of providing acknowledgments when committing changes to switches, enabling us to track the duration of the update process effectively:

- OFPBCT_OPEN_REQUEST: to open the bundle
- OFPT_BUNDLE_ADD_MESSAGE: to add rules
- OFPBCT_CLOSE_REQUEST: to close the bundle
- OFPBCT_COMMIT_REQUEST: to commit the change.

After committing the change, the switch acknowledges the controller by an OFPBCT_COMMIT_REPLY message.

6.2 Results

The results for the first part of our evaluation are shown in Fig. 3,4 and 5. Figure 3 shows that at both small and large scale, our algorithm reduces the number of rounds. In small scale runs, all cases were solved in 3 rounds, while they were solved in 3 to 7 rounds with Peacock. Equally, at large scale, our solution decreases the number of rounds.

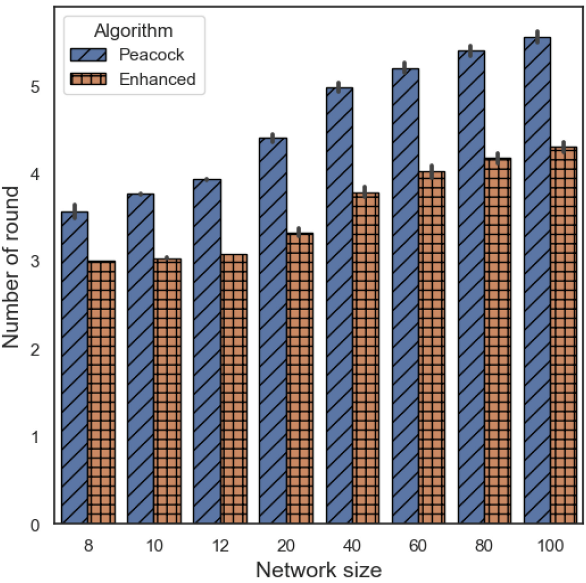

Fig. 3. Average number of rounds

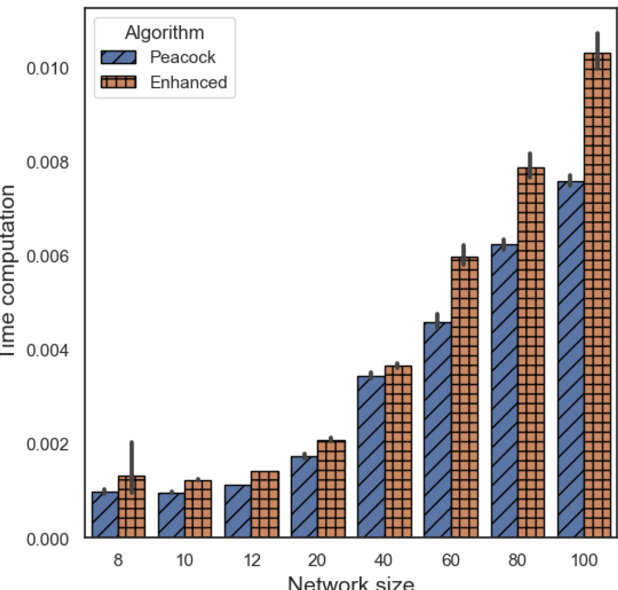

Fig. 4. Time computation in seconds

Figure 4 illustrates that our algorithm may exhibit longer execution times compared to Peacock. This discrepancy arises from the utilization of topological sorting in Peacock, which operates at a faster time complexity of $O(|V| + |E|)$. In contrast, our algorithm relies on the calculation of the shortest path, which has a time complexity of $O(|E| + |V|\log(|V|)$.

Figure 5 presents the cumulative distribution function (CDF) distribution, depicting the network sizes employed in the simulation. The results reveal that our algorithm achieved a remarkable 96% success rate in solving use cases within three rounds, whereas Peacock only managed to solve 29% of them within the same timeframe. This high efficiency is attributed to the utilization of optimization criteria, specifically designed to maximize the number of nodes updated during alternating rounds—both odd and even.

In the second part, Fig. 6 illustrates the time required to update an entire network, which includes both the time to compute the round and the time to update switches. Our algorithm demonstrates significantly faster update times compared to Peacock. This efficiency is primarily attributed to the considerably fewer rounds generated by our algorithm, despite the computation taking longer than Peacock.

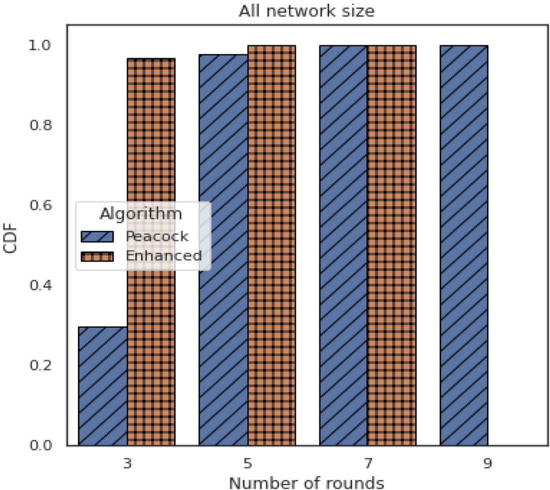

Fig. 5. CDF

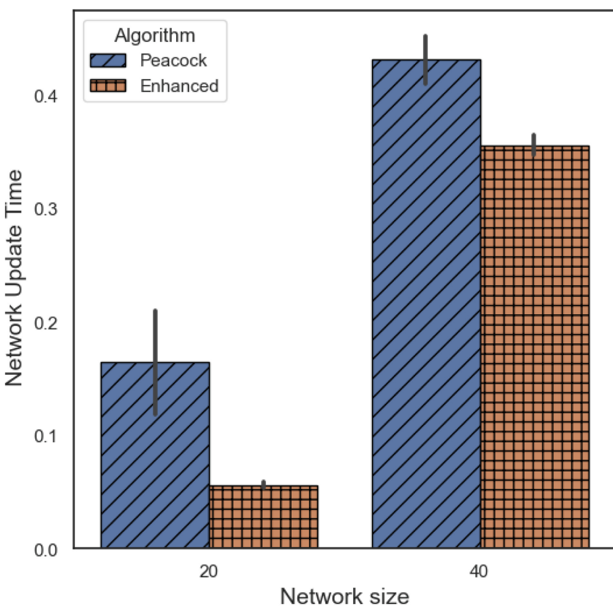

Fig. 6. Time Update in seconds

7 Conclusion

This study addresses the issue of achieving consistent updates in software-defined networks, with a specific emphasis on temporary loops that arise during arbitrary updates. Our approach involved modeling the network with relaxed loop freedom and proposing a round-based algorithm. This algorithm significantly reduced the number of rounds required, achieving a three-fold improvement compared to the Peacock Algorithm. As a result, our solution enables faster updates when compared to previous algorithms. We conducted simulations to evaluate the performance of our proposed solution, demonstrating the reduced number of rounds and highlighting the improved efficiency of round-based updates while maintaining loop-free consistency in the context of relaxed loop freedom. Considering multiple source/multiple destination in the context of datacenter is an interesting topic for future research as DC technologies have been widely adopted and an increase in demands has been exhibited.

References

1. Reitblatt, M., et al.: Consistent updates for software-defined networks: change you can believe in! In: Proceedings of the 10th ACM Workshop on Hot Topics in Networks - HotNets 2011, pp. 1–6. ACM Press (2011). https://doi.org/10.1145/2070562.2070569
2. Mai, H., et al.: Debugging the data plane with anteater. In: Proceedings of the ACM SIGCOMM 2011 Conference on SIGCOMM - SIGCOMM 2011, p. 290. ACM Press (2011). https://doi.org/10.1145/2018436.2018470

3. Reitblatt, M., et al.: Abstractions for network update. In: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication - SIGCOMM 2012, p. 323. ACM Press (2012). https://doi.org/10.1145/2342356.2342427
4. Ludwig, A., et al.: Scheduling loop-free network updates: it's good to relax! In: Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, pp. 13–22. ACM (2015). https://doi.org/10.1145/2767386.2767412
5. Foerster, K.-T., et al.: Loop-free route updates for software-defined networks. IEEE/ACM Trans. Netw. **26**(1), 328–341 (2018). https://doi.org/10.1109/TNET.2017.2778426
6. Liu, H.H., et al.: ZUpdate: updating data center networks with zero loss. In: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, pp. 411–422. ACM (2013). https://doi.org/10.1145/2486001.2486005
7. Hong, C.-Y., et al.: Achieving high utilization with software-driven WAN. In: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, pp. 15–26. ACM (2013). https://doi.org/10.1145/2486001.2486012
8. Zheng, J., et al.: Minimizing transient congestion during network update in data centers. In: 2015 IEEE 23rd International Conference on Network Protocols (ICNP), pp. 1–10. IEEE Xplore (2015). https://doi.org/10.1109/ICNP.2015.33
9. Jin, X., et al.: Dynamic scheduling of network updates. ACM SIGCOMM Comput. Commun. Rev. **44**(4), 539–550 (2015). https://doi.org/10.1145/2740070.2626307
10. Forster, K.-T., Wattenhofer, R.: The power of two in consistent network updates: hard loop freedom, easy flow migration. In: 2016 25th International Conference on Computer Communication and Networks (ICCCN), pp. 1–9. IEEE Xplore (2016). https://doi.org/10.1109/ICCCN.2016.7568583
11. Brandt, S., et al.: On consistent migration of flows in SDNs. In: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9. IEEE Xplore (2016). https://doi.org/10.1109/INFOCOM.2016.7524332
12. Foerster, K.-T.: On the consistent migration of unsplittable flows: upper and lower complexity bounds. In: 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA), pp. 1–4. IEEE Xplore (2017). https://doi.org/10.1109/NCA.2017.8171348
13. Mahajan, R., Wattenhofer, R.: On consistent updates in software defined networks. In: Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, pp. 1–7. ACM (2013). https://doi.org/10.1145/2535771.2535791
14. Förster, K.-T., et al.: Consistent updates in software defined networks: on dependencies, loop freedom, and blackholes. In: 2016 IFIP Networking Conference (IFIP Networking) and Workshops, pp. 1–9 (2016). IEEE Xplore. https://doi.org/10.1109/IFIPNetworking.2016.7497232
15. Amiri, S.A., Ludwig, A., Marcinkowski, J., Schmid, S.: Transiently consistent SDN updates: being greedy is hard. In: Suomela, J. (eds.) SIROCCO 2016. LNCS, vol. 9988, pp. 391–406. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48314-6_25
16. Ludwig, A., et al.: Good network updates for bad packets: waypoint enforcement beyond destination-based routing policies. In: Proceedings of the 13th ACM Workshop on Hot Topics in Networks, pp. 1–7. ACM (2014). https://doi.org/10.1145/2670518.2673873
17. Maity, I., et al.: CURE: consistent update with redundancy reduction in SDN. IEEE Trans. Commun. **66**(9), 3974–3981 (2018). https://doi.org/10.1109/TCOMM.2018.2825425
18. Basta, A., et al.: Efficient loop-free rerouting of multiple SDN flows. IEEE/ACM Trans. Netw. **26**(2) 948–961 (2018). https://doi.org/10.1109/TNET.2018.2810640
19. Zheng, J., et al.: Scheduling congestion-free updates of multiple flows with chronicle in timed SDNs. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), 12–21 (2018). https://doi.org/10.1109/ICDCS.2018.00012

20. Foerster, K.-T., et al.: Survey of consistent software-defined network updates. IEEE Commun. Surv. Tutor. **21**(2), 1435–1461 (2019). https://doi.org/10.1109/COMST.2018.2876749
21. Even, S., Gillis, J.: Derangements and laguerre polynomials. In: Mathematical Proceedings of the Cambridge Philosophical Society, vol. 79, no 1, pp. 135–143, January 1976. https://doi.org/10.1017/S0305004100052154