Abstractions for Reconfigurable Hybrid Network Update and A Consistent Update Approach

Weitao Wang, Sushovan Das, and T. S. Eugene Ng
Rice University

ABSTRACT
Reconfigurable Hybrid (electrical/optical) Network (RHN) [1–4, 6, 8, 10, 11, 13–19] for modern datacenter architectures has gained significant momentum during the last decade. The primary advantage of such RHN architectures is the dynamic topological reconfigurability enabled by optical circuit switches (OCS). On one hand, RHN can benefit throughput-intensive applications by providing on-demand high-bandwidth links between the hosts (CPU/GPU/TPU), such as distributed deep neural network training and recommendation systems, etc. On the other hand, RHN can reduce the hop-count between the host pairs, improving the performance for latency-sensitive applications such as real-time customer interactions with in-memory file system. However, previous works mostly focused on finding a suitable topology to efficiently handle a given traffic demand. Performing such topology update together with SDN policy update in a holistic manner while maintaining per-packet consistency and other network invariants is still an open issue. Existing network maintenance and policy update solutions define the notion of per-packet consistency assuming a pure SDN network where the physical network topology is static. This assumption does not hold for RHN because dynamic topology reconfiguration is inherent to RHN. In this paper, first, we define an extended notion of per-packet consistency assuming a pure SDN network where the physical network topology is static. This assumption does not hold for RHN because dynamic topology reconfiguration is inherent to RHN. In this paper, first, we define an extended notion of per-packet consistency and discuss the other critical requirements for RHN updates. Next, we provide an abstraction of RHN update and propose Transtate, a general method to perform such RHN update while satisfying the critical requirements. We believe such innovations remove one of the key obstacles towards reconfigurable-hybrid SDN.

CCS CONCEPTS
• Networks → Network services; Network architectures;

KEYWORDS
Reconfigurable hybrid network, Consistency, Update

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

OptSys ’21, August 23, 2021, Virtual Event, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8650-0/21/08...
https://doi.org/10.1145/3473938.3474506

1 MOTIVATION

1.1 Extended Per-packet Consistency

During the network update, one critical requirement is to satisfy per-packet consistency [12], which requires any packet to only use either the configuration prior to the update or the configuration after the update, but never a mix of two. That definition of consistency assumes the physical topology remains the same during the updates. However, reconfigurability is one of the main advantages of RHN. And for an RHN update, it should include both optical topology updates and the corresponding SDN policy updates. In this case, we extend the per-packet consistency for the RHN and include the network topology change as a part of the update.

RHN Per-packet Consistency (RHNPC). Each packet is processed either using the previous SDN policies and previous network topology before the update, or the new SDN policies and new network topology after the update. Not only a mixture of two SDN policy is not allowed, a mixture of the previous SDN policy and the new topology is also not allowed, and vice versa.

Figure 1: (a) Some RHN solutions ignore the per-packet consistency, which causes transient problems during the update; (b) network suffers from severe bandwidth drop and unreachability issue when the RHN uses existing SDN solutions directly.
1.2 Insufficient Existing Solutions

**RHN architectures’ solutions.** Most of the papers about the RHN architectures introduce an update plan to instruct the reconfiguration, like the flexible rack-level inter-connection in OSA [2], and the longest time-slot first scheduling in Mordia [11]. Firstly, the RHNPC cannot be preserved despite the short optical link interruption time (11.5 µs for 2D-MEMS switches), since the update of the optical link and the SDN policy is not guaranteed to finish at the same time. As Figure 1(a) shows, if the optical update or SDN update finishes earlier than the other one, the SDN switch will send the packet to the wrong destination; and during the optical port downtime, the packet will be sent to an optical link that hasn’t been established yet. Secondly, the extra retransmission delay introduced by wrong routing and packet loss is also not favorable for the nowadays NVMe technologies and in-memory file systems, which requires 10 µs latency at 1M+ IOPS. Thirdly, for switches with ACL policies or firewalls, the packets with the wrong routing path may even break the security rules.

**SDN networks’ solutions.** The existing network maintenance and network policy update works are both feasible to be applied for RHN update if the traffic is drained from the optical links affected by topology reconfiguration. But such naïve solution can be disruptive for applications. Most of the SDN network update works only consider fixed topology and usually conduct consistent updates by packets version changing, which indicate both the previous and new policies exist at the same time [9, 12]. Similarly, they also assume the previous routing path and the new routing path can always co-exist in the same topologies. While in RHN, the new links cannot be established until the previous links have been turned down. Though by draining the traffic from affected optical links, some of the network update strategies [9] can be applied to the RHN updates. However, the RHN may get split apart during the update as shown in Figure 1, which may create reachability issues or experience severe bandwidth drop during the updates.

1.3 RHN Update Principles

Based on the above discussion, we summarized three principles for the RHN updates.

**Guarantee RHNPC during updates.** By ensuring the RHNPC is preserved during the update, we could avoid the problems like the wrong routing path, unnecessary packet drop, and security rule violation. Moreover, for some specific network applications, RHNPC can also help them achieve better performance, like a more accurate measurement for network monitoring and better link utilization for the load balancer.

**Preserve network invariants.** During the updates of the RHN, the network invariants still need to be preserved at all times, including no forwarding loop for every packet, all-to-all connectivity between all end-points, no violation of the access control policies, and isolation between virtual networks. If the RHNPC is provided already, then the network invariants only need to be checked for the previous policies, the new policies, and the policies during the transition states.

**Minimize bandwidth reduction.** To relieve the congestion and reduce the tail latency during the update, the bandwidth reduction should also be minimized. If the bandwidth between two end-points or two groups of end-points drops dramatically during the update, though the update only takes a short time, a lot of packets will be dropped and flows may decrease the congestion window size largely or even invoke TCP timeout.

2 TRANSTATE DESIGN

We propose Transtate, which is an RHN update planner and scheduler to help achieve the above three reconfiguration principles.

2.1 Abstractions

An RHN has two parts, the SDN network part, and the reconfigurable optical network part. To abstract the RHN, we view the whole network as a reconfigurable optical network connecting multiple regions of network referred to as *islands*, where the optical network either behaves as the only connectivity substrate among the islands or complements the SDN network.

The scope of the island varies across different reconfigurable hybrid networks: In general, an island is the biggest network region beyond where the packets may be transmitted through the optical links, if necessary. Namely, the packets generated and destined within an island do not cross the optical boundary. Therefore, an island has the following three properties: 1) An island may consist of an SDN switch or a group of SDN switches. 2) The network components in an island are always connected by the SDN network, and the connectivity never gets impacted during the optical network topology reconfiguration. 3) Different islands may or may not get always connected by the SDN network. The connectivity among the islands may get impacted during the optical topology reconfiguration.

For example, rack-level reconfigurable optical networks [5, 7, 14] connect the ToR switches in a pod, complementing the SDN network. As a result, each ToR is an island, since intra-ToR traffic never uses the optical links while the traffic across the ToRs may use the reconfigurable optical links. For the ones like Rotornet [10] and Sirius [1], the optical network acts as the only connectivity substrate among ToRs. Therefore, each ToR switch should also be considered as an island. Similarly, if the reconfigurable optical network provides direct connectivity between multiple pods or multiple fat-tree units, then the individual pod or fat-tree unit would be considered an island.

**Topology abstraction.** In summary, we abstract any RHN topology to be an undirected graph $G = (V, E)$ where islands
will use this method for update whenever the SDN policies are changed.

As a connected group. For example, Figure 2(c) shows three connected groups as \( \{1\}, \{2, 3, 5, 6\}, \{4\} \). Note that, if there is no common link between \( G \) and \( G' \), each island is considered as a connected group.

**2.2 Updating Strategies**

Similar to many SDN network update solutions, Transtate uses the packet version changing (two-phase update) as atomic update operations, which will pre-install the new rules onto the SDN switches and then switch the packet version to use different rules to achieve consistent update. Since the RHN update also includes the SDN policy updates, we will use this method for update whenever the SDN policies are changed.

If we use the packet version changing method, the packets may have different version numbers during the update. Thus, it requires both the previous topology and the new topology to coexist at the same time. To satisfy that, RHN must use packet version change to drain the traffic from the whole network to a subgraph of the network, where the previous and new topology can coexist. Then those link without traffic, can be reconfigured with RHNPC guarantees since no packet is involved. By draining the traffic from all the affected optical links and then bring the traffic back after optical reconfiguration, existing SDN network solution can perform an update that satisfies RHNPC. However, this one-transtate solution (when the traffic is only placed only on a subgraph of the network and wait for the optical to be updated, we call it a transtate.) may fail to preserve the reachability between islands and has severe bandwidth drop as shown in Figure 1.

**Two-transtate update.** To preserve the reachability and reduce the bandwidth reduction, we propose the two-transtate update strategy by introducing two different transtates instead of one. Intuitively, this strategy moves the traffic to a subgraph (transtate 1) and reconfigure some of the optical links that need to be updated; then move the traffic to another subgraph (transtate 2) and reconfigure the rest of the optical links. As shown in Figure 3, we could achieve all the requirements if we could find a valid two-transtate update plan.

The detailed algorithms are shown in Algorithm 1. (Some notations are introduced in §2.3.)

**Algorithm 1:** Transtate Updating Algorithms

```python
1 def OneTranstate(G, G'):
2     | drain traffic from E_{update} in G;
3     | update E_{update} to E_{update}'
4     | move the traffic back to G';
5 def TwoTranstate(G, G'):
6     | drain traffic from E_{update,1} in G;
7     | update E_{update,1} to E_{update,1}';
8     | move the traffic to E_{fix} \cup E_{update,1}';
9     | update E_{update,2} to E_{update,2}';
10    | move the traffic back to G';
```

**2.3 Find Valid Transtates**

For the two valid subgraphs as the two transtates, \( G_{t,1} \) and \( G_{t,2} \), transtate 1 takes down some links \( E_{update,1} \) in \( G \), and establish some links \( E_{update,1}' \) in \( G' \) using the same ports, while transtate 2 takes down links \( E_{update,2} \) and update its connection to \( E_{update,2}' \). We can easily get \( G_{t,1} = E_{fix} \cup E_{update,2} \) (fixed links and links that will not be updated in transtate 1) and \( G_{t,2} = E_{fix} \cup E_{update,1} \) (fixed links and links that finished update in transtate 1). Then we can summarize the following key properties: 1. in any transtate, the update should turn links in the initial topology to links in final topology (Equa 1&2); 2. the links updated during the first transtate and the second transtate should contain all the links in \( E_{update} \) (Equa 1&2); 3. the node degree should never exceed its original degree in \( G \).
We implemented a simple zero-one integer linear programming algorithm to find the update plans for the two-transtate update, using Gurobi with 367 lines of Python code.

To evaluate the performance of two-transtate update, we vary the number of islands (nodes) and each node's degree, then randomly generate an initial network topology and a final network topology which satisfied the network invariants. Then we apply both approaches to find a valid two-transtate plan with our Gurobi Solver. For the results, we show the number of unreachable island pairs in Table. 1, and the average computation time for finding a valid update plan with different sizes of the network in Table 2. For each network size, we did 20 repeat experiments for each data point.

<table>
<thead>
<tr>
<th>Degree</th>
<th>Number of islands (nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-Transtate</td>
</tr>
<tr>
<td>2</td>
<td>8.6</td>
</tr>
<tr>
<td>4</td>
<td>15.4</td>
</tr>
<tr>
<td>6</td>
<td>9.8</td>
</tr>
<tr>
<td>8</td>
<td>2.7</td>
</tr>
<tr>
<td>10</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Table 1: The number of unreachable island pairs for two different update methods.

<table>
<thead>
<tr>
<th>Degree</th>
<th>Number of islands (nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>1.1</td>
</tr>
<tr>
<td>4</td>
<td>1.9</td>
</tr>
<tr>
<td>6</td>
<td>2.7</td>
</tr>
<tr>
<td>8</td>
<td>3.7</td>
</tr>
<tr>
<td>10</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Table 2: The valid transtates computation time (ms) for both approaches.

As shown in Table 1, a one-transtate update leads to a lot of reachability issue during the update, while two-transtate update solves the connectivity problems for most of the networks with nodal degrees larger than 2. And Table 2 shows the approach B takes slightly longer time than A, but both approaches can be found in a very short time. The computation time scales well with the increasing network size, which only takes less than 1 second to find a valid plan for a network with up to 1000 nodes.

4 CONCLUSION

We extend the definition of per-packet consistency and propose reconfigurable hybrid network per-packet consistency, which adds topology change into the problem scope. On top of RHNPC, we present Transtate, which is a reconfigurable hybrid network update planner and scheduler. The two-transtate update method solves the reachability issue and bandwidth drop issue when applying the current SDN policies on the reconfigurable hybrid networks directly, which make the RHN more friendly to latency-sensitive workloads and throughput-intensive workloads.
5 ACKNOWLEDGEMENT

We thank the anonymous reviewers for their insightful feedback. This research is sponsored by the NSF under CNS-1718980, CNS-1801884, and CNS-1815525.

REFERENCES


A APPENDIX

A.1 Two-transtate Properties

For a hybrid network update, we have the initial topology \( G \) and the final topology \( G' \). For every node in the graph, its degree \( \text{deg}(v) \) and the final topology \( G' \), respectively. The rest of the links remaining fixed during update is \( E_{\text{fix}} \). If some of the islands are connected by the fixed links \( E_{\text{fix}} \), we call this subset of islands that remains connected during the update as a connected group. For example, 2(c) shows the common links (marked red), which leads to three connected groups such as \( \{1\}, \{2, 3, 5, 6\}, \{4\} \). Note that, if there is no common link between \( G \) and \( G' \), each individual island is considered as a connected group.

For the two valid subgraphs as the two transtates, \( G_1 \) and \( G_2 \). And transtate 1 break the links set \( E_{\text{update}} \) and establish links in \( E_{\text{update}} \), while transtate 2 disable \( E_{\text{update}} \) and update its connection to \( E_{\text{update}} \).

We can summarize the following properties:

- In any transtate, the update should turn links in the initial topology to links in final topology (Equa 1&2);
- The links updated during the first transtate and the second transtate contains all the links in \( E_{\text{update}} \) (Equa 1&2);
- The node degree should never exceed its original degree in \( G \) during any transtate (Equa 3);
- Any transtate topology should be a connected graph to satisfy network invariants (Equa 4);
- All the fixed links should not be changed during any transtate (Equa 5&6);
- The links that are not updated in the current transtate should be active (Equa 7&8).
\[ E_{\text{fix}} \cap E_{\text{update}_k} = \emptyset, \forall k \in \{1, 2\} \]  
\[ E_{\text{fix}} \cap E'_{\text{update}_k} = \emptyset, \forall k \in \{1, 2\} \]  
\[ G_{t-1} = E_{\text{fix}} \cup E_{\text{update}_2} \]  
\[ G_{t-2} = E_{\text{fix}} \cup E'_{\text{update}_1} \]