MULTICAST AND CONTROL CLEARANCE BASED SDN ROUTE UPDATE

by

Sylvie Delaet, Shlomi Dolev, and Shimrit Tzur-David

Technical Report #15-01

January 2015

Multicast and Control Clearance Based SDN Route Update

(Technical Report – Preliminary Version)

Sylvie Delaet* Shlomi Dolev† Shimrit Tzur-David‡

Abstract

Software-Defined Networking (SDN) decouples the control and data planes, enabling limitless possibilities for implementing services and applications on top of the network abstraction layer. The centralized controller provides a real-time view of the entire underlying network infrastructure and therefore, management of the agile network becomes more simplified. This flexibility requires online routing updates, but during these updates the consistency has to be saved, i.e. no packet losses or unrecognized duplications. In this paper we handle the consistency of the routing updates. The main idea is to use multicast on portions of the route, i.e., to send a packet both in the old and new route and only when the controller verifies the specific portion of the new route, it can remove the corresponding portion from the old route.

Keywords: SDN, Routing updates

*University of Paris, France. Email: `sylvie.delaet@lri.fr`.
†Department of Computer Science, Ben-Gurion University of the Negev, Israel. Email: `dolev@cs.bgu.ac.il`.
‡Department of Computer Science, Ben-Gurion University of the Negev, Israel. Email: `shimritd@gmail.com`.

1 Introduction

Software defined network (SDN) is a new and emerging technology that introduces new challenges. In SDN, the network control is decoupled from the forwarding functions, this enables the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. An SDN controller is an application that manages flow control to enable intelligent networking.

Maybe the most significant advantage of SDN is its flexibility, i.e., it is easy for programmers to make frequent changes to networks. However, the consistency of these changes is a major vulnerability.

As indicated in [**?**], 20% of all failures is due to planned maintenance routing activities, such as updates. In order to deal with these failures, there have been developed several extensions to some specific protocols, such as BGP and OSPF [**? ? ? ? ? ?**]. Apart from the complexity issues of these solutions, they were designed for specific protocols and none of them supply a complete solution.

Due to the flexibility of SDN, this problem even exacerbated and therefore one of the most important challenges is to find a way for the controller to update routes in the network in a consistent manner. Consider a route from a source s to a destination d, that has to be changed, possibly due to quality of service requirements of the connection session between s and d or the current status of the network. An attempt of the controller to remove the relevant entries from the routing tables of the routers along the current route and then, when done, establish the new route by instructing the routers along the new route to have an entry for the s to d connection, will result in stopping the connection for a while, and possibly losing packets that are already traversing the original route.

Another approach is to try to establish an "atomic" change by synchronizing clocks e.g., [**?**], and make the change simultaneous. Beyond the impossibility for achieving perfect synchronization, there is still a risk to loss the packets that are already started traversing the original route from s to d, as the forwarding information in the original route is lost at once.

We propose new approaches to the problem of consistent multi-switch updates for SDN. Our approach is based on examining the operational status of the new route prior to dismantling the current route. In contrast to other works, we do not guarantee that every packet in the set exclusively uses either the old policy or the new policy; Moreover, we show that by using a combination of the two routes, we gain a consistent as well as better performed solution.

First we use (a careful) muti-cast where the new route information is inserted to the routing tables and packets are duplicated to traverse (portions of) the original and the new routes. We demonstrate that such a multi-cast on *all* joint nodes may results in exponential number of packets and even cycles. Then we present a multi-cast based algorithm for changing the routing while keeping the connection operational throughout the change, without losing packets. Our solution preserves the (in some cases only eventual) FIFO ordering of the packets, while keeping the total number of packets in the same order as the number of packets in both the routs when each operates alone, namely, before and after the route update.

Then we use a controller examination of the readiness of the new route, by means of sending packets from the controller to the controller over the new route (portion) and when ready, switch to the new route (portion) dismantling the current route only when all traversing packets are forwarded to the destination.

Next we overview the related work, in Section 3 we describe the assumed NDS settings. Then in Section 4 and Section 5 we present the multicast and the control clearness based solutions, respectively.

2 Related Work

The SDN controller has to perform frequent configuration updates in the network. Of course, these updates should not cause packet drops or misroutes that cause inconsistencies. In [**?**] the authors present general abstractions for managing network updates. The authors present the notion of "per-packet consistency", where every packet traversing the network is processed by exactly one consistent global network configuration. In our notation, this single configuration can be a mixture of the two configurations, i.e., the configuration in place prior to the update, and the configuration in place after the update. We show that merging these two configuration in some specific points in the paths, we still get consistency. Furthermore, there can be cases where there are two copies of a single packet, one copy follows (portion of) the route in the configuration in place prior to the update and the second copy follows (portion of) the route that is configured after the update.

An extensive research was done on the topic of network update. In [**?**], the author presents a paradigm in which each packet that affected by the change is sent to the controller. When all switches have been updated to send affected packets to the controller, the update is performed and all packets that were sent to the controller are re-released into the network. This work suffers from cost in control plane bandwidth, and end-to-end latency for affected network flows.

The work presented in [**?**] describes an algorithm that stamps packets with a configuration version at ingress switches and tests for the version number in all other rules. The controller is required to install the new rules on all of the switches, leaving also the rules for the old configuration in place. The controller deletes the old rules only after some maximum transmission delay that depends on the sum of propagation and queuing delay, maximized over all paths, has elapsed. Note that in case of several updates in a short period, there can be cases where each switch holds several rules configurations. The switch rule-space resources might be very expensive, especially when using a memory such a TCAM. Another work, presented in [**?**], seeks to find a safe, switch-update ordering. Since such order does not always exist, in case there is no saved update, also this work uses configuration versioning and two phase update as in [**?**].

[[As [**?**], the work presented in [**?**] also derives an algorithm to find a safe update sequence. The author expresses this sequence as a logic circuit and he shows that there exists a consistency preserving update if and only if the circuit can be satisfied.]]

Mizrahi et al. [**?**] present a time-based configuration method that can be used to invoke simultaneous updates. The presented method does not require modification of traversing packets but is suffers from inconsistency for some period when packets are en-route during the configuration update.

3 Software Defined Network Settings

Consider two routes, the new one (navy) and the current one (champagne) as illustrated in Figure 1. s is the common source and d the common destination of these two routes. Let $c_0 = s, c_1, \ldots, c_i, c_{i+1}, \ldots, c_l = d$ be the routers along the original route, where, for each router c_i on the route, all the flows that the couple (s,d) is in their matched fields have an entry in the router's routing table, with the forwarding instruction c_{i+1}.

The SDN controller decides to update the current route to the new route, while the current route is operational and the new route is still virtual.

The update from the current route to the new route must ensure that after the update is completed we have a path $n_0 = s, n_1, \ldots, n_i, n_{i+1}, \ldots, n_m = d$ such that for each router n_i on this route, all the flows with (s, d) in their matched fields has the instruction to be sent to n_{i+1}; and that only the n_i nodes of the network have these flows in their routing tables. The constraint that we need to ensure during the update is that the sender does not stop sending packets, the receiver does not stop receiving packets, no packets are lost and (when duplicates are removed) packets (eventually) arrive in a FIFO order.

In the presence of possible duplication and reordering of packets, we distinguish two kinds of packet sequences. In the first one we present a memoryless procedure where it is sufficient to remove duplicates in order to obtain a FIFO sequence; Note that we not necessarily drop the second copy. In case of an arrival of not-in-order packet, the algorithm can drop the first copy if it can ensure that the copy arrives through the new route. In the second kind of packet sequences, memory is required for reordering. For example, the sequence p_1, p_1, p_2, p_3, p_2 can be converted to p_1, p_2, p_3 by dropping all arriving packets that are late. In contrast, the sequence p_1, p_3, p_1, p_2, p_2 implies the need to store p_3 until p_2 arrives, in order to obtain FIFO output. In the sequel we present the multi-cast based technique that produces a sequence with duplications that can be converted to FIFO exactly once sequence by memoryless procedure and the control clearance based technique that requires (bounded) memory (possibly at the destination) to obtain a FIFO exactly once sequence.

4 Multicast Based SDN Route Update

In SDN, the controller can delete, add or replace a (s,d) entry of a routing table. We base the update on multi-cast, namely, the controller can instruct a router to (temporally) have two (s,d) entries in the routing table in addition to its ability to instruct to delete, replace, or create a (s,d) entry of a router r. The packet transmission along a certain path is FIFO, namely, if two packets p_1 and p_2 are routed along the same path and both traverse a router r_1 and then r_2, then if p_1 traverses r_1 prior to p_2 then p_1 must traverse r_2 prior to p_2. When a router $r \neq d$ receives a packet from s to d it sends it according to the forwarding instructions of all of its (s, d) routing table entries.

The controller communicates with this router in an asynchronous way. Thus, we propose several ways to support an indication on a completion of a new (portion of the) route construction. One such technique is to identify the arrival of two identical packets to a certain router, i.e., packet that was multi-casted over the current and new routes. When an indication on the arrival of the two identical packets arrives, the controller can dismantle the current route, as the new route is ready. We note that the controller may instruct a router to send to the controller a copy of every packet received in a certain connection, simply by adding an additional entry to the routing table of the router tagged with the relevant connection and instruction to (also) forward the packets to the controller. This way the router does not need to identify duplications, the controller identifies them instead. Thus, the standard operation of the router as well as the standard OpenFlow commands can support the required operations. Furthermore, by adding IP timestamp option to each packet, it is easy to identify a retransmission of a packet rather than a second copy of the same packet that was sent by multicast.

We start presenting a simple version of our algorithm, where only a single portion of the new (s,d) path is established and replaces the current equivalent portion at a time. We later present criteria for applying our technique in parallel.
Notation: Lets consider a route from s to d $P \equiv p_0 = s, p_1, \ldots, p_i, p_{i+1}, \ldots, p_m = d$. Lets consider two nodes α and β such that $\alpha \neq \beta$ and $\alpha \in P$, $\beta \in P$. We note $[\alpha, \beta]_P$ the sub-route from α to β relatively to P. $[\alpha, \beta]_P \equiv p_j = \alpha, p_{j+1}, \ldots, p_i, p_{i+1}, \ldots, p_k = \beta$.

Algorithm 1: Sequential Replacement

```
1 Input $N \equiv n_0 = s, n_1, \ldots, n_i, n_{i+1}, \ldots, n_l = d$, $C \equiv c_0 = s, c_1, \ldots, c_i, c_{i+1}, \ldots, c_m = d$;
2 let $N' = [s, d]_N$, $C' = [s, d]_C$
3 while $C' \neq N$
4   let $NIC \equiv \{nic_0, nic_1, \ldots nic_x\}$ be the set of all the routers each appears in both $N'$ and $C'$, ordered according to $N$
5   compute $y$ to be the highest index in $\{0, x-1\}$ such that $[nic_y, nic_{y+1}]_{N'} \neq [nic_y, nic_{y+1}]_{C'}$
6   add the sub-route $[nic_y, nic_{y+1}]_{N'}$
7   wait until two identical packets arrive to $nic_{y+1}$
8   remove sub-route $[nic_y, nic_{y+1}]_{C'}$
9   $N' = [s, nic_y]_{N'}$, $C' = [s, nic_y]_{C'} \cup [nic_y, d]_{N'}$
```

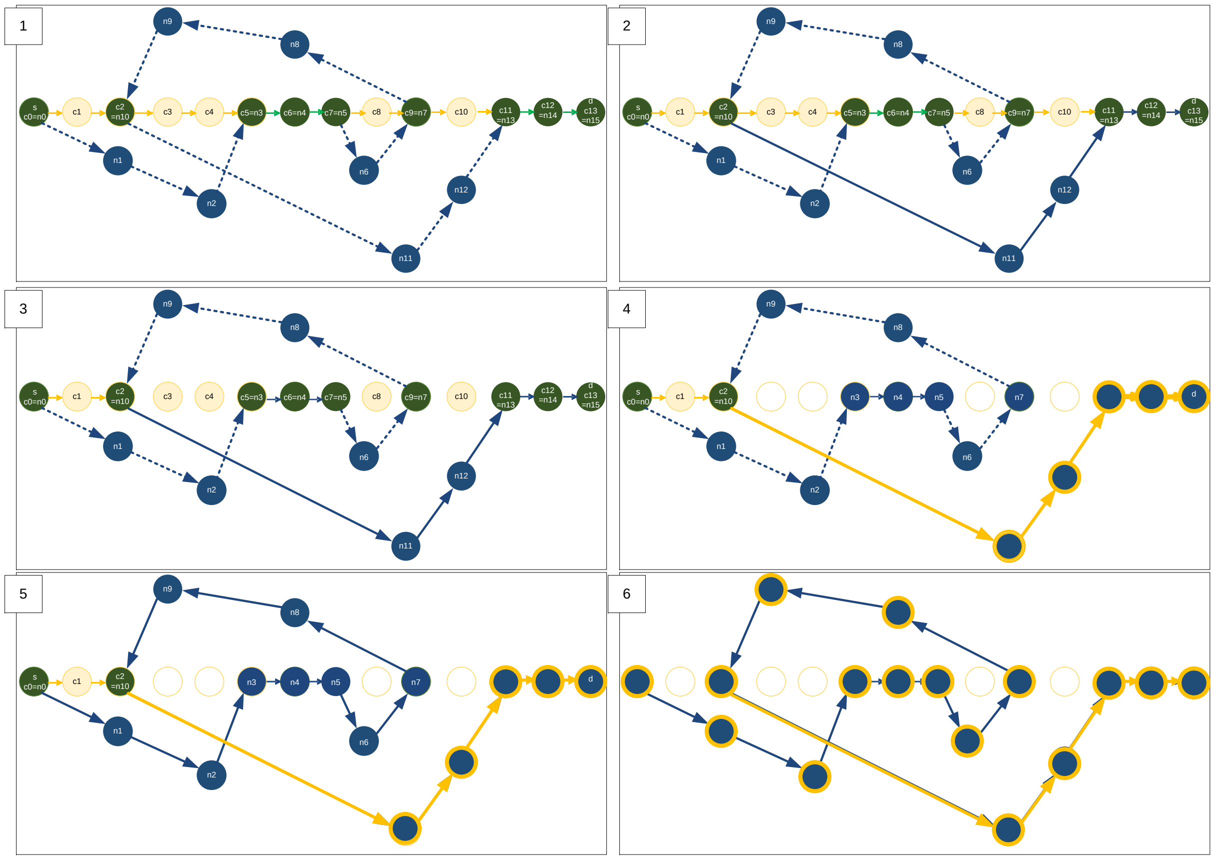

Figure 1: Algorithm 1 Execution Sample

Algorithm 1 starts with the new and the current routes, N and C as inputs and computes the intersection routers set NIC to be the set of all the routers that appear in both routes, ordered according to N. Then the algorithm replaces the segment of the current route C different from the one of the new route N, starting in the last intersection, namely $[nic_{last}, d]_C$, where NIC_{last} is the last intersection according to N. This is done by adding the above new sub-route of N, in particular instructing nic_y to send two copies of the (s, d) packets, one to the next hop according to C and the other to the next hop according to N. Then waiting for two identical packets to arrive to nic_{y+1} (or, in the last segment, to d). Once such two packets arrive, we have an indication that both sub-routes are operating, so we can remove the sub-route of C that starts in nic_y and ends in nic_{y+1}. Then nic_y becomes the actual new destination, and the procedure repeats, until nic_y is identical to s.

Note, that at any stage, nic_{y+1} does not need to buffer any of the packets, either the packets arrive in-order and it throws the second copy of the duplicated packet, or, the copy that is sent through the new route arrives before a packet with a lower sequence number, e.g., P_1 is sent through nic_y, then the route $[nic_y, nic_{y+1}]$ is added and P_2 is duplicated and sent over both route from nic_y to nic_{y+1}. The copy of P_2 that is sent through the new route, reaches nic_{y+1} before P_1. Since, assuming there are no losses, getting not in-order packets, necessarily means that a duplicated copy arrives earlier, nic_{y+1} can throw this not in-order copy. The router that duplicate the packets, keeps sending one copy at each route until the controller updates its forwarding table. Therefore, even in case of losses, the algorithm eventually updates the route when the next router gets two copies of *some* packet.

Figure 1 depicts an execution sample in six stages. In the first stage both the current (champagne colored) and the new (colored navy) are marked, where the operating current route has full arrows between the routers, while the new virtual route has dashed arrows. The intersection nodes of the current and new routes are colored green, and the last segment of the new route (between the last two intersection nodes) has bold dashed line, which is selected to replace the segment in the current route, that is between these two intersection nodes. Stage 2 of the figure depicts the co-existence of the two segments, where the bold dashed line becomes a solid line. Then, part 3 of the figure follows the arrival of a duplicate packets to the destination and the removal of the corresponding segment in the current route. Part 4 recalculates the current route to have the new suffix. The new last alternative segment is computed and removed in stages 5 and 6, respectively.

Theorem 1 *Algorithm 1 is correct. The algorithm is correct if all the packets that are sent from s arrive to d, also during the update process from C to N and that by the end of the process, only the new route exists. We assume that both C and N do not contain cycles.*

Proof. We prove the correctness of the algorithm by induction on the number of routers in the NIC set, $|NIC|$.

Basis: $|NIC| = 2$, $NIC = s, d$. There are no intersections in the paths C and N. In that case $y = s$. After step (5) in the algorithm, there are two disjoint paths from s to d, (1) $(s, y)_C \cup (y, d)_C$, and (2) $(s, y)_C \cup (y, d)_N$. In the base case, since $s = y$, these paths become $(s, d)_C$ and $(s, d)_N$. Assuming two identical packets are sent from s and all the sent packets arrive to d, d will get two identical packets. Then, at step (7), the old route $(y, d)_C$ is removed, in our case, the whole old route C. The algorithm is correct for $|NIC| = 2$.

Inductive step: We show that if the algorithm is correct when $|NIC| = k$, it is also correct when $|NIC| = k + 1$, i.e. we assume the correctness of the algorithm for $|NIC| = k$ and we prove it for $|NIC| = k + 1$. Let $\overline{N}$ and $\overline{C}$ be the routes $[nic_1, d]_N$ and $[nic_1, d]_C$ respectively (in the example above, $\overline{N} = [n_3, d]$ and $\overline{C} = [c_5, d]$). The NIC set of these two routes contains k routers namely, $NIC = \{nic_1, nic_2, \ldots nic_{k+1}\}$, by the induction hypothesis, the algorithm is correct for these routes. Now lets' look at the routes $(s, nic_1)_C$ and $(s, nic_1)_N$ ((s, n_3) and (s, c_5) in the example), these routes are disjoint paths and therefore, the algorithm is correct also for them. Constructing this portion of the route is the last step of the algorithm. ■ ■

Algorithm 2: Parallel Replacement

```
1  Input $N \equiv n_0 = s, n_1, \ldots, n_i, n_{i+1}, \ldots, n_l = d$, $C \equiv c_0 = s, c_1, \ldots, c_i, c_{i+1}, \ldots, c_m = d$;
2  let $C' = [s, d]_C$
3  while $C' \neq N$
4    let $NIC \equiv \{nic_0, nic_1, \ldots nic_x\}$ be the set of all the routers each appears in both $N$ and $C'$, ordered according to $N$
5    compute a subset $I$ of indices in $\{0, x-1\}$ such that:
6      $\forall y \in I$ $[nic_y, nic_{y+1}]_{N'} \neq [nic_y, nic_{y+1}]_{C'}$ and
7      $\forall z, y \in I$, $z \neq y$, $[nic_z, nic_{z+1}]_{C'} \cap [nic_y, nic_{y+1}]_{C'} = \emptyset$ and
8      there are no $z, y \in \{0, x-1\}$ such that $\exists$ $[nic_z, nic_y]_N$ and $[nic_y, nic_z]_C$
9    $\forall y \in I$, do in parallel:
10       add the sub-route $(nic_y, nic_{y+1})$ of $(s, d)_{N'}$
11       wait until two identical packets arrive to $nic_{y+1}$
12       remove sub-route $[nic_y, nic_{y+1}]_{C'}$
13   $C' = \bigcup_{y \in I} [nic_y, nic_{y+1}]_N \cup \bigcup_{y \notin I} [nic_y, nic_{y+1}]_{C'}$
```

Algorithm 2 finds in a greedy manner all the possible sub-routes in N that can replace sub-routes in C. Starting from the beginning of N, the algorithm finds whether the sub-route $[nic_1, nic_2]_N$ can replace at this stage the sub-route $[nic_1, nic_2]_C$, then checks whether the sub-route $[nic_2, nic_3]_N$ can replace at this stage the sub-route $[nic_1, nic_2]_C$ and so on and so forth, until it checks whether the sub-route $[nic_{x-1}, nic_x]_N$ can replace at this stage the sub-route $[nic_{x-1}, nic_x]_C$. A sub-route from nic_i to nic_{i+1} can be replaced by our multi-cast scheme *if and only if* the index of nic_i in C is smaller than the index of nic_{i+1} in C (no cycles are introduced) and $[nic_i, nic_{i+1}]_C$ is not in a process of replacement already.

Whenever one of the sub-route replacement is finalized, the definition of N' and C' is changed to represent this entire sub-route as a common single router, and the algorithm is applied again for sub-routes that are still not replaced.

5 Controller *Clearness Packet* Technique

In this section we describe an algorithm that does not produce duplicate data packets on the process of updating the routes. In this technique, for each sub-route the controller first construct a new sub-route $[nic_{x-1}, nic_x]_N$ to transmit its own packets, namely, from the controller to the controller through the new sub-route. By getting its own packets, the controller can get an indication that the new sub-route is ready.

Then the controller direct all the packets to go through the new sub-route instead of the current route, this is done, by removing the entry (s, d) from nic_{x-1}, all other routers in the current route still maintain the (s,d) entry for packets that currently are traversing the current route. In order to be sure that all these *on-the-fly* packets from the current route arrive to nic_x before releasing this route, the controller adds a ($cntrl$,$cntrl$) entry to each router in the current route after the existing entry of (s, d), then the controller sends it own packet also in the current route. Assuming the all routers send packet in the same order of their forwarding table entries, when the controller packet arrive to nic_x, it can be sure that all packets that traverse the current path when the new route was established, arrive to nic_c and therefore, the current sub-route can be dismantled. The idea to ensure that when a packet created by the controller arrives to the controller, the controller can be sure that it can now dismantle the current route.

Algorithm 3: Controler Cleareness Packet Technique

```
1  Input $N \equiv n_0 = s, n_1, \ldots, n_i, n_{i+1}, \ldots, n_l = d$, $C \equiv c_0 = s, c_1, \ldots, c_i, c_{i+1}, \ldots, c_m = d$;
2  let $N' = N$, $C' = C$
3  while $C'$ is not empty
4      let $NIC \equiv nic_0, nic_2, \ldots nic_x$ be all the routers each appears in both $N'$ and $C'$, ordered according to $N'$
5      for each router $r \in [nic_{x-1}, nic_x]_{N}$‘, add entry with the tag ($cntrl$,$cntrl$) according to $N$‘
6      send a packet from $nic_{x-1}$ to $nic_x$ to be forwarded over the sub-route defined by the entries ($cntrl$,$cntrl$) and wait
   for the packet arrival
7      after packet arrival    for each router $r \in [nic_{x-1}, nic_x]_{N}$‘, remove the entry with the tag ($cntrl$,$cntrl$) according
   to $N$‘
8          for each router $r \in [nic_{x-1}, nic_x]_{N}$‘, add entry with the tag ($s$,$d$) according to $N$‘
9          remove the entry $(s, d)$ of $C$‘ from $nic_{x-1}$    for each router $r \in [nic_{x-1}, nic_x]_{C}$‘, add entry with the tag
   ($cntrl$,$cntrl$) according to $C$‘
10         send a packet from $nic_{x-1}$ to $nic_x$ to be forwarded over the sub-route defined by the entries ($cntrl$,$cntrl$) and wait
   for the packet arrival
11         after packet arrival    for each router $r \in [nic_{x-1}, nic_x]_{C}$‘, remove ($s$,$d$) and ($cntrl$,$cntrl$) entries
12     $N' = \texttt{prefix}(N', nic_{x-1})$, $C' = \texttt{prefix}(C', nic_{x-1})$
```

Algorithm 3 uses ($cntrl$,$cntl$) packets, where first the controller set a double sub-route along the current and new routes then get an indication that the new is functioning, redirect the (s,d) traffic to the new, and redirect its traffic from it to itself via the current sub-route to clear leftovers before removing the sub-route of the current route.

Note that a similar technique can be used in all algorithms to ensure the completion of dismantling a segment, instructing each router in the segment to first dismantle and then create an entry of $(cntrl, cntr)$ for the segment, when a $(cntrl, cntrl)$ packet traverses the segment, the original segment is completely dismantled.

The parallel version of Algorithm 3 is analogous to Algorithm 2 where we examine segments between any two sequential intersection (according to the ordering of N) and choose to replace non intersecting segment of C, as long as the directed segment of C does not close a loop with the new segment of N, whenever a segment is replaced we glue together all routers in the segment of N and the respective segment of B, and try to find more segment to replace with the same ctiteria, namely, no cycles of the new and old segments and no overlapping of segments that are now in a removal process.

Proof outline: The algorithm replaces segments between two consequent intersections nic_i, nic_{i+1} of the two routs (the current and new) according to the ordering of the new, if and only if the current path that connect the two intersecting routers $nic_i = c_y$ $nic_{i+1} = c_z$ is directed from c_y to c_z, thus, no cycles are introduced in the process, and there is a single destination checking point $nic_{i+1} = c_z$ for checking the completion of the new route creation.

6 Conclusion

This paper presents two solutions to the problem of routing consistency during updates. By using multicast on the routes segments, we can safely update the segments, one-by-one, to the new route. Our approaches can simply be applied in the SDN network, even without requiring the routers to identify this multicast connection.

Acknowledgement

We would like to thank Dr. Brian Coan and Dr. Sanjai Narain for their Contributions to this research.