SEAMLESS ROUTE UPDATES IN SOFTWARE-DEFINED NETWORKING VIA QUALITY OF SERVICE COMPLIANCE VERIFICATION

S. L. Frenkel[1] and D. Khankin[2]

Abstract: In software-defined networking (SDN), the control plane and the data plane are decoupled. This allows high flexibility by providing abstractions for network management applications and being directly programmable. However, reconfiguration and updates of a network are sometimes inevitable due to topology changes, maintenance, or failures. In the scenario, a current route C and a set of possible new routes $\{N_i\}$, where one of the new routes is required to replace the current route, are given. There is a chance that a new route N_i is longer than a different new route N_j, but N_i is a more reliable one and it will update faster or perform better after the update in terms of quality of service (QoS) demands. Taking into account the random nature of the network functioning, the present authors supplement the recently proposed algorithm by Delaet *et al.* for route updates with a technique based on Markov chains (MCs). As such, an enhanced algorithm for complying QoS demands during route updates is proposed in a seamless fashion. First, an extension to the update algorithm of Delaet *et al.* that describes the transmission of packets through a chosen route and compares the update process for all possible alternative routes is suggested. Second, several methods for choosing a combination of preferred subparts of new routes, resulting in an optimal, in the sense of QoS compliance, new route is provided.

Keywords: software-defined networking; Markov chains; quality of service

DOI: 10.14357/19922264180408

1 Introduction

Software-defined networking is an emerging network paradigm, in which the control plane is decoupled from the data plane enabling centralized control logic. Such a dynamic network may require frequent modifications and updates to the network topology and configuration. Also, the network topology is available to the centralized control entity, there, due to this flexibility, it is possible to perform offline optimized calculations.

Network functions virtualization (NFV) allows replacing traditional network devices with software that is running on commodity servers. This software implements the functionality that was previously provided by dedicated hardware. Network functions virtualization allows services to be composed of virtual network functions (VNF) hosted on different data centers. Software-defined networking, when applied to NFV, helps in addressing challenges of dynamic resource management and intelligent service orchestration [1]. Sometimes, traffic is often required to pass through and be processed by an ordered sequence of possibly remote VNFs [2]. For example, traffic may be required to pass through intrusion detection system, proxy, load balancer, or a firewall. Such concatenation of services is called *service function chaining* (SFC).

Consider, for example, two communicating parties in a network featuring complex network topology (e. g., Small-world network), and the communication flow is passed over a series of VNFs. It may be the case that the network operator is required to move the communicating flow to a different path due to QoS requirements or other possible cost considerations. We are interested to model the anticipated expected number of steps until the update is complete given a possible new route following the required QoS demands, e. g., delay, communication rounds, cost, etc.

Let us consider a pair $(C, \{N_i\})$ where a current route C from s to d is scheduled to be replaced by a new route from the set $\{N_i\}$, each from s to d either. Let us model each route as an ordered list of network elements, such as VNFs (SFCs) or generally saying routers. Each new route N_i is constructed during the update process, and thus, certain delays may be introduced due to initial packet processing or due to possible losses.

The design goals must be achieved by constructing effective algorithms for efficient packet QoS routing in NFV/SDN computer network. Depending on the QoS

[1]Institute of Informatics Problems, Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, 44-2 Vavilov Str., Moscow 119333, Russian Federation, fsergei51@gmail.com

[2]Computer Science Department, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, danielkh@post.bgu.ac.il

metric, the lower (e. g., for reliability) or upper (e. g., for a delay) constraints represent the desired bounds that the orchestration must meet. Since different configurations could meet these bounds, the designer should also optimize against a specific metric by using these both ends of the extreme.

Methods based on integer linear programming (ILP) were proposed in several works (see section 2). The difficulty of using tools based on ILP in the operational work of an administrator is that in view of the possible infeasibility of the resulting solution, it may take not a few resources (time, efforts) until acceptable QoS values can be ensured.

We consider the use of "design via verification" approach, suggesting a method for complying QoS demands. The method is based on augmenting the update algorithm with a verification logic. Namely, we suggest the use of *Probabilistic real-time Computation Tree Logic* (PCTL) [3] for expressing real-time and probability in systems. Using PCTL, we can express the probability for a process to complete after a certain number of steps along an execution path and verify the selected route for the update.

Delaet *et al.* proposed a multicast-based scheme for seamlessly updating a current route to a new one [4]. According to the multicast scheme, the controller instructs a router to temporarily have two (s, d) entries in the routing table. When a router $r \neq d$ receives a packet from s to d, it sends the packet according to the forwarding instructions of all of its (s, d) routing table entries. When two identical copies of a packet that was multicasted over the current and new portion of a route arrive, the controller can dismantle the current route, as the new route is ready. During the update process, packets should not be lost, no cycles should be formed, and communication should not be disrupted.

Our contribution is a model for a successful route update, including its intermediate steps, as MC states, each with a given probability. With our model, we are able to characterize the quality of an update by expected number of steps in the MC.

We suggest an enhanced update method for the network administrator to augment his decision regarding QoS demands in terms of various network parameters and possible failure of the update process. Moreover, in contrast to other works, we are able to provide a version of an algorithm that can perform real-time QoS assessment during a route update, for each subpart of a route. At last, using our method, it is possible that the active new route will be comprised of subparts of different new routes, providing optimal route update service in regard of required network QoS.

Extended abstract of this work appeared as a conference paper in [5] which presented preliminary results. In this work, we describe in detail the system settings and bring new results by providing two additional algorithms.

In the following section, we overview the related work. Next, we provide the required definitions and the system settings and describe the MC characterization of the network. Further, we describe different update setting, accordingly accompanying algorithms and data structures, used for QoS assessment during route updates.

2 Related Work

Quality of service routing using multipath was proposed in [6]. The routing algorithm, initially, eliminates all links that do not meet the bandwidth requirements. Then, it finds disjoint shortest paths based on the residual network graph in each iteration.

The work [7] proposed a QoS optimized routing over multidomain OpenFlow networks managed by a distributed control plane, where each controller performs optimal routing within its domain. The QoS routing problem was posed as a constrained shortest path (CSP) problem, and the proposed solution computes a near-optimal route, based on LARAC (Lagrange relaxation based aggregated cost) algorithm [8]. The proposed algorithm is an approximation algorithm; it always gives a suboptimal solution.

For traditional network architecture, a routing strategy approach based on ILP was introduced in [9]. The main disadvantage of using ILP is that the problem is NP-hard. Additionally, ILP cannot be applied to probabilistic values. Using linear programming (not limited to integers) rounded to integer solutions will not yield an optimal solution.

Route updates are extensively researched in SDN [10], standing on the work by Reitblatt *et al.* where requirements for SDN updates were examined. This work focused on per-packet consistency property, stating that packets have to be forwarded either using the initial configuration or the final configuration but never a mixture of them, throughout the update process [11]. The authors proposed a 2-phase commit technique which relies on packets tagging so that either of the rules is applied. However, such technique wastes critical network resources and complications are formed due to packet tagging [10]. Further, Delaet *et al.* showed in [4] that using a careful multicast during route updates provides a better working solution.

Hogan and Esposito propose in [12] the use of Bayesian networks for delay estimation as a traffic engineering tool and model the path selection problem using a risk minimization technique. However, the authors state that the accuracy of their model is limited by its ability to correctly identify dependencies in the data. In our work, we suggest a general tool for probabilistic verification of any network parameter, which does not depend on variance within the dataset.

In [13], an update protocol proposed where packets are sent to the controller during updates; such approach adds a significant cost to the control plane bandwidth [4]. In [14], an algorithm to find a safe update sequence expressed as a logic circuit has been proposed. However, the algorithm requires a dedicated protocol which is not currently supported [10]. The authors of [15] propose to perform the 2-phase update scheme from [11] incrementally, making the update longer.

Software-defined networking allows the involvement of the network administrator into the network management during route udpdates and, in particular, during packet transmission. Thus, it would be highly desirable to support the decision making process with the right tools. Our novelty is exactly such tool, for augmenting online decision making of the network administrator during network management in a stochastic environment.

The work by Delaet *et al.* [4] introduced the Make&Activate-Before-Break approach for seamless route update in SDN. The authors described in a high-level the multicasting-based update, which we employ in this work. Also, they introduced a controller-based method for verifying the correctness of a new route before the traffic redirection. Dinitz *et al.* [16] extended the work [4] to the general case of several dependent (via shared links) routes pairs. The routes update problem was proved to be NP-hard [17]. The authors of [16] suggested the use of artificial intelligence (AI) methods for solving the problem. As a basis for AI-based solutions, Dinitz *et al.* proposed a dependence graph model describing the current state of the problem instance at any replacement stage. In addition, route readiness verification similar to that in [4] was implemented in [16] as a high-level network protocol.

In this work, we investigate a different problem; we consider the route updates problem from a QoS perspective and describe in high-level both the prediction and the update processes.

3 Preliminaries and Definitions

The basic system settings are as follows. For a (route) sequence X, we denote by x_i the ith element in it. In a (directed) communication network, we are given a route C from source s to destination d. Additionally, we are given a set of different new routes N_i, each going from s to d. We model each route as an ordered set of network nodes connected by network links. We assume that neither of the routes contains cycles. Each router in a route matches a packet from s to d and forwards the packet to the next router in order. After the update is complete, each router in the new route should forward the packets from s to d to the next router in order along the new route.

In our work, we consider the route replacement problem as a sequence of subroutes replacements. The routes replacement subsystem was in great detail described by Dinitz *et al.* in [16]. We borrow from [16] the relevant parts which we briefly describe here.

Definition 1. We define a subset from $a \in X$ to $b \in X$ of an ordered set X, when a precedes b, as a subroute from a to b, and denote such subroute by $[a, b]$.

Subroutes. The current route C subdivides each new route to k common subroutes (a subroute may consist of one router in the simplest case) and $k-1$ noncommon subroutes. For illustration, see Fig. 1. In Fig. 1 and figures below, the current route is depicted in a light grey color full nodes, connected with solid edges. The new route is depicted in white colored nodes, connected with dashed edges. The common nodes are depicted as shaded. If there are several new routes, the nodes of each route are filled with a designating pattern. Additionally, for easier reading, when it is possible, we denote subroutes of some route X as X', X'', etc. In other cases, a subroute j of a new (current) route i is denoted as $N^i_j(C^j_i)$. Similarly, routers of some route X are denoted by r', r'', etc.

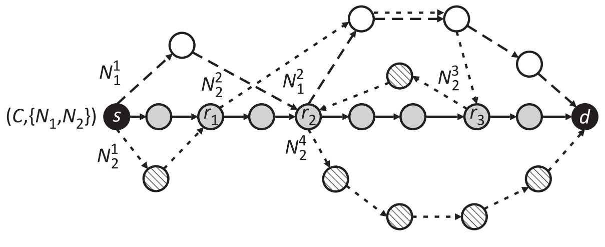

Figure 1 Route C with two possible new routes sharing a link

In the example in Fig. 1, noncommon new subroutes of route N_1 are denoted by $N^1_1 = [s, r_2]$ and $N^2_1 = [r_2, d]$, while the noncommon new subroutes of N_2 are denoted by $N^1_2 = [s, r_1]$, $N^2_2 = [r_1, r_3]$, $N^3_2 = [r_3, r_2]$, and $N^4_2 = [r_2, d]$.

Note that in general, the order of common subroutes along C and along N can be different. See, for example, the common subroutes of C and N_2 in Fig. 1.

Definition 2. A new noncommon subroute of N from router a to router b is legitimate for update only if a precedes b on the route C.

Definition 2 guides us on which subroutes can be launched without creating routing cycles in the network system. (See [4] for details.)

When an update of a subroute N' from router r to r' is finished, the update flow goes along C from s to r, continues along N' up to r', and finishes along C from r' to d. For illustration, see the result of launching N^4_2 in Fig. 2.

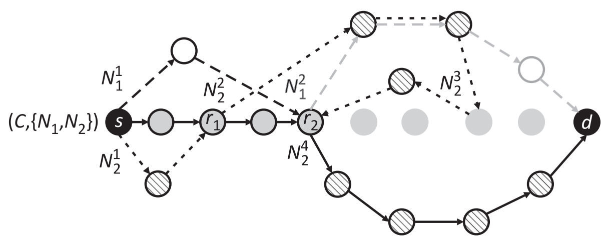

Figure 2 N_2^4 was launched

Note that launching a currently nonlegitimate new subroute, for example, N_2^3 in Fig. 1, is forbidden since it will form a cycle resulting in packets circulating and overwhelming the network.

Dynamics of the system. Dinitz *et al.* performed a detailed analysis on the dynamics of a subroutes system. After an update of a subroute is complete, the set of current subroutes C and the set of new subroutes N are recalculated. This may result in different system of subroutes. For example, see Fig. 2 where after the launch of N_2^4 from the example in Fig. 1, the sets of subroutes are recalculated. As a result, we obtain different subroutes (for clarity, the previous labels are kept). See also [16] for details and extensive analysis.

3.1 Markov chain characterization of the network states

We characterize execution of some (sub)route in the network by a packet delay time between the (sub)route's common sender and common destination routers as well the probability of a packet drop. Let us for now define our network routing model (conceptual model) informally in the following terms. Delay of a packet is obtained using a physical delay and the total processing time in the router. We consider that transmission of packets in a network can have a random behavior, caused by the random character of both, the input, and possible loss of packets. There we are interested in a probabilistic model, namely, a Markov model. In order to fully characterize the network as an MC, the internal state of each router (and, in particular, the buffer occupancies), as well as the characteristics of all flows, need to be expressed as states in the chain.

However, such approach would result in an enormous and intractable number of states. Therefore, to simplify these computations, let us characterize the delay time as an abstract variable t. This abstract variable can be interpreted in different ways, e. g., the current processing queue length and a packet transmission rate of the link, or possibly a fixed value, such as an interval between the beginning of a packet transmission after being processed in some node and the end of processing at the next node.

We describe the functioning of the network in the transmission of packets as transitions of a discrete-time MC (DTMC). The state space corresponds to the set of nodes such that the transmission of a packet from a node that has finished processing the packet to the next node corresponds to the transition of the chain to the next state.

Discrete-time MC is defined as a tuple $D = (S, s_0, P)$. In the tuple, S is the finite set of states, $s_0 \in S$ is the initial state, $P : S \times S \to [0,1]$ is the transition probability matrix in which $\forall s \in S$, $\sum_{s' \in S} P(s, s') = 1$. For any two states $s, s' \in S$, if $P(s, s') > 0$, then s' is the successor of s. For a subset of states $T \subseteq S$, the probability of moving from a state s to any state $t \in T$ in a single step is denoted by $P(s,T)$ and is given by $P(s,T) = \sum_{t \in T} P(s,t)$.

3.2 Verification syntax

For implementation of our PCTL-based model, we use PRISM — probabilistic model checker [18]. There, we follow PRISM property specification language. Here, we briefly describe the essential syntax while more details can be found in [19].

Given a property Ψ, we say that Ψ is true with probability p and write that as $P_p[\Psi]$. If the probability p is unknown, PRISM allows, for DTMC, writing properties queries of the form $P_{=?}[\Psi]$, meaning "what is the probability that Ψ is true?". Additionally, it is possible to use a time bound and write properties queries such as $P_{=?}[F^{\leq T}\Psi]$, meaning "what is the probability that Ψ is true after less than T steps?". At last, it is possible to compute properties such as expected time or expected number of steps. For example, $R_{=?}[F\Psi]$, meaning "what is the expected number of steps until Ψ is true?".

4 Prediction of Preferred Update

The states of a DTMC describe the nodes in the new route and the transition probabilities in the chain represent the possible delay or a packet loss in the routers along the new route. The states are defined as $\{s_1, \ldots, s_n\}$ where n is the number of nodes in the new route. The network achieves the state s_i if a packet has reached the ith node. For example, in Fig. 3, the self-transition edge represents the probability for a delay due to packet loss, rules installation at the router, or congestion on the router-controller link, while the forward transition edge represents the probability for a successful transition to the next state. These probabilities can be estimated from network statistics (see, for example, [12]). The labels on edges are the probability values, when edge has no label means probability 1.

The initial probability distribution of states is given by the vector P_0 of size n. We can determine the prob-

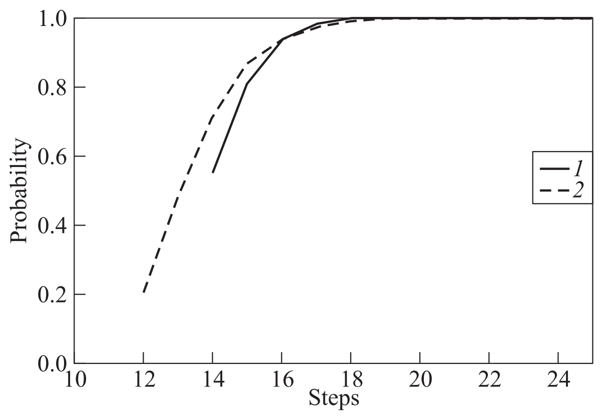

Figure 3 Probability as a function of number of steps to update routes N_1 (*1*) and N_2 (*2*)

ability that a particular route delays the update process by k, that is, the number of steps required for a successful update is given by $p(k) = P_0P^k$. Using this characteristic, which is, in fact, the probability distribution of the number of steps $P(k < x)$, one can calculate various properties like average delay time for the new route, maximum or minimum number of steps to update, etc.

Consider the example illustrated in Fig. 4. Figure 4*a* illustrates the current route C and a candidate new route N_1. Figure 4*b* shows the same current route C with another candidate new route N_2. Figures 4*c* and 4*d* show the MCs for new routes N_1 and N_2, accordingly, with given transition probabilities.

During the update process, packets are sent along the current and the new routes. Since the new route is not operational yet, packets can be delayed due to congestion on certain nodes or due to switch configurations. For example, if routing rules have not yet been installed in some switch, then an arriving packet is sent to the controller [20]. The controller then decides reactively on further actions whether to install an appropriate rule for the packet. Also, the controller may be busy with other work and not respond immediately. Those packet processing actions may delay the update process. In the case buffer becomes full, for example, if the network is being congested, packets may be dropped. There, the transition to the next state during the update process depends on the likelihood of a delay or a loss of a packet in the current state.

In the example, the number of steps required for launching N_2 is smaller than the number of steps required for launching N_1. However, due to a higher likelihood of delays along the route N_2, it is possible that N_1 is preferred having a higher probability for a successful update. The network administrator may ask which new route is recommended for the update process, considering the expected number of steps required for the update. That is, updating paths requires the operator to decide on the possible choice of a subroute for the next step. One should consider the possibility of including a decision tool augmenting the controller during route updates.

There were many attempts to use the LP/ILP approach, as it was already mentioned above (see, e. g., [8]), but they have encountered the same difficulties, especially when taking into account online implementation. We show that it is possible to describe the routing process

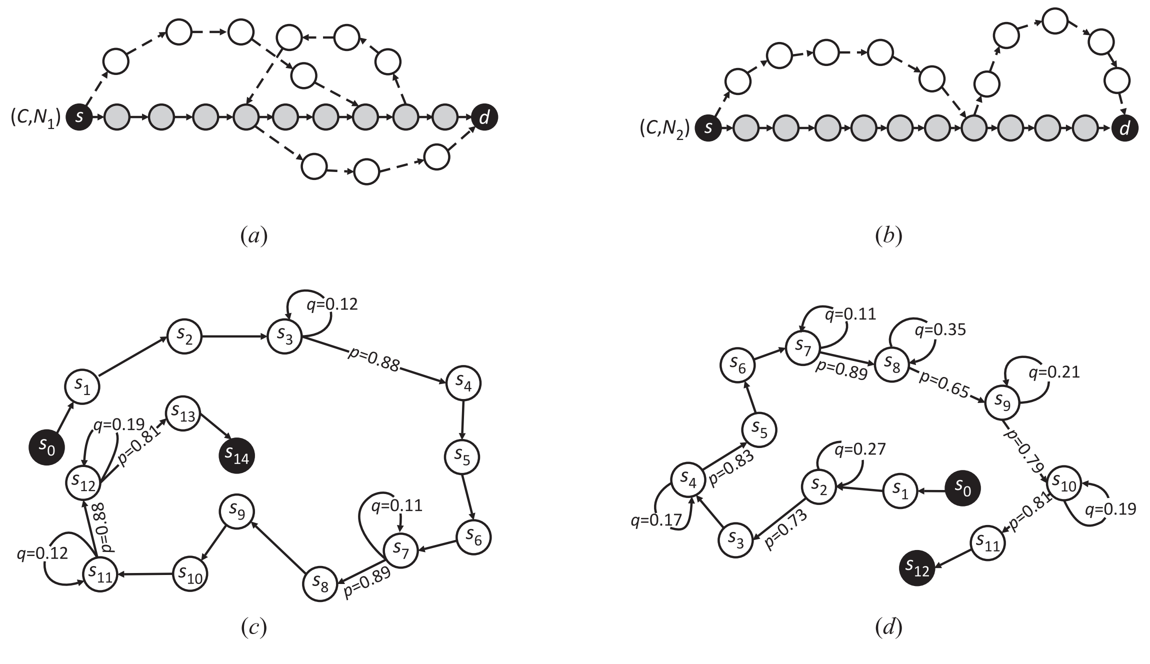

Figure 4 New routes N_1 (*a*) and N_2 (*b*) and MC states for N_1 (*c*) and N_2 (*d*)

as DTMC. Thus, taking into consideration $O(n^3)$ worst case computation complexity, we consider using the "design via verification" mentioned above based on PCTL verification, similar to the one used in PRISM [18].

We have calculated the probability for a successful update as a function of number of steps for routes N_1 and N_2 from the example in Fig. 4. See Fig. 3 where this function is shown. Curve *1* represents the plot for N_1 and curve *2* represents the plot for N_2.

Observe that after 20 steps, both new routes will be launched with probability 1 which can be written as

$$P_1 \left[F^{>20} N_1\right] = P_1 \left[F^{>20} N_2\right] = 1\,.$$

The expected number of steps required for N_1 is smaller than the required for N_2:

$$R\left[F\ N_1\right] < R\left[F\ N_2\right]\,.$$

However, the probability for successfully updating in less than 15 steps is higher for route N_2 (0.55 ± 0.040 for N_1 and 0.717 ± 0.036 for N_2, based on 99% confidence level): $P_{0.717\pm0.036}\left[F^{\leq 15} N_2\right]$.

5 Route Updates per Quality of Service

In this section, we show algorithm that we propose for various settings. First, we show an enhancement for the sequential update algorithm from [4], which during the update process decides on preferred subroute from the set of possible subroutes as part of QoS requirements. In the multicast-based update, several methods were proposed in [4] for eliminating duplicated packets. In the case the common destination router is not able to immediately eliminate duplicated packets, the algorithm begins the update from the end, ensuring a correct update process [4].

After that, we show an algorithm that chooses the subroutes for update arbitrary, assuming that the common destination node will not leak duplicated packets. However, the packets sending rate along the new subroute need to be temporarily limited [4].

At last, we present a supplementing algorithm that suggests which subroutes can be updated in parallel.

5.1 Sequential update

Let us begin the update from the end, namely, from the last alternative subroute of any new route. Provably, this prevents the formation of cycles [4]. In order to represent all possible choices of a path from a current state of the update process to the end of the update process, we propose to use a directed graph which nodes are the new, legitimate for launching, subroutes of the network. The edges of the graph represent a legal order of launching new subroutes. Each path in this graph from a current node to the last node in the path represents a legal combination of chosen subroutes. The update process is continued as long as there is a possible node to transition to.

Let us examine the two possible new routes N_1 and N_2 that can replace the current route C from the example depicted in Fig. 1. The new route N_1 is composed of N_1^1 and N_1^2, while the new route N_2 composed of N_2^1, N_2^2, N_2^3, and N_2^4. Starting from the end, the only new subroutes that are allowable to launch are N_1^2 and N_2^4. Assume that based on the DTMC calculations performed as described in section 4, the subroute N_2^4 is chosen for update. After the update of the subroute is complete, the current route C is composed of not updated yet part of the old route and N_2^4. See Fig. 2 where the change in C is depicted.

After the subroute N_2^4 is launched, we arrive at a smaller problem in which less subroutes are left to update. Due to dynamics of the system (see section 3), some new subroutes can merge into a single new subroute. See Fig. 2 where after N_2^4 was launched, the new subroutes N_2^3 and N_2^2 are merged into a single subroute. Now, one can launch either N_1^1 or N_2^2 merged with N_2^3. Assume that we choose to launch N_1^1, which launch finishes the update. The route C updated to N_1^1 and N_2^4. See Fig. 5 illustrating that.

Figure 6 shows the directed graph that represents the possible update sequences. Initially, the subroutes that are legal for launch are N_1^2 and N_2^4. As such, these are the only subroutes that have in-degree 0. Launching N_2^3 is forbidden; hence, there is no node in the graph G that represents this subroute. After launching N_2^4, we

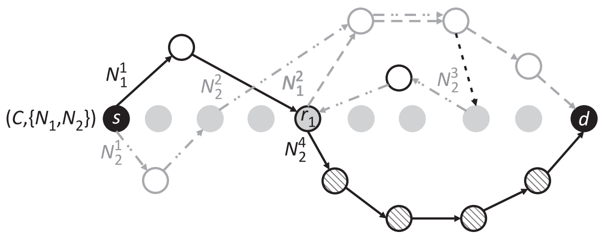

Figure 5 N_1^1 was launched

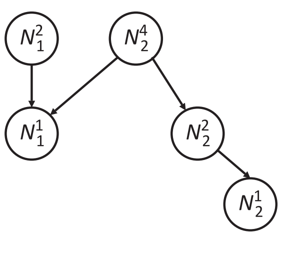

Figure 6 Graph representation for possible update paths for routes update example from Fig. 1

Algorithm 1: Update per QoS Algorithm

```
1  directed graph $G$
   /* $A$ is a collection of nodes                                  */
2  $A \leftarrow$ choose nodes from $G$ with in-degree 0
3  repeat
4      foreach $v \in A$ do
5          calculate $R[F\ v]$
6      end
7      $N_i^t \leftarrow \arg\max_v(R[F\ v])$
8      launch $N_i^t$
9      update $C$ accordingly
10     merge any new and common subroutes as described in section 3
11     $A \leftarrow$ choose nodes neighboring to $N_i^t$
12 until out-degree of node $N_i^t > 0$;
```

can proceed by launching N_1^1 or N_2^2. However, if N_1^2 was launched first, it would be forbidden to launch N_2^2 since it shares a common edge with N_1^2. This is reflected in the graph G by not having a directed edge from the node N_1^2 to the node N_2^2. We finish the update process by arriving either to N_1^1 or to N_2^1. Notably, these nodes have out-degree 0.

Algorithm 1 updates subroutes according to calculated QoS for each new subroute, by choosing at each step the new subroute that maximizes QoS.

The algorithm starts by selecting the initial set of subroute nodes. These are nodes with in-degree 0. The algorithm continues traversing the graph up to arrival at a node with out-degree 0 which would be the last subroute to launch. The inner loop at lines 4–6 calculates the QoS for each neighboring node. Afterward, at line 7, the algorithm chooses the node that maximizes QoS. Then launches this node and updates the route C, accordingly (see Figs. 1–5 for illustration). Afterward, the algorithm selects the next neighboring nodes.

After execution of Algorithm 1, the resulting new route maximally complies QoS requirements.

5.2 Arbitrary subroutes selection

In this subsection, we assume that immediate duplicate packets elimination is possible. It may be that some of the subroutes are not ready for an update yet. Thus, meanwhile, the administrator may want to proceed with the update process to other subroutes or see possible variations of the update. For such scenario, we provide an algorithm which can select a subroute for update arbitrary and continue the update process from there. We create a forest graph of all possible update combinations from which the desired update sequence can be chosen.

Figure 7 shows all possible combinations from example in Fig. 1. Noticeable, as mentioned earlier, some

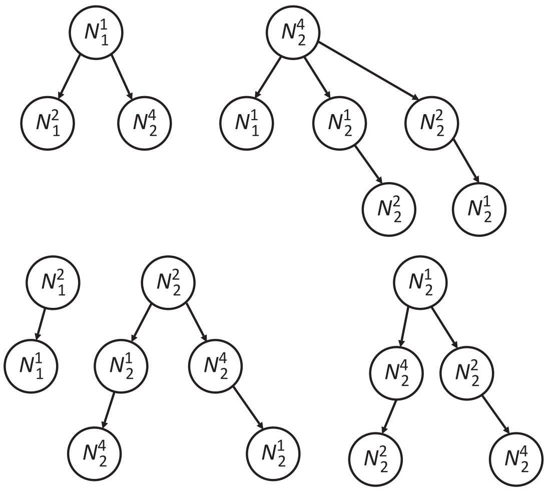

Figure 7 Forest graph representing execution combinations for example from Fig. 1

combinations exhibit fewer steps, though possible that its QoS compliance is worse than others.

Algorithm 2 starts by iterating over all roots of the forest graph and calculating QoS using Algorithm 1 each tree. Afterward, launch the update of the tree that maximizes QoS.

5.3 Parallel update

In certain cases, it is possible to update in parallel several subroutes and, as such, decrease update time. However, launching subroutes in parallel is not always possible since subroute may share a link and, thus, leads to congestion during the update process, close a cycle, or lead to an inconsistent state of the system. In [4], it was shown that two new subroutes N' from a to b and N'' from c to d can be launched in parallel only if c succeeds b or a succeeds d.

Algorithm 2: Arbitrary Selection Update

```
1 directed graph $G$
2 $A_0 \leftarrow$ choose nodes from $G$ with in-degree 0
3 $Q \leftarrow \{\}$
  /* iterate over all roots of trees in the forest $G$ */
4 foreach $v_r \in A_0$ do
5     $q \leftarrow$ get the expected QoS using Algorithm 1 for $v_r$
6     $Q \leftarrow Q \cup \{q \rightarrow \text{root}\}$
7 end
8 $q_{\max} \leftarrow \max_{\text{QoS}}(Q)$
9 launch maximizing QoS update order in root $= Q[q_{\max}]$
```

We create a supplementary graph G_S, in which nodes are the new legitimate for launching subroutes, and edges represent restrictions on parallel launching of subroutes. See Fig. 8 for illustration, depicting subroutes from example in Fig. 1 and their parallel restrictions. For example, N_2^4 and N_2^1 can be launched in parallel since there is no edge connecting them.

Clearly, any independent set of subroutes from the supplementary graph contains subroutes that can be launched in parallel. This can be further enhanced by setting QoS calculated values as weights on nodes of the graph and finding the subroutes that can be launched in parallel by finding a maximum-weight independent set of the graph G_S. Since G_S has few number of nodes (several tens), it is possible to find the maximum-weight independent set even by enumerating all possible independent sets [21] and comparing their total weights.

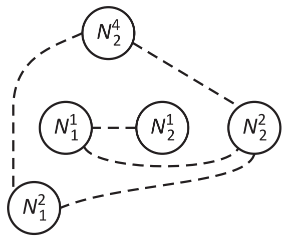

Figure 8 Supplementary graph of the example in Fig. 1, showing which subroutes cannot be run in parallel

Important, the parallel method should not be launched on its own. For example, assume that at the first iteration of Algorithm 3, the independent sets of nodes are A_1 and A_2. Let us assume that A_1 complies better to QoS demands than A_2 and, thus, A_1 will be selected. Also, let us assume that B_1 is the next independent set in the graph if A_1 was selected and B_2 if A_2 was selected. Also, let us assume that B_1 is the next independent set in the graph if A_1 was selected and B_2 if A_2 was selected. It is possible that due to the dynamics of the system (see section 3), we could obtain overall higher QoS results if we initially launched the subroutes from the sets A_2 and B_2 afterwards than from the sets A_1 and B_1.

Therefore, the graph that we create in this section for parallelization constraints is a supplementary graph which must be used in conjunction with the graphs from previous sections. Optimal results will be obtained when used in conjunction with the forest graph from subsection 5.2.

It is also important to note that, in the worst case, when there are no disjoint subroutes, the parallel method is reduced to the sequential method thought with a higher running time.

6 Implementation

We implemented the update algorithms from [4] as services for our QoS verification module. The update

Algorithm 3: Parallel Update

```
1 weighted graph $G_S$
2 while there are still current subroutes to update do
3     $A \leftarrow$ find maximum-weight independent set in $G_S$
      /* do in parallel */
4     foreach $N_i^t \in A$ do
5         launch $N_i^t$
6     end
7 end
```

algorithm itself was not modified. In other words, we treated the update itself as an atomic action. The route updates algorithms are implemented as applications interacting with the northbound interface of an SDN controller. We used POX [22] as a platform for controller development and Mininet [23] for network topology emulation. Figure 9 depicts the schematic arrangement of the functional elements.

We created networks with topology of random graph and small-world features. During each simulation trial, a pair of common source and destination nodes (s, d) were selected. A path connecting s and d was selected as a current route and a set of 4 new routes connecting (s, d), to replace the current route, were selected, possibly with shared links among themselves and the current route.

We considered latency due to the formed congestion as QoS demands for the update, implemented by forming congestion on randomly selected subroutes. Route update was executed by the update algorithm from [4] for each pair of current and new routes. Further, one of the enhanced versions was executed, updating to the preferred combination of subroutes, by identifying the congested subroutes (e. g., by estimating latency).

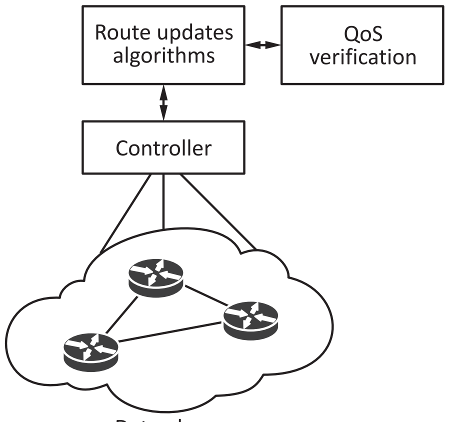

Figure 9 Description of the system

7 Concluding Remarks

The study in this paper illustrates a feasibility of modeling and designing the route update process via verification using DTMC. The goal was to strengthen the network administrator involvement in management and decision making during route update. In the present model, the network administrator is able to consider network parameters such as packet losses, delay, communication rounds, flow table updates, congestion, and other inherent unreliabilities of the network.

We extended the updating algorithm with the ability to compute QoS as the MC characteristics, where the MC corresponds to the states of the update process. Using this MC computation ability, it is possible to predict the expected number of steps (delay time) required to complete the update process. These prediction results allow the administrator to make a decision whether a new route can satisfy the user requirements per QoS or a more reliable route will be selected.

We provided sequential update algorithm and an arbitrary order algorithm when for the later, it is assumed that immediate duplicate packets elimination is possible. Further, we suggest a supplementary graph and algorithm for launching updates in parallel when it is possible.

This paper proposes a conceptual approach. In future research, we will focus on optimization of predictions supplementing the network administrator with a powerful tool which will be able to enhance the update process with fine grained analysis of the network.

Acknowledgments

The first author has partially been supported by the Russian Foundation for Basic Research under grants RFBR 18-07-00669 and 18-29-03100. The second author has partially been supported by the Rita Altura Trust Chair in Computer Sciences; The Lynne and William Frankel Center for Computer Science.

The authors thank Prof. Shlomi Dolev for his valuable input and Prof. Yefim Dinitz for his comments.

References

1. Rao, S. K. 2014. SDN and its use-cases — NV and NFV: A state-of-the-art survey. NEC Technologies India Ltd. 25 p.
2. Ghaznavi, M., N. Shahriar, R. Ahmed, and R. Boutaba. 2016. Service function chaining simplified. arXiv.org. arXiv:1601.00751.
3. Hansson, H., and B. Jonsson. 1994. A logic for reasoning about time and reliability. *Form. Asp. Comput.* 6(5):512–535.
4. Delaet, S., S. Dolev, D. Khankin, S. Tzur-David, and T. Godinger. 2015. Seamless SDN route updates. *IEEE 14th Symposium (International) on Network Computing and Applications*. IEEE. 120–125.
5. Frenkel, S., D. Khankin, and A. Kutsyy. 2017. Predicting and choosing alternatives of route updates per QoS VNF in SDN. *IEEE 16th Symposium (International) on Network Computing and Applications*. IEEE. 1–6.
6. Devi, G., and S. Upadhyaya. 2015. An approach to distributed multi-path QoS routing. *Indian J. Sci. Technol.* 8(20):1–14. doi: 10.17485/ijst/2015/v8i20/49253.
7. Egilmez, H. E., S. Civanlar, and A. M. Tekalp. 2012. A distributed QoS routing architecture for scalable video streaming over multi-domain OpenFlow networks. *19th IEEE Conference (International) on Image Processing*. IEEE. 2237–2240.

8. Juttner, A., B. Szviatovski, I. Mecs, and Z. Rajko. 2001. Lagrange relaxation based method for the QoS routing problem. *IEEE Conference on Computer Communications. 20th Annual Joint Conference of the IEEE Computer and Communications Society Proceedings*. IEEE. 2:859–868.
9. Yu, Z., F. Ma, J. Liu, B. Hu, and Z. Zhang. 2013. An efficient approximate algorithm for disjoint QoS routing. *Math. Probl. Eng.* 2013:489149. 9 p. doi: 10.1155/2013/489149.
10. Foerster, K.-T., S. Schmid, and S. Vissicchio 2016. A survey of consistent network updates. Arxiv.org. arXiv: 1609.02305.
11. Reitblatt, M., N. Foster, J. Rexford, and D. Walker. 2011. Consistent updates for software-defined networks: Change you can believe in! *10th ACM Workshop on Hot Topics in Networks Proceedings*. New York, NY: ACM. Art. No. 7. doi: 10.1145/2070562.2070569.
12. Hogan, M., and F. Esposito. 2017. Stochastic delay forecasts for edge traffic engineering via Bayesian networks. *IEEE 16th Symposium (International) on Network Computing and Applications*. IEEE. 1–4.
13. McGeer, R. 2012. A safe, efficient Update Protocol for Openflow Networks. *1st Workshop on Hot Topics in Software Defined Networks Proceedings*. New York, NY: ACM. 12:61–66.
14. McGeer, R. 2013. A correct, zero-overhead protocol for network updates. *2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking Proceedings*. New York, NY: ACM. 13:161–162.
15. Katta, N. P., J. Rexford, and D. Walker. 2013. Incremental consistent updates. *2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking Proceedings*. New York, NY: ACM. 13:49–54.
16. Dinitz, Y., S. Dolev, and D. Khankin. 2017. Dependence graph and master switch for seamless dependent routes replacement in SDN. *IEEE 16th Symposium (International) on Network Computing and Applications*. IEEE. 1–7.
17. Amiri, S. A., S. Dudycz, S. Schmid, and S. Wiederrecht. 2016. Congestion-free rerouting of flows on DAGs. ArXiv.org. arXiv:1611.09296.
18. Kwiatkowska, M., G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of probabilistic real-time systems. *Computer aided verification*. Eds. G. Gopalakrishnan and S. Qadeer. Lecture notes in computer science ser. Springer. 6806:585–591.
19. Kwiatkowska, M., G. Norman, and D. Parker. 2018. PRISM manual. Available at: http://www.prismmodelchecker.org/manual/ (accessed December 10, 2018).
20. Open Networking Foundation. 2015. OpenFlow Switch Specification Ver 1.5.1.
21. Wu, Q., and J.-K. Hao. 2015. A review on algorithms for maximum clique problems. *Eur. J. Oper. Res.* 242(3):693–709.
22. Kaur, S., J. Singh, and N. S. Ghumman. 2014. Network programmability using POX controller. *Conference (International) on Communication, Computing and Systems*. 138.
23. Lantz, B., B. Heller, and N. McKeown. 2010. A network in a laptop: Rapid prototyping for software-defined networks. *9th ACM SIGCOMM Workshop on Hot Topics in Networks Proceedings*. New York, NY: ACM. Art. No. 19. doi: 10.1145/1868447.1868466.

Received October 9, 2018

Contributors

Frenkel Sergey L. (b. 1951) — Candidate of Science (PhD) in technology, associate professor, senior scientist, Institute of Informatics Problems, Federal Research Center "Computer Sciences and Control" of the Russian Academy of Sciences, 44-2 Vavilov Str., Moscow 119333, Russian Federation; fsergei51@gmail.com

Khankin D. (b. 1983) — MSc, doctorate student, Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel; danielkh@post.bgu.ac.il

НЕПРЕРЫВНЫЕ ОБНОВЛЕНИЯ МАРШРУТА В SDN С ИСПОЛЬЗОВАНИЕМ ПРОВЕРКИ СООТВЕТСТВИЯ КАЧЕСТВУ ОБСЛУЖИВАНИЯ[*]

С. Л. Френкель[1], Д. Ханкин[2]

[1]Институт проблем информатики Федерального исследовательского центра «Информатика и управление» Российской академии наук

[2]Университет им. Бен-Гуриона в Негеве, Беэр-Шева, Израиль

Аннотация: В программно-определяемой сети (SDN — software-defined networking) уровень управления и уровень данных разделены. Это обеспечивает высокую гибкость эксплуатации, предоставляя абстракции для управления сетью приложений и возможность непосредственного программирования маршрутов. Однако из-за изменений топологии, процедуры обслуживания или происходящих сбоев иногда необходима реконфигурация и обновление сети. В предлагаемом сценарии рассматривается текущий маршрут C и набор возможных новых маршрутов $\{N_i\}$, где для замены текущего маршрута требуется один из

[*]Работа была частично поддержана РФФИ (гранты 18-07 00669 и 18-29-03100), а также Rita Altura Trust Chair in Computer Sciences; The Lynne and William Frankel Center for Computer Science.

новых маршрутов. Существует вероятность того, что новый маршрут N_i окажется длиннее некоторого другого нового маршрута N_j, но при этом N_i будет более надежным и он будет обновляться быстрее или работать лучше после обновления с точки зрения требований качества обслуживания (QoS — quality of service). Принимая во внимание случайный характер функционирования сети, авторы дополнили недавно предложенный алгоритм обновления маршрута Delaet с соавт. методом оценки соблюдения требований QoS во время непрерывного обновления маршрута, основанным на использовании цепей Маркова. При этом, во-первых, предлагается расширить алгоритм передачи пакетов по выбранному маршруту, сравнивая процесс обновления для возможных альтернатив маршрута. Во-вторых, предлагается несколько способов выбора комбинаций предпочтительных отрезков путей новых маршрутов, что приводит к оптимальному в смысле соответствия QoS маршруту.

Ключевые слова: программно-определяемые сети; цепи Маркова; качество обслуживания

DOI: 10.14357/19922264180408

Литература

1. *Rao S. K.* SDN and its use-cases — NV and NFV: A state-of-the-art survey. — NEC Technologies India Ltd., 2014. 25 p.
2. *Ghaznavi M., Shahriar N., Ahmed R., Boutaba R.* Service function chaining simplified // Arxiv.org, 2016. arXiv:1601.00751cs.
3. *Hansson H., Jonsson B.* A logic for reasoning about time and reliability // Form. Asp. Comput., 1994. Vol. 6. No. 5. P. 512–535.
4. *Delaet S., Dolev S., Khankin D., Tzur-David S., Godinger T.* Seamless SDN route updates // IEEE 14th Symposium (International) on Network Computing and Applications. — IEEE, 2015. P. 120–125.
5. *Frenkel S., Khankin D., Kutsyy A.* Predicting and choosing alternatives of route updates per QoS VNF in SDN // IEEE 16th Symposium (International) on Network Computing and Applications. — IEEE, 2017. P. 1–6.
6. *Devi G., Upadhyaya S.* An approach to distributed multipath QoS routing // Indian J. Sci. Technol., 2015. Vol. 8. Iss. 20. P. 1–14. doi: 10.17485/ijst/2015/v8i20/49253.
7. *Egilmez H. E., Civanlar S., Tekalp A. M.* A distributed QoS routing architecture for scalable video streaming over multi-domain OpenFlow networks // 19th IEEE Conference (International) on Image Processing. — IEEE, 2012. P. 2237–2240.
8. *Juttner A., Szviatovski B., Mecs I., Rajko Z.* Lagrange relaxation based method for the QoS routing problem // IEEE INFOCOM 2001 Conference on Computer Communications. 20th Annual Joint Conference of the IEEE Computer and Communications Society Proceedings. — IEEE, 2001. Vol. 2. P. 859–868.
9. *Yu Z., Ma F., Liu J., Hu B., Zhang Z.* An efficient approximate algorithm for disjoint QoS routing // Math. Probl. Eng., 2013. Vol. 2013. Art. No. 489149. 9 p. doi: 10.1155/2013/489149.
10. *Foerster K.-T., Schmid S., Vissicchio S.* A survey of consistent network updates // Arxiv.org, 2016. arXiv:1609.02305.
11. *Reitblatt M., Foster N., Rexford J., Walker D.* Consistent updates for software-defined networks: Change you can believe in! // 10th ACM Workshop on Hot Topics in Networks Proceedings. — New York, NY, USA: ACM, 2011. Art. No. 7. doi: 10.1145/2070562.2070569.
12. *Hogan M., Esposito F.* Stochastic delay forecasts for edge traffic engineering via Bayesian Networks // IEEE 16th Symposium (International) on Network Computing and Applications. — IEEE, 2017. P. 1–4.
13. *McGeer R.* A safe, efficient Update Protocol for Openflow Networks // 1st Workshop on Hot Topics in Software Defined Networks Proceedings. — New York, NY, USA: ACM, 2012. Vol. 12. P. 61–66.
14. *McGeer R.* 2013. A correct, zero-overhead protocol for network updates // 2nd Workshop on Hot Topics in Software Defined Networking Proceedings. — New York, NY, USA: ACM, 2013. Vol. 13. P. 161–162.
15. *Katta N. P., Rexford J., Walker D.* Incremental consistent updates // 2nd Workshop on Hot Topics in Software Defined Networking Proceedings. — New York, NY, USA: ACM, 2013. Vol. 13. P. 49–54.
16. *Dinitz Y., Dolev S., Khankin D.* Dependence graph and master switch for seamless dependent routes replacement in SDN // IEEE 16th Symposium (International) on Network Computing and Applications. — IEEE, 2017. P. 1–7.
17. *Amiri S. A., Dudycz S., Schmid S., Wiederrecht S.* Congestion-free rerouting of flows on DAGs // ArXiv.org, 2016. arXiv:1611.09296.
18. *Kwiatkowska M., Norman G., Parker D.* PRISM 4.0: Verification of probabilistic real-time systems // Computer aided verification / Eds. G. Gopalakrishnan, S. Qadeer. — Lecture notes in computer science ser. — Springer, 2011. Vol. 6806. P. 585–591.
19. *Kwiatkowska M., Norman G., Parker D.* PRISM manual, 2018. http://www.prismmodelchecker.org/manual.
20. Open Networking Foundation. OpenFlow Switch Specification Ver 1.5.1, 2015.
21. *Wu Q., Hao J.-K.* A review on algorithms for maximum clique problems // Eur. J. Oper. Res., 2015. Vol. 242. No. 3. P. 693–709.
22. *Kaur S., Singh J., Ghumman N. S.* Network programmability using POX controller // Conference (International) on Communication, Computing and Systems, 2014. P. 138.
23. *Lantz B., Heller B., McKeown N.* A network in a laptop: Rapid prototyping for software-defined networks // 9th ACM SIGCOMM Workshop on Hot Topics in Networks Proceedings. — New York, NY, USA: ACM, 2010. Art. No. 19. doi: 10.1145/1868447.1868466.

Поступила в редакцию 09.10.2018